☒

Dark mode

Comparison of RAM and HDD Read/Write Speeds

Abstract

This article provides a detailed comparison of the speed at which data is read and written on RAM and a hard disk drive (HDD). The underlying technology of both RAM and HDD is explained, followed by a comparison of their average read/write speeds considering technological complexities.

Purpose of Comparing RAM and HDD Speed

RAM vs HDD

A few years ago, during a discussion on a programming forum, I mentioned that the average read/write speed of a hard disk drive is approximately 100,000 times slower than RAM. This statement seemed strange to many programmers, and not everyone accepted it. To justify my claim, I had to explain the technical aspects of RAM and HDD.

In this article, I will discuss why the average speed of a HDD is, on average, a hundred thousand times slower than RAM.

First of all, it is important to note that an accurate statement cannot be made regarding the speed comparison between HDD and RAM. Claiming that HDD is exactly that much slower than RAM shows a lack of understanding of the subject in a person.

Secondly, Comparing hard disk drives and RAM cannot be accomplished in a single article :) One can write a multi-volume book about RAM and another multi-volume book about HDD, and then compare them in a multi-volume book together! The depth of the subject is vast, and it extends beyond a simple comparison based on internet speed mentions.

It's not as simple as saying, "Hey, the internet states that HDD speed is X, and RAM speed is Y, so let's just do the math and divide." Nope, this topic is way more complex! It's the kind of thing that could be explored in a doctoral thesis, with someone dedicating their entire PhD research to it and writing a lengthy paper. Yet, even then, the subject would still not be fully covered.

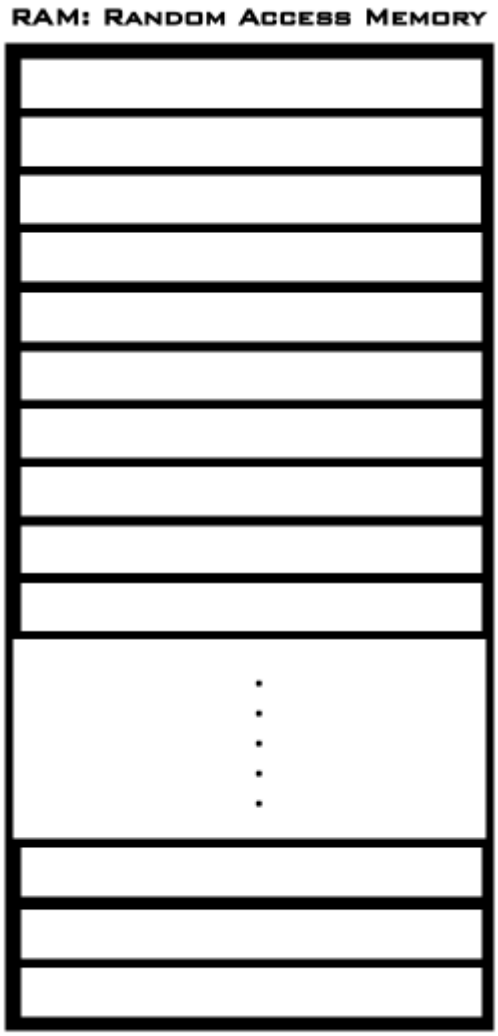
I said these stuff to highlight the complexity of the topic... □□

Most of the information presented here regarding hard disk drives is sourced from Professor Abraham Silberschatz's book, "Database System Concepts, 6th Edition."

Now let's move on to the main story.

What is a RAM?

RAM stands for Random Access Memory, which is a type of memory with random access. The name itself explains all the necessary information about RAM that we need to know. Imagine you have a large piece of memory that has relatively high speed (compared to technologies like disks and even things like EEPROMs) and its abstract representation is something like this:



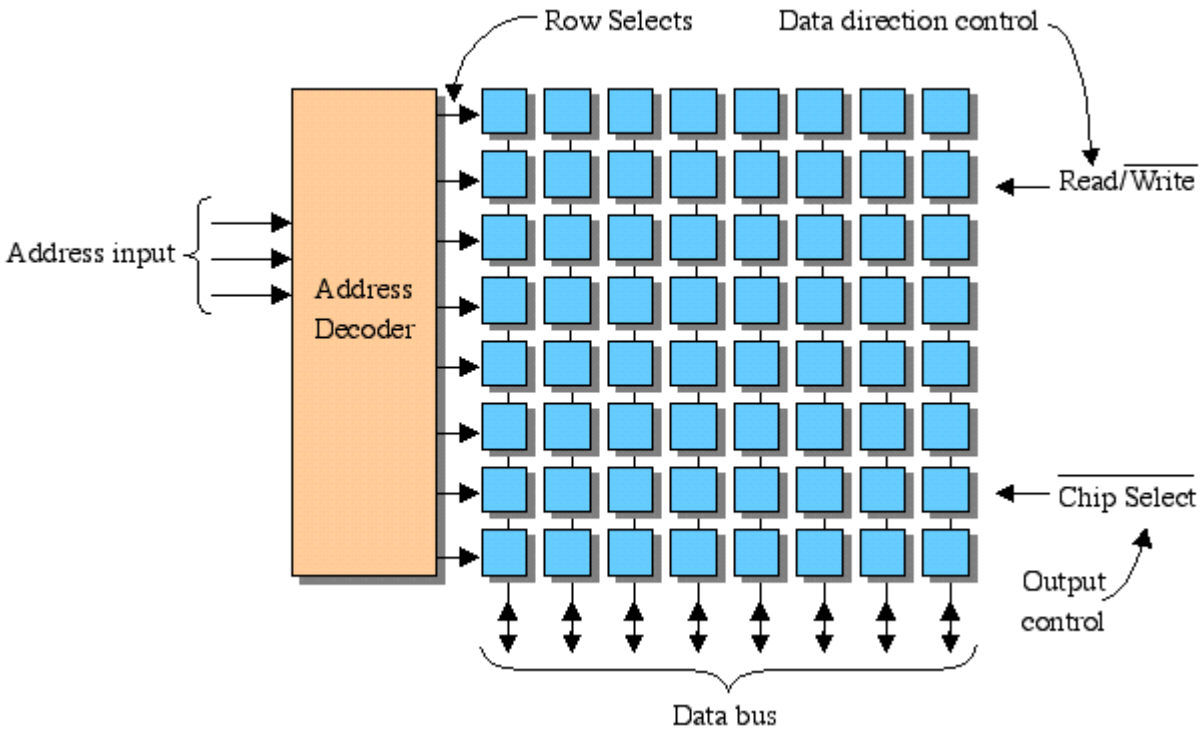
An Abstract RAM Model

Imagine each cell as a byte. You can store information in a continuous sequence of n bytes with a decently fast speed. This level of speed allows the operating system to quickly store its current computation data (Look up for Memory address register).

But why is it so fast?

From a hardware implementation perspective, this memory is structured like a two-dimensional array, similar to a matrix. The "Random Access" structure allows us to access all different cells of memory simultaneously at the same time, but how?

Let's examine the logical block structure of RAM:



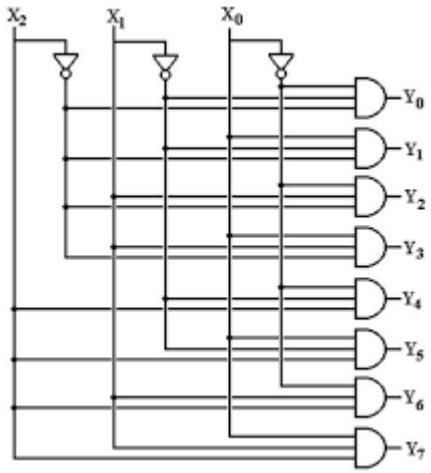
An 8bit RAM - image source: ETSU

Nowadays, Random Access Memories (RAMs) consist of small cells arranged in a two-dimensional grid. Each RAM contains an `Address Decoder` unit that takes in a binary encoded address and decodes it to determine which memory cell should be accessed for reading or writing. Additionally, there is a `Read/Write` unit that determines whether data should be written to or read from the selected cell. When using multiple memory chips (So as real world RAMs), a unit called `Cheap-Select` is employed to facilitate the selection process. The RAM chip cells also have a data input/output component known as the `databus`. It uses low voltage to represent 0 and high voltage to represent 1. To understand the schema of RAM, you can assume that each row of the RAM is 8 bits long, which is equivalent to 1 byte.

Suppose you want to write in the 1000th (1000) memory cell. To do so, you need a 10-bit address-decoder (By using a 10-bit address-decoder, you can access memory locations up to 1024. This is because 2 raised to the power of 10 equals 1024). So, you convert the decimal number 1000 to binary, which is `1111101000`. This binary address is sent to the address decoder. The address decoder uses internal logic gates to interpret the binary address and select the corresponding memory row for read/write operations. This is where it differs from

technology like a disk. In a disk, you would need to iterate and pass through a sequence of cells to reach the desired location. But here, there's an address-decoder that activates a special circuit for direct access to the desired memory location. In this way we not only avoid physical movement (compared to a disk), but we also eliminate the need for moving from one memory location to another (compared to other technologies), so the speed rises up.

The figure below shows a 3-bit address-decoder. With 3 bits, you can have 8 different states or combinations. So, a 3-bit address-decoder can address and select 8 memory locations with the same speed.



A 3-bit Decoder with 8 Outputs - Image Source: Codestall

A 10-bit address decoder is just like this, and it can select up to 1024 different rows. But hey, you can't fit an image of a 10-bit address decoder in this article!

So that's the meaning of the "Random Access" structure, which refers to a hardware design where the access time to each memory cell is the same. We can achieve that by implementing a logical gate called an address decoder.

If access time is very fast, why don't we replace slow things like HDDs with RAMs and use "Random Access" instead? Well, because of how RAM is designed (logical and electronic structure), it loses all its data, and cells are reset to zeros when the power is turned off. So

it can't be used as a stable memory for long-term storage. It's only suitable for storing the information needed for ongoing computations.

These days, memory devices like SSDs, which have no physical movement, are being used in some places instead of things like HDDs. They are much faster because they don't have physical activities, but they also have their own issues. Maybe in the future, we'll discuss those technologies in a separate article.

Let's get back to RAM, When it comes to designing hardware technologies, RAM can be categorized into two types: Static RAM (SRAM) and Dynamic RAM (DRAM).

- Static RAMs (SRAM):

Static random-access memory (SRAM) stores data in transistor-based circuits similar to D-flip flops. The data remains there as long as the RAM has power and will not be lost. SRAMs are smaller and faster and have simpler decoders compared to dynamic random-access memory (DRAM). They are commonly used in high-speed processes such as caching and very fast applications. In some cases where important data needs to be stored in RAM, SRAM is used, and a backup battery is added to the circuit. This way, even if the circuit loses power for some reason, the system's battery will supply power and prevent data loss.

- Dynamic RAMs (DRAM):

Dynamic random access memory (DRAM) stores information on capacitors. These capacitors have a small structure, which increases the overall memory capacity. However, they have a significant problem: Each capacitor naturally loses its internal current and de-charges after a certain period of time due to its natural structure, resulting in the loss of stored information. This process is called "leakage current". To prevent data loss, the contents of all capacitors need to be rapidly read and written back, so that the capacitors remain charged and the information is not lost. This process is referred to as "refreshing circuit". Therefore, in DRAMs, there must be a circuit that continuously reads and writes the information on all capacitors quickly. Furthermore, dynamic RAMs are often cheaper than static RAMs. And of course, the RAMs currently used on your computer are likely to be dynamic.

I suggest you read the interview with Steve Wozniak from here. Despite it being a personal and ordinary interview, it contains a lot of professional electronic explanations! In one part of the interview, Wozniak talks about RAM technologies, which is very interesting:

«So I searched around. My thinking was always, in making something possible, you've got to get it down to a reasonable cost, but I needed 4K bytes of RAM minimum. The first dynamic RAMs got introduced that year, 1975, the first 4K dynamic RAMs. That was the first time ever that RAMs were lower in price than magnetic core memories, which every computer up to that day had used. So all of a sudden, the world was going to change to RAMs. Silicon was going to be our memory. Everybody else in the world, the Altair, the Sphere computers, the Polymorphic computers, the Insight computers, every one was designed by basically insufficient engineers, not top quality engineers. They were designed by technicians who knew how to look at the datasheets for some RAM, look at the datasheets for a microprocessor and see if the microprocessor had some lines called "address," and the RAMs had lines called "address" and they would hook a wire from one to the other. It's a very simple job. If your RAMs are static RAMs. The dynamic RAMs were going to be 1/4 the price. The dynamic RAMs meant that instead of 32 chips to have enough memory for a computer to have a language, you only needed 8 chips of RAMs. But dynamic RAM needs all this circuitry to get into every single address in the RAM every 2000th of a second, read what was there and write it back, or it forgets it. Dynamic RAM (this is what we have in our computers today) will forget every single bit in a 2000th of a second unless something reads it and writes it back the way it was to hold its state. It's like little electrons stored on a plate and they'll leak off in a 2000th of a second.»

The interview link:
http://www.foundersatwork.com/steve-wozniak.html

And I have almost covered all the technical issues I wanted to discuss regarding RAM. The operating system is responsible for managing RAM. The computer's CPU works with Virtual Addressing, while RAM works with Logical Addressing. In future, I will write a separate article on how the operating system allocates RAM to processes, and another article on how we can access the entire system RAM with it's current cell values.

What is a Hard Disk?

Hard disks are circular plates that store information based on their structure. The first type of disks can be seen in gramophones, where sound vibrations create a spiral groove on a rotating disk. A softer needle moves within the groove to reproduce the sound using vibrations.

Later, magnetic hard disks and optical disks like CDs and DVDs were developed. Magnetic hard disks use magnetization to write information in binary form on round disks. Optical disks use lasers to burn information.

In general, a disk refers to a rotating, circular plate where information is written in different parts of it.

A hard disk consists of magnetic round disks on which information is written by magnetizing each tiny particle in binary form. It typically has multiple layers called platters, with the number of platters usually ranging from 1 to 5.

A cross-sectional view of a hard disk would look like this:

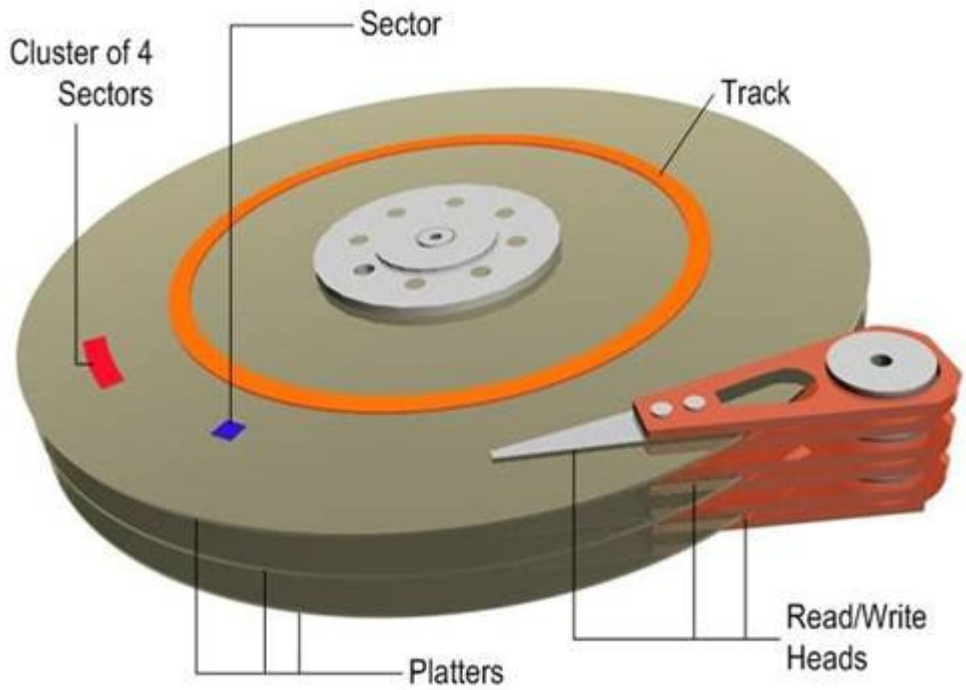


Image source: [compuclever]

An abstract representation of the cross-sectional view:

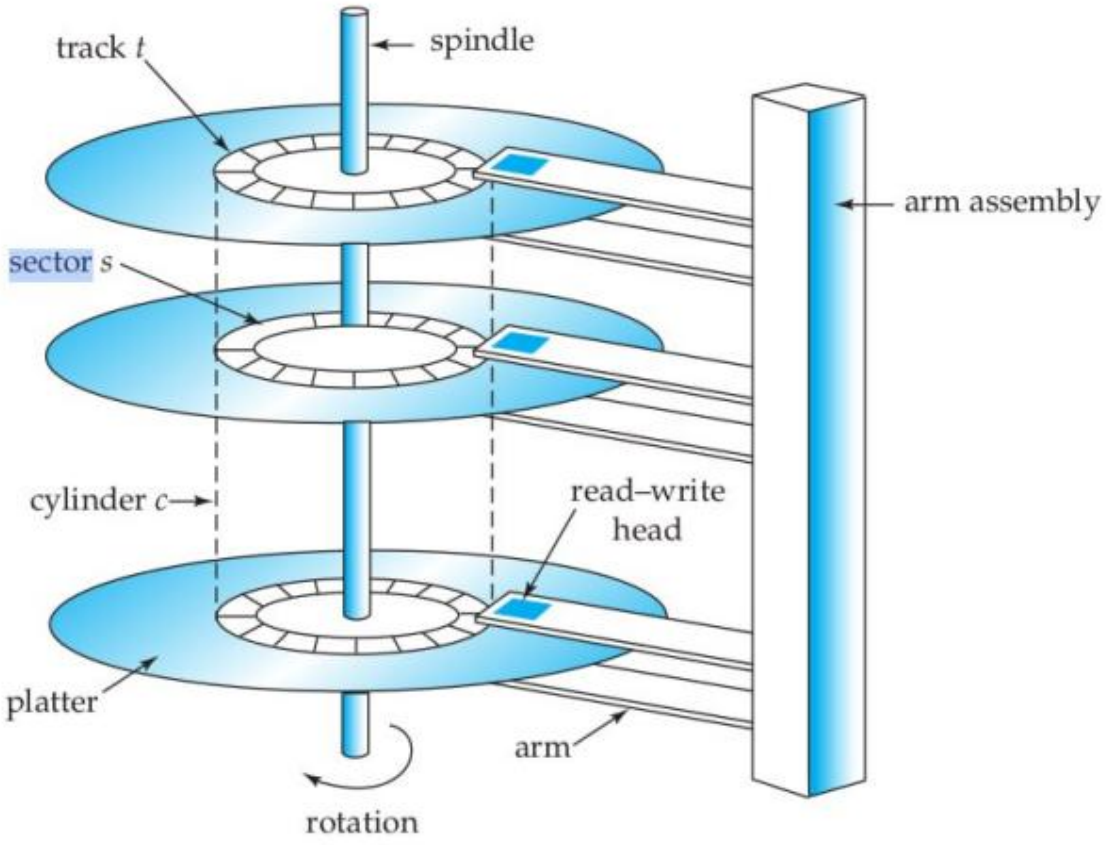


Image source: The Book: Database System Concepts, 6th Edition

The platters of a hard-disk are magnetic surfaces that can store 0s and 1s by being magnetized. These platters are logically divided into different sets of concentric circles, which are composed of circular tracks that form the disk. Each of these concentric circles is referred to as a "track".

In the image below, you can observe several abstract tracks:

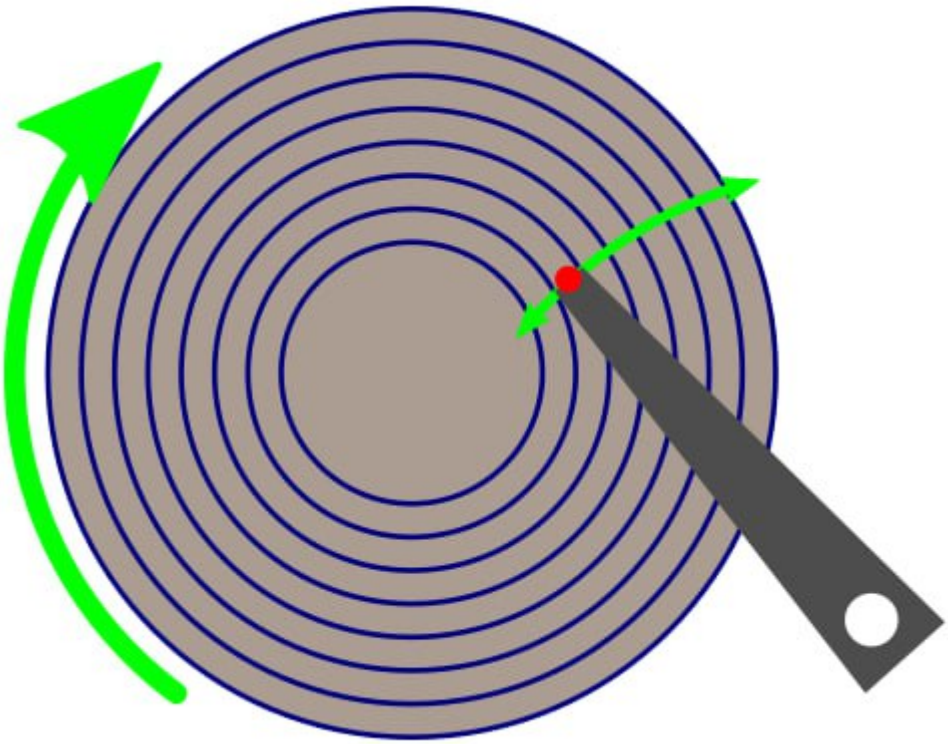


Image source: [schoolcoders]

The track itself is divided into smaller parts called "sectors". You can see the sector portion in the image below:

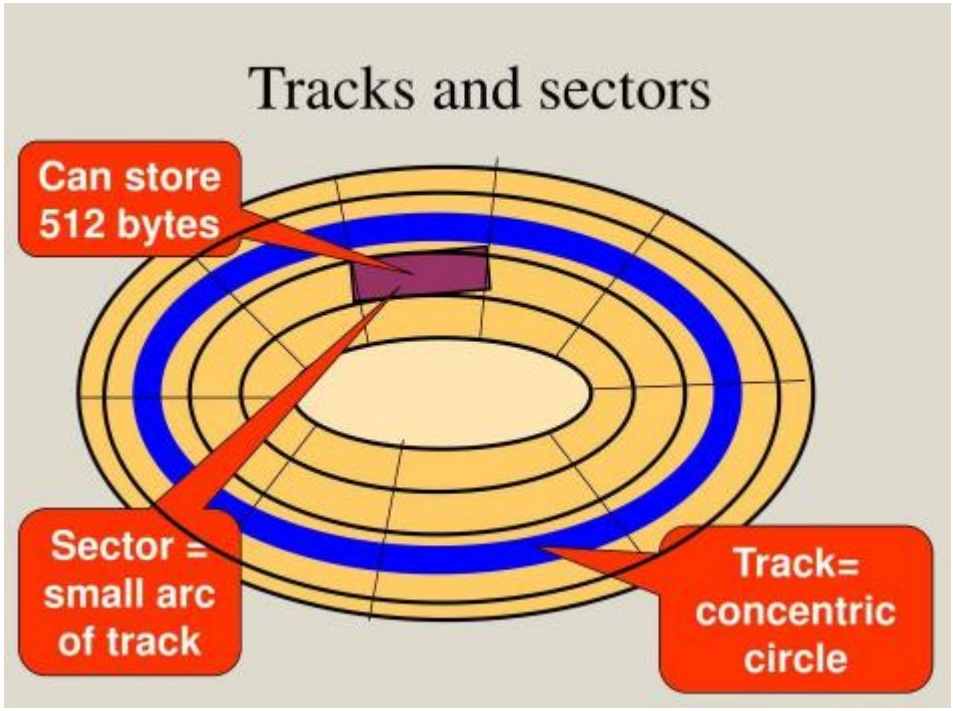


Image source: slideplayer

A sector is also the smallest unit on a disk that is written/read. Usually, a sector has a size of 512 bytes.

There is also a component in a hard-disk called the "arm", which has a "head" that moves back and forth. The arm itself rotates and reads/writes these sectors.

Naturally, outer tracks (those with larger radius) have more sectors than inner tracks (smaller radii)...

Any type of information that needs to be written on a disk must be written in the form of 0s and 1s, (say true or flase, magnetized or not not-magnetized) encoded in binary format, and given to the arm to be written by the head. The same process applies for reading.

Depending on the different software formats used to format hard disks, there are various software technologies related to data storage, such as inode and MFT, which we will discuss in other articles later on.

Comparison of Hard Disk and RAM Speed

Let's ignore all the complexities of implementation, operational details, and stuff related to the operating system and RAM, and focus on the hard-disk for now:

When considering the speed of a hard-disk, we need to take into account the average speed of I/O operations, which means the input/output activities of the hard disk:

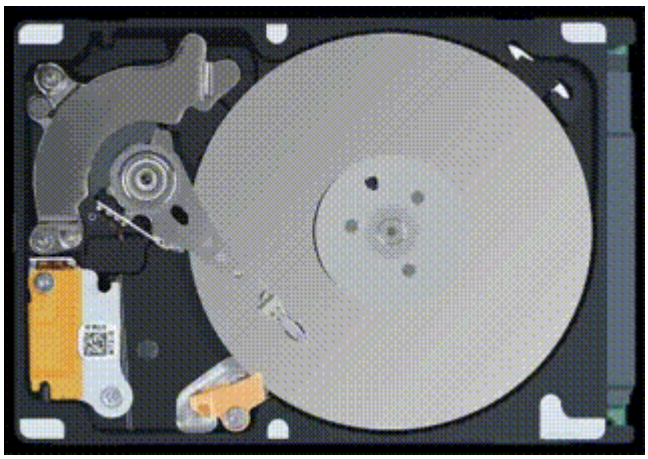
- I/O = Input/Output

Calculating this speed is a highly complex task, and there are thousands of factors involved. Each factor requires an extensive explanation, and ultimately, if we manage to accurately model all of these factors (which is very difficult), the performance of the hard disk drive (HDD) can vary significantly across various scenarios. It can be as much as 10,000 times slower or even up to 500,000 times slower than RAM. To provide a statistical representation, we often present an average, such as 100,000 times slower.

Generally, hard disk drives (HDDs) are slower compared to many other technologies because they involve physical and mechanical movements to store data. To give a real-world example, computer experts often observe that using a mouse for interacting with a computer significantly slows down the speed of interaction. If you can accomplish a task using a keyboard, it can be much, much faster than using a mouse. This is a well-known principle. The reason behind this is that the mouse requires a lot of physical movements. For instance, you need to scroll multiple times to perform a task. The keyboard, on the other hand, minimizes physical movements, which makes it faster. The hard disk drive operates in a similar way to a mouse, we will discuss it further.

A hard disk drive consists of one or multiple rotating disks (similar to CDs) and a head (like the needle on a gramaphone) that can move up and down on a circuit placed on the disk's radius. This way, it can process the entire disk.

Take a look at the animation below, showing the phisical movement of an HDD head:



I don't remember the source of the GIF - if you know it, let me know!

We measure the speed of a hard disk by "Transfer time", which is a combination of several factors, including:

i. Read/Write speed of the head: This speed itself depends on various factors, and it's quite complex to explain in detail here.

ii. Arm movement speed: Refers to the speed at which the arm moves up and down.

iii. Disk rotation speed: Different hard disks have different rotation speeds.

iv. Track capacity: Represents the size of each track on the disk.

v. Sector capacity: The capacity of each sector on the disk.

vi. Amount of data requested: The size of the data that needs to be accessed.

vii. Rotational latency: Refers to the delay caused by disk rotation speed. A brief explanation will be provided here, but for more details, please refer to the introduced book.

viii. Seek time: Represents the delay in positioning the head for accessing specific data.

ix. Hard-disk interleaving: I will provide an explanation for this interesting concept and discuss its difference compared to RAM interleaving.

x. Read ahead technique: I will provide an explanation for this as well.

xi. Buffer sizes: The capacity of the buffer used in the buffering technique.

xii. Disk scheduling algorithms: This topic could fill an entire 12-volume book! It is challenging to explain it briefly. It covers algorithms such as the "Elevator algorithm" or SCAN and more stuff. Without these scheduling algorithms, the hard disk's performance would be significantly slower.

xiii. Storage of the Files (Data Placement): This requires a very detailed explanation. You might have heard of disk fragmentation, which slows down disk speed. I will explain it further.

xiv. Storing logs on the hard disk - This aspect will not be discussed in this article.

xv. Caching of requests by the operating system, which goes beyond the scope of our

discussion.

xvi. Device management, or as Windows folks call it, drive management, is a highly specialized topic that I won't discuss.

xvii. ... There are so many factors affecting disk speed that if I were to mention all of them, I might never be able to finish this article. To be honest, I'm not familiar with all the details as it is a highly professional field. Experts in file systems can provide better explanations..

Now, transfer time, as mentioned earlier, is the sum of all these delays and timings mentioned above.

Now let's move on to the explanation of the factors I mentioned. Let's start with the explanation of the first two ones, which are really cool and important:

RAM is independent of physical movements, but hard disks consume time/energy in physical movement.

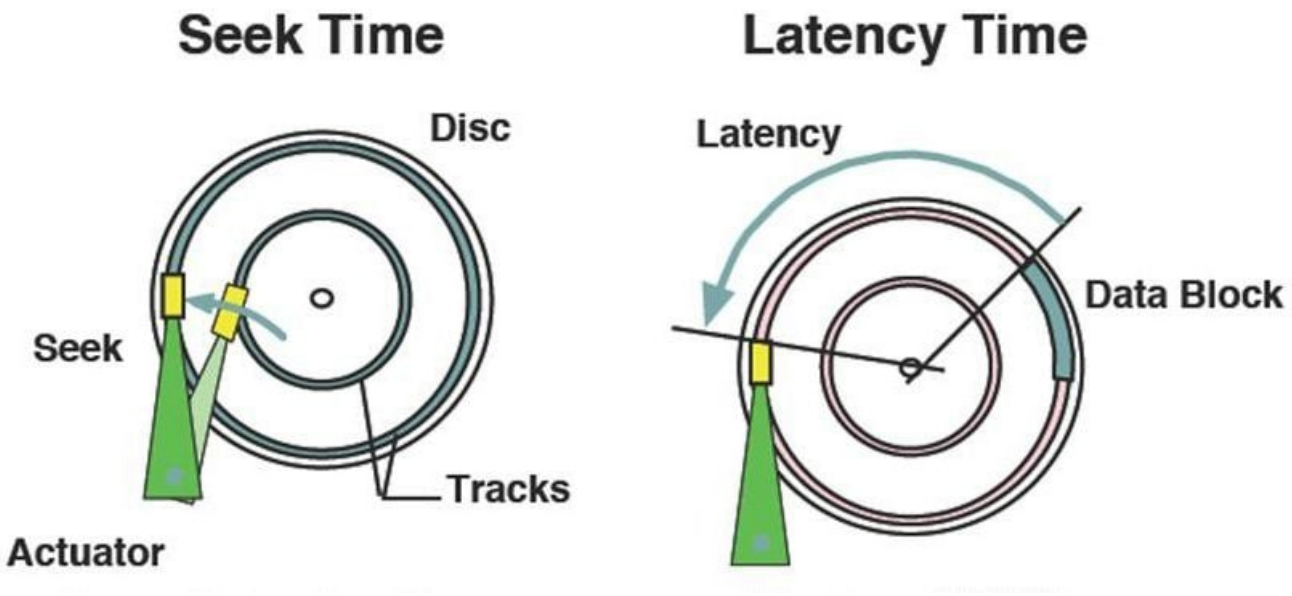
Two important physical time consumptions in hard disks:

1- Rotational latency time
Hard disks have a time consumption when they rotate disks.
Suppose a piece of data that you want to read or write is not in that specific position/degree of the disk where the arm is currenty positioned, and the disk needs to rotate like 260 degrees to reach that specific position of the data! The time spent on rotating is called rotational latency time.

2- Seek time
Disks have another time consumption, which is when the arm of HDD needs to move up and down and precisely position on the corresponding radius to find the sector. It's called seek time.

Just these two physical movements are enough to make the hard disk very slow and prevent it from being used for storing current computing data like RAM. But the good thing is that the data is not lost when power is gone:3

See the image below:



I don't remember the source of image too - let me know if you do!

The right one is rotational latency time,
the time when the disk has to wait and do nothing, while the disk rotates and reaches the desired sector.

On the left is seek time,
the time when the arm has to move and go to the radius where the desired sector is.

Block Interleaving in Hard Disk:

We have several types of interleaving which are data handling architectures, including:

- Bit-interleaved architecture: Means that data is read/written bit by bit in some memory/storage.
- Byte-interleaved architecture: Means that data is read/written byte by byte in some memory/storage.
- Block-interleaved architecture: Means that data is read/written in chunks or in other word blocks, in some memory/storage - the block size is determined by the operating systems.

So, what's the difference?

Back in the early days of computers, data used to be read/written bit by bit, which is quite inefficient since most data we work with requires at least one byte. That's why byte-interleaved architecture was introduced and used...

Today, It is safe to say that you can't read/write just a single bit of information from/to a disk or any memory like RAM; you have to read/write at least one byte. For example, a Boolean variable (TRUE/FALSE) only requires one bit to represent TRUE or FALSE, but it is allocated with 8 bits because the memory is byte-interleaved :)

Now, in hard disks, block-interleaving has been introduced. When you request a specific byte from the hard disk, it would be inefficient to make the disk rotate and move the arm and head up and down just for one byte! And what if you ask for the next byte afterward? Should the HDD make the disk rotate again to find the next byte?
The HDD is smarter than that! It retrieves a block of data and give it to you and tells you to fetch as much data as you need from that block. If you need more, just ask!
For example, it provides 1024 bytes to the operating system, and if the operating system only needs the first byte, it retrieves first byte and drops the rest...
This concept is not arbitrary; it has been developed for years and tested by thousands of people, who have found out that block-interleaving brings better performance for HDDs compared to byte-interleaving.

Now, let's assume that RAM is byte-interleaved, and the hard disk is block-interleaved.

This difference significantly slows down the HDD compared to RAM.

Read Ahead Technique for Hard Disk:

This is a concept similar to what I explained earlier but quite different!

The read-ahead technique in hard disks is a strategy used to optimize data retrieval and loading times. It involves reading data ahead of time into a buffer or cache before it is actually requested by the system or application.

When the system or application requests the next set of data, it is already available in the cache, eliminating the need for the hard disk to seek and retrieve it from the disk platters. This significantly reduces the access time and improves overall performance, especially in cases where consecutive data blocks are accessed sequentially.

For instance, a hard disk has understanded through years of system's behavior that consecutive sectors may be needed by the user (operating system or application). So, even though the operating system doesn't explicitly request a specific block count, the hard disk provides it in the cache. This means that if those fetched sectors (cached) are needed and used, the OS or application doesn't have to make another system-call, and wait for the request to be directed to the hardware driver, and then wait for disk's motor to be turned on, the storage table to be read again, the sector number to be determined, the head to be moved up and down, and the data to be read magnetically and returned and...

Buffering Capacity in Computer Science:

Buffering, in computer science, refers to a temporary storage area. The buffering capacity

represents the amount of temporary storage allocated for buffering data in a computer system.
When it comes to HDD buffering, a portion of the computer's RAM is utilized to store data that is fetched from the hard disk. This data is held in the buffer temporarily, allowing it to be processed or accessed more efficiently before being transferred to other parts of the system.
The buffering capacity differs among different operating systems, and it plays a crucial role in determining the overall speed and efficiency of data processing. Operating system experts should provide better detailed explanation and discussion about buffering and its impact on system performance :3

Hard-Disk Scheduling Algorithms:

There is a common misconception that only one I/O operation can occur on a hard disk at a time. However, in reality, numerous read and write requests are submitted simultaneously when interacting with a hard disk. Thousands of requests are being sent and received. To efficiently handle these requests, sophisticated scheduling algorithms are needed.

Let's consider a scenario where hard-disks have a single arm, head, and disk (we won't focus on multiplatters). When there is only one head and one disk, it becomes necessary to implement optimized scheduling algorithms to efficiently satisfy the given requests. These algorithms can be bloody sophisticated!

Disk scheduling is a highly complex topic. Apart from discussions about the operating system and its software, powerful hardware algorithms are also needed to optimize disk performance.

For example, there is an algorithm called Elevator or SCAN. This algorithm prioritizes certain requests based on the current position of the disk head and the direction of rotation. It aims to minimize head movement and prioritize requests that follow the current path. In the Elevator or SCAN algorithm, when the disk is rotating and the head is on specific sectors, it takes into consideration upcoming requests (let's say three I/O requests after the current one) and performs read/write operations on these sectors. This allows the algorithm to efficiently utilize the current position and avoid ignoring relevant requests. Additionally, while the head is moving in a particular direction, the algorithm does not change the

direction of the head and performs read operations on the next request. This may require significant effort and time for HDD. It helps to seamlessly continue processing another request (say the fifth one) on the same track without unnecessary head movements, and then return for another request (say the first one).

It's like an elevator that goes up and is on floor 16, but if someone presses the button on the first floor, it won't go down and pick them up and then go to the 20th floor. Instead, it will pick up the person on the 20th floor and then go to the first floor. In between, if someone else presses the button on the second floor, the elevator will pick them up before the person on the first floor. This algorithm itself can be explained in detail, but let's move on... :))

There are thousands of other algorithms! This is a very extensive optimization topic. ^_^

If these algorithms were not implemented, we would have to wait for 70 years to perform an I/O operation on a disk.

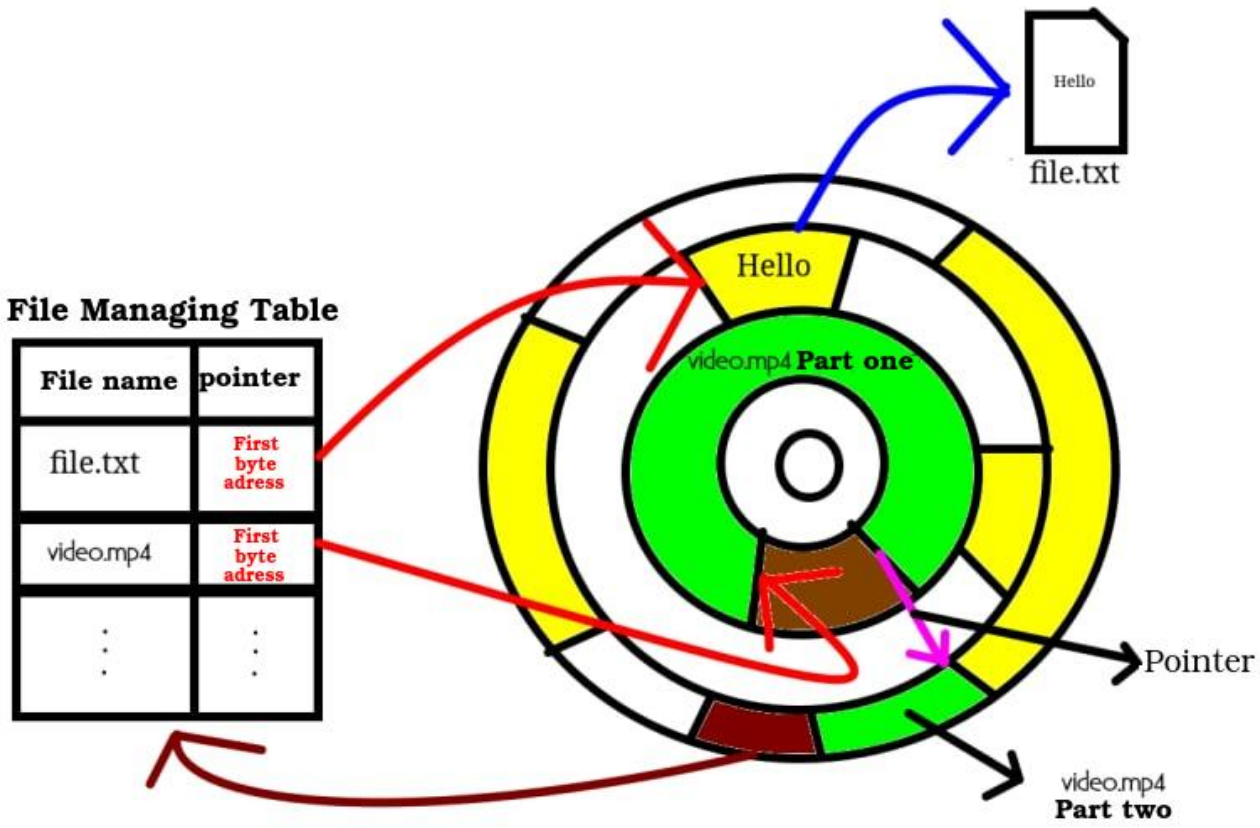
Storage of the Files (Data Placement):

This is perhaps the most important discussion...

You can never use a disk as a processing memory (like RAM) because of its sophisticated data placement...

I'm also writing an article about recovery where I explain disk fragmentation.

Fragmented Disk:



File fragmentation occurs when the available disk space is not large enough to fit the entire file in a contiguous manner. As a result, the file is split into fragments (gets scattered) and stored in non-adjacent sectors on the disk. This can lead to inefficient disk usage and slower file access times.

When a fragmented file is accessed, the disk's read/write head needs to move between different locations on the disk to retrieve all the fragments. This physical movement of the read/write head increases the access time, resulting in slower file operations. Fragmentation can significantly impact system performance, especially for larger files or when there are numerous fragmented files on the disk.

A fragmented disk, is a disk full of fragmented files. Over the time most of disks get fragmented due to centain amounts of altering/rewriting/deleting files.

In truth, the speeds that you refer as transfer-time, must always be based on a fragmented disk, which itself slows down the process thousands of times.
In RAM, we don't have fragmentation, or rather, we do have it, but it doesn't affect speed - A

RAM is designed to be fragmented. RAM, as I explained, allows simultaneous access to all its cells ("Random Access").

Reading/Writing data and working with tables on a hard disk (things like inode or MFT) are also highly resource-intensive and time-consuming.

So, in short, the transfer time of the hard disk, which is affected by everything I mentioned along with thousands of other factors, is very slow.
The physical movement of the disk is so bloody slow that it can be watched by the human eye!

In RAM, the architecture is implemented with logical gates, and its speed depends on the flow of electrons (free electrons).

I found a video that demonstrates a simple representation of how a "Copy and Paste" process works on a hard disk in slow motion.

The physical movements of the hard disk, which can be observed when the HDD box is opened and carefully observed :

https://www.youtube.com/watch?v=3owqvmMf6No

The movement that is visible to the naked eye and can be captured by a camera is definitely much slower than the speed of electron movement :)
In general, being 100,000 times slower is completely logical, but let's use our programming knowledge and test for ourselves to see how the theory in practice!

Speed Comparison between RAM and HDD

Our goal is to run some scripts and programs that can test the claim we made about the hard disk being 100,000 times slower than RAM.

We will divide our test and experiment into two different phases, and I'll explain why: Before we proceed, let's understand the unique characteristics of RAM and the hard disk. As I mentioned earlier, RAMs are higly fast and are ideal for real-time calculations. They are suitable for computations with a high allocation rate and also for small, frequent datas. Hard-disks on the other hand, are suitable for storing large amounts of data, usually files, and retaining it for a long period. A RAM cannot store very large data because its capacity is limited and usually lower than things like disks and additionally, all its contents are lost when the power supply is cut off. The speed limitations of a hard disk prevent it from efficiently storing small, intricate data at a rapid rate and achieving optimal allocation for such data. So, as they both have different mechanisms they also have different functions, and Therefore we perform our experiment in two phases.

In the first phase, we will compare the speed of working with large datas, which are typically handled by the hard disk, in both RAM and the hard disk itself.

In the second phase, we perform a comparison experiment with small and computational data that require fast storage, usually RAM, in both RAM and HDD.

Regarding the comparison tools, my favorite programming language is `PHP`. But when it comes to measuring time complexity/resource usage of algorithms and we want to perform experiments comparing the time/resource consumption of different methods, my opinion is that we should always use languages closer to a lower-level machine codes. Compiler-based languages like `C` and `C++` always give us much more access, proximity, and efficiency to resources such as RAM and disk. Also, because of their compiler structure, they translate code into machine language, which is solely responsible for executing the algorithm and is less involved in overhead computations like interpreting. By using these languages, the results of our experiments are less affected by the time/resource waste caused by code interpretation or the execution of interpreter, translator, and compiler processes. That's why we use the `C++` language in this experiment.

Phase 1: Comparing the Speed of RAM and Hard Disk in Handling Large Data

Consider the following code:

```
#include <iostream>
#include <sys/resource.h>
#include <chrono>

using namespace std;

int main() {

    // Start execution time
    clock_t start_1 = clock();
    auto start_2 = chrono::high_resolution_clock::now();

    // Start memory usage
    struct rusage usage;
    getrusage(RUSAGE_SELF, &usage);
    long memory_usage_start = usage.ru_maxrss; // in kilobytes

    // Code ------------------------------------------------------------
    char* arr = new char[1073741824];
    for(int i = 0; i < 1073741824 ; i++)
        arr[i] = 'A';

    // -----------------------------------------------------------------

    // Stop measuring memory usage
    getrusage(RUSAGE_SELF, &usage);
    long memory_usage_end = usage.ru_maxrss; // in kilobytes

    // Stop measuring execution time
    clock_t end_1 = clock();
    auto end_2 = chrono::high_resolution_clock::now();
    double execution_time_1 = double(end_1 - start_1) / CLOCKS_PER_SEC;
    chrono::duration<double, milli> execution_time_2 = end_2 - start_2;

    // Printing result
    cout << "Execution Time (Based on ctime): " << execution_time_1 * 1000.0 << "
ms" << endl;
    cout << "Execution Time (Based on chrono): " << execution_time_2.count() << "
ms" << endl;
    cout << "Memory Usage: " << memory_usage_end - memory_usage_start << " KB" <<
```

```
endl;

    return 0;
}
```

Source code

This code measures the execution time and the amount of RAM consumed by a specific piece of code. The specific piece of code is the segment below, which allocates exactly 1 gigabyte (1073741824 bytes) of RAM to an array of characters. In simple terms, this code, exactly allocates 1 gigabyte of data in the RAM:

```
char* arr = new char[1073741824];
for(int i = 0; i < 1073741824 ; i++)
    arr[i] = 'A';
```

When you run this code, you can get a good estimate of how long it takes to store 1 gigabyte of data in memory in "Random Access" method and how much memory (RAM) is used (which should use exactly 1 gigabyte). This code measures the execution time of that specific segment, once using the clocks in the ctime library and again using the chrono library. This way, we can get a relatively accurate output time. Remember that including iostream (at least in my compiler `g++`) also links the ctime library to the code. If that didn't happen in your compiler, make sure to manually include the library:

`#include <ctime>`

Let's run the code to see the output. I'm using a Linux machine, so I compile the code with `g++ RAM_test.cpp` and run it using `./a.out`.

Note: Always remember to repeat the experiment (running the code) at least 3 times in different conditions and consider the overall average as the answer. At any given moment, there are thousands of other processes running on a computer that interact with resources. The behavior of these processes may affect your experiment's results at different times.

Note 2: There is a famous Unix command called `time` that allows you to have some extra information about the command you are running. I'm using this package to monitor the times and CPU consumption during program execution. I will explain the meaning of this information exactly after the execution.

Running the code:

```
┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ g++ RAM_write_test.cpp

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
Execution Time (Based on ctime): 2501.88 ms
Execution Time (Based on chrono): 2501.97 ms
Memory Usage: 1046800 KB

real    2.51s
user    2.36s
sys     0.14s
cpu     99%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
Execution Time (Based on ctime): 2440.13 ms
Execution Time (Based on chrono): 2440.23 ms
Memory Usage: 1046864 KB

real    2.45s
user    2.32s
sys     0.13s
cpu     99%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
Execution Time (Based on ctime): 2446.16 ms
Execution Time (Based on chrono): 2446.23 ms
Memory Usage: 1046772 KB

real    2.45s
user    2.32s
sys     0.13s
cpu     99%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
Execution Time (Based on ctime): 2444.35 ms
Execution Time (Based on chrono): 2444.5 ms
Memory Usage: 1046848 KB
```

```
real    2.45s
user    2.29s
sys     0.16s
cpu     99%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
Execution Time (Based on ctime): 2515.03 ms
Execution Time (Based on chrono): 2515.09 ms
Memory Usage: 1046832 KB

real    2.52s
user    2.39s
sys     0.13s
cpu     99%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
Execution Time (Based on ctime): 2438.22 ms
Execution Time (Based on chrono): 2438.28 ms
Memory Usage: 1046844 KB

real    2.45s
user    2.32s
sys     0.12s
cpu     99%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
Execution Time (Based on ctime): 2493.81 ms
Execution Time (Based on chrono): 2493.94 ms
Memory Usage: 1046784 KB

real    2.50s
user    2.35s
sys     0.15s
cpu     99%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
Execution Time (Based on ctime): 2501.99 ms
Execution Time (Based on chrono): 2502.09 ms
Memory Usage: 1046852 KB

real    2.51s
user    2.37s
sys     0.14s
cpu     99%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
Execution Time (Based on ctime): 2459.47 ms
```

```
Execution Time (Based on chrono): 2460.05 ms
Memory Usage: 1046872 KB

real    2.47s
user    2.32s
sys     0.15s
cpu     99%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
Execution Time (Based on ctime): 2429.63 ms
Execution Time (Based on chrono): 2429.75 ms
Memory Usage: 1046772 KB

real    2.43s
user    2.32s
sys     0.12s
cpu     99%
```

I ran this code on my system in 10 different time intervals when the system was engaged in different computations each time, and the output was as mentioned above. As you can see, the execution time varies within a certain range, but there are slight changes each time. These are the results of the parallel processing capabilities of our operating system. I will write an article later about parallel computations in the operating system. Another interesting matter is that the amount of memory consumed is different in each execution, which seems a bit strange. However, it can be explained in detail, and then we will see that it is completely decent. Explaining the matter in a detailed manner is beyond the scope of this article, but in short, we can say that the performance of the code depends on the behavior of the libraries and functions we have included from the system. These functions, when called, each time bring different data to the system cache, causing the amount of data stored in RAM to vary. However, overall, it is about a range of 1 gigabyte consumed by our mentioned segment.

Now let's move on to the detailed explanations of the `time` command, which includes reports such as real, user, sys, and CPU, and see what they mean.

Real Time

It is the total time it took for the process to start and complete, as if we measured it with a stopwatch (or folks say with a wall clock). In technical terms, this time includes all the times

spent, and represents the real time of a proccess from start to the end, including: the computation time, the times the process was waiting (blocked), and the times it was waiting for I/O.

User Time

It is the time that the CPU spent on this specific program calculations. In technical terms, it is the time when the process is sitting on the CPU and being executed (referred to as user mode).

Sys Time

It is the time the CPU spends on tasks performed by the operating system kernel realated the process, such as memory allocation and I/O operations (referred to as kernel mode).

cpu

And finally, this one is about how much of the CPU is allocated to this process. It might go beyond 100% (e.g. 650%) if you are running multi-threading, where multiple CPU cores are involved.

I mentioned the two execution modes of processes, user mode and kernel mode. These are modes of executing programs in Unix-like systems or any other memory-protected systems. There are differences between user mode and kernel mode, which I will try to explain briefly, and then I will provide a source for more detailed and specialized explanations:

Kernel Mode

If code is executed in this mode, that piece of code has complete and unrestricted access to the hardware. It can execute any system-call, such as any instruction for the CPU or

interaction with any part of the memory. This is the highest level of access and happens at the lowest layer. It is risky if there is any misbehavior or issues in code running at this layer, as it can jeopardize the whole system. Therefore, the trusted system functions and APIs are executed in this mode.

User Mode

In contrast to the previous mode, in user mode, the program executor does not have direct access to the hardware and cannot interact directly with entities such as RAM and disks. Instead, it relies on an API provided by the operating system to interact and perform operations on its behalf. I will write an article later explaining the concept of an API. Therefore, this mode is more secure, and the code is isolated from the hardware, making it recoverable if any issues arise. Most of the codes executed in the system run in this mode.

References for this section:

- *https://blog.codinghorror.com/understanding-user-and-kernel-mode*
- *https://askubuntu.com/questions/920920/how-to-interpret-time-real-user-and-sys*
- *https://stackoverflow.com/questions/1311402/what-is-the-difference-between-user-and-kernel-modes-in-operating-systems*
- *https://unix.stackexchange.com/questions/53302/why-would-the-real-time-be-much-higher-than-the-user-and-sys-times-combine*
- *https://stackoverflow.com/questions/556405/what-do-real-user-and-sys-mean-in-the-output-of-time1*

Now we can have an estimate of the average time:

```
(2501.88+2440.13+2446.16+2444.35+2515.03+2438.22+2493.81+2501.99+2459.47+2429.63) /
10 = 24670.67 / 10 = 2467.067
(2501.97+2440.23+2446.23+2444.5+2515.09+2438.28+2493.94+2502.09+2460.05+2429.75) /
10 = 24672.13 / 10 = 2467.213
(1046800+1046864+1046772+1046848+1046832+1046844+1046784+1046852+1046872+1046772) /
10 = 10468240 / 10 = 1046824
(2.51s+2.45s+2.45s+2.45s+2.52s+2.45s+2.50s+2.51s+2.47s+2.43s) / 10 = 24.74 / 10 =
2.474
(2.36s+2.32s+2.32s+2.29s+2.39s+2.32s+2.35s+2.37s+2.32s+2.32s) / 10 = 23.36 / 10 =
2.336
(0.14s+0.13s+0.13s+0.16s+0.13s+0.12s+0.15s+0.14s+0.15s+0.12s) / 10 = 1.37 / 10 =
0.137
(99%+99%+99%+99%+99%+99%+99%+99%+99%+99%) / 10 = 990 / 10 = 99
```

```
Therefore:

* Average Execution Time(Based on ctime): 2467.067 ms
* Average Execution Time(Based on chrono): 2467.213 ms
* Average Memory Usage: 1046824 KB
* Average Real Time: 2.474 s
* Average User Time: 2.336 s
* Average Sys Time: 0.137 s
* Average CPU Usage: 99 %
```

Keep these results in mind for later comparison. This time order of storing data in random access memory (RAM) gives us the speed we need for comparison. In practice, this speed is marely equivalent to reading something from RAM in random access.

Now let's move on to the hard disk and see how much time/resources it takes to read/write one gigabyte of data on an HDD. However, in this comparison, we are no longer concerned about other resources like the amount of RAM it consumes, but rather its time is important to us. When we compare RAM to a disk, we no longer need to measure the RAM that undergoes I/O operations.

Consider the following code:

```
#include <iostream>
#include <fstream>
#include <sys/resource.h>
#include <chrono>

using namespace std;

int main() {

    ofstream outfile;
    char character = 'A';

    // Start execution time
    clock_t start_1 = clock();
    auto start_2 = chrono::high_resolution_clock::now();

    // Start memory usage
    struct rusage usage;
    getrusage(RUSAGE_SELF, &usage);
    long memory_usage_start = usage.ru_maxrss; // in kilobytes
```

```
    // Code ------------------------------------------------------------
    outfile.open("output.txt");
    if (!outfile.is_open()) {
        cout << "Error opening file!" << endl;
        return 1;
    }
    for (int i = 0; i < 1073741824; i++) {
        outfile << character;
    }
    outfile.close();
    // ----------------------------------------------------------------

    // Stop measuring memory usage
    getrusage(RUSAGE_SELF, &usage);
    long memory_usage_end = usage.ru_maxrss; // in kilobytes

    // Stop measuring execution time
    clock_t end_1 = clock();
    auto end_2 = chrono::high_resolution_clock::now();
    double execution_time_1 = double(end_1 - start_1) / CLOCKS_PER_SEC;
    chrono::duration<double, milli> execution_time_2 = end_2 - start_2;

    // Printing result
    cout << "Execution Time (Based on ctime): " << execution_time_1 * 1000.0 << "
ms" << endl;
    cout << "Execution Time (Based on chrono): " << execution_time_2.count() << "
ms" << endl;
    cout << "Memory Usage: " << memory_usage_end - memory_usage_start << " KB" <<
endl;

    return 0;
}
```

Source code

This code opens a file named `output.txt` (creates it if it doesn't exist and overwrites it if it does {I'll return to the overwriting part later}) and writes 1 gigabyte of data into it. I will run it 10 times for the sake of experimentation:

```
┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ g++ HDD_write_test.cpp

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ rm output.txt && time ./a.out
Execution Time (Based on ctime): 9536.65 ms
Execution Time (Based on chrono): 9538.49 ms
Memory Usage: 0 KB
```

```
real    9.54s
user    8.85s
sys     0.68s
cpu     99%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ rm output.txt && time ./a.out
Execution Time (Based on ctime): 9530.77 ms
Execution Time (Based on chrono): 10150.1 ms
Memory Usage: 0 KB

real    10.15s
user    8.82s
sys     0.72s
cpu     93%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ rm output.txt && time ./a.out
Execution Time (Based on ctime): 9625.24 ms
Execution Time (Based on chrono): 10063.4 ms
Memory Usage: 0 KB

real    10.07s
user    8.99s
sys     0.64s
cpu     95%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ rm output.txt && time ./a.out
Execution Time (Based on ctime): 9655.58 ms
Execution Time (Based on chrono): 9657.09 ms
Memory Usage: 0 KB

real    9.66s
user    8.96s
sys     0.70s
cpu     99%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ rm output.txt && time ./a.out
Execution Time (Based on ctime): 9479.76 ms
Execution Time (Based on chrono): 9634.6 ms
Memory Usage: 0 KB

real    9.64s
user    8.89s
sys     0.60s
cpu     98%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ rm output.txt && time ./a.out
Execution Time (Based on ctime): 9477.25 ms
```

```
Execution Time (Based on chrono): 9641.66 ms
Memory Usage: 0 KB

real    9.65s
user    8.92s
sys     0.56s
cpu     98%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ rm output.txt && time ./a.out
Execution Time (Based on ctime): 9468.07 ms
Execution Time (Based on chrono): 9475.96 ms
Memory Usage: 0 KB

real    9.48s
user    8.70s
sys     0.77s
cpu     99%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ rm output.txt && time ./a.out
Execution Time (Based on ctime): 9761.29 ms
Execution Time (Based on chrono): 10151.8 ms
Memory Usage: 0 KB

real    10.16s
user    9.05s
sys     0.72s
cpu     96%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ rm output.txt && time ./a.out
Execution Time (Based on ctime): 9632.74 ms
Execution Time (Based on chrono): 10079 ms
Memory Usage: 0 KB

real    10.08s
user    8.92s
sys     0.72s
cpu     95%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ rm output.txt && time ./a.out
Execution Time (Based on ctime): 9536.97 ms
Execution Time (Based on chrono): 9540.68 ms
Memory Usage: 0 KB

real    9.55s
user    8.89s
sys     0.65s
cpu     99%
```

And the average result of the experiment is:

```
(9536.65+9530.77+9625.24+9655.58+9479.76+9477.25+9468.07+9761.29+9632.74+9536.97) /
10 = 95704.32 / 10 = 9570.432
(9538.49+10150.1+10063.4+9657.09+9634.6+9641.66+9475.96+10151.8+10079+9540.68) / 10
= 97932.78 / 10 = 9793.278
(0+0+0+0+0+0+0+0+0+0) / 10 = 0 / 10 = 0
(9.54s+10.15s+10.07s+9.66s+9.64s+9.65s+9.48s+10.16s+10.08s+9.55s) / 10 = 97.98 / 10
= 9.798
(8.85s+8.82s+8.99s+8.96s+8.89s+8.92s+8.70s+9.05s+8.92s+8.89s) / 10 = 88.99 / 10 =
8.899
(0.68s+0.72s+0.64s+0.70s+0.60s+0.56s+0.77s+0.72s+0.72s+0.65s) / 10 = 6.76 / 10 =
0.676
(99%+93%+95%+99%+98%+98%+99%+96%+95%+99%) / 10 = 971 / 10 = 97.1


Therefore:

* Average Execution Time(Based on ctime): 9570.432 ms
* Average Execution Time(Based on chrono): 9793.278 ms
* Average Memory Usage: 0 KB
* Average Real Time: 9.798 s
* Average User Time: 8.899 s
* Average Sys Time: 0.676 s
* Average CPU Usage: 97.1 %
```

Well, according to this HDD almost 5 times slower than RAM, isn't it?! So what was all that stuff about HDD being 10,000 times slower than RAM on average?! Don't rush, let's slowly add the factors involved in the experiment.
Let's assume we want to overwrite existing 1 gigabyte of data.

To measure this operation time on RAM, I will modify the code as follows:

```
#include <iostream>
#include <sys/resource.h>
#include <chrono>

using namespace std;

int main() {

    char* arr = new char[1073741824];
    for(int i = 0; i < 1073741824 ; i++)
        arr[i] = 'A';
```

```
    // Start execution time
    clock_t start_1 = clock();
    auto start_2 = chrono::high_resolution_clock::now();

    // Start memory usage
    struct rusage usage;
    getrusage(RUSAGE_SELF, &usage);
    long memory_usage_start = usage.ru_maxrss; // in kilobytes

    // Code ------------------------------------------------------------
    for(int i = 0; i < 1073741824 ; i++)
        arr[i] = 'B';
    // -----------------------------------------------------------------

    // Stop measuring memory usage
    getrusage(RUSAGE_SELF, &usage);
    long memory_usage_end = usage.ru_maxrss; // in kilobytes

    // Stop measuring execution time
    clock_t end_1 = clock();
    auto end_2 = chrono::high_resolution_clock::now();
    double execution_time_1 = double(end_1 - start_1) / CLOCKS_PER_SEC;
    chrono::duration<double, milli> execution_time_2 = end_2 - start_2;

    // Printing result
    cout << "Execution Time (Based on ctime): " << execution_time_1 * 1000.0 << "
ms" << endl;
    cout << "Execution Time (Based on chrono): " << execution_time_2.count() << "
ms" << endl;
    cout << "Memory Usage: " << memory_usage_end - memory_usage_start << " KB" <<
endl;

    return 0;
}
```

Source code

And we run the code:

```
┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ g++ RAM_overwrite_test.cpp

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
Execution Time (Based on ctime): 2315.81 ms
Execution Time (Based on chrono): 2315.86 ms
Memory Usage: 0 KB
```

```
real    4.76s
user    4.61s
sys     0.14s
cpu     99%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
Execution Time (Based on ctime): 2321.07 ms
Execution Time (Based on chrono): 2321.15 ms
Memory Usage: 0 KB

real    4.80s
user    4.66s
sys     0.14s
cpu     99%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
Execution Time (Based on ctime): 2305.9 ms
Execution Time (Based on chrono): 2305.95 ms
Memory Usage: 0 KB

real    4.78s
user    4.67s
sys     0.11s
cpu     99%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
Execution Time (Based on ctime): 2331.96 ms
Execution Time (Based on chrono): 2332.03 ms
Memory Usage: 0 KB

real    4.79s
user    4.70s
sys     0.09s
cpu     99%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
Execution Time (Based on ctime): 2291.69 ms
Execution Time (Based on chrono): 2296.94 ms
Memory Usage: 0 KB

real    4.74s
user    4.57s
sys     0.16s
cpu     99%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
Execution Time (Based on ctime): 2336.1 ms
```

```
Execution Time (Based on chrono): 2339.92 ms
Memory Usage: 0 KB

real    4.85s
user    4.71s
sys     0.12s
cpu     99%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
Execution Time (Based on ctime): 2330.54 ms
Execution Time (Based on chrono): 2330.6 ms
Memory Usage: 0 KB

real    4.82s
user    4.70s
sys     0.12s
cpu     99%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
Execution Time (Based on ctime): 2283.55 ms
Execution Time (Based on chrono): 2283.58 ms
Memory Usage: 0 KB

real    4.72s
user    4.59s
sys     0.12s
cpu     99%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
Execution Time (Based on ctime): 2328.87 ms
Execution Time (Based on chrono): 2328.91 ms
Memory Usage: 0 KB

real    4.79s
user    4.67s
sys     0.12s
cpu     99%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
Execution Time (Based on ctime): 2295.38 ms
Execution Time (Based on chrono): 2296.14 ms
Memory Usage: 0 KB

real    4.75s
user    4.60s
sys     0.15s
cpu     99%
```

And the average result:

```
(2315.81+2321.07+2305.9+2331.96+2291.69+2336.1+2330.54+2283.55+2328.87+2295.38) / 10
= 23140.87 / 10 = 2314.087
(2315.86+2321.15+2305.95+2332.03+2296.94+2339.92+2330.6+2283.58+2328.91+2296.14) /
10 = 23151.08 / 10 = 2315.108
(0+0+0+0+0+0+0+0+0+0) / 10 = 0 / 10 = 0
(4.76s+4.80s+4.78s+4.79s+4.74s+4.85s+4.82s+4.72s+4.79s+4.75s) / 10 = 47.8 / 10 =
4.78
(4.61s+4.66s+4.67s+4.70s+4.57s+4.71s+4.70s+4.59s+4.67s+4.60s) / 10 = 46.48 / 10 =
4.648
(0.14s+0.14s+0.11s+0.09s+0.16s+0.12s+0.12s+0.12s+0.12s+0.15s) / 10 = 1.27 / 10 =
0.127
(99%+99%+99%+99%+99%+99%+99%+99%+99%+99%) / 10 = 990 / 10 = 99


Therefore:

* Average Execution Time(Based on ctime): 2314.087 ms
* Average Execution Time(Based on chrono): 2315.108 ms
* Average Memory Usage: 0 KB
* Average Real Time: 4.78 s
* Average User Time: 4.648 s
* Average Sys Time: 0.127 s
* Average CPU Usage: 99 %
```

As you can see, the overwriting time is equal to the writing time.

Note 1: The specific part of the code we are measuring its time, is this specific segment of the code:

```
for(int i = 0; i < 1073741824 ; i++)
        arr[i] = 'B';
```

And the time is:

```
* Average Execution Time(Based on ctime): 2314.087 ms
* Average Execution Time(Based on chrono): 2315.108 ms
```

Note 2: The amount of memory consumed by this code block is reported as 0 because we did not allocate any new memory (although we had, for example, a loop counter for 'i', but it is

so small that it is not noticeable in kilobytes) - when we did not create new memory and only overwritten the previous one, no new memory usage was reported. But in reality, we worked with 1 gigabyte of memory.

Now let's move on to the hard disk. I will use the same previous code, but this time I will not delete (rm) the previously created 1-gigabyte files so that they can be overwritten.

```
┌──(user dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ g++ HDD_write_test.cpp

┌──(user dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
Execution Time (Based on ctime): 10162.1 ms
Execution Time (Based on chrono): 16532.7 ms
Memory Usage: 0 KB

real    16.54s
user    9.30s
sys     0.87s
cpu     61%

┌──(user dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
Execution Time (Based on ctime): 9961.45 ms
Execution Time (Based on chrono): 19035.1 ms
Memory Usage: 0 KB

real    19.04s
user    9.10s
sys     0.86s
cpu     52%

┌──(user dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
Execution Time (Based on ctime): 9925.02 ms
Execution Time (Based on chrono): 19885.7 ms
Memory Usage: 0 KB

real    19.89s
user    9.14s
sys     0.79s
cpu     49%

┌──(user dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
Execution Time (Based on ctime): 9927.78 ms
Execution Time (Based on chrono): 20518.4 ms
Memory Usage: 0 KB
```

```
real    20.52s
user    9.04s
sys     0.89s
cpu     48%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
Execution Time (Based on ctime): 9996.59 ms
Execution Time (Based on chrono): 17915.4 ms
Memory Usage: 0 KB

real    17.92s
user    9.16s
sys     0.84s
cpu     55%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
Execution Time (Based on ctime): 9659.78 ms
Execution Time (Based on chrono): 19085.9 ms
Memory Usage: 0 KB

real    19.09s
user    8.91s
sys     0.76s
cpu     50%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
Execution Time (Based on ctime): 10051.7 ms
Execution Time (Based on chrono): 19261.9 ms
Memory Usage: 0 KB

real    19.27s
user    9.25s
sys     0.81s
cpu     52%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
Execution Time (Based on ctime): 10124.3 ms
Execution Time (Based on chrono): 19965.3 ms
Memory Usage: 0 KB

real    19.97s
user    9.31s
sys     0.82s
cpu     50%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
Execution Time (Based on ctime): 9867.27 ms
```

```
Execution Time (Based on chrono): 19433.7 ms
Memory Usage: 0 KB

real    19.44s
user    9.05s
sys     0.82s
cpu     50%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
Execution Time (Based on ctime): 9850.91 ms
Execution Time (Based on chrono): 18046 ms
Memory Usage: 0 KB

real    18.05s
user    8.96s
sys     0.89s
cpu     54%
```

The average:

```
(10162.1+9961.45+9925.02+9927.78+9996.59+9659.78+10051.7+10124.3+9867.27+9850.91) /
10 = 99526.9 / 10 = 9952.69
(16532.7+19035.1+19885.7+20518.4+17915.4+19085.9+19261.9+19965.3+19433.7+18046) / 10
= 189680.1 / 10 = 18968.01
(0+0+0+0+0+0+0+0+0+0) / 10 = 0 / 10 = 0
(16.54s+19.04s+19.89s+20.52s+17.92s+19.09s+19.27s+19.97s+19.44s+18.05s) / 10 =
189.73 / 10 = 18.973
(9.30s+9.10s+9.14s+9.04s+9.16s+8.91s+9.25s+9.31s+9.05s+8.96s) / 10 = 91.22 / 10 =
9.122
(0.87s+0.86s+0.79s+0.89s+0.84s+0.76s+0.81s+0.82s+0.82s+0.89s) / 10 = 8.35 / 10 =
0.835
(61%+52%+49%+48%+55%+50%+52%+50%+50%+54%) / 10 = 521 / 10 = 52.1


Therefore:

* Average Execution Time(Based on ctime): 9952.69 ms
* Average Execution Time(Based on chrono): 18968.01 ms
* Average Memory Usage: 0 KB
* Average Real Time: 18.973 s
* Average User Time: 9.122 s
* Average Sys Time: 0.835 s
* Average CPU Usage: 52.1 %
```

Awwwww! The result became 10 times slower. But wait, there's more to the story...

Note 1: Look at the execution time of the first HDD overwrite and compare it to the calculated average time. You will understand why I said each experiment should be repeated multiple times.

Note 2: As you can see, the time measured by "chrono" is much higher (almost twice) than the time measured by "ctime", and it represents the actual execution time of that code block. But why? It's similar to the real, user, and sys times I mentioned when discussing the `time` command. Scientifically, these two libraries calculate time using different methods. "ctime" is a library specific to the C language, and here it calculates time using the `clock()` function, but it's not the real time. it's the time the CPU spends on executing the current process. But that statement is not entirely accurate. Let me explain in a more technical language. ctime logs the number of clocks the CPU has executed for this specific program during its computations, and then converts these clocks into seconds. Simply put, it means it doesn't include the times like when we were performing I/O operations, or waiting for resources or I/O blocking, or waiting for other processes. Bui measuring the amount of I/O time is actually a significant part of our computations! "chrono" on the other hand, has a completely different story...
The chrono library, when called on resolution methods (such as `high_resolution_clock`), calculates what is called the precise wall clock time, from the start point to the end point. This time includes interrupts and I/O operations. So this is the precise time that we use for comparing.

Now let's extend the scope of comparison a bit. Let's say we want to compare a simple copy-paste in RAM and in HDD. For RAM, consider the following code:

```
#include <iostream>
#include <sys/resource.h>
#include <chrono>

using namespace std;

int main() {

    char* arr = new char[1073741824];
    for(int i = 0; i < 1073741824 ; i++)
        arr[i] = 'A';

    // Start execution time
    clock_t start_1 = clock();
    auto start_2 = chrono::high_resolution_clock::now();

    // Start memory usage
```

```
    struct rusage usage;
    getrusage(RUSAGE_SELF, &usage);
    long memory_usage_start = usage.ru_maxrss; // in kilobytes

    // Code ------------------------------------------------------------
    char* arr_2 = new char[1073741824];
    for(int i = 0; i < 1073741824 ; i++)
        arr_2[i] = arr[i];
    // -----------------------------------------------------------------

    // Stop measuring memory usage
    getrusage(RUSAGE_SELF, &usage);
    long memory_usage_end = usage.ru_maxrss; // in kilobytes

    // Stop measuring execution time
    clock_t end_1 = clock();
    auto end_2 = chrono::high_resolution_clock::now();
    double execution_time_1 = double(end_1 - start_1) / CLOCKS_PER_SEC;
    chrono::duration<double, milli> execution_time_2 = end_2 - start_2;

    // Printing result
    cout << "Execution Time (Based on ctime): " << execution_time_1 * 1000.0 << "
ms" << endl;
    cout << "Execution Time (Based on chrono): " << execution_time_2.count() << "
ms" << endl;
    cout << "Memory Usage: " << memory_usage_end - memory_usage_start << " KB" <<
endl;

    return 0;
}
```

Source code

And let's jump right into execution:

```
┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ g++ RAM_copy_test.cpp

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
Execution Time (Based on ctime): 2676.01 ms
Execution Time (Based on chrono): 2676.22 ms
Memory Usage: 1048824 KB

real    5.12s
user    4.89s
sys     0.23s
```

```
cpu     99%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
Execution Time (Based on ctime): 2616.64 ms
Execution Time (Based on chrono): 2616.71 ms
Memory Usage: 1048764 KB

real    5.06s
user    4.85s
sys     0.20s
cpu     99%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
Execution Time (Based on ctime): 2618.32 ms
Execution Time (Based on chrono): 2618.4 ms
Memory Usage: 1048712 KB

real    5.02s
user    4.81s
sys     0.20s
cpu     99%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
Execution Time (Based on ctime): 2616.12 ms
Execution Time (Based on chrono): 2616.2 ms
Memory Usage: 1048824 KB

real    5.05s
user    4.80s
sys     0.24s
cpu     99%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
Execution Time (Based on ctime): 2617.68 ms
Execution Time (Based on chrono): 2617.75 ms
Memory Usage: 1048824 KB

real    5.05s
user    4.88s
sys     0.17s
cpu     99%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
Execution Time (Based on ctime): 2620.01 ms
Execution Time (Based on chrono): 2620.11 ms
Memory Usage: 1048824 KB
```

```
real    5.06s
user    4.86s
sys     0.20s
cpu     99%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
Execution Time (Based on ctime): 2620.11 ms
Execution Time (Based on chrono): 2620.16 ms
Memory Usage: 1048824 KB

real    5.02s
user    4.79s
sys     0.23s
cpu     99%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
Execution Time (Based on ctime): 2615.65 ms
Execution Time (Based on chrono): 2615.84 ms
Memory Usage: 1048764 KB

real    5.03s
user    4.78s
sys     0.25s
cpu     99%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
Execution Time (Based on ctime): 2623.07 ms
Execution Time (Based on chrono): 2623.13 ms
Memory Usage: 1048780 KB

real    5.03s
user    4.86s
sys     0.17s
cpu     99%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
Execution Time (Based on ctime): 2616.85 ms
Execution Time (Based on chrono): 2616.94 ms
Memory Usage: 1048780 KB

real    5.07s
user    4.87s
sys     0.20s
cpu     99%
```

And the average:

```
(2676.01+2616.64+2618.32+2616.12+2617.68+2620.01+2620.11+2615.65+2623.07+2616.85) /
10 = 26240.46 / 10 = 2624.046
(2676.22+2616.71+2618.4+2616.2+2617.75+2620.11+2620.16+2615.84+2623.13+2616.94) / 10
= 26241.46 / 10 = 2624.146
(1048824+1048764+1048712+1048824+1048824+1048824+1048824+1048764+1048780+1048780) /
10 = 10487920 / 10 = 1048792
(5.12s+5.06s+5.02s+5.05s+5.05s+5.06s+5.02s+5.03s+5.03s+5.07s) / 10 = 50.51 / 10 =
5.051
(4.89s+4.85s+4.81s+4.80s+4.88s+4.86s+4.79s+4.78s+4.86s+4.87s) / 10 = 48.39 / 10 =
4.839
(0.23s+0.20s+0.20s+0.24s+0.17s+0.20s+0.23s+0.25s+0.17s+0.20s) / 10 = 2.09 / 10 =
0.209
(99%+99%+99%+99%+99%+99%+99%+99%+99%+99%) / 10 = 990 / 10 = 99


Therefore:

* Average Execution Time(Based on ctime): 2624.046 ms
* Average Execution Time(Based on chrono): 2624.146 ms
* Average Memory Usage: 1048792 KB
* Average Real Time: 5.051 s
* Average User Time: 4.839 s
* Average Sys Time: 0.209 s
* Average CPU Usage: 99 %
```

Now let's copy that same 1 gigabyte of data from the disk, but to a different location on the same disk, let's say in another partition ^_^:

```
#include <iostream>
#include <fstream>
#include <sys/resource.h>
#include <chrono>

using namespace std;

int main() {

    char chunk;
    ifstream infile;
    ofstream outfile;

    // Start execution time
    clock_t start_1 = clock();
    auto start_2 = chrono::high_resolution_clock::now();

    // Start memory usage
    struct rusage usage;
```

```
    getrusage(RUSAGE_SELF, &usage);
    long memory_usage_start = usage.ru_maxrss; // in kilobytes

    // Code ------------------------------------------------------------
    infile.open("output.txt");
    outfile.open("/media/user/MyDrive1/output_copy.txt");
    if (!outfile.is_open()) {
        cout << "Error opening file!" << endl;
        return 1;
    }
    while (infile.get(chunk))
        outfile.put(chunk);
    infile.close();
    outfile.close();
    // -----------------------------------------------------------------

    // Stop measuring memory usage
    getrusage(RUSAGE_SELF, &usage);
    long memory_usage_end = usage.ru_maxrss; // in kilobytes

    // Stop measuring execution time
    clock_t end_1 = clock();
    auto end_2 = chrono::high_resolution_clock::now();
    double execution_time_1 = double(end_1 - start_1) / CLOCKS_PER_SEC;
    chrono::duration<double, milli> execution_time_2 = end_2 - start_2;

    // Printing result
    cout << "Execution Time (Based on ctime): " << execution_time_1 * 1000.0 << "
ms" << endl;
    cout << "Execution Time (Based on chrono): " << execution_time_2.count() << "
ms" << endl;
    cout << "Memory Usage: " << memory_usage_end - memory_usage_start << " KB" <<
endl;

    return 0;
}
```

Source code

Execution:

```
┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ g++ HDD_copy_test.cpp

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
Execution Time (Based on ctime): 18386.5 ms
```

```
Execution Time (Based on chrono): 25923.6 ms
Memory Usage: 0 KB

real    25.93s
user    16.58s
sys     1.81s
cpu     70%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ rm /media/user/MyDrive1/output_copy.txt && !!

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ rm /media/user/MyDrive1/output_copy.txt && time ./a.out
Execution Time (Based on ctime): 17681.3 ms
Execution Time (Based on chrono): 23741.9 ms
Memory Usage: 0 KB

real    23.74s
user    15.98s
sys     1.70s
cpu     74%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ rm /media/user/MyDrive1/output_copy.txt && time ./a.out
Execution Time (Based on ctime): 17245.1 ms
Execution Time (Based on chrono): 24897.7 ms
Memory Usage: 0 KB

real    24.90s
user    15.49s
sys     1.76s
cpu     69%


┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ rm output.txt && dd if=/dev/random bs=1024 count=1048576 > output.txt
1048576+0 records in
1048576+0 records out
1073741824 bytes (1.1 GB, 1.0 GiB) copied, 10.1874 s, 105 MB/s

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ rm /media/user/MyDrive1/output_copy.txt && time ./a.out
Execution Time (Based on ctime): 19407.6 ms
Execution Time (Based on chrono): 33033.6 ms
Memory Usage: 0 KB

real    33.04s
user    17.42s
sys     1.99s
cpu     58%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
```

```
└─$ rm output.txt && dd if=/dev/random bs=1024 count=1048576 > output.txt
1048576+0 records in
1048576+0 records out
1073741824 bytes (1.1 GB, 1.0 GiB) copied, 10.4907 s, 102 MB/s

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ rm /media/user/MyDrive1/output_copy.txt && time ./a.out
Execution Time (Based on ctime): 20163.5 ms
Execution Time (Based on chrono): 36565.9 ms
Memory Usage: 0 KB

real    36.57s
user    18.03s
sys     2.13s
cpu     55%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ rm output.txt && dd if=/dev/random bs=1024 count=1048576 > output.txt
1048576+0 records in
1048576+0 records out
1073741824 bytes (1.1 GB, 1.0 GiB) copied, 10.1993 s, 105 MB/s

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ rm /media/user/MyDrive1/output_copy.txt && time ./a.out
Execution Time (Based on ctime): 20450.9 ms
Execution Time (Based on chrono): 38170.3 ms
Memory Usage: 0 KB

real    38.17s
user    18.30s
sys     2.15s
cpu     53%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ rm output.txt && dd if=/dev/random bs=1024 count=1048576 > output.txt
1048576+0 records in
1048576+0 records out
1073741824 bytes (1.1 GB, 1.0 GiB) copied, 10.1726 s, 106 MB/s

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ rm /media/user/MyDrive1/output_copy.txt && time ./a.out
Execution Time (Based on ctime): 20228.9 ms
Execution Time (Based on chrono): 33791.1 ms
Memory Usage: 0 KB

real    33.79s
user    18.12s
sys     2.11s
cpu     59%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ rm output.txt && dd if=/dev/random bs=1024 count=1048576 > output.txt
```

```
1048576+0 records in
1048576+0 records out
1073741824 bytes (1.1 GB, 1.0 GiB) copied, 10.2377 s, 105 MB/s

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ rm /media/user/MyDrive1/output_copy.txt && time ./a.out
Execution Time (Based on ctime): 20001.4 ms
Execution Time (Based on chrono): 35478 ms
Memory Usage: 0 KB

real    35.48s
user    17.79s
sys     2.21s
cpu     56%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ rm output.txt && dd if=/dev/random bs=1024 count=1048576 > output.txt
1048576+0 records in
1048576+0 records out
1073741824 bytes (1.1 GB, 1.0 GiB) copied, 10.5428 s, 102 MB/s

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ rm /media/user/MyDrive1/output_copy.txt && time ./a.out
Execution Time (Based on ctime): 19954.7 ms
Execution Time (Based on chrono): 34560.4 ms
Memory Usage: 0 KB

real    34.56s
user    17.55s
sys     2.41s
cpu     57%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ rm output.txt && dd if=/dev/random bs=1024 count=1048576 > output.txt
1048576+0 records in
1048576+0 records out
1073741824 bytes (1.1 GB, 1.0 GiB) copied, 10.1674 s, 106 MB/s

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ rm /media/user/MyDrive1/output_copy.txt && time ./a.out
Execution Time (Based on ctime): 21618.5 ms
Execution Time (Based on chrono): 39370.6 ms
Memory Usage: 0 KB

real    39.37s
user    19.18s
sys     2.44s
cpu     54%
```

Average time:

```
(18386.5+17681.3+17245.1+19407.6+20163.5+20450.9+20228.9+20001.4+19954.7+21618.5) /
10 = 195138.4 / 10 = 19513.84
(25923.6+23741.9+24897.7+33033.6+36565.9+38170.3+33791.1+35478+34560.4+39370.6) / 10
= 325533.1 / 10 = 32553.31
(0+0+0+0+0+0+0+0+0+0) / 10 = 0 / 10 = 0
(25.93s+23.74s+24.90s+33.04s+36.57s+38.17s+33.79s+35.48s+34.56s+39.37s) / 10 =
325.55 / 10 = 32.555
(16.58s+15.98s+15.49s+17.42s+18.03s+18.30s+18.12s+17.79s+17.55s+19.18s) / 10 =
174.44 / 10 = 17.444
(1.81s+1.70s+1.76s+1.99s+2.13s+2.15s+2.11s+2.21s+2.41s+2.44s) / 10 = 20.71 / 10 =
2.071
(70%+74%+69%+58%+55%+53%+59%+56%+57%+54%) / 10 = 605 / 10 = 60.5


Therefore:

* Average Execution Time(Based on ctime): 19513.84 ms
* Average Execution Time(Based on chrono): 32553.31 ms
* Average Memory Usage: 0 KB
* Average Real Time: 32.555 s
* Average User Time: 17.444 s
* Average Sys Time: 2.071 s
* Average CPU Usage: 60.5 %
```

Here, I performed a subtle trick:))

I'll explain it in technical language only. By ovserving files getting copied quickly within a range of 20 seconds, I realized that an optimization is goin on under the hood. It's possible that the file has been cached by an entity - the operating system or I/O-related API modules - (and we can see how practical concepts are approving theoretical ones). And since the file entropy is very low (because it's all full of ASCII bytes of "A" character), our experiment may be influenced by the optimizations I explained earlier theoretically. Therefore, for this purpose, I obtained 1 gigabyte of random data from the system entropy instead of our low-entropy file and observed that the time became much longer (up to 20 times longer).

```
rm output.txt && dd if=/dev/random bs=1024 count=1048576 > output.txt
```

I want to do the final comparing and change the workflow. For this matter, we compare HDD and RAM in a way that several processes consume resources in parallel and interact directly with the hard disk or RAM. We want to see how the servicing time decreases when our storage device is under shared access, both in RAM and disk.

From what we discussed in theory, we know that RAM shouldn't really make much of difference in its behaviour and its speed is not heavily affected by this matter, so it shouldn't slow down significantly. However, the disk on the other hand, due to its physical movements, is likely to be severely affected in terms of speed. Let's go and see it in practice:

The following program eats the RAM! It takes 1 gigabyte of RAM and repeatedly refills and makes it empty.

```
#include <iostream>

using namespace std;

int main() {

    cout << "EATING RAM!!!" << endl;
    while( true )
    {
        char* arr = new char[1073741824];
        for(int i = 0; i < 1073741824 ; i++)
            arr[i] = 'A';
        delete arr;
    }

    return 0;
}
```

Source code

I run four instances of this program in parallel at the same time:

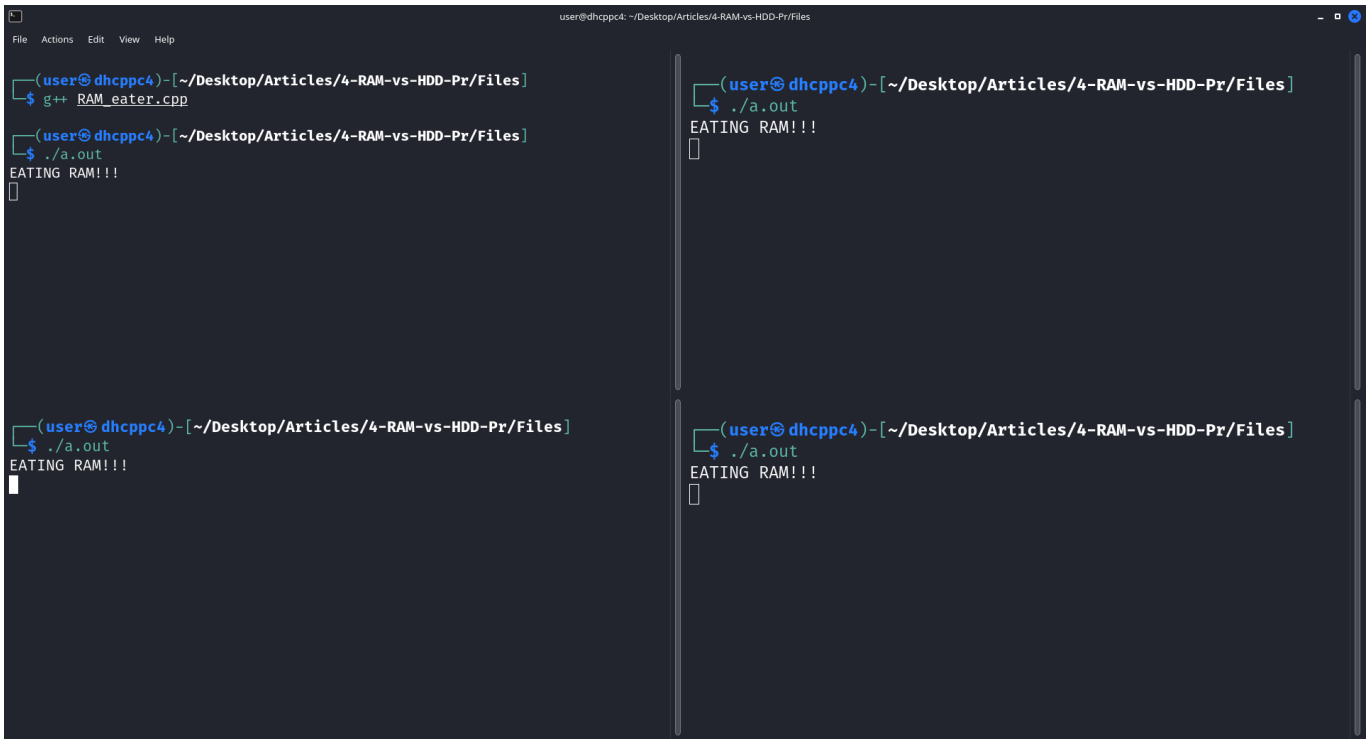
```
┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ g++ RAM_eater.cpp

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ ./a.out
EATING RAM!!!
```

```
┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ ./a.out
EATING RAM!!!
```

```
┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ ./a.out
```

```
EATING RAM!!!
```

```
┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ ./a.out
EATING RAM!!!
```



And the output of the `top` command when four instances of the memory-consuming process are running:

```
top - 18:36:01 up  5:32,  3 users,  load average: 4.15, 2.75, 1.43
Tasks: 262 total,   6 running, 256 sleeping,   0 stopped,   0 zombie
%Cpu(s): 47.8 us,  2.7 sy,  0.0 ni, 49.4 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :  11857.5 total,    357.3 free,   6196.1 used,   5304.1 buff/cache
MiB Swap:      0.0 total,      0.0 free,      0.0 used.   4796.5 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
   9778 user      20   0 1054380 606472   2976 R 100.0   5.0   4:38.14 a.out
   9781 user      20   0 1054380 650644   2976 R 100.0   5.4   4:36.12 a.out
   9782 user      20   0 1054380 999648   2984 R 100.0   8.2   4:35.26 a.out
   9780 user      20   0 1054380 891044   2976 R  99.7   7.3   4:37.18 a.out
   3630 user      20   0 3830408 637764 259504 S   1.7   5.3   6:02.55 firefox-esr
   9646 user      20   0 2481200 138708  99384 S   1.0   1.1   0:13.06 file:// Content
  10043 user      20   0 1366756 131252  99252 S   0.7   1.1   0:03.90 qterminal
     69 root      39  19       0      0      0 S   0.3   0.0   0:01.57 khugepaged
    942 root      20   0  790648 222772 198516 S   0.3   1.8   2:34.06 Xorg
   1879 user      20   0  286516  40608  31964 S   0.3   0.3   0:02.03 kglobalaccel5
   3835 user      20   0 2605300 152900  93456 S   0.3   1.3   1:34.58 WebExtensions
   4069 user      20   0 2857660 314132 152772 S   0.3   2.6   2:42.75 Web Content
   4485 user      20   0 2534784 177768  96788 S   0.3   1.5   0:25.08 Web Content
  10121 user      20   0   10532   3996   3092 R   0.3   0.0   0:00.11 top
      1 root      20   0  166548  11964   8308 S   0.0   0.1   0:03.26 systemd
      2 root      20   0       0      0      0 S   0.0   0.0   0:00.02 kthreadd
      3 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_gp
```

Now, concurrently with these, I run our first program, which is this:

```
┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ g++ RAM_write_test.cpp -o test.out

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./test.out
Execution Time (Based on ctime): 3638.12 ms
Execution Time (Based on chrono): 3638.24 ms
Memory Usage: 1047004 KB

real    3.67s
user    3.45s
sys     0.22s
cpu     99%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./test.out
Execution Time (Based on ctime): 3539.23 ms
Execution Time (Based on chrono): 3539.71 ms
Memory Usage: 1046948 KB

real    3.56s
user    3.36s
sys     0.20s
cpu     99%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./test.out
```

```
Execution Time (Based on ctime): 3587.88 ms
Execution Time (Based on chrono): 3588.17 ms
Memory Usage: 1046956 KB

real    3.63s
user    3.43s
sys     0.20s
cpu     99%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./test.out
Execution Time (Based on ctime): 3629.68 ms
Execution Time (Based on chrono): 3630.07 ms
Memory Usage: 1046968 KB

real    3.67s
user    3.43s
sys     0.24s
cpu     99%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./test.out
Execution Time (Based on ctime): 3568.61 ms
Execution Time (Based on chrono): 3568.84 ms
Memory Usage: 1046920 KB

real    3.59s
user    3.39s
sys     0.21s
cpu     99%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./test.out
Execution Time (Based on ctime): 3578.81 ms
Execution Time (Based on chrono): 3578.97 ms
Memory Usage: 1046828 KB

real    3.61s
user    3.39s
sys     0.22s
cpu     99%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./test.out
Execution Time (Based on ctime): 3585.52 ms
Execution Time (Based on chrono): 3585.62 ms
Memory Usage: 1046760 KB

real    3.61s
user    3.33s
sys     0.28s
cpu     99%
```

```
┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./test.out
Execution Time (Based on ctime): 3566.91 ms
Execution Time (Based on chrono): 3567.07 ms
Memory Usage: 1046944 KB

real    3.60s
user    3.37s
sys     0.22s
cpu     99%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./test.out
Execution Time (Based on ctime): 3594.36 ms
Execution Time (Based on chrono): 3594.45 ms
Memory Usage: 1046972 KB

real    3.63s
user    3.39s
sys     0.24s
cpu     99%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./test.out
Execution Time (Based on ctime): 3584.68 ms
Execution Time (Based on chrono): 3588.76 ms
Memory Usage: 1046840 KB

real    3.62s
user    3.35s
sys     0.27s
cpu     99%
```

And Calculating average:

```
(3638.12+3539.23+3587.88+3629.68+3568.61+3578.81+3585.52+3566.91+3594.36+3584.68) /
10 = 35873.8 / 10 = 3587.38
(3638.24+3539.71+3588.17+3630.07+3568.84+3578.97+3585.62+3567.07+3594.45+3588.76) /
10 = 35879.9 / 10 = 3587.99
(1047004+1046948+1046956+1046968+1046920+1046828+1046760+1046944+1046972+1046840) /
10 = 10469140 / 10 = 1046914
(3.67s+3.56s+3.63s+3.67s+3.59s+3.61s+3.61s+3.60s+3.63s+3.62s) / 10 = 36.19 / 10 =
3.619
(3.45s+3.36s+3.43s+3.43s+3.39s+3.39s+3.33s+3.37s+3.39s+3.35s) / 10 = 33.89 / 10 =
3.389
(0.22s+0.20s+0.20s+0.24s+0.21s+0.22s+0.28s+0.22s+0.24s+0.27s) / 10 = 2.3 / 10 = 0.23
(99%+99%+99%+99%+99%+99%+99%+99%+99%+99%) / 10 = 990 / 10 = 99


Therefore:
```

```
* Average Execution Time(Based on ctime): 3587.38 ms
* Average Execution Time(Based on chrono): 3587.99 ms
* Average Memory Usage: 1046914 KB
* Average Real Time: 3.619 s
* Average User Time: 3.389 s
* Average Sys Time: 0.23 s
* Average CPU Usage: 99 %
```

We can see that the average time has increased approximately by 1.5 times. However, this increase in time does not necessarily mean that the RAM chips have become slower. It could be due to the congestion of the RAM bus, the congestion of the operating system's RAM module, the congestion of the API that processes interact with for memory allocation, or even the congestion of the processor itself!
To make sure it's not because of the processor, I repeat the above experiment in a multithreading or multicore manner, and I put each memory-consuming process on a separate core of the CPU, and also put the test process on a separate core (altogether 5 cores).
In simple terms, a multithreaded program assigns different computation threads to different cores. With this approach, we can ensure that the slowdown is not only due to CPU congestion... (Later, we will discuss multithreading or multiprocessing in detail in an article.)

Consider the following program:

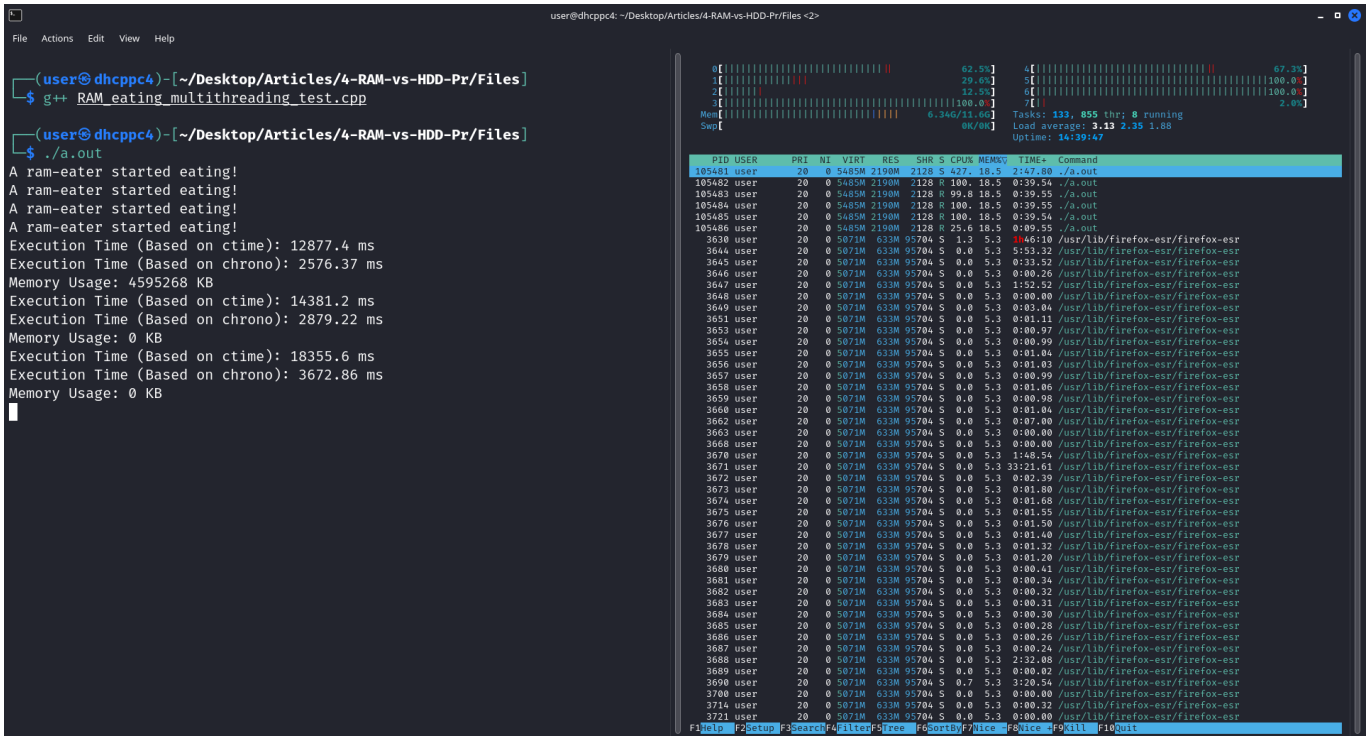
```
#include <iostream>
#include <sys/resource.h>
#include <chrono>
#include <thread>
#include <vector>

using namespace std;

void ram_eater()
{
    cout << "A ram-eater started eating!" << endl;
    while( true )
    {
        char* arr = new char[1073741824];
        for(int i = 0; i < 1073741824 ; i++)
            arr[i] = 'A';
        delete arr;
    }
}
```

```
void test()
{
    for( int i = 0; i < 10; i++ )
    {
        // Start execution time
        clock_t start_1 = clock();
        auto start_2 = chrono::high_resolution_clock::now();

        // Start memory usage
        struct rusage usage;
        getrusage(RUSAGE_SELF, &usage);
        long memory_usage_start = usage.ru_maxrss; // in kilobytes

        // Code ----------------------------------------------------------
        char* arr = new char[1073741824];
        for(int j = 0; j < 1073741824 ; j++)
            arr[j] = 'A';

        // ---------------------------------------------------------------

        // Stop measuring memory usage
        getrusage(RUSAGE_SELF, &usage);
        long memory_usage_end = usage.ru_maxrss; // in kilobytes

        // Stop measuring execution time
        clock_t end_1 = clock();
        auto end_2 = chrono::high_resolution_clock::now();
        double execution_time_1 = double(end_1 - start_1) / CLOCKS_PER_SEC;
        chrono::duration<double, milli> execution_time_2 = end_2 - start_2;

        // Printing result
        cout << "Execution Time (Based on ctime): " << execution_time_1 * 1000.0 <<
" ms" << endl;
        cout << "Execution Time (Based on chrono): " << execution_time_2.count() <<
" ms" << endl;
        cout << "Memory Usage: " << memory_usage_end - memory_usage_start << " KB"
<< endl;

        // Free memory to prevent memory-leak
        delete arr;

        // Sleep for 10 seconds
        this_thread::sleep_for(chrono::milliseconds(10000));
    }
}

int main() {
    vector<thread> threads;

    threads.emplace_back(ram_eater);
    threads.emplace_back(ram_eater);
    threads.emplace_back(ram_eater);
```

```
    threads.emplace_back(ram_eater);

    threads.emplace_back(test);

    // Wait for test to be over
    threads[4].join();

    return 0;
}
```

[Source code]

Execution:




```
┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ g++ RAM_eating_multithreading_test.cpp

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ ./a.out
A ram-eater started eating!
A ram-eater started eating!
A ram-eater started eating!
A ram-eater started eating!
Execution Time (Based on ctime): 12877.4 ms
-Execution Time (Based on chrono): 2576.37 ms
```

```
Memory Usage: 4595268 KB
Execution Time (Based on ctime): 14381.2 ms
-Execution Time (Based on chrono): 2879.22 ms
Memory Usage: 0 KB
Execution Time (Based on ctime): 18355.6 ms
-Execution Time (Based on chrono): 3672.86 ms
Memory Usage: 0 KB
Execution Time (Based on ctime): 18452.8 ms
-Execution Time (Based on chrono): 3692.51 ms
Memory Usage: 0 KB
Execution Time (Based on ctime): 17189.4 ms
-Execution Time (Based on chrono): 3442.85 ms
Memory Usage: 0 KB
Execution Time (Based on ctime): 18136.1 ms
-Execution Time (Based on chrono): 3625.81 ms
Memory Usage: 0 KB
Execution Time (Based on ctime): 17982.1 ms
-Execution Time (Based on chrono): 3599.67 ms
Memory Usage: 0 KB
Execution Time (Based on ctime): 18238.8 ms
-Execution Time (Based on chrono): 3648.24 ms
Memory Usage: 0 KB
Execution Time (Based on ctime): 18639.8 ms
-Execution Time (Based on chrono): 3729.46 ms
Memory Usage: 0 KB
Execution Time (Based on ctime): 18598.2 ms
-Execution Time (Based on chrono): 3719.59 ms
Memory Usage: 0 KB
terminate called without an active exception
zsh: IOT instruction  ./a.out
```

You see that the program is running on multiple cores, and since everything is under one process, the `ctime` reported time and the amount of memory consumed by the irrelevant data are not relevant to our experiment. Only the `chrono` time, marked in red (as a Markdown symbol), is the accurate time that should be involved in the comparison. The average time is as follows:

```
(2576.37+2879.22+3672.86+3692.51+3442.85+3625.81+3599.67+3648.24+3729.46+3719.59) /
10 = 34586.58 / 10 = 3458.658

Therefore:

* Average Execution Time(Based on chrono): 3458.658 ms
```

The average time obtained in a multithreaded manner is `3458.658` milliseconds, which is approximately equal to the average time of multiprocessing, which is `3587.99` milliseconds.

So, we can confidently say that, on average, by allocating 100 milliseconds to processor overhead, RAM under a Unix system, during shared access - in a deadly manner! - becomes only 1.5 times slower.

Now let's move on to the same shared access on the hard disk. For this purpose, we are writing a disk-consuming program:

```
#include <iostream>
#include <fstream>
#include <signal.h>
#include <ctime>

using namespace std;

ofstream outfile;

void signalHandler(int signum) {
    outfile.close();
    cout << "Interrupt signal received. File closed." << endl;
    exit(signum);
}

int main() {
    signal(SIGINT, signalHandler);
    srand(time(NULL));
    char character = 'A';
    string filename = to_string(rand()) + ".txt";

    cout << "EATING HDD!!!" << endl;


    while (true) {

        outfile.open(filename);
        if (!outfile.is_open()) {
            cout << "Error opening file: " << filename << endl;
            return 1;
        }

        for (int i = 0; i < 1073741824; i++) {
            outfile << character;
        }

        outfile.close();
    }

    return 0;
```

```
}
```

Source code

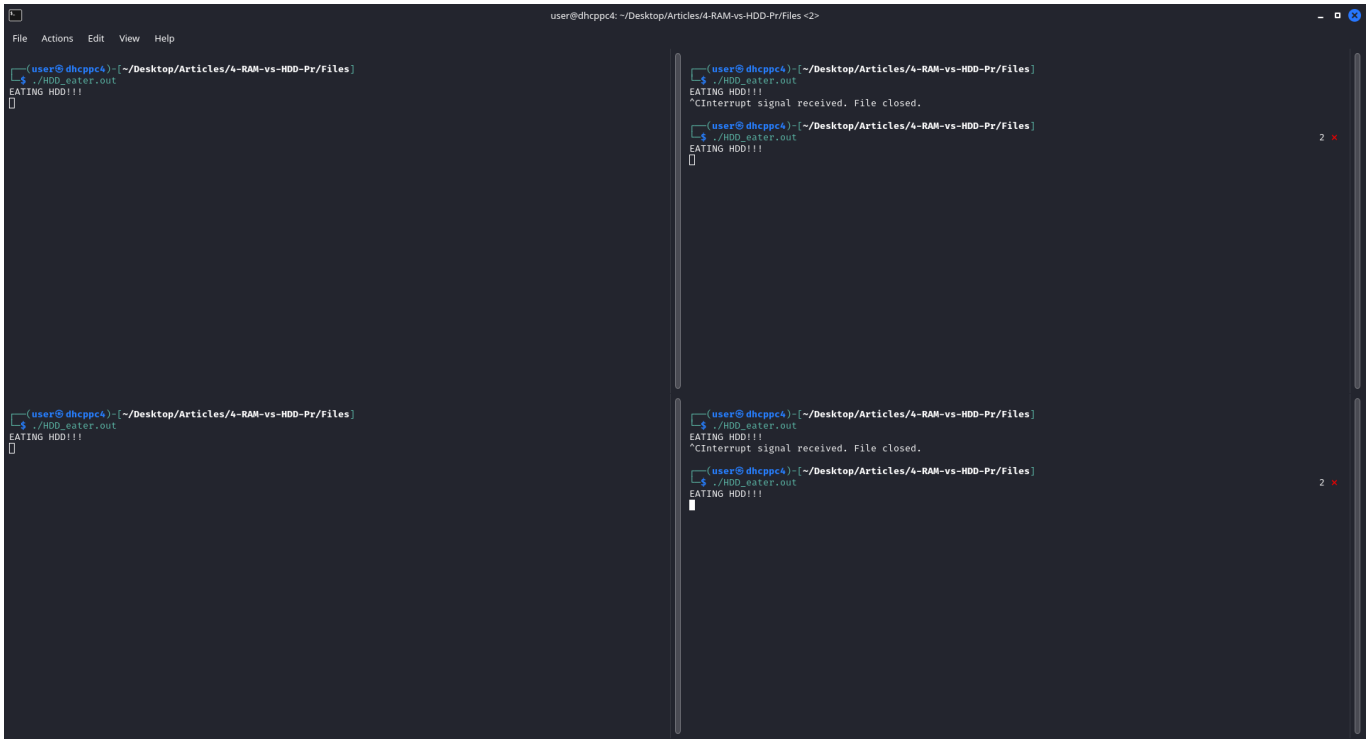
Let me give you a brief explanation. This program creates a file with random names and writes 1 gigabyte of data into it. When the writing process is complete, it starts over and writes from the beginning again. This process continues indefinitely unless a interrup-signal is sent to the program. Upon receiving the interruption, it closes the file and the program finishes.

```
┌──(user dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ g++ HDD_eater.cpp -o HDD_eater.out

┌──(user dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ ./HDD_eater.out
EATING HDD!!!
```

```
┌──(user dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ ./HDD_eater.out
EATING HDD!!!
```

```
┌──(user dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ ./HDD_eater.out
EATING HDD!!!
```

```
┌──(user dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ ./HDD_eater.out
EATING HDD!!!
```



And during this process, I am running this program:

```
┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ g++ HDD_write_test.cpp -o HDD_write.out

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./HDD_write.out
Execution Time (Based on ctime): 11230.8 ms
Execution Time (Based on chrono): 25468.1 ms
Memory Usage: 0 KB

real    25.48s
user    10.30s
sys     0.93s
cpu     44%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./HDD_write.out
Execution Time (Based on ctime): 11096.1 ms
Execution Time (Based on chrono): 20764 ms
Memory Usage: 0 KB

real    20.77s
user    10.16s
sys     0.94s
cpu     53%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./HDD_write.out
```

```
Execution Time (Based on ctime): 13815.2 ms
Execution Time (Based on chrono): 32682 ms
Memory Usage: 0 KB

real    32.68s
user    12.63s
sys     1.18s
cpu     42%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./HDD_write.out
Execution Time (Based on ctime): 15544.6 ms
Execution Time (Based on chrono): 47560 ms
Memory Usage: 0 KB

real    47.57s
user    13.91s
sys     1.64s
cpu     32%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./HDD_write.out
Execution Time (Based on ctime): 16873.7 ms
Execution Time (Based on chrono): 45354.9 ms
Memory Usage: 0 KB

real    45.36s
user    15.25s
sys     1.63s
cpu     37%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./HDD_write.out
Execution Time (Based on ctime): 15391.3 ms
Execution Time (Based on chrono): 39712.6 ms
Memory Usage: 0 KB

real    39.92s
user    13.94s
sys     1.46s
cpu     38%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./HDD_write.out
Execution Time (Based on ctime): 14051.7 ms
Execution Time (Based on chrono): 29361.3 ms
Memory Usage: 0 KB

real    29.36s
user    12.61s
sys     1.44s
cpu     47%
```

```
┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./HDD_write.out
Execution Time (Based on ctime): 13137.5 ms
Execution Time (Based on chrono): 35374 ms
Memory Usage: 0 KB

real    35.64s
user    11.94s
sys     1.21s
cpu     36%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./HDD_write.out
Execution Time (Based on ctime): 13510.7 ms
Execution Time (Based on chrono): 28392.7 ms
Memory Usage: 0 KB

real    28.40s
user    12.42s
sys     1.10s
cpu     47%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./HDD_write.out
Execution Time (Based on ctime): 13595 ms
Execution Time (Based on chrono): 44944.2 ms
Memory Usage: 0 KB

real    44.95s
user    12.24s
sys     1.35s
cpu     30%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./HDD_write.out
Execution Time (Based on ctime): 18275.6 ms
Execution Time (Based on chrono): 70660.9 ms
Memory Usage: 0 KB

real    70.66s
user    16.16s
sys     2.11s
cpu     25%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./HDD_write.out
Execution Time (Based on ctime): 17255 ms
Execution Time (Based on chrono): 49479.4 ms
Memory Usage: 0 KB

real    49.48s
user    15.43s
```

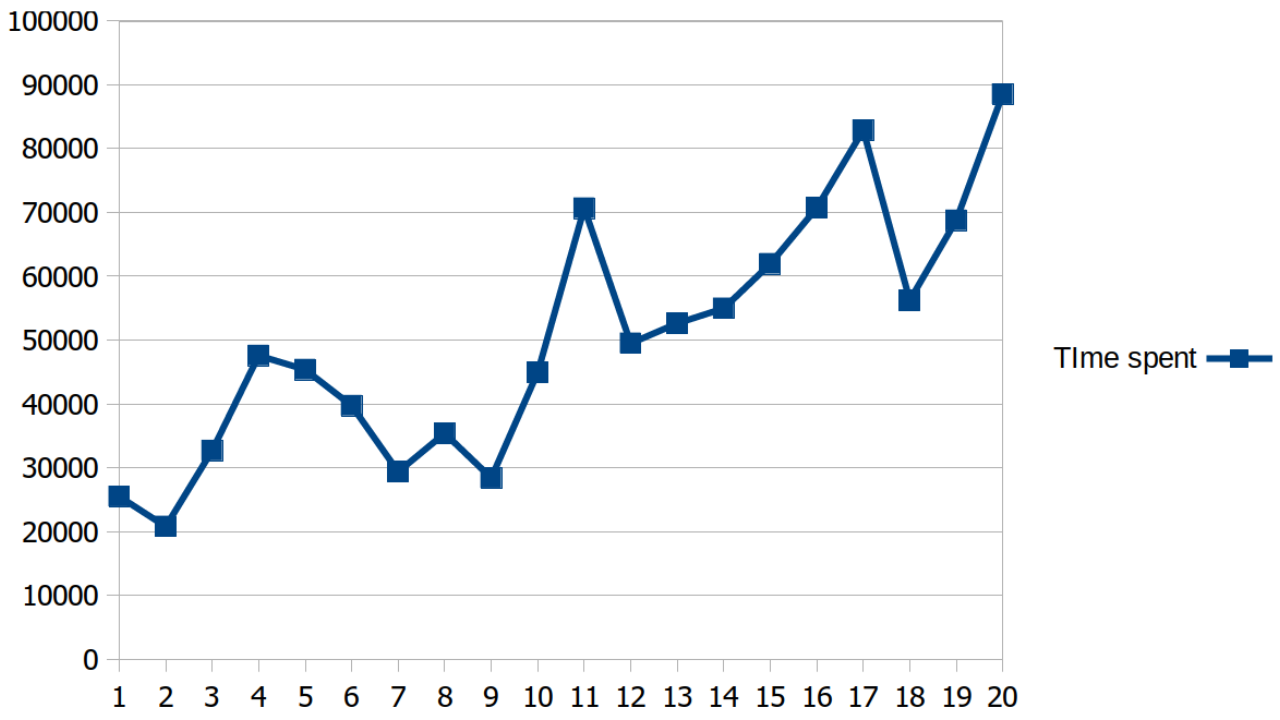
```
sys     1.82s
cpu     34%

┌──(user dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./HDD_write.out
Execution Time (Based on ctime): 15195.9 ms
Execution Time (Based on chrono): 52613.5 ms
Memory Usage: 0 KB

real    52.62s
user    13.64s
sys     1.56s
cpu     28%

┌──(user dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./HDD_write.out
Execution Time (Based on ctime): 14688.6 ms
Execution Time (Based on chrono): 54965.7 ms
Memory Usage: 0 KB

real    54.97s
user    13.33s
sys     1.36s
cpu     26%

┌──(user dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./HDD_write.out
Execution Time (Based on ctime): 18287.1 ms
Execution Time (Based on chrono): 61944 ms
Memory Usage: 0 KB

real    61.95s
user    16.30s
sys     1.98s
cpu     29%

┌──(user dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./HDD_write.out
Execution Time (Based on ctime): 21118.9 ms
Execution Time (Based on chrono): 70709.4 ms
Memory Usage: 0 KB

real    70.71s
user    19.07s
sys     2.05s
cpu     29%

┌──(user dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./HDD_write.out
Execution Time (Based on ctime): 24345 ms
Execution Time (Based on chrono): 82900.9 ms
Memory Usage: 0 KB
```

```
real    82.90s
user    21.94s
sys     2.41s
cpu     29%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./HDD_write.out
Execution Time (Based on ctime): 18640.5 ms
Execution Time (Based on chrono): 56225.7 ms
Memory Usage: 0 KB

real    56.23s
user    16.70s
sys     1.94s
cpu     33%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./HDD_write.out
Execution Time (Based on ctime): 17671.1 ms
Execution Time (Based on chrono): 68766 ms
Memory Usage: 0 KB

real    68.77s
user    16.09s
sys     1.58s
cpu     25%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./HDD_write.out
Execution Time (Based on ctime): 22847.9 ms
Execution Time (Based on chrono): 88525.2 ms
Memory Usage: 0 KB

real    88.53s
user    20.43s
sys     2.42s
cpu     25%
```

```
(11230.8+11096.1+13815.2+15544.6+16873.7+15391.3+14051.7+13137.5+13510.7+13595+18275
.6+17255+15195.9+14688.6+18287.1+21118.9+24345+18640.5+17671.1+22847.9) / 20 =
326572.2 / 20 = 16328.61
(25468.1+20764+32682+47560+45354.9+39712.6+29361.3+35374+28392.7+44944.2+70660.9+494
79.4+52613.5+54965.7+61944+70709.4+82900.9+56225.7+68766+88525.2) / 20 = 1006404.5 /
20 = 50320.225
(0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0) / 20 = 0 / 20 = 0
(25.48s+20.77s+32.68s+47.57s+45.36s+39.92s+29.36s+35.64s+28.40s+44.95s+70.66s+49.48s
+52.62s+54.97s+61.95s+70.71s+82.90s+56.23s+68.77s+88.53s) / 20 = 1006.95 / 20 =
50.3475
(10.30s+10.16s+12.63s+13.91s+15.25s+13.94s+12.61s+11.94s+12.42s+12.24s+16.16s+15.43s
+13.64s+13.33s+16.30s+19.07s+21.94s+16.70s+16.09s+20.43s) / 20 = 294.49 / 20 =
14.7245
(0.93s+0.94s+1.18s+1.64s+1.63s+1.46s+1.44s+1.21s+1.10s+1.35s+2.11s+1.82s+1.56s+1.36s
```

```
+1.98s+2.05s+2.41s+1.94s+1.58s+2.42s) / 20 = 32.11 / 20 = 1.6055
(44%+53%+42%+32%+37%+38%+47%+36%+47%+30%+25%+34%+28%+26%+29%+29%+29%+33%+25%+25%) /
20 = 689 / 20 = 34.45


Therefore:

* Average Execution Time(Based on ctime): 16328.61 ms
* Average Execution Time(Based on chrono): 50320.225 ms
* Average Memory Usage: 0 KB
* Average Real Time: 50.3475 s
* Average User Time: 14.7245 s
* Average Sys Time: 1.6055 s
* Average CPU Usage: 34.45 %
```

Take a look at this chart, in the first 20 experiments, the time duration has become slower in this way:



Test#	Time
1	25468.1
2	20764
3	32682
4	47560
5	45354.9

Test#	Time
6	39712.6
7	29361.3
8	35374
9	28392.7
10	44944.2
11	70660.9
12	49479.4
13	52613.5
14	54965.7
15	61944
16	70709.4
17	82900.9
18	56225.7
19	68766
20	88525.2

Note: I created this chart using the free and open-source software `LibreOffice`.

Firstly, what is observable from the chart is that over time, there is a linear growth in the time taken for I/O on the disk during shared access. We can conclude that as time passes, the disk writing time becomes slower and slower, unlike RAM, which follows a consistent trend.

Secondly, we can see that the disk has performed its job 50 times slower, but we expected it to be much slower. This is related to the optimizations made by the operating system. The operating system does not allow one process to monopolize I/O on the disk, preventing other processes from accessing it. Instead, it schedules them in turns on the processor and shares the shared-resources. For more information, you can study the concepts of "deadlock" and "mutual exclusion" in computing. We may discuss them in separate article later.

Thirdly, the optimization in the operating system is not the entire issue. Our HDD-eaters are not really standard. They are just four processes that write a limited and constant amount of data (which can be cached) in sequential sectors of the disk, repeating this movement. What if, instead of these HDD-eaters, we used real ones?

Alright, that's enough for now. I'll stop this experiment here because I don't want to damage my disk! But this story will remain unfinished:

- What other scenarios can we test to compare HDD and RAM in large datas?

The comparison we made was a one-dimensional comparison. It was solely about a bunch of data being written in the most optimal way on the HDD/RAM, and we could observe that it could be up to 80 times slower in different scenarios. However, this experimental setup and comparison do not cover all aspects of the situation. We need another dimension of comparison.

Stay tuned for the article as we move on to the second phase of comparison...

Phase Two: Comparing the Speed of RAM and Hard Disk for Small Data and Fast Sequential Allocations

In this phase, we measure and compare the time spent for computation that are typically uses RAM for storage, once on RAM and once on the hard disk (how?!). In simple terms, we compare the speed of a specific computation, which is usually performed on RAM, with the speed of the same computation under the condition that it uses the hard disk for storage instead of RAM!

Consider the following code:

```
#include <iostream>
#include <sys/resource.h>
#include <chrono>
using namespace std;

int main() {

    // Start execution time
    clock_t start_1 = clock();
```

```
    auto start_2 = chrono::high_resolution_clock::now();

    // Start memory usage
    struct rusage usage;
    getrusage(RUSAGE_SELF, &usage);
    long memory_usage_start = usage.ru_maxrss; // in kilobytes

    // Code ----------------------------------------------------------

    unsigned long long n = 94, t1 = 0, t2 = 1, t3 = 0;

    if(n == 1)
        cout << t1 << endl;
    else if(n == 2)
        cout << t2 << endl;
    else{
        unsigned long long counter = 3;
        while (counter <= n) {
            t3 = t1 + t2;
            t1 = t2;
            t2 = t3;
            ++counter;
        }
        cout << t3 << endl;
    }

    // ---------------------------------------------------------------

    // Stop measuring memory usage
    getrusage(RUSAGE_SELF, &usage);
    long memory_usage_end = usage.ru_maxrss; // in kilobytes

    // Stop measuring execution time
    clock_t end_1 = clock();
    auto end_2 = chrono::high_resolution_clock::now();
    double execution_time_1 = double(end_1 - start_1) / CLOCKS_PER_SEC;
    chrono::duration<double, milli> execution_time_2 = end_2 - start_2;

    // Printing result
    cout << "Execution Time (Based on ctime): " << execution_time_1 * 1000.0 << "
ms" << endl;
    cout << "Execution Time (Based on chrono): " << execution_time_2.count() << "
ms" << endl;
    cout << "Memory Usage: " << memory_usage_end - memory_usage_start << " KB" <<
endl;

    return 0;
}
```

Source code

This code returns the nth term of the well-known Fibonacci sequence. The Fibonacci sequence is a famous mathematical example that is often used to explain many programming concepts. It is defined as follows:

Each term is the sum of the two precvious terms, assuming the first term is and the second term is `1`:

```
0,1,1,2,3,5,8,13,21,34,55,89,…
```

This code can calculate Fibonacci terms up to the 94th term. However, beyond that, it overflows due to the size of `long long`, which is 8 bytes (64 bits). When it exceeds 2 to the power of 64, which is `18446744073709551616`, it overflows and produces incorrect results (read about big numbers. Hint: Big-Int).

In this code, we have stored all variables solely in RAM. Now let's see its execution time:

```
┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ g++ Fibo_RAM.cpp

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 0.066 ms
Execution Time (Based on chrono): 0.035751 ms
Memory Usage: 0 KB

real    0.00s
user    0.00s
sys     0.00s
cpu     87%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 0.127 ms
Execution Time (Based on chrono): 0.094721 ms
Memory Usage: 0 KB

real    0.01s
user    0.00s
sys     0.01s
cpu     88%
```

```
┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 0.122 ms
Execution Time (Based on chrono): 0.08808 ms
Memory Usage: 0 KB

real    0.01s
user    0.00s
sys     0.00s
cpu     88%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 0.05 ms
Execution Time (Based on chrono): 0.034632 ms
Memory Usage: 0 KB

real    0.00s
user    0.00s
sys     0.00s
cpu     89%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 0.15 ms
Execution Time (Based on chrono): 0.111781 ms
Memory Usage: 0 KB

real    0.01s
user    0.01s
sys     0.00s
cpu     88%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 0.14 ms
Execution Time (Based on chrono): 0.101224 ms
Memory Usage: 0 KB

real    0.01s
user    0.01s
sys     0.00s
cpu     88%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 0.15 ms
```

```
Execution Time (Based on chrono): 0.111455 ms
Memory Usage: 0 KB

real    0.01s
user    0.00s
sys     0.00s
cpu     89%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 0.048 ms
Execution Time (Based on chrono): 0.032223 ms
Memory Usage: 0 KB

real    0.00s
user    0.00s
sys     0.00s
cpu     86%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 0.053 ms
Execution Time (Based on chrono): 0.037364 ms
Memory Usage: 0 KB

real    0.00s
user    0.00s
sys     0.00s
cpu     88%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 0.046 ms
Execution Time (Based on chrono): 0.030818 ms
Memory Usage: 0 KB

real    0.00s
user    0.00s
sys     0.00s
cpu     87%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 0.125 ms
Execution Time (Based on chrono): 0.09257 ms
Memory Usage: 0 KB

real    0.01s
```

```
user    0.01s
sys     0.00s
cpu     87%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 0.12 ms
Execution Time (Based on chrono): 0.088397 ms
Memory Usage: 0 KB

real    0.01s
user    0.01s
sys     0.00s
cpu     88%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 0.121 ms
Execution Time (Based on chrono): 0.089428 ms
Memory Usage: 0 KB

real    0.01s
user    0.00s
sys     0.00s
cpu     87%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 0.159 ms
Execution Time (Based on chrono): 0.111137 ms
Memory Usage: 0 KB

real    0.01s
user    0.00s
sys     0.01s
cpu     88%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 0.148 ms
Execution Time (Based on chrono): 0.110892 ms
Memory Usage: 0 KB

real    0.01s
user    0.01s
sys     0.00s
cpu     88%
```

```
┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 0.053 ms
Execution Time (Based on chrono): 0.03688 ms
Memory Usage: 0 KB

real    0.00s
user    0.00s
sys     0.00s
cpu     89%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 0.129 ms
Execution Time (Based on chrono): 0.096686 ms
Memory Usage: 0 KB

real    0.01s
user    0.00s
sys     0.01s
cpu     88%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 0.044 ms
Execution Time (Based on chrono): 0.029967 ms
Memory Usage: 0 KB

real    0.00s
user    0.00s
sys     0.00s
cpu     86%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 0.045 ms
Execution Time (Based on chrono): 0.029736 ms
Memory Usage: 0 KB

real    0.00s
user    0.00s
sys     0.00s
cpu     86%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 0.128 ms
```

```
Execution Time (Based on chrono): 0.094682 ms
Memory Usage: 0 KB

real    0.01s
user    0.00s
sys     0.00s
cpu     89%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 0.121 ms
Execution Time (Based on chrono): 0.088579 ms
Memory Usage: 0 KB

real    0.01s
user    0.00s
sys     0.00s
cpu     86%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 0.103 ms
Execution Time (Based on chrono): 0.051734 ms
Memory Usage: 0 KB

real    0.00s
user    0.00s
sys     0.00s
cpu     89%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 0.047 ms
Execution Time (Based on chrono): 0.030919 ms
Memory Usage: 0 KB

real    0.00s
user    0.00s
sys     0.00s
cpu     88%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 0.145 ms
Execution Time (Based on chrono): 0.106697 ms
Memory Usage: 0 KB

real    0.01s
```

```
user    0.00s
sys     0.00s
cpu     89%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 0.051 ms
Execution Time (Based on chrono): 0.034342 ms
Memory Usage: 0 KB

real    0.00s
user    0.00s
sys     0.00s
cpu     88%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 0.148 ms
Execution Time (Based on chrono): 0.110057 ms
Memory Usage: 0 KB

real    0.01s
user    0.01s
sys     0.00s
cpu     89%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 0.126 ms
Execution Time (Based on chrono): 0.094865 ms
Memory Usage: 0 KB

real    0.01s
user    0.00s
sys     0.01s
cpu     88%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 0.051 ms
Execution Time (Based on chrono): 0.035199 ms
Memory Usage: 0 KB

real    0.00s
user    0.00s
sys     0.00s
cpu     89%
```

```
┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 0.14 ms
Execution Time (Based on chrono): 0.102388 ms
Memory Usage: 0 KB

real    0.01s
user    0.00s
sys     0.01s
cpu     89%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 0.045 ms
Execution Time (Based on chrono): 0.030435 ms
Memory Usage: 0 KB

real    0.00s
user    0.00s
sys     0.00s
cpu     85%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 0.048 ms
Execution Time (Based on chrono): 0.034097 ms
Memory Usage: 0 KB

real    0.00s
user    0.00s
sys     0.00s
cpu     85%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 0.124 ms
Execution Time (Based on chrono): 0.092128 ms
Memory Usage: 0 KB

real    0.01s
user    0.00s
sys     0.00s
cpu     87%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 0.046 ms
```

```
Execution Time (Based on chrono): 0.031791 ms
Memory Usage: 0 KB

real    0.00s
user    0.00s
sys     0.00s
cpu     88%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 0.045 ms
Execution Time (Based on chrono): 0.030651 ms
Memory Usage: 0 KB

real    0.00s
user    0.00s
sys     0.00s
cpu     86%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 0.046 ms
Execution Time (Based on chrono): 0.031053 ms
Memory Usage: 0 KB

real    0.00s
user    0.00s
sys     0.00s
cpu     86%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 0.06 ms
Execution Time (Based on chrono): 0.045026 ms
Memory Usage: 0 KB

real    0.00s
user    0.00s
sys     0.00s
cpu     87%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 0.047 ms
Execution Time (Based on chrono): 0.031791 ms
Memory Usage: 0 KB

real    0.00s
```

```
user    0.00s
sys     0.00s
cpu     88%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 0.129 ms
Execution Time (Based on chrono): 0.095468 ms
Memory Usage: 0 KB

real    0.01s
user    0.00s
sys     0.00s
cpu     89%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 0.122 ms
Execution Time (Based on chrono): 0.087091 ms
Memory Usage: 0 KB

real    0.01s
user    0.00s
sys     0.00s
cpu     89%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 0.044 ms
Execution Time (Based on chrono): 0.029332 ms
Memory Usage: 0 KB

real    0.00s
user    0.00s
sys     0.00s
cpu     86%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 0.13 ms
Execution Time (Based on chrono): 0.096376 ms
Memory Usage: 0 KB

real    0.01s
user    0.00s
sys     0.00s
cpu     88%
```

```
┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 0.125 ms
Execution Time (Based on chrono): 0.091047 ms
Memory Usage: 0 KB

real    0.01s
user    0.00s
sys     0.00s
cpu     89%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 0.124 ms
Execution Time (Based on chrono): 0.090338 ms
Memory Usage: 0 KB

real    0.01s
user    0.00s
sys     0.00s
cpu     87%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 0.124 ms
Execution Time (Based on chrono): 0.090225 ms
Memory Usage: 0 KB

real    0.01s
user    0.00s
sys     0.00s
cpu     88%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 0.121 ms
Execution Time (Based on chrono): 0.0879 ms
Memory Usage: 0 KB

real    0.01s
user    0.00s
sys     0.00s
cpu     88%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 0.124 ms
```

```
Execution Time (Based on chrono): 0.091002 ms
Memory Usage: 0 KB

real    0.01s
user    0.00s
sys     0.00s
cpu     89%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 0.127 ms
Execution Time (Based on chrono): 0.093424 ms
Memory Usage: 0 KB

real    0.01s
user    0.01s
sys     0.00s
cpu     87%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 0.123 ms
Execution Time (Based on chrono): 0.090351 ms
Memory Usage: 0 KB

real    0.01s
user    0.00s
sys     0.01s
cpu     88%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 0.136 ms
Execution Time (Based on chrono): 0.103215 ms
Memory Usage: 0 KB

real    0.01s
user    0.00s
sys     0.00s
cpu     88%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 0.118 ms
Execution Time (Based on chrono): 0.085639 ms
Memory Usage: 0 KB

real    0.01s
```

```
user    0.00s
sys     0.00s
cpu     88%
```

Because the reported time intervals were bloody small, I increased the number of executions for different time intervals (50 times) to have a more accurate estimate of the average time:

```
(0.066+0.127+0.122+0.05+0.15+0.14+0.15+0.048+0.053+0.046+0.125+0.12+0.121+0.159+0.14
8+0.053+0.129+0.044+0.045+0.128+0.121+0.103+0.047+0.145+0.051+0.148+0.126+0.051+0.14
+0.045+0.048+0.124+0.046+0.045+0.046+0.06+0.047+0.129+0.122+0.044+0.13+0.125+0.124+0
.124+0.121+0.124+0.127+0.123+0.136+0.118) / 50 = 4.964 / 50 = 0.09928
(0.035751+0.094721+0.08808+0.034632+0.111781+0.101224+0.111455+0.032223+0.037364+0.0
30818+0.09257+0.088397+0.089428+0.111137+0.110892+0.03688+0.096686+0.029967+0.029736
+0.094682+0.088579+0.051734+0.030919+0.106697+0.034342+0.110057+0.094865+0.035199+0.
102388+0.030435+0.034097+0.092128+0.031791+0.030651+0.031053+0.045026+0.031791+0.095
468+0.087091+0.029332+0.096376+0.091047+0.090338+0.090225+0.0879+0.091002+0.093424+0
.090351+0.103215+0.085639) / 50 = 3.571584 / 50 = 0.07143168
(0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0
+0+0+0+0+0+0+0+0) / 50 = 0 / 50 = 0
(0.00s+0.01s+0.01s+0.00s+0.01s+0.01s+0.01s+0.00s+0.00s+0.00s+0.01s+0.01s+0.01s+0.01s
+0.01s+0.00s+0.01s+0.00s+0.00s+0.01s+0.01s+0.00s+0.00s+0.01s+0.00s+0.01s+0.01s+0.00s
+0.01s+0.00s+0.00s+0.01s+0.00s+0.00s+0.00s+0.00s+0.00s+0.01s+0.01s+0.00s+0.01s+0.01s
+0.01s+0.01s+0.01s+0.01s+0.01s+0.01s+0.01s+0.01s) / 50 = 0.3 / 50 = 0.006
(0.00s+0.00s+0.00s+0.00s+0.01s+0.01s+0.00s+0.00s+0.00s+0.00s+0.01s+0.01s+0.00s+0.00s
+0.01s+0.00s+0.00s+0.00s+0.00s+0.00s+0.00s+0.00s+0.00s+0.00s+0.01s+0.00s+0.00s+0.00s
+0.00s+0.00s+0.00s+0.00s+0.00s+0.00s+0.00s+0.00s+0.00s+0.00s+0.00s+0.00s+0.00s+0.00s
+0.00s+0.00s+0.00s+0.00s+0.01s+0.00s+0.00s+0.00s) / 50 = 0.07 / 50 = 0.0014
(0.00s+0.01s+0.00s+0.00s+0.00s+0.00s+0.00s+0.00s+0.00s+0.00s+0.00s+0.00s+0.00s+0.01s
+0.00s+0.00s+0.01s+0.00s+0.00s+0.00s+0.00s+0.00s+0.00s+0.00s+0.00s+0.00s+0.01s+0.00s
+0.01s+0.00s+0.00s+0.00s+0.00s+0.00s+0.00s+0.00s+0.00s+0.00s+0.00s+0.00s+0.00s+0.00s
+0.00s+0.00s+0.00s+0.00s+0.00s+0.01s+0.00s+0.00s) / 50 = 0.06 / 50 = 0.0012
(87%+88%+88%+89%+88%+88%+89%+86%+88%+87%+87%+88%+87%+88%+88%+89%+88%+86%+86%+89%+86%
+89%+88%+89%+88%+89%+88%+89%+89%+85%+85%+87%+88%+86%+86%+87%+88%+89%+89%+86%+88%+89%
+87%+88%+88%+89%+87%+88%+88%+88%) / 50 = 4385 / 50 = 87.7


Therefore:

* Average Execution Time(Based on ctime): 0.09928 ms
* Average Execution Time(Based on chrono): 0.07143168 ms
* Average Memory Usage: 0 KB
* Average Real Time: 0.006 s
* Average User Time: 0.0014 s
* Average Sys Time: 0.0012 s
* Average CPU Usage: 87.7 %
```

Now let's write the same program in a way that stores the variables it uses on the disk instead of RAM!

```
#include <iostream>
#include <sys/resource.h>
#include <chrono>
#include <fstream>
#include <string>
using namespace std;

void set_unsigned_long_long_variable(string name, unsigned long long var)
{
    ofstream outfile(name + ".bin"  , ofstream::binary);
    if (outfile)
    {
        outfile.write(reinterpret_cast<const char*>(&var), sizeof(var));
        outfile.close();
    }
    else cerr << "Error opening file: " << name + ".bin" << endl;
}

unsigned long long get_unsigned_long_long_variable(string name)
{
    unsigned long long var = 0;
    ifstream infile(name + ".bin", ifstream::binary);
    if (infile)
    {
        infile.read(reinterpret_cast<char*>(&var), sizeof(var));
        infile.close();
    }
    else cerr << "Error opening file: " << name + ".bin" << endl;

    return var;
}

int main() {

    // Start execution time
    clock_t start_1 = clock();
    auto start_2 = chrono::high_resolution_clock::now();

    // Start memory usage
    struct rusage usage;
    getrusage(RUSAGE_SELF, &usage);
    long memory_usage_start = usage.ru_maxrss; // in kilobytes

    // Code ----------------------------------------------------------

    set_unsigned_long_long_variable( "n", 94 );
    set_unsigned_long_long_variable( "t1", 0 );
    set_unsigned_long_long_variable( "t2", 1 );
    set_unsigned_long_long_variable( "t3", 0 );
```

```
    if(get_unsigned_long_long_variable("n") == 1)
        cout << get_unsigned_long_long_variable("t1") << endl;
    else if(get_unsigned_long_long_variable("n") == 2)
        cout << get_unsigned_long_long_variable("t2") << endl;
    else{
        set_unsigned_long_long_variable( "counter", 3 );
        while (get_unsigned_long_long_variable("counter") <=
get_unsigned_long_long_variable("n")) {
            set_unsigned_long_long_variable ( "t3",
get_unsigned_long_long_variable("t1") + get_unsigned_long_long_variable("t2"));
            set_unsigned_long_long_variable ("t1" ,
get_unsigned_long_long_variable("t2"));
            set_unsigned_long_long_variable ("t2" ,
get_unsigned_long_long_variable("t3"));
            set_unsigned_long_long_variable ("counter" ,
get_unsigned_long_long_variable("counter") + 1);
        }
        cout << get_unsigned_long_long_variable("t3") << endl;
    }

    // ------------------------------------------------------------

    // Stop measuring memory usage
    getrusage(RUSAGE_SELF, &usage);
    long memory_usage_end = usage.ru_maxrss; // in kilobytes

    // Stop measuring execution time
    clock_t end_1 = clock();
    auto end_2 = chrono::high_resolution_clock::now();
    double execution_time_1 = double(end_1 - start_1) / CLOCKS_PER_SEC;
    chrono::duration<double, milli> execution_time_2 = end_2 - start_2;

    // Printing result
    cout << "Execution Time (Based on ctime): " << execution_time_1 * 1000.0 << "
ms" << endl;
    cout << "Execution Time (Based on chrono): " << execution_time_2.count() << "
ms" << endl;
    cout << "Memory Usage: " << memory_usage_end - memory_usage_start << " KB" <<
endl;

    return 0;
}
```

Source code

This code is equivalent to the previous program, only difference is that it stores its variables on the disk. I will run it 50 times to see what happens:

```
┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ g++ Fibo_HDD.cpp

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 27.961 ms
Execution Time (Based on chrono): 44.9589 ms
Memory Usage: 0 KB

real    0.05s
user    0.00s
sys     0.03s
cpu     63%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 31.508 ms
Execution Time (Based on chrono): 31.4675 ms
Memory Usage: 0 KB

real    0.04s
user    0.00s
sys     0.03s
cpu     98%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 33.407 ms
Execution Time (Based on chrono): 173.327 ms
Memory Usage: 0 KB

real    0.18s
user    0.01s
sys     0.03s
cpu     21%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 37.039 ms
Execution Time (Based on chrono): 60.1359 ms
Memory Usage: 0 KB

real    0.07s
user    0.00s
sys     0.04s
cpu     63%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
```

```
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 26.895 ms
Execution Time (Based on chrono): 53.0744 ms
Memory Usage: 0 KB

real    0.06s
user    0.00s
sys     0.03s
cpu     53%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 30.281 ms
Execution Time (Based on chrono): 30.9933 ms
Memory Usage: 0 KB

real    0.03s
user    0.00s
sys     0.03s
cpu     97%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 26.92 ms
Execution Time (Based on chrono): 27.194 ms
Memory Usage: 0 KB

real    0.03s
user    0.00s
sys     0.03s
cpu     97%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 28.805 ms
Execution Time (Based on chrono): 45.9833 ms
Memory Usage: 0 KB

real    0.05s
user    0.00s
sys     0.03s
cpu     63%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 35.667 ms
Execution Time (Based on chrono): 94.1039 ms
```

```
Memory Usage: 0 KB

real    0.10s
user    0.01s
sys     0.03s
cpu     40%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 28.217 ms
Execution Time (Based on chrono): 40.1542 ms
Memory Usage: 0 KB

real    0.04s
user    0.00s
sys     0.03s
cpu     71%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 52.593 ms
Execution Time (Based on chrono): 68.8812 ms
Memory Usage: 0 KB

real    0.07s
user    0.01s
sys     0.05s
cpu     77%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 74.053 ms
Execution Time (Based on chrono): 335.553 ms
Memory Usage: 0 KB

real    0.34s
user    0.02s
sys     0.06s
cpu     23%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 29.317 ms
Execution Time (Based on chrono): 42.8699 ms
Memory Usage: 0 KB

real    0.05s
user    0.01s
```

```
sys     0.02s
cpu     70%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 50.748 ms
Execution Time (Based on chrono): 83.1821 ms
Memory Usage: 0 KB

real    0.09s
user    0.01s
sys     0.05s
cpu     62%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 50.086 ms
Execution Time (Based on chrono): 59.5969 ms
Memory Usage: 0 KB

real    0.06s
user    0.00s
sys     0.05s
cpu     84%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 26.691 ms
Execution Time (Based on chrono): 66.4044 ms
Memory Usage: 0 KB

real    0.07s
user    0.01s
sys     0.02s
cpu     43%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 27.622 ms
Execution Time (Based on chrono): 42.9472 ms
Memory Usage: 0 KB

real    0.05s
user    0.00s
sys     0.03s
cpu     66%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
```

```
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 26.011 ms
Execution Time (Based on chrono): 42.8584 ms
Memory Usage: 0 KB

real    0.05s
user    0.00s
sys     0.03s
cpu     63%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 46.855 ms
Execution Time (Based on chrono): 59.3697 ms
Memory Usage: 0 KB

real    0.06s
user    0.01s
sys     0.04s
cpu     79%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 43.295 ms
Execution Time (Based on chrono): 55.3742 ms
Memory Usage: 0 KB

real    0.06s
user    0.00s
sys     0.04s
cpu     79%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 56.283 ms
Execution Time (Based on chrono): 56.3353 ms
Memory Usage: 0 KB

real    0.06s
user    0.02s
sys     0.04s
cpu     98%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 55.732 ms
Execution Time (Based on chrono): 153.311 ms
```

```
Memory Usage: 0 KB

real    0.16s
user    0.02s
sys     0.04s
cpu     37%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 36.473 ms
Execution Time (Based on chrono): 90.0586 ms
Memory Usage: 0 KB

real    0.09s
user    0.01s
sys     0.03s
cpu     42%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 35.634 ms
Execution Time (Based on chrono): 60.2903 ms
Memory Usage: 0 KB

real    0.07s
user    0.01s
sys     0.03s
cpu     61%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 37.919 ms
Execution Time (Based on chrono): 41.4295 ms
Memory Usage: 0 KB

real    0.05s
user    0.00s
sys     0.04s
cpu     90%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 53.446 ms
Execution Time (Based on chrono): 55.0082 ms
Memory Usage: 0 KB

real    0.06s
user    0.02s
```

```
sys     0.04s
cpu     96%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 57.878 ms
Execution Time (Based on chrono): 215.491 ms
Memory Usage: 0 KB

real    0.22s
user    0.02s
sys     0.04s
cpu     28%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 51.583 ms
Execution Time (Based on chrono): 79.8935 ms
Memory Usage: 0 KB

real    0.08s
user    0.02s
sys     0.04s
cpu     65%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 66.857 ms
Execution Time (Based on chrono): 318.953 ms
Memory Usage: 0 KB

real    0.32s
user    0.01s
sys     0.06s
cpu     22%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 44.318 ms
Execution Time (Based on chrono): 44.4082 ms
Memory Usage: 0 KB

real    0.05s
user    0.01s
sys     0.04s
cpu     98%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
```

```
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 68.406 ms
Execution Time (Based on chrono): 391.37 ms
Memory Usage: 0 KB

real    0.40s
user    0.00s
sys     0.07s
cpu     18%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 64.93 ms
Execution Time (Based on chrono): 266.554 ms
Memory Usage: 0 KB

real    0.27s
user    0.01s
sys     0.06s
cpu     25%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 53.075 ms
Execution Time (Based on chrono): 95.7015 ms
Memory Usage: 0 KB

real    0.10s
user    0.01s
sys     0.05s
cpu     57%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 48.386 ms
Execution Time (Based on chrono): 51.8912 ms
Memory Usage: 0 KB

real    0.06s
user    0.00s
sys     0.05s
cpu     92%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 64.722 ms
Execution Time (Based on chrono): 282.728 ms
```

```
Memory Usage: 0 KB

real    0.29s
user    0.00s
sys     0.06s
cpu     24%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 30.692 ms
Execution Time (Based on chrono): 51.0716 ms
Memory Usage: 0 KB

real    0.05s
user    0.01s
sys     0.03s
cpu     61%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 42.179 ms
Execution Time (Based on chrono): 71.7145 ms
Memory Usage: 0 KB

real    0.08s
user    0.00s
sys     0.04s
cpu     60%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 27.902 ms
Execution Time (Based on chrono): 49.6614 ms
Memory Usage: 0 KB

real    0.05s
user    0.01s
sys     0.02s
cpu     57%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 50.868 ms
Execution Time (Based on chrono): 65.8676 ms
Memory Usage: 0 KB

real    0.07s
user    0.00s
```

```
sys     0.06s
cpu     77%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 38.906 ms
Execution Time (Based on chrono): 229.074 ms
Memory Usage: 0 KB

real    0.23s
user    0.01s
sys     0.04s
cpu     18%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 45.282 ms
Execution Time (Based on chrono): 95.5347 ms
Memory Usage: 0 KB

real    0.10s
user    0.01s
sys     0.04s
cpu     49%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 32.539 ms
Execution Time (Based on chrono): 37.7129 ms
Memory Usage: 0 KB

real    0.04s
user    0.00s
sys     0.03s
cpu     86%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 47.754 ms
Execution Time (Based on chrono): 60.9285 ms
Memory Usage: 0 KB

real    0.07s
user    0.01s
sys     0.04s
cpu     79%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
```

```
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 34.381 ms
Execution Time (Based on chrono): 41.1787 ms
Memory Usage: 0 KB

real    0.05s
user    0.00s
sys     0.04s
cpu     84%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 52.619 ms
Execution Time (Based on chrono): 291.559 ms
Memory Usage: 0 KB

real    0.30s
user    0.01s
sys     0.05s
cpu     19%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 24.712 ms
Execution Time (Based on chrono): 85.2908 ms
Memory Usage: 0 KB

real    0.09s
user    0.00s
sys     0.02s
cpu     30%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 54.553 ms
Execution Time (Based on chrono): 72.2719 ms
Memory Usage: 0 KB

real    0.08s
user    0.02s
sys     0.04s
cpu     76%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 61.797 ms
Execution Time (Based on chrono): 396.595 ms
```

```
Memory Usage: 0 KB

real    0.40s
user    0.01s
sys     0.06s
cpu     16%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 24.581 ms
Execution Time (Based on chrono): 44.3478 ms
Memory Usage: 0 KB

real    0.05s
user    0.00s
sys     0.03s
cpu     58%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
12200160415121876738
Execution Time (Based on ctime): 41.316 ms
Execution Time (Based on chrono): 74.7477 ms
Memory Usage: 0 KB

real    0.08s
user    0.00s
sys     0.04s
cpu     57%
```

The average:

```
(27.961+31.508+33.407+37.039+26.895+30.281+26.92+28.805+35.667+28.217+52.593+74.053+
29.317+50.748+50.086+26.691+27.622+26.011+46.855+43.295+56.283+55.732+36.473+35.634+
37.919+53.446+57.878+51.583+66.857+44.318+68.406+64.93+53.075+48.386+64.722+30.692+4
2.179+27.902+50.868+38.906+45.282+32.539+47.754+34.381+52.619+24.712+54.553+61.797+2
4.581+41.316) / 50 = 2139.694 / 50 = 42.79388
(44.9589+31.4675+173.327+60.1359+53.0744+30.9933+27.194+45.9833+94.1039+40.1542+68.8
812+335.553+42.8699+83.1821+59.5969+66.4044+42.9472+42.8584+59.3697+55.3742+56.3353+
153.311+90.0586+60.2903+41.4295+55.0082+215.491+79.8935+318.953+44.4082+391.37+266.5
54+95.7015+51.8912+282.728+51.0716+71.7145+49.6614+65.8676+229.074+95.5347+37.7129+6
0.9285+41.1787+291.559+85.2908+72.2719+396.595+44.3478+74.7477) / 50 = 5329.4088 /
50 = 106.588176
(0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0
+0+0+0+0+0+0+0+0+0) / 50 = 0 / 50 = 0
(0.05s+0.04s+0.18s+0.07s+0.06s+0.03s+0.03s+0.05s+0.10s+0.04s+0.07s+0.34s+0.05s+0.09s
+0.06s+0.07s+0.05s+0.05s+0.06s+0.06s+0.06s+0.16s+0.09s+0.07s+0.05s+0.06s+0.22s+0.08s
+0.32s+0.05s+0.40s+0.27s+0.10s+0.06s+0.29s+0.05s+0.08s+0.05s+0.07s+0.23s+0.10s+0.04s
+0.07s+0.05s+0.30s+0.09s+0.08s+0.40s+0.05s+0.08s) / 50 = 5.57 / 50 = 0.1114
```

```
(0.00s+0.00s+0.01s+0.00s+0.00s+0.00s+0.00s+0.00s+0.01s+0.00s+0.01s+0.02s+0.01s+0.01s
+0.00s+0.01s+0.00s+0.00s+0.01s+0.00s+0.02s+0.02s+0.01s+0.01s+0.00s+0.02s+0.02s+0.02s
+0.01s+0.01s+0.00s+0.01s+0.01s+0.00s+0.00s+0.01s+0.00s+0.01s+0.00s+0.01s+0.01s+0.00s
+0.01s+0.00s+0.01s+0.00s+0.02s+0.01s+0.00s+0.00s) / 50 = 0.34 / 50 = 0.0068
(0.03s+0.03s+0.03s+0.04s+0.03s+0.03s+0.03s+0.03s+0.03s+0.03s+0.05s+0.06s+0.02s+0.05s
+0.05s+0.02s+0.03s+0.03s+0.04s+0.04s+0.04s+0.04s+0.03s+0.03s+0.04s+0.04s+0.04s+0.04s
+0.06s+0.04s+0.07s+0.06s+0.05s+0.05s+0.06s+0.03s+0.04s+0.02s+0.06s+0.04s+0.04s+0.03s
+0.04s+0.04s+0.05s+0.02s+0.04s+0.06s+0.03s+0.04s) / 50 = 1.97 / 50 = 0.0394
(63%+98%+21%+63%+53%+97%+97%+63%+40%+71%+77%+23%+70%+62%+84%+43%+66%+63%+79%+79%+98%
+37%+42%+61%+90%+96%+28%+65%+22%+98%+18%+25%+57%+92%+24%+61%+60%+57%+77%+18%+49%+86%
+79%+84%+19%+30%+76%+16%+58%+57%) / 50 = 2992 / 50 = 59.84


Therefore:

* Average Execution Time(Based on ctime): 42.79388 ms
* Average Execution Time(Based on chrono): 106.588176 ms
* Average Memory Usage: 0 KB
* Average Real Time: 0.1114 s
* Average User Time: 0.0068 s
* Average Sys Time: 0.0394 s
* Average CPU Usage: 59.84 %
```

And comparing results:

```
106.588176 / 0.07143168 = 1492
```

Wow! This time the disk appeared to be about 1500 times slower...

I'm thinking that our comparison scenario is quite simple ×□× We only got 94 iterations, which is very small... Let's write another code and increase the number of iterations to make it closer to real programs!

Consider the following code:

```
#include <iostream>
#include <sys/resource.h>
#include <chrono>
using namespace std;
```

```
int main() {

    // Start execution time
    clock_t start_1 = clock();
    auto start_2 = chrono::high_resolution_clock::now();

    // Start memory usage
    struct rusage usage;
    getrusage(RUSAGE_SELF, &usage);
    long memory_usage_start = usage.ru_maxrss; // in kilobytes

    // Code ----------------------------------------------------------

    unsigned long long n = 1000, counter = n, sum_divable = 0;

    while (counter > 0) {
        if( n % counter == 0 )
            ++sum_divable;
        --counter;
    }
    cout << sum_divable << endl;

    // ---------------------------------------------------------------

    // Stop measuring memory usage
    getrusage(RUSAGE_SELF, &usage);
    long memory_usage_end = usage.ru_maxrss; // in kilobytes

    // Stop measuring execution time
    clock_t end_1 = clock();
    auto end_2 = chrono::high_resolution_clock::now();
    double execution_time_1 = double(end_1 - start_1) / CLOCKS_PER_SEC;
    chrono::duration<double, milli> execution_time_2 = end_2 - start_2;

    // Printing result
    cout << "Execution Time (Based on ctime): " << execution_time_1 * 1000.0 << "
ms" << endl;
    cout << "Execution Time (Based on chrono): " << execution_time_2.count() << "
ms" << endl;
    cout << "Memory Usage: " << memory_usage_end - memory_usage_start << " KB" <<
endl;

    return 0;
}
```

Source code

This code, for each number n, iterates through all its previous numbers and prints the total

number of its divisors. It is clear that for prime numbers, it prints only 2 (n and 1). It is a simple code, and it can be written more efficiently, but we don't want to do that... The purpose of this code is to perform iterational calculations so that we can measure the speed.

```
┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ g++ Iteration_RAM.cpp

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
36
Execution Time (Based on ctime): 1.288 ms
Execution Time (Based on chrono): 1.27333 ms
Memory Usage: 0 KB

real    0.00s
user    0.00s
sys     0.00s
cpu     93%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
36
Execution Time (Based on ctime): 1.196 ms
Execution Time (Based on chrono): 1.18127 ms
Memory Usage: 0 KB

real    0.00s
user    0.00s
sys     0.00s
cpu     92%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
36
Execution Time (Based on ctime): 1.101 ms
Execution Time (Based on chrono): 1.08655 ms
Memory Usage: 0 KB

real    0.00s
user    0.00s
sys     0.00s
cpu     93%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
36
Execution Time (Based on ctime): 2 ms
Execution Time (Based on chrono): 1.96631 ms
Memory Usage: 0 KB

real    0.01s
```

```
user    0.00s
sys     0.01s
cpu     92%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
36
Execution Time (Based on ctime): 2.147 ms
Execution Time (Based on chrono): 2.11425 ms
Memory Usage: 0 KB

real    0.01s
user    0.00s
sys     0.00s
cpu     92%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
36
Execution Time (Based on ctime): 1.894 ms
Execution Time (Based on chrono): 1.86002 ms
Memory Usage: 0 KB

real    0.01s
user    0.00s
sys     0.00s
cpu     92%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
36
Execution Time (Based on ctime): 2.371 ms
Execution Time (Based on chrono): 2.33654 ms
Memory Usage: 0 KB

real    0.01s
user    0.01s
sys     0.00s
cpu     92%

┌──(user□dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
36
Execution Time (Based on ctime): 2.297 ms
Execution Time (Based on chrono): 2.26285 ms
Memory Usage: 0 KB

real    0.01s
user    0.00s
sys     0.00s
cpu     92%
```

```
┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
36
Execution Time (Based on ctime): 1.832 ms
Execution Time (Based on chrono): 1.79954 ms
Memory Usage: 0 KB

real    0.01s
user    0.00s
sys     0.00s
cpu     91%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
36
Execution Time (Based on ctime): 2.074 ms
Execution Time (Based on chrono): 2.03888 ms
Memory Usage: 0 KB

real    0.01s
user    0.00s
sys     0.00s
cpu     92%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
36
Execution Time (Based on ctime): 2.393 ms
Execution Time (Based on chrono): 2.36894 ms
Memory Usage: 0 KB

real    0.01s
user    0.00s
sys     0.00s
cpu     93%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
36
Execution Time (Based on ctime): 2.423 ms
Execution Time (Based on chrono): 2.38987 ms
Memory Usage: 0 KB

real    0.01s
user    0.01s
sys     0.00s
cpu     92%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
36
Execution Time (Based on ctime): 2.164 ms
```

```
Execution Time (Based on chrono): 2.13099 ms
Memory Usage: 0 KB

real    0.01s
user    0.01s
sys     0.00s
cpu     92%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
36
Execution Time (Based on ctime): 2.153 ms
Execution Time (Based on chrono): 2.12008 ms
Memory Usage: 0 KB

real    0.01s
user    0.01s
sys     0.00s
cpu     92%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
36
Execution Time (Based on ctime): 2.221 ms
Execution Time (Based on chrono): 2.18724 ms
Memory Usage: 0 KB

real    0.01s
user    0.01s
sys     0.00s
cpu     92%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
36
Execution Time (Based on ctime): 1.274 ms
Execution Time (Based on chrono): 1.25876 ms
Memory Usage: 0 KB

real    0.00s
user    0.00s
sys     0.00s
cpu     92%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
36
Execution Time (Based on ctime): 2.239 ms
Execution Time (Based on chrono): 2.20461 ms
Memory Usage: 0 KB

real    0.01s
```

```
user    0.00s
sys     0.00s
cpu     92%

┌──(user dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
36
Execution Time (Based on ctime): 1.031 ms
Execution Time (Based on chrono): 1.01659 ms
Memory Usage: 0 KB

real    0.00s
user    0.00s
sys     0.00s
cpu     92%

┌──(user dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
36
Execution Time (Based on ctime): 2.485 ms
Execution Time (Based on chrono): 2.45185 ms
Memory Usage: 0 KB

real    0.01s
user    0.00s
sys     0.00s
cpu     92%

┌──(user dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
36
Execution Time (Based on ctime): 1.276 ms
Execution Time (Based on chrono): 1.26014 ms
Memory Usage: 0 KB

real    0.00s
user    0.00s
sys     0.00s
cpu     92%
```

And the average:

```
(1.288+1.196+1.101+2+2.147+1.894+2.371+2.297+1.832+2.074+2.393+2.423+2.164+2.153+2.2
21+1.274+2.239+1.031+2.485+1.276) / 20 = 37.859 / 20 = 1.89295
(1.27333+1.18127+1.08655+1.96631+2.11425+1.86002+2.33654+2.26285+1.79954+2.03888+2.3
6894+2.38987+2.13099+2.12008+2.18724+1.25876+2.20461+1.01659+2.45185+1.26014) / 20 =
37.30861 / 20 = 1.8654305
(0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0) / 20 = 0 / 20 = 0
(0.00s+0.00s+0.00s+0.01s+0.01s+0.01s+0.01s+0.01s+0.01s+0.01s+0.01s+0.01s+0.01s+0.01s
+0.01s+0.00s+0.01s+0.00s+0.01s+0.00s) / 20 = 0.14 / 20 = 0.007
```

```
(0.00s+0.00s+0.00s+0.00s+0.00s+0.00s+0.01s+0.00s+0.00s+0.00s+0.00s+0.01s+0.01s+0.01s
+0.01s+0.00s+0.00s+0.00s+0.00s+0.00s) / 20 = 0.05 / 20 = 0.0025
(0.00s+0.00s+0.00s+0.01s+0.00s+0.00s+0.00s+0.00s+0.00s+0.00s+0.00s+0.00s+0.00s+0.00s
+0.00s+0.00s+0.00s+0.00s+0.00s+0.00s) / 20 = 0.01 / 20 = 0.0005
(93%+92%+93%+92%+92%+92%+92%+92%+91%+92%+93%+92%+92%+92%+92%+92%+92%+92%+92%+92%) /
20 = 1842 / 20 = 92.1


Therefore:

* Average Execution Time(Based on ctime): 1.89295 ms
* Average Execution Time(Based on chrono): 1.8654305 ms
* Average Memory Usage: 0 KB
* Average Real Time: 0.007 s
* Average User Time: 0.0025 s
* Average Sys Time: 0.0005 s
* Average CPU Usage: 92.1 %
```

Let's suppose we write this code just like the previous section, but in a way that only stores its main variables on the disk, let's say only 3 main variables, nothing more. No addresses, no registers, no function calls, nothing else... Just store the values of the 3 main variables on the hard disk.

```
#include <iostream>
#include <sys/resource.h>
#include <chrono>
#include <fstream>
#include <string>
using namespace std;

void set_unsigned_long_long_variable(string name, unsigned long long var)
{
    ofstream outfile(name + ".bin"  , ofstream::binary);
    if (outfile)
    {
        outfile.write(reinterpret_cast<const char*>(&var), sizeof(var));
        outfile.close();
    }
    else cerr << "Error opening file: " << name + ".bin" << endl;
}

unsigned long long get_unsigned_long_long_variable(string name)
{
    unsigned long long var = 0;
    ifstream infile(name + ".bin", ifstream::binary);
    if (infile)
    {
        infile.read(reinterpret_cast<char*>(&var), sizeof(var));
        infile.close();
```

```
    }
    else cerr << "Error opening file: " << name + ".bin" << endl;

    return var;
}

int main() {

    // Start execution time
    clock_t start_1 = clock();
    auto start_2 = chrono::high_resolution_clock::now();

    // Start memory usage
    struct rusage usage;
    getrusage(RUSAGE_SELF, &usage);
    long memory_usage_start = usage.ru_maxrss; // in kilobytes

    // Code ------------------------------------------------------------

    set_unsigned_long_long_variable( "n", 100000);
    set_unsigned_long_long_variable( "counter",
get_unsigned_long_long_variable("n"));
    set_unsigned_long_long_variable( "sum_divable", 0);

    while (get_unsigned_long_long_variable("counter") > 0) {
        if( get_unsigned_long_long_variable("n") %
get_unsigned_long_long_variable("counter") == 0 )
            set_unsigned_long_long_variable( "sum_divable",
get_unsigned_long_long_variable("sum_divable") + 1);
        set_unsigned_long_long_variable( "counter",
get_unsigned_long_long_variable("counter") - 1);
    }
    cout << get_unsigned_long_long_variable("sum_divable") << endl;

    // -----------------------------------------------------------------

    // Stop measuring memory usage
    getrusage(RUSAGE_SELF, &usage);
    long memory_usage_end = usage.ru_maxrss; // in kilobytes

    // Stop measuring execution time
    clock_t end_1 = clock();
    auto end_2 = chrono::high_resolution_clock::now();
    double execution_time_1 = double(end_1 - start_1) / CLOCKS_PER_SEC;
    chrono::duration<double, milli> execution_time_2 = end_2 - start_2;

    // Printing result
    cout << "Execution Time (Based on ctime): " << execution_time_1 * 1000.0 << "
ms" << endl;
    cout << "Execution Time (Based on chrono): " << execution_time_2.count() << "
ms" << endl;
    cout << "Memory Usage: " << memory_usage_end - memory_usage_start << " KB" <<
```

```
endl;

    return 0;
}
```

Source code

Let's go ahead and run it to see what disaster occurs!

```
┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ g++ Iteration_HDD.cpp

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
36
Execution Time (Based on ctime): 8134.59 ms
Execution Time (Based on chrono): 22359.2 ms
Memory Usage: 0 KB

real    22.36s
user    1.38s
sys     6.75s
cpu     36%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
36
Execution Time (Based on ctime): 8427.09 ms
Execution Time (Based on chrono): 22312.2 ms
Memory Usage: 0 KB

real    22.31s
user    1.49s
sys     6.94s
cpu     37%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
36
Execution Time (Based on ctime): 8167.64 ms
Execution Time (Based on chrono): 22296.8 ms
Memory Usage: 0 KB

real    22.30s
user    1.45s
sys     6.72s
cpu     36%
```

```
┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
36
Execution Time (Based on ctime): 8040.17 ms
Execution Time (Based on chrono): 22315.6 ms
Memory Usage: 0 KB

real    22.32s
user    1.43s
sys     6.61s
cpu     36%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
36
Execution Time (Based on ctime): 8921.16 ms
Execution Time (Based on chrono): 30141.7 ms
Memory Usage: 0 KB

real    30.14s
user    1.43s
sys     7.49s
cpu     29%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
36
Execution Time (Based on ctime): 7915.38 ms
Execution Time (Based on chrono): 22112.7 ms
Memory Usage: 0 KB

real    22.12s
user    1.35s
sys     6.57s
cpu     35%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
36
Execution Time (Based on ctime): 8791.03 ms
Execution Time (Based on chrono): 26041.5 ms
Memory Usage: 0 KB

real    26.04s
user    1.41s
sys     7.38s
cpu     33%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
36
Execution Time (Based on ctime): 9897.57 ms
```

```
Execution Time (Based on chrono): 29009.1 ms
Memory Usage: 0 KB

real    29.01s
user    1.70s
sys     8.20s
cpu     34%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
36
Execution Time (Based on ctime): 8693.6 ms
Execution Time (Based on chrono): 23570.2 ms
Memory Usage: 0 KB

real    23.57s
user    1.43s
sys     7.27s
cpu     36%

┌──(user㉿dhcppc4)-[~/Desktop/Articles/4-RAM-vs-HDD-Pr/Files]
└─$ time ./a.out
36
Execution Time (Based on ctime): 9362.56 ms
Execution Time (Based on chrono): 28498.6 ms
Memory Usage: 0 KB

real    28.50s
user    1.74s
sys     7.62s
cpu     32%
```

The average time:

```
(8134.59+8427.09+8167.64+8040.17+8921.16+7915.38+8791.03+9897.57+8693.6+9362.56) /
10 = 86350.79 / 10 = 8635.079
(22359.2+22312.2+22296.8+22315.6+30141.7+22112.7+26041.5+29009.1+23570.2+28498.6) /
10 = 248657.6 / 10 = 24865.76
(0+0+0+0+0+0+0+0+0+0) / 10 = 0 / 10 = 0
(22.36s+22.31s+22.30s+22.32s+30.14s+22.12s+26.04s+29.01s+23.57s+28.50s) / 10 =
248.67 / 10 = 24.867
(1.38s+1.49s+1.45s+1.43s+1.43s+1.35s+1.41s+1.70s+1.43s+1.74s) / 10 = 14.81 / 10 =
1.481
(6.75s+6.94s+6.72s+6.61s+7.49s+6.57s+7.38s+8.20s+7.27s+7.62s) / 10 = 71.55 / 10 =
7.155
(36%+37%+36%+36%+29%+35%+33%+34%+36%+32%) / 10 = 344 / 10 = 34.4


Therefore:
```

```
* Average Execution Time(Based on ctime): 8635.079 ms
* Average Execution Time(Based on chrono): 24865.76 ms
* Average Memory Usage: 0 KB
* Average Real Time: 24.867 s
* Average User Time: 1.481 s
* Average Sys Time: 7.155 s
* Average CPU Usage: 34.4 %
```

And we have:

```
24865.76 / 1.8654305 = 13329
```

Bloody hell! We're up to being 13 thousands time slower!

My guess is that in this type of computations and consecutive I/O operations for small data on the disk, the speed of calculations grows exponentially, whereas the same computations on the RAM have a linear growth.

Let's test the validity of this hypothesis. I will try different values of n in the last two programs, whic are Iteration_RAM and Iteration_HDD, and report the results. Of course, I admit that I might do something malicious and perform this final experiment on a VPS to prevent any harm to my own disk. By the way, it's also good for the credibility of the article's results because the experiment's outcomes won't be heavily overfit to my machine.

Specifications of the testing machine so far:

So far, The code of the article has been tested on a standard Linux machine with the following specifications:

```
OS: Kali GNU/Linux Rolling x86_64
Kernel: 5.16.0-kali5-amd64
CPU: Intel i5-8250U (8) @ 3.400GHz
Memory: 11857MiB
Shell: zsh 5.8.1
```

HDD details:

```
duf
```

```
┌──────────────────────────────────────────────────────────────────────────────
────────┐
│ 2 local devices
│
├──────────────┬─────────┬─────────┬─────────┬──────────────────────────────────┬──────┬──
──────┤
│ MOUNTED ON │   SIZE │   USED │  AVAIL │              USE%              │ TYPE │
FILESYSTEM │
├──────────────┼─────────┼─────────┼─────────┼──────────────────────────────────┼──────┼──
──────┤
│ /          │ 200.2G │ 164.0G │  26.0G │ [################....]  81.9% │ ext4 │
/dev/sda6  │
│ /boot/efi  │ 512.0M │  12.5M │ 499.5M │ [....................]   2.4% │ vfat │
/dev/sda5  │
└──────────────┴─────────┴─────────┴─────────┴──────────────────────────────────┴──────┴──
──────┘
┌──────────────────────────────────────────────────────────────────────────────
──────────────┐
│ 6 special devices
│
├──────────────────┬─────┬────────┬─────┬──────────────────────────────────┬──────────────
──────┤
│ MOUNTED ON     │ SIZE │   USED │ AVAIL │              USE%              │ TYPE
│ FILESYSTEM │
├──────────────────┼─────┼────────┼─────┼──────────────────────────────────┼──────────────
──────┤
│ /dev           │ 5.7G │     0B │  5.7G │                                │ devtmpfs
│ udev       │
│ /dev/shm       │ 5.8G │  23.5M │  5.8G │ [....................]   0.4% │ tmpfs
│ tmpfs      │
│ /run           │ 1.2G │   1.9M │  1.2G │ [....................]   0.2% │ tmpfs
│ tmpfs      │
│ /run/lock      │ 5.0M │     0B │  5.0M │                                │ tmpfs
│ tmpfs      │
│ /run/snapd/ns  │ 1.2G │   1.9M │  1.2G │ [....................]   0.2% │ tmpfs
│ tmpfs      │
│ /run/user/1000 │ 1.2G │ 192.0K │  1.2G │ [....................]   0.0% │ tmpfs
│ tmpfs      │
└──────────────────┴─────┴────────┴─────┴──────────────────────────────────┴──────────────
──────┘
```

```
sudo hdparm -I /dev/sda6
```

```
/dev/sda6:

ATA device, with non-removable media
```

```
        Model Number:       HGST HTS541010B7E610
        Serial Number:      WXF1E673XX4S
        Firmware Revision:  03.01A03
        Transport:          Serial, SATA 1.0a, SATA II Extensions, SATA Rev 2.5,
SATA Rev 2.6, SATA Rev 3.0
Standards:
        Used: unknown (minor revision code 0x006d)
        Supported: 10 9 8 7 6 5
        Likely used: 10
Configuration:
        Logical         max     current
        cylinders       16383   0
        heads           16      0
        sectors/track   63      0
        --
        LBA    user addressable sectors:   268435455
        LBA48  user addressable sectors:  1953525168
        Logical  Sector size:                   512 bytes
        Physical Sector size:                  4096 bytes
        Logical Sector-0 offset:                  0 bytes
        device size with M = 1024*1024:      953869 MBytes
        device size with M = 1000*1000:     1000204 MBytes (1000 GB)
        cache/buffer size  = unknown
        Form Factor: 2.5 inch
        Nominal Media Rotation Rate: 5400
Capabilities:
        LBA, IORDY(can be disabled)
        Queue depth: 32
        Standby timer values: specd by Standard, with device specific minimum
        R/W multiple sector transfer: Max = 16  Current = 16
        Advanced power management level: 254
        DMA: mdma0 mdma1 mdma2 udma0 udma1 udma2 udma3 udma4 udma5 *udma6
             Cycle time: min=120ns recommended=120ns
        PIO: pio0 pio1 pio2 pio3 pio4
             Cycle time: no flow control=120ns  IORDY flow control=120ns
Commands/features:
        Enabled Supported:
           *    SMART feature set
                Security Mode feature set
           *    Power Management feature set
           *    Write cache
           *    Look-ahead
           *    WRITE_BUFFER command
           *    READ_BUFFER command
           *    NOP cmd
           *    DOWNLOAD_MICROCODE
           *    Advanced Power Management feature set
                Power-Up In Standby feature set
           *    SET_FEATURES required to spinup after power up
           *    48-bit Address feature set
           *    Mandatory FLUSH_CACHE
           *    FLUSH_CACHE_EXT
```

```
           *    SMART error logging
           *    SMART self-test
           *    General Purpose Logging feature set
           *    64-bit World wide name
           *    IDLE_IMMEDIATE with UNLOAD
           *    {READ,WRITE}_DMA_EXT_GPL commands
           *    Segmented DOWNLOAD_MICROCODE
           *    Gen1 signaling speed (1.5Gb/s)
           *    Gen2 signaling speed (3.0Gb/s)
           *    Gen3 signaling speed (6.0Gb/s)
           *    Native Command Queueing (NCQ)
           *    Host-initiated interface power management
           *    Phy event counters
           *    Idle-Unload when NCQ is active
           *    NCQ priority information
           *    READ_LOG_DMA_EXT equivalent to READ_LOG_EXT
           *    DMA Setup Auto-Activate optimization
           *    Device-initiated interface power management
           *    Software settings preservation
           *    SMART Command Transport (SCT) feature set
           *    SCT Write Same (AC2)
           *    SCT Features Control (AC4)
           *    SCT Data Tables (AC5)
                unknown 206[12] (vendor specific)
                unknown 206[13] (vendor specific)
           *    Extended number of user addressable sectors
           *    DOWNLOAD MICROCODE DMA command
           *    WRITE BUFFER DMA command
           *    READ BUFFER DMA command
           *    Data Set Management TRIM supported (limit 10 blocks)
           *    Deterministic read data after TRIM
Security:
        Master password revision code = 65534
                supported
        not     enabled
        not     locked
        not     frozen
        not     expired: security count
                supported: enhanced erase
        180min for SECURITY ERASE UNIT. 180min for ENHANCED SECURITY ERASE UNIT.
Logical Unit WWN Device Identifier: 50014ee608088121
        NAA             : 5
        IEEE OUI        : 0014ee
        Unique ID       : 608088121
Checksum: correct
```

Let's move on to the last part of the story, which is also the most fun part of the article in my opinion ^_^

In this section, we want to do some cool scientific stuff and see how the time complexity grows, draw a plot, and make comparisons, and all those things.

Consider this code as the final code of the article:

```
#include <iostream>
#include <chrono>
#include <fstream>
#include <cstring>
using namespace std;

void set_unsigned_long_long_variable(string name, unsigned long long var)
{
    ofstream outfile(name + ".bin"  , ofstream::binary);
    if (outfile)
    {
        outfile.write(reinterpret_cast<const char*>(&var), sizeof(var));
        outfile.close();
    }
    else cerr << "Error opening file: " << name + ".bin" << endl;
}

unsigned long long get_unsigned_long_long_variable(string name)
{
    unsigned long long var = 0;
    ifstream infile(name + ".bin", ifstream::binary);
    if (infile)
    {
        infile.read(reinterpret_cast<char*>(&var), sizeof(var));
        infile.close();
    }
    else cerr << "Error opening file: " << name + ".bin" << endl;

    return var;
}

int main(int argc, char** argv)  {

    // Start execution time
    auto start = chrono::high_resolution_clock::now();

    // Code ---------------------------------------------------------

    if( strcmp(argv[1], "RAM") == 0 )
    {
        unsigned long long n = stoull(argv[2]), counter = n, sum_divable = 0;
        while (counter > 0) {
            if( n % counter == 0 )
                ++sum_divable;
```

```
            --counter;
        }
        cout << sum_divable << endl;
    }
    else if( strcmp(argv[1], "HDD") == 0 )
    {
        set_unsigned_long_long_variable( "n", stoull(argv[2]) );
        set_unsigned_long_long_variable( "counter",
get_unsigned_long_long_variable("n"));
        set_unsigned_long_long_variable( "sum_divable", 0);
        while (get_unsigned_long_long_variable("counter") > 0) {
            if( get_unsigned_long_long_variable("n") %
get_unsigned_long_long_variable("counter") == 0 )
                set_unsigned_long_long_variable( "sum_divable",
get_unsigned_long_long_variable("sum_divable") + 1);
            set_unsigned_long_long_variable( "counter",
get_unsigned_long_long_variable("counter") - 1);
        }
        cout << get_unsigned_long_long_variable("sum_divable") << endl;
    }
    else cout << "ERROR: Wronge input argument." << endl;

    // ----------------------------------------------------------------

    // Stop measuring execution time
    auto end = chrono::high_resolution_clock::now();
    chrono::duration<double, milli> execution_time = end - start;

    // Printing result
    cout << "Execution Time (Based on chrono): " << execution_time.count() << " ms"
<< endl;

    return 0;
}
```

Source code

This code is exactly the same as the previous code, but it is dynamically written. It performs the task based on the inputs received as arguments from the standard input buffer. It can perform calculations on the RAM or the hard disk, depending on the input parameters `n`.

I will test the time values spent on calculations on RAM/hard disk using this code on a Google Colab machine:

```
counter=0; while [ $counter -lt 50 ]; do ./a.out RAM 10; echo "";
```

```
counter=$((counter+1)); done
```

```
(0.05557+0.04354+0.037682+0.055058+0.059299+0.040107+0.044795+0.033862+0.03316+0.030
346+0.031631+0.039248+0.031442+0.030144+0.031636+0.030273+0.044436+0.035037+0.032099
+0.032844+0.029726+0.031982+0.03101+0.034369+0.033597+0.042123+0.03926+0.037468+0.03
927+0.042095+0.039197+0.04276+0.043037+0.058996+0.032882+0.034359+0.032656+0.034285+
0.031125+0.029796+0.035702+0.030816+0.03118+0.036636+0.029955+0.029841+0.029912+0.04
7903+0.038854+0.033695) / 50 = 1.856696 / 50 = 0.03713392

Therefore:
* Average Execution Time(Based on chrono): 0.03713392 ms
```

```
counter=0; while [ $counter -lt 50 ]; do ./a.out RAM 100; echo "";
counter=$((counter+1)); done
```

```
(0.056729+1.59152+0.048881+0.049231+0.04922+0.043113+0.050367+0.044923+0.042751+0.04
3592+0.041487+0.039294+0.043805+0.047856+0.040708+0.044756+0.041053+0.040405+0.04784
1+0.049873+0.040557+0.039935+0.040469+0.042405+0.043633+0.042428+0.040278+0.064401+0
.042753+0.049396+0.059291+0.050488+0.045674+0.047207+0.043165+0.041103+0.041197+0.03
7907+0.0587+0.043548+0.04071+0.04312+0.041312+0.040731+0.041073+0.047526+0.066859+0.
057422+0.042882+0.043553) / 50 = 3.837128 / 50 = 0.07674256

Therefore:
* Average Execution Time(Based on chrono): 0.07674256 ms
```

```
counter=0; while [ $counter -lt 50 ]; do ./a.out RAM 1000; echo "";
counter=$((counter+1)); done
```

```
(0.068803+0.059274+3.0206+0.053834+0.051958+0.051055+0.057953+0.059117+0.042785+0.04
8859+0.045849+0.042131+0.05188+0.042523+0.049469+0.038819+0.0395+0.041249+0.040282+0
.040635+0.037002+0.044957+0.040858+0.038883+0.04276+0.061451+0.042524+0.040479+0.053
96+0.041289+0.039723+0.049896+0.041293+0.039501+0.040528+0.049634+0.053361+0.041704+
0.044657+0.041357+0.040148+0.045694+0.058381+0.041357+0.040849+0.036245+0.044352+0.0
43624+0.040995+0.036902) / 50 = 5.260939 / 50 = 0.10521878

Therefore:
* Average Execution Time(Based on chrono): 0.10521878 ms
```

```
counter=0; while [ $counter -lt 50 ]; do ./a.out RAM 10000; echo "";
counter=$((counter+1)); done
```

```
(0.185036+0.154471+0.165113+0.157347+0.18171+0.132968+0.115558+0.121692+0.121769+0.1
13374+0.116244+0.149405+0.150177+0.125981+0.123877+0.128397+0.132772+0.172368+0.1271
96+0.126897+0.12715+0.115237+0.126738+0.133597+0.123583+0.124886+0.118664+0.125119+0
.129162+0.115475+0.15031+0.15324+0.128633+0.158191+0.209703+0.123722+0.128189+0.1354
94+0.127625+0.167394+0.126804+0.124159+0.154155+0.116885+0.14578+0.119057+0.123312+0
.11629+0.143316+0.114863) / 50 = 6.809085 / 50 = 0.1361817

Therefore:
```

```
* Average Execution Time(Based on chrono): 0.1361817 ms
```

```
counter=0; while [ $counter -lt 50 ]; do ./a.out RAM 100000; echo "";
counter=$((counter+1)); done
```

```
(2.8513+1.01647+0.976667+0.970932+0.97964+0.995396+0.982495+0.97912+0.966483+0.97071
5+0.934009+0.950327+0.974002+0.975446+0.986908+0.980763+1.1284+0.982268+0.983068+0.9
87813+0.993953+1.02334+1.01758+1.11104+0.987194+1.04501+0.979295+0.995212+0.987408+1
.01927+1.01664+0.987328+0.973787+0.975659+0.973153+0.978747+0.952589+1.09384+1.37752
+1.01558+0.982577+1.0207+0.974533+1.11337+0.985381+1.01944+1.04177+1.0397+1.02442+1.
04134) / 50 = 52.319598 / 50 = 1.04639196

Therefore:
* Average Execution Time(Based on chrono): 1.04639196 ms
```

```
counter=0; while [ $counter -lt 50 ]; do ./a.out RAM 1000000; echo "";
counter=$((counter+1)); done
```

```
(11.7602+11.6177+11.6259+9.55161+9.84909+9.6777+9.83686+9.87394+10.1924+9.75504+9.56
854+9.53915+10.2645+9.75044+9.52632+9.57528+10.5565+9.7492+9.98362+13.526+11.1387+11
.8642+11.5241+12.129+13.6267+11.4817+10.573+11.2104+11.0466+11.0755+10.6779+12.0456+
10.9347+11.2068+11.2398+11.4614+11.3066+10.9896+11.3568+10.9231+10.6759+16.8868+14.9
99+16.2878+12.3337+12.8563+12.1443+10.4334+10.7875+10.6904) / 50 = 561.68729 / 50 =
11.2337458

Therefore:
* Average Execution Time(Based on chrono): 11.2337458 ms
```

```
counter=0; while [ $counter -lt 50 ]; do ./a.out RAM 10000000; echo "";
counter=$((counter+1)); done
```

```
(122.437+111.11+116.911+111.118+114.321+112.377+117.222+110.622+110.967+107.209+119.
983+117.109+113.132+112.927+117.166+112.409+110.612+121.124+115.648+99.2176+100.546+
96.6624+97.9584+95.4014+98.3823+98.5019+98.32+98.3984+95.9734+96.6039+97.2092+98.200
6+96.5731+92.4531+93.6754+96.8256+97.0719+92.4971+93.3109+93.8384+92.9399+99.6457+96
.2368+90.9912+90.5309+94.4321+97.6597+93.3925+94.4273+97.0904) / 50 = 5149.3715 / 50
= 102.98743

Therefore:
* Average Execution Time(Based on chrono): 102.98743 ms
```

```
counter=0; while [ $counter -lt 50 ]; do ./a.out RAM 100000000; echo "";
counter=$((counter+1)); done
```

```
(1153.99+1090+927.97+950.748+952.153+934.874+951.585+934.285+931.199+935.237+951.123
+946.343+1044.16+1104.66+1060.29+933.39+932.839+943.719+944.238+936.994+952.082+938.
932+921.093+929.203+930.171+1018.32+1079.7+1059.41+936.524+948.65+950.927+956.4+955.
033+944.512+943.176+933.907+939.347+950.281+1052.11+1087.17+1069.19+952.72+921.658+9
21.936+932.571+954.542+931.82+944.327+943.054+937.911) / 50 = 48496.474 / 50 =
```

```
969.92948

Therefore:
* Average Execution Time(Based on chrono): 969.92948 ms
```

```
counter=0; while [ $counter -lt 50 ]; do ./a.out RAM 1000000000; echo "";
counter=$((counter+1)); done
```

```
(9929.45+9894.97+9401.29+9779.2+9740.14+9711.45+9411.57+9575.79+9740.8+9762.82+10157
.3+9470.99+9807.61+9827.57+9903.24+9546.48+9817.56+9914.88+9835.63+9757.12+9494.87+9
768.5+9804.84+9879.44+9978.79+10010.1+9716.13+9654.43+9452.93+9741.28+9716.39+9671.2
9+9534.45+9422.55+9740.14+9631.31+9571.77+9396.79+9789.33+10178.4+9683.12+9338.5+954
1.9+9655.58+9713.08+9510.57+9530.52+9780.9+9780.28+9769.28) / 50 = 485443.32 / 50 =
9708.8664

Therefore:
* Average Execution Time(Based on chrono): 9708.8664 ms
```

```
counter=0; while [ $counter -lt 5 ]; do ./a.out RAM 10000000000; echo "";
counter=$((counter+1)); done
```

```
(98059.1+98639.7+99670.5+99134.6+99792.6) / 5 = 495296.5 / 5 = 99059.3

Therefore:
* Average Execution Time(Based on chrono): 99059.3 ms
```

```
counter=0; while [ $counter -lt 1 ]; do ./a.out RAM 100000000000; echo "";
counter=$((counter+1)); done
```

```
144
Execution Time (Based on chrono): 982447 ms
```

Iterations	RAM Time Complexity(ms)
10	0.03713392
100	0.07674256
1000	0.10521878
10000	0.1361817
100000	1.04639196
1000000	11.2337458
10000000	102.98743
100000000	969.92948
1000000000	9708.8664
10000000000	99059.3
100000000000	982447

On this machine, it takes only 16 minutes to perform 100 billion iterations on RAM.

Now let's move on to the hard disk:

```
counter=0; while [ $counter -lt 50 ]; do ./a.out HDD 10; echo "";
counter=$((counter+1)); done
```

```
(3.84055+1.47793+1.36111+1.18943+1.25027+1.06003+1.15582+1.31169+0.988887+1.08511+0.
996868+1.02288+1.15027+0.970606+1.30461+1.24018+1.30114+1.01147+1.1003+1.16134+1.053
86+1.00953+1.08559+1.09392+1.02795+1.04478+1.11548+1.02791+1.13904+1.43361+1.41642+1
.34454+1.04273+1.10764+3.06246+1.34444+1.09299+1.02791+0.98998+1.40493+1.2629+1.3766
6+1.1086+1.0706+1.67073+1.05198+1.37261+1.31193+1.34244+1.18202) / 50 = 63.596671 /
50 = 1.27193342

Therefore:
* Average Execution Time(Based on chrono): 1.27193342 ms
```

```
counter=0; while [ $counter -lt 50 ]; do ./a.out HDD 100; echo "";
counter=$((counter+1)); done
```

```
(10.1258+8.54431+8.57312+9.30554+7.19626+8.57967+7.81807+6.76671+8.19687+11.3563+8.7
6072+8.89945+7.93218+7.15151+9.06521+8.95591+9.04594+8.77245+9.37136+7.49228+8.1699+
7.5217+8.37757+8.27919+9.53812+9.14541+8.06788+7.49188+8.24989+10.9161+8.53771+8.612
68+6.91477+8.33953+8.44163+6.35417+6.49716+8.86702+8.77506+9.01159+8.56295+6.69499+8
.88008+8.89771+9.1845+8.44426+9.3007+9.81496+8.01047+8.43163) / 50 = 424.24087 / 50
= 8.4848174

Therefore:
* Average Execution Time(Based on chrono): 8.4848174 ms
```

```
counter=0; while [ $counter -lt 50 ]; do ./a.out HDD 1000; echo "";
counter=$((counter+1)); done
```

```
(83.7239+89.2962+87.393+86.0499+82.4329+92.4111+83.0183+93.2184+98.6888+88.1423+93.1
49+104.383+94.8324+95.2283+104.694+98.6493+92.5682+91.8928+101.478+95.2763+96.7348+9
6.7451+95.3717+96.2584+95.0568+104.564+94.3572+92.9837+94.5809+96.7802+95.193+94.719
+94.5504+98.0115+104.346+90.0224+89.8571+84.6048+90.1275+89.0622+92.2847+91.2552+89.
7745+91.942+90.7563+89.0043+94.7468+89.3705+85.9757+95.2816) / 50 = 4660.8444 / 50 =
93.216888

Therefore:
* Average Execution Time(Based on chrono): 93.216888 ms
```

```
counter=0; while [ $counter -lt 10 ]; do ./a.out HDD 10000; echo "";
counter=$((counter+1)); done
```

```
(878.908+868.796+868.818+868.208+921.707+945.511+940.521+901.627+872.928+876.26) /
```

```
10 = 8943.284 / 10 = 894.3284

Therefore:
* Average Execution Time(Based on chrono): 894.3284 ms
```

```
counter=0; while [ $counter -lt 5 ]; do ./a.out HDD 100000; echo "";
counter=$((counter+1)); done
```

```
(8944.41+9181.89+9291.08+9176.2+9019.04) / 5 = 45612.62 / 5 = 9122.524

Therefore:
* Average Execution Time(Based on chrono): 9122.524 ms
```

```
counter=0; while [ $counter -lt 5 ]; do ./a.out HDD 1000000; echo "";
counter=$((counter+1)); done
```

```
(93730+92787+91645.4+92535.1+94361.4) / 5 = 465058.9 / 5 = 93011.78

Therefore:
* Average Execution Time(Based on chrono): 93011.78 ms
```

```
counter=0; while [ $counter -lt 1 ]; do ./a.out HDD 10000000; echo "";
counter=$((counter+1)); done
```

```
64
Execution Time (Based on chrono): 904366 ms
```

```
counter=0; while [ $counter -lt 1 ]; do ./a.out HDD 100000000; echo "";
counter=$((counter+1)); done
```

```
81
Execution Time (Based on chrono): 9.06014e+06 ms
```

And we know that:

```
9.06014e+06 = 9060140
```

Iterations	HDD Time Complexity(ms)
10	1.27193342
100	8.4848174
1000	93.216888
10000	894.3284
100000	9122.524
1000000	93011.78

Iterations	HDD Time Complexity(ms)
10000000	904366
100000000	9060140
1000000000	NULL
10000000000	NULL
100000000000	NULL

When we stored only three variables on the disk, what happened? Our 100 million iterations took about 3 hours, and we couldn't even reach the billionth order... This is in contrast to the approximately 9 seconds it took for 100 million iterations on RAM.

From the observed behavior of the processing time on the hard disk, we can deduce that the three values we don't know (NULLs at the table) are likely to increase tenfold each time, so the first one would probably take around 1 day, the second one would be around 12 days, and the third one should be 125 days...!

Please cosider to the table below:

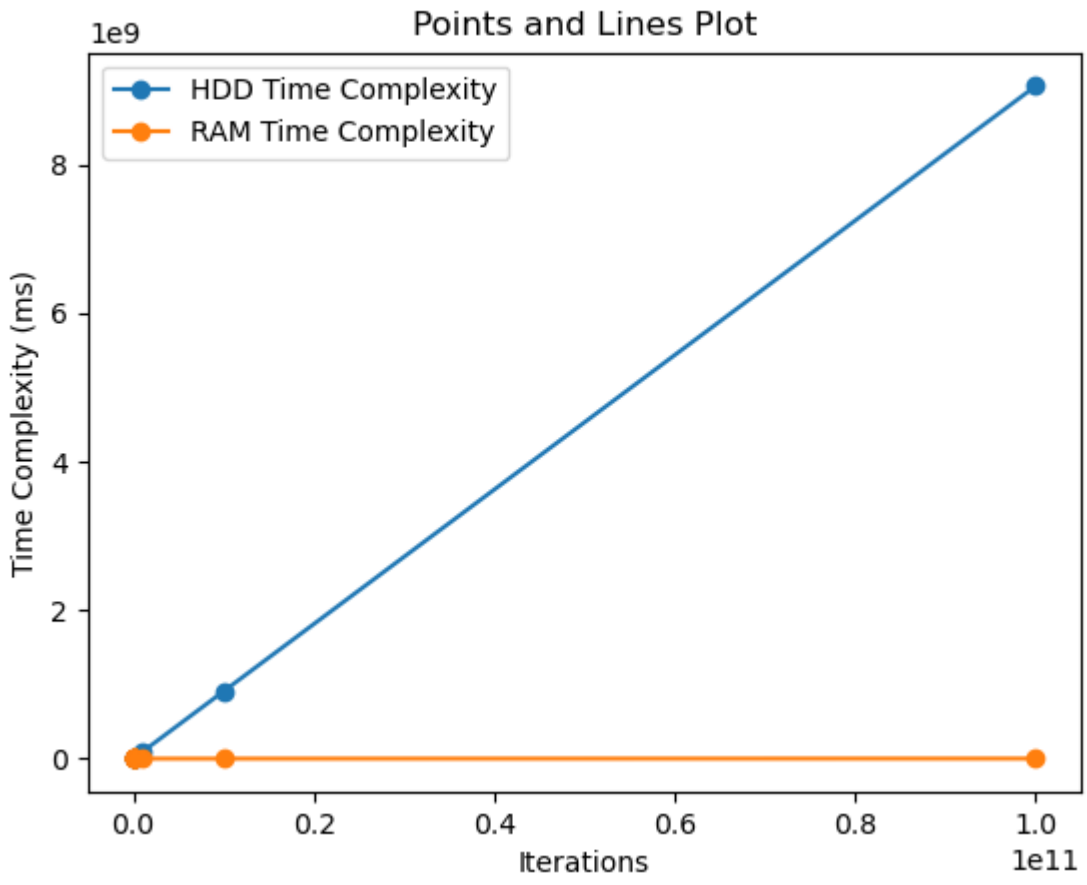
Iterations	RAM Time Complexity(ms)	HDD Time Complexity(ms)	Ratio(HDD/RAM)
10	0.03713392	1.27193342	34.2526030
100	0.07674256	8.4848174	110.5620844
1000	0.10521878	93.216888	885.9339368
10000	0.1361817	894.3284	6567.1701851
100000	1.04639196	9122.524	8718.0753949
1000000	11.2337458	93011.78	8279.6764014
10000000	102.98743	904366	8781.3240897
100000000	969.92948	9060140	9341.0296179
1000000000	9708.8664	NULL	NULL
10000000000	99059.3	NULL	NULL
100000000000	982447	NULL	NULL

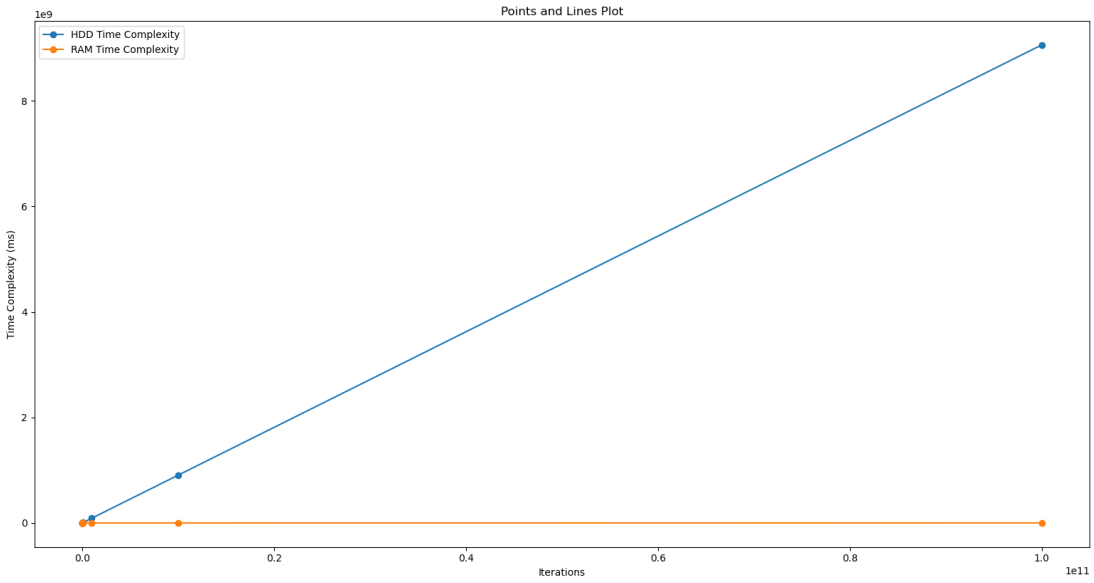
And if we want to draw a plot of the growth, I will estimate the last three values and plot a points-and-lines chart using the Python code below:

```
import matplotlib.pyplot as plt

iterations = [10, 100, 1000, 10000, 100000, 1000000, 10000000, 100000000,
1000000000, 10000000000, 100000000000]
```

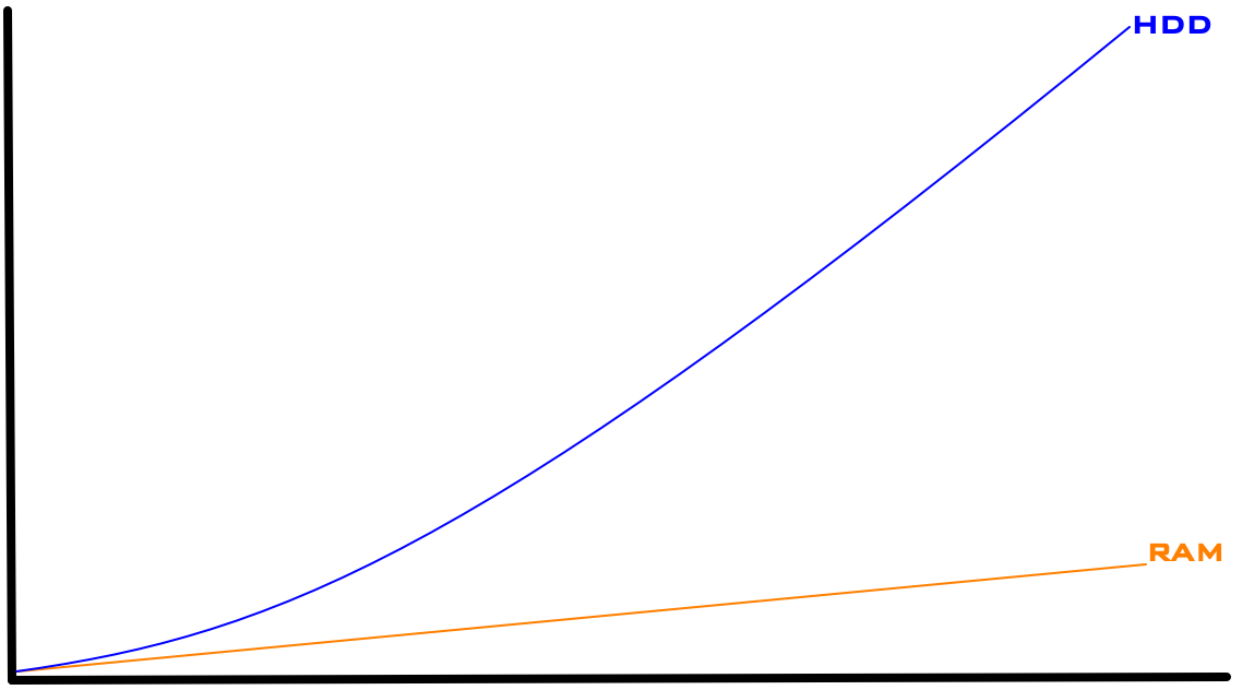
```
ram_times = [0.03713392, 0.07674256, 0.10521878, 0.1361817, 1.04639196, 11.2337458,
102.98743, 969.92948, 9708.8664, 99059.3, 982447]
# hdd_times = [1.27193342, 8.4848174, 93.216888, 894.3284, 9122.524, 93011.78,
904366, 9060140, None, None, None]
last_hdd_time = 9060140 # Estimate times
hdd_times = [1.27193342, 8.4848174, 93.216888, 894.3284, 9122.524, 93011.78, 904366,
9060140, last_hdd_time*10, last_hdd_time*100, last_hdd_time*1000]


plt.plot(iterations, hdd_times, marker='o', label='HDD Time Complexity')
plt.plot(iterations, ram_times, marker='o', label='RAM Time Complexity')

plt.xlabel('Iterations')
plt.ylabel('Time Complexity (ms)')
plt.title('Points and Lines Plot')

plt.legend()

plt.show()
```

Source code





If we want to abstractly represent this chart, it should look like this:



Lastly, I would like to mention that you can imply parallelism in this comparison and suddenly go from being a few thousand-fold slow, up to being several hundred thousand-fold slow! Of course, if you have a disk for testing... I'll stop the experiment here because I don't

want to damage colab's disks. ×□×

One of the practical side results could be that for performing time complexity tests, if the range of execution times is very small and changes significantly each time, we need to increase the number of tests to obtain a good estimation. On the other hand, if the range of output times is very large and changes very little each time, a smaller number of tests can give us a decent estimation. □

Some related question and answer:

- *https://superuser.com/questions/1173675/how-much-faster-is-memory-ram-compared-to-ssd-for-random-access*
- *https://stackoverflow.com/questions/1371400/how-much-faster-is-the-memory-usually-than-the-disk*
- *https://queue.acm.org/detail.cfm?id=1563874*

And that's it, the hard disk is on average 100,000 times slower than RAM:))

Future Work

In your own article, you can work on scenarios that can show even greater speed differences. You can perform parallelism in the second phase of the experiment, use different disks, or even use different clusters on different RAID systems. Additionally, if feasible, you can actually let the disk perform calculations for 125 days to obtain a real comparison result:)

Copyright

Author

- Behrad B. (behroora@yahoo.com)

References

- https://faculty.etsu.edu/tarnoff/ntes2150/memory/memory.htm
- https://electronics.stackexchange.com/questions/562038/how-is-a-memory-location-accessed-by-random-access/562041#562041
- Database System Concepts, 6th Edition
- http://www.foundersatwork.com/steve-wozniak.html
- https://jadi.net/2014/02/radiogeek-36-wozniak-and-narenji-90
- https://blog.codinghorror.com/understanding-user-and-kernel-mode
- https://askubuntu.com/questions/920920/how-to-interpret-time-real-user-and-sys
- https://stackoverflow.com/questions/1311402/what-is-the-difference-between-user-and-kernel-modes-in-operating-systems
- https://unix.stackexchange.com/questions/53302/why-would-the-real-time-be-much-higher-than-the-user-and-sys-times-combine
- https://stackoverflow.com/questions/556405/what-do-real-user-and-sys-mean-in-the-output-of-time1

Issues related to this Article

This free and open-source article is available at Free Books/Documents repository with the following link:

https://github.com/TadavomnisT/Free_Books-Documents/tree/main/Articles/4-RAM-vs-HDD-En

The Persian version of this article:

https://github.com/TadavomnisT/Free_Books-Documents/tree/main/Articles/4-RAM-vs-HDD-Pr

Any scientific mistakes, issues, questions, or discussions related to this article can be posted through Issues section of repository or sent by email:

- https://github.com/TadavomnisT/Free_Books-Documents/issues
- behroora@yahoo.com