Facultatea de Automatică și Calculatoare

N.A.M.E

Project realised by *N.A.M.E*

Students:

Claudiu Lintes
Adrian Kalamar
Raul Danciu
Axel Alabeatrix
Victor Simon
Luka Mirza

Timișoara

May, 2023

Chapter 1

Introduction

1.1 Context

- **What have we done?**

  We have developed a benchmark application designed to evaluate the performance of CPU and RAM. This application measures computational speed and memory efficiency through a series of standardized tests.

- **Which component and what specific functionality (or functionalities) are we testing?**

  The benchmark focuses on testing the processing power of the CPU and the read/write speeds and latency of the RAM. Specific functionalities tested include single-threaded and multi-threaded performance through file compression and decompression tasks using the LZW (Lempel-Ziv-Welch) algorithm.

1.2 Motivation

- **What is the motivation behind choosing this application?**

  The motivation for selecting compression and decompression tasks utilizing the LZW algorithm lies in their effectiveness as standardized benchmarks for evaluating CPU performance. These tasks represent common computational workloads encountered in various computing scenarios.

- **What is our project about?**

  This project aims to implement a standard benchmark tool to evaluate CPU and RAM performance. While it does not introduce new methodologies, it combines established testing techniques to provide a measure of CPU and RAM capabilities.

Chapter 2

State of the Art

In this chapter, we will explore existing benchmarks that have served as sources of inspiration for our project. We will describe two similar benchmarks and then select one for a more detailed comparison with our application.

2.1 Existing Benchmarks

2.1.1 RAM Benchmark: Memtest86

Memtest86 is a widely recognized tool for testing and diagnosing system memory (RAM). It works by systematically testing different areas of RAM to identify potential errors or faults. Memtest86 runs independent of the operating system, allowing for thorough testing without any interference from software. It performs various memory access patterns, including sequential and random reads and writes, to assess the stability and reliability of the RAM modules.

2.1.2 CPU Benchmark: Geekbench

Geekbench is a popular cross-platform benchmarking tool used to evaluate CPU and system performance. It measures both single-core and multi-core CPU performance by executing a series of compute-intensive tasks, including integer and floating-point calculations, memory access patterns, and image processing algorithms. Geekbench provides detailed scores for various performance metrics, such as integer performance, floating-point performance, and memory bandwidth.

2.2 Comparison with Geekbench

For the purpose of this analysis, we will compare our application with Geekbench, a widely used benchmarking tool for assessing CPU and system performance.

2.2.1 Functionality

Our CPU benchmark application functions by executing a series of predefined tests designed to stress different aspects of CPU functionality, such as arithmetic operations, data processing, and cache utilization. It measures both single-threaded and multi-threaded performance, providing detailed metrics on execution times, throughput, and latency, similar to Geekbench.

2.2.2 Advantages and Disadvantages

Advantages

- Simplicity: Our application is much lighter than Geekbench and is also much simpler, which allows anyone with a basic knowledge of code to understand how our application works.
- Options: In the Geekbench GUI, there are no options for benchmarking unlike our application. We can also export the results since they are stored in a csv file, unlike Geekbench which displays its results in the browser.

Disadvantages

- Popularity: Geekbench is a very popular application which makes it easier to compare with other CPUs because it will be very easy to find CPU benchmark results from this application. There are also many
- User Interface: Geekbench offers a user-friendly interface with intuitive navigation and comprehensive result visualization, whereas our application may have a less polished UI design.

Chapter 3

Design and Implementation

3.1 Benchmark Features and Measurements

The benchmark program evaluates CPU performance by measuring the time taken to compress and decompress files using the Lempel-Ziv-Welch (LZW) algorithm. Key features include:

- **Single-Thread and Multi-Thread Scores**: Measures performance in single-thread and multi-thread modes.
- **Execution Time Measurement**: Tracks time to compress and decompress files of various sizes.
- **Cache Fill Option**: Option to fill CPU cache before benchmark runs for accurate measurements.
- **Configuration Settings**: Configurable number of runs, result logging, and other settings.

3.2 Algorithms and Special Features

The benchmark uses the LZW algorithm for compression and decompression, which is efficient for lossless data compression.

3.2.1 LZW Compression Algorithm

The LZW algorithm compresses data by replacing strings of characters with single codes. It starts with a dictionary of single-character strings and adds new strings as it processes the input.

```
private static void encodeLZW(String inputFile, String outputFile) throws IOException {
    int alphabetSize = mainAlphabetSize;
    HashMap<String, Integer> LZWdictionary = new HashMap<>();
    for (int i = 0; i < alphabetSize; i++) {
        LZWdictionary.put(String.valueOf((char) i), i);
    }
    // Encoding process...
}
```

Listing 3.1: Snippet of LZW Compression

3.2.2 LZW Decompression Algorithm

The decompression reverses the compression, reconstructing the original data from the codes.

```
private static void decodeLZW(String inputFile, String outputFile) throws IOException {
    int alphabetSize = mainAlphabetSize;
    HashMap<Integer, String> LZWdictionary = new HashMap<>();
    for (int i = 0; i < alphabetSize; i++) {
        LZWdictionary.put(i, String.valueOf((char) i));
    }
    // Decoding process...
}
```

Listing 3.2: Snippet of LZW Decompression

3.3 Software Architecture

The software architecture is modular, with clear separation of concerns:

- **Main Class (`FileCompresserBenchmark`)**: Manages initialization, configuration, and task execution.
- **Task Class (`MainTask`)**: Implements `Runnable` for concurrent task execution.
- **Utility Methods**: Handle compression, decompression, CPU information retrieval, and database writing.
- **Configuration Class (`BenchmarkSettings`)**: Stores benchmark configuration settings.
- **Specifications Class (`Specs`)**: Contains system specifications.
- **Results Class (`Results`)**: Stores single-thread and multi-thread scores.

The benchmark uses Java's `ExecutorService` for multi-threading, leveraging multi-core CPUs efficiently.

3.4 Git Repository

The source code and related files for this benchmark are available in the Git repository[1].

Chapter 4

Usage

4.1 Step-by-Step Guide

This section provides a detailed guide on how to use the CPU benchmark tool. Screenshots and descriptions of the user interface (UI) elements are included to facilitate understanding.

4.1.1 Launching the Application

To start the benchmark tool, simply launch the GUI class from the root project directory. This will launch the main application window. You can also launch it from the executable project file.

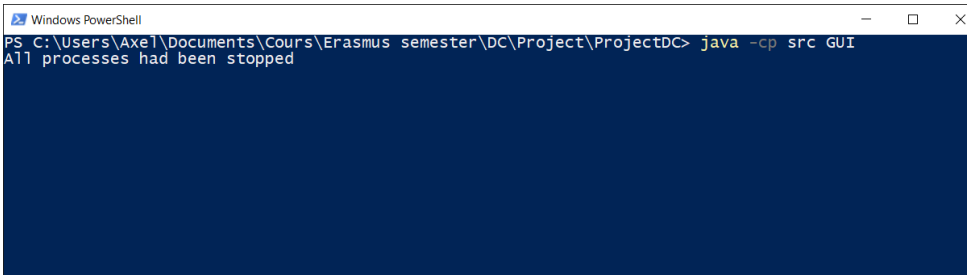
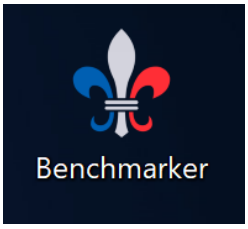


Figure 4.1: Launching the Benchmark Tool



4.1.2 Main Window Overview

The main window of the benchmark tool display several buttons:

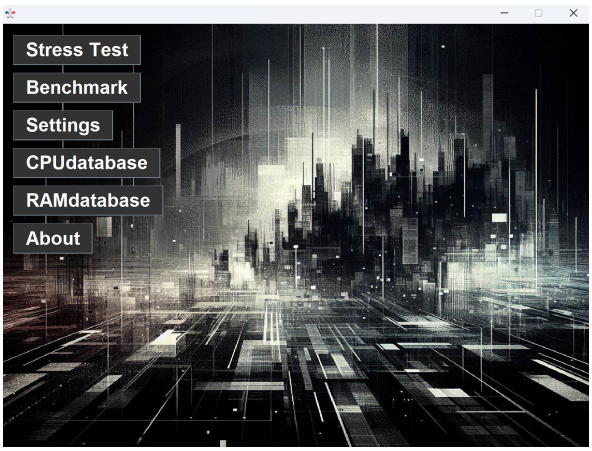


Figure 4.2: Main Window Layout

You can then choose what you want to do by pressing the corresponding button, to do a benchmark, simply go to benchmark, but you can also simply stress the CPU or RAM by going in the Stress Test section. Clicking on the ABOUT button will open the project documentation and the CPU and RAM databases will open a csv file with all the results we collected.

4.1.3 Setting Benchmark Parameters

In the Settings, you can set the parameters for the benchmark, if you not sure what to do and you want just launch a benchmark, don't do anything:

- **Database**: You can choose if you want to add your result to two database, only the main or not.
- **Number of Runs**: Specify how many times the benchmark should be run to ensure accuracy.
- **Cache Settings**: Choose whether to fill the CPU cache before running the benchmark for more accurate results.

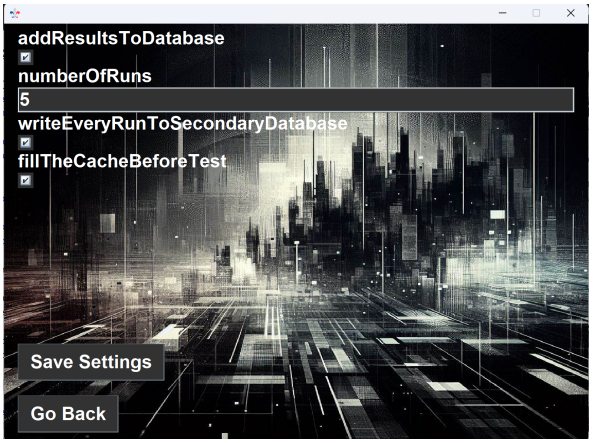


Figure 4.3: Settings

4.1.4 Starting the Benchmark

After setting the desired parameters, click the **Save Settings** button, the **Go Back** and then the **Benchmark** button to access to the benchmark menu.

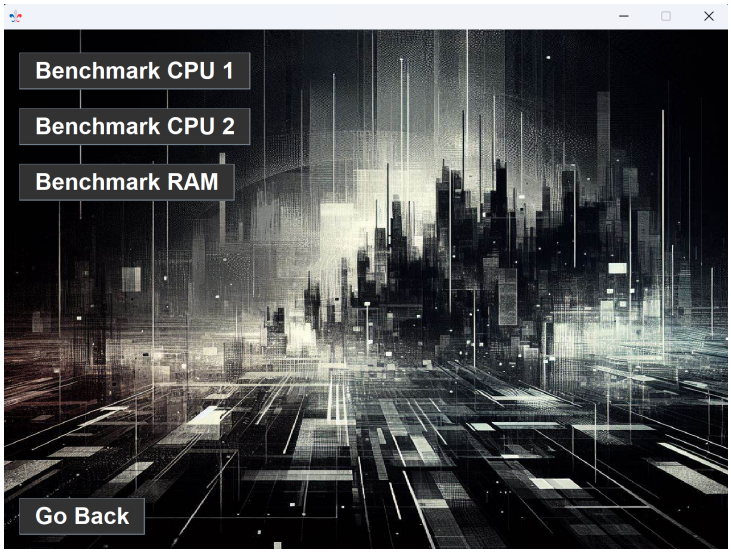


Figure 4.4: Starting the Benchmark

You can now choose if you want to launch a CPU benchmark or a RAM benchmark and then click on the corresponding button.

4.1.5 Monitoring Progress

After clicking on the button for the desired benchmark type, the benchmark starts, you can see the progress of the benchmark using the progress bar and interrupt the benchmark with the **Stop Test** button.

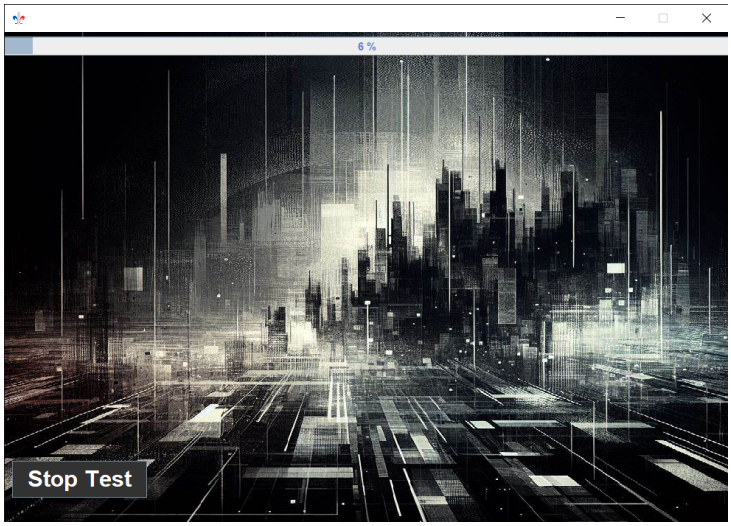


Figure 4.5: Monitoring Progress

4.1.6 Viewing Results

For the CPU benchmark

Once the benchmark is completed, the results will be displayed by itself. The results include the scores for single-thread and multi-thread performance.

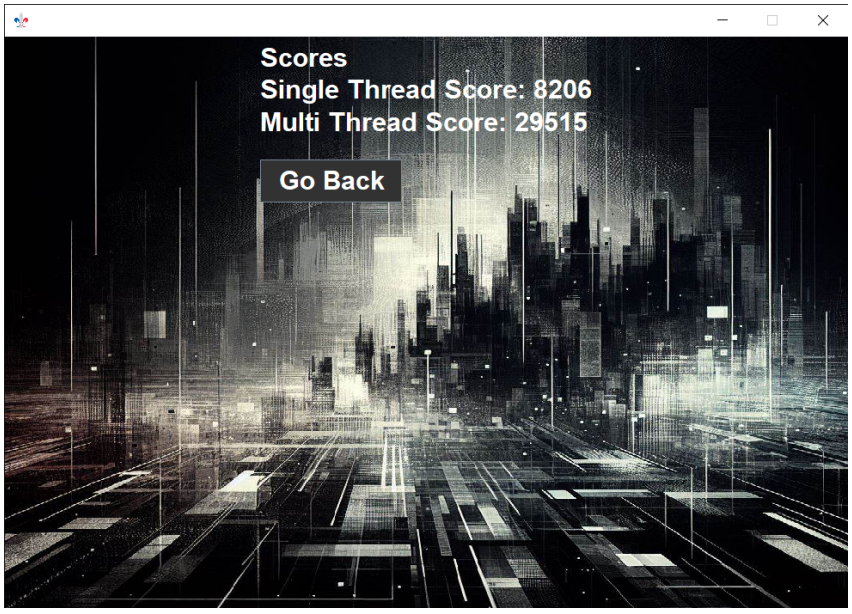
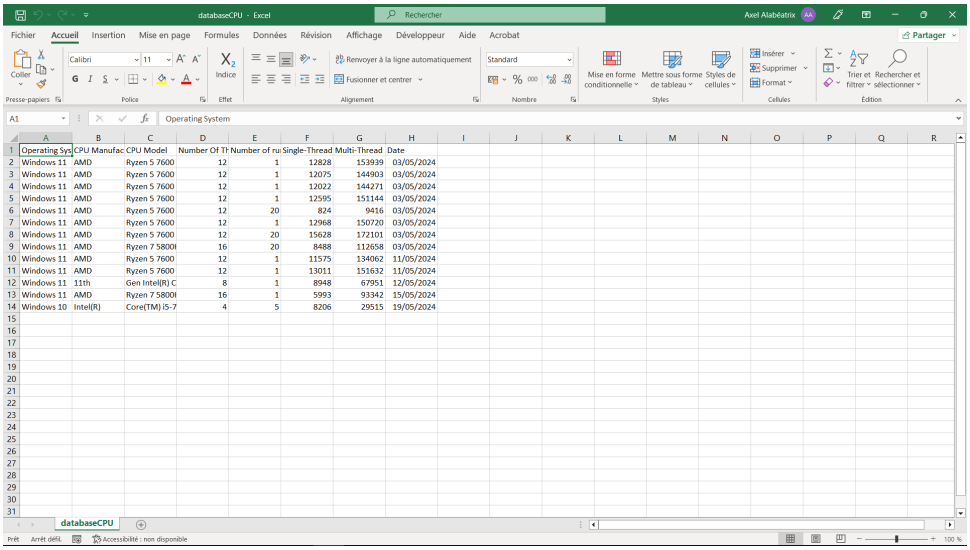


Figure 4.6: Viewing Score for RAM

You can also see the old results if you have not unchecked the option in the settings by clicking on the **CPUdatabase** button in the main menu. It opens with the default application for csv files on your computer.



Operating Sys	CPU Manufac	CPU Model	Number Of Th	Number of ru	Single-Thread	Multi-Thread	Date
Windows 11	AMD	Ryzen 5 7600	12	1	12828	153939	03/05/2024
Windows 11	AMD	Ryzen 5 7600	12	1	12075	144903	03/05/2024
Windows 11	AMD	Ryzen 5 7600	12	1	12022	144271	03/05/2024
Windows 11	AMD	Ryzen 5 7600	12	1	12595	151144	03/05/2024
Windows 11	AMD	Ryzen 5 7600	12	20	824	9416	03/05/2024
Windows 11	AMD	Ryzen 5 7600	12	1	12968	150720	03/05/2024
Windows 11	AMD	Ryzen 5 7600	12	20	15628	172101	03/05/2024
Windows 11	AMD	Ryzen 7 5800I	16	20	8488	112658	03/05/2024
Windows 11	AMD	Ryzen 5 7600	12	1	11575	134062	11/05/2024
Windows 11	AMD	Ryzen 5 7600	12	1	13011	151632	11/05/2024
Windows 11	11th	Gen Intel(R) C	8	1	8948	67951	12/05/2024
Windows 11	AMD	Ryzen 7 5800I	16	1	5993	93342	15/05/2024
Windows 10	Intel(R)	Core(TM) i5-7	4	5	8206	29515	19/05/2024

Figure 4.7: Viewing Results History

For the RAM benchmark

Once the RAM benchmark is completed, the results will be displayed by itself and it will be given as a score, the higher the score, the better the performance.

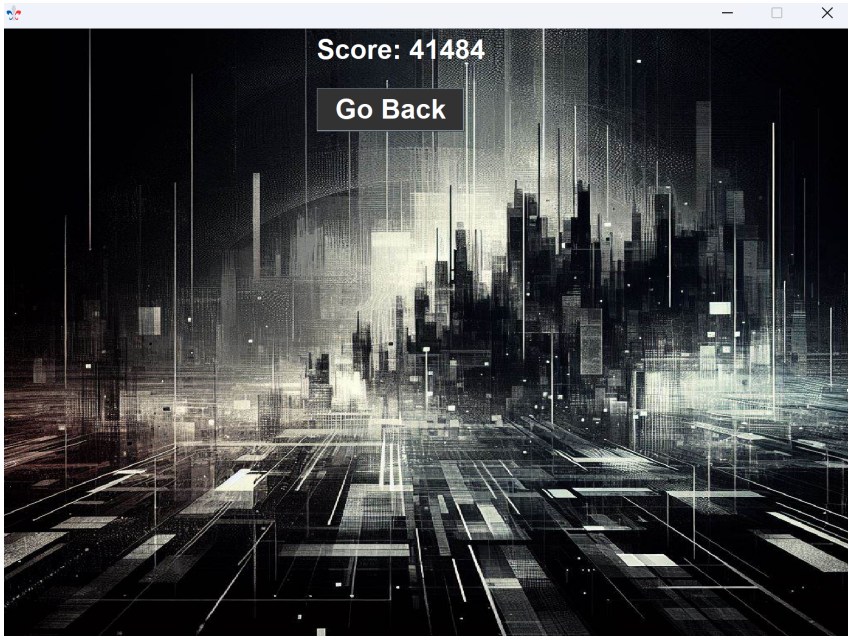


Figure 4.8: Viewing Results

You can also see the old results by clicking on the **RAMdatabase** button in the main menu. It opens with the default application for csv files on your computer.

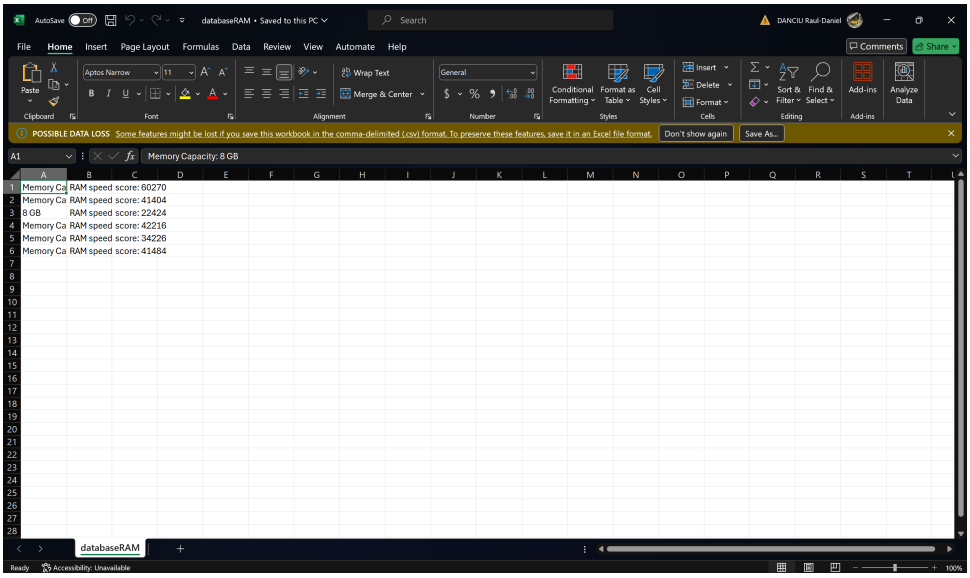


Figure 4.9: Viewing Results History for RAM

4.2 Output and Scoring

4.2.1 For the CPU benchmark

The benchmark outputs the result in the form of a score for the single thread and for the multi thread which is displayed at the end of the benchmark and is also saved in a csv file with the date of the benchmark and the configuration used during execution.

4.2.2 For the RAM benchmark

Here, the benchmark outputs the result in the form of a score again, which is displayed at the end of the benchmark and is also saved in a csv file and will display the memory capacity and the frequence.

By following these guidelines, users can effectively utilize the benchmark tool to evaluate CPU and RAM performance on their systems.

Chapter 5

Results

These tables are taken from data collected on different machines[2].

5.1 Data

We have tested the benchmark on different configuration to collect data :

5.1.1 For the CPU benchmark

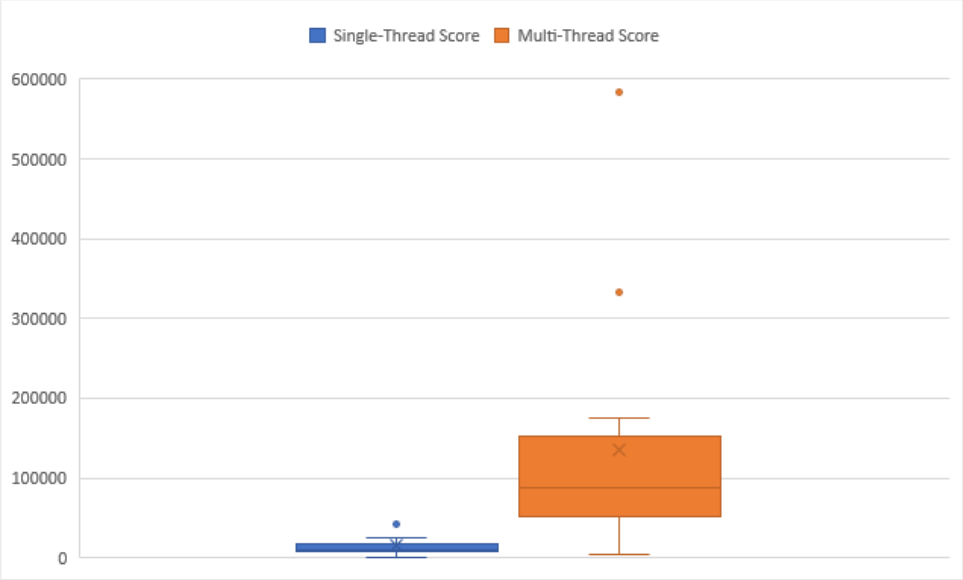


Figure 5.1: Box plot of CPU benchmark score on different PCs with the LZW Algorithm

Figure 5.2: Box plot of CPU benchmark score on different PCs with the Huffman Algorithm

5.1.2 For the RAM benchmark

Figure 5.3: Box plot of RAM benchmark score on different PCs

5.2 Analysis

With the data we have we can conclude several things, first of all, logically for the CPU benchmark the best scores are for the most powerful CPUs, however this is only if we exclude the non-standard points from the graph , if we look at them we see that they all come from Linux, it indeed seems that the execution of the Linux benchmark provides, perhaps this is due to the fact that certain instructions are more optimized on Linux than on Windows.

For RAM, the operating system seems to have little importance because the differences between operating systems are negligible, and we see that the higher the speed of the RAM, the better the score, which is expected.

Chapter 6

Conclusions

- **What did we learn?**
  - Understanding System Resources:

    We gained a deeper understanding of how CPU and RAM work and interact with software. This includes learning how to measure and interpret resource usage effectively.

  - Java Programming:

    Our Java skills improved significantly, particularly in areas related to multithreading, memory management, and optimization techniques.

  - Stress Testing:

    We learned how to design and implement stress tests to push the limits of CPU and RAM, and how to analyze the results to understand the performance and stability of a system under load.

  - Problem-Solving:

    Debugging complex issues and optimizing code for better performance were key skills we developed during the project.

  - Team Collaboration:

    Working as a team taught us valuable lessons in communication, task delegation, and integrating different parts of a project efficiently.

- **What mark would I give to me? How much did I contribute to the teamwork (project, application, presentation)? What mark do I think my colleagues deserve?**

- **What was hard, what did we enjoy, what did we hate, what did we like and dislike about the CO project?**

  Likes:

  - Real-World Relevance:

The project had practical applications which made it interesting and motivating.

– Skill Development:
We appreciated the opportunity to develop and hone our technical and soft skills.

Dislikes:

– Unclear Requirements:
At times, the project requirements were not clearly defined, which led to confusion and extra work.

– Working as a team:
Even though working as a team makes the project less time consuming, being able to properly work as a team is a challenge. Assigning tasks, making sure everything synchronizes well and making sure everyone finishes on time can get very hard.

- **How would you change (improve) the project meetings for the next generation of students?**

– Clear Agenda:

Ensure each meeting has a clear agenda and objectives to keep discussions focused and productive.

– Regular Check-ins:

Implement more frequent but shorter check-ins to monitor progress and address issues promptly.

– Role Assignments:

Assign specific roles and responsibilities to each team member to ensure accountability and efficient task management.

– Guest Speakers:

Invite industry professionals or alumni to share their experiences and provide insights into similar projects.

– Peer Reviews:

Incorporate regular peer review sessions to get feedback and improve the quality of work continuously.

– Flexible Meeting Times:

Schedule meetings at times that accommodate everyone's availability to ensure full attendance and participation.

– Interactive Workshops:

Include workshops on specific skills related to the project, such as advanced Java programming, performance profiling, and data analysis techniques.

By implementing these changes, we believe future students will have an even better experience and achieve greater success with their CPU and RAM stress application projects

Bibliography

[1] *Git repository*. https://github.com/ClaudiuLintes/ProjectDC, 2023. [Online; Accesat: 19.05.2023].

[2] *Data from the execution of the benchmark on different systems*. https://github.com/ClaudiuLintes/ProjectDC/blob/main/saves/databasefinal.csv, 2023. [Online; Accesat: 21.05.2023].