THE UNNAMED BENCHMARK

Project realised by *The Unnamed*

Students:

Cristian-Ovidiu FAUR
Sergiu VARGA
Mohammad ZEIN Al-Abidin
Andrei DUNCA

Timișoara
May, 2024

Chapter 1

Introduction

The Unnamed Benchmark is an application developed with Python. It aims to stress the CPU and RAM of a computer system using different algorithms and provides an interface for displaying the computer information, storing it together with a score and the options to run individual tests, based on user input, displaying graphs for the results.

1.1 Context

- We chose a simplistic retro-looking desktop application, combining Python UI with Python scripts to create an easy-to-use final product.
- For the RAM memory we use sequential/random read/write operations on a given number of MB, while for the CPU we use complex matrix operations, but also Pi operations together with digits of Pi computation.

1.2 Motivation

- Our motivation was to delve deep into the world's most popular programming language for its flexibility. Based on other programming knowledge, we wanted to put together a project with a language we never used before.
- We combined classical benchmarking algorithms like the digits of Pi calculation and reading/writing from a generic array, with an extensive algorithm for computing matrix operations on processes running in parallel.

Chapter 2

State of the art

- Our main inspiration in creating this application comes from two widely used benchmarks that stress the CPU.
- One of them is Matthew x83 which is an online stress test on CPU that uses parallelism to run algorithms on CPU giving you the possibility to run a multi-thread or a single-thread test.
- Another source of inspiration is Super PI, a popular benchmarking software that calculates pi to a specified number of digits after the decimal point—up to a maximum of 32 million.
- The Unnamed Benchmark gives you the possibility to stress your machine's CPU by using both techniques presented in these applications which I may say is an advantage over them.However, our application implementing the same concepts won't support parameters that are as big as they can get in previously mentioned applications due to the fact that the algorithms implementations could be improved.

Chapter 3

Design and implementation

We are going to offer some insights into our application's development process and organisation.

- **Main standalone features** we implemented are computing a certain number of digits of Pi and computing the RAM reading and writing speed for a given number of MB. These algorithms operate in separate scripts, while the matrix benchmark operates in four different scripts. The main benchmark, together with the score computation have their own script, and the plotting of the several graphs are also put together in one script.
- **The Database** implemented as a simple CSV file, is managed in 2 scripts. One for retrieving already existing information and one for storing the result given upon running the main benchmark from the top of the 'Benchmarking' section of the application.
- **The Python libraries** we used are popular libraries like pandas and numpy, for the CSV management and fast operations on matrices and arrays using **Fortran**, but also other, more specific ones. For operating system management, computer system data retrieval and also subprocess management we used multiprocessing, os, threading and subprocess. For the UI we used tkinter and PIL and for the plotting, time and basic calculations matplotlib, mpmath, time.
- In **the development** of the application we used Git and Github and we worked on different operating systems, such as Windows and Linux. The link to our repository: https://github.com/sxergiu/dc-benchmark-proj

Chapter 4

Usage

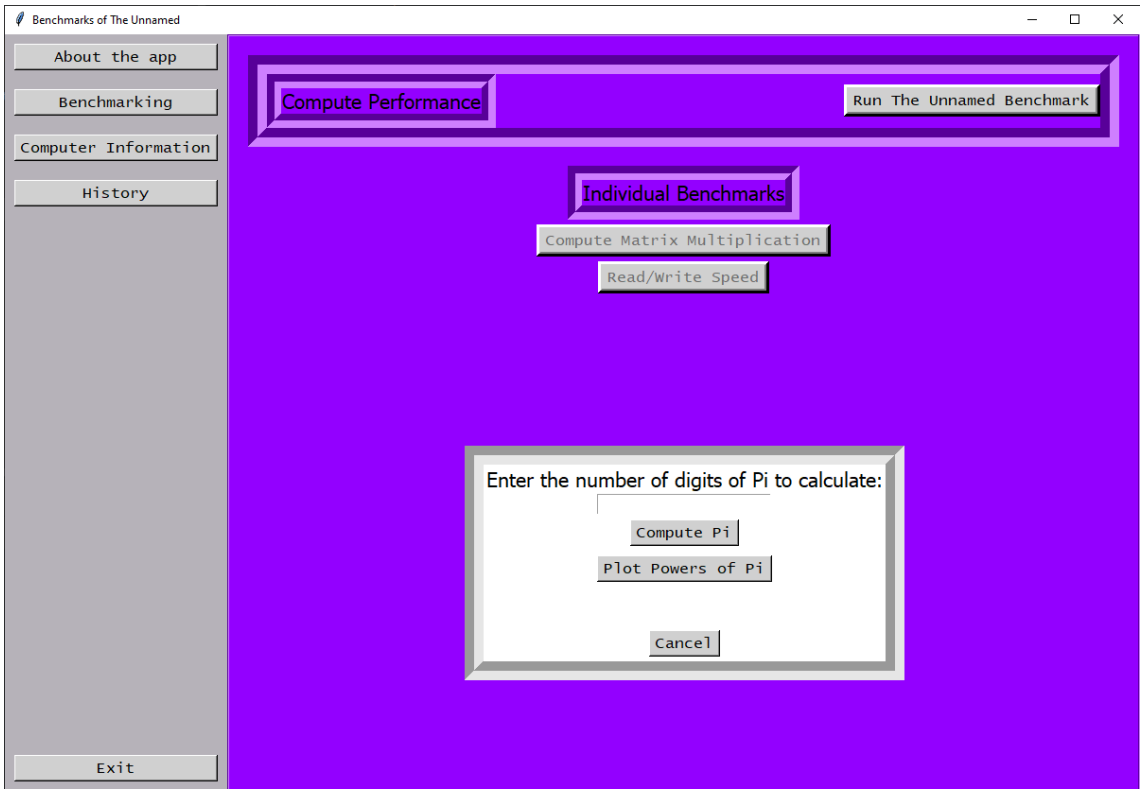

Figure 4.1: Most encompassing view of the application

- The usage of the application is simple.The **'About the app'** section displays a description and a short guide on navigating the menu. The **'Computer Information'** section displays all the necessary retrieved information about the computer system: Operating System, Total RAM, CPU and GPU information. The **'History'** section is used to display the specifications once again, together with the score and also previous results.
- The **'Benchmarking'** section offers the user the possibility to run the displayed individual benchmarks. For Read/Write Speed and Pi, a separate frame will be displayed to ask the user for input (e.g number of MB to benchmark).

- For the **Compute Pi** button, the time taken will be displayed, while for the Plot **Powers of Pi**, for the given number of digits, Pi operations will be computed and graph will be displayed. For the **Read/Write Speed** button the times for each Sequential/Random Reading/Writing will be displayed both as text and as a graph. For the **Compute Matrix Multiplication** the extensive algorithm will be run and the normalized graph of the time variations for the tests will be displayed.All the previously mentioned graphs are simply displayed for a short duration upon finishing the certain benchmark.

- The **Run The Unnamed Benchmark** button will put together the individual benchmarks and will compute the score. For the formula of the score, we chose a weighted average of the normalised results depending on the execution time. Since time is the single factor we took into consideration, we took liberty creating the motto of our benchmark **'Lower is better.'**, contrary to popular score calculation beliefs.

```
digits = 1000  # digits of pi
mb = 100 # read and write

mat_op_times = compMat.mat()
pi_op_times, time_for_pi_digits = compPi.powers_of_pi(digits)
_, ram_times = ramTest(mb)

max_mat_time = max(mat_op_times)
max_pi_time = max(pi_op_times)
max_ram_time = max(ram_times)

normalized_mat_times = [time / max_mat_time for time in mat_op_times]
normalized_pi_times = [time / max_pi_time for time in pi_op_times]
normalized_ram_times = [time / max_ram_time for time in ram_times]

normalized_avg_mat = sum(normalized_mat_times) / len(normalized_mat_times)
normalized_avg_pi = sum(normalized_pi_times) / len(normalized_pi_times)
normalized_avg_ram = sum(normalized_ram_times) / len(normalized_ram_times)
```

Listing 4.1: The calculation of the normalized values, which are attributed weights and summed up to give the total score.

```
def mat():
times = []
cnt = 0
sum = 0

for _ in range(number_of_tests):
    array_of_mat = matrixGen.generate_matrices()
    start = time.perf_counter()
    generate_chunks(array_of_mat)
    MatrixInversion.inverse_matrix_parallel(array_of_mat)
    MatrixTransposition.transpose_matrix_parallel(array_of_mat)
    end = time.perf_counter()
    cnt += 1
    sum += end - start
    times.append(end - start)
print("Time spent: ", sum)
print("Average time is: ", sum / cnt)
return times
```

Listing 4.2: The extensive matrix operation main algorithm

Chapter 5

Results

- Individual benchmark example:

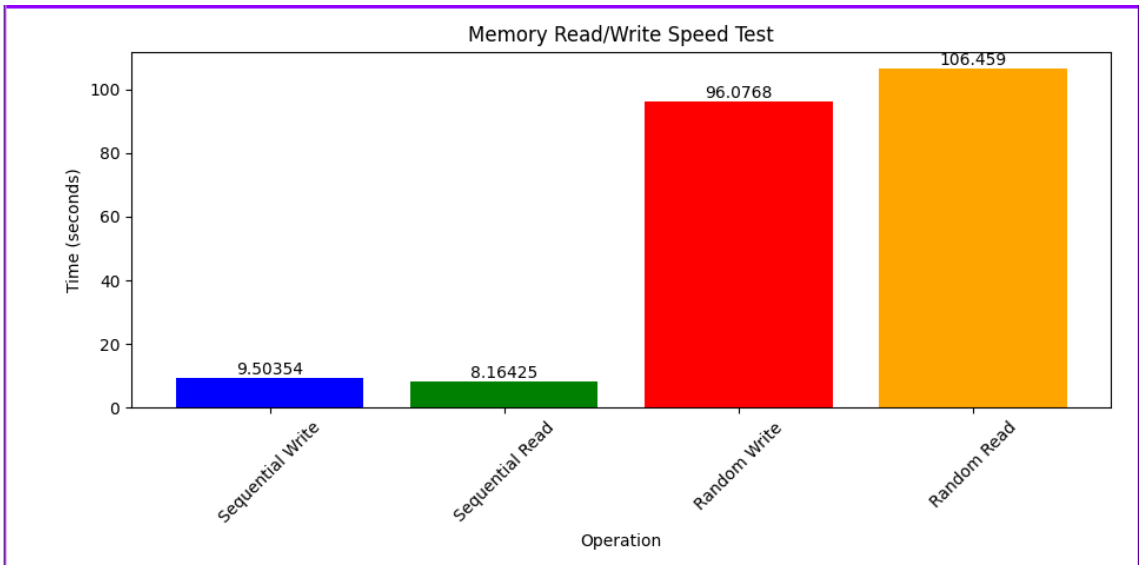

Figure 5.1: RAM memory benchmark results for 1GB

  - Results:

History

OS	CPU	RAM	GPU	SCORE
Windows 10	Intel64 Family 6 Model 140 Stepping 1, GenuineIntel	7.70 GB	Intel(R) Iris(R) Xe Graphics	71.55874793
Linux (specif	x86_64	15.53 GB	01:00.0 VGA compatible controller [0300]: NVIDIA Corporation TU106 [(	57.5365187
Windows 10	Intel Core i5-10600K (12 CPUs)	15.9 GB	NVIDIA GEFORCE GTX 2060 SUPER	44.14566799
Windows 10	Intel64 Family 6 Model 140 Stepping 1, GenuineIntel	7.70 GB	Intel(R) Iris(R) Xe Graphics	71.55874793
Windows 10	Intel64 Family 6 Model 158 Stepping 12, GenuineIntel	15.92 GB	Radeon RX 580 Series	72.13810578

Figure 5.2: History view

  - Remember, the history view displays the score only for the main benchmark. (i.e. the combined individual ones) Lower is better!

Chapter 6

Conclusions

- **What did we learn?**

  During this semester, we learned the hardships of developing software in an orchestrated group. While learning a whole no programming language and battling its most hidden insights, we learned some stuff about the connection between algorithms and physical hardware components.

- **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?**

  I think appreciating my own work is a useful quality, however I couldn't come to the point to grade myself. I think I put everything that has been taught to me in practice in the best way I could and in the best interest of the project and class. I think we worked as a team and everybody did what they had to do. Improvements could have been made in the direction of starting the workflow earlier and reduce procrastination.

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

  The whole process was hard. There was a lot of stuff we had find out on our own and unfortunately, the expectations with each project update was not luminating in any way. The informations we had to provide regarding the project, especially the starting ones were more on the way to throw us in the dark than clearing stuff up. Fortunately, we managed. I know the intention of the Project and it outweighs the hardships we encountered. And I do not know a way in which this could have been improved, especially regarding a "hardware" subject such as Digital Computers. Apart from that, strictly speaking from my opinion, the real changes that can be made place themselves way higher up the ladder. :)

Bibliography

[1] *Python*. https://www.w3schools.com/python/, 2024. [Online; Accesat: 05.24.2024].

[2] *Github Repo*. https://github.com/sxergiu/dc-benchmark-proj, 2024. [Online; Accesat: 24.05.2024].

[3] *Inspiration Benchmark*. https://www.matthew-x83.com/online/cpu-benchmark.php, 2024. [Online; Accesat: 05.24.2024].

[1] [2] [3]