CO PROJECT TITLE

Project realised by Echipa Team

Students:

Fazecas Dora

Golban Diana

Constanstin Mihai

Mage Andrei

Chapter 2

2.1 Context

We have decided to implement CPU benchmarks, namely calculating the digits of Pi, and made a multi-threading benchmark, documenting ourselves on the average runtimes for fixed workloads.

2.2 Motivation

We decided to implement various benchmarks so that we could cover several use cases of the computer raw power by using them.

Chapter 3

State of the art

2 benchmarks that inspired our project were the Bailey–Borwein–Plouffe (BBP) Formula and Monte Carlo Simulation Benchmark benchmark for calculating the digits of Pi and the Newton-Raphson method to find the square root of a large number.

Compared to BBP, our benchmark uses the Monte Carlo:

1. Monte Carlo Method for Pi Calculation

Functionality:

- **Description:** The Monte Carlo method uses random sampling to approximate the value of pi. It involves generating random points within a unit square and counting how many fall inside the quarter circle inscribed within the square.

- **Implementation:** The benchmark runs the Monte Carlo simulation using different numbers of digits to see how the time taken to approximate pi changes.
- **Advantages:**
    - **Simplicity:** Easy to implement and understand.
- **Disadvantages:**
    - **Accuracy:** Convergence to an accurate value of pi is slow.
    - **Randomness:** Performance can vary due to the stochastic nature of the method.

2. Newton-Raphson Method for Root Finding

Functionality:

- **Description:** The Newton-Raphson method is an iterative technique for finding successively better approximations to the roots (or zeroes) of a real-valued function.
- **Implementation:** This benchmark calculates the square root of a large number using the Newton-Raphson method, testing the performance with different thread counts.
- **Advantages:**
    - **Speed:** Fast convergence to an accurate result.
    - **Deterministic:** Performance is consistent and not dependent on random factors.
- **Disadvantages:**
    - **Complexity:** Implementation is more complex than Monte Carlo methods.
    - **Parallelization:** Requires careful handling of dependencies between iterations for parallel execution.

Chapter 4

Design and implementation

Monte Carlo Benchmark

Benchmark Features and Measurements:

- **Precision Control:** Users can specify the number of points to be used in the Monte Carlo simulation, determining the precision of the pi approximation.
- **Performance Measurement:** The benchmark measures the time taken to compute pi to the specified precision, providing insights into the computational efficiency of the hardware.

Implementation:

The core algorithm used in this benchmark is the Monte Carlo method for approximating pi, implemented as follows:

```
class CPUDigitsOfPI:
    def __init__(self, numbers):
        self.numbers = numbers

    def start(self):
        for n in self.numbers:
            self.monte_carlo(n)

    def monte_carlo(self, n):
        inside_circle = 0
        total_points = 0

        for _ in range(n):
            x = random.uniform(0, 1)
            y = random.uniform(0, 1)
            distance = math.sqrt(x**2 + y**2)

            if distance <= 1:
                inside_circle += 1
            total_points += 1

        pi_estimate = 4 * inside_circle / total_points
        return pi_estimate
```

Newton-Raphson Benchmark

Benchmark Features and Measurements:

- **Precision Control:** Developers can specify the tolerance level for the root-finding process, determining how close the approximation should be to the actual square root.
- **Performance Measurement:** The benchmark measures the time taken to compute the square root to the specified tolerance, providing insights into the computational efficiency of the hardware.

Implementation:

The core algorithm used in this benchmark is the Newton-Raphson method, implemented as follows:

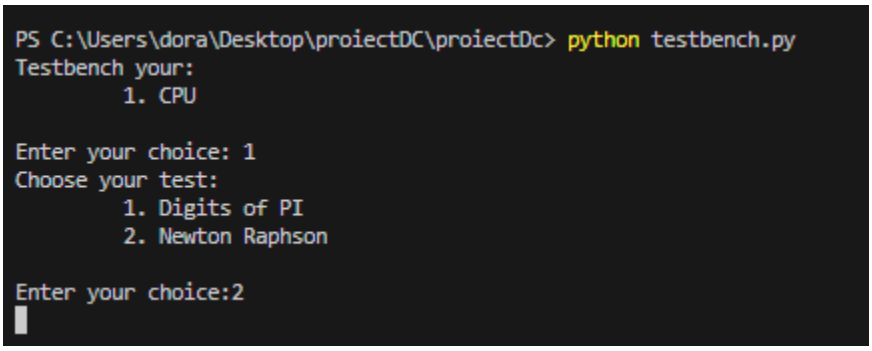
```
class CPUMultiPcRoots:

    def __init__(self, numbers , nThreads):
        self.numbers = numbers
        self.nThreads = nThreads
        self.results = []
        self.threads = []

    def Newton_Raphson(self, n):
        x = n
        while True:
            root = 0.5 * (x + n / x)
            if abs(root - x) < 0.0000001:
                break
            x = root
        return root

    def start(self):
        with multiprocessing.Pool(self.nThreads) as p:
            self.results = p.map(self.Newton_Raphson, self.numbers)

        return self.results
```

Chapter 5

Usage

User Interface:

- **Command-Line Interface:** The benchmark features a simple terminal interface where users can select the type of test they want to run (CPU or Memory) and then choose a specific benchmark.
- **Benchmark Selection:**
    - **CPU Tests:**
        - **Digits of Pi:** Users can specify the number of digits of pi to calculate.
        - **Newton-Raphson:** Users can run the Newton-Raphson method to find square roots of large numbers.



Data Visualization

Plotting Results:

- **Function:** plot(self, result) generates a plot of runtime versus the number of digits of pi calculated.
- **Implementation:**

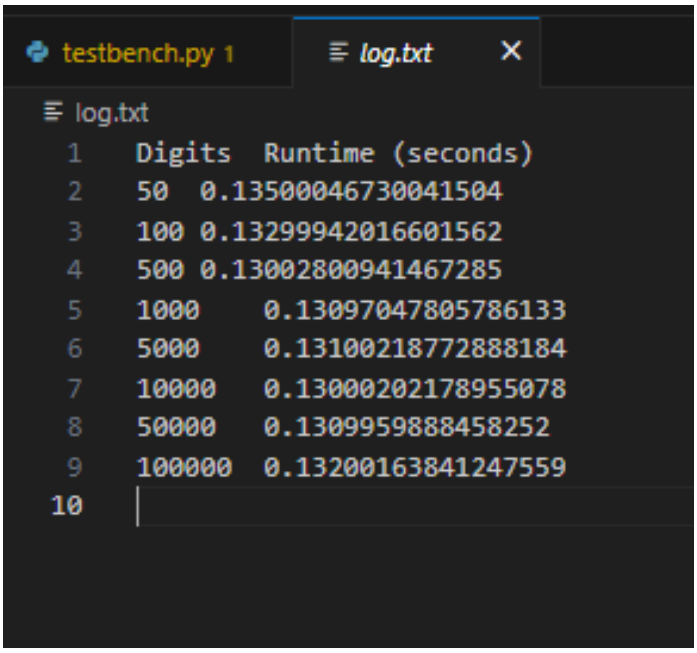
```
def plot(self, result):
    plt.plot(result.keys(), result.values(), marker='o')
    plt.xlabel('Digits of Pi')
    plt.ylabel('Runtime (seconds)')
    plt.title('Monte Carlo Pi Calculation Runtime')
    plt.grid(True)
    plt.show()
```

- **Function:** plot(result) generates a plot of runtime versus the number of threads used in the Newton-Raphson calculation.

- **Implementation:**

```
def plot(result):
    plt.plot(result.keys(), result.values(), marker='o')
    plt.xlabel('Number of Threads')
    plt.ylabel('Runtime (seconds)')
    plt.title('Newton Raphson Calculation Runtime')
    plt.grid(True)
    plt.show()
```

In addition, the runtime values are stored inside the log.txt file:



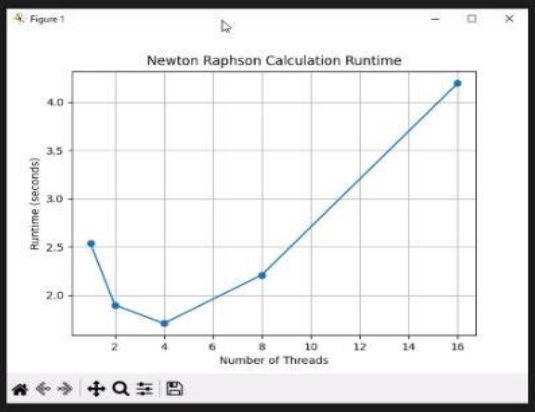
Step-by-Stept instructions:

1. Clone the repository
2. Compile and run the progam with the command: python testbench.py
3. You will receive as output a graph with the values for the operation you selected and can also find the data in the log.txt f
4. The following results were obtained by running the benchmark on two different machines with varying hardware configurations

testbench.py M log.txt M

log.txt

```
1   2.5393760204315186
2   1.8980004787445068
4   1.7069971561431885
8   2.2089977264404297
16  4.1960024833679199
```

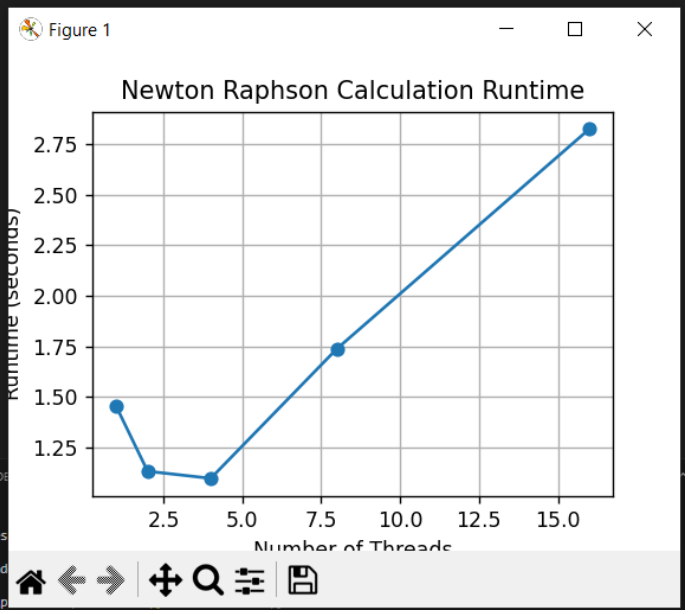


PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
        1. CPU

Enter your choice: 1
Choose your test:
        1. Digits of PI
        2. Newton Raphson

Enter your choice:2
```

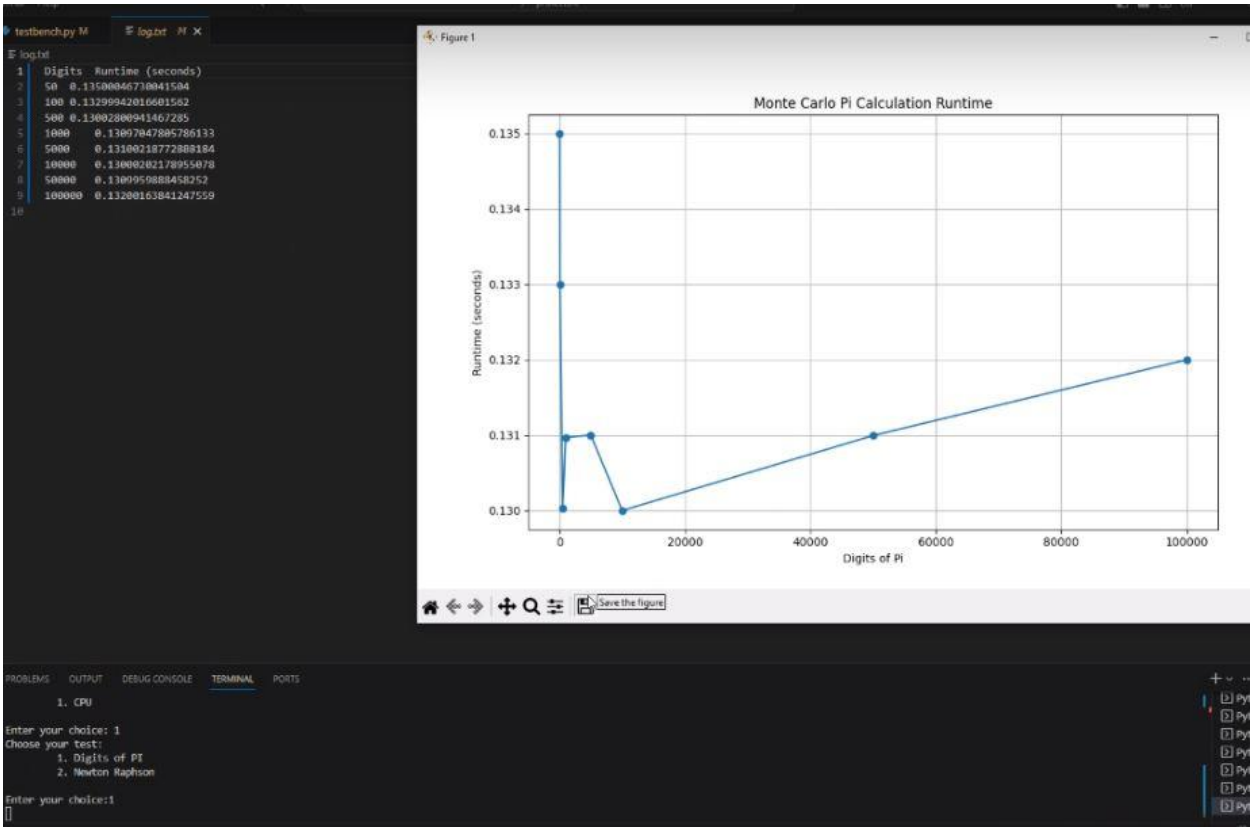
log.txt

```
Digits  Runtime (seconds)
50  0.13500046730041504
100 0.13299942016601562
500 0.13002800941467285
1000    0.13097047805786133
5000    0.13100218772888184
10000   0.13000202178955078
50000   0.1309959888458252
100000  0.13200163841247559
```



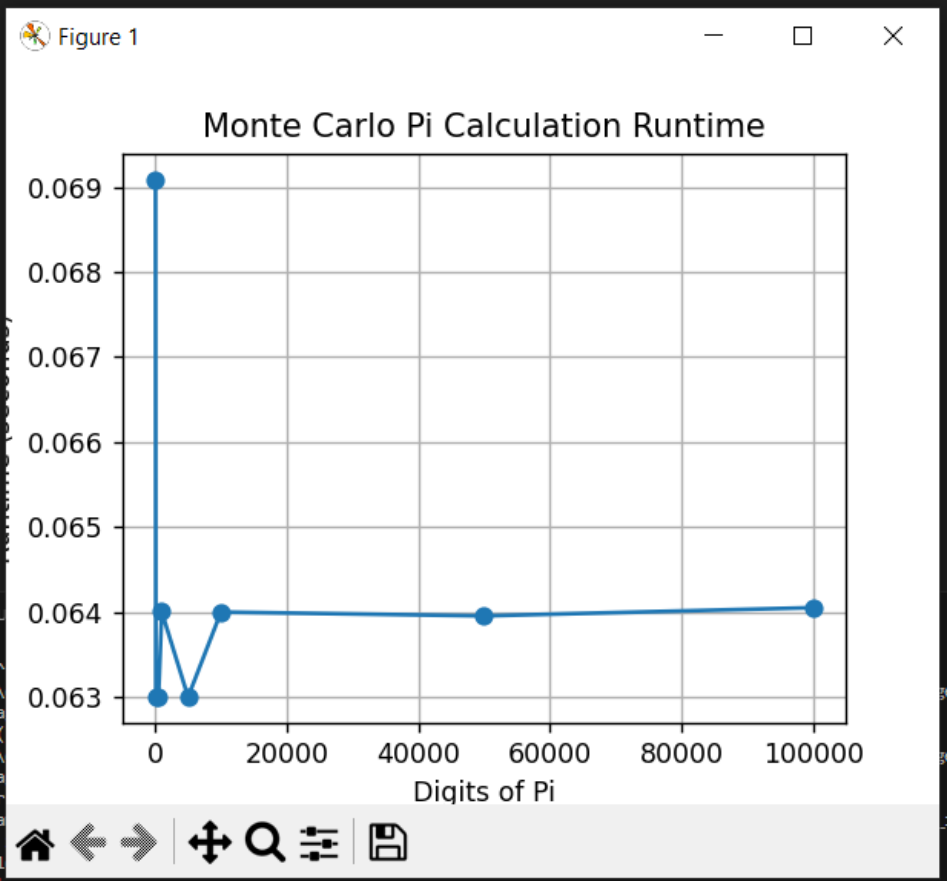
PROBLEMS 1 OUTPUT DE

```
Enter your choice:2
Newton Raphson multiproces

Newton Raphson multithread

PS C:\Users\dora\Desktop\p
```


testbench.py M
log.txt M
log.txt
Digits Runtime (seconds)
50 0.13500046710041504
100 0.13299942016601562
500 0.13002800941467285
1000 0.1300704780578613 3
5000 0.13100218772888184
10000 0.13000202178955078
50000 0.1300959088458252
100000 0.13200163841247559
Figure 1
Monte Carlo Pi Calculation Runtime
Runtime (seconds)
Digits of Pi
0.135
0.134
0.133
0.132
0.131
0.130
0
20000
40000
60000
80000
100000
Save the figure
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
1. CPU
Enter your choice: 1
Choose your test:
1. Digits of PI
2. Newton Raphson
Enter your choice:1

log.txt

```
Digits  Runtime (seconds)
50  0.13500046730041504
100 0.13299942016601562
500 0.13002800941467285
1000    0.13097047805786133
5000    0.13100218772888184
10000   0.13000202178955078
50000   0.1309959888458252
100000  0.13200163841247559
```




```
ROBLEMS 1
^^^^^^
File "C:\Users\
on311\site-packa
cls.mainloop(
File "C:\Users\
on311\site-packa
first_manager
File "C:\Progra
py", line 1504,
self.tk.mainl
```

Capter 6

Conclusions:

Fazecas Dora: Based on this project, I have gained a lot of knowledge about the CPU performance benchmarking by researching several algorithms (e.g. Newton Raphson)

I would give myself 9/10

Golban Casandra: By working on this project I have learned about the versatility of this programming language as well as the differences and types of operations.

I would give myself 8/10

Constantin Mihai: I learned about the differences between multi-processing and multi-threading, as well as the Global Interpreter Lock in Python

I would give myself: 10/10

Mage Andrei: I leaned about the Monte Carlo formula for computing Pi's digits.

I would give myself: 8/10