Digital Computers Benchmarking Project

Project realised by:

Oboroceanu Alexandru
Mihoci Paula
Picek Ionut
Gherga Mario

Timișoara
May, 2024

Overview of the Project:

The project me and my colleagues designed aims to benchmark the cpu and ram of the computer using different computational operations such as sorting algorithms, calculating pi using the monte carlo method and generating prime numbers recursively.The benchmarks then will either be displayed on the screen or saved in a file using two separate loggers.

The most important aspect of this project is the menu class. Here all of the other functions and classes are called letting the user choose which of the benchmark tests he wants to run on his machine also provides a way to keep the code neat and clean.

```
public class Menu {
    public static void main(String[] args) {
        try {
            Scanner scanner = new Scanner(System.in);
            int choice;
            boolean exit = false;

            do {
                System.out.println("Benchmark Menu:");
                System.out.println("1. Single Benchmark");
                System.out.println("2. Multiple Benchmarks");
                System.out.println("3. Benchmark with File Logging");
                System.out.println("4. Benchmark with Timer Pause and Resume");
                System.out.println("5. Run Benchmark");
                System.out.println("6. Test Timer with Main Thread Interruption");
                System.out.println("7. Run Benchmark with Time Conversion");
                System.out.println("8. Run Benchmark with MonteCarlo");
                System.out.println("9. Generate primes");
                System.out.println("0. Exit");
                System.out.print("Enter your choice: ");
                choice = scanner.nextInt();

                switch (choice) {
                    case 1:
                        singleBenchmark();
                        break;
```

Other important classes include : **ConsoleLogger** Logs messages to the console,**FileLogger** Logs messages to a file, **Timer** Measures elapsed time with start, stop, pause, and resume functionality. These are all helpful classes that help with keeping the code organised and easy to edit in the future if the need occurs.

Timișoara
May, 2024

```java
import Timing.Timer;

public class ConsoleLogger implements ILogger {
    @Override
    public void write(long value) {
        System.out.println(value);
    }

    @Override
    public void write(String value) {
        System.out.println(value);
    }

    @Override
    public void write(Object... values) {
        for (Object value : values) {
            System.out.print(value + " ");
        }
        System.out.println();
    }

    @Override
    public void close() {
        // No resources to close for ConsoleLogger
    }
```

```java
public class Timer implements ITimer {
    private long startTime;
    private long endTime;
    private long pausedTime;
    private boolean isPaused;

    @Override
    public void start() {
        startTime = System.nanoTime();
        isPaused = false;
    }

    @Override
    public void stop() {
        if (isPaused) {
            resume();
        }
        endTime = System.nanoTime();
    }

    @Override
    public void pause() {
        if (!isPaused) {
            pausedTime = System.nanoTime();
            isPaused = true;
        }
```

Timișoara
May, 2024

Dummy Benchmark:

Now to get into the benchmarking.The first testing was done on a dummy benchmark wich just sorted an array using bubble sort to test all the other functionalities and validate the framework we were going with.

Monte Carlo Method:

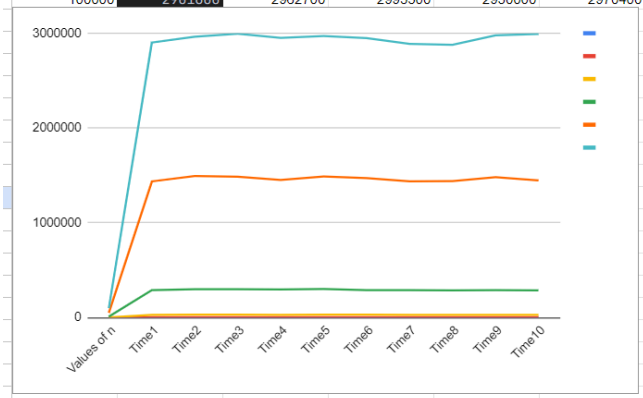
The other big step up was implementing the monte Carlo algorithm to approximate the value of pi.

```
                return 4.0 * pointsInsideCircle / numPoints;
            }

            @Override
private double monteCarloPi(int numPoints) {
    Random random = new Random();
    int pointsInsideCircle = 0;

    for (int i = 0; i < numPoints; i++) {
        double x = random.nextDouble();
        double y = random.nextDouble();

        if (x * x + y * y <= 1) {
            pointsInsideCircle++;
        }
    }

    return 4.0 * pointsInsideCircle / numPoints;
}

@Override
public void clean() {
    // No cleanup needed for this benchmark
}
```

Also i will provide a excel of a few runs of this method for different values as well as a graph (The excel will also be provided in the git hub repository)

Timișoara
May, 2024

Values of n	Time1	Time2	Time3	Time4	Time5	Time6	Time7	Time8	Time9	Time10
50	2400	2000	2000	2000	2000	2000	2000	2000	2500	2100
100	3200	3400	3300	3300	3300	3200	3200	3400	3200	4000
1000	29000	30000	31200	29500	29800	30400	28900	28600	29100	29000
10000	289400	299600	298900	296000	301100	289200	288400	287300	289800	287300
50000	1436200	1494300	1486500	1451800	1487400	1471900	1436300	1440600	1482100	1448000
100000	2901600	2962700	2995500	2950000	2970400	2949400	2887000	2877400	2977200	2993000



Recursive prime generation:

The other and final benchmark included in our project was implementing the generation of recursive primes.The recursive prime generation benchmark will generate primes recursively until a stack overflow occurs. It will then record the last prime reached and the runtime, and calculate a custom score based on these values.
(This implementation is not 100% working :( )

Timișoara
May, 2024

```
public static void generatePrimesAndDisplayResults() {
    long startTime = System.currentTimeMillis();
    int lastPrime = 0;

    try {
        lastPrime = generatePrimes(2, 1000000); // Start from 2 and generate primes up to a large number
    } catch (StackOverflowError e) {
        System.out.println("Stack Overflow Exception occurred.");
    }

    long endTime = System.currentTimeMillis();
    long runtime = endTime - startTime;

    // Calculate custom score
    double customScore = calculateCustomScore(lastPrime, runtime);

    System.out.println("Last prime reached before crashing: " + lastPrime);
    System.out.println("Runtime: " + runtime + " milliseconds");
    System.out.println("Custom Score: " + customScore);
}
```

Custom Scoring Formula:

The custom scoring formula can be adjusted to suit the specific requirements and objectives of the benchmark. In this example, the custom score is calculated as the number of primes generated divided by the runtime.

Example Usage:

The user can choose an option from the menu to run the desired benchmark. The results, including runtime and custom score, will be displayed on the screen or logged to a file, depending on the chosen option.

Timișoara
May, 2024

```
Benchmark Menu:
1. Single Benchmark
2. Multiple Benchmarks
3. Benchmark with File Logging
4. Benchmark with Timer Pause and Resume
5. Run Benchmark
6. Test Timer with Main Thread Interruption
7. Run Benchmark with Time Conversion
8. Run Benchmark with MonteCarlo
9. Generate primes
0. Exit
Enter your choice:
```

```
Enter your choice: 8
n = 50: 7500 nanoseconds
n = 100: 7800 nanoseconds
n = 1000: 64100 nanoseconds
n = 10000: 665500 nanoseconds
n = 50000: 3755900 nanoseconds
n = 100000: 6498500 nanoseconds
```

Timișoara
May, 2024

Conclusion

This benchmarking project provides a versatile framework for measuring CPU performance using different computational tasks. The flexible logging and timing functionalities make it easy to customize and extend the benchmarks according to specific needs.

We are hoping this documentation provides enough insight into the project as well as the project itself to be a good representation of our skill and we are hopeful for a passing grade (5) if that is possible :).

Timișoara
May, 2024