Structura Sistemelor de Calcul

Program de testare a parametrilor de performanta ai unui PC

Autori: Rus Ionel

Grupa: 30234

Facultatea de Automatica si Calculatoare

1. Cuprins

1. Cuprins 2
2. Introducere 3
3. Studiu bibliografic 3
4. Analiza si design 3
5. Concluzii 7
6. Bibliografie 7

2. Introducere

Acest benchmark ofera o modalitate de a masura timpul necesar pentru a efectua hash pe un set de stringuri generate random, analizand performanta lor într-un context specific, respectiv de a afisa date despre componentele hardware ale sistemului.

Obiective:

- ***Evaluarea performantei***: Benchmark-ul are scopul de a masura performanta algoritmului de hashing, oferind o perspectiva asupra timpului necesar pentru procesarea unui set de 1000 de șiruri aleatoare.
- ***Hardware***: Prin utilizarea bibliotecii **oshi**, benchmark-ul da specificatiile hardware ale sistemului, precum CPU, GPU, RAM.
- **Îmbunătățirea performanței**: Rezultatele obtinute vor putea fi utilizate pentru a identifica posibile puncte de optimizare sau imbunatatire a performantei in cadrul operaiiilor de hashing.

3. Studiu bibliografic

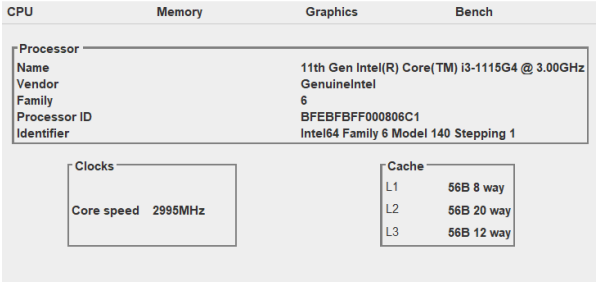
In dezvoltarea acestui benchmark, am explorat domeniul benchmark-urilor de hashing, cu accent pe implementarile in limbajul Java. Am analizat diversele abordari si tehnologii utilizate pentru a evalua eficienta algoritmilor de hashing si pentru a obtine informatii despre hardware.

4. Analiza si design

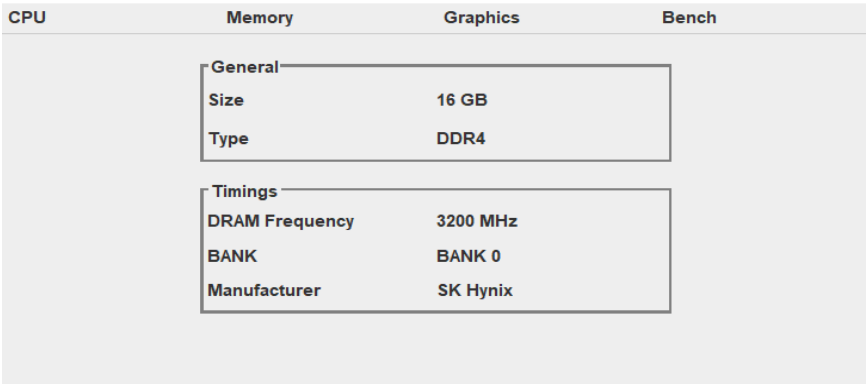
Interfata Grafica:

Interfata grafica este structurata intr-un meniu principal cu trei optiuni distincte: Processor (CPU), Memory (RAM) si Graphic (GPU). Aceasta organizare permite utilizatorului sa exploreze detaliile specifice fiecarei componente hardware a sistemului.

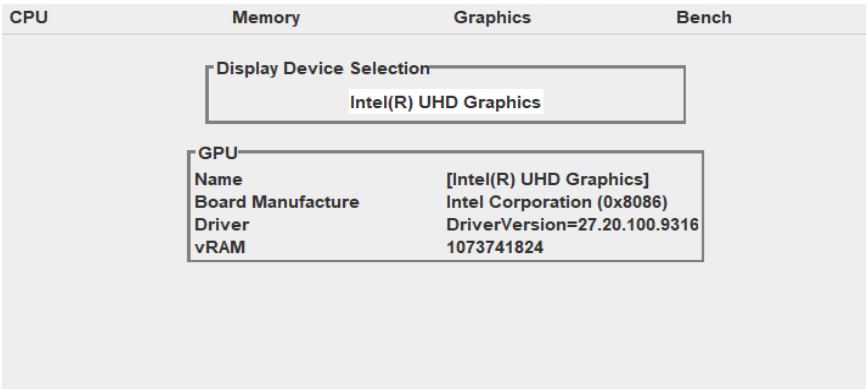
- **Panoul CPU**: La deschiderea aplicatiei, se afiseaza automat panoul CPU, care furnizeaza detalii esentiale despre procesorul sistemului. Aceste informatii includ vendorul, modelul, familia, frecventa si alte caracteristici relevante ale procesorului.



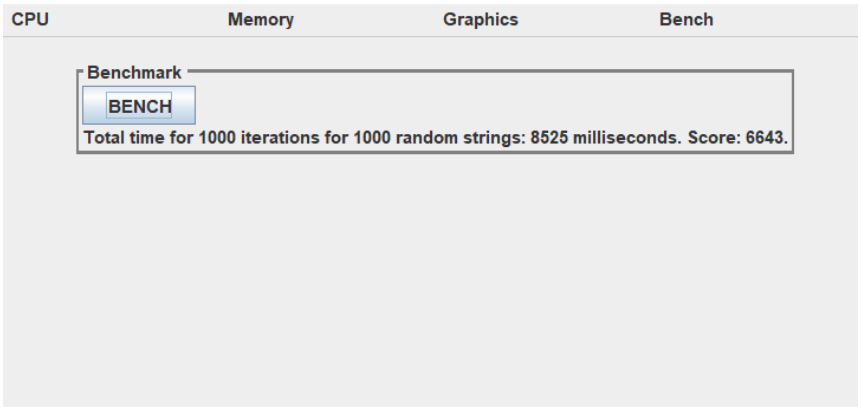
- **Panoul RAM**: Al doilea tab din meniu este dedicat memoriei RAM. Similar cu panoul CPU, acesta prezinta utilizatorului informatii despre dimensiunea, frecventa, tipul si producatorul memoriei RAM, contribuind la o intelegere comprehensiva a performantelor acestei componente.



- **Panoul GPU**: Al treilea panou este destinat unitatii de procesare grafica (GPU). Aici, utilizatorul poate examina detalii precum numele GPU-ului, producatorul placii grafice, versiunea driver-ului si cantitatea de memorie video (VRAM).



- **Panoul de Benchmark**: Programul ruleaza un hash de 1000 de ori pe un set de 1000 de siruri generate aleatoriu. Aceasta operatie ofera o masura a performantei sistemului in ceea ce priveste algoritmii de hashing, contribuind la evaluarea eficientei acestora intr-un mediu Java.



Implementare:

- Arhitectura proiectului este de tip MVC, modelul fiind realizat cu ajutorul bibliotecii OSHI, iar view-ul este realizat in Java Swing.

```
public final class GPU {
    3 usages
    private static List<String> name;
    2 usages
    private static String boardManufacture;
    2 usages
    private static String driver;
    2 usages
    private static Long vRAM;
    1 usage  unknown
    public GPU(){
        SystemInfo systemInfo = new SystemInfo();
        HardwareAbstractionLayer hardware = systemInfo.getHardware();
        List<GraphicsCard> graphicsCard = hardware.getGraphicsCards();
        name = new ArrayList<>();
        for(GraphicsCard graphicCard: graphicsCard)
            name.add(graphicCard.getName());
        boardManufacture = graphicsCard.get(0).getVendor();
        driver = graphicsCard.get(0).getVersionInfo();
        vRAM = graphicsCard.get(0).getVRam();
    }
```

- Clasa GPU primeste de la variabila systemInfo detaliile necesare pentru a le putea afisa. Clasele RAM, respective Processor sunt implementate la fel.

```
public class View extends JFrame {
    11 usages
    public static CPUJPanel cpuJPanel;
    7 usages
    public static MemoryJPanel memoryJPanel;
    7 usages
    public static GraphicsJPanel graphicsJPanel;
    4 usages
    public static BenchJPanel benchJPanel;
    9 usages
    public static JPanel contentJPanel;
    4 usages
    private static CardLayout cardLayout;
    3 usages
    private static JMenuItem cpuJMenuItem;
    3 usages
    private static JMenuItem memoryJMenuItem;
    3 usages
    private static JMenuItem graphicsJMenuItem;
    3 usages
    private static JMenuItem benchJMenuItem;
```

- Pentru partea de View ne folosim de implementare pentru fiecare panou separat unde le initializam si adaugam logica pentru butoane.

```java
public final class Bench {

    1 usage  unknown
    public static String run() {
        int numStrings = 1000;
        int numIterations = 1000;
        String[] randomStrings = generateRandomStrings(numStrings);
        long totalHashTime = benchmark(() -> {
            for (int i = 0; i < numIterations; i++) {
                for (String input : randomStrings) {
                    try {
                        hashSHA256(input);
                    } catch (NoSuchAlgorithmException e) {
                        throw new RuntimeException(e);
                    }
                }
            }
        });
        int score = calculateScore(totalHashTime, numIterations, numStrings);
        return "Total time for " + numIterations + " iterations for " +
                numStrings + " random strings: " + totalHashTime + " milliseconds. Score: " + score + ".";
    }
```

- Clasa Bench contine mai multe metode printre care metoda run cu ajutorul careia facem operatia si primim rezultatul in urma operatiilor facute.

```java
    1 usage  unknown
    private static String bytesToHex(byte[] bytes) {
        StringBuilder hexString = new StringBuilder();
        for (byte b : bytes) {
            hexString.append(String.format("%02x", b));
        }
        return hexString.toString();
    }

    1 usage  unknown
    public static void hashSHA256(String input) throws NoSuchAlgorithmException {
        MessageDigest md = MessageDigest.getInstance( algorithm: "SHA-256");
        byte[] hashBytes = md.digest(input.getBytes());
        StringBuilder hexString = new StringBuilder();
        for (byte hashByte : hashBytes) {
            hexString.append(String.format("%02x", hashByte));
        }
    }
    1 usage  unknown
    public static int calculateScore(long totalTime, int numIterations, int numStrings) {
        double weightedTime = 0.75 * totalTime;
        double weightedIterations = 0.1 * numIterations;
        double weightedStrings = 0.15 * numStrings;

        double totalWeightedScore = weightedTime + weightedIterations + weightedStrings;
        return (int) totalWeightedScore;
    }
```

- Metoda de hashSHA256 a fost aleasa pentru ca e una dintre cele mai bune metode de tip SHA oferind rezultate bune intr-un timp relativ mic. Metoda de calculateScore este implementata folosind o formula de suma ponderata in functie de numarul de iteratii, numarul de stringuri si de timpul total cat a durat operatia.

5. Concluzii

Prin integrarea unei interfete grafice, programul nu doar furnizeaza informatii utile despre componentele hardware ale sistemului, ci si ofera un instrument practic de testare a performantelor acestuia.

6. Bibliografie

1. https://en.wikipedia.org/wiki/SHA-2
2. https://simplesolution.dev/java-get-cpu-information-oshi-library/
3. https://www.javatpoint.com/java-swing
4. https://www.geeksforgeeks.org/introduction-to-java-swing/
5. https://www.simplilearn.com/tutorials/cyber-security-tutorial/sha-256-algorithm