**Cerinte**

**Laborator 1 (PPD)**

**(saptamana 1,2)**

Implementati in Java

A) adunarea a doua matrice de dimensiune (nxm)

B) inmultirea a doua matrice de dimensiune (nxk), respectiv (k,m) folosind multithreading.

-Numarul de threduri p trebuie sa fie un parametru care poate fi citit (modificat) inainte de inceperea executiei.

Datele de intrare corespunzatoare elementelor matricilor *se vor citi din fisiere* (care au fost anterior create folosind generare aleatoare de numere)!

Programul va afisa la sfarsit timpul global/total T de executie corespunzator operatiei de adunare(respective operatiei de inmultire) de matrice.

(T = timp\_*evaluat*\_ dupa\_terminarea\_executiei\_fiecarui\_thread – timp\_evaluat\_inainte\_de pornirea\_threadurilor)

Important: folositi o incarcare echilibrata de calcul pe fiecare thread.

*Balanced Distribution* (dimensiunea datelor pe care lucreaza fiecare thread este aproximativ

egala).

**Testarea** va include:

-variante orientate pe testarea corectitudinii (dimensiuni relativ mici; de exemplu < 10x10)

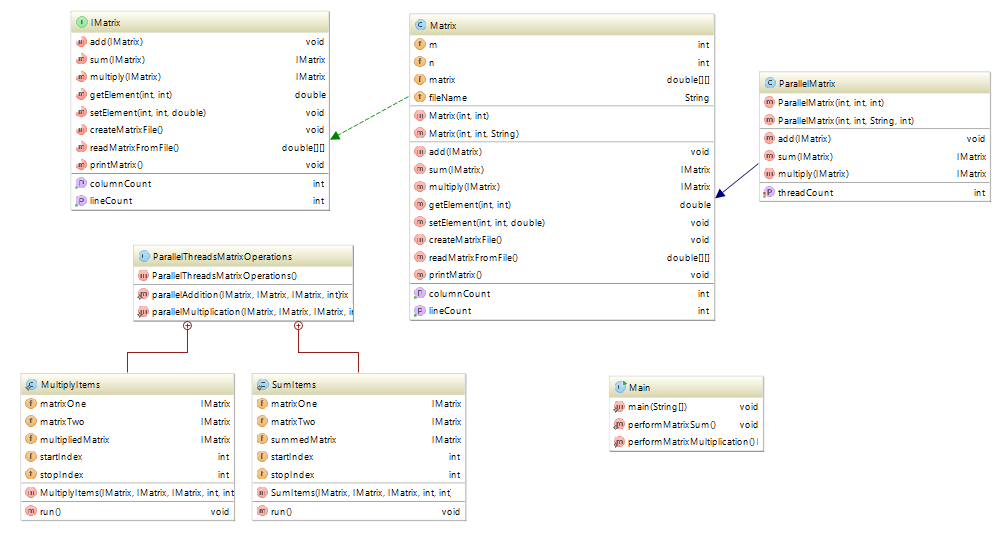
-variante orientate pe masurarea performantei (dimensiuni relativ mari; de exemplu > 1000x1000);

Pentru fiecare testare includeti in documentatie informatii despre:

**Dim\_matrice | Nr\_threaduri | Timp de executie | Obs- referitoare la sistemul pe care s-a facut testarea**

**Proiectare**

**Clase, functii si relatiile dintre ele: (diagrama de clase)**



IMatrix – interfata matricelor.

Matrix – implementarea matricelor si a operatiilor pe matrice (fara paralelism).

ParallelMartrix – implementarea matricelor si a operatiilor pe matrice (cu paralelism).

ParallelThreadMatrixOperations – implementarea operatiilor pe matrice cu threaduri paralele.

MultiplyItems – Thread care calculeaza inmultirea unui element dintr-o matrice.

SumItems – Thread care calculeaza suma unui element dintr-o matrice.

Main – clasa principala si punctul de intrare in program.

**Detalii de implementare**

Am folosit o abordare cat mai obiectuala si abstracta, astfel ca:

1. In Main.peformMatrixSum(), respectiv in Main.performMatrixMultiplication() se construiesc 2 matrice.
2. In constructorii matricelor se apeleaza functiile de creare de fisier si citire din el (sau se pot construi si matrice goale).
3. Se apeleaza functia de adaugare/inmultire de matrice, adica: add(IMatrix altaMatrice), respectiv multiply(IMatrix altaMatrice), care la randul lor apeleaza functia corespunzatoare din clasa statica ParallelThreadsMatrixOperations.
4. Clasa ParallelThreadsMatrixOperations este responsabila sa imparta operatiile pe thread-uri intr-un mod cat mai echilibrat.
5. Am considerat matricele ca vectori, si astfel am redus impartirea operatiilor din matrice la o impartire a operatiilor din vectori, si anume, fiecare thread ia n \* m / p operatii, iar daca n \* m nu este divizibil cu p, primele thread-uri iau cate o operatie in plus. Din aceasta cauza, diferenta maxima dintre numarul operatiilor unui thread oarecare si altul este 1 (poate fi si 0 cand n \* m este divizibil cu p).

Alte observatii:

1. La rulare, Main.peformMatrixSum(), respectiv in Main.performMatrixMultiplication() vor afisa atat valoarea initiala a matricelor, cat si valoarea finala.
2. Diferenta dintre add() si sum() este ca add() modifica matricea apelanta, iar sum() nu o modifica, ci ca si multiply() returneaza o noua matrice.

**Detalii de sistem**

Laptop: Acer Aspire V5 573G

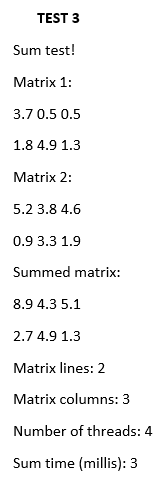
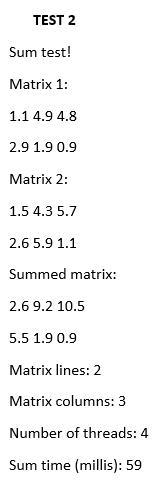
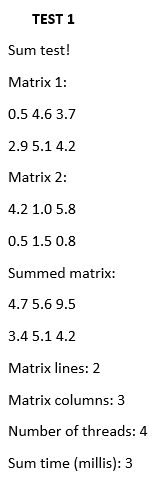
Procesor: Inter Core i5-4200U 1.6 GHz 2.3GHz

Capacitate RAM: 8GB DDR 3

Unitate de stocare: OCZ Vertex 2.0 200GB (SSD)

**Cazuri de testare**

1. **Testarea corectitudinii:**



1. **Testarea performantei:**

**TEST 1:**

Matrix lines: 1000 | Matrix columns: 1000 | Number of threads: 4 | Sum time (millis): 11

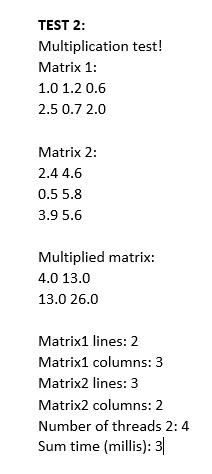
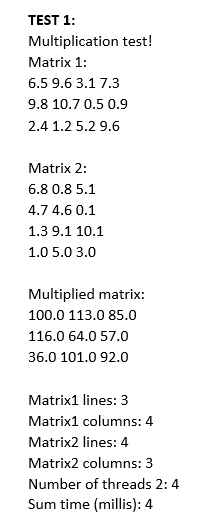
**TEST 2:**

Matrix lines: 1000 | Matrix columns: 1000 | Number of threads: 8 | Sum time (millis): 12

**TEST 3:**

Matrix lines: 2000 | Matrix columns: 2000 | Number of threads: 4 | Sum time (millis): 3

1. **Testarea corectitudinii:**



**B) Testarea performantei:**

**TEST 1:**

Matrix1 lines: 1000 | Matrix1 columns: 1000

Matrix2 lines: 1000 | Matrix2 columns: 1000

Number of threads: 4

Multiplication time (millis): 9252

**TEST 2:**

Matrix1 lines: 1000 | Matrix1 columns: 1500

Matrix2 lines: 1500 | Matrix2 columns: 1000

Number of threads: 4

Multiplication time (millis): 15226