**Laborator 2**

**Cerința:**

**Laborator 2**

Se considera problema de la laboratorul 1 cu urmatoarea modificare:

Se cere un program care sa asigure urmatoarea postconditie:

**Postconditie:** Matricea initiala contine imaginea filtrata.

**Constrangere:** NU se aloca o matrice rezultat (V) temporara!

**Obiectiv**: optimizarea complexitatii-spatiu in conditiile obtinerii unei performante ridicate.

**Datele de intrare** se citesc dintr-un fisier de intrare “date.txt”.

(Fisierul trebuie creat anterior prin adaugare de numere generate aleator.)

**Implementare**

1. Java
2. C++ ( cel putin C++11 )

**Testare:** masurati timpul de executie pentru

1. N=M=10 si n=m=3; p=2;
2. N=M=1000 si n=m=5; p=1,2,4,8,16
3. N=10 M=10000 si n=m=5; p=1,2,4,8,16
4. N=10000 M=10 si n=m=5; p=1,2,4,8,16

**ObservatII (valabile pentru fiecare caz de testare 1-4):**

* Fiecare test trebuie repetat de 10 ori si evaluarea timpul de executie se considera media aritmetica a celor 10 rulari.
* Pentru fiecare varianta de testare (dintre cele 10) folositi acelasi fisier “date.txt”;
* Pentru fiecare varianta de testare (dintre cele 10) verificati corectitudinea prin comparatia rezultatului cu fisierul rezultat prin executia secventiala.

**Analiza:**

Verificati corectitudinea prin comparatie cu rezultatul de la laboratorul 1.

Comparati performanta pentru fiecare caz !

Comparati timpii obtinuti cu implementarea Java versus implementarea C++.

Evaluati complexitatea-spatiu.

**Deadline:**

* saptamana 5

**Proiectare:**

Pentru matrice si kernel am folosit vectori bidimensionali de tip **int**.

Împărțirea pe thread-uri am făcut-o pe linii, astfel încât diferența dintre două thread-uri să fie de maxim o linie.

Calculul unei valori se face in felul următor: se parcurge fiecare element din kernel, se află cel mai apropiat element de acesta și se înmulțește valoarea cea mai apropiată cu valoarea din kernel și se adună la sumă.

**Java:** Programul are o clasă **Main,** care conține partea de citire, scriere și verificare corectitudine, dar și abordarea paralelă. În clasa **Main** se află o clasă statică **MyThread**, care extinde clasa **Thread** din Java. Aceasta reprezintă worker-ul care efectuează calculele. De asemenea, conține o clasă **Utils** pe care am folosit-o pentru a genera matricile de numere random.

Pentru partea de bariere, am folosit clasa **CyclicBarrier**. Fiecare thread modifică inițial în buffer, apoi așteaptă la barieră. După ce se termină așteptarea, fiecare thread își scrie rezultatul în matricea inițială.

**C++:** Avem două fișiere: **main.cpp** și **barrier.h**, Fișierul **main.cpp** conține mai multe metode, pentru citire, scriere, verificare, dar și abordarea paralelă. Fișierul **barrier.h** conține implementarea barierei furnizată in secțiunea Files de pe Teams, acolo fiind implementată clasa **Barrier**, pe care o folosesc în cod similar cu modul în care o folosesc și în Java.

Complexitatea de timp a algoritmului este: **O(N \* M \* n \* m).**

Complexitatea de spațiu a algoritmului este: **O(N \* M + n \* m).**

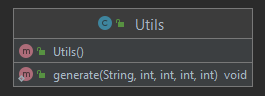
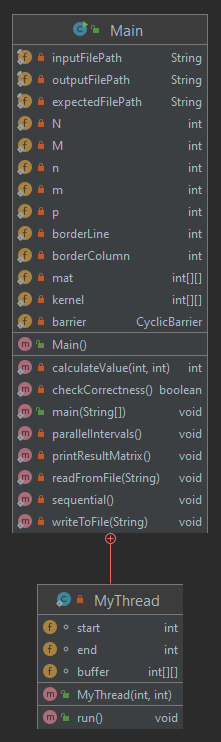
Diagramele claselor sunt la finalul documentului.

Cazurile de testare sunt cele din enunțul problemei, la care am mai adăugat și scenariul cu 6 thread-uri.

Rezultatele testelor **(Java)** sunt următoarele:

|  |  |  |
| --- | --- | --- |
| Tip Matrice | Nr threads | Timp executie |
| N = M = 10; n = m = 3 | 1 | 0.73966 |
|  | 2 | 1.57595 |
|  | 4 | 1.63609 |
|  | 6 | 1.74352 |
|  | 8 | 1.93403 |
|  | 16 | 2.36424 |
| N = M = 1000; n = m = 5 | 1 | 51.62853 |
|  | 2 | 43.30108 |
|  | 4 | 54.66694 |
|  | 6 | 62.99935 |
|  | 8 | 76.26361 |
|  | 16 | 221.8959 |
| N = 10 M = 10000; n = m = 5 | 1 | 27.1988 |
|  | 2 | 29.719 |
|  | 4 | 37.34364 |
|  | 6 | 39.33526 |
|  | 8 | 35.86967 |
|  | 16 | 36.81636 |
| N = 10000 M = 10; n = m = 5 | 1 | 15.82188 |
|  | 2 | 23.22823 |
|  | 4 | 35.56709 |
|  | 6 | 39.40758 |
|  | 8 | 39.56108 |
|  | 16 | 44.28758 |

Diagramele claselor **(Java)**:

****

Rezultate testelor **(C++)** sunt următoarele:

|  |  |  |  |
| --- | --- | --- | --- |
| Tip Matrice | Tip alocare | Nr threads | Timp executie |
| N = M = 10; n = m = 3 | dinamic | 1 | 0.49282 |
|  |  | 2 | 0.57022 |
|  |  | 4 | 0.80962 |
|  |  | 6 | 0.83618 |
|  |  | 8 | 0.97916 |
|  |  | 16 | 1.51559 |
| N = M = 1000; n = m = 5 | dinamic | 1 | 62.8071 |
|  |  | 2 | 34.40324 |
|  |  | 4 | 24.64055 |
|  |  | 6 | 20.93335 |
|  |  | 8 | 23.81939 |
|  |  | 16 | 32.35107 |
| N = 10 M = 10000; n = m = 5 | dinamic | 1 | 6.08314 |
|  |  | 2 | 3.71375 |
|  |  | 4 | 3.09723 |
|  |  | 6 | 3.21396 |
|  |  | 8 | 3.41372 |
|  |  | 16 | 4.62029 |
| N = 10000 M = 10; n = m = 5 | dinamic | 1 | 8.36745 |
|  |  | 2 | 5.17489 |
|  |  | 4 | 4.04452 |
|  |  | 6 | 4.40245 |
|  |  | 8 | 5.19589 |
|  |  | 16 | 8.32993 |

Diagramele claselor **(C++)**:

