**Laborator 1 - PPD**

De Miros Razvan-Andrei

Grupa 234, semigrupa 2

**Analiza Cerintelor**

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, letter

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Matricea initiala este reprezentata de numere cuprinse intre 1 si 100 inclusiv. In urma aplicarii filtrarii, fiecare pixel/pozitie din matrice va fi actualizata cu o noua valoare calculata dupa formula:

.

In termini mai simpli, ne vom imagina kernelul (matrice 3x3 sau 5x5) ca o tabla pe care o vom aseza astfel: elemental central (OBS. Se accepta doar kernel de size impar KxK!!) se suprapune peste fiecare element din matricea initiala. Elementul peste care se suprapune este pixelul a carui valoare trebuie sa o actualizam dupa formula de calcul.

Se observa ca incepand cu elementul de pe pozitia [1,1] , din matricea initiala ( in cazul kernelului de 3x3 sau [2,2] in cazul kernelului de 5x5) pana la pozitia [N-1][M-1] ( pt 3x3) sau [N-2][M-2] (pt 5x5) nu avem probleme in efectuarea calcului, dar pentru restul liniilor si coloanelor vom avea nevoie de o “bordare” a matricei pentru a ne oferi suportul complet al “tablei” noastre. Bordarea o vom face cu valorile adiacente.

Proiectare:

1. Citirea matricilor si a kernelului se face din fisiere .txt unde pe prima linie avem dimensiunile matricei, pe linia a doua dimensiunea kernelului, iar fiecare din urmatoarele linii reprezinta o linie din matricea initiala.

Graphical user interface, application

Description automatically generated A screenshot of a computer

Description automatically generated with medium confidence

1. Urmatorul pas este de a “borda matricile”:

***JAVA/ C++ Dinamic:***

* Declaram initial o matrice mai mare (**dar conform dimensiunilor citite din fisier**), corespunzatoare cazului in care ne aflam(dimensiunea kernelului):
* Text

  Description automatically generated
* Apoi vom citi valorile din fisier in pozitiile potrivite

**C++ Static:**

* Declaram global, o matrice cu dimensiunile maxime posibile pentru a satisfice toate cazurile de testare:
* **Text

  Description automatically generated**

In toate cazurile, ulterior ne vom folosi de o functie “borderMatrix” care pe liniile adiacente marginilor matrici initiale vor dubla/ copia valorile respective. (evident corespunzator cerintei noastre, vom borda fie o singura data in cazul 3x3 sau de doua ori in cazul matricei 5x5)

Graphical user interface, text

Description automatically generated

A picture containing text, electronics, keyboard

Description automatically generated

borderMatrix()

A picture containing text, electronics, keyboard

Description automatically generated

1. In functie de modul de lucru ( secvential sau cu threaduri) vom efectua filtrarea cu ajutorul functiei transformPixel()
2. Pentru lucrul cu **threaduri:**

* Threadurile vor lucre fie pe linii, fie pe coloane in functie de care este mai mare
* Am folosit distributia liniara astfel incat fiecare thread va lucra pe un anumit numar de linii sau coloane, egal impartite, iar in cazul restului se atribui inca o linie sau coloanal fiecarui thread incepand de la primul.

1. Fiecare thread va actualiza liniile corespunzatoare din matricea output.
2. La final va rezulta o noua matrice cu valorile actualizate (output\_matrix).

**Diagrama de clase in Java**

**A picture containing text

Description automatically generated**

Analiza performantei

JAVA:

|  |  |  |
| --- | --- | --- |
| Tip matrice | Nr threads | Timp executie(ms) |
| N = M = 10  n = m = 3 | secvential | 0.47272 |
| 4 | 0.53523 |
| N = M = 1000  n = m = 5 | secvential | 48.61968 |
| 2 | 34.96906 |
| 4 | 32.79928 |
| 8 | 30.45426 |
| 16 | 32.93828 |
| N = 10 , M = 10000  n = m = 5 | secvential | 12.02507 |
| 2 | 14.62907 |
| 4 | 11.31074 |
| 8 | 11.644 |
| 16 | 12.5077 |
| N = 10000, M = 10  n = m = 5 | Secvential | 11.79074 |
| 2 | 12.73752 |
| 4 | 16.28701 |
| 8 | 13.83672 |
| 16 | 13.8779 |
|  |  |  |

C++ Static:

|  |  |  |
| --- | --- | --- |
| Tip matrice | Nr threads | Timp executie(ms) |
| N = M = 10  n = m = 3 | secvential | 1.83119 |
| 4 | 2.05354 |
| N = M = 1000  n = m = 5 | secvential | 102.96567 |
| 2 | 61.8447 |
| 4 | 36.98935 |
| 8 | 25.97224 |
| 16 | 103.67557 |
| N = 10 , M = 10000  n = m = 5 | secvential | 8.10994 |
| 2 | 5.3755 |
| 4 | 4.00378 |
| 8 | 3.76358 |
| 16 | 8.24423 |
| N = 10000, M = 10  n = m = 5 | secvential | 24.98055 |
| 2 | 16.9756 |
| 4 | 10.62716 |
| 8 | 8.99255 |
| 16 | 27.27193 |

C++ Dinamic:

|  |  |  |
| --- | --- | --- |
| Tip matrice | Nr threads | Timp executie(ms) |
| N = M = 10  n = m = 3 | secvential | 0.02003 |
| 4 | 2.07691 |
| N = M = 1000  n = m = 5 | secvential | 104.0959 |
| 2 | 57.293 |
| 4 | 40.1137 |
| 8 | 31.10104 |
| 16 | 105.8426 |
| N = 10 , M = 10000  n = m = 5 | secvential | 22.48544 |
| 2 | 15.90106 |
| 4 | 22.96242 |
| 8 | 22.87301 |
| 16 | 24.13443 |
| N = 10000, M = 10  n = m = 5 | secvential | 10.99647 |
| 2 | 8.22878 |
| 4 | 5.804 |
| 8 | 5.754 |
| 16 | 13.6975 |

Am observant ca la 8 threaduri avem rezultate mult mai bune in c++ static/dynamic fata de java.

Dar in rest, codul java pare mai efficient.

Trecand la 16 threaduri timpul creste enorm de mult.