Documentatie Laborator 1

**Analiza cerintelor:**

Se considera o imagine reprezentataprintr-o matrice de pixeli,F, de dimensiune(MxN).

Se cere transformarea ei aplicand o filtrare cu o fereastra definita de multimea de indici

Ind[n,m] = {(k,l) | -n/2<=k<=n/2, -m/2<=l<=m/2}

si de coeficientii *wkl*

n,m imparesi n<N, m<M.

Transformarea unui pixel:

V[i,j] = {**+** (k,l): -n/2<=k<=n/2, -m/2<=l<=m/2:w[k,l] \* F[i-k,j-l]} unde

De exemplu:

multimea de indicieste

Ind [3,3]={ (-1,-1), (-1,0), (-1,1), (0,-1), (0,0), (-0,1), (1,-1), (1,0), (1,1)}

Si ponderile asociate



Actualizarea unui pixel de pe pozitia (i,j)

v[i,j] =

f[i,j]\*1/9+

f[i-1,j]\*1/9+

f[i,j-1]\*1/9+

f[i-1,j-1]\*1/9+

f[i+1,j]\*1/9+

f[i,j+1]\*1/9+

f[i+1,j+1]\*1/9

Pentru frontiere se considera ca un element este egal cu elementul din celula vecina din matrice  
f[-1,-1]= f[0,0]; f[-1,j]= f[0,j]; f[i,-1]=f[i,0]; f[M,j]= f[M-1,j]; f[i,N]=f[i,N-1];

Exemplificare ->https://de.wikipedia.org/wiki/Datei:2D\_Convolution\_Animation.gif

Se cere asigurarea urmatoarei postconditii:

Postconditie: Matricea rezultat V contine imaginea filtrate a imaginii initiale F (undeV != F)

1. Program secvential
2. Program paralel: folositi**p** threaduripentrucalcul.

**Obiectiv:**Impartire cat maiechilibrata si eficienta a calculul pe threaduri!

Pentru impartirea sarcinilor de calcul (taskuri) se foloseste descompunere geometrica care poate fi  
 (puteti alege o variant sau sa incercati mai multe si sa o identificati pe cea mai buna):

* Pe orizontala (mai multe linii allocate unui thread)
* Pe verticala (mai multe coloane allocate unui thread)
* Bloc – submatrici allocate unui thread
* bazat pe o functie de distributie prin care unui index al unui thread i se distribuie o submultime de indecsi din matrice;

distributia se poate face prin:

- distributie liniara (indicni alaturati la acelasi thread) sau

- distributie ciclica( cu pas egal cu p)

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

(Fisierul trebuie creat anterior prin adaugare de numere generate aleator. Toate rularile trebuie executate cu acelasi fisier.)

**Implementare:**

1. Java
2. C++ ( cel putin C++11 )
   1. matricile sunt alocate static (int f[MAX][MAX] )
   2. matricile sunt allocate dynamic (new…)

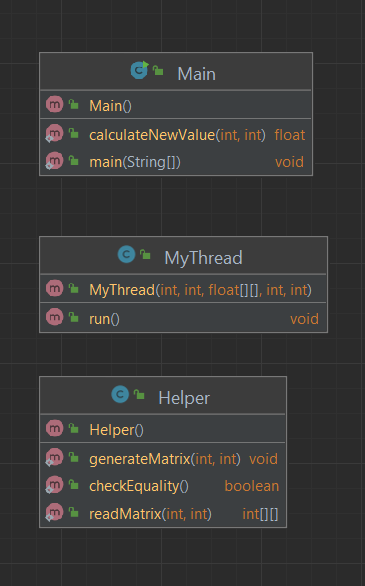
Folosire directa a threadurilor(creare explicita) => nu se permite folosirea executorilor.

**Testare:** masurati timpul de executie pentru

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

**Proiectare:**

**Java:**

****

**Main() -** clasa de intrare in program

**Main(String[]):void** – functia principala a programului

**calculateNewValue(int, int) :float** – functie care calculeaza noua valoare din matrice, folosind matricea de filtrare. Se inmultesc pozitiile din kernel cu cele din matrice corespunzatoare.

**MyThread(int,int,float[][],int,int)** – constructorul pentru clasa MyThread, care primeste un id, un numar p de thread-uri, matricea rezultat V, si dimensiunile matricei initiale, M si N

**run()** – functie suprascrisa a clasei Thread, in care fiecare thread calculeaza valorile din matricea rezultat V pe baza matricei initiale F si a kernelului W. Fiecare thread calculeaza linii intregi din matrice, din p in p linii.

**generateMatrix(int,int): void** – functie pentru generarea unei matrici random, de dimensiune MxN

**checkEquality():Boolean** - functie pentru verificarea egalitatii fisierelor: secvential si parallel

**readMatrix(int,int): int[][]** – functie pentru citirea matricii initiale din fisier

**C++:**

|  |
| --- |
| +main(int, char\*): int |
| +readMatrix(int, int): vector<vector<int>> |
| +calculateNewValue(vector<vector<float>>, vector<vector<int>>, int, int): float |
| +calculateThread(int, int, vector<vector<float>>, vector<vector<int>>, int, int): void |
| +checkEquality(): bool |

Functiile **main(int, char\*):int** **, readMatrix(int,int): vector<vector<int>> , calculateNewValue(int, int) :float,** **checkEquality():bool** sunt ca si cele din Java.

Functia calculateThread**(int, int, vector<vector<float>>, vector<vector<int>>, int, int): void** este o functie care se transmite ca si parametru unui thread, pentru a putea face calculul noilor valori.

**Detalii de implementare:**

Codul din Java si C++ este similar. Atat in Java cat si in C++, modul de lucru este la fel. Se citeste matricea din fisierul “inputMatrix.txt”, dupa care formam matricea kernel, W, in functie de dimensiunea acesteia. Am considerat ca fiecare element al matricii kernel sa fie 1/9. Matricea initiala este declarata din start mai mare cu 4 unitati iar bordarea acesteia are loc prin asignarea de valori in functie de valorile celulelor vecine.

Astfel, formule de calcul pentru calcularea colturilor bordurii, unde N=nr coloane, M=nr linii:

*F*[0][0] = *F*[0][1] = *F*[1][0] = *F*[1][1] = *F*[2][2];  
*F*[*M* + 3][1] = *F*[*M* + 2][1] = *F*[*M* + 3][0] = *F*[*M* + 2][0] = *F*[*M* + 1][2];  
*F*[0][*N* + 3] = *F*[0][*N* + 2] = *F*[1][*N* + 3] = *F*[1][*N* + 2] = *F*[2][*N* + 1];  
*F*[*M* + 3][*N* + 3] = *F*[*M* + 3][*N* + 2] = *F*[*M* + 2][*N* + 3] = *F*[*M* + 2][*N* + 2] = *F*[*M* + 1][*N* + 1];

Si pentru spatiul dintre colturi:

for (int i = 2; i < *M* + 2; i++) {  
 *F*[i][0] = *F*[i][1] = *F*[i][2];  
 *F*[i][*N* + 2] = *F*[i][*N* + 3] = *F*[i][*N* + 1];  
}

for (int j = 2; j < *N* + 2; j++) {  
 *F*[0][j] = *F*[1][j] = *F*[2][j];  
 *F*[*M* + 2][j] = *F*[*M* + 3][j] = *F*[*M* + 1][j];  
}

Mai departe, in functie de optiunea utilizatorului, se poate face un calcul secvential sau paralel, care utilizeaza aceeasi functie de calcul a noii valori: calculateNewValue, care face suma inmultirii matricii kernel, W, cu pozitiile corespunzatoare din matricea initiala, F. In cazul in care se opteaza pentru varianta paralela, avem un vector de p thread-uri, in care fiecare thread calculeaza noile valori ale unei linii, din p in p linii. Astfel, thread-urile sunt distribuite liniar.

Rezultatele obtinute in urma calculelor sunt scrise in fisierul de output: “outputMatrix.txt”. De asemenea, o functie de checkEquality verifica daca rezultatele obtinute in mod paralel si secvential sunt aceleasi.

**Cazuri de testare:**

**Java:**

|  |  |  |
| --- | --- | --- |
| Tip Matrice | Nr threads | Timp executie |
| N=M=10 n=m=3 | secvential | 25.45868 |
|  | 4 | 2.5422 |
| N=M=1000 n=m=5 | secvential | 360.5636 |
|  | 2 | 92.9018 |
|  | 4 | 86.26889 |
|  | 8 | 81.67527 |
|  | 16 | 91.10594 |
| N=10 M=10000 n=m=5 | secvential | 165.4507 |
|  | 2 | 28.36873 |
|  | 4 | 27.50546 |
|  | 8 | 31.35387 |
|  | 16 | 31.18269 |
| N=10000 M=10 n=m=5 | secvential | 156.6644 |
|  | 2 | 29.34556 |
|  | 4 | 29.59581 |
|  | 8 | 30.45581 |
|  | 16 | 30.67056 |

**C++:**

|  |  |  |  |
| --- | --- | --- | --- |
| Tip Matrice | Tip alocare | Nr threads | Timp executie |
| N=M=10 n=m=3 | dinamica | 4 | 7.76594 |
| N=M=1000 n=m=5 |  | 2 | 1632.943 |
|  |  | 4 | 1369.449 |
|  |  | 8 | 1431.524 |
|  |  | 16 | 1501.96 |
| N=10 M=10000 N=M=5 | dinamica | 2 | 172.5148 |
|  |  | 4 | 175.6676 |
|  |  | 8 | 171.5698 |
|  |  | 16 | 172.7792 |
| N=10000 M=10 N=M=5 | dinamica | 2 | 183.4525 |
|  |  | 4 | 173.5954 |
|  |  | 8 | 167.4312 |
|  |  | 16 | 166.9398 |

**Analiza rezultatelor:**

**Java: secvential-paralel:** se poate observa ca performanta este mai scazuta in cazul calculului secvential, iar in cazul calculului parallel, cu cat creste numarul de thread-uri, cu atat creste si performanta. La matricile mai mari, absolute ca timpul va fi putin mai ridicat.

**C++: dynamic:** in cazul matricei de mici dimensiuni, calculul este rapid. Cu cat dimensiunea creste, timpul creste. Nu putem spune ca avem imbunatatire foarte mare, doar pe alocuri, in unele cazuri. Acest timp este probabil influentat si de utilizarea librariei <vector>, utilizand vector<vector<>>.

**Java comparat cu C++:** in Java, dupa cate putem observa, performanta este mult mai ridicata, comparata cu rezultatele obtinute din C++.