**Laborator 1**

**Cerința:**

**Laborator 1**

**Deadline:** saptamana 4

Se considera o imagine reprezentata printr-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 impare si 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] }

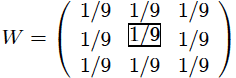
unde

De exemplu:

multimea de indici este

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 elemental 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 filtrata a imaginii initiale F (unde V != F)

1. Program secvential
2. Program paralel: folositi **p** threaduri pentru calcul.

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

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

* Pe orizontala (mai multe linii alocate unui thread)
* Pe verticala (mai multe coloane alocate unui thread)
* Bloc – submatrici alocate 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 (indici alaturati la acelasi thread) sau

- distributie ciclica( cu pas egal cu p).

**Datele de intrare** se citesc dintr-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 alocate 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=10000 si n=m=5; p=2,4,8,16
4. N=10000 M=10 si n=m=5; p=2,4,8,16

Rezultatele acestor teste trebuie sa fie reflectate in documentatie in tabele

**Java:**

|  |  |  |
| --- | --- | --- |
| **Tip matrice** | **Nr threads** | **Timp executie** |
| N=M=10  n=m=3 | secvential | …. |
| 4 | …. |
| N=M=1000  n=m=5 | secvential | …. |
| 1 | …. |
| 2 | …. |
| 4 | …. |
| 8 | …. |
| 16 | …. |
| …. | …. | …. |

**C++**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tip matrice** | **Tip alocare** | **Nr threads** | **Timp executie** |
| N=M=10  n=m=3 | Static | 4 | …. |
| dinamic | 4 | …. |
| N=M=1000  n=m=5 | static | 1 | …. |
| 2 | …. |
| 4 | …. |
| 8 | …. |
| 16 | …. |
| dinamic | 1 | …. |
| 2 | …. |
| 4 | …. |
| 8 | …. |
| 16 | …. |
| …. | …. | …. | …. |

**ObservatII:**

* Fiecare test trebuie repetat de 10 ori si pentru evaluarea timpul de executie se considera media aritmetica a celor 10 rulari.
* Pentru fiecare varianta (secvential, paralele) folositi acelasi fisier “date.txt”;

Folositi recomandarile din fisierele “Testare” “Verificare corectitudii”.

**Analiza**

Comparati performanta pentru fiecare caz – secvential versus paralel si variantele paralele intre ele.

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

Comparati cele doua variante pentru implementarea C++.

Analiza trebuie evidentiata in documentatie.

**Proiectare:**

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

Împărțirea pe thread-uri am făcut-o pe linii/coloane, în funcție de numărul de linii/coloane (am împărțit pe linii dacă numărul de linii era mai mare decât cel de coloane, sau pe coloane dacă nu). Împărțirea este făcută astfel încât diferența dintre două thread-uri să fie de maxim o linie / coloană.

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 cele două abordări, cea secvențială și cea 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.

**C++:** Avem două fișiere diferite, main.cpp și main\_static.cpp, care conține aceeași soluție, implementată dinamic și static. Fișierele conțin mai multe metode, pentru citire, scriere, verificare, dar și cele două abordări.

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

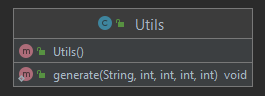
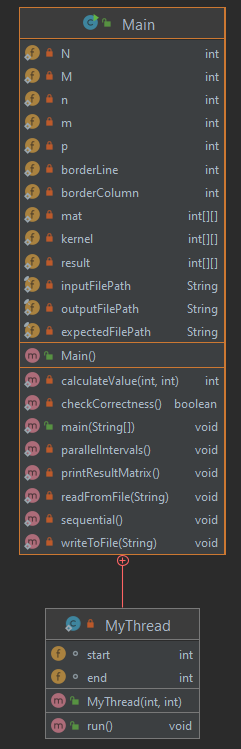
Diagramele claselor sunt la finalul documentului.

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 | 0.07885 |
|  | 2 | 0.6098 |
|  | 4 | 0.79276 |
|  | 6 | 0.86505 |
|  | 8 | 1.0068 |
|  | 16 | 1.57407 |
| N = M = 1000; n = m = 5 | 1 | 44.23523 |
|  | 2 | 43.35478 |
|  | 4 | 39.62737 |
|  | 6 | 40.89574 |
|  | 8 | 74.85999 |
|  | 16 | 249.1119 |
| N = 10 M = 10000; n = m = 5 | 1 | 27.63486 |
|  | 2 | 33.27821 |
|  | 4 | 33.68265 |
|  | 6 | 39.5362 |
|  | 8 | 41.73558 |
|  | 16 | 58.74662 |
| N = 10000 M = 10; n = m = 5 | 1 | 13.54338 |
|  | 2 | 34.90139 |
|  | 4 | 32.03345 |
|  | 6 | 39.46621 |
|  | 8 | 39.74142 |
|  | 16 | 41.4084 |

Diagramele claselor **(Java)**:

****

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

|  |  |  |  |
| --- | --- | --- | --- |
| Tip Matrice | Tip alocare | Nr threads | Timp executie |
| N = M = 10; n = m = 3 | dinamic | 1 | 0.00313 |
|  |  | 2 | 0.86478 |
|  |  | 4 | 0.83164 |
|  |  | 6 | 0.90702 |
|  |  | 8 | 1.03011 |
|  |  | 16 | 1.554 |
| N = M = 1000; n = m = 5 | dinamic | 1 | 69.79027 |
|  |  | 2 | 36.60117 |
|  |  | 4 | 22.37964 |
|  |  | 6 | 18.3162 |
|  |  | 8 | 15.00962 |
|  |  | 16 | 14.39628 |
| N = 10 M = 10000; n = m = 5 | dinamic | 1 | 6.35657 |
|  |  | 2 | 4.07301 |
|  |  | 4 | 2.73676 |
|  |  | 6 | 2.83634 |
|  |  | 8 | 2.35604 |
|  |  | 16 | 2.54552 |
| N = 10000 M = 10; n = m = 5 | dinamic | 1 | 7.36028 |
|  |  | 2 | 4.63406 |
|  |  | 4 | 2.91301 |
|  |  | 6 | 2.79403 |
|  |  | 8 | 2.50961 |
|  |  | 16 | 2.83789 |
|  |  |  |  |
| Tip Matrice | Tip alocare | Nr threads | Timp executie |
| N = M = 10; n = m = 3 | static | 1 | 0.01242 |
|  |  | 2 | 0.65784 |
|  |  | 4 | 0.89328 |
|  |  | 6 | 0.95532 |
|  |  | 8 | 1.11586 |
|  |  | 16 | 1.58431 |
| N = M = 1000; n = m = 5 | static | 1 | 64.2342 |
|  |  | 2 | 33.75596 |
|  |  | 4 | 20.15835 |
|  |  | 6 | 16.66539 |
|  |  | 8 | 14.40651 |
|  |  | 16 | 14.59874 |
| N = 10 M = 10000; n = m = 5 | static | 1 | 5.78011 |
|  |  | 2 | 4.04755 |
|  |  | 4 | 2.782 |
|  |  | 6 | 2.52432 |
|  |  | 8 | 2.32763 |
|  |  | 16 | 2.31451 |
| N = 10000 M = 10; n = m = 5 | static | 1 | 18.30303 |
|  |  | 2 | 12.53215 |
|  |  | 4 | 8.94475 |
|  |  | 6 | 7.15368 |
|  |  | 8 | 6.08594 |
|  |  | 16 | 7.07028 |
|  |  |  |  |

Diagramele metodelor **(C++, varianta statică și dinamică)**:

