# Rezultate tema2

Enunț

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)

A) Program secvential

B) 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.)

**Constrangere:** nu se va aloca o matrice separata pentru rezultat. Calculele se efectueaza pe matrica initiala.

Implementare:

a) Java

b) C++ ( cel putin C++11 )

i. matricile sunt alocate static (int f[MAX][MAX] )

ii. matricile sunt alocate dynamic (new…)

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

Testare: masurati timpul de executie pentru

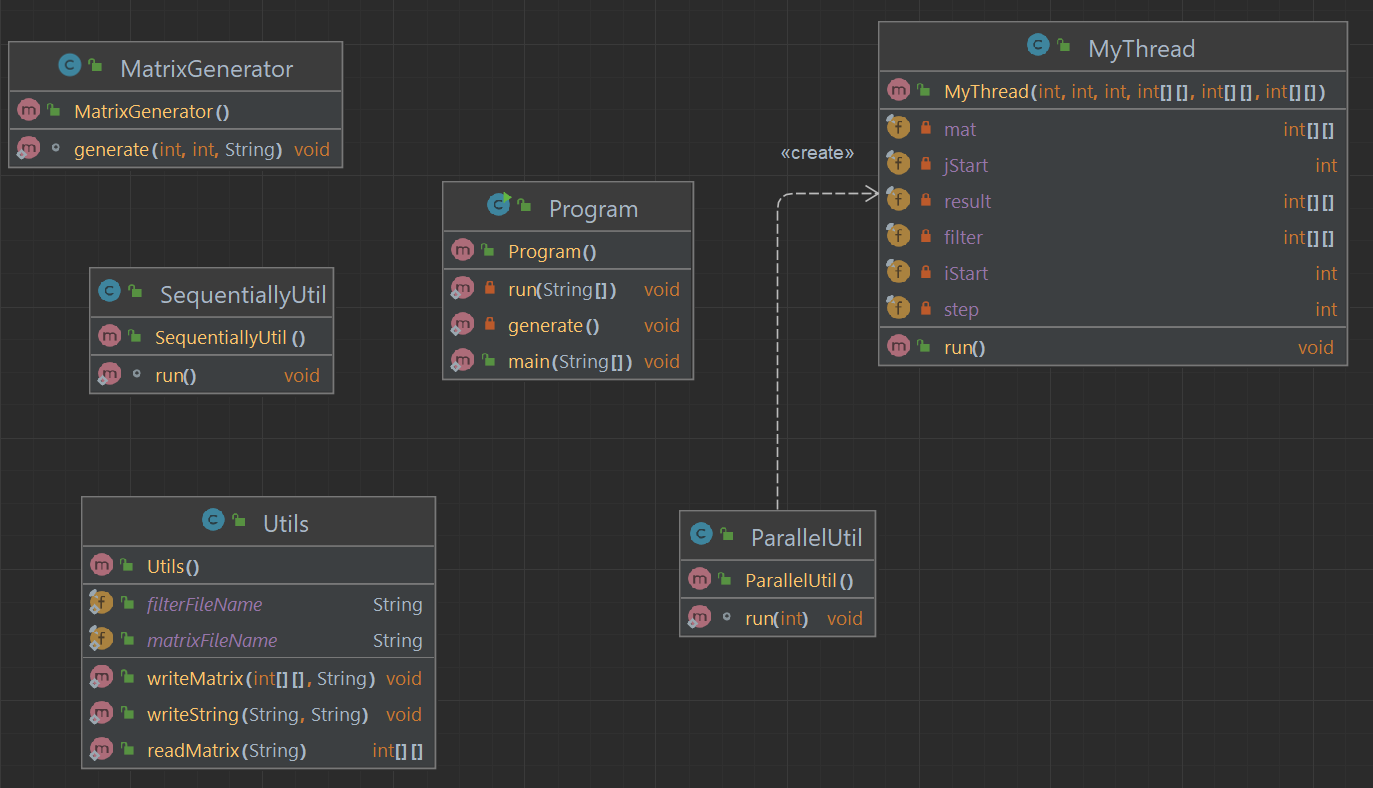
1) N=M=10 si n=m=3; p=4;

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

Java



C++

|  |
| --- |
| Program |
| N: Int  M: Int  n: Int  p: Int  mat: Int\*\*  filter: Int\*\* |
| runSequentially(): void  runParallel(): void  worker(int iStart, int jStart, int count): void |

Partiționarea datelor

* Matricea rămâne cum este citită din fișier si fiecare thread primește un iStart, jStart și count (cate elemente pornind de pe pozitia [iStart][jStart] trebuie sa calculeze;

count = (N \* M) / p +- 1)

* Thread-ul copiaza mai intai toate elementele de care va avea nevoie intr-un array temporar, iar apoi asteapta la bariera
* Dupa ce toate thread-urile ajung la bariera, fiecare isi calculeaza portiunea lui, folosind datele din tmp
* Thread-ul calculează valoarea elementului de pe poziția iStart jStart și după trece la urmatorul (jStart++); dacă jStart depășește numărul de coloane, iStart crește cu o unitate iar jStart scade cu M(numărul de coloane)

j = j++;

if (j >= M) {

j = j - M;

i++;

}

* Astfel, fiecare thread calculeaza o zonă continuă
* Implementările în Java și C++ folosesc aceeași idee, structurată diferit

Java

|  |  |  |
| --- | --- | --- |
| **Tip matrice** | **Nr threads** | **Timp executie** |
| N = M = 10  n = m = 3 | secvential | 42.84411 |
| 4 | 47.07114 |
| N = M =1000  n = m = 5 | secvential | 622.83517 |
| 2 | 655.99512 |
| 4 | 664.28031 |
| 8 | 634.66826 |
| 16 | 669.3908 |
| N = 10  M = 10000  n = m = 5 | secvential | 186.69734 |
| 2 | 230.30273 |
| 4 | 240.88178 |
| 8 | 254.49794 |
| 16 | 270.14039 |
| N = 10000  M = 10  n = m = 5 | secvential | 181.0287 |
| 2 | 195.265223 |
| 4 | 188.85799 |
| 8 | 213.20127 |
| 16 | 249.64197 |

C++

|  |  |  |  |
| --- | --- | --- | --- |
| **Tip matrice** | **Tip alocare** | **Nr threads** | **Timp executie** |
| N = M = 10  n = m = 3 | static | 1 | 0.13078 |
| 4 | 1.02616 |
| dinamic | 1 | 0.12777 |
| 4 | 1.14534 |
| N = M =1000  n = m = 5 | static | 1 | 521.2619 |
| 2 | 484.8437 |
| 4 | 486.5957 |
| 8 | 486.393 |
| 16 | 498.7358 |
| dinamic | 1 | 507.7722 |
| 2 | 529.9027 |
| 4 | 516.1243 |
| 8 | 504.913 |
| 16 | 501.9867 |
| N = 10  M = 10000  n = m = 5 | static | 1 | 52.74223 |
| 2 | 48.72486 |
| 4 | 49.15213 |
| 8 | 49.23801 |
| 16 | 50.10617 |
| dinamic | 1 | 51.53995 |
| 2 | 55.4827 |
| 4 | 54.01011 |
| 8 | 53.02424 |
| 16 | 53.41939 |
| N = 10000  M = 10  n = m = 5 | static | 1 | 55.55428 |
| 2 | 50.42991 |
| 4 | 50.19962 |
| 8 | 51.36288 |
| 16 | 51.35965 |
| dinamic | 1 | 53.71191 |
| 2 | 55.50094 |
| 4 | 53.35682 |
| 8 | 54.90689 |
| 16 | 54.77289 |

Concluzii

* Implementarea in C++ este mult mai rapida decat cea in java
* Surprinzator, cele mai bune rezultate s-au obtinut cu alocare statica, mai putin in cazul cu date putine
* In implementarea in java, se vede ca cel mai bun rezultat a fost in cazul secvential, indiferent de dimensiunea datelor