**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 alta matrice rezultat si nici o matrice temporara!

Se pot folosi/aloca doar vectori temporari pentru care complexitatea spatiu se incadreaza in O(n).

**Atentie**  si implementarea secventiala trebuie adaptata la constrangerea specificata.   
Va fi nevoie sa se foloseasca vectori auxiliari temporari.

**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.)

**Rezultatul** se salveaza intr-un fisier **output.txt**

**Implementare**

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

**Fata de Laboratorul 1 se cere doar:**

**-Distributie pe linii**

**-Kernel de dimensiune k=3**

**-Pentru C++ alocare dinamica**

**Testare:** masurati timpul de executie pentru

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

**ObservatII:**

* Fiecare test trebuie repetat de 10 ori si evaluarea timpul de executie se considera media aritmetica a celor 10 rulari.
* Pentru fiecare executie (cele 10) a fiecarui caz de testare folositi acelasi fisier de intrare “date.txt” si verificati corectitudinea prin comparatie cu fisierul output obtinut la executia secventiala.

**Analiza:**

Comparati performanta pentru fiecare caz !

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

Evaluati complexitatea-spatiu

**Documentarea performantei**

Pentru fiecare dintre variantele de testare rezultate din diferitele implementari si din diferitele date de test analizati performata folosind tabele similare celor evidentiate mai jos.

Aceste tabele trebuie adaugate in documentatie!

**Java:**

|  |  |  |
| --- | --- | --- |
| **Tip matrice** | **Nr threads** | **Timp executie** |
| N=M=10  n=m=3 | secvential | 6.95537 |
| 2 | 8.96431 |
| N=M= 1000  n=m=3 | secvential | 110.54435 |
|  | 2 | 139.95523 |
|  | 4 | 138.61497 |
|  | 8 | 181.40419 |
|  | 16 | 215.48423 |
| N=10 000 M= 10 000  n=m=3 | secvential | 4570.17781 |
|  | 2 | 4391.9312 |
|  | 4 | 4857.67184 |
|  | 8 | 3651.90907 |
|  | 16 | 4105.30934 |

**C++**

|  |  |  |
| --- | --- | --- |
| **Tip matrice** | **Nr threads** | **Timp executie** |
| N=M=10  n=m=3 | seq | 0.01092 |
| 2 | 0.56811 |
| N=M=1000  n=m=3 | seq | 38.39764 |
| 2 | 22.19391 |
| 4 | 13.58617 |
| 8 | 8.289 |
| 16 | 7.23161 |
|
|
|
|
|
| N= 10000 M=10000  n=m=3 | seq |  |
| 2 |  |
| 4 |  |
| 8 |  |
| 16 |  |
|
|
|
|
|
|

Analize rezultate:  
  
- impactul numar threaduri asupra timpului de executie – imbunatateste in mare parte, dar la un numar prea mare de threaduri supraincarca.

* Imbunatatirea se observa in principal pe matricile de marimi mari
* Pt dimensiuni mici, paralelizarea nu este avantajoasa
* 4-8 threaduri e mai avantajos
* C++ mai performant ca Java