Documentație Tema 1

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];

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.

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

**Implementare:**

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

Pentru impartirea sarcinilor de calcul (taskuri) am create un nou array liniar care continue matricea initiala. Apoi, am impartit restul ramas impartirii lui N la numarul de thread-uri, la fiecare thread pana cand nu mai ramane nimic. Astfel, pentru fiecare thread am alocat un interval pe care sa-l calculeze. In executia fiecarui thread m-am folosit de urmatoarele formule pentru a ajunge de la indexul vectorului liniar, la indecsii matricii initiale:

Avem matricea initiala de dimensiune MxN, Indexul vectorului liniar k, Indexul matricii i, j

k = i \* m + j

i= k / N

j= k % N

Am verificat de asemenea in functia rulata de fiecare thread, cazul in care se iasa din matrice, caz in care am folosit valoarea vecina, valorii din afara matricii.

Detalii de implementare in Java:

Graphical user interface

Description automatically generated

Am clasa abstracta Worker, care continue matricile: matrix, kernel si result, functiile de citire a matricii din fisier, functia de scriere in fisier, si functia abstracta doOperation, implementata de clasele Sequential si Parallel.

Clasa Sequential implementeaza doOperation secvential, cu for-uri pana la N si M, pentru a parcurge matricea si for-uri de – n / 2 pana la n / 2 si - m / 2 pana la m / 2 pentru a trece prin matricea de kernel.

Clasa Parallel, de asemenea implementeaza doOperation, care foloseste clasa MyThread. MyThread contine functia run care va fi rulata de fiecare thread pentru obtinerea rezultatului dorit. Calculul pe thread-uri este la fel in ambele limbaje si este scris mai sus.

Clasa Main contine functia care genereaza o matrice random conform dimensiunilor, functia de comparare de fisiere folosita si de programul c++. writeToFile scrie in fisierul date.txt, matricele care vor fi citite de cele 2 programe.

Detalii de implementare in C++

Text

Description automatically generated

Functiile read\_from\_file citesc matricile din fisier, aloca/dezaloca memoria pentru cazul dinamic.

Functiile run, run\_dynamic sunt functiile pe care thread-urile le vor rula.

Functiile do\_parallel sunt functiile care vor crea thread-uri si vor imparti sarcinile.

Functiile do\_sequential sunt functiile care fac calculul secvential.

**Testare:**

**Java:**

|  |  |  |
| --- | --- | --- |
| **Tip matrice** | **Nr threads** | **Timp executie** |
| N=M=10  n=m=3 | secvential | 32.20286 |
| 4 | 33.37371 |
| N=M=1000  n=m=5 | secvential | 665.66991 |
| 1 | 707.51813 |
| 2 | 667.69857 |
| 4 | 658.07118 |
| 8 | 652.31758 |
| 16 | 663.36653 |
| N= 10 M=10000  n=m=5 | secvential | 167.65731 |
| 1 | 169.53474 |
| 2 | 169.64738 |
| 4 | 173.31514 |
| 8 | 172.6847 |
| 16 | 195.13113 |

|  |  |  |
| --- | --- | --- |
| N= 10000 M=10  n=m=5 | secvential | 183.17693 |
| 1 | 191.34692 |
| 2 | 189.28867 |
| 4 | 189.64357 |
| 8 | 191.75515 |
| 16 | 203.02251 |

**C++**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tip matrice** | **Tip alocare** | **Nr threads** | **Timp executie** |
| N=M=10  n=m=3 | Static | 4 | 9.756944444 |
| dinamic | 4 | 8.696666667 |
| N=M=1000  n=m=5 | static | 1 | 419.7337 |
| 2 | 392.087 |
| 4 | 392.9267 |
| 8 | 370.8495 |
| 16 | 391.9041 |
| dinamic | 1 | 417.2012 |
| 2 | 388.5995 |
| 4 | 386.879 |
| 8 | 380.5617 |
| 16 | 379.1648 |
| N= 10 M=10000  n=m=5 | static | 1 | 83.47073333 |
| 2 | 74.98911 |
| 4 | 72.18135 |
| 8 | 66.41283333 |
| 16 | 70.58706667 |
| dinamic | 1 | 50.09214 |
| 2 | 50.55802 |
| 4 | 49.6025 |
| 8 | 48.83321 |
| 16 | 48.75508889 |
| N= 10000 M=10  n=m=5 | static | 1 | 51.08728 |
| 2 | 47.27556 |
| 4 | 48.24347 |
| 8 | 46.97077 |
| 16 | 46.34081111 |
| dinamic | 1 | 50.18035 |
| 2 | 48.25988 |
| 4 | 47.21749 |
| 8 | 46.95934 |
| 16 | 47.6254 |

Analiza testarii

* La dimeniuni mici de date, variantele secventiale sunt mai rapide decat cele paralele
* La cele paralele se observa o crestere a performantei cand programele sunt rulate la 8 sau 16 thread-uri
* Aproape de fiecare data programul C++ e mai performant decat cel Java
* Pe exemplele rulate personal nu se oberva o diferenta majora intre programele rulate static si cele rulate dinamic
* Programul c++ a fost rulat cu Release x64