**Laborator 1: Aplicarea unei matrici de convolutie**

**Timp de lucru: 2 saptamani (deadline – saptamana 4)**

Cerinta e stiuta, mai mentionam doar paddingul “virtual”.

TEMA:

Considerand ca se da o matrice V(n,m) si o matrice de convolutie C(k,k) se cere sa se calculeze matricea V(n,m) rezultata in urma aplicarii convolutiei cu matricea de convolutie C pe matricea 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 va folosi descompunere geometrica cu urmatoarele 2 variante:

* Pe orizontala (mai multe linii alocate unui thread)

si

* Pe verticala (mai multe coloane alocate unui thread)

Pentru urmatoarele variante se acorda suplimentar cate 2 puncte la nota finala.

* 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;

testarea se va face definind functia de distributie astfel incat sa obtinem:

- 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

**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 | …. 0.9ms |
| 4 | …. 4.1ms |
| N=M=1000  n=m=5 | secvential | …. 52.4ms |
| 1 | …. 58.4ms |
| 2 | …. 92.9ms |
| 4 | …. 122.4ms |
| 8 | …. 176ms |
| 16 | …. 266ms |
| N=10, M=10000  n=m=5 | Secvential | …. 30ms |
| 1 | … 35ms |
| 2 | … 51.8ms |
| 4 | … 93.4ms |
| 8 | … 114.5ms |
| 16 | … 141.1ms |
| N=10000,M=10  n=m=5 | Secvential | … 10ms |
| 1 | … 13.9ms |
| 2 | … 41.7ms |
| 4 | … 87.1ms |
| 8 | … 132ms |
| 16 | … 117.3ms |

**C++**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tip matrice** | **Tip alocare** | **Nr threads** | **Timp executie** |
| N=M=10  n=m=3 | Static | 4 | …. 0.1573ms |
| dinamic | 4 | …. 0.0226ms |
| N=M=1000  n=m=5 | static | 1 | …. 182.131ms |
| 2 | …. 100.985ms |
| 4 | …. 66.0249ms |
| 8 | …. 63.7568ms |
| 16 | …. 59.3127ms |
| dinamic | 1 | …. 185.741ms |
| 2 | …. 117.717ms |
| 4 | …. 62.8146ms |
| 8 | …. 67.4907ms |
| 16 | …. 64.169ms |
| N=10, M=10000  n=m=5 | static | 1 | …. 18.7329ms |
| 2 | …9.4114ms |
| 4 | … 6.77218ms |
| 8 | …8.67282ms |
| 16 | …6.28122ms |
| dinamic | 1 | …18.6027ms |
| 2 | …10.2087ms |
| 4 | …8.03455ms |
| 8 | …7.70883ms |
| 16 | …6.6668ms |
| N=10000,M=10  n=m=5 | static | 1 | …18.9219ms |
| 2 | …10.6997ms |
| 4 | …7.41834ms |
| 8 | …7.01891ms |
| 16 | …6.19408ms |
| dinamic | 1 | …19.8592ms |
| 2 | …10.0396ms |
| 4 | …7.40112ms |
| 8 | …6.68141ms |
| 16 | …5.74967ms |

**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” si “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.

**Analiza**

* Java beneficiaza de multithreading doar pentru calculi mai lungi, threadurile trebuie “incalzite”, trebuie rulate o vreme pana sa inceapa optimizarea la nivel de jvm
* C++ se obisnuieste mai repede cu threadurile, nu face la fel de multa optimizare in spate deci se misca mai bine de la inceput; rezultatul e ca mai multe threaduri = timp mai mic de executie, pe cand la Java overheadul de creare threaduri a facut timpul sa creasca
* Accesul in memorie se face liniar pe cat posibil pentru eficienta, salturile cresc timpii de rulare
* Java timp mai mare, C++ timp mai mic in situatia mea ; am masurat partea de executie, nu si de alocare memorie