**Documentatie – Laborator 2**

Run option:

0 – secvential:

* Se parcurge matricea secvential linie cu linie, coloana cu coloana.

1 – impartire pe linii

* Fiecare thread primeste initial un numar de linii consecutive pe care le va actualiza egal cu n / p. Daca aceasta impartire are un rest, atunci primele n % p threaduri vor primi o linie in plus.

**Testare:**

**Testare Java:**

|  |  |  |
| --- | --- | --- |
| Tip Matrice | Nr threads | Timp executie |
| 10x10 | secvential | 0.019 |
| 2 | 1.538 |
| 4 | 1.447 |
| 8 | 1.633 |
| 16 | 1.846 |
| 1000x1000 | secvential | 9.230 |
| 2 | 20.586 |
| 4 | 25.673 |
| 8 | 29.830 |
| 16 | 25.084 |
| 10000x10000 | secvential | 559.964 |
| 2 | 284.762 |
| 4 | 178.548 |
| 8 | 143.907 |
| 16 | 161.275 |

**Testare C++:**

|  |  |  |
| --- | --- | --- |
| Tip Matrice | Nr threads | Timp executie |
| 10x10 | secvential | 0.001 |
| 2 | 0.300 |
| 4 | 0.368 |
| 8 | 0.517 |
| 16 | 0.744 |
| 1000x1000 | secvential | 2.172 |
| 2 | 1.962 |
| 4 | 1.723 |
| 8 | 2.007 |
| 16 | 2.412 |
| 10000x10000 | secvential | 214.778 |
| 2 | 117.427 |
| 4 | 80.957 |
| 8 | 64.549 |
| 16 | 64.521 |

**Analiza:**

*Testarile s-au realizat pe un CPU Intel I3 12100F 4 cores 8 threads.*

* **Java -** secvential versus paralel

In Java se poate observa ca rularea secventiala este mult mai buna ca timp comparativ cu rularile paralele pentru cazurile in care dimensiunile matricii sunt reduse, insa semnificativ mai slaba pentru cazul matricilor cu multe elemente. O cauza principala ar putea fi modul in care Java creaza threadurile, fiind nevoie de o comunicare intre masina virtuala Java si OS ceea ce poate adauga un delay. De asemenea, se poate observa o crestere in timp mai mult sau mai putin direct proportionala cu numarul de threaduri. Pe langa cauza precizata anterior, aceasta marire se poate datora si timpului mort in care toate threadurile asteapta ca fiecare sa isi genereze cacheul pana sa inceapa lucrul efectiv, perioada care poate fi mai mare decat timpul necesar lucrului efectiv (ceea ce poate explica si performanta mai buna in cazul matricelor cu dimensiuni foarte mari).

* **Java** – paralel versus paralel

In cazurile paralele in Java se poate observa o crestere in timp odata cu numarul de threaduri pentru cazurile matricilor mici, cauza ce ar putea fi cea explicata mai sus, justificand si performanta mai buna pentru matricea de dimensiune 10000x10000.

* **C++** – secvential versus paralel

In C++ cazurile secventiale vor avea o performanta mai mica sau extrem de similara cu cele paralele cu exceptia cazului de 10x10 unde matricea este foarte mica si implicit foarte rapid de parcurs.

* **C++** – paralel versus paralel

Pentru rularile paralele, cresterea numarului de threaduri aduce o crestere in durata de executie pentru cazurile matricilor mici, lucru ce se poate datora folosirii unei bariere pentru sincronizare in etapa initiala de creare a cacheurilor pentru fiecare thread ce duce la un timp mort pentru threadurile care au reusit sa isi genereze acest cache mai rapid. In cazul matricilor cu multe elemente in schimb, unde nivelul de munca pentru fiecare thread este cantitativ mai mare, se observa o mare crestere in performanta odata cu cresterea numarului de threaduri.

* **Java** versus **C++**

Timpii obtinuti de implementarea in C++ sunt mult mai buni comparativ cu cei obtinuti de Java in toate cazurile. Aceasta s-ar putea datora modului in care C++ lucreaza mult mai aproape cu sistemul de operare si gestioneaza mult mai bine memoria.

* **Complexitatea spatiu**

In ceea ce priveste spatiul se poate vedea o imbunatatire fata de laboratorul trecut prin modificarea directa a matricei input pentru obtinerea rezultatului. Insa, pentru ca un thread sa nu suprascrie date care sunt necesare pentru calculul altor elemente, fiecare thread va avea nevoie sa salveze liniile de la frontiere (anterioara primei linii de care se ocupa si urmatoarea dupa ultima linie de care se ocupa). Pentru aceasta fiecare thread va avea nevoie de doua array-uri de lungime m. Prin urmare complexitatea spatiu va fi: O(n \* m) pentru matricea initiala si O(2 \* m) (= O(m)) pentru fiecare dintre cele p threaduri. Final:

O(n \* m + p \* m) = O(m \* (n + p))

**Diagrama de clase Java:**

