Tema 2

Java:

|  |  |  |
| --- | --- | --- |
| **Tip matrice** | **Nr threads** | **Timp execuție** |
| N=M=10  n=m=3 | secvențial | 0.38211 |
| 2 | 5.48991 |
| N=M=1000  n=m=3 | secvențial | 92.14967 |
| 2 | 77.1359801 |
| 4 | 77.6562302 |
| 8 | 90.3556202 |
| 16 | 100.9893501 |
| N=M=10000  n=m=3 | secvențial | 8533.9737 |
| 2 | 3631.97313366667 |
| 4 | 1686.6568 |
| 8 | 1177.15885 |
| 16 | 1206.1420005 |
|  |  |  |

C++:

|  |  |  |
| --- | --- | --- |
| **Tip matrice** | **Nr threads** | **Timp execuție** |
| N=M=10  n=m=3 | secvențial | 0.00726 |
| 2 | 0.72645 |
| N=M=1000  n=m=3 | secvențial | 24.30571 |
| 2 | 16.79209 |
| 4 | 11.44591 |
| 8 | 10.57182 |
| 16 | 16.43054 |
| N=M=10000  n=m=3 | secvențial | 2205.46 |
| 2 | 1822.25 |
| 4 | 3030.81666666667 |
| 8 | 2441.69 |
| 16 | 2170.70333333333 |

**Implementare**

* **Distribuirea pe linii**
  + Fiecare thread primește aproximativ N/P​ linii. În cazul în care există un rest, primele N%P thread-uri primesc câte o linie suplimentară. Pentru fiecare thread, se calculează liniile de început și sfârșit, adică intervalul de procesat. Bucla pentru iterarea pe linii folosește aceste limite, de la start la end, iar pe coloane se iterează de la 0 la M.
  + Spre deosebire de prima implementare, nu se folosește o matrice rezultat de aceeași dimensiune ca matricea inițială; în schimb, fiecare thread utilizează un buffer de 3 linii și M coloane. Astfel, complexitatea spațială rămâne O(n) în funcție de memoria folosită.
  + La început, sunt salvate „liniile de front” dinainte de linia de start și după linia de end, inclusiv cazurile speciale în care start este 0 sau end este N.
  + Se utilizează o barieră de sincronizare pentru a coordona thread-urile.
  + În continuare, bufferul de 3 linii consecutive este completat și se procesează fiecare linie, iterând de la start la end - 1. Fiecare linie calculată este actualizată direct în matricea inițială. În această buclă, după actualizarea liniei curente, bufferul este actualizat: prima linie devine a doua, a doua devine a treia, iar a treia linie este preluată din matricea inițială. Dacă se ajunge la final, se folosește linia de front salvată la început.

**Analiză Performanță**

**Pe fiecare caz**

* **Java**: La dimensiunea 10000×10000, varianta paralelă este mult mai rapidă decât cea secvențială. La dimensiuni mai mici, însă, varianta secvențială e ușor mai rapidă din cauza costului de creare și sincronizare a thread-urilor.
* **C++**: La dimensiunea 1000×1000 și mai mare, varianta paralelă este semnificativ mai rapidă decât cea secvențială, chiar și cu doar două thread-uri.
* **C++**: Performanța crește rapid odată cu numărul de thread-uri, penru cazul 1000 x 1000.
* **C++**: C++ este mai rapid decat Java, exceptand cazul 10000 x 10000, unde Java este mai rapid.

Java:

A screenshot of a computer program

Description automatically generated

C++

|  |
| --- |
| Program - Static |
| N: Int  M: Int  n: Int  p: Int  mat: Int[][]  filter: Int[][]  result: Int[][] |
| verify(string filePath) : bool  \*\*allocateMatrix(int n, int m) : int  deallocateMatrix(int \*\*matrix, int n) : void  readMatrixFromFile(const string &filePath, int \*\*matrix, int n, int m) : void  resultAtConvolute(int \*\*buffer, int j) : int |