Am analizat performanța execuției paralele în cadrul problemei de calcul al matricei de convoluție între două matrici, evitând utilizarea unei matrice auxiliare pentru stocarea datelor.

# Timpi de executie

## C++

|  |  |  |
| --- | --- | --- |
| Tip Matrice | Nr threads | Timp executie |
| 10x10 | Secvential | 0.254 |
| 2 | 1.442 |
| 1000x1000 | Secvential | 3025.081 |
| 2 | 3669.438 |
| 4 | 4742.523 |
| 8 | 5618.765 |
| 16 | 6843.675 |
| 10000x10000 | Secvential | 307465.967 |
| 2 | 204253.532 |
| 4 | 201125.789 |
| 8 | 216321.234 |
| 16 | 270031.658 |

## Java

|  |  |  |
| --- | --- | --- |
| Tip Matrice | Nr threads | Timp executie |
| 10x10 | Secvential | 0.062 |
| 2 | 1.681 |
| 1000x1000 | Secvential | 22.987 |
| 2 | 35.532 |
| 4 | 38.856 |
| 8 | 54.687 |
| 16 | 52.475 |
| 10000x10000 | Secvential | 541.865 |
| 2 | 302.138 |
| 4 | 202.627 |
| 8 | 165.652 |
| 16 | 224.792 |

# Pentru a soluționa această problemă, am optat să realizăm o distribuție pe linii, astfel încât să obținem o împărțire echilibrata a numărului de linii pe care fiecare fir de execuție trebuie să calculeze matricea de convoluție.

# Rezolvare

# Pentru a aborda această problemă, am optat pentru un algoritm care distribuie cantitatea de lucru la un număr p de fire de execuție, citit din linia de comandă. În fiecare iterație, încerc să împart cantitatea de calcul la numărul de fire de execuție într-un mod cât mai echitabil, astfel încât distribuția să fie uniformă. Mai exact, împart numărul de linii/coloane la numărul de fire de execuție și rețin și restul. Apoi, calculez două poziții de start și end, pe care le actualizez la fiecare pas al inițializării threadurilor.

# Deoarece nu putem utiliza o matrice auxiliară pentru a stoca rezultatele, am ales să copiez în mod constant într-un buffer rezultatele de la prima și ultima linie din submatricea alocată fiecărui fir de execuție (deoarece aceste linii sunt implicate în convoluția celulelor alocate altor fire de execuție) în timp. Celelalte elemente le modificăm distructiv. Acest lucru este posibil deoarece reținem în permanență linia actuală și linia precedentă.Actualizăm distructiv valorile celor două linii salvate în buffer după ce toate firele de execuție au terminat de lucrat cu acele valori (folosind o barieră de sincronizare).

# Rezultate

În C++, se observă că execuțiile secvențiale sunt mai rapide în cazul matricei de 10x10 și celei de 1000x1000, dar pentru cea de 10000x10000, paralelizarea aduce un plus de eficiență din punct de vedere temporal.

În Java, varianta secvențială este, de asemenea, mai rapidă în cazul matricelor de 10x10 și celei de 1000x1000, dar pentru matricea de 10000x10000, paralelizarea se dovedește mai eficientă.

Comparând Java cu C++, în general, Java se dovedește a fi mai eficientă în realizarea acestei operații. În cazul matricelor mai mari, C++ pare să întâmpine probleme semnificative care pot extinde timpul de execuție la valori exagerat de mari.