**Java**

|  |  |  |
| --- | --- | --- |
| **Tip Matrice** | **Nr Threads** | **Timp executie** |
| N=M=10,  n=m=3 | Secvential | 0.02792 |
| P=2 | 4.66192 |
| N=M=1000,  n=m=3 | Secvential | 28.53058 |
| 2 | 18.96246 |
| 4 | 27.75642 |
| 8 | 25.5932 |
| 16 | 34.2404 |
| N=M=10000,  n=m=3 | Secvential | 1055.3658667 |
| 2 | 314.4780667 |
| 4 | 197.6713 |
| 8 | 167.62143 |
| 16 | 154.9782 |

Obs:

* In primul caz de testare (N=M=10) a fost mai eficienta metoda secventiala, insa in celelalte 2 cazuri metoda paralela a fost considerabil mai eficienta
* Observam ca in cazul N=M=1000, cel eficient timp il obtinem folosind 8 threaduri, iar in cazul N=M=10000, cel eficient timp il obtinem folosind 16 threaduri.

**C++**

|  |  |  |
| --- | --- | --- |
| **Tip Matrice** | **Nr Threads** | **Timp executie** |
| N=M=10,  n=m=3 | Secvential | 0.0025 |
| P=2 | 6.86096667 |
| N=M=1000,  n=m=3 | Secvential | 87.9683 |
| 2 | 67.0709 |
| 4 | 50.9338 |
| 8 | 37.3541 |
| 16 | 45.4611 |
| N=M=10000,  n=m=3 | Secvential | 75489 |
| 2 | 68507.6 |
| 4 | 45826.3 |
| 8 | 41922 |
| 16 | 41761.5 |

Obs:

* La fel ca si la varianta in Java, pentru primul caz de testare a fost mai rapida metoda secventiala decat cea paralela
* Pentru celelalte cazuri a fost mai eficienta metoda care foloseste threaduri, observand ca cei mai buni timpi de executie s-au obtinut oentru 8 threaduri.
* Observam ca pentru toate cazurile de testare algoritmul scris in Java, a obtinut un timp mai mic decat cel scris in C++
* In comparatie cu primul laborator unde plasam rezultatele intr-o alta matrice, am obtinut un timp mai bun prin metoda aplicata acum, atat in Java cat si in C++.

Detalii implementare:

* Am citit matricea din fisier, apoi am bordat-o cu inca o linie si coloana, la inceput si sfarsit
* Fiecare thread calculeaza convolutia liniilor de la indicele start la end. Initial start e 1, si end primeste start + (N/p) (+1 daca N%p>0). Deoarece vrem ca rezultatul sa fie plasat in aceeasi matrice, vom memora prima data in fiecare thread, linia de la indicele start –1, end+1 si linia curenta , si ne vom folosi de o bariera pentru a le sincroniza. După ce fiecare thread a calculat convoluția pentru liniile sale, se folosește o barieră pentru a asigura că toate thread-urile s-au terminat înainte de a continua. Bariera asigură că toate thread-urile au terminat de calculat convoluția și au actualizat corect matricea înainte ca oricare dintre ele să continue.