**Documentarea performanței**

**Java .\run\_all\_scripts.ps1**

|  |  |  |
| --- | --- | --- |
| **Tip matrice** | **Nr threads** | **Timp execuție** |
| N=M=10  n=m=3 | secvențial | 0,15081 |
| 2 | 3,73534 |
| N=M=1000  n=m=3 | secvențial | 22,42328 |
| 2 | 32,30973 |
| 4 | 34,26801 |
| 8 | 39,48752 |
| 16 | 35,58383 |
| N=M=10000  n=m=3 | secvențial | 1597,28517 |
| 2 | 771,8493 |
| 4 | 424,05292 |
| 8 | 268,17987 |
| 16 | 237,36143 |

**C++ .\run\_all\_scripts.ps1**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tip matrice** | **Tip alocare** | **Nr threads** | **Timp execuție** |
| N=M=10  n=m=3 | dinamic | secvențial | 0,01018 |
| 2 | 0,44951 |
| N=M=1000  n=m=3 | dinamic | secvențial | 89,82532 |
| 2 | 15,90082 |
| 4 | 8,58016 |
| 8 | 6,34335 |
| 16 | 5,4872 |
| N=M=10000  n=m=3 | dinamic | secvențial | 8908,50323 |
| 2 | 1538,45988 |
| 4 | 776,24507 |
| 8 | 463,55248 |
| 16 | 396,94343 |

**Analiza**

* Performanța pentru fiecare caz
  + Java
    - Doar pentru cazul 10000x10000 timpul pentru secvențial este mai mare decât pentru paralel (substanțial mai mare). În celelalte, varianta secvențială este un pic mai rapidă.
  + C++
    - Aici, deja de la cazul 1000x1000 secvențial se obțin timpi mai mari. Diferența dintre secvențial și paralel este una evidentă, până și 2 thread-uri făcând timpii de aproximativ 6 ori mai buni decât în cazul secvențial.
* Timpii din Java vs. timpii din C++
  + În C++, deja de la cazul 1000x1000 observăm cum, cu creșterea numărului de thread-uri, performanța se îmbunătățește considerabil. Diferențe între 8 thread-uri și 16 nu sunt foarte mari, de vreme ce laptopul pe care s-a testat are 8 core-uri fizice și 16 logice. Acest caz aduce performanțe mai bune decât în Java, la fel ca și cel de 10x10.
  + În Java, în schimb, timpii sunt mai buni (nu se iau în calcul alocările și dealocările de memorie, care sunt realizate explicit în C++ și consumă timp). În cazul cu 10000x10000, Java se descurcă mai bine decât C++.

**Implementare**

* Pe linii
  + Se distribuie thread-urilor linii N/P și, în cazul în care există rest, primele N%P thread-uri vor primi câte o linie în plus (mai precis, se calculează liniile de start și end pentru fiecare thread). Instrucțiunile de ciclare pe linii și coloane merg cu liniile de la start la end, iar coloanele de la 0 la M.
  + De această dată, spre deosebire de primul laborator, nu se mai folosește o matrice rezultat de dimensiunea matricii inițiale, ci fiecare thread folosește câte un buffer de 3 linii și M coloane, care se încadrează în clasa de complexitate O(n) pentru spațiul de memorie utilizat.
  + Prima dată sunt salvate liniile de front (de dinainte de start și de după end, cu cazurile speciale start = 0 și end = N).
  + Se folosește o barieră de sincronizare.
  + O imagine care conține text, captură de ecran, Font, diagramă

    Descriere generată automatDupă aceea, se formează buffer-ul de 3 linii consecutive și se calculează, pe rând, într-o instrucțiune de ciclare de la start la end - 1, câte o linie ce este actualizată direct în matricea inițială. În această iterație, după actualizarea liniei curente, bufferul se modifică: prima linie devine a doua, a doua devine a treia, iar a treia ia următoarea linie din matricea inițială, sau, dacă s-a ajuns la final, se ia acea linie de front salvată la început.