**Documentarea performanței**

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

|  |  |  |
| --- | --- | --- |
| **Tip matrice** | **Nr threads** | **Timp execuție** |
| N=M=10  n=m=3 | secvențial | 0,14099 |
| 2 | 2,63652 |
| N=M=1000  n=m=3 | secvențial | 20,36893 |
| 2 | 22,41366 |
| 4 | 22,34325 |
| 8 | 28,10746 |
| 16 | 29,18894 |
| N=M=10000  n=m=3 | secvențial | 1459,07605 |
| 2 | 889,89796 |
| 4 | 534,48317 |
| 8 | 355,40912 |
| 16 | 324,61949 |

**C++ .\run\_all\_scripts.ps1**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tip matrice** | **Tip alocare** | **Nr threads** | **Timp execuție** |
| N=M=10  n=m=3 | dinamic | secvențial | 0,00948 |
| 4 | 0,38417 |
| N=M=1000  n=m=3 | dinamic | secvențial | 83,13466 |
| 2 | 42,30376 |
| 4 | 23,12095 |
| 8 | 12,93338 |
| 16 | 12,58646 |
| N=M=10000  n=m=3 | dinamic | secvențial | 8419,45201 |
| 2 | 4226,99587 |
| 4 | 2374,90004 |
| 8 | 1462,83327 |
| 16 | 1267,606 |

**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.
* 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.
  + 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). Doar pentru cazul 1000x1000 C++ reușește să întreacă performanțele din Java, ca în cazul cu 10000x10000, abia 16 thread-uri să ajungă la timpul secvențial obțiunut în Java.

**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 dimensiunea (end – start) linii și M coloane, care se încadrează în clasa de complexitate O(n) pentru spațiul de memorie utilizat.
  + În acest buffer-e sunt calculate valorile în urma aplicării convoluției.
  + Se folosește o barieră de sincronizare, la finalul calculului buffer-elor. Când toate thread-urile au ajuns la acel punct, se pot copia buffer-ele peste matricea inițială.

O imagine care conține text, captură de ecran, diagramă, linie

Descriere generată automat