**Documentatie Lab1 PPD**

**Cerinta:**

**Text, letter

Description automatically generated**

**Proiectare**

A picture containing text

Description automatically generated



**Implementare:**

Impartire pe thread-uri: dupa lini cand nr de lini > nr coloane si dupa coloane cand nr de coloane > nr de lini

Padding: in java – se adauga padding-ul la matricea initiala

In c++ - se creaza o noua matrice in care se scrie padding-ul si continutul matricii initiale

Expansionarea matrici este diferita la java si c++ insa apelarea thread-urilor si executia lor este identica.

Java:

Citesc din 2 fisiere matricea matrix si kernel-ul. La matrix creez un padding de dimensiune k/2(k=nr de lini sau coloane a kernel-ului). Apoi in functie de N si M(N=nr de lini si M nr de coloane a lui matrix) thread-urile vor lua parti diferite din matrix pe care sa lucreze. Daca N > M fiecare thread primeste aprox N / NR\_THREAD linii, iar in caz contrar fiecare thread primeste aprox M/ NR\_THREAD coloane. Apoi fiecare thread folosindu-se de kernel va opera pe o matrice noua(initial goala).

C++:

Citesc din 2 fisiere matricile matrix si kernel. Pentru crearea matricei „expandate” creez o noua matrice de dimensiuni N+(k/2)\*2 si M+(k/2)\*2.Apoi in functie de N si M(N=nr de lini si M nr de coloane a lui matrix) thread-urile vor lua parti diferite din matrix pe care sa lucreze. Daca N > M fiecare thread primeste aprox N / NR\_THREAD linii, iar in caz contrar fiecare thread primeste aprox M/ NR\_THREAD coloane. Apoi fiecare thread folosindu-se de kernel va opera pe o matrice noua(initial goala).

**Java:**

|  |  |  |
| --- | --- | --- |
| Tip Matrice | Nr. Threads | Timp Executie(milisecunde) |
| N=M=10  n=m=3 | secvential | 32.4 |
| 4 | 32.3 |
| N=M=1000  n=m=5 | secvential | 125.8 |
| 1 | 131.4 |
| 2 | 89.3 |
| 4 | 89.3 |
| 8 | 64 |
| 16 | 51.8 |
| N=10  M=10000  n=m=5 | secvential | 36.2 |
| 1 | 38.5 |
| 2 | 45.6 |
| 4 | 44.8 |
| 8 | 40.1 |
| 16 | 41.8 |
| N=10000  M=10  n=m=5 | secvential | 34.5 |
| 1 | 41.2 |
| 2 | 36.9 |
| 4 | 41.6 |
| 8 | 40 |
| 16 | 75.7 |

**C++:**

|  |  |  |  |
| --- | --- | --- | --- |
| Tip Matrice | Tip Alocare | Nr. Threads | Timp Executie(milisecunde) |
| N=M=10  n=m=3 | static | 4 | 7.8 |
| dinamic | 4 | 7.5 |
| N=M=1000  n=m=5 | static | secvential | 99.1 |
| 1 | 84.9 |
| 2 | 62 |
| 4 | 31.3 |
| 8 | 30.5 |
| 16 | 27.9 |
| dinamic | secvential | 64.5 |
| 1 | 82.2 |
| 2 | 65.1 |
| 4 | 22.8 |
| 8 | 24 |
| 16 | 25.6 |
| N=10  M=10000  n=m=5 | static | secvential | 14 |
| 1 | 27.1 |
| 2 | 17.2 |
| 4 | 13.7 |
| 8 | 16.3 |
| 16 | 21.4 |
| dinamic | secvential | 25.1 |
| 1 | 23.4 |
| 2 | 20.9 |
| 4 | 15.8 |
| 8 | 17.1 |
| 16 | 22.2 |
| N=10000  M=10  n=m=5 | dinamic | secvential | 21 |
| 1 | 24.4 |
| 2 | 15.5 |
| 4 | 12.1 |
| 8 | 13.3 |
| 16 | 14.6 |
| static | secvential | 21.6 |
| 1 | 32.5 |
| 2 | 18.1 |
| 4 | 20.2 |
| 8 | 21.6 |
| 16 | 26.7 |

C++ vs Java:

* Se poate observa ca c++ este mult mai eficient din punct de vedere al timpului
* Thread-urile in java pare ca consuma multe resurse deoarece pentru inputuri mici cu cat crestea nr de thread-uri cu atat crestea si timpul, pe cand in c++ thread-urile imbunatateau timpi chiar si la inputuri mici

Paralelism in:

1. Java

* Se observa ca numarul 8 de thread-uri are cele mai bune rezultate in majoritatea cazurile cu input mare
* In cazul inputurilor mici thread-urile in java pierd mult timp in crearea lor care nu este salvat din impartirea timpului de lucru pe o operatie scurta
* Deci de obicei la inputuri mici cu cresterea thread-urilor crestea si timpul

1. C++

* Se observa ca numerele 4,8 de thread-uri au cele mai bune rezultate in majoritatea cazurilor
* Timpul de executie scade de obicei pana la 8, iar apoi creste incet indiferent de dimensiunea inputuilor din ex.
* In comparatie cu java, in aceasta problema, in C++ numarul de thread-uri pare sa influenteze mai putin timpul de executie

Paralelism vs Secv. :

1. Java

* In java paralelismul a avut un timp mai bun ca var. Secventiala doar in cazul cand numarul operatiilor a crescut la 10^6(n=m=1000)
* in celelalte cazuri algoritmul secvential a avut un timp mai bun
* Asta are loc deoarece in cazurile input <= 10^5 nu erau destul de multe operati cat sa justifice crearea de thread-uri noi.

1. C++

* Metoda secventiala para sa fi fost mai putin eficienta din punct de vedere a timpului cand se foloseau 4 sau 8 thread-uri
* In unele cazuri folosirea a 2 thread-uri nu imbunatatea timpul

C++ static vs dinamic:

* In general urmeaza cam acelasi model de urcare scadere a timpilor in functie de nr thread-uri insa metoda dinamica in cazurile mele este un pic mai rapida de obicei