Analiza perfomantei – documentatie Lab 2

**JAVA**

Structura Codului

Codul începe cu metodele de citire și scriere a matricelor din fișiere, readMatrixFromFile și writeMatrixToFile, respectiv. Aceste metode facilitează manipularea datelor matricei.

convolutionSequential este metoda care realizează convoluția secvențială. În această metodă, matricea principala (matrix) este modificată pentru a conține rezultatul convoluției.

Compute este metoda prin care se calculeaza valoarea convolutionala a lementelor de pe fiecare linie a kernelului inmultite cu elementele din matricea initiala.

Worker este o clasă internă care extinde clasa Thread și este folosită pentru parcurgerea paralelă a matricei principale. Această clasă primește un interval de linii de procesat, practice 2 pozitii: start si end, linii pe care le va parcurge threadul respective.

convolutionParallel este metoda care realizează convoluția paralelă. Aceasta utilizează thread-uri pentru a procesa diferite părți ale matricei de intrare în paralel. Thread-urile sunt create și pornite pentru a procesa porțiuni ale matricei, iar apoi rezultatele sunt sincronizate cu ajutorul barierei.

Parcurgere secvențială

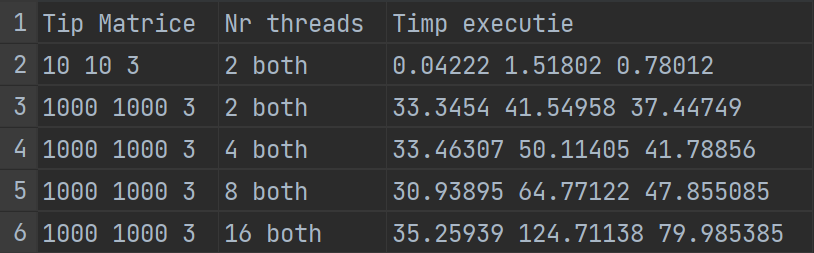
În metoda convolutionSequential, se realizează parcurgerea secvențială a matricei de intrare matrix. Această metodă utilizează o serie de buffer-uri (previousLine currentLine si buffer) pentru a stoca valorile vecine ale matricei în timpul convoluției. Buffer-ul previousLine stochează linia anterioară, currentLine reprezintă linia curentă, iar buffer va stoca rezultatul liniei curente. La finalul for-ului exterior, pentru fiecare linie, actualizam aceste buffere.

Practic pentru fiecare element de pe linia curenta, calculam cu ajutorul celor doua buffere de previous si current, iar ca sa calculam si elementele de pe linia urmatoare, folosim linia urmatoare din matricea principala, deoarece nu e modificata.

Parcurgere paralelă

În cadrul Parcurgerii paralele, fiecare thread gestionează o porțiune a matricei de intrare și reține frontierele acestei porțiuni în vectorii bufferUp și bufferDown. Această abordare este implementată astfel încât modificările asupra matricei principale, matrix, să fie efectuate doar când toate thread-urile au ajuns la barieră. Prin intermediul barierei CyclicBarrier, thread-urile așteaptă reciproc până când toate au terminat procesarea, apoi sincronizează rezultatele din vectorii bufferUp și bufferUp în matricea principală matrix. Astfel, se asigură că modificările pe matricea matrix sunt efectuate corect și eficient în mediul paralel.

Timpi de executie:



Concluzie

Acest cod ilustrează o abordare eficientă pentru a efectua operații complexe precum convoluția în mod paralel. Prin utilizarea thread-urilor și a barierei, se accelerează procesul de calcul, îmbunătățind performanța în cazul unor matrice mari. Utilizatorul poate alege între parcurgerea secvențială, cea paralelă sau ambele, în funcție de necesitățile și resursele sale. Buffer-urile previousLine, currentLine, și buffer asigură corectitudinea și eficiența algoritmului de convoluție secvențială, stocând și actualizând linii consecutive ale matricei pentru a calcula corect convoluția. Utilizarea barierei și gestionarea frontierelor în vectori asigură sincronizarea corectă și procesarea corespunzătoare a matricei în mod paralel.

**C++**

Structura codului

MyBarrier: Această clasă implementează o barieră personalizată, care permite sincronizarea thread-urilor. Ea este utilizată pentru a asigura că toate thread-urile așteaptă la barieră înainte de a continua.

MyBarrier (int count): Constructorul clasei, care primește numărul de thread-uri așteptate la barieră.

wait(): Metoda folosită pentru a aștepta la barieră. Thread-urile se blochează până când numărul specificat de thread-uri au ajuns la barieră, apoi continuă.

readMatrixFromFile: Această funcție primește doua nume de fișiere, numărul de rânduri și coloane pentru matricea principala si cea kernel si stocheaza datele in cele 2 matrici: matrix si kernel.

writeMatrixToFile: Această funcție primește o matrice, numele fișierului, numărul de rânduri și coloane și salvează matricea în fișierul dat.

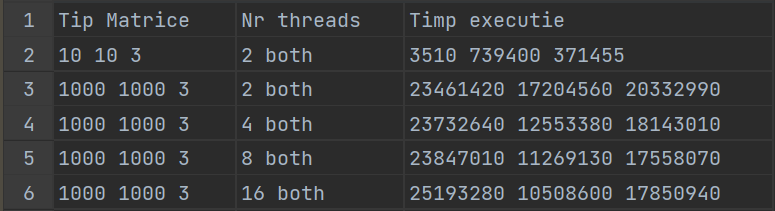
convolutionSequential: Această funcție realizează convoluția matricelor în mod secvențial. Această metodă utilizează o serie de buffer-uri (previousLine, currentLine, și buffer) pentru a stoca valorile vecine ale matricei în timpul convoluției. Buffer-ul previousLine stochează linia anterioară, currentLine reprezintă linia curentă, iar buffer corespunde liniei curente rezultate in urma convolutiei pe elemental curent. Calculul convoluției se face pentru fiecare element al matricei de ieșire, utilizând aceste buffer-uri pentru accesul la vecinii necesari.

parallelConvolution este emtoda transmisa ca parametru fiecarui thread, aceasta primeste bariera, si cele doua pozitii de start si end. Ea continue logica problemei de convolutie, la fel ca si la java.

parallelization: Această metoda initiaza convoluția matricilor folosind parcurgere paralelă. Ea primește bariera care urmeaza a fi folosita, calculeaza pozitiile de start si end pentru fiecare dintre threaduri, si le da run ca sa calculeze valorile noii matrici.

Utilizarea Codului

În funcția main, programul primește argumente de la linia de comandă pentru a specifica dimensiunile matricilor, numărul de thread-uri si metoda de rulare. Apoi, în funcție de metoda specificată, programul calculează convoluția folosind parcurgere secvențială, parcurgere paralelă sau ambele, în funcție de necesități. Matricele rezultate sunt salvate în fișiere corespunzătoare.



Concluzie

Codul oferă o implementare eficientă a convoluției matricelor cu parcurgere paralelă, utilizând thread-uri pentru a accelera procesul de calcul. Bariera personalizată (MyBarrier) asigură sincronizarea corectă a thread-urilor. Utilizatorul poate alege între diverse moduri de rulare, în funcție de nevoile sale. Această implementare permite procesarea eficientă a datelor pe matrici mari, folosind resursele hardware disponibile.

Perfomanta

Pentru matrici de dimensiuni mici, parcurgerea secvențială poate fi mai eficientă, deoarece costurile asociate cu gestionarea thread-urilor și sincronizarea pot depăși beneficiile paralelizării. În schimb, pentru matrici de dimensiuni mai mari, parcurgerea paralelă devine mai eficientă, deoarece permite exploatarea puterii de calcul a mai multor nuclee de procesor, accelerând semnificativ procesul de convoluție.

Comparatie Java vs C++

In urma testelor atat in Java, cat si in C++, am observat ca timpii de executie in urma rularii in C++ sunt mai buni decat cei in Java.

Am observant ca secventialul la java e mereu mai bun fata de parallel, insa la c++ este invers la matrici mari.