Documentatie lab 2 PPD

Analiza cerintelor

Multe dintre filtrele pe imagini utilizează operatia de convolutie bazata pe matrice de convoluție.
Se cere sa se evolueze o matrice de dimensiune NxM folosind o matrice de convolutie de dimensiune KxK.
Considerand ca se da o matrice F(N,M) si o matrice de convolutie C(K,K) se cere sa se calculeze matricea V(N,M) rezultata in urma aplicarii convolutiei cu matricea de convolutie C pe matricea F.
A) Program secvential
B) Program paralel (folosind **p** threaduri pentru calcul)
Constrangeri: Impartire cat mai echilibrata si eficienta a calculul pe threaduri (max_task_count_per_thread <= min_task_count_per_thread + 1). Nu se aloca o alta matrice rezultat si nici o matrice temporara. Se pot folosi doar vectori temporari pentru care complexitatea spatiu este liniara.

Proiectare

Ca si structuri de date folosite, avem: tablouri unidimensionale si bidimensionale alocate dinamic.
Partitionarea pe threaduri a fost facuta respectand constrangerea, adica: pentru fiecare thread a fost alocat un numar egal de task-uri; ceea ce a ramas a fost impartit egal si distribuit in mod aleator thread-urilor.

Detalii de implementare

```
void
CalculateConvSequential(
    _In_ int                                    Displacement,
    _In_ int**                                  Matrix,
    _In_ int**                                  ConvMatrix,
    _In_ int                                    StartLine,
    _In_ int                                    StopLine,
    _In_ int                                    StartColumn,
    _In_ int                                    StopColumn,
    _In_ std::barrier<void(*)(void) noexcept>* Barrier = nullptr
)
{
    if (StartLine > StopLine)  // if we specify more threads than the total number of tasks then some thr
    {                          // and therefore StartLine would be greater than StopLine
        return;
    }

    ThreadData threadData = { 0 };
    threadData.DataLineAbove = new int[m + Displacement * 2]();
    threadData.DataLineBelow = new int[m + Displacement * 2]();

    for (int j = StartColumn - Displacement; j <= StopColumn + Displacement; ++j)
    {
        threadData.DataLineAbove[j] = Matrix[StartLine - 1][j];
        threadData.DataLineBelow[j] = Matrix[StopLine + 1][j];
    }

    if (Barrier)
    {
        Barrier->arrive_and_wait();
    }

```

```
    for (int i = StartLine; i <= StopLine; ++i)
    {
        int valueFromLeft = Matrix[i][StartColumn - 1];
        for (int j = StartColumn; j <= StopColumn; ++j)
        {
            int matrixValue = 0;
            for (int p = i - Displacement; p < i + k - Displacement; ++p)
            {
                for (int q = j - Displacement; q < j + k - Displacement; ++q)
                {
                    int value = 0;
                    if (p == i && q == j - 1)  // use the value on the left before it was previously modi
                    {
                        value = valueFromLeft;
                    }
                    else if (p == i - 1)  // use the values above before they were modified
                    {
                        value = threadData.DataLineAbove[q];
                    }
                    else if (p == StopLine + 1)  // use the values below before they were modified
                    {
                        value = threadData.DataLineBelow[q];
                    }
                    else
                    {
                        value = Matrix[p][q];
                    }

                    matrixValue += value * ConvMatrix[p + Displacement - i][q + Displacement - j];
                }
            }

            threadData.DataLineAbove[j - 1] = valueFromLeft;
            valueFromLeft = Matrix[i][j];
            Matrix[i][j] = matrixValue;
        }

        threadData.DataLineAbove[StopColumn] = valueFromLeft;
        threadData.DataLineAbove[StopColumn + 1] = Matrix[i][StopColumn + 1];
    }

    delete[] threadData.DataLineAbove;
    delete[] threadData.DataLineBelow;
}
```

Aceasta este functia “worker” pentru fiecare thread. Acest va face calcul doar intre (StartLinie, StopLinie) - (StartColoana, StopColoana). Pentru a respecta contrangerea ca modificarile sa fie facute tot in matricea initiala, este nevoie sa salvam cateva informatii pentru fiecare thread inainte ca alt thread sa le modifice. Astfel, salvam pentru fiecare thread linia de deasupra liniei de start si linia dedesubt-ul linie de stop; in acest fel ne asiguram 2 thread-uri nu se impacteaza reciproc; mai este nevoie ca sa facem acest lucru pentru toate threadurile inainte ca oricare dintre acestea sa inceapa sa lucreze, iar pentru asta folosim o bariera. Mai departe, fiecare thread isi gestioneaza informatia necesara refolosind memoria deja alocata.
Pentru valoarea (i, j+1) avem nevoie de valoarea (i, j) inainte sa fie modificata pe care o salvam in variabila valueFromLeft. Dupa ce am procesat o linie, i, folosind linia i - 1, refolosim spatiul deja alocat memorand in el linia i inainte ca aceasta sa fie modificata astfel: dupa ce am procesat un element de pe linie, sa-i spunem (i, j), putem deja actualiza elementul (j - 1) din vectorul cu linia de deasupra cu elementul (i, j - 1) inainte de modificare adica valueFromLeft.

Testare si analiza performantei

C++/Java

Tip matrice	Tip calcul	Numar threaduri	Timp executie (secunde)	
			C++	Java
N=M=10 n=m=3	Secvential	1	0.38690	0.46624
	Paralel	2	0.43314	0.45504
N=M=1000 n=m=3	Secvential	1	1.06801	1.14481
	Paralel	2	0.99611	1.14924
		4	0.94875	1.12875
		8	0.95454	1.16146
		16	0.89153	1.18848
N=10 M=10000 n=m=3	Secvential	1	0.34583	0.64126
	Paralel	2	0.34603	0.64783
		4	0.35630	0.66258
		8	0.34038	0.64685
		16	0.34846	0.65729
N=10000 M=10 n=m=3	Secvential	1	0.34016	0.64810
	Paralel	2	0.35565	0.64668
		4	0.32150	0.68262
		8	0.37663	0.69050
		16	0.32501	0.68464

Concluzii

Putem observa ca diferenta ca si timp de executie intre C++ si Java s-a diminuat, dar totusi C++ este aproape de 2 ori mai rapid in unele cazuri.
De asemenea, se poate observa o diferenta notabila atunci cand folosim 16 threaduri in loc de 2,4 sau 8 pentru unele cazuri in C++.
Fata de laboratorul 1, timpii de executie sunt mai mici semnificativ in multe cazuri pentru implementarea Java. Pentru C++, pare ca timpii de executie sunt oarecum tot pe acolo.
Complexitate spatiu: fara a lua in calcul matricile, avem 2 * O(m) * nr_threaduri ~ O(m) care este mai mica decat O(n*m) folosita in laboratorul precedent.