Documentatie lab 3 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.
Calcul paralel distribuit (folosind **p** procese 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. (P-1) | N. Procesul 0 (master) nu participa la efectuarea calculelor de actualizare a elementelor din matrice; acesta doar citeste/scrie in fisier si trimite informatiile necesare celorlalte procese.

Proiectare

Ca si structuri de date folosite, avem: tablouri unidimensionale si bidimensionale alocate dinamic.
Partitionarea pe procese a fost facuta respectand constrangerea, adica: pentru fiecare proces a fost alocat un numar egal de task-uri, iar datorita faptului ca p este impar si (p-1) | N nu avem rest.

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
)
{
    ProcessData processData = { 0 };
    processData.DataLineAbove = new int[MAX_COLUMNS + Displacement * 2]();
    processData.DataLineBelow = new int[MAX_COLUMNS + Displacement * 2]();

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

    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 + CONV_K - Displacement; ++p)
            {
                for (int q = j - Displacement; q < j + CONV_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 = processData.DataLineAbove[q];
                    }
                    else if (p == StopLine + 1)  // use the values below before they were modified
                    {
                        value = processData.DataLineBelow[q];
                    }
                    else
                    {
                        value = Matrix[p][q];
                    }

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

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

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

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

Aceasta este functia “worker” pentru fiecare proces; va face calcul doar intre (StartLinie, StopLinie) - (StartColoana, StopColoana). Pentru a respecta contrangerea ca modificarile sa fie facute tot in matricea initiala, este nevoie de informatii de la alte procese. Astfel, salvam pentru fiecare proces linia de deasupra liniei de start si linia dedesubt-ul linie de stop. Mai departe, fiecare proces 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.

La inceput, procesul master trimite tuturor celorlalte procese matricea de convolutie (broadcast). Apoi, calculeaza pentru fiecare proces un start si un stop si ii trimite informatiile necesare incat acesta sa poata face calculul, iar la final asteapta dupa rezultat. Dupa ce procesul master primeste toate rezultatele, le combina si scrie rezultatul final in fisier.

Procesele isi trimit unul altuia informatiile necesare, adica linia de dedesubt este trimisa urmatorului proces, iar linia de deasupra este trimisa procesului anterior; cu exceptia liniei anterioare primei linii si liniei de dupa ultima linie care sunt trimise de procesul master.

Testare si analiza performantei

C++

Tip matrice	Numar procese	Timp executie (secunde)	
		T1	T2
N=M=1000 n=m=3	5	1.35430	0.01558
	9	1.79385	0.02109

	21	3.16444	0.04136

Concluzii

Putem observa ca odata cu cresterea numarului de procese si timpul de executie creste, pare ca liniar. Cred ca acest lucru este datorita faptului ca este un overhead mai mare pe retea, este nevoie sa se trimita datele in mai multe locuri si sa se si primeasca inapoi. Mai mult decat atat, chiar si crearea efectiva de alte procese are un impact.
Pentru 4 threaduri la laboratorul 2 avem un timp de 0.94875, iar aici de 1.35430, deci putem observa o crestere atunci cand folosim procese vs threaduri. Pentru 16 threaduri avem 0.89153, deci cresterea e una semnificativ mai mare atunci cand si numarul proceselor creste, nu este direct proporionala.
Folosindu-ne de T2 putem deduce ca timpul efectiv necesar calculului este unul foarte mic, iar restul de timp este datorita overheadului generat de comunicarea intre procese si crearea acestora.