Documentatie lab 1

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)
Constrangere: Impartire cat mai echilibrata si eficienta a calculul pe threaduri!
(max_task_count_per_thread <= min_task_count_per_thread + 1)

Proiectare

Ca si structure de date folosite, avem: tablouri unidimensionale si bidimensionale alocate atat static cat si 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 StartLine,
    _In_     int StopLine,
    _In_     int StartColumn,
    _In_     int StopColumn
)
{
    for (int i = StartLine; i <= StopLine; ++i)
    {
        for (int j = StartColumn; j <= StopColumn; ++j)
        {
            for (int p = i - Displacement; p < i + k - Displacement; ++p)
            {
                for (int q = j - Displacement; q < j + k - Displacement; ++q)
                {
                    ResultMatrix[i][j] += Matrix[p][q] * ConvMatrix[p + Displacement - i][q + Displacemen
                }
            }
        }
    }
}
```

Aceasta functie este una mai “generica” folosita pentru ambele distributii (pe linie, pe coloana) prin cei 4 parametri. Acestia trebuie calculati apriori apelului functie, iar un thread va face calcul doar intre (StartLinie, StopLinie) - (StartColoana, StopColoana).

Testare si analiza performantei

C++

Tip matrice	Tip alocare	Tip calcul	Numar threaduri	Strategie calcul	Timp executie (secunde)
N=M=10 n=m=3	Static	Secvential	1	-	0.09789
		Paralel	4	Linie	0.10499
				Coloana	0.09470
	Dinamic	Secvential	1	-	0.11401
		Paralel	4	Linie	0.10469
				Coloana	0.12838
N=M=1000 n=m=5	Static	Secvential	1	-	0.81584
		Paralel	2	Linie	0.81006
				Coloana	0.80559
			4	Linie	0.76613
				Coloana	0.78878
			8	Linie	0.78283
				Coloana	0.79437
			16	Linie	0.82446
				Coloana	0.79960
	Dinamic	Secvential	1	-	0.83830
		Paralel	2	Linie	0.80336
				Coloana	0.82821
			4	Linie	0.81304
				Coloana	0.81024
			8	Linie	0.81647
				Coloana	0.85685
			16	Linie	0.85009
				Coloana	0.87902
N=10 M=10000 n=m=5	Static	Secvential	1	-	0.16596
		Paralel	2	Linie	0.16538
				Coloana	0.19866
			4	Linie	0.18649
				Coloana	0.20835
			8	Linie	0.18072
				Coloana	0.16939
			16	Linie	0.16118

				Coloana	0.15844
	Dinamic	Secvential	1	-	0.16453
		Paralel	2	Linie	0.17785
				Coloana	0.16057
			4	Linie	0.16527
				Coloana	0.19081
			8	Linie	0.16908
				Coloana	0.17051
			16	Linie	0.16209
				Coloana	0.16828
N=10000 M=10 n=m=5	Static	Secvential	1	-	0.18416
		Paralel	2	Linie	0.17912
				Coloana	0.20096
			4	Linie	0.21345
				Coloana	0.22778
			8	Linie	0.21600
				Coloana	0.18600
			16	Linie	0.18673
				Coloana	0.15901
	Dinamic	Secvential	1	-	0.18182
		Paralel	2	Linie	0.16891
				Coloana	0.20789
			4	Linie	0.16297
				Coloana	0.21296
			8	Linie	0.19224
				Coloana	0.21108
			16	Linie	0.18949
				Coloana	0.16137

Java

Tip matrice	Tip calcul	Numar threaduri	Strategie calcul	Timp executie (secunde)
N=M=10 n=m=3	Secvential	1	-	0.66496
	Paralel	4	Linie	0.67788
			Coloana	0.64345

N=M=1000 n=m=5	Secvential	1	-	1.41645
	Paralel	2	Linie	1.35424
			Coloana	1.39390
		4	Linie	1.38764
			Coloana	1.44079
		8	Linie	1.36988
			Coloana	1.45705
		16	Linie	1.42140
			Coloana	1.49171
N=10 M=10000 n=m=5	Secvential	1	-	0.93264
	Paralel	2	Linie	0.89415
			Coloana	0.99171
		4	Linie	0.91678
			Coloana	0.94991
		8	Linie	0.93141
			Coloana	0.94569
		16	Linie	0.94703
			Coloana	0.97650
N=10000 M=10 n=m=5	Secvential	1	-	0.89518
	Paralel	2	Linie	0.95900
			Coloana	0.98489
		4	Linie	0.93889
			Coloana	0.94264
		8	Linie	0.97422
			Coloana	1.00605
		16	Linie	1.00220
			Coloana	0.98017

Concluzii

Putem observa ca timpii de executie masurati pentru implementarea Java sunt cu mult mai mari (de aproximativ 5 ori) decat cei masurati pentru implementarea C++.
Pentru implementarea C++, alocarea dinamica are un overhead aproape nesemnificativ fata de alocarea statica.
De asemenea, putem observa ca in unele cazuri, mai multe threaduri inseamna mai bine (timp mai mic de executie), dar nu este general valabil. Mai mult decat atat, putem observa si cazuri in care calculul paralel este mai rapid decat cel secvential.