Programare paralela si distribuita

Laborator 2

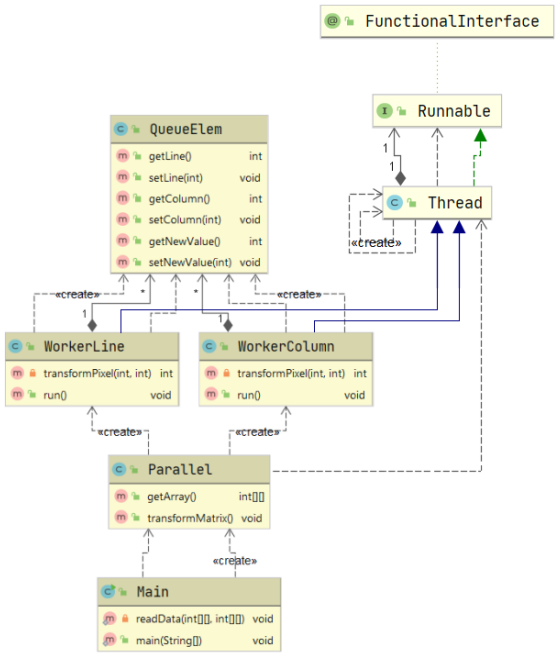
Sintea Maria, 237, 237/1

Cerinta

Se da o matrice N*M ce reprezinta o imagine reprezentata ca pixeli. Se cere transformarea ei prin operatia de convolutie cu o matrice n*m. Implementarea se va face paralel. Corectitudinea rezultatului se va verifica cu implementarea secventiala. Matricea rezultat va fi aceeasi ca si cea initiala.

Proiectare

Diagrama de clase din Java:



La implementare, am folosit if-uri in loc de padding. Am creat 4 if-uri, si fiecare verifica daca indicii liniei si coloanei sunt in limitele definite ale matricei(N, M). In cazul in care indicii depaseau o limita, indicii creati temporar erau modificati cu limita matricii, deoarce acest lucru insemna ca acel element se afla pe frontiera. Valoarea unui element de pe frontiera se afla prin copierea vecinului definit de pe linie, in cazul elementelor de pe frontiera verticala si respectiv prin copierea vecinilului definit de pe coloana, in cazul elementelor de pe frontiera orizontala.
Thread-urile au fost impartite in mod egal dupa linii, in cazul in care matricea are mai multe linii, si dupa coloane, in cazul in care matricea are mai multe coloane, iar indicele linilor ce sunt parcurse intr-un thread au fost alese in mod liniar.

Pentru a asigura corectitudinea rezultatului, inainte de a salva valoarea calculata in matrice, trebuie sa ne asiguram ca valoarea initiala nu va mai fi folosita. Pentru a realiza acest lucru, utilizat tehnica descrisa

la 2b. Am salvat valorile nou calculate care se aflta pe frontiere in doi vectori separati: unul pentru frontiera superioara(din stanga, pentru cazul parcurgerii pe coloane) si unul pentru frontiera inferioara(din dreapta, pentru cazul parcurgerii pe coloane) unei portiuni de linii(coloane) asignate unui thread. Pentru valorile care nu se aflau pe frontiera, am retinut valorile nou calculate intr-o coada, impreuna cu linia si coloana pe care ar trebui sa se afle. In timp ce parcurgeam matricea pentru calculul noilor valori, verificam daca varful cozii are o pozitie care mai are sanse sa fie folosita intr-un calcul viitor. In cazul in care nu va mai fi folosita, adica in cazul in care aceasta valoare se afla in coltul din stanga, sus al frontierei elementului curent, aceasta valoare va fi actualizata in matrice si se va scoate din coada.

Testare

1. Java:

Tip Matrice	Nr threads	Timp executie
N = M = 10 n = m = 3	2	7.55704
N = M = 1000 n = m = 5	1	320.17953
	2	271.97355
	4	221.58313
	8	223.83982
	16	242.25906
N = 10 M = 10000 n = m =5	1	129.04421
	2	123.43691
	4	143.40069
	8	153.13604
	16	163.62372
N = 10000 M = 10 n = m = 5	1	72.811229
	2	71.765689
	4	67.251719
	8	71.323689
	16	73.047259

2. C++:

Tip Matrice	Tip Alocare	Nr threads	Timp executie
N = M = 10 n = m = 3	static	2	0.0064
	dinamic	2	0.0052
N = M = 1000 n = m = 5	static	1	4.5636
		2	2.8628
		4	3.1971
		8	3.9019
		16	4.7488
	dinamic	1	1.0333
		2	0.6298
		4	0.7623
		8	1.001
		16	1.1182

N = 10 M = 10000 n = m =5	static	1	0.1312
		2	0.0952
		4	0.091
		8	0.1278
		16	0.1252
	dinamic	1	0.1369
		2	0.0935
		4	0.0911
		8	0.1127
		16	0.1137
N = 10000 M = 10 n = m = 5	static	1	0.152
		2	0.157
		4	0.0984
		8	0.1267
		16	0.1264
	dinamic	1	0.1405
		2	0.1576
		4	0.0917
		8	0.1255
		16	0.1209

Analiza

1. **Corectitudinea codului:** a fost verificata prin comparatia rezultatelor curente cu rezultatele obtinute in prima tema si acestea au fost egale.
2. **Comparati performanta pentru fiecare caz – secvential versus paralel si variantele paralele intre ele:**
   - **Cazul N=M=10, n=m=3:** in acest caz, performanta este mai buna in cazul implementarii in C++ cu alocare dinamica, urmata de cea in C++ cu alocare statica si in cele din urma implementarea din Java
   - **Cazul N=M=1000, n=m=5:** in acest caz, performanta este mai buna in implementarea cu 4 threaduri in Java, acest punct fiind un minim local pentru graficul timpilor rularilor paralele; in C++, performanta este mai buna in cazul cu 2 threaduri
   - **Cazul N=10, M=10000, n=m=5: :** in acest caz, performanta este mai buna in implementarea cu 2 threaduri in Java, iar in C++, pentru ambele variante de alocari, cazul cu 4 threaduri este un minim local pentru graficul timpilor rularilor
   - **Cazul N=10000, M=10, n=m=5:** in acest caz, performanta este mai buna in implementarea cu 4 threaduri, in toate cazurile
3. **Comparati timpii de executie obtinuti cu implementarea Java versus implementarea C++:** in general, timpii de executie in C++ sunt mai buni decat in Java , exceptand ultimul cazurile cu matricile mari de dimensiuni 1000*1000 si respectiv 10000*10, unde timpii de executie sunt mai buni in Java
4. **Evaluati complexitatea-spatiu**: O(N * M + n * m + NR_THRS * n * M), pentru cazul impartirii pe linii si O(N * M + n * m + NR_THRS * N * m), pentru cazul impartirii pe coloane.

Comparativ cu implementarea facuta la laboratorul trecut, spatiul utilizat se reduce semnificativ.

5. **Comparatie a variantelor din punct de vedere al complexitatii spatiu**: Din punct de vedere al spatiului, variantele care folosesc bariere sunt mai eficiente, deoarece stocheaza doar un intreg, comparativ cu flagurile, in care va trebui sa fie stocata cate o variablia atomica pentru fiecare thread. Din punct de vedere al operatiilor efectuate nu exista o diferenta semnificativa, pentru ambele retinandu-se frontiera unui element ce nu este aflat pe frontiera portiunii, plus copiile elementelor de pe frontiera in doi vectori.
6. **Comparatie a variantelor din punct de vedere al performantei:** Din punct de vedere al performantei, nu exista nici o diferenta intre variantele cu flaguri si variantele cu bariera, dat fiind faptul ca in ambele trebuie parcurse toate threadurile pentru a verifca ca au ajuns pana in punctul cerut. In rest, varianta cu prelucrarea elementelor inainte de bariera este mai eficienta, deoarece nu mai trebuie sa pastreze frontiera unui element si pentru frontiera portiunii asignata unui flag, aceste elemente fiind salvate direct intr-un sector separat, iar calculul lor nu va fi influentat de acest lucru, comparativ cu cazul prelucarii elementelor dupa bariera, caz in care frontiera unui element va trebui salvata pentru toate elementele din portiune.