Programare paralela si distribuita

Laborator 1

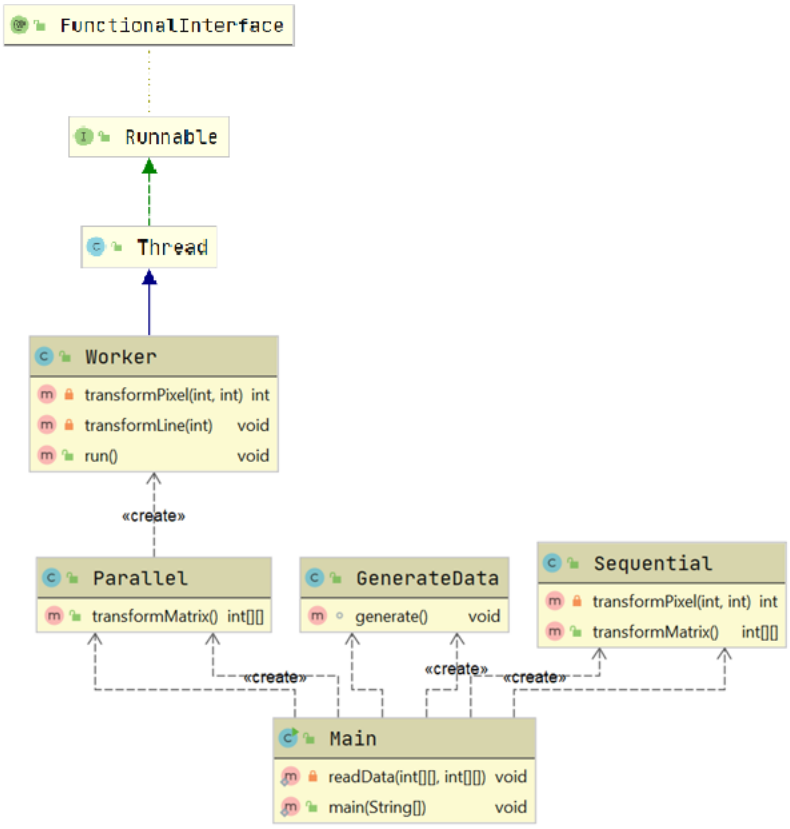
Sintea Maria, 237, 237/1

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 secvential si recursiv. Matricea rezultat va fi diferita de 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 dupa linii, iar indicele linilor ce sunt parcurse intr-un thread au fost alese in mod ciclic.

Testare

1. Java:

Tip matrice	Nr threads	Timp executie(milisec)
N=M=10 n=m=3	secvential	0.07994
	4	0.88129
N=M=1000 n=m=5	secvential	59.433
	1	112.14581
	2	73.73524
	4	69.8361
	8	57.94275
	16	65.92954
N=10 M=10000 n=m=5	secvential	36.90068
	1	72.21424
	2	46.09014
	4	40.37148
	8	35.98682
	16	32.42455
N=10000 M=10 n=m=5	secvential	13.93745
	1	26.30123
	2	16.49996
	4	18.75022
	8	19.62107
	16	20.03474

2. C++:

Tip matrice	Tip alocare	Nr threads	Timp executie(sec)
N=M=10 n=m=3	static	secvential	0
		4	0.007
	dinamic	secvential	0
		4	0.0071
N=M=1000 n=m=5	static	secvential	0.1489
		1	0.317
		2	0.211
		4	0.123
		8	0.061
		16	0.052
	dinamic	secvential	0.2074
		1	0.2119
		2	0.11741
		4	0.0777
		8	0.0527
		16	0.0518

N=10 M=10000 n=m=5	static	secvential	0.0088
		1	0.0267
		2	0.0154
		4	0.0119
		8	0.0134
		16	0.0107
	dinamic	secvential	0.0243
		1	0.0274
		2	0.0165
		4	0.0146
		8	0.0098
		16	0.0114
N=10000 M=10 n=m=5	static	secvential	0.0086
		1	0.0285
		2	0.0149
		4	0.0116
		8	0.0123
		16	0.0113
	dinamic	secvential	0.0231
		1	0.0298
		2	0.0189
		4	0.0128
		8	0.0127
		16	0.0112

Analiza

1. **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 rularea cu implementare secventiala, decat in cazul implementarii paralele
    - **Cazul N=M=1000, n=m=5:** in acest caz, performanta este mai buna in implementarea cu 8 threaduri in Java, acest punct fiind un minim local pentru graficul timpilor rularii paralele; in C++, performanta este mai buna in cazul cu 16 threaduri
    - **Cazul N=10, M=10000, n=m=5: :** in acest caz, performanta este mai buna in implementarea cu 16 threaduri in Java si in C++, varianta cu alocare statica, iar la alocare dinamica in C++, performanta este mai buna in cazul cu 8 threaduri
    - **Cazul N=10000, M=10, n=m=5:** in acest caz, performanta este mai buna in implementarea secventiala
2. **Comparati timpii de executie obtinuti cu implementarea Java versus implementarea C++:** in general, timpii de executie in C++ sunt mai buni decat in Java
3. **Comparati cele doua variante pentru implementarea C++:** in general, timpii de executie la implementariile cu alocare statica sunt mai buni decat cei cu alocare dinamica