Documentatie

Implementari

Java

- secvential
- linii
- coloane
- bloc

C++ (static si dinamic)

- secvential
- linii
- coloane
- bloc

Observatii

- implementarea Java merge mai incet decat cea C++
- varianta dinamica in C++ este un pic mai rapida decat cea statica
- varianta secventiala tinde sa mearga mai incet decat cele paralele

Distributie

Toate thread-urile primesc aceeasi parametri de intrare:

- matricea careia i se aplica convolutia
- matricea de convolutie
- matricea unde se salveaza rezultatul
- start
- stop

In functie de metoda (care se da din linia de comanda), se seteaza parametrul `n` (reprezentand fie numarul total de linii, fie numarul total de coloane, fie numarul total de elemente din matrice), si se alege functia care va fi rulata pe thread-uri.

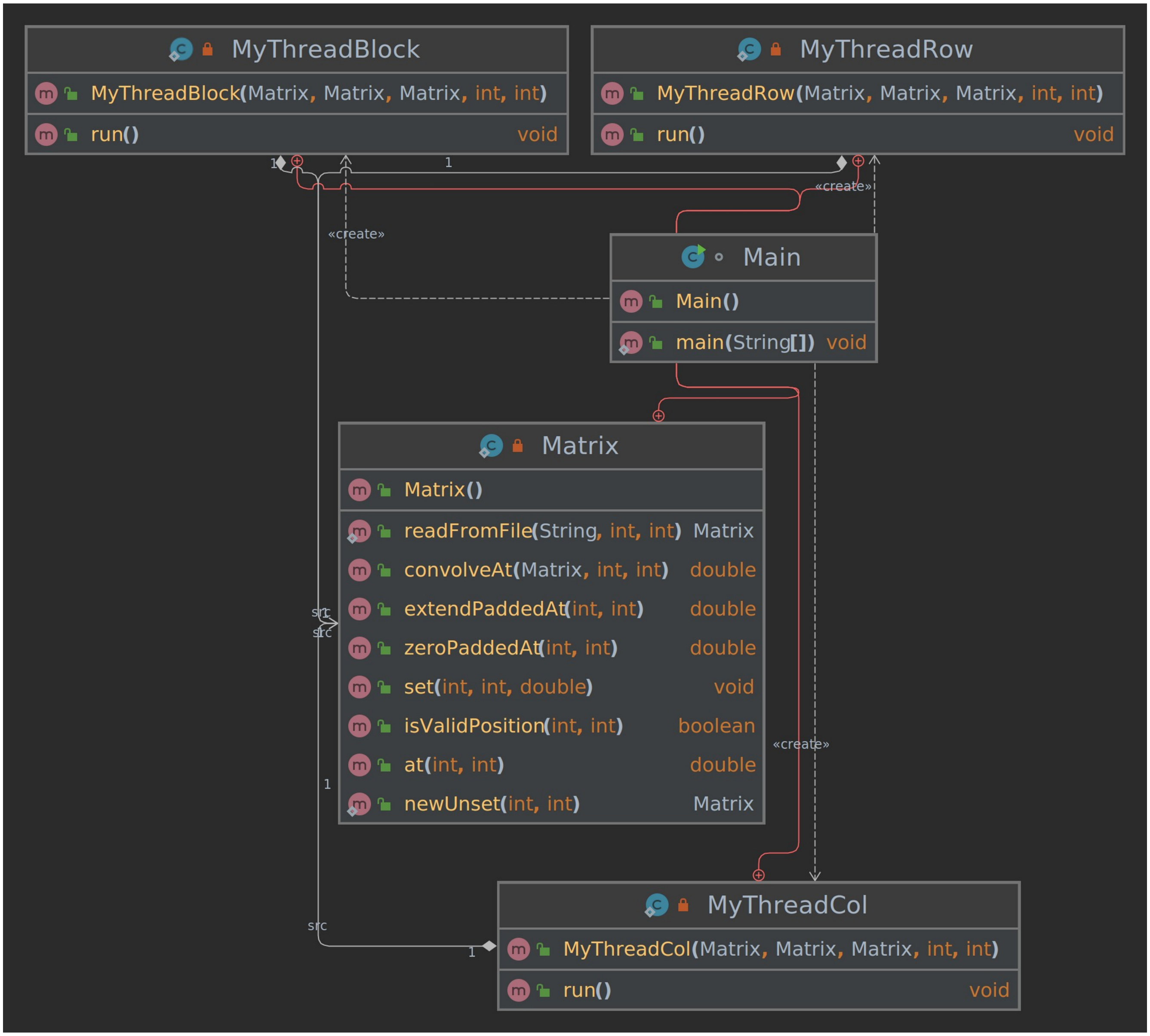
Parametrul n se imparte mereu cat mai "fair" intre thread-uri (diferenta dintre volumul de munca primit de fiecare thread va fi mereu cel mult 1).

```
daca metoda = "linii" atunci
     n <- N
daca metoda = "coloane" atunci
     n <- M
daca metoda = "blocuri" atunci
     n <- N*M

start <- 0
stop <- 0
pentru i <- 1,p executa
     len <- n/p
     daca i < n%p atunci
          len <- len + 1

     stop <- stop + len
     daca metoda = "linii" atunci
          incepeThreadPtLinii(mat, conv, dst, start, stop)
     daca metoda = "coloane" atunci
          incepeThreadPtColoane(mat, conv, dst, start, stop)
     daca metoda = "blocuri" atunci
          incepeThreadPtBlocuri(mat, conv, dst, start, stop)
     start <- start + len
```

Diagrama UML implementare Java


MyThreadBlock
MyThreadBlock(Matrix, Matrix, Matrix, int, int)
run() void
MyThreadRow
MyThreadRow(Matrix, Matrix, Matrix, int, int)
run() void
1
1
«create»
«create»
Main
Main()
main(String[]) void
Matrix
Matrix()
readFromFile(String, int, int) Matrix
convolveAt(Matrix, int, int) double
extendPaddedAt(int, int) double
zeroPaddedAt(int, int) double
set(int, int, double) void
isValidPosition(int, int) boolean
at(int, int) double
newUnset(int, int) Matrix
src
src
1
«create»
src
1
MyThreadCol
MyThreadCol(Matrix, Matrix, Matrix, int, int)
run() void

Tabel performanta

Tabelul de performanta include si metoda folosita.

runner	mat_size	conv_size	method	p	elapsed
cpp_dynamic	N=10,M=10	N=3,M=3	block	4	0.000
			col	4	0.002
			row	4	0.002

			sequential	0	0.002
	N=10,M=10000	N=5,M=5	block	2	0.05
				4	0.05
				8	0.05
				16	0.05
			col	2	0.05
				4	0.04
				8	0.05
				16	0.04
			row	2	0.04
				4	0.04
				8	0.04
				16	0.05
			sequential	0	0.05
	N=1000,M=1000	N=5,M=5	block	2	0.42
				4	0.42
				8	0.41
				16	0.41
			col	2	0.42
				4	0.42
				8	0.42
				16	0.42
			row	2	0.42
				4	0.41
				8	0.41
				16	0.4
			sequential	0	0.71
	N=10000,M=10	N=5,M=5	block	2	0.05
				4	0.04
				8	0.05
				16	0.05
			col	2	0.05
				4	0.05
				8	0.05

				16	0.05
			row	2	0.05
				4	0.05
				8	0.04
				16	0.05
			sequential	0	0.05
cpp_static	N=10,M=10	N=3,M=3	block	4	0.004
			col	4	0.002
			row	4	0.002
			sequential	0	0.002
	N=10,M=10000	N=5,M=5	block	2	0.05
				4	0.05
				8	0.05
				16	0.05
			col	2	0.05
				4	0.05
				8	0.05
				16	0.05
			row	2	0.05
				4	0.05
				8	0.05
				16	0.05
			sequential	0	0.06
	N=1000,M=1000	N=5,M=5	block	2	0.42
				4	0.42
				8	0.42
				16	0.42
			col	2	0.43
				4	0.41
				8	0.43
				16	0.41
			row	2	0.43
				4	0.41
				8	0.41

				16	0.41
			sequential	0	0.79
	N=10000,M=10	N=5,M=5	block	2	0.05
				4	0.05
				8	0.05
				16	0.05
			col	2	0.05
				4	0.05
				8	0.05
				16	0.05
			row	2	0.05
				4	0.05
				8	0.05
				16	0.05
			sequential	0	0.05
java	N=10,M=10	N=3,M=3	block	4	0.72
			col	4	1.01
			row	4	1.03
			sequential	0	1.01
	N=10,M=10000	N=5,M=5	block	2	1.77
				4	1.79
				8	1.76
				16	1.83
			col	2	1.75
				4	1.75
				8	1.77
				16	1.77
			row	2	1.73
				4	1.74
				8	1.77
				16	1.75
			sequential	0	2.3
	N=1000,M=1000	N=5,M=5	block	2	4.85
				4	4.77

				8	4.83
				16	4.84
			col	2	4.81
				4	4.81
				8	5.08
				16	4.86
			row	2	5.11
				4	4.75
				8	4.84
				16	4.86
			sequential	0	8.19
	N=10000,M=10	N=5,M=5	block	2	1.75
				4	1.75
				8	1.77
				16	1.75
			col	2	1.74
				4	1.78
				8	1.76
				16	1.77
			row	2	1.78
				4	1.75
				8	1.76
				16	1.77
			sequential	0	1.98