# Documentatie lab 1

**Hodosi Attila - g234/1**

**Analiza cerintelor:**

Se considera o imagine reprezentata printr-o matrice de dimensiune (MxN).

Aceasta se va transforma aplicandui o filtrare cu o matrice mai mica, numita kernel.

Kernelul va avea numarul liniilor si coloanelor, numar impar pentru a avea un element central.

Pe fiecare element din matricea initiala asezam elementul central din kernel, iar rezultatul filtrari este suma produsul elementelor dintre elementele kernelului si elementele matricei intiale, incepand de la elementul central.

**Idee generala:**

Pentru impartirea sarcinilor am alocat sarcini in mod echilibrat pentru fiecare thread.

Fiecare thread primeste un numar de linii (daca matricea are mai multe sau egale linii decat coloane) sau coloane (daca matricea are mai multe coloane decat linii) cu cel mult o linie sau coloana diferenta.

**Bordare:**

- Pentru bordarea matricii am calculat linia si coloana  care raspunde kernelului din matricea initiala:

```
int linConvolution = linMatrix + linKernel - n / 2;
int colConvolution = colMatrix + colKernel - m / 2;
```

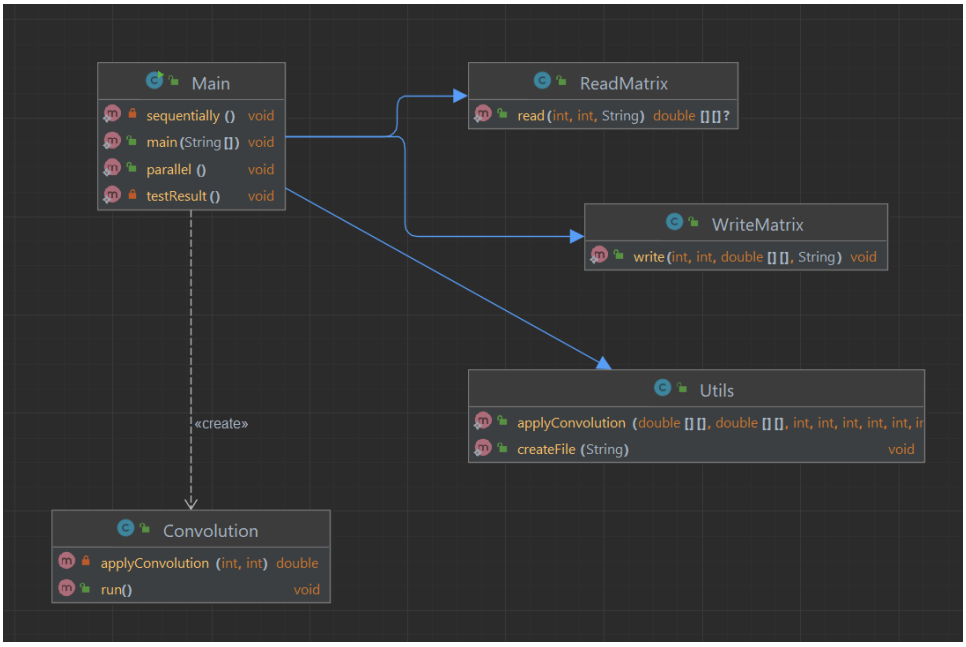
- Daca linia/coloana de convolutie(linia/coloana corespunzatoare kernelului) este invalida, atunci aduc la cea mai apropiata pozitie valida:

```
if (linConvolution < 0) {
    linConvolution = 0;
}
if (colConvolution < 0) {
    colConvolution = 0;
}
if (linConvolution >= N) {
    linConvolution = N - 1;
}
if (colConvolution >= M) {
    colConvolution = M - 1;
}
```

Iar la final calculez rezultatul:

```
convolutionResult += matrix[linConvolution][colConvolution] * kernel[linKernel][colKernel];
```

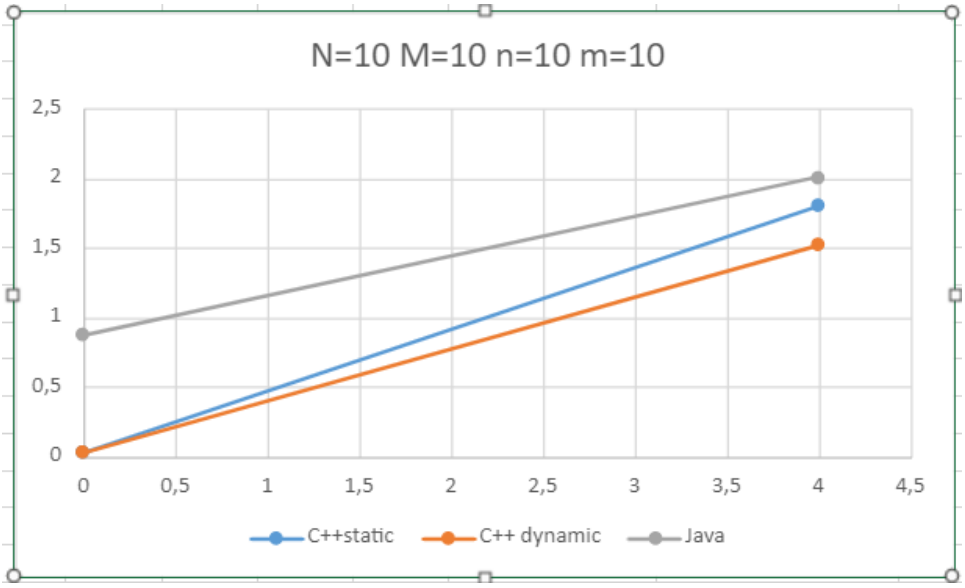
**Diagrama de clase:**

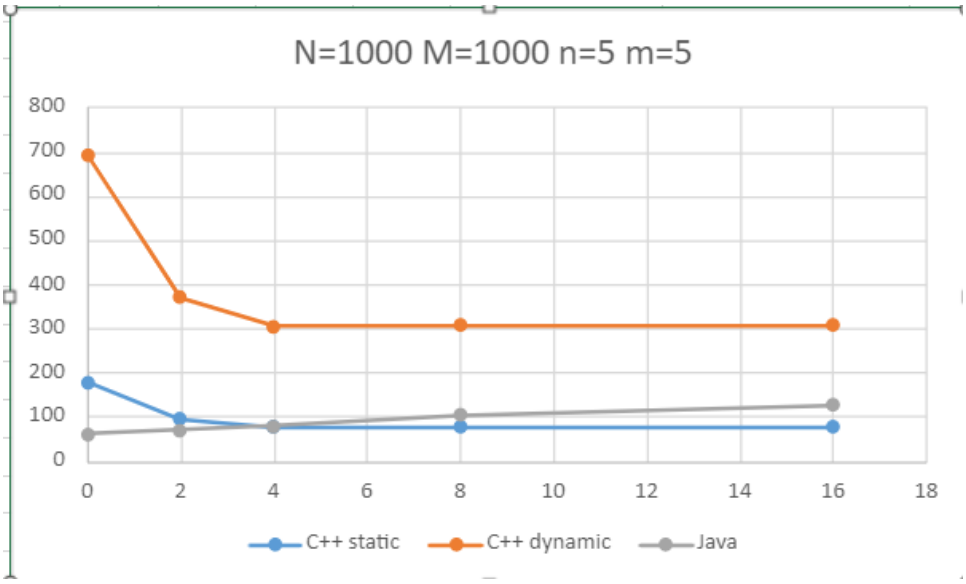


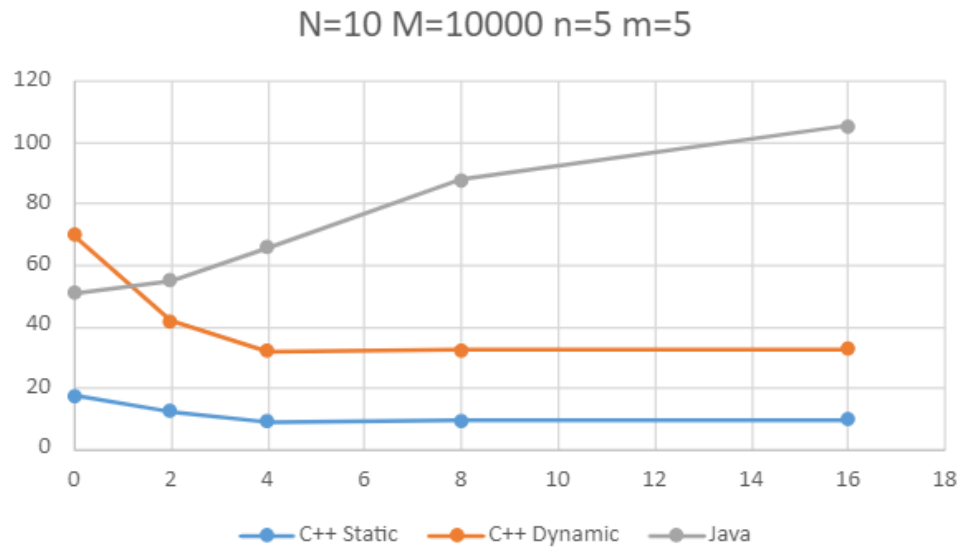
**Comparatia generala in ms:**

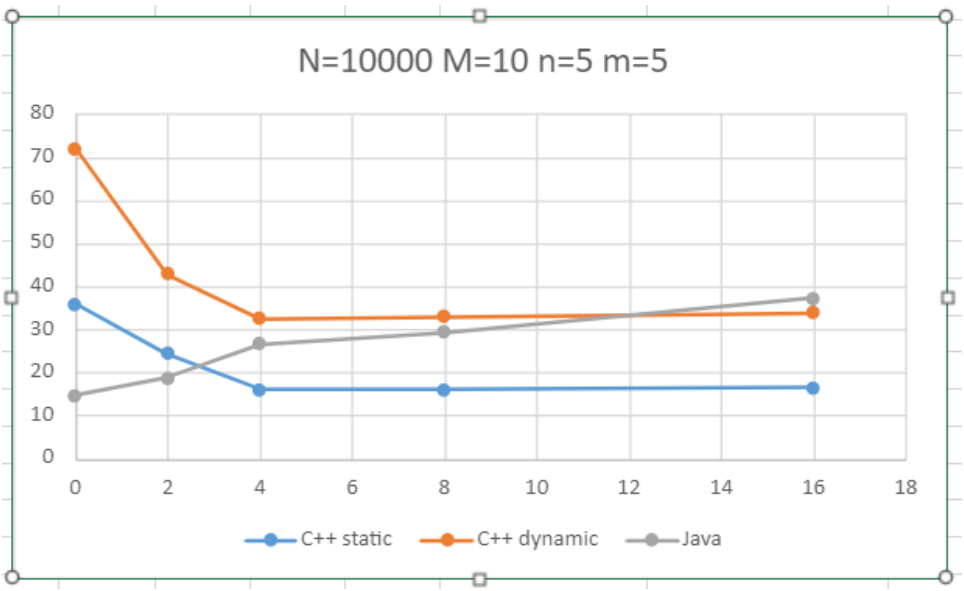
| N | M | n2 | m2 | p | Time C++ stati | Time C++ Dynami | Time Java |
|---|---|---|---|---|---|---|---|
| 10 | 10 | 3 | 3 | 0 | 0,02812 | 0,0283 | 0,86960667 |
| 10 | 10 | 3 | 3 | 4 | 1,801173333 | 1,518593333 | 2,00939333 |
| 1000 | 1000 | 5 | 5 | 0 | 178,5689333 | 691,5171333 | 61,05948 |
| 1000 | 1000 | 5 | 5 | 2 | 93,58858 | 368,6852667 | 68,6954333 |
| 1000 | 1000 | 5 | 5 | 4 | 76,84003333 | 303,4039333 | 78,0958133 |
| 1000 | 1000 | 5 | 5 | 8 | 76,14774667 | 305,5660667 | 102,424853 |
| 1000 | 1000 | 5 | 5 | 16 | 75,81612 | 305,4562667 | 125,66244 |
| 10 | 10000 | 5 | 5 | 0 | 17,16679333 | 69,60418 | 50,6236067 |
| 10 | 10000 | 5 | 5 | 2 | 12,03338 | 41,58885333 | 54,9059933 |
| 10 | 10000 | 5 | 5 | 4 | 8,65256 | 31,67846 | 65,4616133 |
| 10 | 10000 | 5 | 5 | 8 | 8,919693333 | 31,93340667 | 87,5408 |
| 10 | 10000 | 5 | 5 | 16 | 9,401453333 | 32,55434667 | 105,100373 |
| 10000 | 10 | 5 | 5 | 0 | 35,90747333 | 71,92208 | 14,67646 |
| 10000 | 10 | 5 | 5 | 2 | 24,40782 | 42,93436 | 18,5974 |
| 10000 | 10 | 5 | 5 | 4 | 15,90873333 | 32,50539333 | 26,55646 |
| 10000 | 10 | 5 | 5 | 8 | 15,98459333 | 32,85304667 | 29,2567467 |
| 10000 | 10 | 5 | 5 | 16 | 16,40878667 | 33,68956667 | 37,1535 |

**Diagrame:**




N=1000 M=1000 n=5 m=5
800
700
600
500
400
300
200
100
0
0
2
4
6
8
10
12
14
16
18
C++ static
C++ dynamic
Java


N=10 M=10000 n=5 m=5
120
100
80
60
40
20
0
0
2
4
6
8
10
12
14
16
18
C++ Static
C++ Dynamic
Java




**Concluzie generala:**

- folosirea threadurilor pe matrici mici incetineaza calculele

- folosirea threadurilor pe matrici mari aduce imbunatatiri semnificative, dar exista o limita la numarul threadurilor de unde performanta nu creste semnificativ sau chiar scade, deci trebuie avut grija sa alegem numarul corect de threaduri, nici prea putin nici, dar nici prea mult