Laborator 1

1. Analiza Cerintelor

Se considera o imagine reprezentata printr-o matrice de pixeli, F , de dimensiune (MxN). Se cere transformarea ei aplicand o filtrare cu o fereastra definita de multimea de indici W cu coeficientii w_{kl} (reprezentati prin matricea W[k,l], unde -n/2<=k<=n/2, -m/2<=l<=m/2; si n<N, m<M, n,m impare).
Transformarea unui pixel:

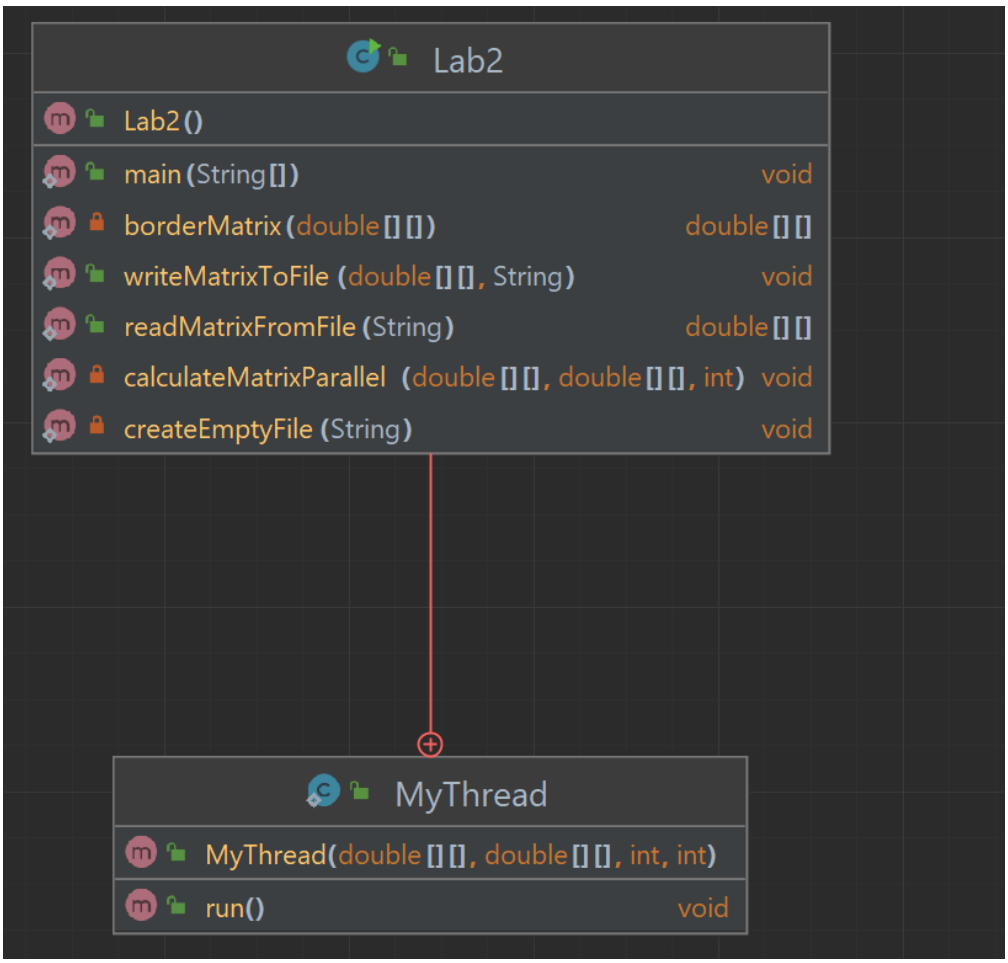
$$v(m,n) = \sum_{(k,l)\in W} w_{kl} f(m-k, n-l)$$

Implementarile trebuie realizate in:

1) Java
2) C++
   a) Matrici alocate dinamic

2. Proiectare si implementare

a. Java



- Metoda **double[][] borderMatrix(double[][] matrix1)** Bordeaza matricea primita ca paremetru cu adaugandu-i 2 linii si 2 coloane. Borderul contine cel mai apropiat element de la marginea matricii initiale.

```
double** borderMatrix(double** inputMatrix, int M, int N ) {

    double** borderedMatrix = new double* [M + 2];
    for (int i = 0; i < M + 2; i++) {
        borderedMatrix[i] = new double[N + 2];
    }
    for (int j = 0; j < N; j++) {
        borderedMatrix[0][j + 1] = inputMatrix[0][j];
        borderedMatrix[M + 1][j + 1] = inputMatrix[M - 1][j];
    }
    for (int i = 0; i < M; i++) {
        borderedMatrix[i + 1][0] = inputMatrix[i][0];
        borderedMatrix[i + 1][N + 1] = inputMatrix[i][N - 1];
    }

    for (int i = 0; i < M; i++) {
        for (int j = 0; j < N; j++) {
            borderedMatrix[i + 1][j + 1] = inputMatrix[i][j];
        }
    }
    borderedMatrix[0][0] = inputMatrix[0][0];
    borderedMatrix[0][N + 1] = inputMatrix[0][N - 1];
    borderedMatrix[M + 1][0] = inputMatrix[M - 1][0];
    borderedMatrix[M + 1][N + 1] = inputMatrix[M - 1][N - 1];
    return borderedMatrix;
}
```

- Metoda **void calculateMatrixParallel(double[][] inputMatrix, double[][] W, int noThreads)** aplica transformarea fiecarui element al matricei initiale (parametrul 1) si se salveaza rezultatul in aceeasi matrice, folosind **noThreads** (parametrul 3) Threaduri .
    - Initial, fiecarui thread i-a fost atributi un numar de linii egal cu raportul dintr M(numarul de linii ale matricii input)/P (numarul de threaduri). In cazul in care impartirea nu se realizeaza exact, fiecare thread primeste cate o noua linie din rest pana acesta devine 0.
- **Executia unui Thread**
    - Fiecare tread incepe prin a-si salva datele necesare calcularii fiecarui element: **double[][] auxiliaryMemory – este o in care se salveaza toate elemente necesare calculul conform formulei de mai jos, salvand folosind borderul calculat anterior.**

$$v(m,n) = \sum_{(k,l)\in W} w_{kl} f(m-k, n-l)$$

```
auxiliaryMemory = new double[endLine - startLine + 1 + 2 * (k / 2)][N];
for(int i = startLine - k/2; i <= endLine + k / 2; i++) {
    for(int j = 0; j < N ; j++) {
        auxiliaryMemory[i - startLine + k / 2 ][j] = inputMatrix[i][j];
    }
}
```

    - Threadurile asteapta la o **bariera**, pana cand toate thread-urile isi salveaza datele necesare.

- Dupa ce toate threadurile trec de baiera se incepe calculul.

```
for(int i = k/2 ; i <= endLine - startLine + k/2; i++){
    for(int j = k/2 ; j < N - k/2; j++){
        double s = 0;
        for(int ii = -k/2; ii <= k / 2 ; ii++){
            for(int jj = -l/2 ; jj <= l / 2; jj++){
                int newI = i + ii;
                int newJ = j + jj;
                s+= auxiliaryMemory[newI][newJ] * w[ii + k/2][jj+l/2];
            }
        }
        inputMatrix[startLine + i - k/2][j] = s;
    }
}
```

- Metoda **double[][] readMatrixFromFile(String fileName)** citeste o matrice dintr-un fisier organizat astfel. Pe prima linie M – numarul de linii, pe a doua linie N – numarul de coloane, pe urmatoarele M*N linii elementele matricei.

b. C++

Abordarea in C++ este similara, difera implementarea **barierei**.

```
class my_barrier
{
public:
    my_barrier(int count) : thread_count(count), counter(0), waiting(0)
    {}

    void wait()
    {
        //fence mechanism
        std::unique_lock<std::mutex> lk(m);
        ++counter;
        ++waiting;
        cv.wait(lk, [&] {return counter >= thread_count;});
        cv.notify_one();
        --waiting;
        if (waiting == 0)
        {  //reset barrier
            counter = 0;
        }
        lk.unlock();
    }

private:
    std::mutex m;
    std::condition_variable cv;
    int counter;
    int waiting;
    int thread_count;
};
```

i. Alocare dinamica

```
void readInputMatrix() {
    Min >> M;
    Min >> N;
    inputMatrix = new double* [M];
    for (int i = 0; i < M; i++) {
        inputMatrix[i] = new double[N];
    }
    for (int i = 0; i < M; i++) {
        for (int j = 0; j < N; j++) {
            Min >> inputMatrix[i][j];
        }
    }

    Min.close();

}
```

- Pentru a trata cazul in care aplicarea “umbrei” W peste un element al matricei initiale.

```
for(int i = k/2 ; i <= endLine - startLine + k/2; i++){
    for(int j = k/2 ; j < N - k/2; j++){
        double s = 0;
        for(int ii = -k/2; ii <= k / 2 ; ii++){
            for(int jj = -l/2 ; jj <= l / 2; jj++){
                int newI = i + ii;
                int newJ = j + jj;
                s+= auxiliaryMemory[newI][newJ] * w[ii + k/2][jj+l/2];
            }
        }
        inputMatrix[startLine + i - k/2][j] = s;
    }
}
```

3. Cazuri de testare

Java

TipMatrice	TipExecutie	TimpExecutieMediu
N=M=10 si n=m=3	Secvential	0.05972
	Paralel 4	2.87558
N=M=1000 si n=m=5;	Secvential	97.60366
	Paralel 2	42.69205
	Paralel 4	28.48004
	Paralel 8	45.3283
	Paralel 16	59.51695
M=10 N=10000 si n=m=5;	Secvential	18.50175

	Paralel 2	19.12661
	Paralel 4	22.04151
	Paralel 8	19.96032
	Paralel 16	40.44917
M=10000 N=10 si n=m=5	Secvential	19.28579
	Paralel 2	16.21547
	Paralel 4	14.2248
	Paralel 8	19.68606
	Paralel 16	31.16999

C++

TipMatrice	TipExecutie	TipAlocare	TimpExecutieMediu
N=M=10 si n=m=3	Secvential	dinamic	0.00542
	Paralel 4	dinamic	0.9611
N=M=1000 si n=m=5;	Secvential	dinamic	124.1648
	Paralel 2	dinamic	52.0723
	Paralel 4	dinamic	28.74635
	Paralel 8	dinamic	18.77149
	Paralel 16	dinamic	17.333
M=10 N=10000 si n=m=5;	Secvential	dinamic	12.07543
	Paralel 2	dinamic	7.32568
	Paralel 4	dinamic	5.88935
	Paralel 8	dinamic	6.35351
	Paralel 16	dinamic	7.81527
M=10000 N=10 si n=m=5	Secvential	dinamic	12.97912
	Paralel 2	dinamic	8.13039
	Paralel 4	dinamic	5.90872
	Paralel 8	dinamic	6.01735
	Paralel 16	dinamic	6.06123

Concluzii : Executia in C++ este mai rapida decat cea in Java. Compartiv cu labul anterior, executia din aces lab este mai lenta cauza ca se aloca mai multa memorie, fiecare thread isi copiaza frontiera alaturi de liniile din matricea initiala

pentru care a fost alocat sa faca calculul. K + M / noThreads – linii pe care le copiaza un thread. K – dimensiunea ferestrei K, M – dimeansiunea matricii input.

Complexitate Memorie: O(noThreads * (K + M / noThreads) * N) = O((K + M)*N)