MARIN LEO

ZUMSTEIN PAULIN

G1

Q1

```
public Matrix times(Matrix B) {
    if(this.N!=B.M) throw new RuntimeException("produit impossible dimension
non compatibles");
    this.complexity=0;
    Matrix matrixC = new Matrix(this.M,B.N);
    for(int i=0;i<matrixC.M;i++){
        for(int j=0;j<matrixC.N;j++){
            for(int k=0;k<this.N;k++){
                matrixC.data[i][j] +=  this.data[i][k]*B.data[k][j];
                this.complexity++;
            }
        }
    }
    return matrixC;
}
```

Q2

On a créé une classe TC de la façon suivante :

```
public class TC implements Callable<Matrix> {

    private Matrix A;
    private Matrix B;

    public TC(Matrix A, Matrix B ) {
        this.A=A;
        this.B=B;
    }

    @Override
    public Matrix call() throws Exception {
        return A.times(B);
    }
}
```

Une fonction decomposition() nous permet de diviser une matrice en 4 sous-matrices.

On applique cette méthode aux 2 matrices que l'on souhaite multiplier. Ensuite on récupère les 8 sous matrices (4 par matrice).

On travaille sur la matrice à l'aide de 8 threads qui travaillent sur leur sous partie respective.

Puis on utilise une autre fonction recomposerMatrix2() qui nous permet de recomposer la grande matrice finale à partir des 4 sous matrices issues du calcul des threads.

```
public Matrix timesThread(Matrix B){
    ArrayList<Matrix> arrayListA = this.decomposition();
    ArrayList<Matrix> arrayListB = B.decomposition();

    TC HG1 = new TC(arrayListA.get(0), arrayListB.get(0));
    TC HG2 = new TC(arrayListA.get(1), arrayListB.get(2));

    TC HD1 = new TC(arrayListA.get(0), arrayListB.get(1));
    TC HD2 = new TC(arrayListA.get(1), arrayListB.get(3));

    TC BG1 = new TC(arrayListA.get(2), arrayListB.get(0));
    TC BG2 = new TC(arrayListA.get(3), arrayListB.get(2));

    TC BD1 = new TC(arrayListA.get(2), arrayListB.get(1));
    TC BD2 = new TC(arrayListA.get(3), arrayListB.get(3));

    //On initialise notre géstionaire de threads
    ExecutorService executorService = Executors.newFixedThreadPool(8);

    //submit() au lieu de execute(). submit() return un objet de type Future
afin de pouvoir acceder à l'execution des différents threads
    Future<Matrix> f_HG1 = executorService.submit(HG1);
    Future<Matrix> f_HG2 = executorService.submit(HG2);

    Future<Matrix> f_HD1 = executorService.submit(HD1);
    Future<Matrix> f_HD2 = executorService.submit(HD2);

    Future<Matrix> f_BG1 = executorService.submit(BG1);
    Future<Matrix> f_BG2 = executorService.submit(BG2);

    Future<Matrix> f_BD1 = executorService.submit(BD1);
    Future<Matrix> f_BD2 = executorService.submit(BD2);

    // creation de la matrix finale avant le try
    Matrix matrix = new Matrix(0,0);

    while (!executorService.isTerminated()){
```

```
        System.out.println("attente");

        //On attend un certain temps en espérant que les threads seront tous
finis. La boucle est là pour éviter les erreurs
        try {
            Thread.sleep(500);
        } catch (InterruptedException e){}

        //On récupère les solutions en utilisant les objets Future.
        try {
            Matrix M_HG1 = f_HG1.get();
            Matrix M_HG2 = f_HG2.get();

            Matrix M_HD1 = f_HD1.get();
            Matrix M_HD2 = f_HD2.get();

            Matrix M_BG1 = f_BG1.get();
            Matrix M_BG2 = f_BG2.get();

            Matrix M_BD1 = f_BD1.get();
            Matrix M_BD2 = f_BD2.get();

            //Matrix matrixFinal = recomposerMatrix1(M_HG1, M_HG2, M_HD1,
M_HD2, M_BG1, M_BG2, M_BD1, M_BD2);
            //Matrix matrixFinal = recomposerMatrix2(M_HG1, M_HG2, M_HD1,
M_HD2, M_BG1, M_BG2, M_BD1, M_BD2);
            //this.data = matrixFinal.data;
            matrix = recomposerMatrix2(M_HG1, M_HG2, M_HD1, M_HD2, M_BG1,
M_BG2, M_BD1, M_BD2);
        }
        catch (ExecutionException e){}
        catch (InterruptedException e){}
        executorService.shutdown();

    }
    return matrix;
}
```

La méthode decomposition :

```
public ArrayList<Matrix> decomposition(){
    ArrayList<Matrix> lesSousMatrix = new ArrayList<>();
    Matrix a = this;
    //double[][] donneesB = new double[b.M/2][b.N/2];
    double[][] donneesA = new double[a.M/2][a.N/2];
    //System.out.println("boucle numero : 1");
    for(int i=0;i<a.M/2;i++) {
        for (int j = 0; j < a.N / 2; j++) {
            //System.out.println("i = "+ i+ " j = " +j );
            donneesA[i][j] = a.data[i][j];

        }
```

```
    }
    //System.out.println("boucle numero : 2");
    lesSousMatrix.add(new Matrix(donneesA));
    for(int i=0;i<a.M/2;i++) {
        for (int j = a.N/2; j < a.N ; j++) {
            //System.out.println("i = "+ i+ " j = " +j );
            donneesA[i][j-a.N/2] = a.data[i][j];

        }
    }
    //System.out.println("boucle numero : 3");
    lesSousMatrix.add(new Matrix(donneesA));
    for(int i=a.M/2;i<a.M;i++) {
        for (int j = 0; j < a.N / 2; j++) {
            //System.out.println("i = "+ i+ " j = " +j );
            donneesA[i-a.M/2][j] = a.data[i][j];

        }
    }
    //System.out.println("boucle numero : 4");
    lesSousMatrix.add(new Matrix(donneesA));
    for(int i=a.M/2;i<a.M;i++) {
        for (int j = a.N/2; j < a.N ; j++) {
            //System.out.println("i = "+ i+ " j = " +j );
            donneesA[i-a.M/2][j-a.N/2] = a.data[i][j];

        }
    }
    lesSousMatrix.add(new Matrix(donneesA));
    return lesSousMatrix;
}
```

La méthode recomposerMatrix2

```
private Matrix recomposerMatrix2(Matrix M_HG1, Matrix M_HG2, Matrix M_HD1,
Matrix M_HD2, Matrix M_BG1, Matrix M_BG2, Matrix M_BD1, Matrix M_BD2) {
    Matrix M_HG = M_HG1.plus(M_HG2);
    Matrix M_HD = M_HD1.plus(M_HD2);
    Matrix M_BG = M_BG1.plus(M_BG2);
    Matrix M_BD = M_BD1.plus(M_BD2);
    int ligne=M_HG.M , colonne = M_HG.N;
    double[][] donneeFinal = new double[ligne*2][colonne*2];
    for(int i = 0; i < ligne;i++) {
        for (int j = 0; j < colonne; j++) {
            //System.out.println("i = " +i + " j = " + j);
            donneeFinal[i][j] = M_HG.data[i][j];
            donneeFinal[i][j+colonne] = M_HD.data[i][j];
            donneeFinal[i+ligne][j] = M_BG.data[i][j];
            donneeFinal[i+ligne][j+colonne] = M_BD.data[i][j];
        }
    }
    //System.out.println(Arrays.deepToString(donneeFinal));
    return new Matrix(donneeFinal);
```

```
}
```

Q3

Q1 = M*N² VS Q2 = ((M*N²)/8)*8

La complexité des 2 méthodes est similaire mais si l'on possède une machine avec plusieurs coeurs il serait plus intéressant de choisir la deuxième méthode car les threads pouvant s'exécuter simultanément elle sera plus rapide.