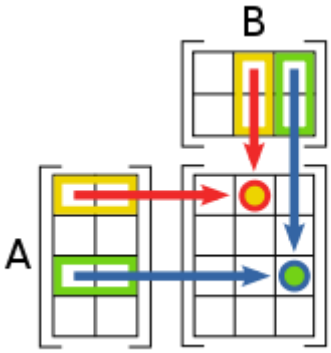
Exercice : Multiplication de matrices

On ne peut multiplier des matrices que si le nombre de lignes de la première matrice correspond au nombre de colonnes de la seconde matrice. Chaque cellule $C_{i,j}$ résultante est calculée selon la formule suivante :

$$C_{i,j} = \Sigma\ A_{i,k} * B_{k,j}$$



- Ecrire un programme qui renvoie le produit de deux matrices

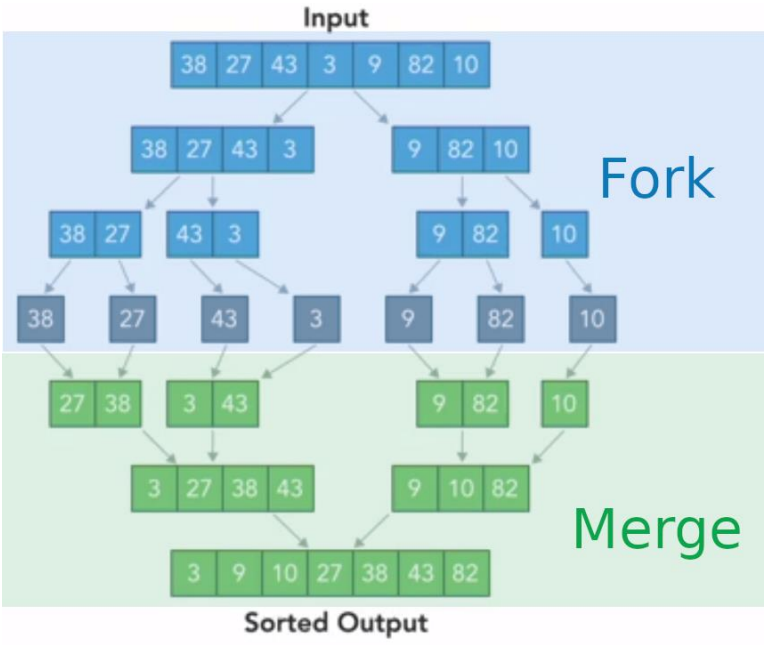
Chaque cellule de la matrice résultante peut être calculée indépendamment des autres.

- Améliorer le programme pour paralléliser les calculs
- Comparer les temps de calculs entre les deux approches sur une matrice 2000x2000
- Comparer les temps de calculs entre les deux approches sur une matrice 50x50
- Que peut-on conclure ?

```
public class MatrixProd {

    private int[][] A, B;
    private int numRowsA, numColsA, numRowsB, numColsB;

    public MatrixProd(int[][] A, int[][] B) {
        //TODO: vérifier la compatibilité des matrices pour la multiplication
        //TODO: Renvoyer une erreur sinon
    }

    public int[][] seqProduct() {
        //TODO: implanter la version séquentielle du produit de matrices
        return null;
    }

    public int[][] parProduct() {
        //TODO: implanter la version parallèle du produit de matrices
        //TODO: tirer profit du parallélisme de vos processeurs
        return null;
    }

    public static void main(String[] argv) {
        //TODO: Utiliser la classe Random pour générer une matrice 2000x2000 d'entiers aléatoirement
        //TODO: Tester et mesurer la performance des deux approches (séquencielle et parallèle)
    }
}
```

Exercice : Algorithme de tri – Merge Sort

L'algorithme du Merge Sort est une illustration parfaite de l'approche « Divide and Conquer ». Il est de fait un candidat idéal à la parallélisation.

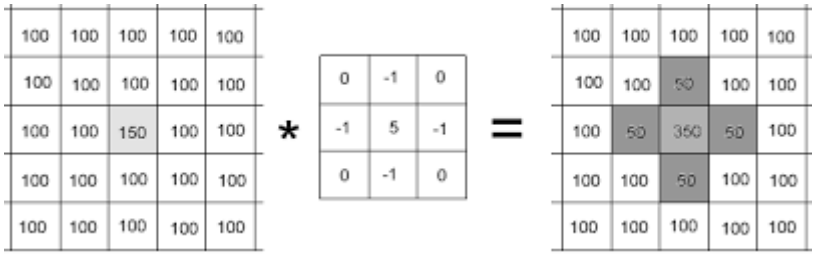


- Ecrire un programme qui implante l'algorithme du Merge Sort
- Ecrire une version qui tire parti de l'approche Divide and Conquer supportée par la classe ForkJoinPool
- Tester et comparer les performances des deux approches
    - Sur un tableau de 1000 éléments
    - Sur un tableau de 100.000.000 d'éléments
    - Que peut-on en conclure ?

Exercice : Traitement d'images

En traitement d'images, un certain nombre d'effets sont obtenus à partir de produits de convolution.

Il s'agit de calculer une image à partir d'une image d'origine en utiliser une matrice de convolution (appelé noyau ou parfois ondelette). Ainsi la valeur de chaque pixel (i,j) de l'image transformée est calculée à partir du pixel(i, j) de l'image source et de la matrice de convolution, comme ci :



La page [wikipedia] donne un certain nombre d'exemples de matrice de convolution.

Chaque pixel est codé sur 24 ou 32 bits, selon que l'image possède une couche alpha ou non. Par exemple les images compressées en PNG possèdent une couche alpha, absente des images compressées en JPG.

Dans cet exercice, il vous est demandé d'écrire une classe java qui tire pleinement parti de vos processeurs pour :

1. Télécharger une image du net - https://bellard.org/bpg/lena30.jpg
2. Transformer l'image en tableau 3 dimensions de l'image (largeur * hauteur * couches_rgb)
3. Générer une nouvelle image en appliquant un pipeline de deux effets : une amélioration de la netteté, suivie d'un flou (Box Blur)
4. Pour les points se trouvant en périphérie, considérer l'image comme un tore.

```java
package fr.cenotelie.formation;

import javax.imageio.ImageIO;
import java.awt.image.BufferedImage;
import java.io.*;
import java.net.URL;

public class Lenna {

    private int[][][] image, tImage;

    public Lenna() {
        loadImageFromURL("https://bellard.org/bpg/lena30.jpg");
    }

    private void loadImageFromURL(String url) {
        //TODO: Charger l'image depuis l'URL passé en paramètre
        //TODO: Transformer l'image en matrice largeur x hauteur x couches_RGB
    }

    public void transform() {
        // TODO: appliquer les filtres netteté et flou en parallèle
        // TODO: tImage = transformée de image
    }

    /**
     * Sauvegarde l'image au format JPG
     * @param name nom du fichier sans l'extension
     */
    public void saveImage(String name) {
        File f = new File("resources/" + name + ".jpg");
        BufferedImage buff = new BufferedImage(tImage.length, tImage[0].length, BufferedImage.TYPE_INT_RGB);
        int px, red, green, blue;
        for (int w = 0; w < buff.getWidth(); w++) {
            for (int h = 0; h < buff.getHeight(); h++) {
                red = tImage[w][h][0];
                green = tImage[w][h][1];
                blue = tImage[w][h][2];
                px = (red << 16) | (green << 8) | blue;
                buff.setRGB(w, h, px);
            }
        }
        try {
            ImageIO.write(buff, "jpg", f);
        } catch (IOException e) {
            e.printStackTrace();
        }
```

```
    }

    public static void main(String[] argv) {
        Lenna l = new Lenna();
        l.transform();
        l.saveImage("lenna2");
    }

}
```