Compte-rendu du TP D1 de l'UE « Programmation parallèle »

Vendredi 16 mars 2018

Binôme : Mme Mélanie BASSETTE & M. Johan MABILY

Auto-évaluation

Nous pensons avoir 10 car les résultats sont conformes, et notre code nous paraît bien conçu.

Pédagogie

Pour nous l'intérêt a principalement été d'apprendre à utiliser les librairies java facilitant la manipulation des threads. De plus, cela nous a permis d'être confronté à une situation où la séparation thread/tâche est importante.
Nous avons mis un peu de temps à comprendre l'ensemble des modifications nécessaires à la mise en place du service de complétion.

Proposition d'application

Une application intéressante serait celle de l'intégration des systèmes d'informations. Il s'agit de créer des tables contenant les informations de fichiers CSV par exemple et d'effectuer une requête sur ces tables afin de répondre à une demande de l'utilisateur, et ce à chaque nouvelle demande de celui-ci afin de lui offrir des données réactualisées. Il faut créer une table par fichiers CSV qui peuvent contenir des millions de lignes et une centaine de colonnes. Il serait intéressant d'utiliser un pool de thread pour créer l'ensemble des requêtes de création de tables. On aurait un tableau de classes héritant de Callable<String> qu'on instancierait et soumettrai au pool de thread. Lorsque ce dernier aurait terminé, on concatènerait dans l'ordre les résultats des tâches. Il faudrait poursuivre de la même façon avec le même pool pour les fichiers CSV suivants.

Codage

Classe Mandelbrot

```
public class Mandelbrot
{
    final static Color noir =  new Color(0, 0, 0);
    final static Color blanc =  new Color(255, 255, 255);
    final static int taille = 500;
    final static Picture image = new Picture(taille, taille);

    final static double xc = -.5 ;
    final static double yc = 0 ;
    final static double region = 2;

    final static int max = 20_000;

    public static void main(String[] args)
    {
        final long startTime = System.nanoTime();
        final long endTime;

        ExecutorService executeur = Executors.newFixedThreadPool(4);
        CompletionService<Long> ecs = new
           ExecutorCompletionService<Long>(executeur);
        int i = 0;
        while (i < taille)
        {
           ecs.submit(new TraceLigne(i));
           i++;
        }

        for (int k = 0; k < taille; k++)
        {
           try
           {
                ecs.take();
           }
           catch(InterruptedException e){}
        }

        endTime = System.nanoTime();
        final long duree = (endTime - startTime) / 1_000_000 ;
        System.out.println("Durée = " + duree + " ms.");
        image.show();
    }
```

```
    public static void colorierPixel(int i, int j)
    {
        double a = xc - region/2 + region*i/taille;
        double b = yc - region/2 + region*j/taille;
        // Le pixel (i,j) correspond au point (a,b)
        if (mandelbrot(a, b, max)) image.set(i, j, noir);
        else image.set(i, j, blanc);
    }

    public static boolean mandelbrot(double a, double b, int max)
    {
        double x = 0;
        double y = 0;
        for (int t = 0; t < max; t++)
        {
            if (x*x + y*y > 4.0) return false;
            double nx = x*x - y*y + a;
            double ny = 2*x*y + b;
            x = nx;
            y = ny;
        }
        return true; // Le point (a,b) est noir
    }

    public static class TraceLigne implements Callable<Long>
    {
      int line;

      public TraceLigne(int line)
      {
            this.line = line;
      }

      public Long call()
      {
             for (int j = 0; j < Mandelbrot2.taille; j++)
             {
                 Mandelbrot2.colorierPixel(line, j);
             }
             return (long)0;
      }
    }
}
```