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

Lundi 29 janvier 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 des threads pour passer d'un programme séquentiel à un programme parallèle et ainsi optimiser le temps de calcul de certains algorithmes. Nous n'avons pas rencontré de difficultés particulières.

Relevé de mesures observées

Indiquez sous la forme du tableau ci-dessous : le nombre de tirages considéré dans le programme séquentiel donné afin d'observer un temps de calcul d'environ 20 secondes ; le temps de calcul effectivement observé pour ce programme séquentiel sur votre machine ; le temps de calcul observé (pour le même nombre de tirages et sur la même machine) pour le programme avec 10 threads produit ; le gain de temps obtenu.

Nombre de tirages	400 000 000
Temps de calcul séquentiel (en ms.)	20491
Temps de calcul parallèle (en ms.)	6477
Gain de temps obtenu	14014

Votre avis

Nous attendions un temps de calcul parallèle de l'ordre de 10^3 pour un calcul séquentiel de l'ordre de 10^4 car nous utilisons 10 threads donc le temps de calcul devrait être divisé par 10. Seulement, nous avons obtenu un temps un peu plus long, de l'ordre de 10^4. Cela peut s'expliquer par la puissance de calcul de l'ordinateur.

Code produit

Classe Calcul_PiSurQuatre

```
import java.lang.Math;

public class Calcul_PiSurQuatre{

    public static void main(String[] args) {
        int nombreDeTirages = 400_000_000 ;
        double resultat = 0; // init à 0

        if (args.length>0) {
            try { nombreDeTirages = Integer.parseInt(args[0]); }
            catch(NumberFormatException nfe) {
                System.err.println
                    ("Usage : java Calcul_PiSurQuatre <nb de tirages>");
                System.err.println(nfe.getMessage());
                System.exit(1);
            }
        }

        System.out.println("Nombre de tirages: " + nombreDeTirages);
        final long startTime = System.nanoTime();
        final long endTime;

        // Threads work
        int nbThreads = 10;
        MyThread[] thread = new MyThread[nbThreads];
        for (int i = 0; i < nbThreads; i++)
        {
            thread[i] = new MyThread(nombreDeTirages/nbThreads);
            thread[i].start();
        }
        for (int j = 0; j < nbThreads; j++)
        {
            try{
                thread[j].join();
            }
            catch (Exception e) {System.out.println(e.getMessage());}
        }

        // Calcul du résultat
        for (MyThread th : thread)
        {
            resultat += th.getTiragesDansLeDisque();
        }
        resultat = (double) resultat / nombreDeTirages;

        System.out.println("Estimation de Pi/4: " + resultat) ;
        System.out.println("Pourcentage d'erreur: "
                           + 100 * Math.abs(resultat-Math.PI/4)/(Math.PI/4)
                           + " %");

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

Classe MyThread

```
import java.util.Random;

public class MyThread extends Thread
{
    private int nbTirages;
    private int tiragesDansLeDisque;

    public MyThread(int nbTirages) {
        this.nbTirages = nbTirages;
        this.tiragesDansLeDisque = 0;
    }

    public int getNbTirages() {
        return nbTirages;
    }

    public int getTiragesDansLeDisque() {
        return tiragesDansLeDisque;
    }

    public void run() {
        double x, y;
        Random r = new Random();

        for (int i = 0; i < nbTirages; i++) {
            x = r.nextDouble();
            y = r.nextDouble();
            if (x * x + y * y <= 1) tiragesDansLeDisque++ ;
        }
    }
}
```