Práctica #0

Factoriales con Threads en Java

Pepper Pots (A01166611), Anthony Stark (A01160611)

18 de enero, 2017.

Tabla de contenido

Este reporte fue elaborado para el curso de *Programación multinúcleo* del Tecnológico de Monterrey, Campus Estado de México.

1. Introducción

Según [Wolfram], el factorial de un número N se define de manera iterativa como:

$$N! = 1 \times 2 \times 3 \times \ldots \times N$$

En esta práctica se calculó el factorial de un número muy grande en Java utilizando la clase `java.math.BigInteger` (ver el API de Java en [Oracle]). El objetivo consistió en resolver este problema de manera secuencial y usando *threads* de Java para obtener una solución paralela.

Hardware y software utilizado

Los programas se probaron en una computadora de escritorio con las siguientes características:

- Procesador Intel Core i7-4770 de 3.40GHz con cuatro núcleos y ocho *hyperthreads*.
- 7.7 GiB de memoria RAM.
- Sistema operativo Ubuntu 14.04, kernel de Linux 3.13.0-107 de 64 bits.
- Compilador Java 1.8.0_51 de Oracle.

2. Solución secuencial

El siguiente listado muestra un programa completo en Java que calcula de forma secuencial el factorial de 250,000:

SequentialFactorial.java

```
/*-----------------------------------------------------------------
 * Práctica 0: Factoriales en Java versión secuencial
 * Fecha: 18-Ene-2017
 * Autores:
 *          A01166611 Pepper Pots
 *          A01160611 Anthony Stark
 *-----------------------------------------------------------------*/

package mx.itesm.cem.pmultinucleo;

import java.math.BigInteger;

public class SequentialFactorial {

    public static void main(String[] args) throws InterruptedException {

        final int n = 250000;

        long timeStart = System.currentTimeMillis();
        BigInteger result = BigInteger.ONE;

        for (int i = 2; i <= n; i++) {
            result = result.multiply(BigInteger.valueOf(i));
        }

        long timeEnd = System.currentTimeMillis();

        System.out.printf("Resultado = %d, Tiempo = %.4f%n", result.bitCount(),
            (timeEnd - timeStart) / 1000.0);
    }
}
```

Dado que el factorial de 250,000 es extremadamente grande para imprimirlo (más de un millón doscientos mil dígitos), se utilizó el método `bitCount` para solo contar el número de bits igual a uno que tiene el valor binario del resultado.

Esta es la salida del programa:

```
Resultado = 1936401, Tiempo = 18.5650
```

3. Solución paralela

La solución paralela en Java involucra usar dos *threads*. El primer *thread* se encarga de calcular la primera mitad de las multiplicaciones: 1 × 2 × 3 × ... × 125,000. El segundo *thread* se encarga de calcular la otra mitad de las multiplicaciones: 125,001 × 125,002 × 125,003 × ... × 250,000. Una vez que terminan ambos *threads*, se multiplican los resultados de ambos para obtener el resultado

final. El siguiente listado contiene el programa completo:

ParallelFactorial.java

```
/*-----------------------------------------------------------------
 * Práctica 0: Factoriales en Java versión paralela con threads
 * Fecha: 18-Ene-2017
 * Autores:
 *          A01166611 Pepper Pots
 *          A01160611 Anthony Stark
 *-----------------------------------------------------------------*/

package mx.itesm.cem.pmultinucleo;

import java.math.BigInteger;

public class ParallelFactorial implements Runnable {

    private int start, end;
    private BigInteger result = BigInteger.ONE;

    public ParallelFactorial(int start, int end) {
        this.start = start;
        this.end = end;
    }

    @Override
    public void run() {
        for (int i = start; i <= end; i++) {
            result = result.multiply(BigInteger.valueOf(i));
        }
    }

    public static void main(String[] args) throws InterruptedException {
        final int n = 250000;

        long timeStart = System.currentTimeMillis();

        ParallelFactorial p1 = new ParallelFactorial(2, n / 2);
        ParallelFactorial p2 = new ParallelFactorial(n / 2 + 1, n);
        Thread t1 = new Thread(p1);
        Thread t2 = new Thread(p2);
        t1.start();
        t2.start();
        t1.join();
        t2.join();
        BigInteger total = p1.result.multiply(p2.result);

        long timeEnd = System.currentTimeMillis();

        System.out.printf("Resultado = %d, Tiempo = %.4f%n", total.bitCount(),
```

```
            (timeEnd - timeStart) / 1000.0);
    }
}
```

El programa produce esta salida:

```
Resultado = 1936401, Tiempo = 6.4300
```

El *bit count* es el mismo que en la versión secuencial, por lo que podemos suponer que nuestra versión paralela produce el resultado correcto.

4. Resultados

A continuación se muestran los tiempos de ejecución de varias corridas de los dos programas:

Tabla 1. Tiempos de ejecución del factorial secuencial

# de corrida	Tiempo T_1 (segundos)
1	18.5650
2	19.1660
3	19.3860
4	18.8000
5	18.8090
Media aritmética	18.9452

Tabla 2. Tiempos de ejecución del factorial paralelo

# de corrida	Tiempo T_2 (segundos)
1	6.4300
2	6.9820
3	6.7190
4	6.5340
5	6.7180
Media aritmética	6.6766

A partir de las medias aritméticas calculadas, el *speedup* obtenido en un CPU que utiliza dos de sus núcleos (un *thread* corriendo en cada núcleo) es:

$$S_2 = T_1 / T_2 = 18.9452 / 6.6766 = 2.8376$$

El *speedup* obtenido es muy bueno, incluso superando al *speedup* ideal. La mejora obtenida en el tiempo compensa la complejidad adicional asociada al uso de *threads* en Java.

5. Agradecimientos

Se agradece a Steve Rogeres por sugerir usar LibreOffice Calc para calcular los promedios de los tiempos de ejecución.

6. Referencias

- [Oracle] Oracle Corporation. *Java Platform, Standard Edition 8 API Specification* http://docs.oracle.com/javase/8/docs/api/ (Consultada el 18 de enero, 2017).
- [Wolfram] Wolfram MathWorld. *Factorial* http://mathworld.wolfram.com/Factorial.html (Consultada el 18 de enero, 2017).