COMPUTACIÓN CONCURRENTE

PRÁCTICA 1

Prof. Manuel Alcántara Juárez
manuelalcantara52@ciencias.unam.mx

Leonardo Hernández Cano
leonardohernandezcano@ciencias.unam.mx

Ricchy Alaín Pérez Chevanier
alain.chevanier@ciencias.unam.mx

Fecha límite de entrega: 30 de Agosto de 2019 a las 23:59:59pm.

1. Objetivo

El objetivo de esta práctica es que el estudiante practique el uso de hilos de ejecución en java por medio de algunos ejercicios de programación concurrente. Por otro lado analizaremos de manera empírica cómo cambia el tiempo de ejecución entre una solución secuencial y una concurrente para el mismo problema utilizando el mismo algoritmo.

2. Introducción

En esta práctica se revisan algunos ejemplos para mostrar la ventaja de usar programas concurrentes. Para medir la mejora obtenida al usar operaciones de forma concurrente, comparamos con la mejora obtenida a partir de la *Ley de Amdahl*.

En cada ejercicio se hará una comparación entre los tiempos de ejecución de la forma secuencial y la forma concurrente. Además, la aceleración obtenida en la solución concurrente será comparada con lo que nos dice la *ley de Amdahl*. Para hacer las comparaciones, en cada ejercicio crearás una tabla con los siguientes datos:

- Número de hilos
- Aceleración teórica
- Aceleración obtenida
- Porcentaje de código paralelo

donde anotarás la aceleración calculada con la ley de Amdahl, la aceleración que obtuvo tu programa concurrente[1] y el porcentaje de código paralelo de tu programa, también calculado con la ley de Amdahl. Incluye esta información en un archicho `REAMDE.txt` adjusto con tu solución de código.

3. Base de Código

En esta práctica trabajarás con una base de código construida con Java 8 y Maven 3. El código es compatible con versiones más nuevas del lenguaje, incluído java 12. Para hacer que el código funcione con otra versión del lenguaje tienes que editar el valor de la etiqueta `<java.version>` que se encuentra en el archivo `pom.xml`, por ejemplo si quires utilizar java 11, tu configuración tiene que verse así:

```
<properties>
  ...
  <java.version>11</java.version>
  ...
</properties>
```

En el código que recibirás junto con este documento, prodrás encontrar la clase `App` que tiene un método *main* que puedes ejecutar como cualquier programa escrito en *Java*. Para eso primero tienes que empaquetar la aplicación y después ejecutar el *jar* generado. En una terminal y posicionando el directorio de trabajo en el directorio raíz de la práctica, ejecuta los siguientes comandos:

```
$ mvn package
...
...
$ java -jar target/practica01-1.0.jar
```

Recuerda que para ejecutar todas las pruebas unitarias es necesario ejecutar el siguiente comando:

[1] La ejecución de las pruebas unitarias te darán los tiempos de ejecución de cada caso, por lo que para obtener un resultado aproximado puedes ejecutarlas veces y sacar un promedio de los valores obtenidos.

```
$ mvn test
```

Para ejecutar solamente las pruebas unitarias contenidas en una clase de pruebas, debes de ejecutar un comando como el siguiente:

```
$ mvn -Dtests=AppTest test
```

4. Ejercicios

4.1. Números Primos

En este problema deseamos implementar una función que decida si un número entero es primo o no. En la clase `PrimeNumberCalculator` tendrás que implementar el método `isPrime(int n)` que decide si n es un número primo. La clase `PrimeNumberCalculatorTest` contiene las pruebas unitarias que tendrás que pasar para poder verificar que tu implementación es correcta. El constructor de la clase `PrimeNumberCalculator` recibe como argumento el número de hilos que debe de utilizar el método `isPrime(int n)` para resolver el problema.

El algoritmo que tienes que utilizar para tu solución es el siguiente; suponiendo que tenemos que utilizar M hilos, dado un número entero N, parte el segmento $[2, N-1]$ en M partes iguales y asigna cada una de esos subsegmentos a un *hilo*. Ahora cada *hilo* debe de averiguar si algún número del segmento que le fue asignado divide a N. Finalmente hay que reportar si alguno de los *hilos* encontró algún número que divide a N en su segmento asignado, en cuyo caso el método regresa `false`, caso contrario regresa `true`.

4.2. Encuentra el mínimo

Dada una matriz de enteros, encuentra su elemento mínimo. En la clase `MatrixUtils` necesitas completar la implementación del método `findMinimum(int[][] matrix)` que encuentra el elemento mínimo en *matrix*. Para verificar tu implementación tendrás que pasar las pruebas unitarias definidas en la clase `MatrixUtilsTest`. El constructor de la clase `MatrixUtils` recibe como argumento el número de *hilos* que utiliza el método `findMinimum(int[][] matrix)` para resolver el problema.

El algoritmo que debes de utilizar para resolver este problema es el siguiente; dados N hilos, separa la matriz en N o más submatrices de tamaños similares y asigna una o mas matrices a cada *hilo*. Ahora cada *hilo* reporta el mínimo de la

matriz que trabajó en un arreglo, al final en una pasada (de forma secuencial) obtienes el mínimo de ese arreglo y se reporta como el mínimo de la matriz.[2].

[2]Cuida no crear una condición de carrera al momento de reportar el valor mínimo general, por eso te sugerimos utilizar un arreglo en el que cada hilo reporta de manera independiente su resultado en una posición fija y después encontrar el valor mínimo de ese arreglo de manera secuencial