COMPUTACIÓN CONCURRENTE

PRÁCTICA 1

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

Alejandro Tonatiuh Valderrama Silva
at.valderrama@ciencias.unam.mx

José de Jesús Barajas Figueroa
jebarfig21@ciencias.unam.mx

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

Fecha de Entrega: 19 de Marzo de 2021 a las 23:59

1. Objetivo

Revisar algunos ejemplos para mostrar la ventaja(o desventaja) de usar programas concurrentes. Para medir la mejora obtenida al usar operaciones de forma concurrente, se compara con la mejora obtenida a partir de la `Ley de Amdahl`.

2. Indicaciones generales

Los ejercicios describen un problema a resolver mediante un programa que puede implementarse de forma secuencial y concurrente sin necesidad de primitivas de sincronización. En cada ejercicio se hará una comparación entre los tiempos de ejecución de la forma secuencial(1 hilo) y la forma concurrente(2, 4 y 8 hilos). Además, se analizará la aceleración obtenida en la solución concurrente, la cual será comparada contra lo que nos dice la `Ley de Amdahl`.

Para hacer las comparaciones, en cada ejercicio realizarás una tabla y una gráfica.

La tabla debe de tener el siguiente formato:

# hilos	Aceleración teórica	Aceleración obtenida	% Código paralelo

Donde anotarás la aceleración calculada con la `Ley de Amdahl`, la aceleración que obtuvo tu programa concurrente y el porcentaje de código paralelo de tu programa, también calculado con la `Ley de Amdahl`.

La gráfica debe de comparar el tiempo de ejecución del algoritmo para cada configuración de hilos disponibles [1,2,4,8], generando una gráfica para cada tamaño de entrada. Es decir, en el eje X vamos a representar el número de hilos disponible y el eje Y el tiempo en segundos obtenido al ejecutar la prueba. La gráfica contendrá 3 funciones, una para cada tamaño de entrada. Agregar el número de ejecuciones que hicieron para obtener el promedio de ejecución y el % paralelo teórico.

`NOTA`: Si la versión concurrente es más lenta argumenta algunas de las posibles razones de por qué está sucediendo de esa manera.

3. Desarrollo

En esta práctica trabajarás con una base de código construida con Java 9[1] y Maven 3, también proveemos pruebas unitarias escritas con la biblioteca `Junit 5.5.1` que te darán retrospectiva inmediata sobre el correcto funcionamiento de tu implementación[2] .
Para ejecutar las pruebas unitarias necesitas ejecutar el siguiente comando:

```
$ mvn test
```

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

```
$ mvn -Dtest=MyClassTest test
```

En el código que recibirás la clase `App` tiene un método *main* que puedes ejecutar como cualquier programa escrito en *Java*. Para eso primero tienes que empaquetar la aplicación y finalmente ejecutar el jar generado. Utiliza un comando como el que sigue:

[1] De nuevo puedes utilizar cualquier versión de java que sea mayor o igual a Java 9 simplemente ajustando el archivo pom.xml

[2] Bajo los casos que vamos a evaluar, mas estas no aseguran que tu es implementación es correcta con rigor formal

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

4. Problemas

4.1. Multiplicación de matrices

Es posible realizar el producto de dos matrices usando varios hilos de forma independiente. Primero recordemos cómo se obtiene el producto de dos matrices:

Sean $A = (a_{ij})$ una matriz de $n \times r$ y $B = (b_{ij})$ una matriz de $r \times m$. La matriz producto $C = AB$ es una matriz de tamaño $n \times m$ que está definida por las entradas

$$c_{ij} = \sum_{k=1}^{r} a_{ik} \cdot b_{kj},$$

es decir, el elemento c_{ij} es igual al producto punto entre la i-ésima fila de A y la j-ésima columna de B. Directamente de esta definición obtenemos un algoritmo secuencial para calcular AB, donde el cálculo principal es el producto punto, que se vería similar al siguiente código:

```
for(k = 0; k < r; k++)
    C[i][j] += A[i][k] * B[k][j]
```

Para calcular el producto de forma concurrente usando varios hilos de ejecución, podemos asignar cada hilo a una fila de la primer matriz y ejecutar la operación por las segunda matriz de manera independiente a otro hilo.

Hagamos un ejemplo con dos hilos:

$$C = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix} = \begin{bmatrix} 2 & 6 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 5 & 3 \end{bmatrix}$$

El primero se encarga de la fila compuesta por las entradas c_{11}, c_{12}, el segundo hilos de las entradas c_{21}, c_{22}, sin embargo no necesariamente se debe cumplir que el número de hilos corresponda con el número de filas.

Especificación del programa

Crea un programa que obtenga el **cuadrado** de una matriz A de tamaño $n \times n$ (puedes suponer que n es par!).

1. En la forma secuencial se calculan las entradas c_{ij} de A^2 usando tres ciclos anidados.

2. En la forma concurrente usarás **n** hilos de ejecución. Para esto hay que dividir la matriz en **n** bloques, y cada hilo obtendrá los coeficientes de un bloque. Cada hilo puede realizar su cálculo de forma independiente, sin tener que esperar alguna operación de otro hilo.

3. Para comparar los tiempos de ejecución deberás utilizar 1, 2, 4 y 8 hilos, graficar resultados.

Recuerda que debes obtener el tiempo de ejecución para poder elaborar tus gráficas.

Compara tus soluciones

Apóyate de las pruebas unitarias para ir corroborando que tus resultados sean correctos Anexa tu comparación con la ley de Amdahl.

4.2. Sopa de letras

En este ejercicio tendrás que escribir un algoritmo para resolver el problema de la sopas de letra. A partir de un tablero como el siguiente

```
O I U Q W E J A A D C
A S D A S D C W E E H
O I M O I O D P F I O
I E P E R R O V N C L
C N J D K D M C L D A
A R O D A T U P M O C
```

Y una lista de palabras como esta:

```
PERRO
COMPUTADORA
HOLA
ROCA
KEMS
```

Tu algoritmo deberá encontrar la posición donde empieza cada palabra, junto con la dirección que siguen las letras para formarla.

Para indicar las direcciones usaremos las letras N, S, E, O, NE, NO, SE, SO (de Norte, Sur, Este, etc.). Con estas convenciones, la solución del ejemplo anterior se vería así:

```
COMPUTADORA (5,10) O
HOLA (1,10) S
KEMS (4,4) NO
PERRO (3,2) E
ROCA (3,4) NE
```

Para encontrar las palabras vamos a utilizar el siguiente algoritmo:

1. Vamos a hacer cuatro barridos sobre la matriz uno para las columnas, uno para los renglones, uno para las diagonales inclinadas hacia la derecha y otra para las diagonales inclinadas hacia la izquierda. En cualquier caso no referiremos al segmento de la matriz que estamos analizando como renglón.

2. Lo que tenemos que hacer es que el renglón actual lo convertimos en una cadena, y en esa cadena por medio de expresiones regulares buscamos cada palabra. Es importante considerar que las palabras pueden estar escritas directamente o al revés, por ejemplo el renglón `I E O R R E P V N C L` contiene la palabra perro pero al revés.

3. Para paralelizar este algoritmo dividimos los renglones que va a examinar cada posible tarea entre el número de hilos que vamos a utilizar para realizar la ejecución, y hacemos los siguiente: el hilo t_i va a procesar todos los renglones $t_i * T$ donde T es el número total de hilos utilizados para la ejecución.

4. Cada hilo crea sus propio conjunto de resultados, y lo deposita en el arreglo de resultados.

5. Finalmente el hilo principal espera que los demás hilos terminen y colectará los resultados producidos por cada hilo, los ordenará lexicográficamente y regresará eso como respuesta completa de la búsqueda.

Nota: El orden lexicográfico implica que los resultados de la búsqueda primero se ordenan alfabéticamente y luego por la posición en la que inician.

En la base del código, tendremos 2 elementos para resolver este problema, primero una interfaz llamada `WordSearch` con la definición de la firma del método que tenemos que implementar para resolver el problema, y luego una clase llamada

`MultiThreadedWordSearch` que implementa dicha interfaz y que al momento de ser instanciada toma como parámetro el número de hilos que utilizará para realizar la ejecución.

Comparación de las ejecuciones

Ejecuta las pruebas unitarias y con ellas podrás encontrar los tiempos de ejecución obtenidos utilizando un número distinto de hilos. Grafica los tiempos de ejecución y haz la comparación con la ley de Amdahl como hiciste con el ejercicio anterior.

```
interface WordSearch {
  List<WordSearchAnswer>
  solve(char[][] board, List<String> words)
  throws InterruptedException;
}

class MultiThreadedWordSearch
    implements WordSearch {

  private int threads;

  public MultiThreadedWordSearch() {
    this(1);
  }

  /*
   * Recibe como parametro el numero de hilos que
   * utilizara para realizar la busqueda
   */
  public MultiThreadedWordSearch(int threads) {
    this.threads = threads;
  }
  ...
}
```