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: Marzo de 2021 a las 23:59

Objetivo

El objetivo de esta práctica es 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*.

Indicaciones generales

Los ejercicios se tratan acerca de resolver un problema mediante un programa que puede implementarse de forma secuencial o concurrente. 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, 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 realizarás una tabla de la siguiente forma

# 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. Finalmente, en una gráfica mostrarás la ventaja de la versión concurrente .

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.

Entrada. Tu programa se debe ejecutar así

```
$ java Matriz hilos archivo
```

Donde `archivo` es el nombre del archivo que contiene los coeficientes de A, y el valor de `hilos` es el numero de hilos que se van a ejecutar(1 hilo es la forma secuencial) .

El archivo de entrada incluye los coeficientes de A, con las columnas separadas por un espacio en blanco y las filas separadas por un salto de línea, como el siguiente ejemplo:

```
82 12 3
65 100 3
0 78 21
```

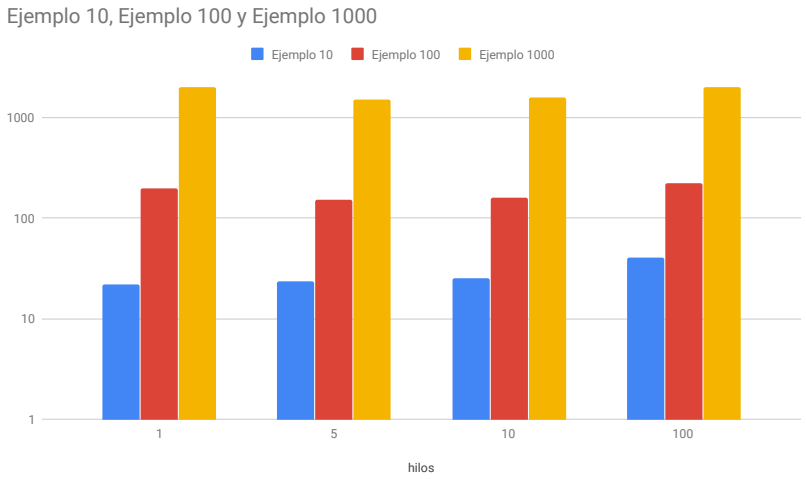

Figura 1: Ejemplo de comparación de tiempos. La opción secuencial es la de 1 hilo.

Salida. Tu programa debe guardar el resultado de A^2 en un archivo con nombre `producto.txt`. Antes del resultado deberá aparecer una línea indicando el tiempo que tardó en realizarse el producto, medido en milisegundos.

Compara tus soluciones

Usa los archivos de prueba incluidos en la carpeta `matrices` para probar tu programa. Se incluyen matrices de tamaños 10×10, 100×100 y 1000×1000. Para cada prueba repite el cálculo 20 veces y obtén el promedio del tiempo tardado, tanto en la opción secuencial como la opción concurrente. Con estos datos elabora una gráfica que compare tamaño de matriz frente a tiempo promedio. La gráfica debe incluir tanto los datos de la opción secuencial como los de la opción concurrente. En la figura **??** se muestra un ejemplo.

Anexa tu comparación con la ley de Amdahl.

2. Sopa de letras

En este ejercicio resolverás una sopa de letras. A partir de un texto 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
```

deberás encontrar la posición donde empieza la palabra encontrada junto con la dirección que siguen las letras para formar la palabra.

Podemos considerar a la sopa de letras como un arreglo bidimensional donde cada entrada guarda una letra. 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
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 y 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 que no hemos encontrado anteriormente en le crucigrama. Es importante considerar que las palabras pueden estar escritas directamente como están o al revés, por ejemplo la cadena $IEORREPVNCL$ 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 el algoritmo, y entonces 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.

La firma del método será la siguiente:

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

Lo más importante es que las respuestas deben de venir ordenadas alfabéticamente y si una palabra aparece más de una vez en el crucigrama, las posiciones deben de ordenarse lexicográficamente.

Hay que revisar el código de este ejercicio directamente en el proyecto que entregaremos junto con este documento para

1. En la opción secuencial tu programa irá revisando letra por letra en el arreglo, buscando en las ocho direcciones posibles.

2. La versión concurrente usará n hilos, deberas dividir el conjuntos de palabras en n subconjuntos, de preferencia de la misma cardinalidad, cada hilo va a tener asignado un subconjunto de palabras y va a tener que buscar en toda la matriz estas palabras, recuerden, una misma palabra no puede exisitir en dos subconuntos diferentes

3. Esta implementación es solo una idea de como pueden resolver el ejercicio, la implementación es libre, recuerden que debe funcionar con n hilos y ser concurrente

4. Deben de hacer una grafica de comparación de tiempos para 1, 2, 4 y 8 hilos

    - Hilo 0. Direcciones N, S.
    - Hilo 1. Direcciones E, O.
    - Hilo 2. Direcciones NE, SO.
    - Hilo 3. Direcciones NO, SE.

Entrada. Se ejecutará el programa de la siguiente forma

```
$ java Sopa hilos archivo_sopa archivo_palabras
```

donde `hilos` es como en los otros ejercicios, `archivo_sopa` es el archivo que contiene la sopa de letras y `archivo_palabras` es el archivo con las palabras a buscar.

Salida. El programa mostrará en pantalla la solución, que es una lista con las palabras junto con la ubicación y la dirección correspondientes. Al final se mostrará el tiempo de ejecución de la búsqueda.

Compara tus soluciones

Para hacer la comparación usa los ejemplos que se incluyen dentro de la carpeta `sopas`. De la misma manera que en el primer ejercicio, grafica los tiempos de ejecución y haz la comparación con la ley de Amdahl.