



## Carátula para entrega de prácticas

Facultad de Ingeniería

Laboratorio de docencia

# Laboratorios de computación salas A y B

*Profesor:* TISTA GARCÍA EDGAR

*Asignatura:* ESTRUCTURA DE DATOS Y ALGORITMOS II

*Grupo:* 5

*No de Práctica(s):* PRÁCTICA #12-13 – ALGORITMOS PARALELOS

*Integrante(s):* CARRILLO CERVANTES IVETTE ALEJANDRA

*No. de Equipo de  
cómputo empleado:* TRABAJO EN CASA

*No. de Lista o Brigada:* 09

*Semestre:* 2022 - 1

*Fecha de entrega:* 07 DICIEMBRE 2021

*Observaciones:*

**CALIFICACIÓN:** \_\_\_\_\_

## PRÁCTICA #12-13 – Algoritmos paralelos

**Objetivo.** El estudiante conocerá y aprenderá a utilizar algunas de las directivas de OpenMP para implementar algún algoritmo paralelo.

*Esta práctica se realizó en la distribución Ubuntu del sistema operativo Linux, con la finalidad de probar cada uno de los códigos que contiene la guía de estudios impartida por la Facultad, estos códigos serán en el lenguaje de programación C; mientras que, los ejercicios proporcionados por el profesor se realizaron en Visual Studio Code, estos códigos serán en el lenguaje de programación JAVA.*

### ACTIVIDADES PARA EL DESARROLLO DE LA PRÁCTICA

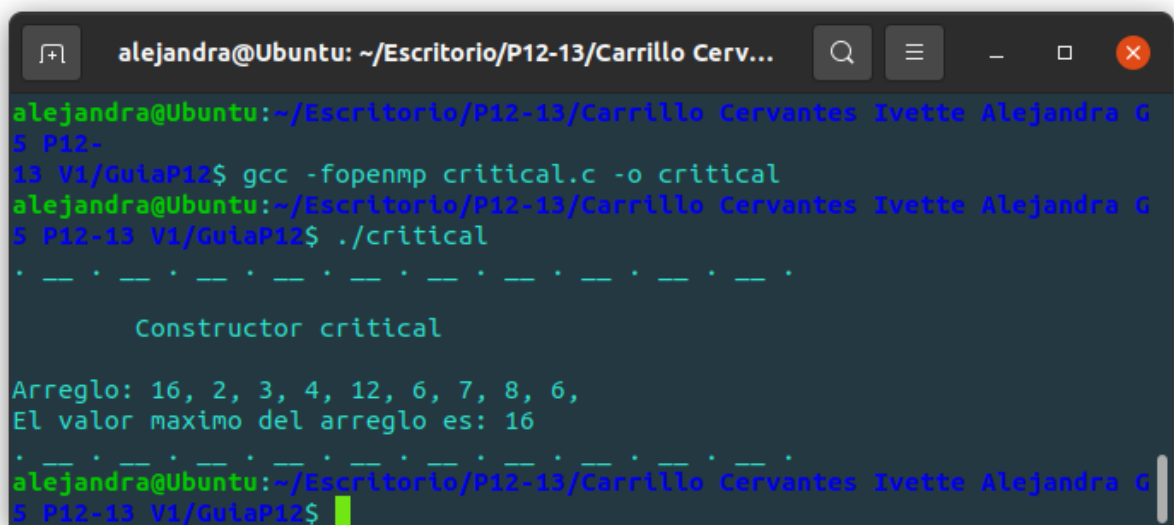
#### Parte 1. Ejercicios de la guía

##### **Actividades Práctica 12**

- Constructor critical

El constructor critical, es una directiva la cual permite que un segmento de código que contiene una secuencia no sea interrumpido por otros hilos, es decir, dentro de lo que se delimito por este constructor solo puede entrar un hilo a la vez con la finalidad de evitar una condición carrera. Este primer ejemplo tiene como función implementar dicho constructor, para ello se creo una función llamada “buscaMaximo” la cual tiene como objetivo buscar el valor máximo de un arreglo de enteros. Esta función se llama desde la función main y se le pasa como parámetros el arreglo y la longitud de este. Un aspecto importante acerca de esta función es que para realizar la búsqueda se tuvo que dividir el arreglo entre hilos para que cada uno fuese buscando en índices diferentes del arreglo, para ello se utilizó el constructor for, además del constructor critical.

La salida de este primer ejemplo es la siguiente:



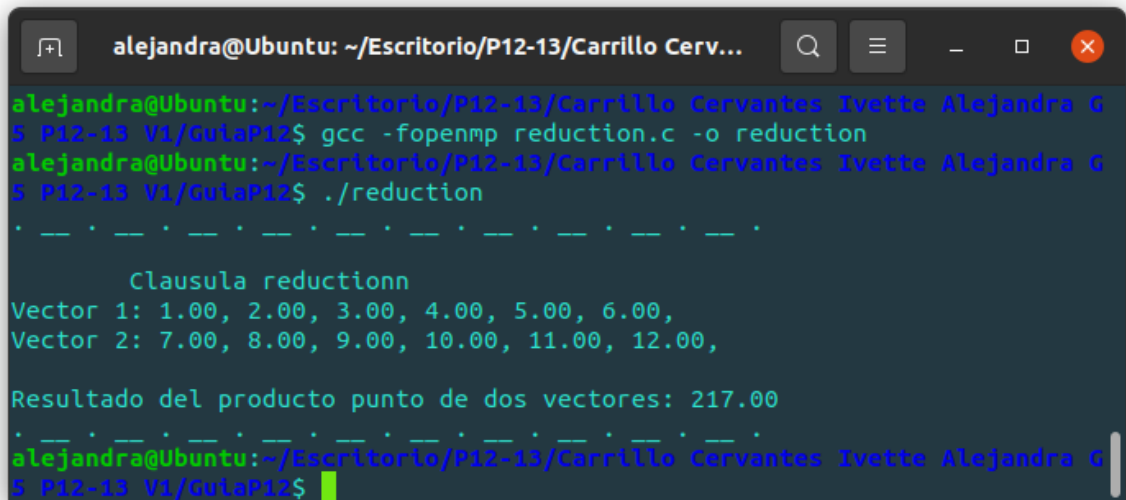
```
alejandra@Ubuntu: ~/Escritorio/P12-13/Carrillo Cervantes Ivette Alejandra G
S P12-
13 V1/GuiaP12$ gcc -fopenmp critical.c -o critical
alejandra@Ubuntu:~/Escritorio/P12-13/Carrillo Cervantes Ivette Alejandra G
S P12-13 V1/GuiaP12$ ./critical
. _ _ . _ _ . _ _ . _ _ . _ _ . _ _ . _ _ .
Constructor critical
Arreglo: 16, 2, 3, 4, 12, 6, 7, 8, 6,
El valor maximo del arreglo es: 16
. _ _ . _ _ . _ _ . _ _ . _ _ . _ _ . _ _ .
alejandra@Ubuntu:~/Escritorio/P12-13/Carrillo Cervantes Ivette Alejandra G
S P12-13 V1/GuiaP12$
```

*critical.c*

- Clausula reduction

La clausula reduction tiene como función tomar el valor de una variable aportada por cada hilo y aplicar la operación sobre esos datos para obtener el resultado. Con el fin de ver su implementación, se realizó un programa el cual tiene como objetivo realizar el producto punto entre dos vectores de n elementos (en este caso, dos matrices de tipo de dato double), para ello se realizó un ciclo for dentro de la clausula reduction, el cual se ira multiplicando elemento por elemento del vector y posteriormente sumando para obtener el resultado final. Cabe recalcar que se ocuparan varios hilos, los cuales realizaran los mismos cálculos, pero sobre diferentes elementos del vector, los cuales se sumarán al terminar cada hilo con su ejecución, por lo que también se estará ocupando el constructor for.

La salida de este ejemplo es la siguiente:



```
alejandra@Ubuntu: ~/Escritorio/P12-13/Carrillo Cervantes Ivette Alejandra G
$ P12-13 V1/GuiaP12$ gcc -fopenmp reduction.c -o reduction
alejandra@Ubuntu:~/Escritorio/P12-13/Carrillo Cervantes Ivette Alejandra G
$ P12-13 V1/GuiaP12$ ./reduction
. _ _ . _ _ . _ _ . _ _ . _ _ . _ _ . _ _ .
          Clausula reductionn
Vector 1: 1.00, 2.00, 3.00, 4.00, 5.00, 6.00,
Vector 2: 7.00, 8.00, 9.00, 10.00, 11.00, 12.00,

Resultado del producto punto de dos vectores: 217.00
. _ _ . _ _ . _ _ . _ _ . _ _ . _ _ . _ _ .
alejandra@Ubuntu:~/Escritorio/P12-13/Carrillo Cervantes Ivette Alejandra G
$ P12-13 V1/GuiaP12$
```

*reduction.c*

\* NOTA: Las actividades realizadas por la cláusula reduction se le conoce como operaciones de reducción y son muy utilizadas en la programación paralela. \*

- Constructor sections

Este constructor permite usar paralelismo funcional, ya que permite asignar secciones de código independiente a hilos diferentes para que trabajen de forma concurrente. Cada una de estas secciones paralelas es ejecutada por un solo hilo y cada hilo ejecuta a ninguna o alguna de ellas. El ejemplo que se implementó para este constructor consiste en ejecutar en paralelo diferentes funciones: alfa(), beta() y delta(), para ello se crearon dichas funciones y dentro de estas se imprimió un mensaje el cual corresponde a la función en donde se encuentre. Se uso el constructor sections y dentro de este varios section correspondiente a cada función establecida previamente.

La salida del programa es la siguiente:

```
alejandra@Ubuntu: ~/Escritorio/P12-13/Carrillo Cervantes Ivette Alejandra G
$ gcc -fopenmp sections.c -o sections
alejandra@Ubuntu:~/Escritorio/P12-13/Carrillo Cervantes Ivette Alejandra G
$ ./sections
. _ _ . _ _ . _ _ . _ _ . _ _ . _ _ . _ _ .
Constructor sections

Alfa
Delta
Beta
Gama
Epsilon
6.00
alejandra@Ubuntu:~/Escritorio/P12-13/Carrillo Cervantes Ivette Alejandra G
$
```

*sections.c*

- Constructor barrier

El constructor barrier tiene como objetivo colocar una “barrera” explícita para que cada hilo espere hasta que todos lleguen a la barrera, con la finalidad de que haya una forma de sincronización.

El primer ejemplo que se realizó para este constructor consiste en que la variable A y B realizan un trabajo; mientras que C y D deben de esperar a que el trabajo de A y B terminen su trabajo, o bien su ejecución.

El segundo ejemplo que se realizó para este constructor consiste en que dentro de la región paralela tener un constructor for para cada hilo y asignarles valores diferentes a los elementos del arreglo a, posteriormente con el constructor master imprimir el arreglo; sin embargo, dado que el constructor master no tiene barrera implícita se usa barrier con el objetivo de que todos los hilos esperen a que se imprima el arreglo antes de poder modificarlo.

La salida de ambos programas es la siguiente:

```
alejandra@Ubuntu: ~/Escritorio/P12-13/Carrillo Cerv...
alejandra@Ubuntu:~/Escritorio/P12-13/Carrillo Cervantes Ivette Alejandra G
S P12-13 V1/GuiaP12$ gcc -fopenmp barrier.c -o barrier
alejandra@Ubuntu:~/Escritorio/P12-13/Carrillo Cervantes Ivette Alejandra G
S P12-13 V1/GuiaP12$ ./barrier
. __ . __ . __ . __ . __ . __ . __ . __ . __ .
Constructor barrier

Realiza un trabajo

Procesado A y B
Realiza un trabajo

Procesado A y B
Realiza un trabajo

Procesado A y B
Realiza un trabajo

Procesado A y B
Realiza un trabajo

Procesado A y B
Realiza un trabajo

Procesado A y B
Realiza un trabajo

Procesado A y B
Realiza un trabajo

Procesado A y B
Realiza un trabajo

Procesando B y C
Realiza un trabajo
```

barrier.c

```
alejandra@Ubuntu: ~/Escritorio/P12-13/Carrillo Cerv...
alejandra@Ubuntu:~/Escritorio/P12-13/Carrillo Cervantes Ivette Alejandra G
S P12-13 V1/GuiaP12$ ./barrier
. __ . __ . __ . __ . __ . __ . __ . __ . __ .
Constructor barrier

a[0] = 0
a[1] = 1
a[2] = 4
a[3] = 9
a[4] = 16
. __ . __ . __ . __ . __ . __ . __ . __ . __ .
alejandra@Ubuntu:~/Escritorio/P12-13/Carrillo Cervantes Ivette Alejandra G
S P12-13 V1/GuiaP12$
```

barrier.c

- Constructor single

Este constructor tiene como objetivo definir un bloque básico de código dentro de una región paralela, que debe ser ejecutado por un hilo único (cualquiera, no se especifica), mientras que todos esperan. El ejemplo que se realizó para este constructor consiste en que dentro de una región paralela, manda a llamar a métodos en donde todos los hilos realizan actividades; sin embargo, dentro del constructor single se mandó a llamar a una actividad en específico llamada “actividades”, la cual solo la debe de realizar un solo hilo.

La salida del programa es la siguiente, se puede observar que solo un hilo ejecuto dicha actividad y los demás esperaron a que dicho hilo la ejecutara:

```

alejandra@Ubuntu: ~/Escritorio/P12-13/Carrillo Cervantes Ivette Alejandra G
$ P12-13 V1/GuiaP12$ gcc -fopenmp single.c -o single
alejandra@Ubuntu:~/Escritorio/P12-13/Carrillo Cervantes Ivette Alejandra G
$ P12-13 V1/GuiaP12$ ./single
. _ _ . _ _ . _ _ . _ _ . _ _ . _ _ . _ _ .
Constructor single
Todos realizan unas Actividades
Actividades
Todos realizan unas Actividades
Todos realizan unas Actividades
Todos realizan unas Actividades
Todos realizan unas Actividades
Todos realizan unas Actividades
Todos realizan unas Actividades
Realizan mas actividades
Realizan mas actividades
Realizan mas actividades
Realizan mas actividades
Realizan mas actividades
Realizan mas actividades
Realizan mas actividades
. _ _ . _ _ . _ _ . _ _ . _ _ . _ _ . _ _ .
alejandra@Ubuntu:~/Escritorio/P12-13/Carrillo Cervantes Ivette Alejandra G
$ P12-13 V1/GuiaP12$

```

*single.c*

- Constructor master

Este constructor tiene una función muy parecida al constructor anterior, ya que dentro de este las actividades son hechas por el hilo maestro y dado que no cuenta con alguna barrera, los hilos restantes no esperan a que el hilo termine la actividad asignada.

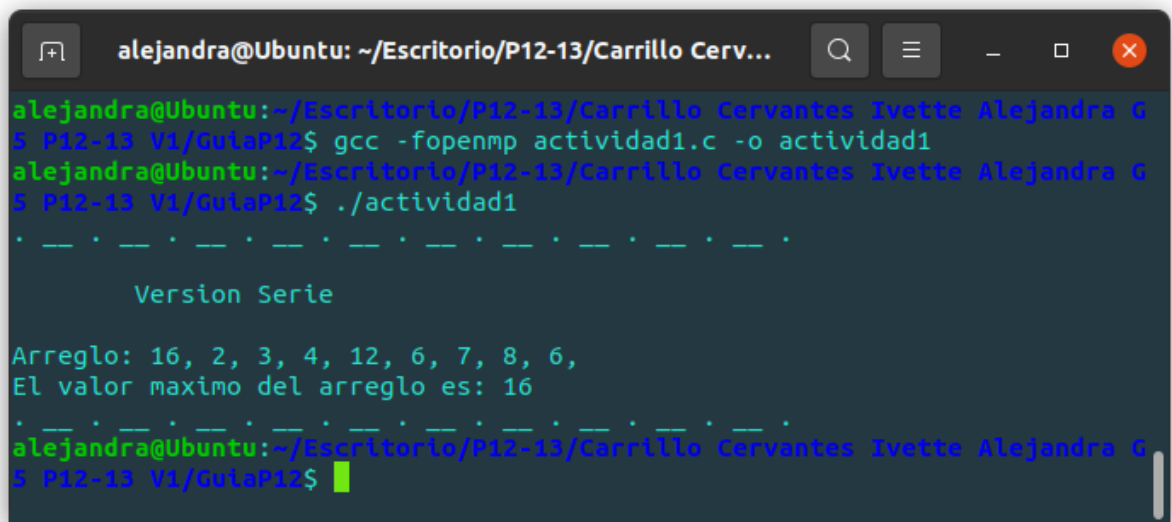
(En la guía no se encuentra algún ejemplo de este; sin embargo, se tiene la misma idea de implementación que single)

- Cláusula nowait

Es una cláusula, la cual permite quitar las barreras implícitas que tienen los constructores (for, single, sections, etc), con la finalidad de que al quitar las barreras que no son necesarias ayudar a mejorar el desempeño del programa.

- Actividad 1

Esta actividad tiene como función principal completar la versión serie y paralela del ejemplo del constructor critical; sin embargo, dado que anteriormente ya se realizó la versión paralela, en este ejemplo se realizará la versión serie. La única diferencia que se encontró entre ambas versiones es que en esta versión serie, se trabaja con hilos, entonces el programa recorre todo el ciclo de repetición for que tiene como finalidad buscar el valor maximo del arreglo; mientras que, en la versión paralela son varios hilos los que van realizando el programa con el objetivo, además de que sea más eficiente este y más rápido.



```

alejandra@Ubuntu: ~/Escritorio/P12-13/Carrillo Cervantes Ivette Alejandra G
S P12-13 V1/GuiaP12$ gcc -fopenmp actividad1.c -o actividad1
alejandra@Ubuntu:~/Escritorio/P12-13/Carrillo Cervantes Ivette Alejandra G
S P12-13 V1/GuiaP12$ ./actividad1
. _ _ . _ _ . _ _ . _ _ . _ _ . _ _ . _ _ .
Version Serie

Arreglo: 16, 2, 3, 4, 12, 6, 7, 8, 6,
El valor maximo del arreglo es: 16
. _ _ . _ _ . _ _ . _ _ . _ _ . _ _ . _ _ .
alejandra@Ubuntu:~/Escritorio/P12-13/Carrillo Cervantes Ivette Alejandra G
S P12-13 V1/GuiaP12$

```

*actividad1.c*

- Actividad 4

Esta actividad tiene como función principal observar los tiempos de ejecución entre las diferentes versiones que tiene un programa dado en específico el cual tiene como objetivo realizar diferentes operaciones. Se realizó la versión serie, la versión utilizando el constructor section y la versión utilizando el constructor for y se tomó el tiempo de ejecución de cada una de estas, se observó que la que más tiempo tomó, fue la versión utilizando el constructor sections, ya que este permite asignar secciones de código independiente a hilos diferentes para que trabajen de forma concurrente; mientras que for fue el que tomó menos tiempo debido a que al trabajar con varios hilos, tiene una mejor eficiencia. Con respecto a la versión serie, se ejecuta de una forma “normal”.

```
alejandra@Ubuntu: ~/Escritorio/P12-13/Carrillo Cerv...
alejandra@Ubuntu:~/Escritorio/P12-13/Carrillo Cervantes Ivette Alejandra G
$ P12-13 V1/GuiaP12$ gcc -fopenmp actividad4.c -o actividad4
alejandra@Ubuntu:~/Escritorio/P12-13/Carrillo Cervantes Ivette Alejandra G
$ P12-13 V1/GuiaP12$ ./actividad4
. _ _ . _ _ . _ _ . _ _ . _ _ . _ _ . _ _ .
Version Serie
TIEMPO=-0.001173
. _ _ . _ _ . _ _ . _ _ . _ _ . _ _ . _ _ .
alejandra@Ubuntu:~/Escritorio/P12-13/Carrillo Cervantes Ivette Alejandra G
$ P12-13 V1/GuiaP12$
```

*actividad4.c*

```
alejandra@Ubuntu: ~/Escritorio/P12-13/Carrillo Cerv...
alejandra@Ubuntu:~/Escritorio/P12-13/Carrillo Cervantes Ivette Alejandra G
$ P12-13 V1/GuiaP12$ gcc -fopenmp actividad4.c -o actividad4
alejandra@Ubuntu:~/Escritorio/P12-13/Carrillo Cervantes Ivette Alejandra G
$ P12-13 V1/GuiaP12$ ./actividad4
. _ _ . _ _ . _ _ . _ _ . _ _ . _ _ . _ _ .
Version Paralelo Section
TIEMPO=-0.002511
. _ _ . _ _ . _ _ . _ _ . _ _ . _ _ . _ _ .
alejandra@Ubuntu:~/Escritorio/P12-13/Carrillo Cervantes Ivette Alejandra G
$ P12-13 V1/GuiaP12$
```

*actividad4.c*

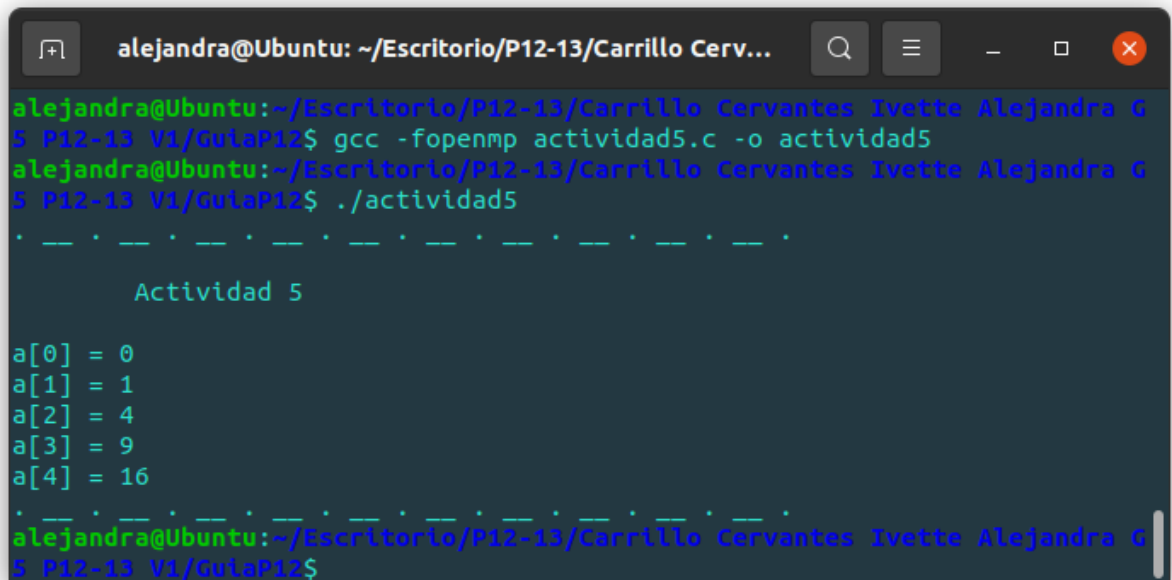
```
alejandra@Ubuntu: ~/Escritorio/P12-13/Carrillo Cerv...
alejandra@Ubuntu:~/Escritorio/P12-13/Carrillo Cervantes Ivette Alejandra G
$ P12-13 V1/GuiaP12$ gcc -fopenmp actividad4.c -o actividad4
alejandra@Ubuntu:~/Escritorio/P12-13/Carrillo Cervantes Ivette Alejandra G
$ P12-13 V1/GuiaP12$ ./actividad4
. _ _ . _ _ . _ _ . _ _ . _ _ . _ _ . _ _ .
Version Paralelo For
TIEMPO=-0.000300
. _ _ . _ _ . _ _ . _ _ . _ _ . _ _ . _ _ .
alejandra@Ubuntu:~/Escritorio/P12-13/Carrillo Cervantes Ivette Alejandra G
$ P12-13 V1/GuiaP12$
```

*actividad4.c*



- Actividad 5

Esta última actividad de la práctica 12 tiene como función ver la implementación de varios constructores y clausulas en un mismo programa: for, master y barrier, cada uno de ellos (excepto barrier) tienen un ciclo de repetición for el cual hace diferentes operaciones; mientras que master, solo imprime los índices de cada arreglo. En esta actividad también notamos que pasaría si se quitan alguna de estos constructores: si se quita la barrera, no se tendría un orden en específico y cada hilo llegaría hasta ahí sin un orden en específico; si en lugar de master se utilizará single solo se imprimiría en pantalla un índice del arreglo con su valor correspondiente, puesto que en master es donde se va imprimiendo el arreglo y al usar single solo un hilo realiza la acción correspondiente.



```
alejandra@Ubuntu: ~/Escritorio/P12-13/Carrillo Cervantes Ivette Alejandra G
$ P12-13 V1/GuiaP12$ gcc -fopenmp actividad5.c -o actividad5
alejandra@Ubuntu:~/Escritorio/P12-13/Carrillo Cervantes Ivette Alejandra G
$ P12-13 V1/GuiaP12$ ./actividad5
. __ . __ . __ . __ . __ . __ . __ . __ . __ . __ .
Actividad 5
a[0] = 0
a[1] = 1
a[2] = 4
a[3] = 9
a[4] = 16
alejandra@Ubuntu:~/Escritorio/P12-13/Carrillo Cervantes Ivette Alejandra G
$ P12-13 V1/GuiaP12$
```

*actividad5.c*

### **Actividades Práctica 13**

- Multiplicación de matrices

Este primer ejercicio de la práctica 13, consiste en realizar multiplicaciones de arreglos bidimensionales, con la finalidad de que sea mucho más eficiente este programa, se implemento la forma paralela del problema; para ello, se realizó un paralelismo de datos donde cada hilo trabaje con datos distintos, pero con el mismo algoritmo, con el objetivo de lograr dividir el problema y dependiendo de la granularidad escogida se necesitará un número de hilos en la región paralela. Para lograr obtener la suma de dos matrices se realizó un ciclo de repetición for anidado, el cual realizará las operaciones necesarias para obtener la multiplicación necesarias de las matrices previamente inicializadas con una dimensión de 500x500.

Cabe destacar que para que el programa lograra ser paralelo, las variables de dos ciclos for deben de ser privadas, ya que cada hilo las modificara al realizar los ciclos internos.

```
alejandra@Ubuntu: ~/Escritorio/P12-13/Carrillo Cerv...
alejandra@Ubuntu:~/Escritorio/P12-13/Carrillo Cervantes Ivette Alejandra G
$ P12-13 V1/GuiaP13$ gcc -fopenmp MultiplicacionMatrices.c -o Multiplicaci
onMatrices
alejandra@Ubuntu:~/Escritorio/P12-13/Carrillo Cervantes Ivette Alejandra G
$ P12-13 V1/GuiaP13$ ./MultiplicacionMatrices
. _ _ . _ _ . _ _ . _ _ . _ _ . _ _ . _ _ .
Multiplicación de Matrices Paralelo
TIEMPO=-0.132607
. _ _ . _ _ . _ _ . _ _ . _ _ . _ _ . _ _ .
alejandra@Ubuntu:~/Escritorio/P12-13/Carrillo Cervantes Ivette Alejandra G
$ P12-13 V1/GuiaP13$
```

*MultiplicacionMatrices.c*

- Actividad 1

Finalmente, la actividad 1 de la práctica 13, tiene como objetivo implementar el programa de multiplicación de matrices en forma serie y paralela; sin embargo, la forma paralela se implementó anteriormente, por lo que se tiene que implementar el programa de forma serie, para ello se utilizó el mismo procedimiento, pero sin usar ningún constructor. En este programa también se realizó un programa con arreglo de dimensión 500x500.

Al ejecutar ambos programas e imprimir el tiempo que se tarda cada uno de estos, pudimos observar que la versión que más se tardó fue la versión serie, ya que no cuenta con hilos que ayuden a que el programa sea más eficiente, recordemos que esta versión al ejecutarse debe de realizar todas las operaciones necesarias y no se pueden repartir mediante hilos como en la versión paralela.

```
alejandra@Ubuntu: ~/Escritorio/P12-13/Carrillo Cerv...
alejandra@Ubuntu:~/Escritorio/P12-13/Carrillo Cervantes Ivette Alejandra G
$ P12-13 V1/GuiaP13$ gcc -fopenmp actividad1.c -o actividad1
alejandra@Ubuntu:~/Escritorio/P12-13/Carrillo Cervantes Ivette Alejandra G
$ P12-13 V1/GuiaP13$ ./actividad1
. _ _ . _ _ . _ _ . _ _ . _ _ . _ _ . _ _ .
Multiplicación de Matrices Serie
TIEMPO=-0.347845
. _ _ . _ _ . _ _ . _ _ . _ _ . _ _ . _ _ .
alejandra@Ubuntu:~/Escritorio/P12-13/Carrillo Cervantes Ivette Alejandra G
$ P12-13 V1/GuiaP13$
```

*actividad1.c*

## Parte 2. Programación concurrente y paralela en JAVA

### **Sincronización en Java**

- Ejemplo 0

Este programa tiene como función principal crear dos hilos y ejecutarlos, para ello se crearon las siguientes clases:

- MyThread1. Tiene como atributos un objeto tipo Table, como métodos constructores uno donde se le pasa el objeto Table y un método sobrescrito run donde manda a llamar al método printTable con un tipo de dato entero 5
- MyThread2. Tiene como atributos un objeto tipo Table, como métodos constructores uno donde se le pasa el objeto Table y un método sobrescrito run donde manda a llamar al método printTable con un tipo de dato entero 100
- Ejemplo0. Tiene un método main.

El método printTable tiene como función imprimir 5 múltiplos de un número dado.

En la clase Ejemplo0, dentro del método main se inicializarán dos hilos, uno de la clase Thread1 y otro de la clase Thread2, pasándoles como parámetro un objeto tipo Table, posterior a ello se inician con el método start, una vez empezados sucede una condición carrera, puesto que primero se inicia el primer hilo que recordemos en su método constructor se llama al método printTable, pero al iniciar también al segundo hilo seguido de este, también se tratará de imprimir, por lo que al terminar la ejecución de ambos hilos y tratarlos de imprimir ambos a la vez, la salida del programa es la siguiente.

```
run:
5
100
10
200
15
300
20
400
25
500
BUILD SUCCESSFUL (total time: 2 seconds)
```

Con el fin de que no suceda esto y que primero se pueda imprimir los primeros 5 múltiplos que tiene el hilo1 y posterior a ello se puedan imprimir los primeros 5 múltiplos que tiene el hilo 2, se tuvo que agregar la palabra synchronized al método printTable. Cabe destacar que esta palabra reservada tiene el objetivo de que solo un hilo pueda acceder al método y nadie más hasta que finalice sus operaciones, en este caso, el hilo 2 no podrá acceder a este método hasta que el hilo terminé de ocupar este. La salida del programa es la siguiente:

```

run:
5
10
15
20
25
100
200
300
400
500
BUILD SUCCESSFUL (total time: 5 seconds)

```

- Ejemplo 1

Para este ejercicio se implementaron las siguientes dos clases:

- ➔ ThreadA. Tiene un método main.
- ➔ ThreadB. Tiene como atributo una variable int, la cual almacenará el tiempo que se tardó el hilo en la ejecución del programa y un método sobrescrito run, el cual tendrá una zona de acceso sincronizado, en este se tendrá un ciclo de repetición for el cual contará el total de milisegundos que se tarde el programa, posterior a ello se mandará a llamar al método notify() para que un hilo pueda salir de la zona de acceso sincronizado.

Este programa tiene como función crear un hilo y posteriormente tenerlo en una zona de acceso sincronizado, para tenerlo en dicha zona lo que se realizó para que el hilo liberara el acceso a la zona para que otro thread entre, fue que este llamará al método wait, el cual lo coloca en una cola de espera, este hilo saldrá hasya que otro thread en la misma zona de acceso haga "notify()", lo cual se realiza en cierto punto del método run() como se mencionó anteriormente. La salida del programa es la siguiente:

```

run:
Waiting for b to complete...
Total is: 4950
BUILD SUCCESSFUL (total time: 0 seconds)

```

- Ejemplo 2

Este programa tiene como función principal mandar dos mensajes. Para ello se crearon las siguientes clases:

- ➔ Sender. Esta clase tiene un método cuya función principal es mandarle el mensaje a enviar.
- ➔ ThreadedSend. Tiene como función principal crear un hilo con los atributos de mensaje y enviar, al igual que el método sobrescrito run, el cual tiene una zona de acceso sincronizado en el cual se manda el mensaje que se desea.
- ➔ Syncro, tiene un método main.

En este programa se crearon dos hilos, se comenzaron y posteriormente cada hilo llamo al método join() el cual tiene como función mantener la ejecución del hilo que se está ejecutando hasta que su ejecución termine. La salida del programa es la siguiente:

```
run:
Sending  Mensaje 1

Mensaje 1 Sent
Sending  Mensaje 2

Mensaje 2 Sent
BUILD SUCCESSFUL (total time: 2 seconds)
```

### ***Problema del Productor-Consumidor***

Este programa cuenta con las siguientes clases:

- ➔ ThreadExample. Tiene el método main.
- ➔ PC. Tiene como atributos una lista de enteros y una variable de tipo int la cual indica el tamaño de la lista, cuenta con un método produce el cual tiene un ciclo de repetición while, el cual se repetirá de siempre que se cumpla la ejecución de los hilos; dentro de esta se encuentra una zona de acceso sincronizado el cual imprimirá el productor. Esta clase también tendrá un método con las mismas características que el anterior, pero en vez de imprimir al productor, imprimirá al consumidor.

Tiene como finalidad crear dos hilos, uno que en su método run llama al método produce y otro que llama al método consume. Finalmente, ambos se ejecutan al mismo tiempo y se van intercalando uno con el otro, puesto que llaman a dos métodos distintos.

```
run:
Producer produced-0
Producer produced-1
Consumer consumed-0
Consumer consumed-1
Producer produced-2
Producer produced-3
Consumer consumed-2
Consumer consumed-3
Producer produced-4
Producer produced-5
Consumer consumed-4
Consumer consumed-5
Producer produced-6
Producer produced-7
Consumer consumed-6
Consumer consumed-7
```

*Productor.java*

## Conclusiones

Se cumplieron con los objetivos de esta práctica, ya que con los ejemplos proporcionados por el manual de la práctica 12 de laboratorio se implementaron algunas directivas de OpenMP para implementar algún algoritmo paralelo. Además, con los ejercicios proporcionados por el profesor, se vio uso de algunos métodos de instancia de los hilos, aprendimos para que sirve cada uno de los métodos al igual vimos la implementación de estos.

También, vimos la función de la sincronización, la cual implica que solo un hilo puede acceder a alguna acción durante el programa y nadie más puede acceder a dicha acción hasta que el hilo que esta en ella termine sus operaciones, para este concepto se debe de usar la palabra reservada "synchronized".

Considero que los ejercicios planteados tanto de laboratorio y por parte del profesor, fueron adecuados para estas dos prácticas, ya que personalmente aprendí los conceptos correspondientes. Además, en los ejemplos de la guía, al solo tener un pedazo de código y tener que modificarlo, comprendí más a que se refería cada constructor que se presentó.

Personalmente, fue una de las prácticas en las que más me tarde en los ejercicios de laboratorio, ya que algunos ejercicios me marcaban varios errores los cuales no sabían lo que significaban, pero una vez investigando todo quedo claro y pude resolverlos. (: