

# Practica N 11 "INTRODUCCIÓN A OpenMP" EDA 2

Oscar Daniel Rosario Morales, Israel Hipolito Mejia Alba

9 de noviembre de 2018

Grupo: 8

Profesor : Tista Garcia Edgar

## 0.1. Objetivo:

El estudiante conocerá y aprenderá a utilizar algunas de las directivas de OpenMP utilizadas para realizar programas paralelos.

## 0.2. Actividades:

- \* Revisar el ambiente necesario para trabajar con OpenMP y el lenguaje C.
- \* Realizar ejemplos del funcionamiento de las primeras directivas de OpenMP en el lenguaje C.
- \* Realizar un primer programa que se resuelva un programa de forma paralela utilizando las directivas de OpenMP.

## 0.3. Desarrollo

### Actividad 1

En un editor de texto teclear el siguiente código y guardarlo con extension ".c", por ejemplo hola.c. Después desde la consola o linea de comandos de linux, situarse en el directorio donde se encuentra el código fuente y compilarlo de la siguiente manera.



```
#include <stdio.h>
```

```
/*Ejercicio 1 */
```

```
int main()
```

{

```
int i;
```

```
printf("Hola mundo\n");
```

```
for( i = 0; i<10; i++)
```

{

```
printf("Iteracion: %d\n" , i);
```

}

```
printf("adios\n");
```

```
return 0;
```

}

// \*/

```
oscar@oscar-VirtualBox: ~/Escritorio
Archivo Editar Ver Buscar Terminal Ayuda
oscar@oscar-VirtualBox:~$ cd Escritorio/
oscar@oscar-VirtualBox:~/Escritorio$ gcc hola.c -o hola
^[[A^[[Aoscar@oscar-VirtualBox:~/Escritorio$ ./hola
Hola mundo
Iteracion: 0
Iteracion: 1
Iteracion: 2
Iteracion: 3
Iteracion: 4
Iteracion: 5
Iteracion: 6
Iteracion: 7
Iteracion: 8
Iteracion: 9
adios
oscar@oscar-VirtualBox:~/Escritorio$
```

Una vez que ya se tiene el código en su version serial, se formara una región paralela. Entonces agregar al código anterior el constructor **parallel** desde la declaración de la variable hasta antes de la ultima impresion de pantalla, como se muestra. Para compilarlo hay que indicar que se agregaran las directivas de openMP, lo que se realiza agregando la bandera en la compilación -fopenmp.

```
11_1.c 11_2.c 11_3.c 11_4.c 11_5.c 11_6.c 11_7.c 11_8.c 11_9.c 11_10.c
1  #include <stdio.h>
2  #include <omp.h>
3
4  int main()
5  {
6      #pragma omp parallel
7      {
8          int i;
9          printf("Hola Mundo\n");
10         for(i = 0; i<10;i++)
11             printf("iteracion: %d\n", i);
12     }
13     printf("adios\n");
14     return 0;
15 }
```

```
C:\Program Files (x86)\D
Hola Mundo
iteracion: 0
iteracion: 1
iteracion: 2
iteracion: 3
iteracion: 4
iteracion: 5
iteracion: 6
iteracion: 7
iteracion: 8
iteracion: 9
Hola Mundo
iteracion: 0
iteracion: 1
iteracion: 2
iteracion: 3
iteracion: 4
iteracion: 5
iteracion: 6
iteracion: 7
iteracion: 8
iteracion: 9
Hola Mundo
iteracion: 0
iteracion: 1
iteracion: 2
iteracion: 3
iteracion: 4
iteracion: 5
iteracion: 6
iteracion: 7
iteracion: 8
iteracion: 9
Hola Mundo
iteracion: 0
iteracion: 1
iteracion: 2
iteracion: 3
iteracion: 4
iteracion: 5
iteracion: 6
iteracion: 7
iteracion: 8
iteracion: 9
adios
-----
```

¿Qué diferencia hay en la salida del programa con respecto al secuencial? En el programa paralelo , se repitió el proceso de imprimir las iteraciones.

¿Por qué se obtiene esa salida? Esto se debe a que se ejecuta el proceso en diversos hilos. Aunque cabe mencionar que no siempre da la misma salida, en algunos casos, al realizar la impresión del hola mundo, no se imprimen las iteraciones y se agregan a otro proceso, repitiendo las iteraciones en uno de los casos.

## Actividad 2

Cambia el numero de hilos que habáa en la región paralela a un numero n ( entero) probar cada una de las formas indicadas.

```

1  #include <stdio.h>
2  #include <omp.h>
3
4  int main()
5  {
6      #pragma omp parallel
7      {
8          omp_set_num_threads(5);
9          int i;
10         printf("Hola Mundo\n");
11         for(i = 0; i<10;i++)
12             printf("iteracion: %d\n", i);
13     }
14     printf("adios\n");
15     return 0;
16 }

```

```

Hola Mundo
Hola Mundo
iteracion: 0
Hola Mundo
iteracion: 0
iteracion: 1
iteracion: 2
iteracion: 3
iteracion: 4
iteracion: 5
iteracion: 6
iteracion: 7
iteracion: 8
iteracion: 9
Hola Mundo
iteracion: 0
iteracion: 1
iteracion: 2
iteracion: 3
iteracion: 4
iteracion: 5
iteracion: 6
iteracion: 7
iteracion: 8
iteracion: 9
iteracion: 1
iteracion: 2
iteracion: 3
iteracion: 4
iteracion: 5
iteracion: 6
iteracion: 7
iteracion: 8
iteracion: 9
iteracion: 0
iteracion: 1
iteracion: 2
iteracion: 3
iteracion: 4
iteracion: 5
iteracion: 6
iteracion: 7
iteracion: 8
iteracion: 9
adios

```

```

1  #include <stdio.h>
2  #include <omp.h>
3
4  int main()
5  {
6      #pragma omp parallel num_threads(3)
7      {
8          int i;
9          printf("Hola Mundo\n");
10         for(i = 0; i<10;i++)
11             printf("iteracion: %d\n", i);
12     }
13     printf("adios\n");
14     return 0;
15 }

```

```

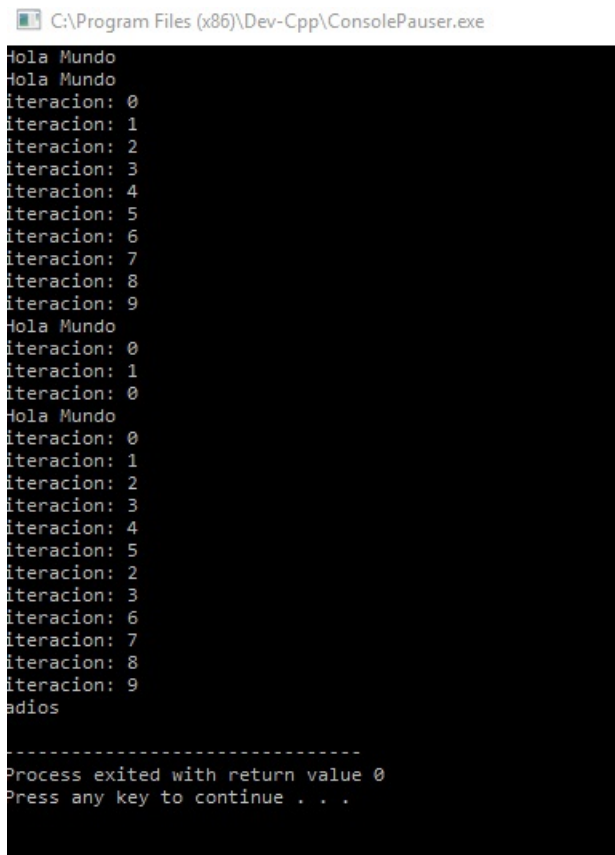
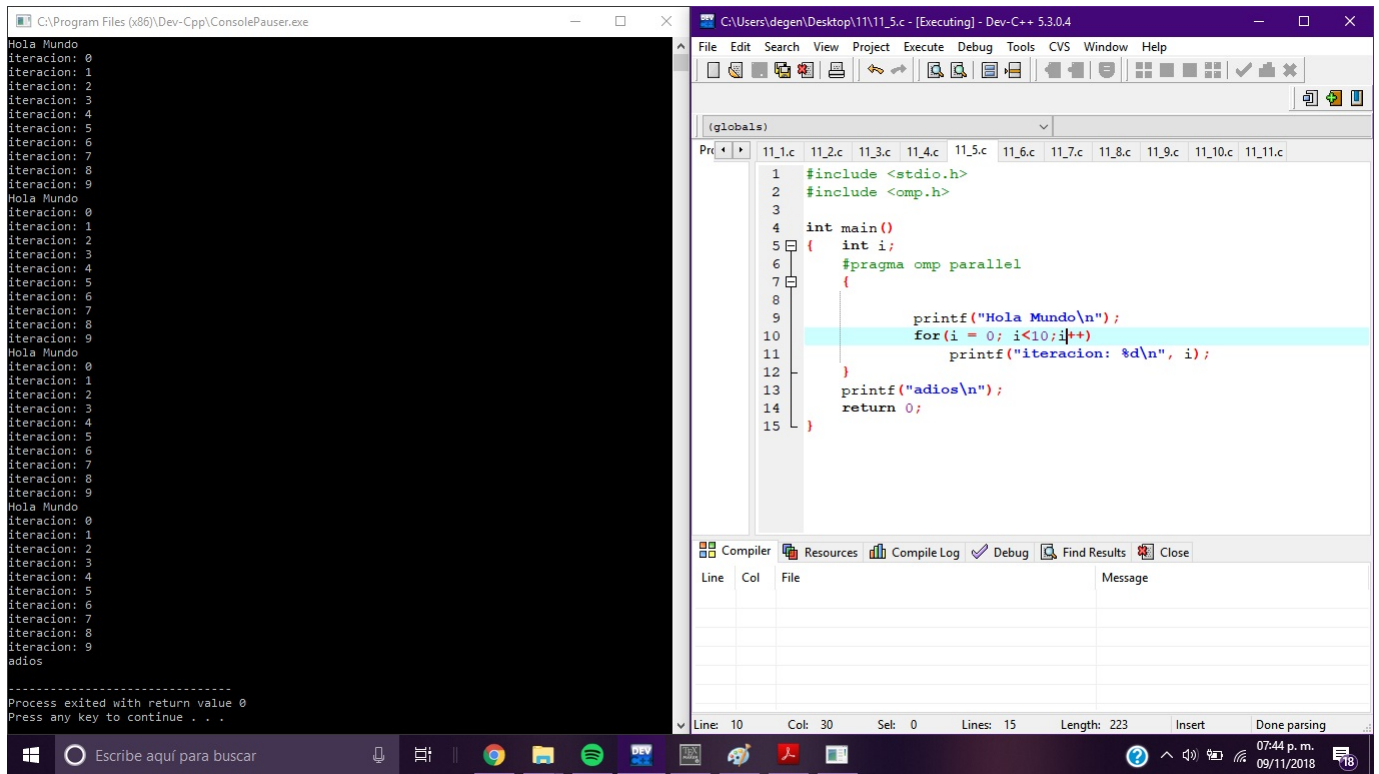
C:\Program Files (x86)\BC...
Hola Mundo
iteracion: 0
iteracion: 1
iteracion: 2
iteracion: 3
iteracion: 4
iteracion: 5
iteracion: 6
iteracion: 7
iteracion: 8
iteracion: 9
Hola Mundo
iteracion: 0
Hola Mundo
iteracion: 0
iteracion: 1
iteracion: 2
iteracion: 3
iteracion: 4
iteracion: 5
iteracion: 6
iteracion: 7
iteracion: 8
iteracion: 9
iteracion: 1
iteracion: 2
iteracion: 3
iteracion: 4
iteracion: 5
iteracion: 6
iteracion: 7
iteracion: 8
iteracion: 9
adios

```

¿Qué sucedió en la ejecución con respecto a la actividad 1? En el primer caso donde se cambia el numero de hilos a través de la función `omp set num threads(n =5)`, se puede observar que las iteraciones muchas de ellas se repiten en un proceso, y en otros solo se realizo una impresión de hola mundo. En el segundo caso donde se agrego la clausula en el constructor `parallel` no `n= 3`, se puede observar que paso lo mismo sin embargo solo se obtuvieron 3 hola mundo y en una proceso se repiten las iteraciones.

### Actividad 3

Del codigo que se esta trabajando, sacar de la region paralela la declaracion de la variable `int`, compilar y ejecutar varias veces.



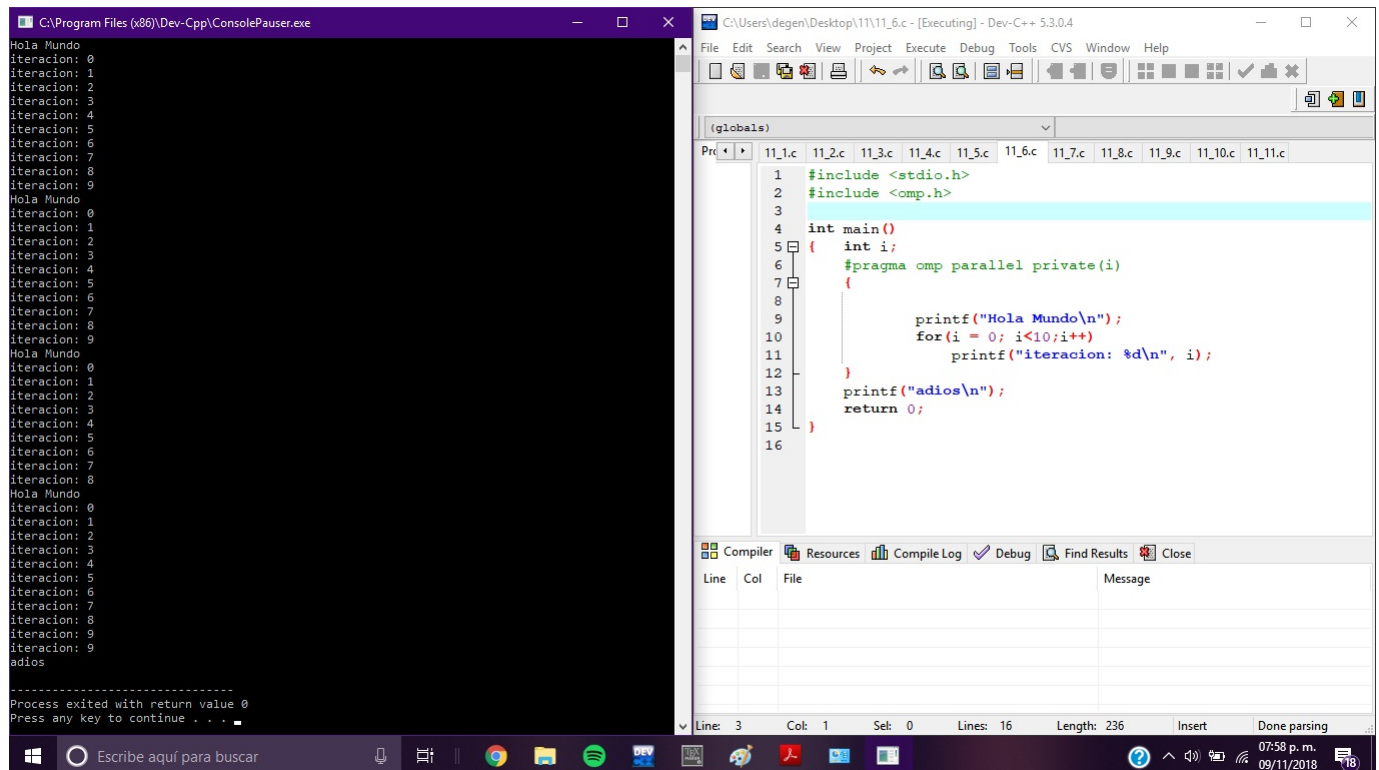
¿Qué sucedió y por qué? En la primera imagen se puede observar que las iteraciones se ejecutan correctamente, aunque el proceso se repite 4 veces, esto se debe a que la variable que se utilizara



en el ciclo for se hizo publica de la paralelización, se sitúa fuera del bloque a paralelizar, esto quiere decir que es una variable que sera compartida en memoria entre los hilos, al estar dentro la variable cada proceso la utilizaba de manera individual.

## Actividad 4

Existen dos cláusulas que pueden forzar que una variable privada sea compartida y una compartida sea privada. Al código resultante de la actividad 3, agregar la cláusula *private()* después del constructor parallel, con la variable i dentro.



```
1 #include <stdio.h>
2 #include <omp.h>
3
4 int main()
5 {
6     #pragma omp parallel private(i)
7     {
8
9         printf("Hola Mundo\n");
10        for(i = 0; i<10;i++)
11            printf("iteracion: %d\n", i);
12    }
13    printf("adios\n");
14    return 0;
15 }
```

¿Qué sucedió? Se puede observar que de nuevo, se realizaron las iteraciones correspondientes a cada hilo , lo que sucedió fue que la variable i paso de ser compartida a privada por lo que crearon copias, una para cada hilo, la cual sera utilizada durante el proceso del hilo y al final serán eliminadas.

## Actividad 5

Dentro de la region paralela hay un número de hilos generados y cada uno tiene asignado un identificador. Estos datos se pueden conocer durante la ejecución con la llamada a las funciones de la biblioteca *omp.get.num.threads()* y *omp.get.thread.num()* respectivamente. Probar el siguiente ejemplo y notar que para su buen funcionamiento se debe indicar que la variable tid sea privada dentro de la región paralela, ya que de no ser así todos los hilos escribirán en la dirección de memoria asignada a dicha variable sin un control ( *race condition*), es decir *competiran*” para ver quien llega antes y el resultado visualizado puede ser inconsistente e impredecible.

```
C:\Program Files (x86)\Dev-Cpp\ConsolePauser.exe
Hola Mundo desde el hilo 1 de un total de 4
Hola Mundo desde el hilo 0 de un total de 4
Hola Mundo desde el hilo 2 de un total de 4
Hola Mundo desde el hilo 3 de un total de 4
Adios
-----
Process exited with return value 0
Press any key to continue . . .

1 #include <stdio.h>
2 #include <omp.h>
3
4 int main()
5 {
6     int tid, nth;
7     #pragma omp parallel private(tid)
8     {
9         tid = omp_get_thread_num();
10        nth = omp_get_num_threads();
11
12        printf("Hola Mundo desde el hilo %d de un total de %d\r\n", tid, nth);
13    }
14
15    printf("Adios");
16    return 0;
17 }
```

Probar el ejemplo quitando del código la clasula private para visualizar el comportamiento del programa.

```
C:\Program Files (x86)\Dev-Cpp\ConsolePauser.exe
Hola Mundo desde el hilo 1 de un total de 4
Hola Mundo desde el hilo 1 de un total de 4
Hola Mundo desde el hilo 1 de un total de 4
Hola Mundo desde el hilo 1 de un total de 4
Adios
-----
Process exited with return value 0
Press any key to continue . . .

1 #include <stdio.h>
2 #include <omp.h>
3
4 int main()
5 {
6     int tid, nth;
7     #pragma omp parallel
8     {
9         tid = omp_get_thread_num();
10        nth = omp_get_num_threads();
11
12        printf("Hola Mundo desde el hilo %d de un total de %d\r\n", tid, nth);
13    }
14
15    printf("Adios");
16    return 0;
17 }
```

¿Qué sucedió? Como se comentaba anteriormente al mantener la variable tid como compartida, se sobre escriben los datos ya que se utilizan repetidas veces, tal como se ve la impresión de identificación del hilo 1/4 se repite una vez.

## Actividad 6

Se requiere realizar la suma de dos arreglos unidimensionales de 10 elementos de forma paralela utilizando solo dos hilos. Para ello se utilizará un paralelismo de datos o descomposición de dominio, es decir, cada hilo trabajará con diferentes elementos de los arreglos a sumar A y B, pero ambos utilizarán el mismo algoritmo para realizar la suma.

Realizar programa en su version serial.

```
38 }
39
40 void suma(int *A, int *B, int *C)
41 {
42     int i;
43     for(i = 0; i<n; i++)
44     {
45         C[i] = A[i] + B[i];
46         printf("%d\t", C[i]);
47     }
48 }
49
```

Resources Compile Log Debug Find Results

C:\Program Files (x86)\Dev-Cpp\ConsolePauser.exe

7	4	0	9	4	8	8	2	4
5	1	7	1	1	5	2	7	6
12	5	7	10	5	13	10	9	10

-----

Process exited with return value 3

Press any key to continue . . .

Para la version paralela, el hilo 0 sumará la primera mitad de A con la primera mitad de B y el hilo 1 sumará las segundas mitades. Para conseguir esto cada hilo realizará las mismas instrucciones, pero utilizará índices diferentes para referirse a diferentes elementos de los arreglos, cada uno iniciará y terminará el índice i en valores distintos.

```
void suma(int *A, int *B, int *C)
{
    int i,tid,inicio,fin;

    omp_set_num_threads(2);
    #pragma omp parallel private(inicio,fin, tid,i)

    {
        tid = omp_get_thread_num();
        inicio = tid * 5;
        fin = (tid+1) * 5 - 1;
        for(i = inicio; i<fin; i++)
        {
            C[i] = A[i] + B[i];
            printf("hilo %d calculo C[%d] = %d\n",tid,i, C[i]);
        }
    }
}
```

```
C:\Program Files (x86)\Dev-Cpp\ConsolePauser.exe
1      7      4      0      9      4      8      8      2      4
5      5      1      7      1      1      5      2      7      6
hilo 0 calculo C[0] = 6
hilo 0 calculo C[1] = 12
hilo 0 calculo C[2] = 5
hilo 0 calculo C[3] = 7
hilo 1 calculo C[5] = 5
hilo 1 calculo C[6] = 13
hilo 1 calculo C[7] = 10
hilo 1 calculo C[8] = 9

-----
Process exited with return value 7672040
Press any key to continue . . .
```

Lo que se cambio fue que los intervalos para cada for que se realizaba en un hilo fueron diferentes, como solo iban a hacer 2 hilos, se tendría el hilo 0 y el hilo 1, para el primer hilo , su for va desde  $i = 0$ , hasta  $i = 4$ , y en el segundo hilo va desde  $i = 5$  ,hasta  $i = 9$  , así se recorre el entero de tamaño 10 por completo pero cada hilo realizando las mismas operaciones en su mitad perteneciente.

## Actividad 7

Otro constructor es el for, el cual divide las iteraciones de una estructura de repetición for. Para utilizarlo se debe estar dentro de una región paralela. Lo que realiza este constructor es que la variable de control que se utiliza para iterar se hará automáticamente privada, esto es para que cada hilo trabaje con su propia variable  $i$ .

Modifica el código de la actividad 1, de manera que se dividan las iteraciones de la estructura de repetición for.

```
1  #include <stdio.h>
2  #include <omp.h>
3
4  int main()
5  {
6      #pragma omp parallel
7      {
8          int i;
9          printf("Hola Mundo\n");
10         #pragma omp for
11
12         for( i = 0; i<10;i++)
13             printf("iteracion: %d\n", i);
14     }
15     printf("adios\n");
16     return 0;
17 }
```

```

Hola Mundo
iteracion: 3
iteracion: 4
iteracion: 5
Hola Mundo
iteracion: 0
iteracion: 1
iteracion: 2
Hola Mundo
iteracion: 8
iteracion: 9
Hola Mundo
iteracion: 6
iteracion: 7
adiós

-----
Process exited with return value 0
Press any key to continue . . .

```

Se puede observar que al repartirse las iteraciones del for entre los hilos, no todos realizan el mismo procedimiento, mostrando que algunos tienen más carga de operaciones que otros.

## Actividad 8

Realiza la suma de dos arreglos unidimensionales de  $n$  elementos de forma paralela, utilizando los hilos por defecto que se generen en la región paralela y el constructor for.

```

39 }
40 void suma(int *A, int *B, int *C)
41 {
42     int i, tid;
43
44
45     #pragma omp parallel private(tid)
46
47     {
48         tid = omp_get_thread_num();
49         #pragma omp for
50         for(i = 0; i < n; i++)
51         {
52             C[i] = A[i] + B[i];
53             printf("hilo %d calculo C[%d] = %d\n", tid, i, C[i]);
54         }
55     }
56 }

```

```

1      7      4      0      9      4      8      8      2      4
5      5      1      7      1      1      5      2      7      6
hilo 0 calculo C[0] = 6
hilo 0 calculo C[1] = 12
hilo 0 calculo C[2] = 5
hilo 1 calculo C[3] = 7
hilo 1 calculo C[4] = 10
hilo 1 calculo C[5] = 5
hilo 2 calculo C[6] = 13
hilo 2 calculo C[7] = 10
hilo 3 calculo C[8] = 9
hilo 3 calculo C[9] = 10

-----
Process exited with return value 13045992
Press any key to continue . . .

```

Al utilizar dos regiones paralelas dentro del programa se puede observar que se generaron mas hilos, los primeros dos se generaron por la primera región paralela como en los ejercicios anteriores, y otros dos hilos se generaron para la región paralela del ciclo for.

## 1. Conclusiones

### Mejía Alba Israel Hipólito

Realizar esta práctica fue de gran provecho ya que ahora se pudo ver de una manera menos conceptual como es que se crean los programas en paralelo y se aprendió desde cero como usar OpenMP, cabe destacar que nos fue de gran ayuda las instrucciones del profesor para configurar Dev-C++ para poder correr las directivas de esta herramienta. El ejercicio de la práctica que más me llamo la atención fue el numero 3 el cual se pudo ver como al declarar una variable fuera de la paralelización y al momento de usarla en el código dentro esta, cada hilo comparte la misma variable, sin embargo, los valores que le da en ejecución son únicos en durante su proceso y cambia cuando se pasa a otro hilo. Este suceso se puede comparar también con el de la actividad 4 la cual se usa la cláusula `private(i)` la cual nos dice que a la variable `i` se le hará una copia por cada hilo que la use, la cual será eliminada después de que el programa se acabe de ejecutar.

### Rosario Morales Oscar Daniel

Al finalizar esta practica pudimos entender de mejor manera aunque básica, el funcionamiento de C en programas paralelos, a través de la clausula de *pragma.omp.parallel* y *pragma.omp.for*, donde la primera es para paralelizar cierta parte del código automáticamente, con la condición de considerar variables privadas o compartidas y el segundo para dividir el trabajo de un ciclo de repetición for, a través de realizar las mismas operaciones sobre distintos datos que no son dependientes entre si. Al principio se tuvo un poco de complicación al utilizar OpenMp ya que se trato de realizar en un entorno de Linux Ubuntu, pero consultando con el profesor, se comento la forma de realizar los programas en un IDE en windows, modificando las opciones de compilación del mismo.