

1.) En el código 3.11 se declaran dos variables, `nthreads` (numero de threads) y `tid` (identificador de un hilo dentro de la zona paralelizada por openMP). Cuando se ejecuta dicho código, cada thread imprime “Hola desde el thread [tid] de [nthreads] threads”. En mi equipo y sin modificar la variable de entorno “OMP_NUM_THREADS” se puede ver que hay un total de 4 threads que se van ejecutando en orden aleatorio. Como podemos ver, cada vez que se ejecuta, el orden es diferente.

```
Aplicaciones ▾ Lugares ▾ Terminal ▾ dom, 29 de abr, 19:20 1
root@Kali: ~/Universidad/PPD/Prácticas/Práctica 9/RecursosPracticaOpenMP

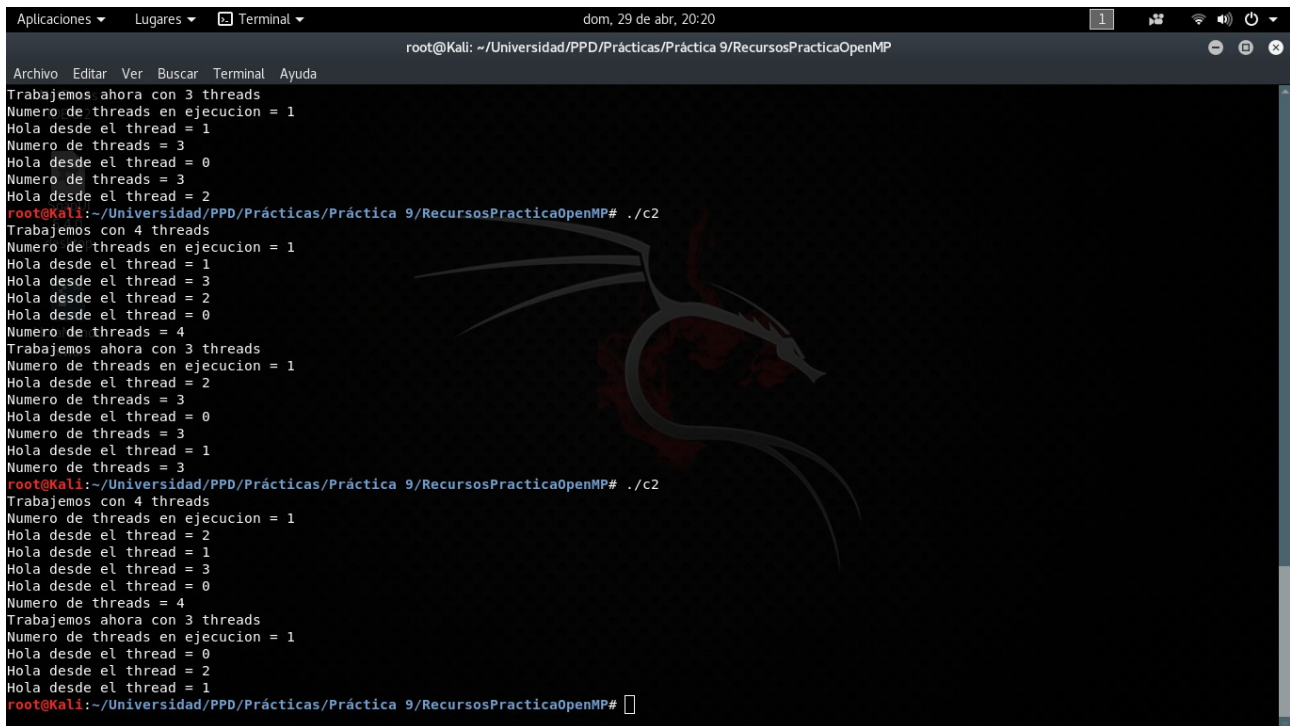
Archivo Editar Ver Buscar Terminal Ayuda
Hola desde el thread 2 de 4 threads
root@Kali:~/Universidad/PPD/Prácticas/Práctica 9/RecursosPracticaOpenMP# ./c1
Hola desde el thread 0 de 4 threads
Hola desde el thread 2 de 4 threads
Hola desde el thread 1 de 4 threads
Hola desde el thread 3 de 4 threads
root@Kali:~/Universidad/PPD/Prácticas/Práctica 9/RecursosPracticaOpenMP# ./c1
Hola desde el thread 0 de 4 threads
Hola desde el thread 2 de 4 threads
Hola desde el thread 1 de 4 threads
Hola desde el thread 3 de 4 threads
root@Kali:~/Universidad/PPD/Prácticas/Práctica 9/RecursosPracticaOpenMP# ./c1
Hola desde el thread 0 de 4 threads
Hola desde el thread 2 de 4 threads
Hola desde el thread 3 de 4 threads
Hola desde el thread 1 de 4 threads
root@Kali:~/Universidad/PPD/Prácticas/Práctica 9/RecursosPracticaOpenMP# ./c1
Hola desde el thread 0 de 4 threads
Hola desde el thread 2 de 4 threads
Hola desde el thread 3 de 4 threads
Hola desde el thread 1 de 4 threads
root@Kali:~/Universidad/PPD/Prácticas/Práctica 9/RecursosPracticaOpenMP# ./c1
Hola desde el thread 0 de 4 threads
Hola desde el thread 2 de 4 threads
Hola desde el thread 3 de 4 threads
Hola desde el thread 1 de 4 threads
root@Kali:~/Universidad/PPD/Prácticas/Práctica 9/RecursosPracticaOpenMP# ./c1
Hola desde el thread 0 de 4 threads
Hola desde el thread 2 de 4 threads
Hola desde el thread 3 de 4 threads
Hola desde el thread 1 de 4 threads
root@Kali:~/Universidad/PPD/Prácticas/Práctica 9/RecursosPracticaOpenMP# ./c1
Hola desde el thread 0 de 4 threads
Hola desde el thread 2 de 4 threads
Hola desde el thread 3 de 4 threads
Hola desde el thread 1 de 4 threads
root@Kali:~/Universidad/PPD/Prácticas/Práctica 9/RecursosPracticaOpenMP#
```

2.) En este ejercicio, el maestro le dice al resto de threads “Hola amigos, soy vuestro maestro”. A continuación, el resto de hilos (3 en mi equipo) le responden “Hola maestro, soy el esclavo [tid]” (siendo “tid” el identificador de cada thread). Como podemos ver, cada vez que se ejecuta, el orden es diferente.

```
Aplicaciones ▾ Lugares ▾ Terminal ▾ dom, 29 de abr, 20:10 1
root@Kali: ~/Universidad/PPD/Prácticas/Práctica 9

Archivo Editar Ver Buscar Terminal Ayuda
Hola maestro, soy el esclavo 1
root@Kali:~/Universidad/PPD/Prácticas/Práctica 9# ./e1
Hola amigos, soy vuestro maestro con id 0
Hola maestro, soy el esclavo 2
Hola maestro, soy el esclavo 3
Hola maestro, soy el esclavo 1
root@Kali:~/Universidad/PPD/Prácticas/Práctica 9# ./e1
Hola amigos, soy vuestro maestro con id 0
Hola maestro, soy el esclavo 1
Hola maestro, soy el esclavo 3
Hola maestro, soy el esclavo 2
root@Kali:~/Universidad/PPD/Prácticas/Práctica 9# ./e1
Hola amigos, soy vuestro maestro con id 0
Hola maestro, soy el esclavo 3
Hola maestro, soy el esclavo 1
Hola maestro, soy el esclavo 2
root@Kali:~/Universidad/PPD/Prácticas/Práctica 9# ./e1
Hola amigos, soy vuestro maestro con id 0
Hola maestro, soy el esclavo 2
Hola maestro, soy el esclavo 1
Hola maestro, soy el esclavo 3
root@Kali:~/Universidad/PPD/Prácticas/Práctica 9# ./e1
Hola amigos, soy vuestro maestro con id 0
Hola maestro, soy el esclavo 2
Hola maestro, soy el esclavo 1
Hola maestro, soy el esclavo 3
root@Kali:~/Universidad/PPD/Prácticas/Práctica 9# ./e1
Hola amigos, soy vuestro maestro con id 0
Hola maestro, soy el esclavo 3
Hola maestro, soy el esclavo 1
Hola maestro, soy el esclavo 2
root@Kali:~/Universidad/PPD/Prácticas/Práctica 9# ./e1
Hola amigos, soy vuestro maestro con id 0
Hola maestro, soy el esclavo 2
Hola maestro, soy el esclavo 1
Hola maestro, soy el esclavo 3
root@Kali:~/Universidad/PPD/Prácticas/Práctica 9#
```

3.) En el código 3.12 se establece el número de threads a 4 y se ejecuta un código parecido al del ejercicio 1. Luego se establece el número de threads a 3 y, por cada hilo, se imprime un saludo. Si el identificador del hilo que se está ejecutando es 0, se imprime un saludo desde los tres hilos, si no es 0, podemos ver como se intercala el número de threads que hay con un saludo desde otro thread distinto al 0. Como podemos ver, cada vez que se ejecuta, el orden es diferente.



```
Aplicaciones ▾ Lugares ▾ Terminal ▾ dom, 29 de abr, 20:20 1
root@Kali: ~/Universidad/PPD/Prácticas/Práctica 9/RecursosPracticaOpenMP

Archivo Editar Ver Buscar Terminal Ayuda
Trabajemos ahora con 3 threads
Numero de threads en ejecucion = 1
Hola desde el thread = 1
Numero de threads = 3
Hola desde el thread = 0
Numero de threads = 3
Hola desde el thread = 2
root@Kali:~/Universidad/PPD/Prácticas/Práctica 9/RecursosPracticaOpenMP# ./c2
Trabajemos con 4 threads
Numero de threads en ejecucion = 1
Hola desde el thread = 1
Hola desde el thread = 3
Hola desde el thread = 2
Hola desde el thread = 0
Numero de threads = 4
Trabajemos ahora con 3 threads
Numero de threads en ejecucion = 1
Hola desde el thread = 2
Numero de threads = 3
Hola desde el thread = 0
Numero de threads = 3
Hola desde el thread = 1
Numero de threads = 3
root@Kali:~/Universidad/PPD/Prácticas/Práctica 9/RecursosPracticaOpenMP# ./c2
Trabajemos con 4 threads
Numero de threads en ejecucion = 1
Hola desde el thread = 2
Hola desde el thread = 1
Hola desde el thread = 3
Hola desde el thread = 0
Numero de threads = 4
Trabajemos ahora con 3 threads
Numero de threads en ejecucion = 1
Hola desde el thread = 0
Hola desde el thread = 2
Hola desde el thread = 1
root@Kali:~/Universidad/PPD/Prácticas/Práctica 9/RecursosPracticaOpenMP#
```

En el código 3.13 se declaran las variables CHUNKSIZE a 2 y N a 10, se crean tres vectores (a, b y c) de 10 posiciones de los cuales solo dos (a y b) se inicializan. A continuación, en la región a paralelizar, las variables “a”, “b” y “chunk” son compartidas mientras que “i” y “tid” son privadas a cada variable. En el for, al ser “schedule(static, chunk)” se divide en tantos bloques como threads del tamaño de “chunk” (chunk=2). En el bucle se suman los contenidos de los vectores “a” y “b” y se guardan en “c” mientras que se va imprimiendo por pantalla la iteración que realiza cada thread, es decir dos iteraciones por thread, excepto uno de ellos que hace dos, debido a que son 10 elementos y solo 4 threads. Si fueran 5, cada thread iteraría solo dos veces. Como podemos ver, cada vez que se ejecuta, el orden es diferente.

```
Aplicaciones ▾ Lugares ▾ Terminal ▾ dom, 29 de abr, 20:34 1
root@Kali: ~/Universidad/PPD/Prácticas/Práctica 9/RecursosPracticaOpenMP

Archivo Editar Ver Buscar Terminal Ayuda
El thread 1, de 4 threads, calcula la iteracion i =3
El thread 0, de 4 threads, calcula la iteracion i =8
El thread 0, de 4 threads, calcula la iteracion i =9
root@Kali:~/Universidad/PPD/Prácticas/Práctica 9/RecursosPracticaOpenMP# ./c3
El thread 0, de 4 threads, calcula la iteracion i =0
El thread 0, de 4 threads, calcula la iteracion i =1
El thread 0, de 4 threads, calcula la iteracion i =8
El thread 0, de 4 threads, calcula la iteracion i =9
El thread 2, de 4 threads, calcula la iteracion i =4
El thread 2, de 4 threads, calcula la iteracion i =5
El thread 1, de 4 threads, calcula la iteracion i =2
El thread 1, de 4 threads, calcula la iteracion i =3
El thread 3, de 4 threads, calcula la iteracion i =6
El thread 3, de 4 threads, calcula la iteracion i =7
root@Kali:~/Universidad/PPD/Prácticas/Práctica 9/RecursosPracticaOpenMP# ./c3
El thread 0, de 4 threads, calcula la iteracion i =0
El thread 0, de 4 threads, calcula la iteracion i =1
El thread 0, de 4 threads, calcula la iteracion i =8
El thread 0, de 4 threads, calcula la iteracion i =9
El thread 2, de 4 threads, calcula la iteracion i =4
El thread 2, de 4 threads, calcula la iteracion i =5
El thread 3, de 4 threads, calcula la iteracion i =6
El thread 3, de 4 threads, calcula la iteracion i =7
El thread 1, de 4 threads, calcula la iteracion i =2
El thread 1, de 4 threads, calcula la iteracion i =3
root@Kali:~/Universidad/PPD/Prácticas/Práctica 9/RecursosPracticaOpenMP# ./c3
El thread 0, de 4 threads, calcula la iteracion i =0
El thread 0, de 4 threads, calcula la iteracion i =1
El thread 2, de 4 threads, calcula la iteracion i =4
El thread 2, de 4 threads, calcula la iteracion i =5
El thread 0, de 4 threads, calcula la iteracion i =8
El thread 0, de 4 threads, calcula la iteracion i =9
El thread 1, de 4 threads, calcula la iteracion i =2
El thread 1, de 4 threads, calcula la iteracion i =3
El thread 3, de 4 threads, calcula la iteracion i =6
El thread 3, de 4 threads, calcula la iteracion i =7
root@Kali:~/Universidad/PPD/Prácticas/Práctica 9/RecursosPracticaOpenMP#
```

4.) Para realizar este ejercicio podemos reutilizar cierta parte del código 3.13 donde también se divide el trabajo en 4 procesos (threads) en bloques de tamaño dos. Para resolverlo, primeramente se calcula el producto escalar de los vectores de forma secuencial y, luego, de forma paralela, dejando la última suma, para realizarla de forma secuencial una vez que todo el código paralelo ha terminado. Como podemos ver, cada vez que se ejecuta, el orden es diferente.

```
Aplicaciones ▾ Lugares ▾ Terminal ▾ dom, 29 de abr, 23:28 1
root@Kali: ~/Universidad/PPD/Prácticas/Práctica 9

Archivo Editar Ver Buscar Terminal Ayuda
Not found.
La suma secuencial es: 285
A continuacion, la suma paralela:
El thread 0, de 4 threads, calcula la iteracion i = 0: 0*0=0
El thread 0, de 4 threads, calcula la iteracion i = 1: 1*1=1
El thread 0, de 4 threads, calcula la iteracion i = 8: 8*8=64
El thread 0, de 4 threads, calcula la iteracion i = 9: 9*9=81
El thread 3, de 4 threads, calcula la iteracion i = 6: 6*6=36
El thread 3, de 4 threads, calcula la iteracion i = 7: 7*7=49
El thread 1, de 4 threads, calcula la iteracion i = 2: 2*2=4
El thread 1, de 4 threads, calcula la iteracion i = 3: 3*3=9
El thread 2, de 4 threads, calcula la iteracion i = 4: 4*4=16
El thread 2, de 4 threads, calcula la iteracion i = 5: 5*5=25
La suma paralela es: 285
root@Kali:~/Universidad/PPD/Prácticas/Práctica 9# ./e2
Primero realizaremos la suma secuencial:
La suma secuencial es: 285
A continuacion, la suma paralela:
El thread 0, de 4 threads, calcula la iteracion i = 0: 0*0=0
El thread 0, de 4 threads, calcula la iteracion i = 1: 1*1=1
El thread 0, de 4 threads, calcula la iteracion i = 8: 8*8=64
El thread 0, de 4 threads, calcula la iteracion i = 9: 9*9=81
El thread 3, de 4 threads, calcula la iteracion i = 6: 6*6=36
El thread 3, de 4 threads, calcula la iteracion i = 7: 7*7=49
El thread 1, de 4 threads, calcula la iteracion i = 2: 2*2=4
El thread 1, de 4 threads, calcula la iteracion i = 3: 3*3=9
El thread 2, de 4 threads, calcula la iteracion i = 4: 4*4=16
El thread 2, de 4 threads, calcula la iteracion i = 5: 5*5=25
La suma paralela es: 285
root@Kali:~/Universidad/PPD/Prácticas/Práctica 9#
```

5.) En el código 3.14 se utiliza la directiva “section” para especificar qué código de los bloques agrupados por dicha directiva, se dividirá entre los threads del equipo. En este código concreto, se realiza una impresión de lo que ejecuta (o calcula, como se muestra por pantalla) cada thread del equipo. Como podemos ver, cada vez que se ejecuta, el orden es diferente.


```
Aplicaciones ▾ Lugares ▾ Terminal ▾ dom, 29 de abr, 21:54
root@Kali: ~/Universidad/PPD/Prácticas/Práctica 9/RecursosPracticaOpenMP

Archivo Editar Ver Buscar Terminal Ayuda
El thread 3, de 4, calcula la seccion 3
root@Kali:~/Universidad/PPD/Prácticas/Práctica 9/RecursosPracticaOpenMP# ./c4
El thread 0, de 4, calcula la seccion 1
El thread 3, de 4, calcula la seccion 4
El thread 2, de 4, calcula la seccion 2
El thread 1, de 4, calcula la seccion 3
root@Kali:~/Universidad/PPD/Prácticas/Práctica 9/RecursosPracticaOpenMP# ./c4
El thread 0, de 4, calcula la seccion 2
El thread 2, de 4, calcula la seccion 1
El thread 3, de 4, calcula la seccion 4
El thread 1, de 4, calcula la seccion 3
root@Kali:~/Universidad/PPD/Prácticas/Práctica 9/RecursosPracticaOpenMP# ./c4
El thread 0, de 4, calcula la seccion 1
El thread 2, de 4, calcula la seccion 2
El thread 1, de 4, calcula la seccion 3
El thread 3, de 4, calcula la seccion 4
root@Kali:~/Universidad/PPD/Prácticas/Práctica 9/RecursosPracticaOpenMP# ./c4
El thread 0, de 4, calcula la seccion 3
El thread 2, de 4, calcula la seccion 4
El thread 1, de 4, calcula la seccion 1
El thread 3, de 4, calcula la seccion 2
root@Kali:~/Universidad/PPD/Prácticas/Práctica 9/RecursosPracticaOpenMP# ./c4
El thread 0, de 4, calcula la seccion 2
El thread 2, de 4, calcula la seccion 3
El thread 3, de 4, calcula la seccion 4
El thread 1, de 4, calcula la seccion 1
root@Kali:~/Universidad/PPD/Prácticas/Práctica 9/RecursosPracticaOpenMP# ./c4
El thread 0, de 4, calcula la seccion 1
El thread 2, de 4, calcula la seccion 2
El thread 1, de 4, calcula la seccion 3
El thread 3, de 4, calcula la seccion 4
root@Kali:~/Universidad/PPD/Prácticas/Práctica 9/RecursosPracticaOpenMP# ./c4
El thread 0, de 4, calcula la seccion 2
El thread 2, de 4, calcula la seccion 1
El thread 3, de 4, calcula la seccion 4
El thread 1, de 4, calcula la seccion 3
root@Kali:~/Universidad/PPD/Prácticas/Práctica 9/RecursosPracticaOpenMP#
```

En el código 3.15 se puede observar la utilización de la directiva “parallel for” para paralelizar bucles “for”. En este ejemplo en concreto, se paraleliza el “for” más externo, por lo que, dentro de cada hilo, habrá un bucle “for” que no estará paralelizado. La salida por pantalla que realiza este código es un indicador de qué iteración ha realizado cada thread. Como podemos ver, cada vez que se ejecuta, el orden es diferente.

```
Aplicaciones ▾ Lugares ▾ Terminal ▾ dom, 29 de abr, 22:41
root@Kali: ~/Universidad/PPD/Prácticas/Práctica 9/RecursosPracticaOpenMP

Archivo Editar Ver Buscar Terminal Ayuda
El thread 0, de 4 threads, calcula la iteracion i = 2
El thread 3, de 4 threads, calcula la iteracion i = 8
El thread 3, de 4 threads, calcula la iteracion i = 9
root@Kali:~/Universidad/PPD/Prácticas/Práctica 9/RecursosPracticaOpenMP# ./c5
El thread 0, de 4 threads, calcula la iteracion i = 0
El thread 0, de 4 threads, calcula la iteracion i = 1
El thread 0, de 4 threads, calcula la iteracion i = 2
El thread 3, de 4 threads, calcula la iteracion i = 8
El thread 3, de 4 threads, calcula la iteracion i = 9
El thread 2, de 4 threads, calcula la iteracion i = 6
El thread 2, de 4 threads, calcula la iteracion i = 7
El thread 1, de 4 threads, calcula la iteracion i = 3
El thread 1, de 4 threads, calcula la iteracion i = 4
El thread 1, de 4 threads, calcula la iteracion i = 5
root@Kali:~/Universidad/PPD/Prácticas/Práctica 9/RecursosPracticaOpenMP# ./c5
El thread 2, de 4 threads, calcula la iteracion i = 6
El thread 2, de 4 threads, calcula la iteracion i = 7
El thread 1, de 4 threads, calcula la iteracion i = 3
El thread 1, de 4 threads, calcula la iteracion i = 4
El thread 1, de 4 threads, calcula la iteracion i = 5
El thread 3, de 4 threads, calcula la iteracion i = 8
El thread 3, de 4 threads, calcula la iteracion i = 9
El thread 0, de 4 threads, calcula la iteracion i = 0
El thread 0, de 4 threads, calcula la iteracion i = 1
El thread 0, de 4 threads, calcula la iteracion i = 2
root@Kali:~/Universidad/PPD/Prácticas/Práctica 9/RecursosPracticaOpenMP# ./c5
El thread 0, de 4 threads, calcula la iteracion i = 0
El thread 0, de 4 threads, calcula la iteracion i = 1
El thread 2, de 4 threads, calcula la iteracion i = 6
El thread 2, de 4 threads, calcula la iteracion i = 7
El thread 3, de 4 threads, calcula la iteracion i = 8
El thread 1, de 4 threads, calcula la iteracion i = 3
El thread 1, de 4 threads, calcula la iteracion i = 4
El thread 1, de 4 threads, calcula la iteracion i = 5
El thread 0, de 4 threads, calcula la iteracion i = 2
El thread 3, de 4 threads, calcula la iteracion i = 9
root@Kali:~/Universidad/PPD/Prácticas/Práctica 9/RecursosPracticaOpenMP#
```

6.) Para realizar este ejercicio, paralelizaremos el bucle “for” más externo para que cada fila la calcule un thread. Por lo tanto quedaría así:

```
Aplicaciones ▾ Lugares ▾ Terminal ▾ dom, 29 de abr, 23:18
root@Kali: ~/Universidad/PPD/Prácticas/Práctica 9

Archivo Editar Ver Buscar Terminal Ayuda
El thread 0, de 4 threads, calcula la iteracion i = 0: 0+0*1=0
El thread 0, de 4 threads, calcula la iteracion i = 0: 0+1*2=2
El thread 0, de 4 threads, calcula la iteracion i = 0: 2+2*3=8
El thread 0, de 4 threads, calcula la iteracion i = 0: 0+0*2=0
El thread 0, de 4 threads, calcula la iteracion i = 0: 0+1*3=3
El thread 0, de 4 threads, calcula la iteracion i = 0: 3+2*4=11
El thread 2, de 4 threads, calcula la iteracion i = 2: 0+2*0=0
El thread 2, de 4 threads, calcula la iteracion i = 2: 0+3*1=3
El thread 2, de 4 threads, calcula la iteracion i = 2: 3+4*2=11
El thread 2, de 4 threads, calcula la iteracion i = 2: 0+2*1=2
El thread 2, de 4 threads, calcula la iteracion i = 2: 2+3*2=8
El thread 2, de 4 threads, calcula la iteracion i = 2: 8+4*3=20
El thread 2, de 4 threads, calcula la iteracion i = 2: 0+2*2=4
El thread 2, de 4 threads, calcula la iteracion i = 2: 4+3*3=13
El thread 2, de 4 threads, calcula la iteracion i = 2: 13+4*4=29
El thread 1, de 4 threads, calcula la iteracion i = 1: 0+1*0=0
El thread 1, de 4 threads, calcula la iteracion i = 1: 0+2*1=2
El thread 1, de 4 threads, calcula la iteracion i = 1: 2+3*2=8
El thread 1, de 4 threads, calcula la iteracion i = 1: 0+1*1=1
El thread 1, de 4 threads, calcula la iteracion i = 1: 1+2*2=5
El thread 1, de 4 threads, calcula la iteracion i = 1: 5+3*3=14
El thread 1, de 4 threads, calcula la iteracion i = 1: 0+1*2=2
El thread 1, de 4 threads, calcula la iteracion i = 1: 2+2*3=8
El thread 1, de 4 threads, calcula la iteracion i = 1: 8+3*4=20
El resultado paralelo es:
5      8      11
8      14     20
11     20     29
root@Kali:~/Universidad/PPD/Prácticas/Práctica 9#
```

La salida por consola del programa ha sido dividida por elemento de la matriz para que se vean los pasos a seguir en cada iteración. Como no se ve demasiado bien la salida del programa, la pongo aquí:

Primero realizamos la multiplicacion secuencial:

El resultado secuencial es:

```
5      8      11
8      14     20
11     20     29
```

A continuacion, realizaremos la multiplicacion paralela:

```
El thread 0, de 4 threads, calcula la iteracion i = 0: 0+0*0=0
El thread 0, de 4 threads, calcula la iteracion i = 0: 0+1*1=1
El thread 0, de 4 threads, calcula la iteracion i = 0: 1+2*2=5
```

```
El thread 0, de 4 threads, calcula la iteracion i = 0: 0+0*1=0
El thread 0, de 4 threads, calcula la iteracion i = 0: 0+1*2=2
El thread 0, de 4 threads, calcula la iteracion i = 0: 2+2*3=8
```

```
El thread 0, de 4 threads, calcula la iteracion i = 0: 0+0*2=0
El thread 0, de 4 threads, calcula la iteracion i = 0: 0+1*3=3
El thread 0, de 4 threads, calcula la iteracion i = 0: 3+2*4=11
```

```
El thread 2, de 4 threads, calcula la iteracion i = 2: 0+2*0=0
El thread 2, de 4 threads, calcula la iteracion i = 2: 0+3*1=3
El thread 2, de 4 threads, calcula la iteracion i = 2: 3+4*2=11
```

```
El thread 2, de 4 threads, calcula la iteracion i = 2: 0+2*1=2
El thread 2, de 4 threads, calcula la iteracion i = 2: 2+3*2=8
```

```

El thread 2, de 4 threads, calcula la iteracion i = 2: 8+4*3=20

El thread 2, de 4 threads, calcula la iteracion i = 2: 0+2*2=4
El thread 2, de 4 threads, calcula la iteracion i = 2: 4+3*3=13
El thread 2, de 4 threads, calcula la iteracion i = 2: 13+4*4=29

El thread 1, de 4 threads, calcula la iteracion i = 1: 0+1*0=0
El thread 1, de 4 threads, calcula la iteracion i = 1: 0+2*1=2
El thread 1, de 4 threads, calcula la iteracion i = 1: 2+3*2=8

El thread 1, de 4 threads, calcula la iteracion i = 1: 0+1*1=1
El thread 1, de 4 threads, calcula la iteracion i = 1: 1+2*2=5
El thread 1, de 4 threads, calcula la iteracion i = 1: 5+3*3=14

El thread 1, de 4 threads, calcula la iteracion i = 1: 0+1*2=2
El thread 1, de 4 threads, calcula la iteracion i = 1: 2+2*3=8
El thread 1, de 4 threads, calcula la iteracion i = 1: 8+3*4=20

El resultado paralelo es:
5      8      11
8      14     20
11     20     29

```

7.) En el código EjemploArea se suman los 10 elementos del vector “bases” y luego, se multiplican por 2 (valor de h).

En su versión secuencial, se realiza el cálculo y se imprime por pantalla.

En su versión paralelo 1, se utiliza la directiva “#pragma omp parallel private(i,tid)” para establecer las variables “i” e “id” como privadas a cada thread. Se realiza el calculo por cada thread y luego se suma en secuencial.

En la versión paralelo 2, se utiliza la directiva “#pragma omp parallel private(i,tid)” para establecer las variables “i” e “id” como privadas para cada thread. Dentro de dicha sección se realiza el cálculo paralelo y luego, con la directiva “#pragma omp critical” se realiza la suma como sección critica. Esto hace que el resultado final varíe debido a que el operador “+=” apunta a la misma variable compartida sin tener en cuenta las condiciones de concurso.

En la versión paralelo 3, se utiliza la directiva “#pragma omp parallel for reduction(+:total_base) private(i,tid)” para paralelizar el código del cálculo. Al contener la cláusula “reduction”, la variable “total_base” contendrá, al finalizar la zona paralela” la suma de las sumas parciales realizadas anteriormente. Esto hace que no haya problemas de concurso como en la versión paralelo 2.

El resultado mostrado en pantalla es el siguiente:

```
Aplicaciones ▾ Lugares ▾ Terminal ▾ lun, 30 de abr, 14:28
root@Kali: ~/Universidad/PPD/Prácticas/Práctica 9/RecursosPracticaOpenMP

Archivo Editar Ver Buscar Terminal Ayuda
El thread 0 calcula la iteracion i = 1
El thread 0 calcula la iteracion i = 2
El thread 1 calcula la iteracion i = 3
El thread 1 calcula la iteracion i = 4
El thread 1 calcula la iteracion i = 5
Solución con reducción: 110
root@Kali:~/Universidad/PPD/Prácticas/Práctica 9/RecursosPracticaOpenMP# ./EjemploArea
Solución secuencial: 110
Solución paralela 1: 110
Solución paralela 2: 318
El thread 2 calcula la iteracion i = 6
El thread 2 calcula la iteracion i = 7
El thread 1 calcula la iteracion i = 3
El thread 1 calcula la iteracion i = 4
El thread 1 calcula la iteracion i = 5
El thread 0 calcula la iteracion i = 0
El thread 0 calcula la iteracion i = 1
El thread 0 calcula la iteracion i = 2
El thread 3 calcula la iteracion i = 8
El thread 3 calcula la iteracion i = 9
Solución con reducción: 110
root@Kali:~/Universidad/PPD/Prácticas/Práctica 9/RecursosPracticaOpenMP# ./EjemploArea
Solución secuencial: 110
Solución paralela 1: 110
Solución paralela 2: 258
El thread 2 calcula la iteracion i = 6
El thread 2 calcula la iteracion i = 7
El thread 0 calcula la iteracion i = 0
El thread 0 calcula la iteracion i = 1
El thread 0 calcula la iteracion i = 2
El thread 1 calcula la iteracion i = 3
El thread 1 calcula la iteracion i = 4
El thread 1 calcula la iteracion i = 5
El thread 3 calcula la iteracion i = 8
El thread 3 calcula la iteracion i = 9
Solución con reducción: 110
root@Kali:~/Universidad/PPD/Prácticas/Práctica 9/RecursosPracticaOpenMP#
```

8.) Para realizar este ejercicio, solo tenemos que cambiar la directiva “#pragma omp for schedule(static, chunk)” por “#pragma omp for reduction(+:suma) schedule(static, chunk)” y modificar la línea que realiza el cálculo “c[i] = a[i] * b[i];” y poner “suma += a[i] * b[i];”. Con estos cambios, conseguimos que la variable suma se combine en el thread maestro una vez que ha terminado la ejecución de la zona paralela.

```
Aplicaciones ▾ Lugares ▾ Terminal ▾ lun, 30 de abr, 16:43
root@Kali: ~/Universidad/PPD/Prácticas/Práctica 9

Archivo Editar Ver Buscar Terminal Ayuda
NetBeans
La suma secuencial es: 285
A continuacion, la suma paralela:
El thread 0, de 4 threads, calcula la iteracion i = 0: 0*0=0
El thread 0, de 4 threads, calcula la iteracion i = 1: 1*1=1
El thread 0, de 4 threads, calcula la iteracion i = 8: 8*8=64
El thread 0, de 4 threads, calcula la iteracion i = 9: 9*9=81
El thread 1, de 4 threads, calcula la iteracion i = 2: 2*2=4
El thread 1, de 4 threads, calcula la iteracion i = 3: 3*3=9
El thread 2, de 4 threads, calcula la iteracion i = 4: 4*4=16
El thread 2, de 4 threads, calcula la iteracion i = 5: 5*5=25
El thread 3, de 4 threads, calcula la iteracion i = 6: 6*6=36
El thread 3, de 4 threads, calcula la iteracion i = 7: 7*7=49
La suma paralela es: 285
root@Kali:~/Universidad/PPD/Prácticas/Práctica 9# ./e4
Primero realizaremos la suma secuencial:
La suma secuencial es: 285
A continuacion, la suma paralela:
El thread 0, de 4 threads, calcula la iteracion i = 0: 0*0=0
El thread 0, de 4 threads, calcula la iteracion i = 1: 1*1=1
El thread 0, de 4 threads, calcula la iteracion i = 8: 8*8=64
El thread 0, de 4 threads, calcula la iteracion i = 9: 9*9=81
El thread 1, de 4 threads, calcula la iteracion i = 2: 2*2=4
El thread 1, de 4 threads, calcula la iteracion i = 3: 3*3=9
El thread 3, de 4 threads, calcula la iteracion i = 6: 6*6=36
El thread 3, de 4 threads, calcula la iteracion i = 7: 7*7=49
El thread 2, de 4 threads, calcula la iteracion i = 4: 4*4=16
El thread 2, de 4 threads, calcula la iteracion i = 5: 5*5=25
La suma paralela es: 285
root@Kali:~/Universidad/PPD/Prácticas/Práctica 9#
```