

2º curso / 2º cuatr.  
Grado Ing. Inform.  
Doble Grado Ing.  
Inform. y Mat.

## Arquitectura de Computadores (AC)

### Cuaderno de prácticas.

### Bloque Práctico 3. Programación paralela III: Interacción con el entorno en OpenMP

Estudiante (nombre y apellidos): David Gil Bautista

Grupo de prácticas: C1

Fecha de entrega:

Fecha evaluación en clase:

### Ejercicios basados en los ejemplos del seminario práctico

1. Usar la cláusula `num_threads(x)` en el ejemplo del seminario `if_clause.c`, y añadir un parámetro de entrada al programa que fije el valor `x` que se va a usar en la cláusula. Incorporar en el cuaderno de trabajo de esta práctica volcados de pantalla con ejemplos de ejecución que ilustren la funcionalidad de esta cláusula y explicar por qué lo ilustran.

#### CÓDIGO FUENTE: `if-clauseModificado.c`

```
/* Tipo de letra Courier new o Liberation Mono. Tamaño 8 o 9 .*/  
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/  
/* INTERLINEADO SENCILLO */  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <omp.h>  
  
int main(int argc, char ** argv)  
{  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <time.h>  
#include <omp.h>  
  
int main(int argc, char **argv)  
{  
    int i, n=20, tid, threads;  
    int a[n], suma=0, sumalocal;  
    if(argc < 3) {  
        fprintf(stderr, "Uso: ./ifclause iteraciones threads");  
        exit(-1);  
    }  
  
    threads=atoi(argv[2]);  
  
    n = atoi(argv[1]); if (n>20) n=20;  
    for (i=0; i<n; i++) {  
        a[i] = i;  
    }  
  
#pragma omp parallel if(n>4) default(none) num_threads(threads) \
```

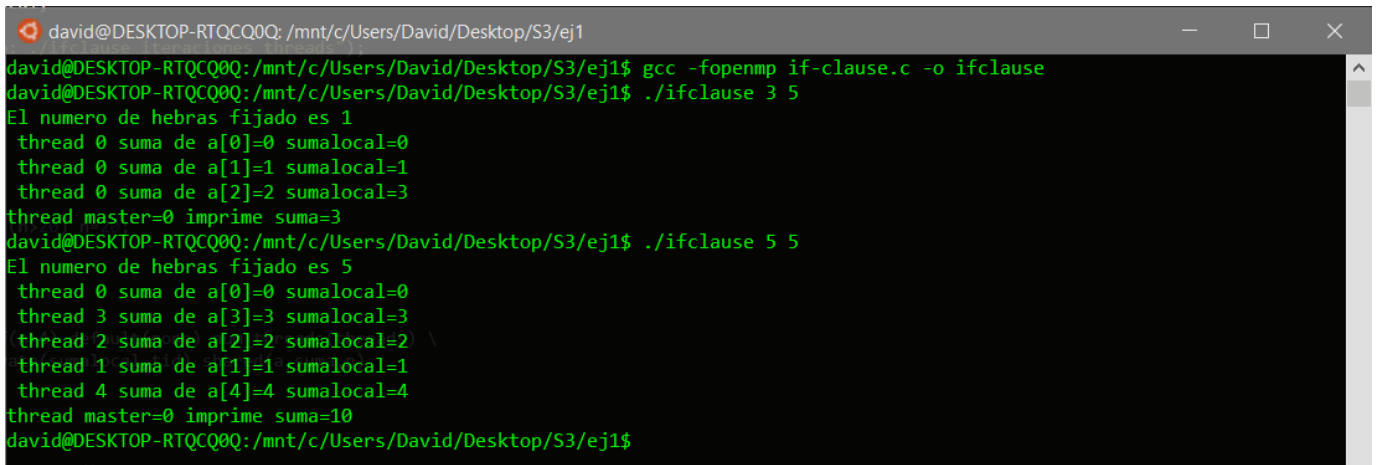
```

private(sumalocal,tid) shared(a,suma,n)
{
    sumalocal=0;
    int t = omp_get_num_threads();

    #pragma omp single
    {
        printf("El numero de hebras fijado es %d\n",t );
        tid=omp_get_thread_num();
    }
    #pragma omp for private(i) schedule(static) nowait
    for (i=0; i<n; i++)
    {
        sumalocal += a[i];
        printf(" thread %d suma de a[%d]=%d sumalocal=%d \n",
            tid,i,a[i],sumalocal);
    }
    #pragma omp atomic
    suma += sumalocal;
    #pragma omp barrier
    #pragma omp master
    printf("thread master=%d imprime suma=%d\n",tid,suma);
}

return(0);
}
}

```

**CAPTURAS DE PANTALLA:**


```

david@DESKTOP-RTQCQ0Q: /mnt/c/Users/David/Desktop/S3/ej1
david@DESKTOP-RTQCQ0Q:/mnt/c/Users/David/Desktop/S3/ej1$ gcc -fopenmp if-clause.c -o ifclause
david@DESKTOP-RTQCQ0Q:/mnt/c/Users/David/Desktop/S3/ej1$ ./ifclause 3 5
El numero de hebras fijado es 1
thread 0 suma de a[0]=0 sumalocal=0
thread 0 suma de a[1]=1 sumalocal=1
thread 0 suma de a[2]=2 sumalocal=3
thread master=0 imprime suma=3
david@DESKTOP-RTQCQ0Q:/mnt/c/Users/David/Desktop/S3/ej1$ ./ifclause 5 5
El numero de hebras fijado es 5
thread 0 suma de a[0]=0 sumalocal=0
thread 3 suma de a[3]=3 sumalocal=3
thread 2 suma de a[2]=2 sumalocal=2
thread 1 suma de a[1]=1 sumalocal=1
thread 4 suma de a[4]=4 sumalocal=4
thread master=0 imprime suma=10
david@DESKTOP-RTQCQ0Q:/mnt/c/Users/David/Desktop/S3/ej1$

```

**RESPUESTA:**

Al poner la cláusula lo primero que hace es comparar el if, para la primera ejecución hemos dado un valor de 3 iteraciones por lo que se ejecutaría secuencialmente pero en la segunda al poner 5 se cumple el if y el código se ejecuta paralelamente modificando el número de threads a 5.

2. (a) Rellenar la Tabla 1 (se debe poner en la tabla el id del *thread* que ejecuta cada iteración) ejecutando los ejemplos del seminario `schedule-clause.c`, `scheduled-clause.c` y `scheduleg-clause.c` con dos *threads* (0,1) y unas entradas de:

- iteraciones: 16 (0,...15)
- chunk= 1, 2 y 4

**Tabla 1 .** Tabla schedule. En la segunda fila, 1, 2 4 representan el tamaño del chunk (consulte seminario)

Iteración	schedule-clause.c			schedule-clause.d.c			schedule-clauseg.c		
	1	2	4	1	2	4	1	2	4
0	1	1	0	1	0	1	1	0	0
1	1	1	0	0	0	0	1	0	1
2	1	1	0	0	0	0	1	0	1
3	1	1	0	0	0	0	1	0	1
4	1	0	0	0	0	0	1	0	1
5	1	0	0	0	0	0	0	0	0
6	1	0	0	0	0	0	0	0	0
7	1	0	0	0	0	0	0	0	0
8	0	0	1	0	0	0	0	0	0
9	0	0	1	0	0	0	0	0	0
10	0	0	1	0	0	0	0	0	0
11	0	0	1	0	0	0	0	0	0
12	0	1	1	0	0	0	0	1	1
13	0	1	1	0	0	1	0	1	1
14	0	1	1	0	1	1	0	1	1
15	0	1	1	1	1	1	1	1	1

(b) Rellenar otra tabla como la de la figura pero esta vez usando cuatro *threads* (0,1,2,3).

**Tabla 2 .** Tabla schedule. En la segunda fila, 1, 2 4 representan el tamaño del chunk (consulte seminario)

Iteración	schedule-clause.c			schedule-clause.d.c			schedule-clause.g.c		
	1	2	4	1	2	4	1	2	4
0	0	0	0	3	1	3	2	3	1
1	0	1	0	3	1	3	2	3	1
2	0	1	0	3	1	3	2	3	1
3	0	1	0	3	0	3	2	2	1
4	3	1	1	3	0	0	2	2	0
5	3	3	1	3	0	0	2	2	0
6	3	3	1	3	0	0	2	2	0
7	3	0	1	3	0	0	1	1	0
8	2	0	2	3	0	1	0	1	3
9	2	0	2	3	0	1	0	1	3
10	2	2	2	3	0	1	0	0	3
11	2	2	2	3	2	1	0	0	3
12	1	2	3	3	2	2	1	0	2
13	1	2	3	0	3	2	3	0	2
14	1	3	3	1	3	2	3	3	2
15	1	3	3	2	1	2	3	3	2

Escriba en el cuaderno de prácticas las diferencias en el comportamiento de `schedule()` con `static`, `dynamic` y `guided`.

#### RESPUESTA:

**Static:** Las iteraciones se asignan a las hebras en el proceso de compilación. Se divide el número de iteraciones entre el número de procesadores. Con `static` aseguramos que los procesadores van a ejecutar las iteraciones que le hemos asignado.

**Dynamic:** Las iteraciones se asignan automáticamente en tiempo de ejecución. Al ir asignándose durante la ejecución se produce un tiempo de sobrecarga que puede hacer al programa más lento que con la asignación estática. Con `dynamic` no se sabe las iteraciones que va a ejecutar cada procesador.

**Guided:** Al igual que con `dynamic` las iteraciones se asignan en tiempo de ejecución con la diferencia de que `guided` reparte más equitativamente las iteraciones. Con `dynamic` un procesador puede ejecutar 7 de 10 iteraciones y el otro 3 de esas 10. Con `guided` se consigue un reparto más uniforme.

3. Añadir al programa `scheduled-clause.c` lo necesario para que imprima el valor de las variables de control `dyn-var`, `nthreads-var`, `thread-limit-var` y `run-sched-var` dentro (debe imprimir sólo un thread) y fuera de la región paralela. Realizar varias ejecuciones usando variables de entorno para modificar estas variables de control antes de la ejecución. Incorporar en su cuaderno de prácticas volcados de pantalla de estas ejecuciones. ¿Se imprimen valores distintos dentro y fuera de la región paralela?

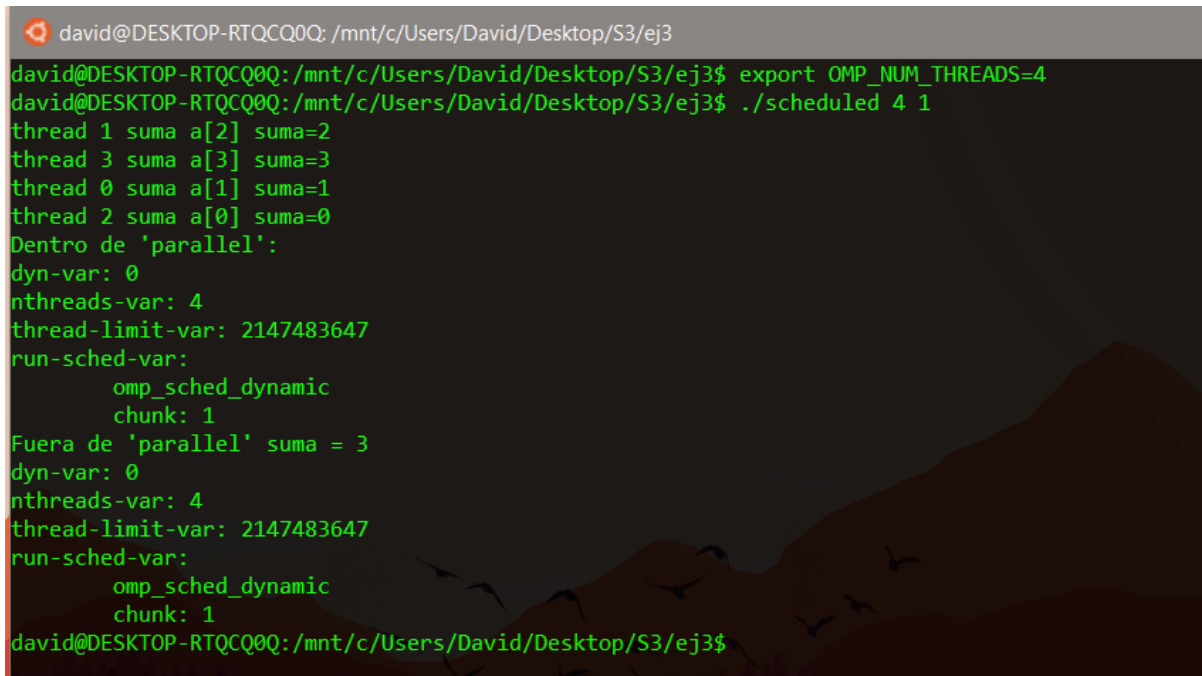
#### CÓDIGO FUENTE: `scheduled-clauseModificado.c`

```
/* Tipo de letra Courier new o Liberation Mono. Tamaño 8 o 9 */
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ */
/* INTERLINEADO SENCILLO */

#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

int main(int argc, char ** argv)
{
...
}
```

#### CAPTURAS DE PANTALLA:



```
david@DESKTOP-RTQCQ0Q: /mnt/c/Users/David/Desktop/S3/ej3
david@DESKTOP-RTQCQ0Q:/mnt/c/Users/David/Desktop/S3/ej3$ export OMP_NUM_THREADS=4
david@DESKTOP-RTQCQ0Q:/mnt/c/Users/David/Desktop/S3/ej3$ ./scheduled 4 1
thread 1 suma a[2] suma=2
thread 3 suma a[3] suma=3
thread 0 suma a[1] suma=1
thread 2 suma a[0] suma=0
Dentro de 'parallel':
dyn-var: 0
nthreads-var: 4
thread-limit-var: 2147483647
run-sched-var:
    omp_sched_dynamic
    chunk: 1
Fuera de 'parallel' suma = 3
dyn-var: 0
nthreads-var: 4
thread-limit-var: 2147483647
run-sched-var:
    omp_sched_dynamic
    chunk: 1
david@DESKTOP-RTQCQ0Q:/mnt/c/Users/David/Desktop/S3/ej3$
```

**Hebras disponibles = 4**

```
david@DESKTOP-RTQCQ0Q: /mnt/c/Users/David/Desktop/S3/ej3
david@DESKTOP-RTQCQ0Q: /mnt/c/Users/David/Desktop/S3/ej3$ export OMP_SCHEDULE="static"
david@DESKTOP-RTQCQ0Q: /mnt/c/Users/David/Desktop/S3/ej3$ ./scheduled 4 1
thread 2 suma a[2] suma=2
thread 1 suma a[1] suma=1
thread 3 suma a[3] suma=3
thread 0 suma a[0] suma=0
Dentro de 'parallel':
dyn-var: 0
nthreads-var: 4
thread-limit-var: 2147483647
run-sched-var:
    omp_sched_static
    chunk: 0
Fuera de 'parallel' suma = 3
dyn-var: 0
nthreads-var: 4
thread-limit-var: 2147483647
run-sched-var:
    omp_sched_static
    chunk: 0
david@DESKTOP-RTQCQ0Q: /mnt/c/Users/David/Desktop/S3/ej3$
```

### Cláusula estática

```
david@DESKTOP-RTQCQ0Q: /mnt/c/Users/David/Desktop/S3/ej3
david@DESKTOP-RTQCQ0Q: /mnt/c/Users/David/Desktop/S3/ej3$ export OMP_DYNAMIC=TRUE
david@DESKTOP-RTQCQ0Q: /mnt/c/Users/David/Desktop/S3/ej3$ ./scheduled 4 1
thread 3 suma a[1] suma=1
thread 2 suma a[2] suma=2
thread 0 suma a[0] suma=0
thread 1 suma a[3] suma=3
Dentro de 'parallel':
dyn-var: 1
nthreads-var: 4
thread-limit-var: 2147483647
run-sched-var:
    omp_sched_static
    chunk: 0
Fuera de 'parallel' suma = 3
dyn-var: 1
nthreads-var: 4
thread-limit-var: 2147483647
run-sched-var:
    omp_sched_static
    chunk: 0
david@DESKTOP-RTQCQ0Q: /mnt/c/Users/David/Desktop/S3/ej3$
```

### Asignación dinámica

```

david@DESKTOP-RTQCQ0Q: /mnt/c/Users/David/Desktop/S3/ej3
david@DESKTOP-RTQCQ0Q: /mnt/c/Users/David/Desktop/S3/ej3$ export OMP_THREAD_LIMIT=2
david@DESKTOP-RTQCQ0Q: /mnt/c/Users/David/Desktop/S3/ej3$ ./scheduled 4 1
thread 1 suma a[1] suma=1
thread 0 suma a[0] suma=0
thread 0 suma a[3] suma=3
thread 1 suma a[2] suma=3
Dentro de 'parallel':
dyn-var: 1
nthreads-var: 4
thread-limit-var: 2
run-sched-var:
    omp_sched_static
    chunk: 0
Fuera de 'parallel' suma = 3
dyn-var: 1
nthreads-var: 4
thread-limit-var: 2
run-sched-var:
    omp_sched_static
    chunk: 0
david@DESKTOP-RTQCQ0Q: /mnt/c/Users/David/Desktop/S3/ej3$

```

### Límite threads = 2

**RESPUESTA:** Se imprimen los mismos valores dentro y fuera. Los valores que hemos modificado previamente con el uso de export en la shell.

4. Usar en el ejemplo anterior las funciones `omp_get_num_threads()`, `omp_get_num_procs()` y `omp_in_parallel()` dentro y fuera de la región paralela. Imprimir los valores que obtienen estas funciones dentro (lo debe imprimir sólo uno de los threads) y fuera de la región paralela. Incorporar en su cuaderno de prácticas volcados de pantalla con los resultados de ejecución obtenidos. Indicar en qué funciones se obtienen valores distintos dentro y fuera de la región paralela.

### CÓDIGO FUENTE: scheduled-clauseModificado4.c

```

/* Tipo de letra Courier new o Liberation Mono. Tamaño 8 o 9 */
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ */
/* INTERLINEADO SENCILLO */

#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

int main(int argc, char ** argv)
{
    ...
}

```

### CAPTURAS DE PANTALLA:

```

david@DESKTOP-RTQCQ0Q: /mnt/c/Users/David/Desktop/S3/ej4
david@DESKTOP-RTQCQ0Q:/mnt/c/Users/David/Desktop/S3/ej4$ gcc -fopenmp scheduled_clauseModificado1.c -o sch
david@DESKTOP-RTQCQ0Q:/mnt/c/Users/David/Desktop/S3/ej4$ ./sch 6 3
thread 0 suma a[0] suma=0
thread 0 suma a[1] suma=1
thread 0 suma a[2] suma=3
thread 1 suma a[3] suma=3
thread 1 suma a[4] suma=7
thread 1 suma a[5] suma=12
Dentro de 'parallel':
omp_get_num_threads(): 2
omp_get_num_procs(): 4
omp_in_parallel(): True
dyn-var: 1
nthreads-var: 4
thread-limit-var: 2
run-sched-var:
    omp_sched_static
    chunk: 0
Fuera de 'parallel' suma = 12
omp_get_num_threads(): 1
omp_get_num_procs(): 4
omp_in_parallel(): False
dyn-var: 1
nthreads-var: 4
thread-limit-var: 2
run-sched-var:
    omp_sched_static
    chunk: 0
david@DESKTOP-RTQCQ0Q:/mnt/c/Users/David/Desktop/S3/ej4$

```

**RESPUESTA:**

En este caso el número de procesadores cambia de una región a otra, en la paralela podemos observar que hay 2 (límite del ejercicio anterior) y en la secuencial solo hay un procesador. Sin embargo, el número de procesadores num\_procs no varía puesto que antes también hemos especificado que había 4 disponibles.

5. Añadir al programa scheduled-clause.c lo necesario para modificar las variables de control dyn-var, nthreads-var y run-sched-var y para poder imprimir el valor de estas variables antes y después de dicha modificación. Incorporar en su cuaderno de prácticas volcados de pantalla con los resultados de ejecución obtenidos.

**CÓDIGO FUENTE:** scheduled-clauseModificado5.c

```

/* Tipo de letra Courier new o Liberation Mono. Tamaño 8 o 9 .*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

int main(int argc, char ** argv)
{
    ...
}

```

**CAPTURAS DE PANTALLA:**



```
david@DESKTOP-RTQCQ0Q: /mnt/c/Users/David/Desktop/S3/ej5
david@DESKTOP-RTQCQ0Q: /mnt/c/Users/David/Desktop/S3/ej5$ gcc -fopenmp scheduled_clauseModificado2.c -o sch
david@DESKTOP-RTQCQ0Q: /mnt/c/Users/David/Desktop/S3/ej5$ ./sch 5 1
dyn-var: 1
nthreads-var: 4
run-sched-var:
    omp_sched_static
    chunk: 0
Introduce dyn-var: 1
Introduce nthreads-var: 8
Introduce schedule_type: omp_sched_dynamic
Introduce chunk_size: 4
thread 0 suma a[0] suma=0
thread 0 suma a[2] suma=2
thread 0 suma a[3] suma=5
thread 0 suma a[4] suma=9
thread 1 suma a[1] suma=1
Fuera de 'parallel for' suma = 9
dyn-var: 1
nthreads-var: 8
run-sched-var:
    omp_sched_dynamic
    chunk: 4
david@DESKTOP-RTQCQ0Q: /mnt/c/Users/David/Desktop/S3/ej5$
```

```
david@DESKTOP-RTQCQ0Q: /mnt/c/Users/David/Desktop/S3/ej5
david@DESKTOP-RTQCQ0Q: /mnt/c/Users/David/Desktop/S3/ej5$ export OMP_THREAD_LIMIT=8
david@DESKTOP-RTQCQ0Q: /mnt/c/Users/David/Desktop/S3/ej5$ ./sch 10 1
dyn-var: 1
nthreads-var: 4
run-sched-var:
    omp_sched_static
    chunk: 0
Introduce dyn-var: 1
Introduce nthreads-var: 9
Introduce schedule_type: omp_sched_guided
Introduce chunk_size: 2
thread 2 suma a[0] suma=0
thread 2 suma a[4] suma=4
thread 2 suma a[5] suma=9
thread 2 suma a[6] suma=15
thread 2 suma a[7] suma=22
thread 2 suma a[8] suma=30
thread 2 suma a[9] suma=39
thread 0 suma a[1] suma=1
thread 1 suma a[2] suma=2
thread 3 suma a[3] suma=3
Fuera de 'parallel for' suma = 39
dyn-var: 1
nthreads-var: 9
run-sched-var:
    omp_sched_guided
    chunk: 2
david@DESKTOP-RTQCQ0Q: /mnt/c/Users/David/Desktop/S3/ej5$
```

**RESPUESTA:**

## Resto de ejercicios

6. Implementar un programa secuencial en C que multiplique una matriz triangular por un vector (use variables dinámicas). Compare el orden de complejidad del código que ha implementado con el código que implementó para el producto matriz por vector.

NOTAS: (1) el número de filas/columnas debe ser un argumento de entrada; (2) se debe inicializar las matrices antes del cálculo; (3) se debe imprimir siempre la primera y última componente del resultado antes de que termine el programa.

**CÓDIGO FUENTE:** pmtv-secuencial.c

```

/* Tipo de letra Courier new o Liberation Mono. Tamaño 8 o 9 .*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

int main(int argc, char ** argv)
{
...
}

```

**CAPTURAS DE PANTALLA:  
(ADJUNTAR CÓDIGO FUENTE AL .ZIP)**

```

david@DESKTOP-RTQCQ0Q: /mnt/c/Users/David/Desktop/S3/ej6
david@DESKTOP-RTQCQ0Q: /mnt/c/Users/David/Desktop/S3/ej6$ gcc -fopenmp pmtv-secuencial.c -o pmtv
david@DESKTOP-RTQCQ0Q: /mnt/c/Users/David/Desktop/S3/ej6$ ./pmtv 5
Matriz:
3 3 3 3 3
0 3 3 3 3
0 0 3 3 3
0 0 0 3 3
0 0 0 0 3
Vector:
5 5 5 5 5
Resultado:
75 60 45 30 15
david@DESKTOP-RTQCQ0Q: /mnt/c/Users/David/Desktop/S3/ej6$ ./pmtv 9
Matriz:
3 3 3 3 3 3 3 3 3
0 3 3 3 3 3 3 3 3
0 0 3 3 3 3 3 3 3
0 0 0 3 3 3 3 3 3
0 0 0 0 3 3 3 3 3
0 0 0 0 0 3 3 3 3
0 0 0 0 0 0 3 3 3
0 0 0 0 0 0 0 3 3
0 0 0 0 0 0 0 0 3
0 0 0 0 0 0 0 0 0
Vector:
5 5 5 5 5 5 5 5 5
Resultado:
135 120 105 90 75 60 45 30 15
david@DESKTOP-RTQCQ0Q: /mnt/c/Users/David/Desktop/S3/ej6$

```

7. Implementar en paralelo la multiplicación de una matriz triangular por un vector a partir del código secuencial realizado para el ejercicio anterior utilizando la directiva `for` de OpenMP. El código debe repartir entre los threads las iteraciones del bucle que recorre las filas. Dibujar en el cuaderno de prácticas la descomposición de dominio utilizada (Lección 4/Tema 2) en el código paralelo implementado para asignar tareas a los threads (Lección 5/Tema 2). Añadir lo necesario para que el usuario pueda fijar la planificación de tareas usando la variable de entorno `OMP_SCHEDULE`. Obtener en `atcgrid` los tiempos de ejecución del código paralelo (usando, como siempre, `-O2` al compilar) que multiplica una matriz triangular por un vector con las alternativas de planificación `static`, `dynamic` y `guided` para `chunk` de 1, 64 y el `chunk` por defecto para la alternativa. Use un tamaño de vector `N` múltiplo del número de cores y de 64 que no sea

inferior a 15360. El número de threads en las ejecuciones debe coincidir con el número de cores. Rellenar la Tabla 3 dos veces con los tiempos obtenidos. Representar el tiempo para static, dynamic y guided en función del tamaño del chunk en una gráfica. ¿Qué alternativa ofrece mejores prestaciones? Razone por qué. Incluya los scripts utilizado en el cuaderno de prácticas. NOTA: Nunca ejecute en atcgrid código que imprima todos los componentes del resultado.

Conteste a las siguientes preguntas: (a) ¿Qué valor por defecto usa OpenMP para chunk con static, dynamic y guided? Indique qué ha hecho para obtener este valor por defecto para cada alternativa. (b) ¿Qué número de operaciones de multiplicación y suma realizan cada uno de los threads en la asignación static para cada uno de los chunks? (c) Con la asignación dynamic y guided, ¿qué cree que debe ocurrir con el número de operaciones de multiplicación y suma que realizan cada uno de los threads?

#### RESPUESTA:

#### CÓDIGO FUENTE: pmtv-OpenMP.c

```
/* Tipo de letra Courier new o Liberation Mono. Tamaño 8 o 9 .*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

int main(int argc, char ** argv)
{
...
}
```

#### DESCOMPOSICIÓN DE DOMINIO:

#### CAPTURAS DE PANTALLA:

(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

#### TABLA RESULTADOS, SCRIPT Y GRÁFICA ATCGRID

SCRIPT: pmtv-OpenMP\_atcgrid.sh

**Tabla 3** .Tiempos de ejecución de la versión paralela del producto de una matriz triangular por un vector r **para vectores de tamaño N=** , 12 threads

Chunk	Static	Dynamic	Guided
por defecto			
1			
64			
Chunk	Static	Dynamic	Guided
por defecto			
1			
64			

8. Implementar un programa secuencial en C que calcule la multiplicación de matrices cuadradas, B y C:

$$A = B \bullet C; A(i,j) = \sum_{k=0}^{N-1} B(i,k) \bullet C(k,j), i,j = 0, \dots, N-1$$

NOTAS: (1) el número de filas/columnas debe ser un argumento de entrada; (2) se deben inicializar las matrices antes del cálculo; (3) se debe imprimir siempre las componentes (0,0) y (N-1, N-1) del resultado antes de que termine el programa.

**CÓDIGO FUENTE:** pmm-secuencial.c

```
/* Tipo de letra Courier new o Liberation Mono. Tamaño 8 o 9 .*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

int main(int argc, char ** argv)
{
...
}
```

**CAPTURAS DE PANTALLA:**  
(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```
david@DESKTOP-RTQCQ0Q: /mnt/c/Users/David/Desktop/S3/ej8
david@DESKTOP-RTQCQ0Q: /mnt/c/Users/David/Desktop/S3/ej8$ gcc -fopenmp pmm-secuencial.c -o p
david@DESKTOP-RTQCQ0Q: /mnt/c/Users/David/Desktop/S3/ej8$ ./p 2
M1:
2 2
2 2
M2:
1 1
1 1
Sol:
4 4
4 4
david@DESKTOP-RTQCQ0Q: /mnt/c/Users/David/Desktop/S3/ej8$ ./p 5
M1:
2 2 2 2 2
2 2 2 2 2
2 2 2 2 2
2 2 2 2 2
2 2 2 2 2
M2:
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
Sol:
10 10 10 10 10
10 10 10 10 10
10 10 10 10 10
10 10 10 10 10
10 10 10 10 10
david@DESKTOP-RTQCQ0Q: /mnt/c/Users/David/Desktop/S3/ej8$
```

9. Implementar en paralelo la multiplicación de matrices cuadradas con OpenMP a partir del código escrito en el ejercicio anterior. Use las directivas, las cláusulas y las funciones de entorno que considere oportunas. Se debe paralelizar también la inicialización de las matrices. Dibuje en su cuaderno de prácticas la descomposición de dominio que ha utilizado en el código paralelo implementado para asignar tareas a los threads (Lección 4/Tema 2, Lección 5/Tema 2).

10.

### DESCOMPOSICIÓN DE DOMINIO:

#### CÓDIGO FUENTE: pmm-OpenMP.c

```
/* Tipo de letra Courier new o Liberation Mono. Tamaño 8 o 9 */
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ */
/* INTERLINEADO SENCILLO */

#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

int main(int argc, char ** argv)
{
    ...
}
```

### CAPTURAS DE PANTALLA:

```
david@DESKTOP-RTQCQ0Q: /mnt/c/Users/David/Desktop/S3/ej9
david@DESKTOP-RTQCQ0Q: /mnt/c/Users/David/Desktop/S3/ej9$ gcc -fopenmp pmmomp.c -o pmm
david@DESKTOP-RTQCQ0Q: /mnt/c/Users/David/Desktop/S3/ej9$ ./pmm 2
M1:
1 1
1 1
M2:
2 2
2 2
Sol:
4 4
4 4
david@DESKTOP-RTQCQ0Q: /mnt/c/Users/David/Desktop/S3/ej9$ ./pmm 5
M1:
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
M2:
2 2 2 2 2
2 2 2 2 2
2 2 2 2 2
2 2 2 2 2
2 2 2 2 2
Sol:
10 10 10 10 10
10 10 10 10 10
10 10 10 10 10
10 10 10 10 10
10 10 10 10 10
david@DESKTOP-RTQCQ0Q: /mnt/c/Users/David/Desktop/S3/ej9$
```

10. Hacer un estudio de escalabilidad (ganancia en velocidad en función del número de cores) en atcgrid y en el PC local del código paralelo implementado para dos tamaños de las matrices. Debe recordar usar `-O2` al compilar. Presente los resultados del estudio en tablas de valores y en gráficas. Escoger los tamaños de manera que se observe diferentes curvas de escalabilidad en las gráficas que entregue en su cuaderno de prácticas (pruebe con valores de  $N$  entre 100 y 1500). Consulte la Lección 6/Tema 2. Incluya los scripts utilizado en el cuaderno de

prácticas. NOTA: Nunca ejecute en atcgrid código que imprima todos los componentes del resultado.

### ESTUDIO DE ESCALABILIDAD EN ATCGRID:

**SCRIPT:** pmm-OpenMP\_atcgrid.sh

### ESTUDIO DE ESCALABILIDAD EN PCLOCAL:

**SCRIPT:** pmm-OpenMP\_pclocal.sh

Tiempos PC Local:

Tiempo secuencial con 100

Tiempo = 0.004574600 Primera = 200 Ultima=200

Tiempo con dos threads con 100

Tiempo = 0.003335000 Primera = 200 Ultima=200

Tiempo con cuatro threads con 100

Tiempo = 0.001662000 Primera = 200 Ultima=200

-----

Tiempo secuencial con 500

Tiempo = 0.853643600 Primera = 1000 Ultima=1000

Tiempo con dos threads con 500

Tiempo = 0.530527000 Primera = 1000 Ultima=1000

Tiempo con cuatro threads con 500

Tiempo = 0.305063000 Primera = 1000 Ultima=1000

-----

Tiempo secuencial con 1000

Tiempo = 8.240074500 Primera = 2000 Ultima=2000

Tiempo con dos threads con 1000

Tiempo = 4.747283000 Primera = 2000 Ultima=2000

Tiempo con cuatro threads con 1000

Tiempo = 2.239729000 Primera = 2000 Ultima=2000

-----

Tiempo secuencial con 1500

Tiempo = 32.169772300 Primera = 3000 Ultima=3000

Tiempo con dos threads con 1500

Tiempo = 20.554163000 Primera = 3000 Ultima=3000

Tiempo con cuatro threads con 1500

Tiempo = 11.344185000      Primera = 3000      Ultima=3000

---

Dado que mi computador solo dispone de 4 procesadores no he hecho el cálculo para más hebras.

cores\tam	100	500	1000	1500
Secuencial	0.004574600	0.853643600	8.240074500	32.169772300
2 Cores	0.003335000	0.530527000	4.747283000	20.554163000
4 Cores	0.001662000	0.305063000	2.239729000	11.344185000

Tam 100:

2 Cores      1.371694153

4 Cores      2.752466907

Tam 500:

2 Cores      1.609048361

4 Cores      2.798253476

Tam 1000:

2 Cores      1.735745373

4 Cores      3.679049787

Tam 1500:

2 Cores      1.56512198

4 Cores      2.835794048

(Haría la gráfica pero lo he intentado y no tengo ni idea de como hacerla)

Se puede observar que al aumentar el número de procesadores se escala casi linealmente, es decir, si tenemos solo un procesador y añadimos otro el programa se ejecuta casi el doble de rápido. Aunque conforme el tamaño del problema aumenta la escalabilidad se reduce.

Para atcgrid obtendríamos unos resultados similares solo que la escalabilidad alcanzaría un máximo de 12 que es el número de cores físicos que tiene.