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):
Grupo de prácticas:
Fecha de entrega:
Fecha evaluación en clase:

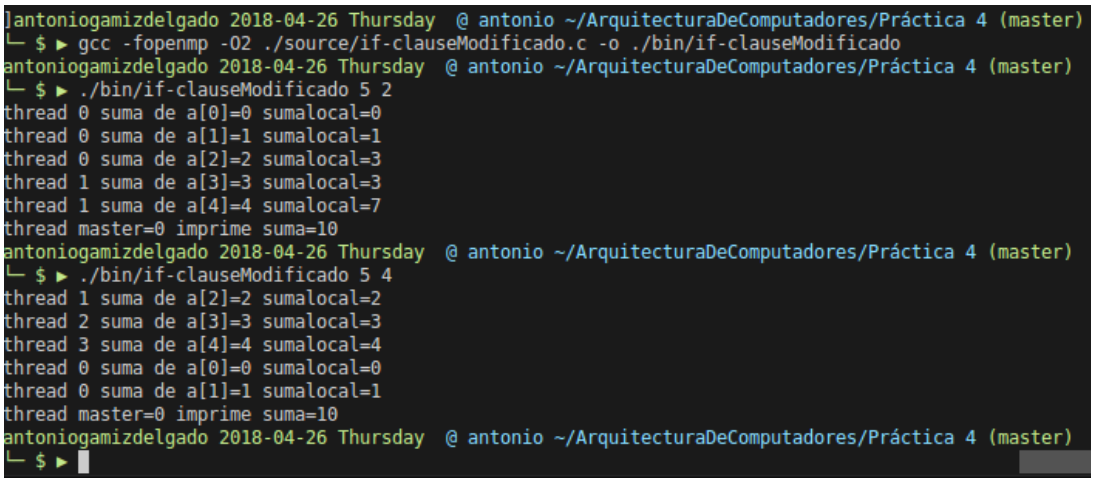
Ejercicios basados en los ejemplos del seminario práctico

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.

CAPTURA CÓDIGO FUENTE: `if-clauseModificado.c`

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

int main(int argc, char **argv)
{
    int i, n=20, tid, x;
    int a[n], suma=0, sumalocal;

    if( argc != 3 )
    {
        fprintf(stderr, "[Formato] ./if-clause <iteraciones> <n-threads> \n]");
        exit(-1);
    }

    n=atoi(argv[1]); if( n > 20 ) n=20;
    x=atoi(argv[2]); if( x > 8 ) n=8; //Adaptado a mi pc
    for(i=0; i<n; i++) a[i]=i;

    #pragma omp parallel if(n>4) default(none) private(sumalocal, tid) \
            shared(a, suma, n) num_threads(x)
    {
        sumalocal=0;
        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);
    }

}
```

CAPTURAS DE PANTALLA:



RESPUESTA:

Como vemos en las capturas, hemos ejecutado if-clauseModificado con los valores 2 y 4 para el número de threads, y como era de esperar al usar la cláusula num_threads(x) esa región parallel sólo se ha ejecutado con 2 y 4 threads cada uno (esto se puede ver al contar el número de identificadores (tid) diferentes que aparecen en la captura).

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)
- chunck= 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-claused.c			schedule-clauseg.c		
	1	2	4	1	2	4	1	2	4
0	0	0	0	0	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0
2	0	1	0	0	1	0	0	0	0
3	1	1	0	0	1	0	0	0	0
4	0	0	1	0	0	1	0	0	0
5	1	0	1	0	0	1	0	0	0
6	0	1	1	0	0	1	0	0	0
7	1	1	1	0	0	1	0	0	1
8	0	0	0	0	0	0	1	1	1
9	1	0	0	0	0	0	1	1	1
10	0	1	0	0	0	0	1	1	1
11	1	1	0	0	0	0	1	1	1
12	0	0	1	0	0	0	0	0	1
13	1	0	1	0	0	0	0	0	1
14	0	1	1	0	0	0	0	1	1
15	1	1	1	0	0	0	0	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-claused.c			schedule-clauseg.c		
	1	2	4	1	2	4	1	2	4
0	0	0	0	3	0	2	0	1	2
1	1	0	0	0	0	2	0	1	2
2	2	1	0	2	2	2	0	1	2
3	3	1	0	1	2	2	0	1	2
4	0	2	1	0	1	1	1	2	3
5	1	2	1	0	1	1	1	2	3
6	2	3	1	1	3	1	1	2	3
7	3	3	1	1	3	1	2	0	3
8	0	0	2	2	0	3	2	0	1
9	1	0	2	3	0	3	2	0	1
10	2	1	2	3	0	3	3	3	1
11	3	1	2	3	0	3	3	3	1
12	0	2	3	3	0	0	0	1	0
13	1	2	3	3	0	0	0	1	0
14	2	3	3	3	1	0	0	1	0
15	3	3	3	3	1	0	0	1	0

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

RESPUESTA:
Vemos que al usar static, los chunks (no las iteraciones) son repartidas en orden, es decir, uno para cada uno, ya que la asignación es estática, es decir, en tiempo de compilación.

Al usar dynamic, como su nombre indica, la distribución del trabajo se hace dinámicamente (por eso vemos las id's de los procesos más desordenados que en el schedule-clause). Además, vemos que no todos los threads ejecutan el mismo númeor de iteraciones, esto es debido a que si algunos threads son más rápidos que otros, entonces ejecutan más iteraciones que los demás.

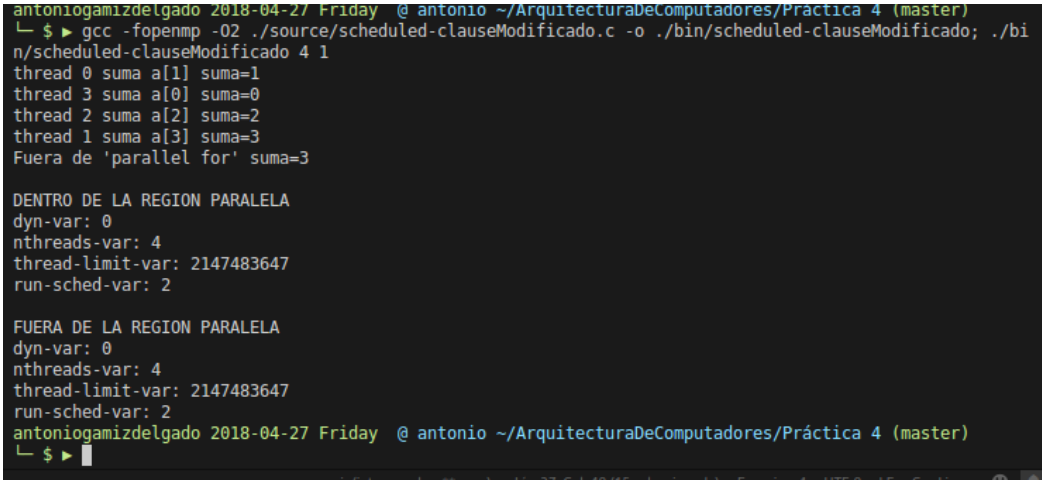
Al usar guided, vemos que el trabajo se reparte entre el número de threads, y luego el trabajo restante se vuelve a repartir entre los threads de nuevo, y así hasta que se acabe el número de iteraciones restantes.

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?

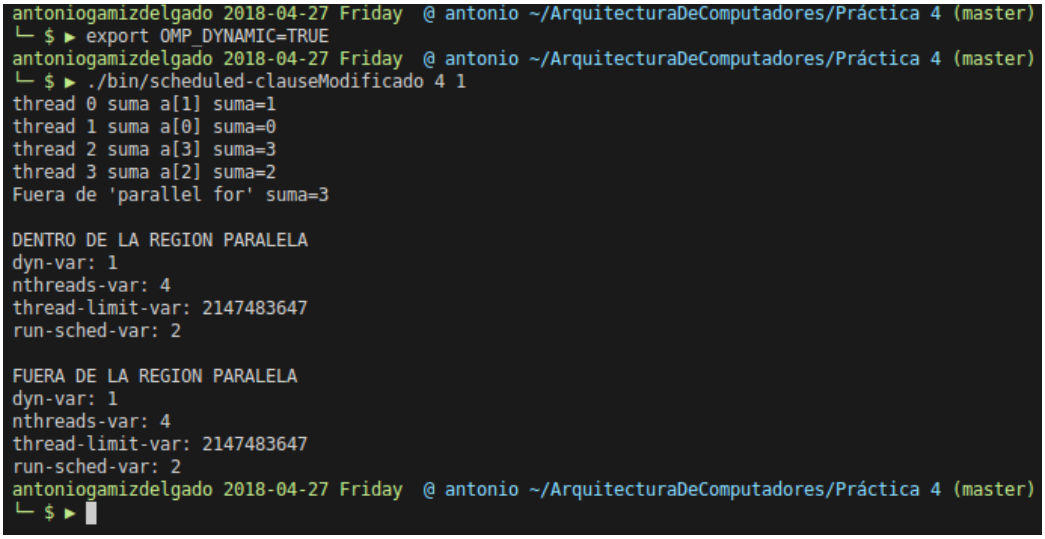
CAPTURA CÓDIGO FUENTE: `scheduled-clauseModificado.c`

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

int main(int argc, char **argv)
{
    int i, n=200, chunk, a[n], suma=0;

    if(argc != 3)
    {
        fprintf(stderr, "\nFalta chunk \n");
        exit(-1);
    }


    n=atoi(argv[1]); if( n>200 ) n=200;
    chunk=atoi(argv[2]);

    for(i=0; i<n; i++) a[i]=i;

    #pragma omp parallel
    {
        #pragma omp for firstprivate(suma) \
                lastprivate(suma) schedule(dynamic, chunk)
        for(i=0; i<n; i++)
        {
            suma+=a[i];
            printf("thread %d suma a[%d] suma=%d\n", omp_get_thread_num(), i, suma);
        }

        #pragma omp single
        {
            printf("Fuera de 'parallel for' suma=%d\n", suma);
            omp_sched_t kind;int chunk_size;
            omp_get_schedule(&kind, &chunk_size);

            printf("\nDENTRO DE LA REGION PARALELA\n");
            printf("dyn-var: %d\n", omp_get_dynamic());
            printf("nthreads-var: %d\n", omp_get_max_threads());
            printf("thread-limit-var: %d\n", omp_get_thread_limit());
            printf("run-sched-var: %d\n", kind);
        }
    }

    omp_sched_t kind;int chunk_size;
    omp_get_schedule(&kind, &chunk_size);

    printf("\nFUERA DE LA REGION PARALELA\n");
    printf("dyn-var: %d\n", omp_get_dynamic());
    printf("nthreads-var: %d\n", omp_get_max_threads());
    printf("thread-limit-var: %d\n", omp_get_thread_limit());
    printf("run-sched-var: %d\n", kind);

}
```

RESPUESTA:



Aquí ejecutamos el programa sin modificar ninguna variable de entorno y vemos que sale lo mismo fuera y dentro de la región paralela. De hecho, esto va a pasar siempre (salvo que entre los dos prints volvamos a modificar alguna de estas variables con las funciones de omp) ya que estemos en una región paralela o no, el número de threads máximo que podamos usar (nthreads-var) va a ser el mismo, así como el límite de threads (dependo del compilador) y dyn-var y run-sched-var.



Modifico dyn-var (OMP_DYNAMIC) a true para que el sistema operativo puede regular el número de threads.

```
antoniogamizdelgado 2018-04-27 Friday  @ antonio ~/ArquitecturaDeComputadores/Práctica 4 (master)
└─ $ ▶ export OMP_NUM_THREADS=8
antoniogamizdelgado 2018-04-27 Friday  @ antonio ~/ArquitecturaDeComputadores/Práctica 4 (master)
└─ $ ▶ ./bin/scheduled-clauseModificado 4 1
thread 7 suma a[2] suma=2
thread 3 suma a[1] suma=1
thread 0 suma a[0] suma=0
thread 1 suma a[3] suma=3
Fuera de 'parallel for' suma=3

DENTRO DE LA REGION PARALELA
dyn-var: 1
nthreads-var: 8
thread-limit-var: 2147483647
run-sched-var: 2

FUERA DE LA REGION PARALELA
dyn-var: 1
nthreads-var: 8
thread-limit-var: 2147483647
run-sched-var: 2
antoniogamizdelgado 2018-04-27 Friday  @ antonio ~/ArquitecturaDeComputadores/Práctica 4 (master)
└─ $ ▶
```

Modifico el número máximo de threads que puede usar mi programa (NUM_THREADS).

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.

CAPTURA CÓDIGO FUENTE: `scheduled-clauseModificado4.c`

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

int main(int argc, char **argv)
{
    int i, n=200, chunk, a[n], suma=0;

    if(argc != 3)
    {
        fprintf(stderr, "\nFalta chunk \n");
        exit(-1);
    }


    n=atoi(argv[1]); if( n>200 ) n=200;
    chunk=atoi(argv[2]);

    for(i=0; i<n; i++) a[i]=i;

    #pragma omp parallel
    {
        #pragma omp for firstprivate(suma) \
                lastprivate(suma) schedule(dynamic, chunk)
        for(i=0; i<n; i++)
        {
            suma+=a[i];
            printf("thread %d suma a[%d] suma=%d\n", omp_get_thread_num(), i, suma);
        }

        #pragma omp single
        {
            printf("Fuera de 'parallel for' suma=%d\n", suma);
            omp_sched_t kind;int chunk_size;
            omp_get_schedule(&kind, &chunk_size);

            printf("\nDENTRO DE LA REGION PARALELA\n");
            printf("dyn-var: %d\n", omp_get_dynamic());
            printf("nthreads-var: %d\n", omp_get_max_threads());
            printf("thread-limit-var: %d\n", omp_get_thread_limit());
            printf("run-sched-var: %d\n", kind);

            printf("Numero de threads: %d\n", omp_get_num_threads());
            printf("Numero de procesadores disponibles: %d\n", omp_get_num_procs());
            printf("En region paralela: %d\n", omp_in_parallel());
        }
    }

    omp_sched_t kind;int chunk_size;
    omp_get_schedule(&kind, &chunk_size);

    printf("\nFUERA DE LA REGION PARALELA\n");
    printf("dyn-var: %d\n", omp_get_dynamic());
    printf("nthreads-var: %d\n", omp_get_max_threads());
    printf("thread-limit-var: %d\n", omp_get_thread_limit());
    printf("run-sched-var: %d\n", kind);

    printf("Numero de threads: %d\n", omp_get_num_threads());
    printf("Numero de procesadores disponibles: %d\n", omp_get_num_procs());
    printf("En region paralela: %d\n", omp_in_parallel());
```

CAPTURAS DE PANTALLA:

```
antoniogamizdelgado 2018-04-27 Friday  @ antonio ~/ArquitecturaDeComputadores/Práctica 4 (master)
└─ $ ▶ gcc -fopenmp -O2 ./source/scheduled-clauseModificado4.c -o ./bin/scheduled-clauseModificado4
antoniogamizdelgado 2018-04-27 Friday  @ antonio ~/ArquitecturaDeComputadores/Práctica 4 (master)
└─ $ ▶ ./bin/scheduled-clauseModificado4 8 2
thread 6 suma a[2] suma=2
thread 6 suma a[3] suma=5
thread 1 suma a[4] suma=4
thread 1 suma a[5] suma=9
thread 0 suma a[0] suma=0
thread 0 suma a[1] suma=1
thread 5 suma a[6] suma=6
thread 5 suma a[7] suma=13
Fuera de 'parallel for' suma=13

DENTRO DE LA REGION PARALELA
dyn-var: 1
nthreads-var: 8
thread-limit-var: 2147483647
run-sched-var: 2
Numero de threads: 7
Numero de procesadores disponibles: 8
En region paralela: 1

FUERA DE LA REGION PARALELA
dyn-var: 1
nthreads-var: 8
thread-limit-var: 2147483647
run-sched-var: 2
Numero de threads: 1
Numero de procesadores disponibles: 8
En region paralela: 0
antoniogamizdelgado 2018-04-27 Friday  @ antonio ~/ArquitecturaDeComputadores/Práctica 4 (master)
└─ $ ▶
```

Anotación: Como vemos, el número de threads en la región paralela dice que es 7, en cambio el máximo está en 8, esto se debe a que a uno de los ejercicios anteriores actualicé la variable OMP_DYNAMIC a TRUE, por lo que el sistema operativo a visto oportuno quitar uno de los threads al programa.

```
antoniogamizdelgado 2018-04-27 Friday  @ antonio ~/ArquitecturaDeComputadores/Práctica 4 (master)
└─ $ ▶ export OMP_DYNAMIC=FALSE
antoniogamizdelgado 2018-04-27 Friday  @ antonio ~/ArquitecturaDeComputadores/Práctica 4 (master)
└─ $ ▶ gcc -fopenmp -O2 ./source/scheduled-clauseModificado4.c -o ./bin/scheduled-clauseModificado4
antoniogamizdelgado 2018-04-27 Friday  @ antonio ~/ArquitecturaDeComputadores/Práctica 4 (master)
└─ $ ▶ ./bin/scheduled-clauseModificado4 8 2
thread 1 suma a[0] suma=0
thread 1 suma a[1] suma=1
thread 5 suma a[2] suma=2
thread 5 suma a[3] suma=5
thread 4 suma a[6] suma=6
thread 4 suma a[7] suma=13
thread 0 suma a[4] suma=4
thread 0 suma a[5] suma=9
Fuera de 'parallel for' suma=13

DENTRO DE LA REGION PARALELA
dyn-var: 0
nthreads-var: 8
thread-limit-var: 2147483647
run-sched-var: 2
Numero de threads: 8
Numero de procesadores disponibles: 8
En region paralela: 1

FUERA DE LA REGION PARALELA
dyn-var: 0
nthreads-var: 8
thread-limit-var: 2147483647
run-sched-var: 2
Numero de threads: 1
Numero de procesadores disponibles: 8
En region paralela: 0
antoniogamizdelgado 2018-04-27 Friday  @ antonio ~/ArquitecturaDeComputadores/Práctica 4 (master)
└─ $ ▶
```

Como vemos, uno de los valores que cambia es el de en región paralela, que evidentemente varía porque en un printf estamos dentro de una y en el otro no.

Además también varía el número de threads (en región paralela 8 y fuera 1) ya que fuera de ella el programa se ejecuta de forma secuencial, es decir, solo lo ejecuta un thread, y dentro de ella, se están ejecutando todos los threads que hayamos especificado.

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.

CAPTURA CÓDIGO FUENTE: `scheduled-clauseModificado5.c`

```
#pragma omp single
{
    printf("Fuera de 'parallel for' suma=%d\n", suma);
    omp_sched_t kind; int chunk_size;
    omp_get_schedule(&kind, &chunk_size);

    printf("\nANTES DE LA MODIFICACION\n");
    printf("dyn-var: %d\n", omp_get_dynamic());
    printf("nthreads-var: %d\n", omp_get_max_threads());
    printf("run-sched-var: %d. Chunk: %d\n", kind, chunk_size);

    omp_set_dynamic(1);
    omp_set_num_threads(4);
    omp_set_schedule(kind, 2);

    printf("\nDESPUES DE LA MODIFICACION\n");
    printf("dyn-var: %d\n", omp_get_dynamic());
    printf("nthreads-var: %d\n", omp_get_max_threads());
    omp_get_schedule(&kind, &chunk_size);
    printf("run-sched-var: %d Chunk: %d\n", kind, chunk_size);
```

```
antoniogamizdelgado 2018-04-27 Friday  @ antonio ~/ArquitecturaDeComputadores/Práctica 4 (master)
└─ $ ▶ gcc -fopenmp -O2 ./source/scheduled-clauseModificado5.c -o ./bin/scheduled-clauseModificado5
antoniogamizdelgado 2018-04-27 Friday  @ antonio ~/ArquitecturaDeComputadores/Práctica 4 (master)
└─ $ ▶ ./bin/scheduled-clauseModificado5 8 1
thread 1 suma a[4] suma=4
thread 1 suma a[7] suma=11
thread 3 suma a[2] suma=2
thread 4 suma a[0] suma=0
thread 5 suma a[3] suma=3
thread 0 suma a[5] suma=5
thread 2 suma a[1] suma=1
thread 7 suma a[6] suma=6
Fuera de 'parallel for' suma=11

ANTES DE LA MODIFICACION
dyn-var: 0
nthreads-var: 8
run-sched-var: 2. Chunk: 1

DESPUES DE LA MODIFICACION
dyn-var: 1
nthreads-var: 4
run-sched-var: 2 Chunk: 2
antoniogamizdelgado 2018-04-27 Friday  @ antonio ~/ArquitecturaDeComputadores/Práctica 4 (master)
└─ $ ▶
```

Como vemos, he modificado la variable dyn-var (de false a true) para que el sistema operativo pueda ajustar los threads,. También he modificado el número de threads máximo que puede usar el programa en paralelo (de 8 a 4)., y también el valor de chunk de 1 a 2. El tipo de run-sched-var no lo he cambiado ya que no he puesto runtime en la clausula schedule de la región parallel.

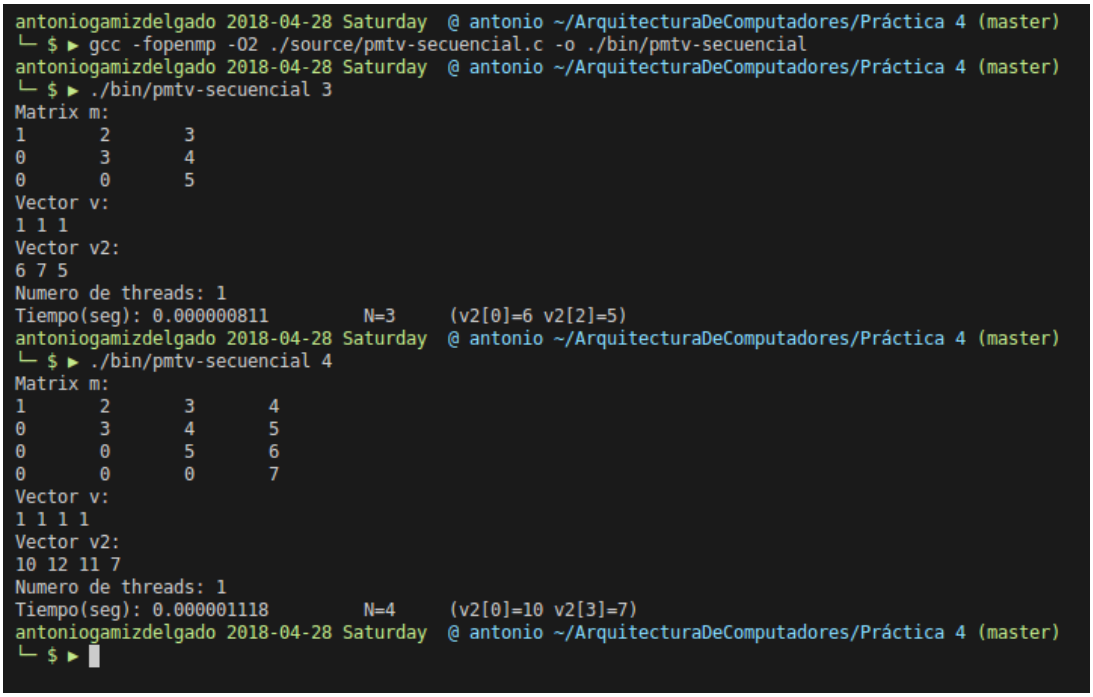
6. Implementar un programa secuencial en C que multiplique una matriz triangular por un vector (use variables dinámicas). Compare el orden de complejidad del código que ha implementado con el código que implementó para el producto matriz por vector.

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.

CAPTURA CÓDIGO FUENTE: `pmtv-secuencial.c`

```c
int main(int argc, char ** argv)
{
    if( argc<2 ) {
        fprintf(stderr, "Formato: %s <N>\n", argv[0]);
        exit(-1);
    }

    int N=atoi(argv[1]);

    int **m, *v, *v2;
    m= (int**) malloc(N*sizeof(int*));
    for(int i=0; i<N; i++) m[i]= (int*) malloc(N*sizeof(int));
    v= (int*) malloc(N*sizeof(int));
    v2= (int*) malloc(N*sizeof(int));

    if ((m == NULL) || (v == NULL) || (v2 == NULL))
    {
    printf("Error en la reserva de espacio para los vectores \n");
    exit(-2);
    }

    //Inicializamos la matriz m.
    for(int i=0; i<N; i++) for(int j=0; j<N; j++) m[i][j]=( i>j ) ? 0: i+j+1;

    //Inicializamos el vector v.
    for(int i=0; i<N; i++) v[i]=1;

    //Inicializamos el vector v2.
    for(int i=0; i<N; i++) v2[i]=0;

    //Calculamos la multiplicacion m*v en v2.
    double start = omp_get_wtime();
    for(int i=0; i<N; i++) for(int j=i; j<N; j++) v2[i]+=m[i][j]*v[j];
    double end = omp_get_wtime();

    //Mostramos matriz/vectores.
    #if MOSTRAR_MATRIZ
        printf("Matrix m: \n");
        for(int i=0; i<N; i++) {for(int j=0; j<N; j++) printf("%d\t", m[i][j]); printf("\n"); }
    #endif
    #if MOSTRAR_V
        printf("Vector v: \n");
        for(int i=0; i<N; i++) printf("%d ", v[i]); printf("\n");
    #endif
    #if MOSTRAR_V2
        printf("Vector v2: \n");
        for(int i=0; i<N; i++) printf("%d ", v2[i]); printf("\n");
    #endif

    #if MOSTRAR_PRIMERO_ULTIMO
        printf("Numero de threads: %d", omp_get_num_threads());
        printf("Tiempo(seg): %11.9f \t N=%d \t (v2[0]=%d v2[%d]=%d)\n", (float)(end-start), N, v2[0], N-1, v2[N-1]);
    #else
        printf("Tiempo(seg): %11.9f \t N=%d\n", (float)(end-start), N);
    #endif
```

CAPTURAS DE PANTALLA:



El orden de de comlejidad del algoritmo del BP2 era n^2, este sigue estando en el mismo orden aunque la constante es menor ($n^2/2 + n/2$).

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:

Respuesta a las cuestiones:
a.) Para obtener el valor de chunk por defecto de cada uno de los tipos de schedule, he hecho el siguiente programa usando las funciones de Open-Mp:

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


int main()
{
    omp_sched_t kind;
    int current_chunk;

    omp_get_schedule(&kind, &current_chunk);

    omp_set_schedule(omp_sched_static, current_chunk);
    omp_get_schedule(&kind, &current_chunk);
    printf("Valor de chunk para 'static': %d\n", current_chunk);

    omp_set_schedule(omp_sched_dynamic, current_chunk);
    omp_get_schedule(&kind, &current_chunk);
    printf("Valor de chunk para 'dynamic': %d\n", current_chunk);

    omp_set_schedule(omp_sched_guided, current_chunk);
    omp_get_schedule(&kind, &current_chunk);
    printf("Valor de chunk para 'guided': %d\n", current_chunk);

}
```

Dando el siguiente resultado al ejecutarlo:

```
antoniogamizdelgado 2018-04-28 Saturday  @ antonio ~/ArquitecturaDeComputadores/Práctica 4 (master)
└─ $ ▶ gcc -fopenmp -O2 ./source/valores-chunk.c -o ./bin/valores-chunk
antoniogamizdelgado 2018-04-28 Saturday  @ antonio ~/ArquitecturaDeComputadores/Práctica 4 (master)
└─ $ ▶ ./bin/valores-chunk
Valor de chunk para 'static': 1
Valor de chunk para 'dynamic': 1
Valor de chunk para 'guided': 1
antoniogamizdelgado 2018-04-28 Saturday  @ antonio ~/ArquitecturaDeComputadores/Práctica 4 (master)
└─ $ ▶
```

Por lo que vamos que el valor por defecto de chunk es 1 para todos los tipos de schedule.

b.) Realizan exactamente la misma cantidad de ITERACIONES (N/n-threads concretamente) pero al ser una matriz triangular cada thread hará un número de operaciones diferentes, ya que el bucle que hay dentro del bucle paralelizado empieza en i, es decir, para cada thread será diferente.

c.) Con la asignación dynamic y guided el número de ITERACIONES que ejecuta cada thread no va a ser, en general, igual, ya que por ejemplo en dynamic dependerá de la velocidad de ejecución de cada thread.

Aclaraciones sobre el programa: Le paso como argumento el tamaño de la matriz y el valor de chunk. Si el valor de chunk pasado es -1 entonces usará el valor por defecto, en caso convtrario, usará el valor dado. Lo ejecuto con N=21*12*64=16128.

CAPTURA CÓDIGO FUENTE: `pmtv-OpenMP.c`

```
int N=atoi(argv[1]);
int chunk=atoi(argv[2]);

int **m, *v, *v2;
m= (int**) malloc(N*sizeof(int*));
for(int i=0; i<N; i++) m[i]= (int*) malloc(N*sizeof(int));
v= (int*) malloc(N*sizeof(int));
v2= (int*) malloc(N*sizeof(int));

if ((m == NULL) || (v == NULL) || (v2 == NULL))
{
printf("Error en la reserva de espacio para los vectores \n");
exit(-2);
}

omp_sched_t kind;
int current_chunk;
omp_get_schedule(&kind, &current_chunk);
current_chunk=( chunk==-1 ) ? current_chunk : chunk;
omp_set_schedule(kind, current_chunk);

//Inicializamos la matriz m.
#pragma omp parallel for
for(int i=0; i<N; i++)
    for(int j=0; j<N; j++) m[i][j]=( i>j ) ? 0: i+j+1;

//Inicializamos el vector v.
#pragma omp parallel for
for(int i=0; i<N; i++) v[i]=1;

//Inicializamos el vector v2.
#pragma omp parallel for
for(int i=0; i<N; i++) v2[i]=0;

//Calculamos la multiplicacion m*v en v2.
double start = omp_get_wtime();
#pragma omp parallel for
for(int i=0; i<N; i++) for(int j=i; j<N; j++) v2[i]+=m[i][j]*v[j];
double end = omp_get_wtime();

//Mostramos matriz/vectores.
#if MOSTRAR_MATRIZ
    printf("Matrix m: \n");
    for(int i=0; i<N; i++) {for(int j=0; j<N; j++) printf("%d\t", m[i][j]); printf("\n"); }
#endif
#if MOSTRAR_V
    printf("Vector v: \n");
    for(int i=0; i<N; i++) printf("%d ", v[i]); printf("\n");
#endif
#if MOSTRAR_V2
    printf("Vector v2: \n");
    for(int i=0; i<N; i++) printf("%d ", v2[i]); printf("\n");
#endif

#if MOSTRAR_PRIMERO_ULTIMO
    printf("Tiempo(seg): %11.9f \t N=%d \t (v2[0]=%d v2[%d]=%d)\n", (float)(end-start), N, v2[0], N-1, v2[N-1]);
#else
    printf("Tiempo(seg): %11.9f \t N=%d\n", (float)(end-start), N);
#endif


for(int i=0; i<N; i++) free(m[i]);
free(m); free(v); free(v2);
```

DESCOMPOSICIÓN DE DOMINIO:

CAPTURAS DE PANTALLA:

```
antoniogamizdelgado 2018-04-28 Saturday  @ antonio ~/ArquitecturaDeComputadores/Práctica 4 (master)
└─ $ ▶ gcc -fopenmp -O2 ./source/pmtv-OpenMP.c -o ./bin/pmtv-OpenMP
antoniogamizdelgado 2018-04-28 Saturday  @ antonio ~/ArquitecturaDeComputadores/Práctica 4 (master)
└─ $ ▶
```

```
[E2estudiante14@atcgrid ~]$ mkdir BP3
[E2estudiante14@atcgrid ~]$ cd BP3/
[E2estudiante14@atcgrid BP3]$
```

```
Connected to atcgrid.ugr.es.
sftp> lpwd
Local working directory: /home/antonio
sftp> lcd ./ArquitecturaDeComputadores/Pr
Práctica 1/  Práctica 2/  Práctica 3/  Práctica 4/
sftp> lcd ./ArquitecturaDeComputadores/Práctica\ 4
sftp> pwd
Remote working directory: /home/E2estudiante14
sftp> cd ./BP3/
sftp> lcd ./scripts/
sftp> put clause.sh
Uploading clause.sh to /home/E2estudiante14/BP3/clause.sh
clause.sh                                     100%  539     0.5KB/s   00:00
sftp> put atcgrid_pmtv_omp.sh
Uploading atcgrid_pmtv_omp.sh to /home/E2estudiante14/BP3/atcgrid_pmtv_omp.sh
atcgrid_pmtv_omp.sh                           100%  695     0.7KB/s   00:00
sftp> lcd ../bin/
sftp> put pmtv-OpenMP
Uploading pmtv-OpenMP to /home/E2estudiante14/BP3/pmtv-OpenMP
pmtv-OpenMP                                   100%   13KB  13.3KB/s   00:00
sftp>
```

```
[E2estudiante14@atcgrid BP3]$ ls
atcgrid_pmtv_omp.sh  pmtv-OpenMP
[E2estudiante14@atcgrid BP3]$ qsub atcgrid_pmtv_omp.sh -q ac
76314.atcgrid
[E2estudiante14@atcgrid BP3]$ ls
atcgrid_pmtv_omp.o76314  atcgrid_pmtv_omp.sh  pmtv-OpenMP
[E2estudiante14@atcgrid BP3]$ ls
atcgrid_pmtv_omp.e76314  atcgrid_pmtv_omp.o76314  atcgrid_pmtv_omp.sh  pmtv-OpenMP
[E2estudiante14@atcgrid BP3]$
```

```
sftp> lcd ../output/
sftp> get atcgrid_pmtv_omp.o76314
Fetching /home/E2estudiante14/BP3/atcgrid_pmtv_omp.o76314 to atcgrid_pmtv_omp.o76314
/home/E2estudiante14/BP3/atcgrid_pmtv_omp.o76314                100% 1358     1.3KB/s   00:00
sftp> get atcgrid_pmtv_omp.e76314
Fetching /home/E2estudiante14/BP3/atcgrid_pmtv_omp.e76314 to atcgrid_pmtv_omp.e76314
sftp>
```

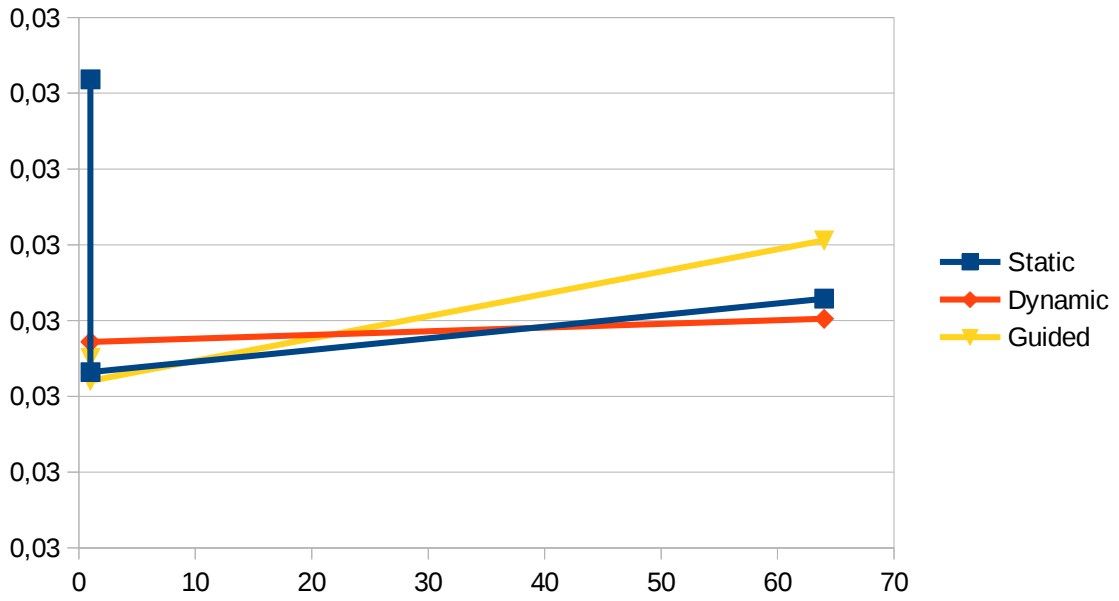
TABLA RESULTADOS, SCRIPT Y GRÁFICA atcgrid

SCRIPT: pmvt-OpenMP_Pcaula.shç

Este es el script para la ejecucion individual del usuario, el que he usado en atcgrid se puede ver en /scripts/atcgrid_pmvt_omp.sh

```
#!/bin/bash

echo "Script para elegir entre 'static', 'dynamic' o 'guided'"

if [ "$#" -ne 1 ]; then
    echo "Illegal number of parameters"
else
    if [ "$1" == "static" ]; then
        echo "'static' schedule elegido"
        export OMP_SCHEDULE="static"
    fi

    if [ "$1" == "dynamic" ]; then
        echo "'dynamic' schedule elegido"
        export OMP_SCHEDULE="dynamic"
    fi

    if [ "$1" == "guided" ]; then
        echo "'guided' schedule elegido"
        export OMP_SCHEDULE="guided"
    fi

    ./pmtv-OpenMP 16128 -1
fi
```

N=16128 N-Threads=12			
Chunk	**Static**	**Dynamic**	**Guided**
por defecto	0.032836519	0.032143977	0.032101970
1	0.032064024	0.032143466	0.032040995
64	0.032257881	0.032204717	0.032411627
Chunk	**Static**	**Dynamic**	**Guided**
por defecto	0.032887511	0.031785198	0.031838186
1	0.032092128	0.032198001	0.032199677
64	0.032040402	0.032212093	0.032156248



Vemos que el que más rápido se ejecuta es el que hace el reparte del trabajo de forma dinámica, esto se debe a que la matriz es triangular, por lo que el producto de la primera fila tiene más costo computacional que la segunda, y así hasta la última. De esta forma, al hacer un ajuste dinámico del trabajo, podemos hacer que los threads que han ejecutado las últimas filas ejecuten varías más que los que ejecutan las primeras.

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

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

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.

CAPTURA CÓDIGO FUENTE: `pmm-secuencial.c`

CAPTURAS DE PANTALLA:

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:

CAPTURA CÓDIGO FUENTE: `pmm-OpenMP.c`

CAPTURAS DE PANTALLA:

10. Hacer un estudio de escalabilidad (ganancia en velocidad en función del número de cores) en atcgrid y en su PC del código paralelo implementado para dos tamaños de las matrices. Debe recordar usar `-O2` al compilar. El número de núcleos máximo en este estudio debe ser el igual al de núcleos físicos del computador. 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