2º curso / 2º cuatr.

Grados Ing. Inform.

Arquitectura de Computadores (AC)

Cuaderno de prácticas.
Bloque Práctico 3. Programación paralela III: Interacción con el entorno en OpenMP

Estudiante (nombre y apellidos): QUINTÍN MESA ROMERO
Grupo de prácticas:

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;
    int a[n], suma=0, sumalocal, x;

    if (argc < 3)
    {
        fprintf(stderr,"[ERROR] en el número de argumentos(iteraciones y nº hebras\n");
        exit(-1);
    }

    n = atoi(argv[1]);
    x = atoi(argv[2]);

    if (n>20)
        n=20;

    if (x > 8) x = 8;   // Este PC tiene 8 CPUs; como máximo puede crear 8 hebras

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

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

CAPTURAS DE PANTALLA:

```
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informátic
a/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica3/bp3$ ./if-claus
e_modified 5 4
 thread 0 suma de a[0]=0 sumalocal=0
 thread 0 suma de a[1]=1 sumalocal=1
 thread 3 suma de a[4]=4 sumalocal=4
 thread 2 suma de a[3]=3 sumalocal=3
 thread 1 suma de a[2]=2 sumalocal=2
thread master=0 imprime suma=10
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informátic
a/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica3/bp3$ ./if-claus
e_modified 11 8
 thread 7 suma de a[10]=10 sumalocal=10
 thread 0 suma de a[0]=0 sumalocal=0
 thread 0 suma de a[1]=1 sumalocal=1
 thread 3 suma de a[6]=6 sumalocal=6
 thread 4 suma de a[7]=7 sumalocal=7
 thread 1 suma de a[2]=2 sumalocal=2
 thread 1 suma de a[3]=3 sumalocal=5
 thread 6 suma de a[9]=9 sumalocal=9
 thread 2 suma de a[4]=4 sumalocal=4
 thread 2 suma de a[5]=5 sumalocal=9
 thread 5 suma de a[8]=8 sumalocal=8
thread master=0 imprime suma=55
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informátic
a/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica3/bp3$
```

RESPUESTA: La captura del código contiene todo lo que se ha modificado del programa inicial.

2. Rellenar la Tabla 1 (se debe poner en la tabla el id del *thread* que ejecuta cada iteración) usando `scheduler-clause.c` con tres *threads* (0,1,2) y un número de iteraciones de 16 (0 a 15 en la tabla). Con este ejercicio se pretende comparar distintas alternativas de planificación de bucles. Se van a usar distintos tipos (`static`, `dynamic`, `guided`), modificadores (`monotonic` y `nonmonotonic`) y tamaños de chunk (x = 1 y 2).

Tabla 1 . Tabla schedule. Rellenar esta tabla ejecutando `scheduler-clause.c` asignando previamente a la variable de entorno `OMP_SCHEDULE` los valores que se indican en la tabla (por ej.: `export OMP_SCHEDULE="non-monotonic:static,2`). En la segunda fila, 1 y 2 representan el tamaño del chunk

Iteración	"monotonic:static,x"		"nonmonotonic:static,x"		"monotonic:dynamic,x"		"monotonic:guided,x"	
	x=1	x=2	x=1	x=2	x=1	x=2	x=1	x=2
0	0	0	0	0	2	0	0	0
1	1	0	1	0	0	0	0	0
2	2	1	2	1	1	1	0	0
3	0	1	0	1	0	1	0	0
4	1	2	1	2	0	2	0	0
5	2	2	2	2	0	2	0	0
6	0	0	0	0	0	0	2	1
7	1	0	1	0	0	0	2	1
8	2	1	2	1	0	0	2	1
9	0	1	0	1	0	0	2	1
10	1	2	1	2	0	0	1	2
11	2	2	2	2	0	0	1	2

12	0	0	0	0	0	0	0	0
13	1	0	1	0	0	0	0	0
14	2	1	2	1	0	1	2	0
15	0	1	0	1	0	1	2	0

```
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informátic
a/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica3/bp3$ export OMP
_NUM_THREADS=3
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informátic
a/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica3/bp3$ export OMP
_SCHEDULE="monotonic:static,1"
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informátic
a/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica3/bp3$ gcc -fopen
mp scheduler-clause.c -o scheduler
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informátic
a/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica3/bp3$ ./schedule
r 16
 thread 0 suma a[0]=0 suma=0
 thread 0 suma a[3]=3 suma=3
 thread 0 suma a[6]=6 suma=9
 thread 0 suma a[9]=9 suma=18
 thread 0 suma a[12]=12 suma=30
 thread 0 suma a[15]=15 suma=45
 thread 2 suma a[2]=2 suma=2
 thread 2 suma a[5]=5 suma=7
 thread 2 suma a[8]=8 suma=15
 thread 2 suma a[11]=11 suma=26
 thread 2 suma a[14]=14 suma=40
 thread 1 suma a[1]=1 suma=1
 thread 1 suma a[4]=4 suma=5
 thread 1 suma a[7]=7 suma=12
 thread 1 suma a[10]=10 suma=22
 thread 1 suma a[13]=13 suma=35
Fuera de 'parallel for' suma=45
```

Destacar las diferencias entre las 4 alternativas de planificación de la tabla, en particular, las que hay entre `static`, `dynamic` y `guided` y las diferencias entre usar `monotonic` y `nonmonotonic`.

RESPUESTA:

- **STATIC**: Las hebras se distribuyen en tiempo de compilación, las iteraciones se dividen en unidades de chunk iteraciones y las unidades se asignan en round-robin. El número de iteraciones que realiza cada hebra está predefinido y será equitativo, siempre y cuando el total sea múltiplo de las hebras.

-**DYNAMIC**: Las hebras se distribuyen en tiempo de ejecución, por lo que no sabemos qué iteraciones realizará cada hebra. La unidad de distribución tiene chunk iteraciones : num unidades = O(n/chunk). Los threads más rápidos ejecutan más unidades, pero como mínimo ejecutarán las chunk iteraciones.

-**GUIDED**: Las hebras se distribuyen también en tiempo de ejecución. Empezamos con bloque largo, cuyo tamaño va menguando: num_iteraciones_restantes / num_threads, y nunca es más pequeño que chunk. Las hebras más ociosas realizan más ejecuciones.

En cuanto a la diferencia de usar monotonic y nomonotonic, no existe diferencia a la hora de ejecución, solo una diferencia teórica.

3. ¿Qué valor por defecto usa OpenMP para `chunk` y `modifier` con `static`, `dynamic` y `guided`? Explicar qué ha hecho para contestar a esta pregunta.

Si es static con modifier se establece como monotic. En cualquier otro caso, se establece a nonmonotic. Las páginas web usadas son: https://www.openmp.org/spec-html/5.0/openmpse49.html

https://www.openmp.org/wp-content/uploads/SC17-Kale-LoopSchedforOMP_BoothTalk.pdf

4. 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`

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

    chunk = atoi(argv[2]);

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

    omp_sched_t kind; int chunk_v;

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

        if (i == 0)  // lo ejecute una sola hebra
        {
           omp_get_schedule(&kind, &chunk_v);

           printf("Dentro del 'parallel for'\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,  chunk = %d\n", kind, chunk_v);
        }
    }

    printf("Fuera de 'parallel for'\n");
    printf("suma=%d\n",suma);
    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, chunk = %d\n", kind, chunk);

}
```

CAPTURAS DE PANTALLA:

```
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informátic
a/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica3/bp3$ gcc -fopen
mp scheduled-clause.c -o scheduled-modified
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informátic
a/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica3/bp3$ export OMP
_NUM_THREADS=2
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informátic
a/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica3/bp3$ export OMP
_DYNAMIC=FALSE
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informátic
a/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica3/bp3$ export OMP
_SCHEDULE="static,4"
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informátic
a/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica3/bp3$ ./schedule
d-modified 6 4
 thread 0 suma a[0]=0 suma=0
Dentro del 'parallel for'
dyn-var = 0
nthreads-var = 2
thread-limit-var = 2147483647
run-sched-var = -2147483647,  chunk = 4
 thread 0 suma a[1]=1 suma=1
 thread 0 suma a[2]=2 suma=3
 thread 0 suma a[3]=3 suma=6
 thread 1 suma a[4]=4 suma=4
 thread 1 suma a[5]=5 suma=9
Fuera de 'parallel for'
suma=9
dyn-var = 0
nthreads-var = 2
thread-limit-var = 2147483647
run-sched-var = -2147483647, chunk = 4
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informátic
a/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica3/bp3$
```

RESPUESTA:

Se obtienen los mismo valores para las distintas variables tanto dentro como fuera de la región paralela. Estos valores se corresponden con lo que hemos definido en las variables de entorno.

NOTA: Te pone bien el kind con non-monotonic

5. 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-clauseModificado2.c`

```
omp_sched_t kind; int chunk_v;

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

    if (i == 0)  // lo ejecute una sola hebra
    {
       omp_get_schedule(&kind, &chunk_v);

       printf("////////////////////////////////////\n");
       printf("Dentro del 'parallel for'\n");
       printf("////////////////////////////////////\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,  chunk = %d\n", kind, chunk_v);
       printf("num_threads = %d\n", omp_get_num_threads());
       printf("num_procs = %d\n", omp_get_num_procs());
       printf("in parallel = %d\n", omp_in_parallel());
       printf("////////////////////////////////////\n");

    }
}

printf("////////////////////////////////////\n");
printf("Fuera de 'parallel for'\n");
printf("////////////////////////////////////\n");
printf("suma=%d\n",suma);
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, chunk = %d\n", kind, chunk);
printf("num_threads = %d\n", omp_get_num_threads());
printf("num_procs = %d\n", omp_get_num_procs());
printf("in parallel = %d\n", omp_in_parallel());
```

CAPTURAS DE PANTALLA:

```
a/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica3/bp3$ export OMP
_NUM_THREADS=3
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informátic
a/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica3/bp3$ export OMP
_DYNAMIC=FALSE
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informátic
a/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica3/bp3$ export OMP
_SCHEDULE="dynamic,2"
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informátic
a/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica3/bp3$ gcc -fopen
mp scheduled-clause.c -o scheduled-modified
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informátic
a/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica3/bp3$ ./schedule
d-modified 6 5
 thread 0 suma a[0]=0 suma=0
////////////////////////////////////
Dentro del 'parallel for'
////////////////////////////////////
dyn-var = 0
nthreads-var = 3
thread-limit-var = 2147483647
run-sched-var = 2,  chunk = 2
num_threads = 3
num_procs = 8
in parallel = 1
////////////////////////////////////
 thread 0 suma a[1]=1 suma=1
 thread 0 suma a[2]=2 suma=3
 thread 0 suma a[3]=3 suma=6
 thread 0 suma a[4]=4 suma=10
 thread 2 suma a[5]=5 suma=5
////////////////////////////////////
Fuera de 'parallel for'
////////////////////////////////////
suma=5
dyn-var = 0
nthreads-var = 3
thread-limit-var = 2147483647
run-sched-var = 2, chunk = 5
num_threads = 1
num_procs = 8
in parallel = 0
```

RESPUESTA:

La diferencia que se puede observar entre lo que se muestra fuera del parallel y dentro del parallel son los valores de las variables num_threads e in_parallel. Esto se debe a que dentro de la región paralela se están empleando las 3 hebras que se han definido con la variable de entorno OMP_NUM_THREADS, y fuera del parallel, como es código secuencial, lo ejecuta una sola hebra. Por su parte, la variable in_parallel, devuelve 1 si está en la región paralela o 0 si no lo está.

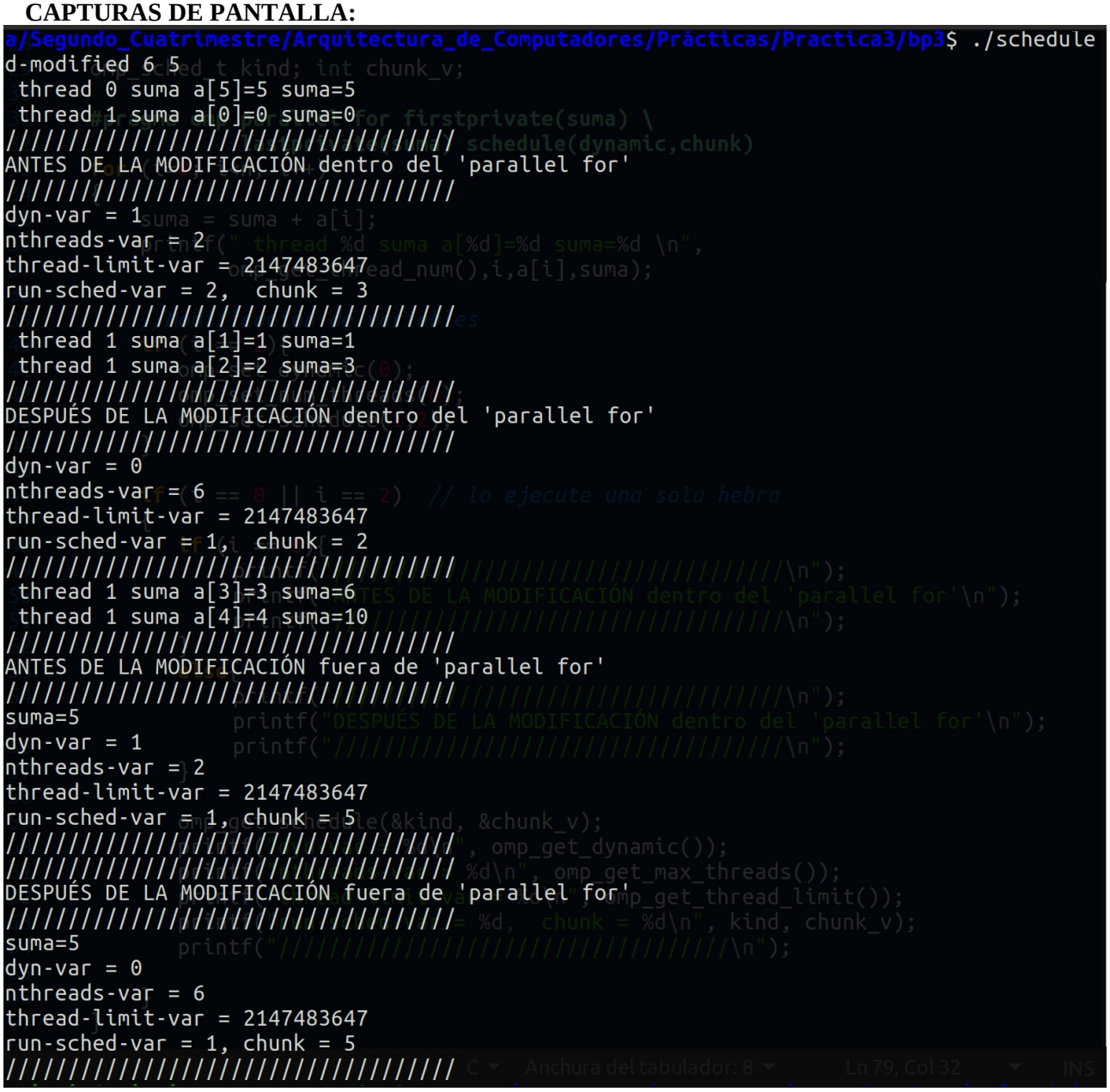
6. Añadir al programa `scheduled-clause.c` lo necesario para, usando funciones, modificar las variables de control `dyn-var`, `nthreads-var` y `run-sched-var` dentro de la región paralela y fuera de la región paralela. En la modificación de `run-sched-var` se debe usar un valor de `kind` distinto al utilizado en la cláusula `schedule()`. Añadir lo necesario para imprimir el contenido de estas variables antes y después de cada una de las dos modificaciones. Comentar los resultados.

CAPTURA CÓDIGO FUENTE: `scheduled-clauseModificado3.c`

```
omp_sched_t kind; int chunk_v;

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

    //Modificación de variables
    if (i == 2){
       omp_set_dynamic(0);
       omp_set_num_threads(6);
       omp_set_schedule(1,2);
    }

    if (i == 0 || i == 2)  // lo ejecute una sola hebra
    {
       if (i == 0){
            printf("////////////////////////////////////\n");
            printf("ANTES DE LA MODIFICACIÓN dentro del 'parallel for'\n");
            printf("////////////////////////////////////\n");
       }
       else{
            printf("////////////////////////////////////\n");
            printf("DESPUÉS DE LA MODIFICACIÓN dentro del 'parallel for'\n");
            printf("////////////////////////////////////\n");
       }

       omp_get_schedule(&kind, &chunk_v);
       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,  chunk = %d\n", kind, chunk_v);
       printf("////////////////////////////////////\n");

    }
}
```

```
    printf("////////////////////////////////////\n");
    printf("ANTES DE LA MODIFICACIÓN fuera de 'parallel for'\n");
    printf("////////////////////////////////////\n");
    printf("suma=%d\n",suma);
    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, chunk = %d\n", kind, chunk);
    printf("////////////////////////////////////\n");

    //Modificación de variables
    omp_set_dynamic(0);
    omp_set_num_threads(6);
    omp_set_schedule(1,2);

    printf("////////////////////////////////////\n");
    printf("DESPUÉS DE LA MODIFICACIÓN fuera de 'parallel for'\n");
    printf("////////////////////////////////////\n");
    printf("suma=%d\n",suma);
    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, chunk = %d\n", kind, chunk);
    printf("////////////////////////////////////\n");
```

CAPTURAS DE PANTALLA:



RESPUESTA:

Resto de ejercicios (usar en atcgrid la cola ac a no ser que se tenga que usar atcgrid4)

7. Implementar en paralelo la multiplicación de una matriz triangular inferior 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. La inicialización de los datos la debe hacer el thread 0. Añadir lo necesario para que el usuario pueda fijar la planificación de tareas usando la variable de entorno `OMP_SCHEDULE`. Mostrar en una captura de pantalla que el código resultante funciona correctamente. NOTA: usar para generar los valores aleatorios, por ejemplo, `drand48_r()`.

CAPTURA CÓDIGO FUENTE: `pmtv-OpenMP.c`

```
// REGIÓN PARALELA  (producto de la matriz por el vector)
#pragma omp parallel
{
    #pragma omp single
      inic = omp_get_wtime();

    #pragma omp for firstprivate(product) schedule(runtime)

      for(i = 0; i < N; i++){

        #ifdef DEFAULT_CHUNK
           if (i == 0){
              omp_get_schedule(&kind, &chunk);
              printf("kind: %d\n", kind);
              printf("chunk: %d\n", chunk);
           }
        #endif

        product = 0;
        for (j = i; j < N; j++){
            product += m[i][j] * v1[j];
        }

        v2[i] = product;
     }

     #pragma omp single
       fin = omp_get_wtime();

}
```

DESCOMPOSICIÓN DE DOMINIO:

CAPTURAS DE PANTALLA:

```
[ac425@atcgrid clase3]$ gcc -O2 -fopenmp pmtv-secuencial_matriz_t_inf.c -o pmtv-secuenc
ial_matriz_t_inf
[ac425@atcgrid clase3]$ srun -p ac -n1 --cpus-per-task=1 --hint=nomultithread pmtv-secu
encial_matriz_t_inf 4
Tiempo: 0.000013295      Tamaño: 4
Matriz:
        0.400000        0.000000        0.000000        0.000000
       -0.600000       -0.500000        0.000000        0.000000
       -1.600000       -1.500000       -1.400000        0.000000
       -2.600000       -2.500000       -2.400000       -2.300000

Vector:
        0.400000 0.500000 0.600000 0.700000

Vector resultado:
        0.160000 -0.250000 -0.840000 -1.610000
[ac425@atcgrid clase3]$ srun -p ac -n1 --cpus-per-task=1 --hint=nomultithread pmtv-secu
encial_matriz_t_inf 6
Tiempo: 0.000011062      Tamaño: 6
Matriz:
        0.600000        0.000000        0.000000        0.000000        0.000000
0.000000
       -0.400000       -0.300000        0.000000        0.000000        0.000000
0.000000
       -1.400000       -1.300000       -1.200000        0.000000        0.000000
0.000000
       -2.400000       -2.300000       -2.200000       -2.100000        0.000000
0.000000
       -3.400000       -3.300000       -3.200000       -3.100000       -3.000000
0.000000
       -4.400000       -4.300000       -4.200000       -4.100000       -4.000000
-3.900000

Vector:
        0.600000 0.700000 0.800000 0.900000 1.000000 1.100000

Vector resultado:
        0.360000 -0.210000 -0.960000 -1.890000 -3.000000 -4.290000
[ac425@atcgrid clase3]$
```

8. Contestar a las siguientes preguntas sobre el código del ejercicio anterior:

(a) ¿Qué número de operaciones de multiplicación y qué número de operaciones de suma realizan cada uno de los threads en la asignación `static` con `monotonic` y un `chunk` de 1?

RESPUESTA:

Dado que estamos trabajando con una matriz tirangular (en particular inferior), el número de iteraciones dependerá de la fila. La primera hará una iteración e irá aumentando en uno hasta llegar a la útima fila, que hará n iteraciones (la matriz es tiriangular inferior).

(b) 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:

Las operaciones dependerán del número de hebras y de cuantas haya disponibles. Cada hebra hará un número de operaciones distintas según haya hebras disponibles o no.

(c) ¿Qué alternativa cree que debería ofrecer mejores prestaciones? Razonar la respuesta.

RESPUESTA:

Será static, ya que evitaremos la sobrecarga en tiempo de ejecución de dynamic y guided.

9. 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 (con `monotonic` en todos los casos). Usar un tamaño de vector N múltiplo del número de cores y de 64 que esté entre 11520 y 23040. El número de threads en las ejecuciones debe coincidir con el número de núcleos del computador. 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 (representar los valores de las dos tablas). Incluir los scripts utilizado en el cuaderno de prácticas.
NOTA: Nunca ejecute en atcgrid código que imprima todos los componentes del resultado.

CAPTURA CÓDIGO FUENTE: `pmtv-OpenMP.c`

```
// REGIÓN PARALELA  (producto de la matriz por el vector)
#pragma omp parallel
{
    #pragma omp single
      inic = omp_get_wtime();

    #pragma omp for firstprivate(product) schedule(runtime)

      for(i = 0; i < N; i++){

        #ifdef DEFAULT_CHUNK
           if (i == 0){
              omp_get_schedule(&kind, &chunk);
              printf("kind: %d\n", kind);
              printf("chunk: %d\n", chunk);
           }
        #endif

        product = 0;
        for (j = i; j < N; j++){
            product += m[i][j] * v1[j];
        }

        v2[i] = product;
     }

     #pragma omp single
       fin = omp_get_wtime();

}
```

DESCOMPOSICIÓN DE DOMINIO:

CAPTURAS DE PANTALLA:

RESULTADOS, SCRIPT Y GRÁFICA atcgrid

```
#!/bin/bash

echo "STATIC MONOTONIC CHUNK=1\n"
export OMP_SCHEDULE="monotonic:static,1"
./pmtv-secuencial_matriz_t_inf 10000

echo "DYNAMIC MONOTONIC CHUNK=1\n"
export OMP_SCHEDULE="monotonic:dynamic,1"
./pmtv-secuencial_matriz_t_inf 10000

echo "GUIDED MONOTONIC CHUNK=1\n"
export OMP_SCHEDULE="monotonic:guided,1"
./pmtv-secuencial_matriz_t_inf 10000

echo "STATIC MONOTONIC CHUNK=64\n"
export OMP_SCHEDULE="monotonic:static,64"
./pmtv-secuencial_matriz_t_inf 10000

echo "DYNAMIC MONOTONIC CHUNK=64\n"
export OMP_SCHEDULE="monotonic:static,64"
./pmtv-secuencial_matriz_t_inf 10000

```

```
[ac425@atcgrid clase3]$ sbatch -p ac --hint=nomultithread eje9.sh
Submitted batch job 151449
[ac425@atcgrid clase3]$ cat slurm-151449.out
STATIC MONOTONIC CHUNK=1\n
Tiempo: 0.089696687      Tamaño: 10000   v2[0]: 1000000.000000   v2[N-1]: -15997400.090
000
DYNAMIC MONOTONIC CHUNK=1\n
Tiempo: 0.089660631      Tamaño: 10000   v2[0]: 1000000.000000   v2[N-1]: -15997400.090
000
GUIDED MONOTONIC CHUNK=1\n
Tiempo: 0.105899290      Tamaño: 10000   v2[0]: 1000000.000000   v2[N-1]: -15997400.090
000
STATIC MONOTONIC CHUNK=64\n
Tiempo: 0.089679036      Tamaño: 10000   v2[0]: 1000000.000000   v2[N-1]: -15997400.090
000
DYNAMIC MONOTONIC CHUNK=64\n
Tiempo: 0.090062152      Tamaño: 10000   v2[0]: 1000000.000000   v2[N-1]: -15997400.090
000
[ac425@atcgrid clase3]$
```

Tabla 2 . Tiempos de ejecución de la versión paralela del producto de una matriz triangular por un vector para vectores de tamaño N= (solo se ha paralelizado el producto, no la inicialización de los datos).

Chunk	Static	Dynamic	Guided
por defecto			
1	0.089696687	0.089660631	0.105899290
64	0.089679036	0.090062152	

Chunk	Static	Dynamic	Guided
por defecto			
1			
64			