

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): Daniel Diaz Pareja

Grupo de prácticas: A2

Fecha de entrega: 12/05/2016

Fecha evaluación en clase: 13/05/2016

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`

```
#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, "[ERROR]-Falta iteraciones\n, \
[ERROR]-Falta n hebras");
        exit(-1);
    }

    n = atoi(argv[1]); if (n>20) n=20;
    x = atoi(argv[2]); if (x>50) x=50;

    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:

```

dani@dani-Aspire-5750G:$ ./if-clauseModificado 10 10
thread 2 suma de a[2]=2 sumalocal=2
thread 8 suma de a[8]=8 sumalocal=8
thread 1 suma de a[1]=1 sumalocal=1
thread 5 suma de a[5]=5 sumalocal=5
thread 6 suma de a[6]=6 sumalocal=6
thread 3 suma de a[3]=3 sumalocal=3
thread 0 suma de a[0]=0 sumalocal=0
thread 4 suma de a[4]=4 sumalocal=4
thread 7 suma de a[7]=7 sumalocal=7
thread 9 suma de a[9]=9 sumalocal=9

```

```

dani@dani-Aspire-5750G:$ ./if-clauseModificado 10 2
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 0 suma de a[3]=3 sumalocal=6
thread 0 suma de a[4]=4 sumalocal=10
thread 1 suma de a[5]=5 sumalocal=5
thread 1 suma de a[6]=6 sumalocal=11
thread 1 suma de a[7]=7 sumalocal=18
thread 1 suma de a[8]=8 sumalocal=26
thread 1 suma de a[9]=9 sumalocal=35
thread master=0 imprime suma=45

```

```

dani@dani-Aspire-5750G:$ ./if-clauseModificado 10 5
thread 4 suma de a[8]=8 sumalocal=8
thread 4 suma de a[9]=9 sumalocal=17
thread 1 suma de a[2]=2 sumalocal=2
thread 1 suma de a[3]=3 sumalocal=5
thread 2 suma de a[4]=4 sumalocal=4
thread 2 suma de a[5]=5 sumalocal=9
thread 3 suma de a[6]=6 sumalocal=6
thread 3 suma de a[7]=7 sumalocal=13
thread 0 suma de a[0]=0 sumalocal=0
thread 0 suma de a[1]=1 sumalocal=1
thread master=0 imprime suma=45

```

RESPUESTA: En el primer ejemplo vemos la ejecución para un valor de $x = 10$. Como vemos, cada hebra ejecuta una iteración del bucle, ya que dicho número de iteraciones también es 10. Por lo tanto, se crean 10 hebras.

En el siguiente ejemplo vemos la ejecución para $x = 2$. Como vemos, solo se crean 2 hebras, la 0 y la 1, y entre ellas se reparten las iteraciones.

En el último ejemplo vemos la ejecución para $x = 5$. Como era de esperar, se crean hebras con identificador de 0 a 4, es decir, 5 en total, y se reparten las iteraciones entre ellas (2 para cada hebra).

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	0	0	0	1	0	0	0	0	0
1	1	0	0	0	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	0
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	0
13	1	0	1	0	0	0	0	0	0
14	0	1	1	0	0	0	0	0	0
15	1	1	1	0	0	0	0	0	0

(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	0	2	3	2	0	0
1	1	0	0	1	2	3	2	0	0
2	2	1	0	2	3	3	2	0	0
3	3	1	0	3	3	3	2	0	0
4	0	2	1	0	1	1	2	3	3
5	1	2	1	0	1	1	2	3	3
6	2	3	1	0	0	1	2	3	3
7	3	3	1	1	0	1	2	2	3
8	0	0	2	1	0	2	2	2	1
9	1	0	2	1	0	2	2	2	1
10	2	1	2	1	0	2	2	1	1
11	3	1	2	1	0	2	2	1	1
12	0	2	3	1	0	0	2	0	2
13	1	2	3	1	0	0	2	0	2
14	2	3	3	1	0	0	2	0	2
15	3	3	3	1	0	0	2	0	2

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

RESPUESTA:

Static: El reparto de hebras es Round Robin, es decir, el primer bloque de “chunk” iteraciones va a la primera hebra, el segundo a la segunda hebra, etc. Una vez se termina la primera tanda de asignaciones, se sigue con las sucesivas rondas de la misma manera.

Dynamic: Se asignan las iteraciones a las hebras conforme vayan llegando, es decir, cuanto más rápida sea una hebra, más iteraciones realizará.

Guided: Se hace de la misma manera que en Dynamic, pero ahora el tamaño mínimo de bloques a repartir entre las hebras va determinado por el parámetro “chunk”. Entonces, como mínimo a cada hebra que llegue se le asignará un número “chunk” de hebras. Es más eficiente que Dynamic ya que se reducen los repartos totales.

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`

```

#include <stdio.h>
#include <stdlib.h>
#ifdef _OPENMP
#include <omp.h>
#else
#define omp_get_thread_num() 0
#endif
main(int argc, char **argv) {
    int i, n=200, chunk, a[n], suma=0, modifier, dyn, max, limit;
    omp_sched_t kind;
    if(argc < 3) {
        fprintf(stderr, "\nFalta iteraciones o 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 for firstprivate(suma) \
    lastprivate(suma, dyn, max, limit) 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);

        dyn = omp_get_dynamic();
        max = omp_get_max_threads();
        limit = omp_get_thread_limit();
        omp_get_schedule(&kind, &modifier);
    }

    printf("Dentro de 'parallel for' dyn-var=%d\n", dyn);
    printf("Dentro de 'parallel for' nthreads-var=%d\n", max);
    printf("Dentro de 'parallel for' thread-limit-var=%d\n", limit);
    printf("Dentro de 'parallel for' run-sched-var=%d\n", kind);

    omp_get_schedule(&kind, &modifier);
    printf("Fuera de 'parallel for' suma=%d\n", suma);
    printf("Fuera de 'parallel for' dyn-var=%d\n", omp_get_dynamic());
    printf("Fuera de 'parallel for' nthreads-var=%d\n", omp_get_max_threads());
    printf("Fuera de 'parallel for' thread-limit-var=%d\n", omp_get_thread_limit());
    printf("Fuera de 'parallel for' run-sched-var=%d\n", kind);
}

```

CAPTURAS DE PANTALLA:

Modificando dyn-var:

```
dani@dani-Aspire-5750G:$ export OMP_DYNAMIC=FALSE
dani@dani-Aspire-5750G:$ ./scheduled-clauseModificado 8 1
thread 0 suma a[0]=0 suma=0
thread 0 suma a[4]=4 suma=4
thread 0 suma a[5]=5 suma=9
thread 0 suma a[6]=6 suma=15
thread 0 suma a[7]=7 suma=22
thread 3 suma a[3]=3 suma=3
thread 1 suma a[1]=1 suma=1
thread 2 suma a[2]=2 suma=2
Dentro de 'parallel for' dyn-var=0
Dentro de 'parallel for' nthreads-var=4
Dentro de 'parallel for' thread-limit-var=2147483647
Dentro de 'parallel for' run-sched-var=2
Fuera de 'parallel for' suma=22
Fuera de 'parallel for' dyn-var=0
Fuera de 'parallel for' nthreads-var=4
Fuera de 'parallel for' thread-limit-var=2147483647
Fuera de 'parallel for' run-sched-var=2
```

```
dani@dani-Aspire-5750G:$ export OMP_DYNAMIC=TRUE
dani@dani-Aspire-5750G:$ ./scheduled-clauseModificado 8 1
thread 2 suma a[0]=0 suma=0
thread 2 suma a[4]=4 suma=4
thread 2 suma a[5]=5 suma=9
thread 2 suma a[6]=6 suma=15
thread 2 suma a[7]=7 suma=22
thread 1 suma a[2]=2 suma=2
thread 0 suma a[1]=1 suma=1
thread 3 suma a[3]=3 suma=3
Dentro de 'parallel for' dyn-var=1
Dentro de 'parallel for' nthreads-var=4
Dentro de 'parallel for' thread-limit-var=2147483647
Dentro de 'parallel for' run-sched-var=2
Fuera de 'parallel for' suma=22
Fuera de 'parallel for' dyn-var=1
Fuera de 'parallel for' nthreads-var=4
Fuera de 'parallel for' thread-limit-var=2147483647
Fuera de 'parallel for' run-sched-var=2
```

Modificando nthreads-var:

```
dani@dani-Aspire-5750G:~$ export OMP_NUM_THREADS=4
dani@dani-Aspire-5750G:~$ ./scheduled-clauseModificado 8 1
thread 0 suma a[0]=0 suma=0
thread 0 suma a[4]=4 suma=4
thread 0 suma a[5]=5 suma=9
thread 0 suma a[6]=6 suma=15
thread 0 suma a[7]=7 suma=22
thread 3 suma a[2]=2 suma=2
thread 2 suma a[3]=3 suma=3
thread 1 suma a[1]=1 suma=1
Dentro de 'parallel for' dyn-var=0
Dentro de 'parallel for' nthreads-var=4
Dentro de 'parallel for' thread-limit-var=2147483647
Dentro de 'parallel for' run-sched-var=2
Fuera de 'parallel for' suma=22
Fuera de 'parallel for' dyn-var=0
Fuera de 'parallel for' nthreads-var=4
Fuera de 'parallel for' thread-limit-var=2147483647
Fuera de 'parallel for' run-sched-var=2
```

```
dani@dani-Aspire-5750G:~$ export OMP_NUM_THREADS=8
dani@dani-Aspire-5750G:~$ ./scheduled-clauseModificado 8 1
thread 2 suma a[0]=0 suma=0
thread 2 suma a[7]=7 suma=7
thread 3 suma a[1]=1 suma=1
thread 5 suma a[2]=2 suma=2
thread 4 suma a[3]=3 suma=3
thread 7 suma a[4]=4 suma=4
thread 6 suma a[5]=5 suma=5
thread 1 suma a[6]=6 suma=6
Dentro de 'parallel for' dyn-var=0
Dentro de 'parallel for' nthreads-var=8
Dentro de 'parallel for' thread-limit-var=2147483647
Dentro de 'parallel for' run-sched-var=2
Fuera de 'parallel for' suma=7
Fuera de 'parallel for' dyn-var=0
Fuera de 'parallel for' nthreads-var=8
Fuera de 'parallel for' thread-limit-var=2147483647
Fuera de 'parallel for' run-sched-var=2
```

Modificando thread-limit-var:


```
dani@dani-Aspire-5750G:$ export OMP_THREAD_LIMIT=8
dani@dani-Aspire-5750G:$ ./scheduled-clauseModificado 8 1
thread 0 suma a[3]=3 suma=3
thread 0 suma a[4]=4 suma=7
thread 0 suma a[5]=5 suma=12
thread 0 suma a[6]=6 suma=18
thread 0 suma a[7]=7 suma=25
thread 2 suma a[0]=0 suma=0
thread 1 suma a[1]=1 suma=1
thread 3 suma a[2]=2 suma=2
Dentro de 'parallel for' dyn-var=0
Dentro de 'parallel for' nthreads-var=4
Dentro de 'parallel for' thread-limit-var=8
Dentro de 'parallel for' run-sched-var=2
Fuera de 'parallel for' suma=25
Fuera de 'parallel for' dyn-var=0
Fuera de 'parallel for' nthreads-var=4
Fuera de 'parallel for' thread-limit-var=8
Fuera de 'parallel for' run-sched-var=2
```

```
dani@dani-Aspire-5750G:$ export OMP_THREAD_LIMIT=500
dani@dani-Aspire-5750G:$ ./scheduled-clauseModificado 8 1
thread 3 suma a[3]=3 suma=3
thread 3 suma a[4]=4 suma=7
thread 3 suma a[5]=5 suma=12
thread 3 suma a[6]=6 suma=18
thread 3 suma a[7]=7 suma=25
thread 2 suma a[1]=1 suma=1
thread 1 suma a[2]=2 suma=2
thread 0 suma a[0]=0 suma=0
Dentro de 'parallel for' dyn-var=0
Dentro de 'parallel for' nthreads-var=4
Dentro de 'parallel for' thread-limit-var=500
Dentro de 'parallel for' run-sched-var=2
Fuera de 'parallel for' suma=25
Fuera de 'parallel for' dyn-var=0
Fuera de 'parallel for' nthreads-var=4
Fuera de 'parallel for' thread-limit-var=500
Fuera de 'parallel for' run-sched-var=2
```

Modificando run-sched-var:


```
dani@dani-Aspire-5750G:$ export OMP_SCHEDULE="static"
dani@dani-Aspire-5750G:$ ./scheduled-clauseModificado 8 1
thread 0 suma a[0]=0 suma=0
thread 2 suma a[3]=3 suma=3
thread 2 suma a[5]=5 suma=8
thread 2 suma a[6]=6 suma=14
thread 2 suma a[7]=7 suma=21
thread 1 suma a[2]=2 suma=2
thread 0 suma a[4]=4 suma=4
thread 3 suma a[1]=1 suma=1
Dentro de 'parallel for' dyn-var=0
Dentro de 'parallel for' nthreads-var=4
Dentro de 'parallel for' thread-limit-var=500
Dentro de 'parallel for' run-sched-var=1
Fuera de 'parallel for' suma=21
Fuera de 'parallel for' dyn-var=0
Fuera de 'parallel for' nthreads-var=4
Fuera de 'parallel for' thread-limit-var=500
Fuera de 'parallel for' run-sched-var=1
```

```
dani@dani-Aspire-5750G:$ export OMP_SCHEDULE="dynamic"
dani@dani-Aspire-5750G:$ ./scheduled-clauseModificado 8 1
thread 0 suma a[0]=0 suma=0
thread 2 suma a[3]=3 suma=3
thread 2 suma a[5]=5 suma=8
thread 2 suma a[6]=6 suma=14
thread 2 suma a[7]=7 suma=21
thread 1 suma a[2]=2 suma=2
thread 0 suma a[4]=4 suma=4
thread 3 suma a[1]=1 suma=1
Dentro de 'parallel for' dyn-var=0
Dentro de 'parallel for' nthreads-var=4
Dentro de 'parallel for' thread-limit-var=500
Dentro de 'parallel for' run-sched-var=2
Fuera de 'parallel for' suma=21
Fuera de 'parallel for' dyn-var=0
Fuera de 'parallel for' nthreads-var=4
Fuera de 'parallel for' thread-limit-var=500
Fuera de 'parallel for' run-sched-var=2
```

```

dani@dani-Aspire-5750G:$ export OMP_SCHEDULE="guided"
dani@dani-Aspire-5750G:$ ./scheduled-clauseModificado 8 1
thread 0 suma a[3]=3 suma=3
thread 0 suma a[4]=4 suma=7
thread 0 suma a[5]=5 suma=12
thread 0 suma a[6]=6 suma=18
thread 0 suma a[7]=7 suma=25
thread 3 suma a[0]=0 suma=0
thread 2 suma a[1]=1 suma=1
thread 1 suma a[2]=2 suma=2
Dentro de 'parallel for' dyn-var=0
Dentro de 'parallel for' nthreads-var=4
Dentro de 'parallel for' thread-limit-var=500
Dentro de 'parallel for' run-sched-var=3
Fuera de 'parallel for' suma=25
Fuera de 'parallel for' dyn-var=0
Fuera de 'parallel for' nthreads-var=4
Fuera de 'parallel for' thread-limit-var=500
Fuera de 'parallel for' run-sched-var=3

```

RESPUESTA: Se imprimen los mismos valores dentro y fuera de la función paralela. La variable run-sched-var indica el tipo de reparto de iteraciones entre las hebras de la siguiente manera:

- 1 → static
- 2 → dynamic
- 3 → guided

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

```

#include <stdio.h>
#include <stdlib.h>
#ifdef _OPENMP
#include <omp.h>
#else
#define omp_get_thread_num() 0
#endif
main(int argc, char **argv) {
    int i, n=200, chunk, a[n], suma=0, num_threads, num_proc, in_parallel;
    if(argc < 3) {
        fprintf(stderr, "\nFalta iteraciones o 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;

```

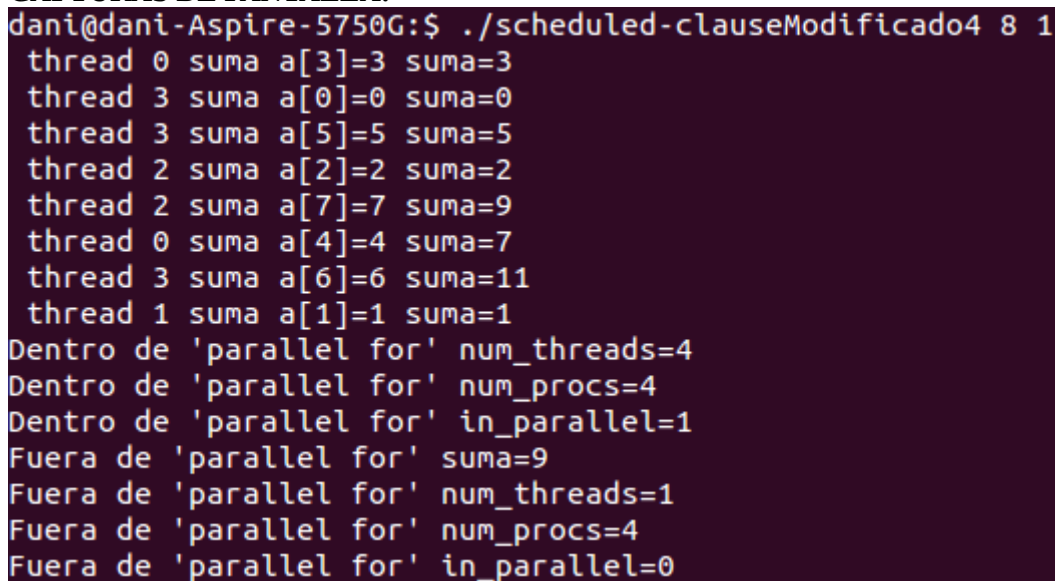
```
#pragma omp parallel for firstprivate(suma) \
lastprivate(suma,num_threads,num_proc,in_parallel)schedule(dynamic,chunk)
for (i=0; i<n; i++)
{
    suma = suma + a[i];
    printf(" thread %d suma a[%d]=%d suma=%d \n",
        omp_get_thread_num(),i,a[i],suma);

    num_threads = omp_get_num_threads();
    num_proc = omp_get_num_procs();
    in_parallel = omp_in_parallel();
}

printf("Dentro de 'parallel for' num_threads=%d\n",num_threads);
printf("Dentro de 'parallel for' num_procs=%d\n",num_proc);
printf("Dentro de 'parallel for' in_parallel=%d\n",in_parallel);

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

printf("Fuera de 'parallel for' num_threads=%d\n",omp_get_num_threads());
printf("Fuera de 'parallel for' num_procs=%d\n",omp_get_num_procs());
printf("Fuera de 'parallel for' in_parallel=%d\n",omp_in_parallel());
}
```

CAPTURAS DE PANTALLA:


```
dani@dani-Aspire-5750G:$ ./scheduled-clauseModificado4 8 1
 thread 0 suma a[3]=3 suma=3
 thread 3 suma a[0]=0 suma=0
 thread 3 suma a[5]=5 suma=5
 thread 2 suma a[2]=2 suma=2
 thread 2 suma a[7]=7 suma=9
 thread 0 suma a[4]=4 suma=7
 thread 3 suma a[6]=6 suma=11
 thread 1 suma a[1]=1 suma=1
Dentro de 'parallel for' num_threads=4
Dentro de 'parallel for' num_procs=4
Dentro de 'parallel for' in_parallel=1
Fuera de 'parallel for' suma=9
Fuera de 'parallel for' num_threads=1
Fuera de 'parallel for' num_procs=4
Fuera de 'parallel for' in_parallel=0
```

RESPUESTA: Como vemos, se obtienen valores diferentes en las funciones `omp_get_num_threads()` y `omp_in_parallel()`. Es lo lógico, ya que la primera imprime el número de hebras creadas en el momento de la ejecución (en la región `parallel` son 4 y fuera solo hay 1), y la segunda imprime 1 si se ejecuta en una región `parallel` y 0 si se hace fuera. La función `omp_get_num_procs()` devuelve el número de procesadores disponibles para el programa en el momento de su ejecución. En este caso siempre han sido 4.

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`

```
#include <stdio.h>
#include <stdlib.h>
#ifdef _OPENMP
#include <omp.h>
#else
#define omp_get_thread_num() 0
#endif
main(int argc, char **argv) {
    int i, n=200, chunk, a[n], suma=0, chunk_size, dyn, nthreads, intkind;
    omp_sched_t kind;

    omp_get_schedule(&kind, &chunk_size);
    printf("Antes de modificar las variables...\n");
    printf("dyn-var=%d\n", omp_get_dynamic());
    printf("nthreads-var=%d\n", omp_get_max_threads());
    printf("run-sched-var=%d\n", kind);

    if(argc < 6) {
        fprintf(stderr, "\nFormato: programa iteraciones chunk dyn-var\n");
        exit(-1);
    }
    n = atoi(argv[1]); if (n>200) n=200; chunk = atoi(argv[2]); dyn =
    atoi(argv[3]); nthreads = atoi(argv[4]); intkind = atoi(argv[5]);

    switch(intkind){
    case 1: kind = omp_sched_static;
    break;

    case 2: kind = omp_sched_dynamic;
    break;

    case 3: kind = omp_sched_guided;
    break;

    case 4: kind = omp_sched_auto;
    break;
    }

    omp_set_dynamic(dyn);
    omp_set_num_threads(nthreads);
    omp_set_schedule(kind, chunk);

    omp_get_schedule(&kind, &chunk_size);
    printf("\nDespués de modificar las variables...\n");
    printf("dyn-var=%d\n", omp_get_dynamic());
    printf("nthreads-var=%d\n", omp_get_max_threads());
    printf("run-sched-var=%d\n", kind);
    printf("\ndonde los valores de run-sched-var indican:\n1 → static\n2 →
    dynamic\n3 → guided\n4 → auto\n");

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

    #pragma omp parallel for firstprivate(suma) \
    lastprivate(suma)
```

```

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);
}

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

```

CAPTURAS DE PANTALLA:

```

dani@dani-Aspire-5750G:$ ./scheduled-clauseModificado5 8 1 0 3 1
Antes de modificar las variables...
dyn-var=0
nthreads-var=4
run-sched-var=2

Después de modificar las variables...
dyn-var=0
nthreads-var=3
run-sched-var=1

donde los valores de run-sched-var indican:
1 → static
2 → dynamic
3 → guided
4 → auto
thread 0 suma a[0]=0 suma=0
thread 0 suma a[1]=1 suma=1
thread 0 suma a[2]=2 suma=3
thread 1 suma a[3]=3 suma=3
thread 1 suma a[4]=4 suma=7
thread 1 suma a[5]=5 suma=12
thread 2 suma a[6]=6 suma=6
thread 2 suma a[7]=7 suma=13
Fuera de 'parallel for' suma=13

```

RESPUESTA: Como vemos, las variables se modifican correctamente utilizando las funciones indicadas para ello, y la función de las mismas se ve reflejada en el programa.

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

```

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

```

```

int main(int argc, char** argv){
    int i, j, f, c;
    double t1, t2, total;
    srand(time(NULL));

    //Leer argumento de entrada (no de componentes del vector)
    if (argc<2){
        printf("Falta tamaño de matriz y vector\n");
        exit(-1);
    }

    unsigned int N = atoi(argv[1]); // Máximo N =2^32-1=4294967295
    (sizeof(unsigned int) = 4 B)

    double *v1, *v2, **M;
    v1 = (double*) malloc(N*sizeof(double)); // malloc necesita el
tamaño en bytes
    v2 = (double*) malloc(N*sizeof(double)); //si no hay espacio
suficiente malloc devuelve NULL
    M = (double**) malloc(N*sizeof(double *));
    if ( (v1==NULL) || (v2==NULL) || (M==NULL) ){
        printf("Error en la reserva de espacio para los
vectores\n");
        exit(-2);
    }

    for (i=0; i<N; i++){
        M[i] = (double*) malloc(N*sizeof(double));
        if ( M[i]==NULL ){
            printf("Error en la reserva de espacio
para los vectores\n");
            exit(-2);
        }
    }
    //A partir de aqui se pueden acceder las componentes de la
matriz como M[i][j]

    //Inicializar matriz y vectores
    printf("Vector 1: \n\n");
    for (i=0; i<N; i++)
    {
        v1[i]=i;
        printf("%.0lf ", v1[i]);
    }

    printf("\n\n");

    printf("Matriz: \n");
    for (f=0; f<N; f++)
    {
        printf("\n");
        for (c=0; c<f; c++){
            M[f][c] = 0;
            printf("%.0lf ", M[f][c]);
        }

        for (c=f; c<N; c++)
        {
            M[f][c] = rand()%(1-10 + 1) + 1;
            printf("%.0lf ", M[f][c]);

```

```

    }
}
//Medida de tiempo
t1 = omp_get_wtime();

//Calcular producto de matriz triangular por vector v2 = M · v1
for (f=0; f<N; f++)
    for (c=f; c<N; c++)
        v2[f] += M[f][c] * v1[c];

//Medida de tiempo
t2 = omp_get_wtime();
total = t2 - t1;

//Imprimir el resultado y el tiempo de ejecución
printf("\nTiempo(seg.):%11.9f\t / Tamaño:%u\t/ V2[0]=%8.6f\n", total, N, v2[0], N-1, v2[N-1]);

if (N<15)
{
    printf("\nv2=[");
    for (i=0; i<N; i++)
        printf("%.0lf ", v2[i]);
    printf("]\n");
}

free(v1); // libera el espacio reservado para v1
free(v2); // libera el espacio reservado para v2
for (i=0; i<N; i++)
    free(M[i]);

free(M);

return 0;
}

```

CAPTURAS DE PANTALLA:

```

dani@dani-Aspire-5750G:$ g++ -O2 -fopenmp -o pmtv-secuencial pmtv-secuencial.c
dani@dani-Aspire-5750G:$ ./pmtv-secuencial 5
Vector 1:

[0 1 2 3 4 ]

Matriz:

3 4 8 6 1
0 2 6 1 8
0 0 6 5 1
0 0 0 4 2
0 0 0 0 7
Tiempo(seg.):0.000000646          / Tamaño:5          / V2[0]=42.000000 V2[4]=28.000000
v2=[42 49 31 20 28 ]

```



```

dani@dani-Aspire-5750G:$ ./pmtv-secuencial 10
Vector 1:

[0 1 2 3 4 5 6 7 8 9 ]

Matriz:

5 2 4 4 7 7 4 1 5 8
0 6 2 7 3 5 2 1 3 4
0 0 7 4 8 5 4 2 7 4
0 0 0 5 2 3 8 7 4 3
0 0 0 0 2 3 1 5 3 5
0 0 0 0 0 5 1 7 3 3
0 0 0 0 0 0 3 4 4 6
0 0 0 0 0 0 0 8 2 1
0 0 0 0 0 0 0 0 7 6
0 0 0 0 0 0 0 0 0 4
Tiempo(seg.):0.000000922          / Tamaño:10      / V2[0]=228.000000 V2[9]=36.000000
v2=[228 147 213 194 133 131 132 81 110 36 ]

```

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: A) En `static`, el valor de `chunk` por defecto es $n^{\circ}_{iteraciones}/n^{\circ}_{hebras}$.

En `dynamic`, el valor de `chunk` por defecto es 1.

En `guided`, el valor de `chunk` por defecto es $n^{\circ}_{iteraciones}/n^{\circ}_{hebras}$.

CÓDIGO FUENTE: `pmtv-OpenMP.c`

```

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

```

```

#include <time.h>
#include <omp.h>

int main(int argc, char** argv){
    int i, j, f, c, intkind, chunk;
    double t1, t2, total;
    srand(time(NULL));
    omp_sched_t kind;

    //Leer argumento de entrada (no de componentes del vector)
    if (argc<4){
        printf("Formato: programa tamaño_matriz sched_var
chunk\n");
        exit(-1);
    }

    unsigned int N = atoi(argv[1]); intkind=atoi(argv[2]);
    chunk=atoi(argv[3]);
    // Máximo N =2^32-1=4294967295 (sizeof(unsigned int) = 4 B)

    double *v1, *v2, **M;
    v1 = (double*) malloc(N*sizeof(double)); // malloc necesita el
tamaño en bytes
    v2 = (double*) malloc(N*sizeof(double)); //si no hay espacio
suficiente malloc devuelve NULL
    M = (double**) malloc(N*sizeof(double *));
    if ( (v1==NULL) || (v2==NULL) || (M==NULL) ){
        printf("Error en la reserva de espacio para los
vectores\n");
        exit(-2);
    }

    switch(intkind){
        case 1: kind = omp_sched_static;
        break;

        case 2: kind = omp_sched_dynamic;
        break;

        case 3: kind = omp_sched_guided;
        break;

        case 4: kind = omp_sched_auto;
        break;
    }

    omp_set_schedule(kind, chunk);
    for (i=0; i<N; i++){
        M[i] = (double*) malloc(N*sizeof(double));
        if ( M[i]==NULL ){
            printf("Error en la reserva de espacio
para los vectores\n");
            exit(-2);
        }
    }
    //A partir de aqui se pueden acceder las componentes de la
matriz como M[i][j]

    //Inicializar matriz y vectores
    //printf("Vector 1: \n\n");

    #pragma omp parallel for schedule(runtime)

```

```

        for (i=0; i<N; i++)
        {
            v1[i]=i;
            //printf("%.0lf ",v1[i]);

        }

        //printf("]\n\n");

        //printf("Matriz: \n");
        #pragma omp parallel for schedule(runtime)
        for (f=0; f<N; f++)
        {
            //printf("\n");
            for (c=0; c<f; c++){
                M[f][c] = 0;
                //printf("%.0lf ", M[f][c]);

            }

            for (c=f; c<N; c++)
            {
                M[f][c] = rand()%(1-10 + 1) + 1;
                //printf("%.0lf ", M[f][c]);

            }
            //n++;
        }
        //Medida de tiempo
        t1 = omp_get_wtime();

        //Calcular producto de matriz triangular por vector v2 = M · v1
        #pragma omp parallel for schedule(runtime)
        for (f=0; f<N; f++)
            for (c=f; c<N; c++)
                v2[f] += M[f][c] * v1[c];

        //Medida de tiempo
        t2 = omp_get_wtime();
        total = t2 - t1;

        //Imprimir el resultado y el tiempo de ejecución
        printf("Tiempo(seg.):%11.9f\t / Tamaño:%u\t/ V2[0]=%8.6f V2[%d]=%8.6f\n", total,N,v2[0],N-1,v2[N-1]);

        if (N<15)
        {
            printf("\nv2=[");
            for (i=0; i<N; i++)
                printf("%.0lf ",v2[i]);
            printf("]\n");
        }

        free(v1); // libera el espacio reservado para v1
        free(v2); // libera el espacio reservado para v2
        for (i=0; i<N; i++)
            free(M[i]);

        free(M);

        return 0;
    }

```

DESCOMPOSICIÓN DE DOMINIO:

Cada hebra trabaja con una parte de datos asignados. Esos datos son las filas de la matriz,

luego cada hebra conoce solamente la fila de la matriz con la que está trabajando.

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

```
dani@dani-Aspire-5750G:~$ ./pmtv-OpenMP 1000 1 0
Tiempo(seg.):0.000833538 / Tamaño:1000 / V2[0]=193655.000000 V2[999]=999.000000
dani@dani-Aspire-5750G:~$ ./pmtv-OpenMP 1000 2 0
Tiempo(seg.):0.000821809 / Tamaño:1000 / V2[0]=945639.000000 V2[999]=2997.000000
dani@dani-Aspire-5750G:~$ ./pmtv-OpenMP 1000 3 0
Tiempo(seg.):0.000905175 / Tamaño:1000 / V2[0]=1384886.000000 V2[999]=2997.000000
dani@dani-Aspire-5750G:~$ ./pmtv-OpenMP 1000 4 0
Tiempo(seg.):0.000865818 / Tamaño:1000 / V2[0]=1551154.000000 V2[999]=6993.000000
```

TABLA RESULTADOS, SCRIPT Y GRÁFICA ATCGRID

SCRIPT: pmvt-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:

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

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)**

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:
(ADJUNTAR CÓDIGO FUENTE AL .ZIP)**

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

--