

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): Juan Manuel Rubio Rodríguez

Grupo de prácticas: D1

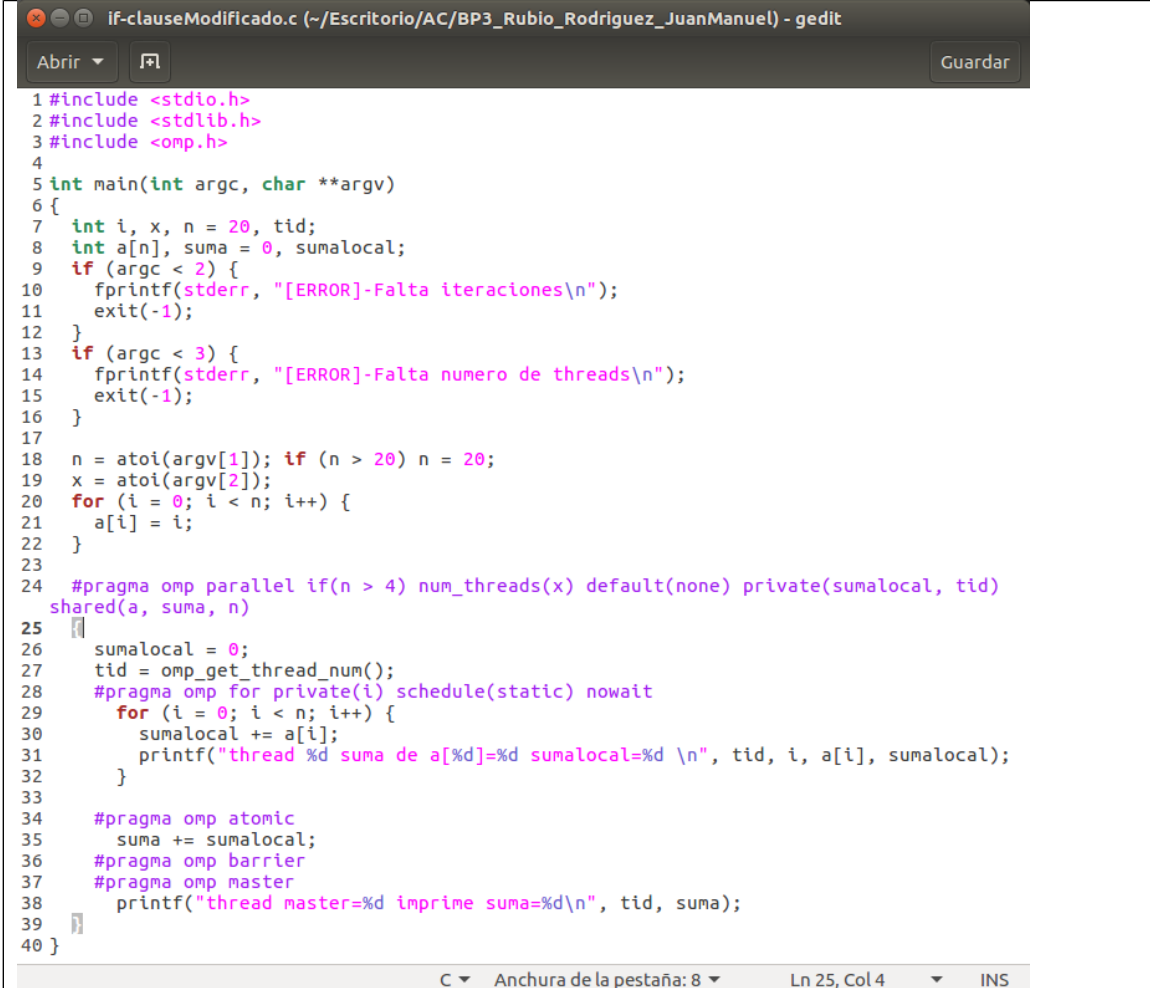
Fecha de entrega:

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



```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <omp.h>
4
5 int main(int argc, char **argv)
6 {
7     int i, x, n = 20, tid;
8     int a[n], suma = 0, sumalocal;
9     if (argc < 2) {
10         fprintf(stderr, "[ERROR]-Falta iteraciones\n");
11         exit(-1);
12     }
13     if (argc < 3) {
14         fprintf(stderr, "[ERROR]-Falta numero de threads\n");
15         exit(-1);
16     }
17
18     n = atoi(argv[1]); if (n > 20) n = 20;
19     x = atoi(argv[2]);
20     for (i = 0; i < n; i++) {
21         a[i] = i;
22     }
23
24     #pragma omp parallel if(n > 4) num_threads(x) default(none) private(sumalocal, tid)
    shared(a, suma, n)
25 {
26     sumalocal = 0;
27     tid = omp_get_thread_num();
28     #pragma omp for private(i) schedule(static) nowait
29     for (i = 0; i < n; i++) {
30         sumalocal += a[i];
31         printf("thread %d suma de a[%d]=%d sumalocal=%d \n", tid, i, a[i], sumalocal);
32     }
33
34     #pragma omp atomic
35     suma += sumalocal;
36     #pragma omp barrier
37     #pragma omp master
38     printf("thread master=%d imprime suma=%d\n", tid, suma);
39 }
40 }
```

CAPTURAS DE PANTALLA:

```

juanma@juanma-X550VX: ~/Escritorio/AC/BP3_Rubio_Rodriguez_JuanManuel
[JuanManuelRubioRodriguez juanma@juanma-X550VX:~/Escritorio/AC/BP3_Rubio_Rodriguez_JuanManuel] 2018-04-29 domingo
$gcc if-clauseModificado.c -o if-clauseModificado -fopenmp
[JuanManuelRubioRodriguez juanma@juanma-X550VX:~/Escritorio/AC/BP3_Rubio_Rodriguez_JuanManuel] 2018-04-29 domingo
$./if-clauseModificado 4 5
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 master=0 imprime suma=6
[JuanManuelRubioRodriguez juanma@juanma-X550VX:~/Escritorio/AC/BP3_Rubio_Rodriguez_JuanManuel] 2018-04-29 domingo
$./if-clauseModificado 5 5
thread 1 suma de a[1]=1 sumalocal=1
thread 4 suma de a[4]=4 sumalocal=4
thread 3 suma de a[3]=3 sumalocal=3
thread 2 suma de a[2]=2 sumalocal=2
thread 0 suma de a[0]=0 sumalocal=0
thread master=0 imprime suma=10
[JuanManuelRubioRodriguez juanma@juanma-X550VX:~/Escritorio/AC/BP3_Rubio_Rodriguez_JuanManuel] 2018-04-29 domingo
$./if-clauseModificado 5 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 1 suma de a[3]=3 sumalocal=3
thread 1 suma de a[4]=4 sumalocal=7
thread master=0 imprime suma=10
[JuanManuelRubioRodriguez juanma@juanma-X550VX:~/Escritorio/AC/BP3_Rubio_Rodriguez_JuanManuel] 2018-04-29 domingo
$

```

RESPUESTA: Como vemos en la captura de la terminal, siendo el primer argumento de entrada el número de iteraciones y el segundo el número de threads que queremos que se ejecuten mediante la cláusula `num_threads()`, si el número de iteraciones es menor o igual que 4, la cláusula `if` tiene prioridad y al ser menor o igual que 4, solo se ejecuta una hebra. Los dos casos siguientes, con un número de iteraciones superior a 4, se abre la región paralela con tantas threads como el segundo argumento que se le pasa al programa.

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

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

RESPUESTA: Con `static`, la distribución es mediante `round robin` y comenzando en la hebra cero, asignando tantas iteraciones a cada `thread` como el tamaño de la variable `chunk`.

Con `dynamic`, la asignación se hace en tiempo de ejecución y en cada asignación le

corresponden un número de iteraciones igual a la variable `chunk`. Con este tipo de distribución, cada hebra puede ejecutar un número distinto de iteraciones, asignándose más iteraciones a las hebras que acaban antes.

Con `guided`, también se asignan las iteraciones en tiempo de ejecución, pero a diferencia de `dynamic`, esta asignación se realiza mediante el cociente entre el número de iteraciones que quedan y el número de threads que se están utilizando, menguando en cada asignación pues el número de iteraciones cada vez es menor. Siempre se asigna al menos el tamaño del `chunk` hasta la última asignación que corresponde al número de iteraciones que resten.

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`

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #ifdef _OPENMP
4 #include <omp.h>
5 #else
6 #define omp_get_thread_num() 0
7 #endif
8
9 int main (int argc, char **argv){
10
11     omp_sched_t kind;
12     int i, n = 16, chunk, a[n], suma = 0, chunk_value;
13
14     if(argc < 3) {
15         fprintf(stderr, "\nFalta iteraciones y/o tchunk \n");
16         exit(-1);
17     }
18     n = atoi(argv[1]);
19
20     if (n > 200) n = 200;
21
22     chunk = atoi(argv[2]);
23
24     for (i = 0; i < n; i++)
25         a[i] = i;
26
27     #pragma omp parallel for firstprivate(suma) lastprivate(suma) schedule(dynamic,
28     chunk)
29     for (i = 0; i < n; i++){
30         suma = suma + a[i];
31
32         if (omp_get_thread_num() == 0){
33
34             omp_get_schedule(&kind, &chunk_value);
35             printf("Dentro de parallel for \n");
36
37             printf("Variable dyn-var=%d, variable nthreads-var=%d, variable thread-limit-
38             var=%d y variable run-sched-var=%d - chunk=%d \n", omp_get_dynamic(),
39             omp_get_max_threads(), omp_get_thread_limit(), kind, chunk_value);
40         }
41
42         printf("Fuera de 'parallel for' \n");
43
44         printf("Variable dyn-var=%d, variable nthreads-var=%d, variable thread-limit-var=%d
45         y variable run-sched-var=%d - chunk=%d \n", omp_get_dynamic(), omp_get_max_threads(),
46         omp_get_thread_limit(), kind, chunk_value);
47     }
48 }

```

CAPTURAS DE PANTALLA:

```

juanma@juanma-X550VX: ~/Escritorio/AC/BP3_Rubio_Rodriguez_JuanManuel
[juanManuelRubioRodriguez juanma@juanma-X550VX:~/Escritorio/AC/BP3_Rubio_Rodriguez_JuanManuel] 2018-04-29 domingo
$./scheduled-clauseModificado 8 4
Dentro de parallel for
Variable dyn-var=1, variable nthreads-var=8, variable thread-limit-var=214748364
7 y variable run-sched-var=2 - chunk=1
Dentro de parallel for
Variable dyn-var=1, variable nthreads-var=8, variable thread-limit-var=214748364
7 y variable run-sched-var=2 - chunk=1
Dentro de parallel for
Variable dyn-var=1, variable nthreads-var=8, variable thread-limit-var=214748364
7 y variable run-sched-var=2 - chunk=1
Dentro de parallel for
Variable dyn-var=1, variable nthreads-var=8, variable thread-limit-var=214748364
7 y variable run-sched-var=2 - chunk=1
Fuera de 'parallel for'
Variable dyn-var=1, variable nthreads-var=8, variable thread-limit-var=214748364
7 y variable run-sched-var=2 - chunk=1
[juanManuelRubioRodriguez juanma@juanma-X550VX:~/Escritorio/AC/BP3_Rubio_Rodriguez_JuanManuel] 2018-04-29 domingo
$export OMP_NUM_THREADS=2
[juanManuelRubioRodriguez juanma@juanma-X550VX:~/Escritorio/AC/BP3_Rubio_Rodriguez_JuanManuel] 2018-04-29 domingo
$export OMP_SCHEDULE="static,2"
[juanManuelRubioRodriguez juanma@juanma-X550VX:~/Escritorio/AC/BP3_Rubio_Rodriguez_JuanManuel] 2018-04-29 domingo
$./scheduled-clauseModificado 8 4
Dentro de parallel for
Variable dyn-var=1, variable nthreads-var=2, variable thread-limit-var=214748364
7 y variable run-sched-var=1 - chunk=2
Dentro de parallel for
Variable dyn-var=1, variable nthreads-var=2, variable thread-limit-var=214748364
7 y variable run-sched-var=1 - chunk=2
Dentro de parallel for
Variable dyn-var=1, variable nthreads-var=2, variable thread-limit-var=214748364
7 y variable run-sched-var=1 - chunk=2
Dentro de parallel for
Variable dyn-var=1, variable nthreads-var=2, variable thread-limit-var=214748364
7 y variable run-sched-var=1 - chunk=2
Fuera de 'parallel for'
Variable dyn-var=1, variable nthreads-var=2, variable thread-limit-var=214748364
7 y variable run-sched-var=1 - chunk=2
[juanManuelRubioRodriguez juanma@juanma-X550VX:~/Escritorio/AC/BP3_Rubio_Rodriguez_JuanManuel] 2018-04-29 domingo
$

juanma@juanma-X550VX: ~/Escritorio/AC/BP3_Rubio_Rodriguez_JuanManuel
[juanManuelRubioRodriguez juanma@juanma-X550VX:~/Escritorio/AC/BP3_Rubio_Rodriguez_JuanManuel] 2018-04-29 domingo
$./scheduled-clauseModificado 8 4
Dentro de parallel for
Variable dyn-var=1, variable nthreads-var=2, variable thread-limit-var=214748364
7 y variable run-sched-var=1 - chunk=2
Dentro de parallel for
Variable dyn-var=1, variable nthreads-var=2, variable thread-limit-var=214748364
7 y variable run-sched-var=1 - chunk=2
Dentro de parallel for
Variable dyn-var=1, variable nthreads-var=2, variable thread-limit-var=214748364
7 y variable run-sched-var=1 - chunk=2
Dentro de parallel for
Variable dyn-var=1, variable nthreads-var=2, variable thread-limit-var=214748364
7 y variable run-sched-var=1 - chunk=2
Fuera de 'parallel for'
Variable dyn-var=1, variable nthreads-var=2, variable thread-limit-var=214748364
7 y variable run-sched-var=1 - chunk=2
[juanManuelRubioRodriguez juanma@juanma-X550VX:~/Escritorio/AC/BP3_Rubio_Rodriguez_JuanManuel] 2018-04-29 domingo
$export OMP_DYNAMIC=FALSE
[juanManuelRubioRodriguez juanma@juanma-X550VX:~/Escritorio/AC/BP3_Rubio_Rodriguez_JuanManuel] 2018-04-29 domingo
$./scheduled-clauseModificado 8 4
Dentro de parallel for
Variable dyn-var=0, variable nthreads-var=2, variable thread-limit-var=214748364
7 y variable run-sched-var=1 - chunk=2
Dentro de parallel for
Variable dyn-var=0, variable nthreads-var=2, variable thread-limit-var=214748364
7 y variable run-sched-var=1 - chunk=2
Dentro de parallel for
Variable dyn-var=0, variable nthreads-var=2, variable thread-limit-var=214748364
7 y variable run-sched-var=1 - chunk=2
Dentro de parallel for
Variable dyn-var=0, variable nthreads-var=2, variable thread-limit-var=214748364
7 y variable run-sched-var=1 - chunk=2
Fuera de 'parallel for'
Variable dyn-var=0, variable nthreads-var=2, variable thread-limit-var=214748364
7 y variable run-sched-var=1 - chunk=2
[juanManuelRubioRodriguez juanma@juanma-X550VX:~/Escritorio/AC/BP3_Rubio_Rodriguez_JuanManuel] 2018-04-29 domingo
$

```

RESPUESTA: Como vemos en la captura de la terminal, antes de exportar las variables de entorno `OMP_SCHEDULE` y `OMP_NUM_THREADS`, la variable `run-sched-var` valía 2 y la variable `chunk` valía 1; después de exportar ambas variables de entorno su valor se modifica a `run-sched-var = 1` y `chunk = 2`. Al exportar la variable de entorno `OMP_NUM_THREADS`, cambiamos el valor de la variable `nthreads-var` de 8 a 2. Tanto dentro como fuera de la región paralela obtenemos los mismos resultados. Cuando cambiamos el valor de la variable de entorno `OMP_DYNAMIC` a `false`, la variable `dyn-var` modifica su valor de 1 a 0.

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

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #ifdef _OPENMP
4 #include <omp.h>
5 #else
6 #define omp_get_thread_num() 0
7 #endif
8
9 int main (int argc, char **argv){
10     int i, n, chunk, a[n], suma = 0;
11
12     if(argc < 3) {
13         fprintf(stderr, "\nFalta iteraciones y/o tchunk \n");
14         exit(-1);
15     }
16     n = atoi(argv[1]);
17     if (n > 200) n = 200;
18     chunk = atoi(argv[2]);
19     for (i = 0; i < n; i++)
20         a[i] = i;
21
22     #pragma omp parallel for firstprivate(suma) lastprivate(suma) schedule(dynamic,
23     chunk)
24     for (i = 0; i < n; i++){
25         suma = suma + a[i];
26
27         if (omp_get_thread_num() == 0){
28             printf("Dentro de parallel for \n");
29             printf("Número de threads=%d, Número de procesadores=%d, ¿Dentro de la region
30     paralela? =%d \n", omp_get_num_threads(), omp_get_num_procs(), omp_in_parallel());
31         }
32     }
33     printf("Fuera de 'parallel for' \n");
34     printf("Número de threads=%d, Número de procesadores=%d, ¿Dentro de la region
35     paralela? =%d \n", omp_get_num_threads(), omp_get_num_procs(), omp_in_parallel());
36 }
37
38 }
39
40
41
42
43
44 }

```


CAPTURAS DE PANTALLA:

```

juanma@juanma-X550VX: ~/Escritorio/AC/BP3_Rubio_Rodriguez_JuanManuel
[JuanManuelRubioRodriguez juanma@juanma-X550VX:~/Escritorio/AC/BP3_Rubio_Rodriguez_JuanManuel] 2018-04-29 domingo
$./scheduled-clauseModificado4 8 4
Dentro de parallel for
Número de threads=2, Número de procesadores=4, ¿Dentro de la region paralela? =1

Dentro de parallel for
Número de threads=2, Número de procesadores=4, ¿Dentro de la region paralela? =1

Dentro de parallel for
Número de threads=2, Número de procesadores=4, ¿Dentro de la region paralela? =1

Dentro de parallel for
Número de threads=2, Número de procesadores=4, ¿Dentro de la region paralela? =1

Fuera de 'parallel for'
Número de threads=1, Número de procesadores=4, ¿Dentro de la region paralela? =0

[JuanManuelRubioRodriguez juanma@juanma-X550VX:~/Escritorio/AC/BP3_Rubio_Rodriguez_JuanManuel] 2018-04-29 domingo
$

```

RESPUESTA: La función `omp_get_num_procs()` devuelve siempre el mismo valor, pues siempre están disponibles el mismo número de procesadores. Sin embargo, la función `omp_get_num_threads()` que devuelve el número de threads que se usan en una región paralela, fuera de dicha región su valor es 1, pues solo se utiliza un thread para una ejecución secuencial. Por último, la función booleana `omp_in_parallel`, devuelve 1 si está en la región paralela y 0 si está fuera.

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`

```

#include <stdio.h>
#include <stdlib.h>
#ifdef _OPENMP
#include <omp.h>
#else
#define omp_get_thread_num() 0
#endif

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

    omp_sched_t kind;
    int i, n, chunk, a[n], suma = 0, chunk_value;

    if(argc < 3) {
        fprintf(stderr, "\nFalta iteraciones y/o tchunk \n");
        exit(-1);
    }
    n = atoi(argv[1]);

    if (n > 200) n = 200;

```

```

chunk = atoi(argv[2]);

printf("Antes de modificar: \n");

printf("Variable dyn-var=%d, variable nthreads-var=%d, variable thread-
limit-var=%d y variable run-sched-var=%d - chunk=%d \n", omp_get_dynamic(),
omp_get_max_threads(), omp_get_thread_limit(), kind, chunk_value);

//Modificamos los valores pertinentes

omp_set_dynamic(2);
omp_set_num_threads(16);
omp_set_schedule(2, 1);

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

#pragma omp parallel for firstprivate(suma) lastprivate(suma)
schedule(dynamic, chunk)

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

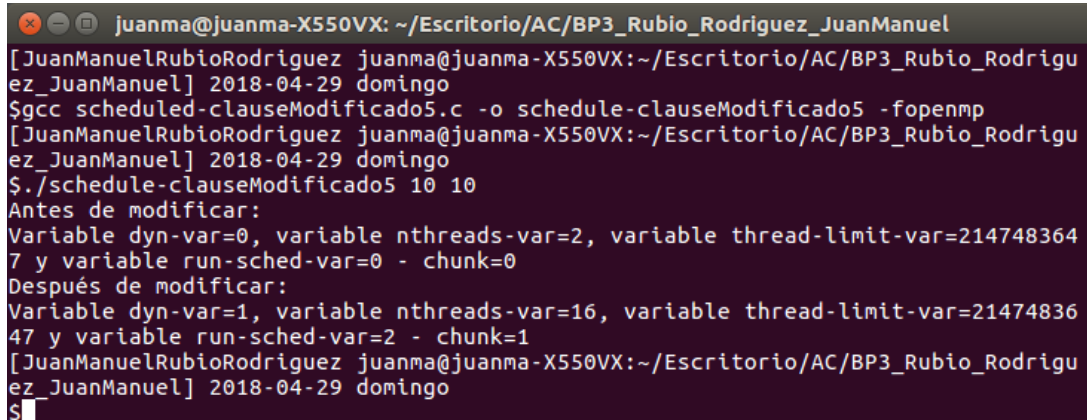
    if (omp_get_thread_num() == 0){

        omp_get_schedule(&kind, &chunk_value);
    }
}

printf("Después de modificar: \n");

printf("Variable dyn-var=%d, variable nthreads-var=%d, variable thread-
limit-var=%d y variable run-sched-var=%d - chunk=%d \n", omp_get_dynamic(),
omp_get_max_threads(), omp_get_thread_limit(), kind, chunk_value);
}

```

CAPTURAS DE PANTALLA:


```

juanma@juanma-X550VX: ~/Escritorio/AC/BP3_Rubio_Rodriguez_JuanManuel
[juanma@juanma-X550VX:~/Escritorio/AC/BP3_Rubio_Rodriguez_JuanManuel] 2018-04-29 domingo
$gcc scheduled-clauseModificado5.c -o schedule-clauseModificado5 -fopenmp
[juanma@juanma-X550VX:~/Escritorio/AC/BP3_Rubio_Rodriguez_JuanManuel] 2018-04-29 domingo
$./schedule-clauseModificado5 10 10
Antes de modificar:
Variable dyn-var=0, variable nthreads-var=2, variable thread-limit-var=2147483647 y variable run-sched-var=0 - chunk=0
Después de modificar:
Variable dyn-var=1, variable nthreads-var=16, variable thread-limit-var=2147483647 y variable run-sched-var=2 - chunk=1
[juanma@juanma-X550VX:~/Escritorio/AC/BP3_Rubio_Rodriguez_JuanManuel] 2018-04-29 domingo
$

```

RESPUESTA: Vemos cómo se modifican las variables `dyn-var`, `nthreads-var` y `run-sched-var` utilizando las funciones `omp_set_dynamic`, `omp_set_num_threads` y `omp_set_schedule` por ese orden. El orden es n^2 , pues multiplicamos $n*(n-i)$ siendo i el número de filas.

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.

CAPTURA CÓDIGO FUENTE: pmtv-secuencial.c

```
#include <stdlib.h>
#include <stdio.h>
#ifdef _OPENMP
    #include <omp.h>
#else
    #define omp_get_thread_num() 0
    #define omp_get_num_threads() 1
#endif

int main(int argc, char** argv)
{
    int i, j;
    double t1, t2, total;

    if (argc<2){
        printf("Falta tamaño de matriz y vector\n");
        exit(-1);
    }

    unsigned int N = atoi(argv[1]);

    //Reservo memoria para los vectores y la matriz

    double *v1, *v3, **M;
    v1 = (double*) malloc(N*sizeof(double));
    v3 = (double*) malloc(N*sizeof(double));
    M = (double**) malloc(N*sizeof(double *));

    for (i=0; i<N; i++){
        M[i] = (double*) malloc(N*sizeof(double));
    }

    //Inicializo matriz y vectores
    for (i=0; i<N; i++)
    {
        v1[i] = i;
        v3[i] = 0;
        for(j=0; j<N; j++){
            if (j < i)
                M[i][j] = 0;
            else
                M[i][j] = i+1;
        }
    }

    //Medida de tiempo
    t1 = omp_get_wtime();

    //Calculo el producto de la matriz por el vector v1
    for (i=0; i<N;i++)
```

```

        for(j=i;j<N;j++)
            v3[i] += M[i][j] * v1[j];

//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/ v3[0]=%8.6f v3[%d]=%8.6f\n", total,N,v3[0],N-1,v3[N-1]);

if (N<5)
    for (i=0; i<N;i++)
        printf(" v3[%d]=%5.2f\n", i, v3[i]);

//Libero memoria

free(v1);
free(v3);
for (i=0; i<N; i++)
    free(M[i]);
free(M);

return 0;
}

```

CAPTURAS DE PANTALLA:

```

juanma@juanma-X550VX: ~/Escritorio/AC/BP3_Rubio_Rodriguez_JuanManuel
[juanma@juanma-X550VX:~/Escritorio/AC/BP3_Rubio_Rodriguez_JuanManuel] 2018-04-29 domingo
$gcc pmtv-secuencial.c -o pmtv-secuencial -fopenmp
[juanma@juanma-X550VX:~/Escritorio/AC/BP3_Rubio_Rodriguez_JuanManuel] 2018-04-29 domingo
$./pmtv-secuencial 8

1.000000 1.000000 1.000000 1.000000 1.000000 1.000000 1.000000 1.000000
0.000000 2.000000 2.000000 2.000000 2.000000 2.000000 2.000000 2.000000
0.000000 0.000000 3.000000 3.000000 3.000000 3.000000 3.000000 3.000000
0.000000 0.000000 0.000000 4.000000 4.000000 4.000000 4.000000 4.000000
0.000000 0.000000 0.000000 0.000000 5.000000 5.000000 5.000000 5.000000
0.000000 0.000000 0.000000 0.000000 0.000000 6.000000 6.000000 6.000000
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 7.000000 7.000000
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 8.000000
Tiempo(seg.):0.000002458 / Tamaño:8 / v3[0]=28.000000 v3[7]=56.000000
[juanma@juanma-X550VX:~/Escritorio/AC/BP3_Rubio_Rodriguez_JuanManuel] 2018-04-29 domingo
$

```

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) He ejecutado un programa sencillo de prueba donde muestro el valor del chunk para asignaciones por defecto. El valor para static es cero, y el valor para dynamic y guided es uno. Se adjunta el ejecutable “prueba”. Asumimos que para el static el valor real de chunk es 1 pues se asignan por defecto en round robin de forma contigua entre threads.
- b) $(n/\text{chunk}) \cdot (n-i)$, siendo i el número de filas y chunk el tamaño del chunk.
- c) Equilibra la carga de trabajo de cada thread.

CAPTURA CÓDIGO FUENTE: pmtv-OpenMP.c

```
#include <stdlib.h>
#include <stdio.h>
#ifdef _OPENMP
    #include <omp.h>
#else
    #define omp_get_thread_num() 0
    #define omp_get_num_threads() 1
#endif

int main(int argc, char** argv)
{
    int i, j;
    double t1, t2, total;

    //Asigno el numero de procesadores al número máximo de threads
    int threads = omp_get_num_procs();
    omp_set_num_threads(threads);

    if (argc<2){
        printf("Falta tamaño de matriz y vector\n");
        exit(-1);
    }

    unsigned int N = atoi(argv[1]);

    //Reservo memoria para los vectores y la matriz

    double *v1, *v3, **M;
    v1 = (double*) malloc(N*sizeof(double));
    v3 = (double*) malloc(N*sizeof(double));
    M = (double**) malloc(N*sizeof(double *));

    for (i=0; i<N; i++){
        M[i] = (double*) malloc(N*sizeof(double));
    }

    //Inicializo matriz y vectores
    for (i=0; i<N; i++)
    {
```

```

        v1[i] = i;
        v3[i] = 0;
        for(j=0; j<N; j++){
            if (j < i)
                M[i][j] = 0;
            else
                M[i][j] = i+1;
        }
    }

    //Muestro la matriz

    if (N<5){
        for (i=0; i<N; i++){
            printf("\n");
            for (j=0; j<N; j++)
                printf("%f ",M[i][j]);
        }
    }

    //Medida de tiempo
    t1 = omp_get_wtime();

    //Calculo el producto de la matriz por el vector v1
    #pragma omp parallel for private(j) schedule(runtime)
    for (i=0; i<N;i++)
        for(j=i;j<N;j++)
            v3[i] += M[i][j] * v1[j];

    //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/ v3[0]=%8.6f v3[%d]=%8.6f\n", total,N,v3[0],N-1,v3[N-1]);

    if (N<5)
        for (i=0; i<N;i++)
            printf(" v3[%d]=%5.2f\n", i, v3[i]);

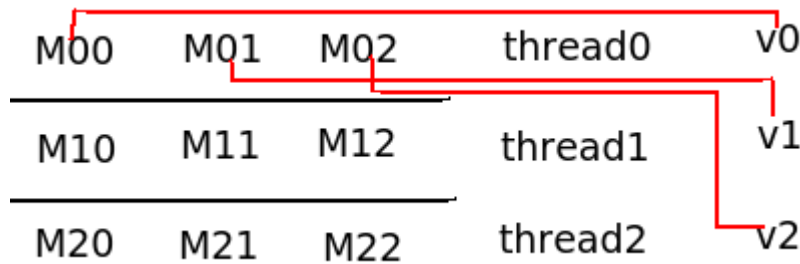
    //Libero memoria

    free(v1);
    free(v3);
    for (i=0; i<N; i++)
        free(M[i]);
    free(M);

    return 0;
}

```

DESCOMPOSICIÓN DE DOMINIO: Al paralelizar por filas y asumiendo un chunk=1, a cada thread se le asigna una fila y este realiza los cálculos con el vector por el que se multiplica la matriz, obteniendo como resultado una componente del vector resultado.



CAPTURAS DE PANTALLA:

TABLA RESULTADOS, SCRIPT Y GRÁFICA atcgrid

SCRIPT: pmvt-OpenMP_PCaula.sh

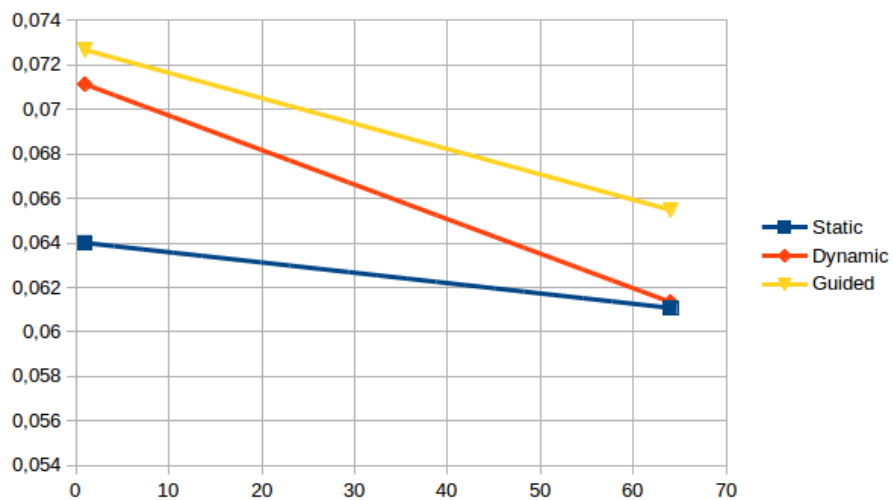
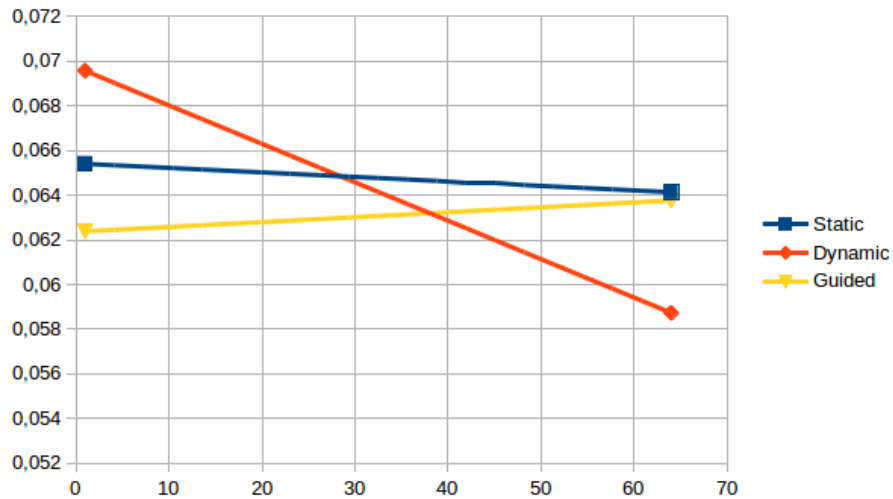
```

1 #! /bin/bash
2
3 export OMP_SCHEDULE="static"
4 echo "static y chunk por defecto"
5 ./pmtv-OpenMP 15360
6
7 export OMP_SCHEDULE="static, 1"
8 echo "static y chunk = 1"
9 ./pmtv-OpenMP 15360
10
11 export OMP_SCHEDULE="static, 64"
12 echo "static y chunk = 64"
13 ./pmtv-OpenMP 15360
14
15 export OMP_SCHEDULE="dynamic"
16 echo "dynamic y chunk por defecto"
17 ./pmtv-OpenMP 15360
18
19 export OMP_SCHEDULE="dynamic, 1"
20 echo "dynamic y chunk = 1"
21 ./pmtv-OpenMP 15360
22
23 export OMP_SCHEDULE="dynamic, 64"
24 echo "dynamic y chunk = 64"
25 ./pmtv-OpenMP 15360
26
27 export OMP_SCHEDULE="guided"
28 echo "guided y chunk por defecto"
29 ./pmtv-OpenMP 15360
30
31 export OMP_SCHEDULE="guided, 1"
32 echo "guided y chunk = 1"
33 ./pmtv-OpenMP 15360
34
35 export OMP_SCHEDULE="guided, 64"
36 echo "guided y chunk = 64"
37 ./pmtv-OpenMP 15360
  
```

Tabla 3 . Tiempos de ejecución de la versión paralela del producto de una matriz triangular por un vector r para vectores de tamaño $N=15360$, 12 threads

| Chunk | Static | Dynamic | Guided |
|-------------|-------------|-------------|-------------|
| por defecto | 0.067618551 | 0.069912982 | 0.062515772 |
| 1 | 0.065404170 | 0.069567504 | 0.062382641 |
| 64 | 0.064134723 | 0.058731472 | 0.063765687 |

| Chunk | Static | Dynamic | Guided |
|-------------|-------------|-------------|-------------|
| por defecto | 0.069019320 | 0.070920341 | 0.077299426 |
| 1 | 0.064004838 | 0.071126489 | 0.072670134 |
| 64 | 0.061080569 | 0.061340063 | 0.065481341 |



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.

CAPTURA CÓDIGO FUENTE: pmm-secuencial.c

```
#include <stdlib.h>
#include <stdio.h>
#ifdef _OPENMP
    #include <omp.h>
#else
    #define omp_get_thread_num() 0
    #define omp_get_num_threads() 1
#endif

int main(int argc, char** argv)
{
    int i, j, k;
    double t1, t2, total;

    if (argc<2){
        printf("Falta tamaño de matriz y vector\n");
        exit(-1);
    }

    unsigned int N = atoi(argv[1]);

    //Reservo memoria para los vectores y la matriz

    double **M1, **M2, **M3;
    M1 = (double**) malloc(N*sizeof(double *));
    M2 = (double**) malloc(N*sizeof(double *));
    M3 = (double**) malloc(N*sizeof(double *));

    for (i=0; i<N; i++){
        M1[i] = (double*) malloc(N*sizeof(double));
        M2[i] = (double*) malloc(N*sizeof(double));
        M3[i] = (double*) malloc(N*sizeof(double));
    }

    //Inicializo matriz y vectores
    for (i=0; i<N; i++)
        for(j=0; j<N; j++){
            M1[i][j] = i;
            M2[i][j] = i;
            M3[i][j] = 0;
        }

    //Medida de tiempo
    t1 = omp_get_wtime();

    //Calculo el producto de M1*M2 en M3
    for (i=0; i<N; i++)
        for (j=0; j<N; j++)
            for (k=0; k<N; k++)
```

```

M3[i][j] += M1[i][k] * M2[k][j];

//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/ M3[0][0]=%8.6f M3[%d]
[%d]=%8.6f\n", total,N,M3[0][0],N-1,N-1,M3[N-1][N-1]);

if (N<5)
    for (i=0; i<N; i++)
        for (j=0; j<N; j++)
            printf(" M3[%d][%d]=%5.2f\n", i, j,
M3[i][j]);

//Libero memoria

for (i=0; i<N; i++){
    free(M1[i]);
    free(M2[i]);
    free(M3[i]);
}

free(M1);
free(M2);
free(M3);

return 0;
}

```

CAPTURAS DE PANTALLA:

```

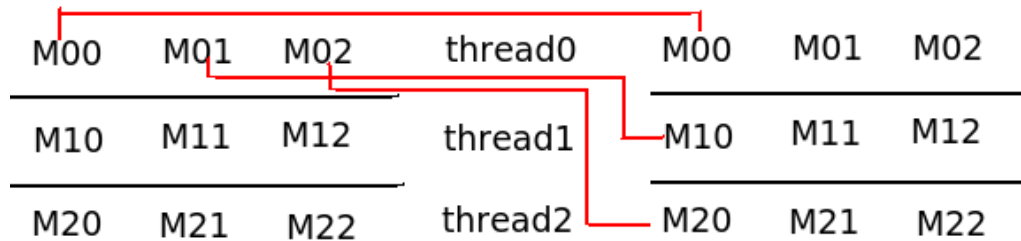
juanma@juanma-X550VX: ~
[juanma@juanma-X550VX:~/Escritorio/AC/BP3_Rubio_Rodriguez_JuanManuel] 2018-05-01 martes
$gcc pmm-secuencial.c -o pmm-secuencial -O2 -fopenmp
[juanma@juanma-X550VX:~/Escritorio/AC/BP3_Rubio_Rodriguez_JuanManuel] 2018-05-01 martes
$./pmm-secuencial 3

Tiempo(seg.):0.000001067          / Tamaño:3          / v3[0]=0.000000 v3[2]=6.000000
M3[0][0]= 0.00
M3[0][1]= 0.00
M3[0][2]= 0.00
M3[1][0]= 3.00
M3[1][1]= 3.00
M3[1][2]= 3.00
M3[2][0]= 6.00
M3[2][1]= 6.00
M3[2][2]= 6.00
[juanma@juanma-X550VX:~/Escritorio/AC/BP3_Rubio_Rodriguez_JuanManuel] 2018-05-01 martes
$

```

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).

DESCOMPOSICIÓN DE DOMINIO: A cada thread se le asignan tantas filas como el tamaño del chunk, en este ejemplo asumimos que su valor es 1, y por tanto a cada fila se le asigna un thread. Cada fila (asignada a un thread) de la primera matriz M1 realiza los cálculos con todas las columnas de la matriz M2. La suma de los productos de la primera fila de M1 por la primera columna de M2, obtiene el resultado de la primera componente de la matriz M3.



CAPTURA CÓDIGO FUENTE: pmm-OpenMP.c

```
#include <stdlib.h>
#include <stdio.h>
#ifdef _OPENMP
#include <omp.h>
#else
#define omp_get_thread_num() 0
#define omp_get_num_threads() 1
#endif

int main(int argc, char** argv)
{
    int i, j, k;
    double t1, t2, total;

    if (argc<2){
        printf("Falta tamaño de matriz y vector\n");
        exit(-1);
    }

    unsigned int N = atoi(argv[1]);

    //Reservo memoria para los vectores y la matriz

    double **M1, **M2, **M3;
    M1 = (double**) malloc(N*sizeof(double *));
    M2 = (double**) malloc(N*sizeof(double *));
    M3 = (double**) malloc(N*sizeof(double *));

    for (i=0; i<N; i++){
        M1[i] = (double*) malloc(N*sizeof(double));
        M2[i] = (double*) malloc(N*sizeof(double));
        M3[i] = (double*) malloc(N*sizeof(double));
    }

    //Inicializo matriz y vectores

    #pragma omp parallel for private (j)

    for (i=0; i<N; i++)
        for(j=0; j<N; j++){
            M1[i][j] = i;
```

```

        M2[i][j] = i;
        M3[i][j] = 0;
    }

    //Medida de tiempo
    t1 = omp_get_wtime();

    //Calculo el producto de M1*M2 en M3

    #pragma omp parallel for private (j, k)

    for (i=0; i<N;i++)
        for (j=0;j<N; j++)
            for (k=0; k<N; k++)
                M3[i][j] += M1[i][k] * M2[k][j];

    //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/ M3[0][0]=%8.6f M3[%d]
[%d]=%8.6f\n", total,N,M3[0][0],N-1,N-1,M3[N-1][N-1]);

    if (N<5)
        for (i=0; i<N; i++)
            for (j=0; j<N; j++)
                printf(" M3[%d][%d]=%5.2f\n", i, j,
M3[i][j]);

    //Libero memoria

    for (i=0; i<N; i++){
        free(M1[i]);
        free(M2[i]);
        free(M3[i]);
    }

    free(M1);
    free(M2);
    free(M3);

    return 0;
}

```

CAPTURAS DE PANTALLA:

```

juanma@juanma-X550VX: ~
[JuanManuelRubioRodriguez juanma@juanma-X550VX:~/Escritorio/AC/BP3_Rubio_Rodriguez_JuanManuel] 2018-05-01 martes
$ ./pmm-OpenMP 4

Tiempo(seg.):0.000014953      / Tamaño:4      / v3[0]=0.000000 v3[3]=18.000000
M3[0][0]= 0.00
M3[0][1]= 0.00
M3[0][2]= 0.00
M3[0][3]= 0.00
M3[1][0]= 6.00
M3[1][1]= 6.00
M3[1][2]= 6.00
M3[1][3]= 6.00
M3[2][0]=12.00
M3[2][1]=12.00
M3[2][2]=12.00
M3[2][3]=12.00
M3[3][0]=18.00
M3[3][1]=18.00
M3[3][2]=18.00
M3[3][3]=18.00
[JuanManuelRubioRodriguez juanma@juanma-X550VX:~/Escritorio/AC/BP3_Rubio_Rodriguez_JuanManuel] 2018-05-01 martes
$

```

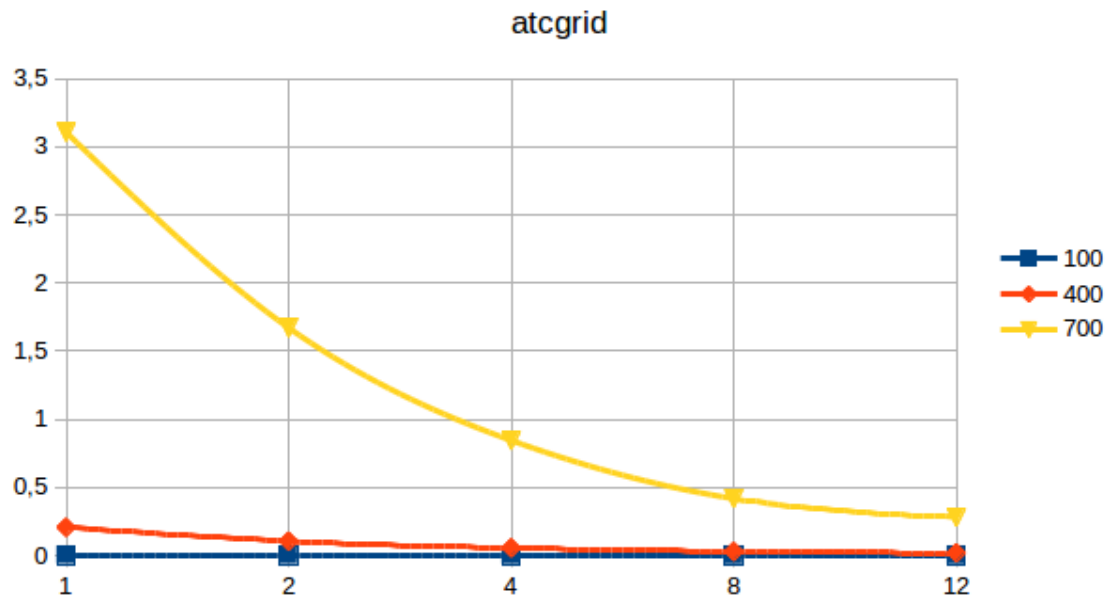
10. Hacer un estudio de escalabilidad (ganancia en velocidad en función del número de cores) en atcgrid y en su PC del código paralelo implementado para dos tamaños de las matrices. Debe recordar usar `-O2` al compilar. El número de núcleos máximo en este estudio debe ser el igual al de núcleos físicos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

Se ha elegido un tamaño máximo de 700 porque con valores superiores rebasaba el límite de tiempo de computación en atcgrid y obtenía un error.

Como se ve claramente en la gráfica, la mejora de paralelizar el cálculo se obtiene en función del tamaño del problema, lo que nos permite conseguir tiempos similares para el cálculo de matrices de tamaño muy superior al inicial al aumentar el número de núcleos disponible.

| atcgrid n.º threads | 100 | 400 | 700 |
|------------------------|-------------|-------------|-------------|
| 1 | 0,002490883 | 0,206696858 | 3,107053562 |
| 2 | 0,001467058 | 0,105336561 | 1,671033627 |
| 4 | 0,0007809 | 0,05601222 | 0,844540887 |
| 8 | 0,00039061 | 0,029550401 | 0,417891079 |
| 12 | 0,000281333 | 0,018984698 | 0,279857716 |



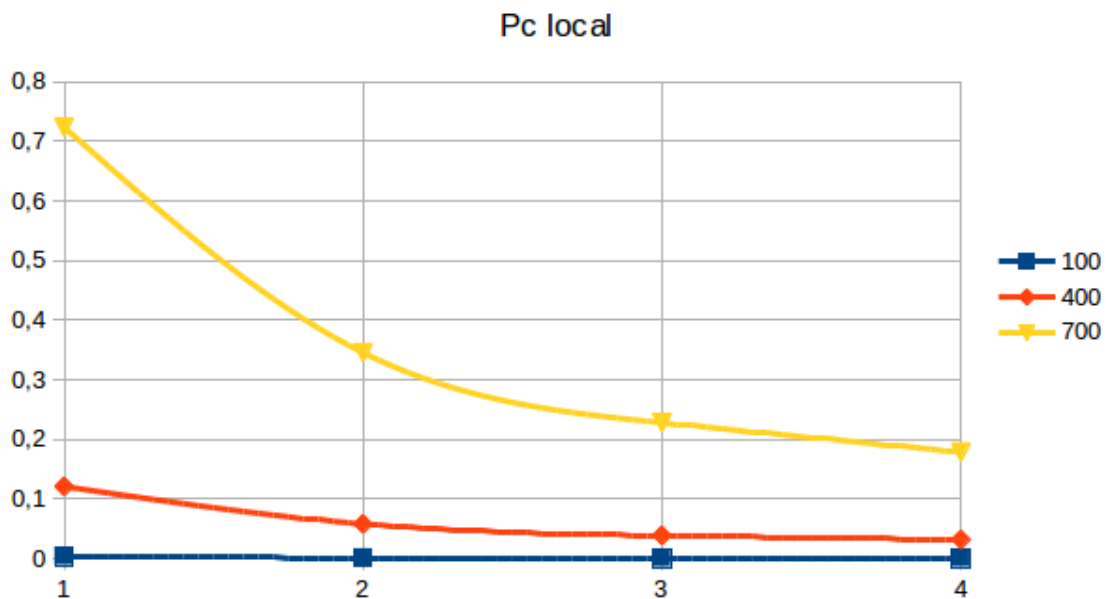
SCRIPT: pmm-OpenMP_atcgrid.sh

```

pmm-OpenMP-atcgrid.sh (~/Escritorio/AC/BP3_Ru...
Abrir  Guardar
pmm-OpenMP-atcgrid.sh x pmm-OpenMP-pcllocal.sh x
1 #!/bin/bash
2
3
4
5 for((M = 1; M < 13; M=M*2))
6 do
7     export OMP_NUM_THREADS=$M
8     echo "Numero de threads = $M"
9     for((N = 100; N <= 700; N=N+300))
10    do
11        ./pmm-OpenMP $N
12    done
13 done
14
15 export OMP_NUM_THREADS=12
16 echo "Numero de threads = 12"
17 for((N = 100; N <= 700; N=N+300))
18 do
19     ./pmm-OpenMP $N
20 done
Anchura de la pestaña: 8  Ln 17, Col 38  INS
    
```


ESTUDIO DE ESCALABILIDAD EN PCLOCAL:

| n.º threads | 100 | 400 | 700 |
|-------------|-------------|-------------|-------------|
| 1 | 0,003753101 | 0,120822387 | 0,723684664 |
| 2 | 0,000964045 | 0,057888133 | 0,345251758 |
| 3 | 0,000661676 | 0,038613948 | 0,227788244 |
| 4 | 0,000479069 | 0,031843036 | 0,178777537 |



SCRIPT: pmm-OpenMP_pclocal.sh

```

pmm-OpenMP-pclocal.sh (~/Escritorio/AC/BP3_Rubio_Rodriguez_)
Abrir  [Icon]  Guardar

pmm-OpenMP-atcgrid.sh x pmm-OpenMP-pclocal.sh x
1 #!/bin/bash
2
3
4
5 for((M = 1; M < 5; M=M+1))
6 do
7     export OMP_NUM_THREADS=$M
8     echo "Número de threads = $M"
9     for((N = 100; N <= 700; N=N+300))
10    do
11        ./pmm-OpenMP $N >> pmm-OpenMP-pclocal
12    done
13 done
sh  Anchura de la pestaña: 8  Ln 11, Col 47  INS
    
```