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): Carlos Sánchez Páez
Grupo de prácticas: A2
Fecha de entrega: 12/05/2018
Fecha evaluación en clase: 15/05/2018

5 c3

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;
    if (argc < 3) {
        fprintf(stderr, "Falta iteraciones\n");
        exit(-1);
    }
    int x=atoi(argv[2]);
    n = atoi(argv[1]);
    if (n > 20)
        n = 20;
    for (i = 0; i < n; i++)
        a[i] = i;

    #pragma omp parallel if(n>4) default(none)\
    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:

```
[CarlosSánchezPáez csp98@csp98-Sobremesa-Linux:~/Desktop/AC/Prácticas/practica3/
src] 2018-05-06 domingo
$gcc -o if-clauseModificado if-clauseModificado.c -fopenmp
[CarlosSánchezPáez csp98@csp98-Sobremesa-Linux:~/Desktop/AC/Prácticas/practica3/
src] 2018-05-06 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
[CarlosSánchezPáez csp98@csp98-Sobremesa-Linux:~/Desktop/AC/Prácticas/practica3/
src] 2018-05-06 domingo
$./if-clauseModificado 4 8
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
[CarlosSánchezPáez csp98@csp98-Sobremesa-Linux:~/Desktop/AC/Prácticas/practica3/
src] 2018-05-06 domingo
$./if-clauseModificado 6 4
Thread 0 suma de a[0]=0 sumalocal=0
Thread 0 suma de a[1]=1 sumalocal=1
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 3 suma de a[5]=5 sumalocal=5
Thread master 0 imprime suma=15
[CarlosSánchezPáez csp98@csp98-Sobremesa-Linux:~/Desktop/AC/Prácticas/practica3/
src] 2018-05-06 domingo
$
```

RESPUESTA:

Como vemos, esta cláusula permite fijar el número de hebras desde dentro del programa. En combinación con la condicional, el programa se ejecutará con el número de hebras indicado si y solo si el tamaño del vector es mayor que 4.

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

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

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

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

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

RESPUESTA:

- *static:* el número de iteraciones se asigna en tiempo de compilación. Las iteraciones se asignan consecutivamente en rodajas de *chunk* unidades (Round Robin).
- *dynamic*: el número de iteraciones se asigna en tiempo de ejecución. Cada hebra realiza como mínimo *chunk* iteraciones. Conforme una hebra termina sus iteraciones, se le asignan más, por lo que las más rápidas trabajarán más.
- *guided*: se comienza con una rodaja grande, que se va reduciendo hasta alcanzar como mínimo *chunk* iteraciones.

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`

```
#pragma omp parallel for firstprivate(suma) lastprivate(suma) schedule(dynamic,chunk)
for (i = 0; i < n; i++) {
    if (i == 0) {   //Master
        omp_get_schedule(&schedule, &chunk_);
        printf("\nDentro de parallel.\ndyn_var=%d \t nthreads_var=%d \t thread_limit_var=%d \t run_sched_var=%d \t chunk=%d\n",
               omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit(), schedule, chunk_);
    }
    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);
omp_get_schedule(&schedule, &chunk_);
printf("Fuera de parallel.\n dyn_var=%d \t nthreads_var=%d \t thread_limit_var=%d \t run_sched_var=%d \t chunk=%d\n",
      omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit(), schedule, chunk_);
```

CAPTURAS DE PANTALLA:

```
[CarlosSánchezPáez csp98@csp98-Sobremesa-Linux:~/Desktop/AC/Prácticas/practica3/src] 2018-05-06 domingo
$gcc -o scheduled-clauseModificado scheduled-clauseModificado.c -fopenmp
[CarlosSánchezPáez csp98@csp98-Sobremesa-Linux:~/Desktop/AC/Prácticas/practica3/src] 2018-05-06 domingo
$./scheduled-clauseModificado 16 1

Dentro de parallel.
dyn_var=0        nthreads_var=2          thread_limit_var=2147483647     run_sched_var=2         chunk=1
 thread 0 suma a[0]=0 suma=0
 thread 0 suma a[2]=2 suma=2
 thread 0 suma a[3]=3 suma=5
 thread 0 suma a[4]=4 suma=9
 thread 0 suma a[5]=5 suma=14
 thread 0 suma a[6]=6 suma=20
 thread 0 suma a[7]=7 suma=27
 thread 0 suma a[8]=8 suma=35
 thread 0 suma a[9]=9 suma=44
 thread 0 suma a[10]=10 suma=54
 thread 0 suma a[11]=11 suma=65
 thread 0 suma a[12]=12 suma=77
 thread 0 suma a[13]=13 suma=90
 thread 0 suma a[14]=14 suma=104
 thread 0 suma a[15]=15 suma=119
 thread 1 suma a[1]=1 suma=1
Fuera de 'parallel for' suma=119
Fuera de parallel.
 dyn_var=0       nthreads_var=2          thread_limit_var=2147483647     run_sched_var=2         chunk=1
```

```
[CarlosSánchezPáez csp98@csp98-Sobremesa-Linux:~/Desktop/AC/Prácticas/practica3/src] 2018-05-06 domingo
$./scheduled-clauseModificado 12 3

Dentro de parallel.
dyn_var=0       nthreads_var=2          thread_limit_var=2147483647     run_sched_var=2         chunk=1
 thread 0 suma a[0]=0 suma=0
 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 1 suma a[6]=6 suma=18
 thread 1 suma a[7]=7 suma=25
 thread 1 suma a[8]=8 suma=33
 thread 1 suma a[9]=9 suma=42
 thread 1 suma a[10]=10 suma=52
 thread 1 suma a[11]=11 suma=63
 thread 0 suma a[1]=1 suma=1
 thread 0 suma a[2]=2 suma=3
Fuera de 'parallel for' suma=63
Fuera de parallel.
 dyn_var=0       nthreads_var=2          thread_limit_var=2147483647     run_sched_var=2         chunk=1
[CarlosSánchezPáez csp98@csp98-Sobremesa-Linux:~/Desktop/AC/Prácticas/practica3/src] 2018-05-06 domingo
$
```

RESPUESTA:

Vemos que los valores son iguales.

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`

```
#pragma omp parallel for firstprivate(suma) lastprivate(suma) schedule(dynamic,chunk)
for (i = 0; i < n; i++) {
    if (i == 0) {   //Master
        omp_get_schedule(&schedule, &chunk_);
        printf("\nDentro de parallel.\ndyn_var=%d \t nthreads_var=%d \t thread_limit_var=%d \t run_sched_var=%d \t chunk=%d\n",
               omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit(), schedule, chunk_);
        printf("omp_get_num_threads()=%d \t omp_get_num_procs()=%d \t omp_in_parallel()=%d\n",
               omp_get_num_threads(), omp_get_num_procs(), omp_in_parallel());
    }
    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);
omp_get_schedule(&schedule, &chunk_);
printf("Fuera de parallel.\n dyn_var=%d \t nthreads_var=%d \t thread_limit_var=%d \t run_sched_var=%d \t chunk=%d\n",
       omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit(), schedule, chunk_);
printf("omp_get_num_threads()=%d \t omp_get_num_procs()=%d \t omp_in_parallel()=%d\n",
       omp_get_num_threads(), omp_get_num_procs(), omp_in_parallel());
```

CAPTURAS DE PANTALLA:

```
[CarlosSánchezPáez csp98@csp98-Sobremesa-Linux:~/Desktop/AC/Prácticas/practica3/src] 2018-05-06 domingo
$gcc -o scheduled-clauseModificado4 scheduled-clauseModificado4.c -fopenmp
[CarlosSánchezPáez csp98@csp98-Sobremesa-Linux:~/Desktop/AC/Prácticas/practica3/src] 2018-05-06 domingo
$./scheduled-clauseModificado4 16 1

Dentro de parallel.
dyn_var=0       nthreads_var=2          thread_limit_var=2147483647     run_sched_var=2         chunk=1
omp_get_num_threads()=2         omp_get_num_procs()=8   omp_in_parallel()=1
 thread 0 suma a[0]=0 suma=0
 thread 0 suma a[2]=2 suma=2
 thread 0 suma a[3]=3 suma=5
 thread 0 suma a[4]=4 suma=9
 thread 0 suma a[5]=5 suma=14
 thread 0 suma a[6]=6 suma=20
 thread 0 suma a[7]=7 suma=27
 thread 0 suma a[8]=8 suma=35
 thread 0 suma a[9]=9 suma=44
 thread 0 suma a[10]=10 suma=54
 thread 0 suma a[11]=11 suma=65
 thread 0 suma a[12]=12 suma=77
 thread 0 suma a[13]=13 suma=90
 thread 0 suma a[14]=14 suma=104
 thread 0 suma a[15]=15 suma=119
 thread 1 suma a[1]=1 suma=1
Fuera de 'parallel for' suma=119
Fuera de parallel.
 dyn_var=0       nthreads_var=2          thread_limit_var=2147483647     run_sched_var=2         chunk=1
omp_get_num_threads()=1         omp_get_num_procs()=8   omp_in_parallel()=0
```

RESPUESTA:

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

CAPTURA CÓDIGO FUENTE: `scheduled-clauseModificado5.c`

```
#pragma omp parallel for firstprivate(suma) lastprivate(suma) schedule(dynamic,chunk)
for (i = 0; i < n; i++) {
    if (i == 0) {   //Master
        omp_get_schedule(&schedule, &chunk_);
        printf("\nDentro de parallel.\ndyn_var=%d \t nthreads_var=%d \t thread_limit_var=%d \t run_sched_var=%d \t chunk=%d\n",
               omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit(), schedule, chunk_);
        printf("omp_get_num_threads()=%d \t omp_get_num_procs()=%d \t omp_in_parallel()=%d\n",
               omp_get_num_threads(), omp_get_num_procs(), omp_in_parallel());
    }
    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);
omp_get_schedule(&schedule, &chunk_);
printf("Fuera de parallel.\n dyn_var=%d \t nthreads_var=%d \t thread_limit_var=%d \t run_sched_var=%d \t chunk=%d\n",
       omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit(), schedule, chunk_);
printf("omp_get_num_threads()=%d \t omp_get_num_procs()=%d \t omp_in_parallel()=%d\n",
      omp_get_num_threads(), omp_get_num_procs(), omp_in_parallel());
```

CAPTURAS DE PANTALLA:

```
[CarlosSánchezPáez csp98@csp98-Sobremesa-Linux:~/Desktop/AC/Prácticas/practica3/src] 2018-05-06 domingo
$./scheduled-clauseModificado4 16 4

Dentro de parallel.
dyn_var=0       nthreads_var=2          thread_limit_var=2147483647     run_sched_var=2         chunk=1
omp_get_num_threads()=2         omp_get_num_procs()=8   omp_in_parallel()=1
 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 0 suma a[3]=3 suma=6
 thread 0 suma a[8]=8 suma=14
 thread 0 suma a[9]=9 suma=23
 thread 0 suma a[10]=10 suma=33
 thread 0 suma a[11]=11 suma=44
 thread 0 suma a[12]=12 suma=56
 thread 0 suma a[13]=13 suma=69
 thread 0 suma a[14]=14 suma=83
 thread 0 suma a[15]=15 suma=98
 thread 1 suma a[4]=4 suma=4
 thread 1 suma a[5]=5 suma=9
 thread 1 suma a[6]=6 suma=15
 thread 1 suma a[7]=7 suma=22
Fuera de 'parallel for' suma=98
Fuera de parallel.
 dyn_var=0      nthreads_var=2          thread_limit_var=2147483647     run_sched_var=2         chunk=1
omp_get_num_threads()=1         omp_get_num_procs()=8   omp_in_parallel()=0
```

RESPUESTA:

Como vemos, fuera de una región paralela las variables *num_threads* y *omp_in_parallel* cambian a 1 y 0, respectivamente. Es decir, un sólo proceso ejecuta y no estamos en región paralela.

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.

ELEGIR SUPERIOR O INFERIOR SEGÚN SE QUIERA.

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`

```
    for (int i = 0; i < TAM; i++) {
        temp = 0;
        for (int j = i; j < TAM; j++) {
            temp += m[i][j] * v1[j];
        }
        v2[i] = temp;
    }
```

CAPTURAS DE PANTALLA:

```
---------Valores iniciales------
1.000000 1.000000 1.000000 1.000000 1.000000
0.000000 1.000000 1.000000 1.000000 1.000000
0.000000 0.000000 1.000000 1.000000 1.000000
0.000000 0.000000 0.000000 1.000000 1.000000
0.000000 0.000000 0.000000 0.000000 1.000000
v1[0]=5.000000
v1[1]=5.000000
v1[2]=5.000000
v1[3]=5.000000
v1[4]=5.000000
--------------------------------
---------Valores finales------
v2[0]=25.000000
v2[1]=20.000000
v2[2]=15.000000
v2[3]=10.000000
v2[4]=5.000000
--------------------------------

Tiempo (seg): 0.000000802        Tamaño: 5      v2[0]=25.000000 v[4]=5.000000
```

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?

CAPTURA CÓDIGO FUENTE: `pmtv-OpenMP.c`

```
#pragma omp parallel for schedule(runtime)
  for (int i = 0; i < TAM; i++) {
    temp = 0;
    for (int j = i; j < TAM; j++) {
      temp += m[i][j] * v1[j];
    }
    v2[i] = temp;
  }
```

DESCOMPOSICIÓN DE DOMINIO:

CAPTURAS DE PANTALLA:

```
[CarlosSánchezPáez csp98@Ubuntu-Sobremesa:~/Desktop/AC/Prácticas/practica3/src] 2018-05-08 martes
$gcc -o pmtv_openmp pmtv_openmp.c -fopenmp -O2
[CarlosSánchezPáez csp98@Ubuntu-Sobremesa:~/Desktop/AC/Prácticas/practica3/src] 2018-05-08 martes
$./pmtv_openmp 1000

Tiempo (seg): 0.001617230        Tamaño: 1000   v2[0]=1000000.000000    v[999]=1000.000000
[CarlosSánchezPáez csp98@Ubuntu-Sobremesa:~/Desktop/AC/Prácticas/practica3/src] 2018-05-08 martes
$
```

TABLA RESULTADOS, SCRIPT Y GRÁFICA atcgrid

```
#!/bin/bash

export OMP_SCHEDULE="static"
echo "Static. Chunk por defecto."
./pmtv_openmp 15360

export OMP_SCHEDULE="static,1"
echo "Static. Chunk=1."
./pmtv_openmp 15360

export OMP_SCHEDULE="static,64"
echo "Static. Chunk=64."
./pmtv_openmp 15360
################################
export OMP_SCHEDULE="dynamic"
echo "Dynamic. Chunk por defecto."
./pmtv_openmp 15360

export OMP_SCHEDULE="dynamic,1"
echo "Dynamic. Chunk=1."
./pmtv_openmp 15360

export OMP_SCHEDULE="dynamic,64"
echo "Dynamic. Chunk=64."
./pmtv_openmp 15360

####################

export OMP_SCHEDULE="guided"
echo "Guided. Chunk por defecto."
./pmtv_openmp 15360

export OMP_SCHEDULE="guided,1"
echo "Guided. Chunk=1."
./pmtv_openmp 15360

export OMP_SCHEDULE="guided,64"
echo "Guided. Chunk=64."
./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	0.060223606	0.062205412	0.069614502
1	0.065692657	0.061345572	0.062420418
64	0.060986769	0.061024545	0.062412237
Chunk	**Static**	**Dynamic**	**Guided**
por defecto	0.054418269	0.061539549	0.067701003
1	0.062523460	0.061753399	0.060809558
64	0.065453784	0.064022752	0.065780441

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

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

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

CAPTURA CÓDIGO FUENTE: `pmm-secuencial.c`

```
for (int i = 0; i < TAM; i++) {
    for (int j = 0; j < TAM; j++) {
        temp = 0;
        for (int k = 0; k < TAM; k++)
            temp += m1[i][k] * m2[k][j];
        m3[i][j] = temp;
    }
}
```

CAPTURAS DE PANTALLA:

```
---------Valores iniciales------
------Matriz 1------
1.000000 0.000000 0.000000
0.000000 1.000000 0.000000
0.000000 0.000000 1.000000
------Matriz 2------
1.000000 0.000000 0.000000
0.000000 1.000000 0.000000
0.000000 0.000000 1.000000
---------------------------------
---------Valores finales------
1.000000 0.000000 0.000000
0.000000 1.000000 0.000000
0.000000 0.000000 1.000000
---------------------------------

Tiempo (seg): 0.000001606       Tamaño: 3       m3[0][0]=1.000000       m3[2][2]
=1.000000
```

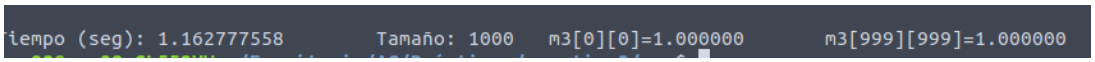
9. Implementar en paralelo la multiplicación de matrices cuadradas con OpenMP a partir del código escrito en el ejercicio anterior. Use las directivas, las cláusulas y las funciones de entorno que considere oportunas. Se debe paralelizar también la inicialización de las matrices. Dibuje en su cuaderno 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:

CAPTURA CÓDIGO FUENTE: `pmm-OpenMP.c`

```
    double temp=0;
    ini=omp_get_wtime();
    #pragma omp parallel for
    for (int i = 0; i < TAM; i++) {
        for (int j = 0; j < TAM; j++) {
            temp = 0;
            for (int k = 0; k < TAM; k++)
                temp += m1[i][k] * m2[k][j];
            m3[i][j] = temp;
        }
    }
    fin=omp_get_wtime();
    ncgt = fin-ini;
```

CAPTURAS DE PANTALLA:



10. Hacer un estudio de escalabilidad (ganancia en velocidad en función del número de cores) en atcgrid y en su PC del código paralelo implementado para dos tamaños de las matrices. Debe recordar usar `-O2` al compilar. El número de núcleos máximo en este estudio debe ser el igual al de núcleos físicos del computador. Presente los resultados del estudio en tablas de valores y en gráficas. Escoger los tamaños de manera que se observe diferentes curvas de escalabilidad en las gráficas que entregue en su cuaderno de prácticas (pruebe con valores de N entre 100 y 1500). Consulte la Lección 6/Tema 2. Incluya los scripts utilizado en el cuaderno de prácticas. NOTA: Nunca ejecute en atcgrid código que imprima todos los componentes del resultado.

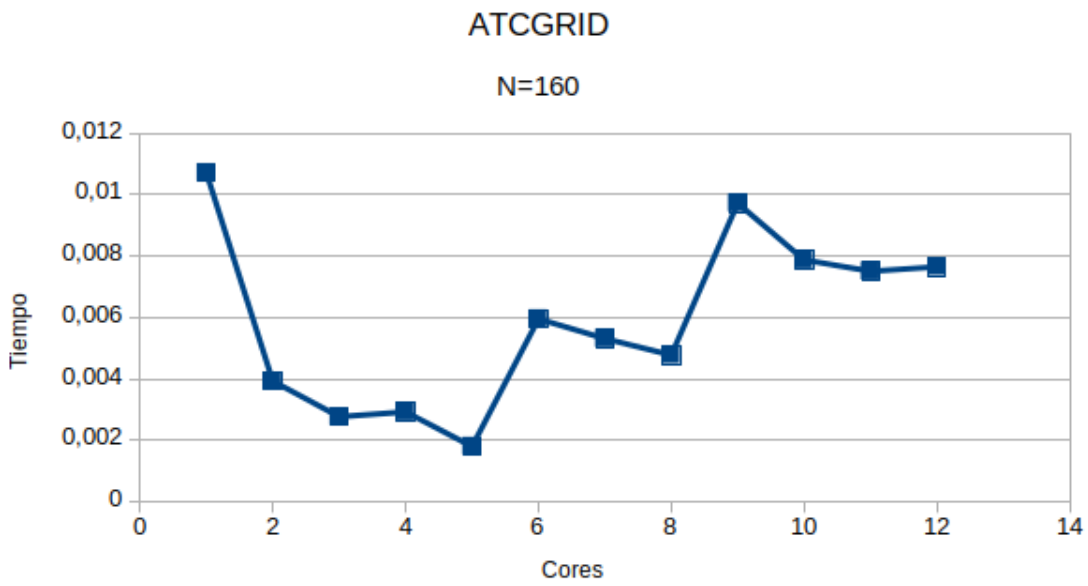
ESTUDIO DE ESCALABILIDAD EN atcgrid:

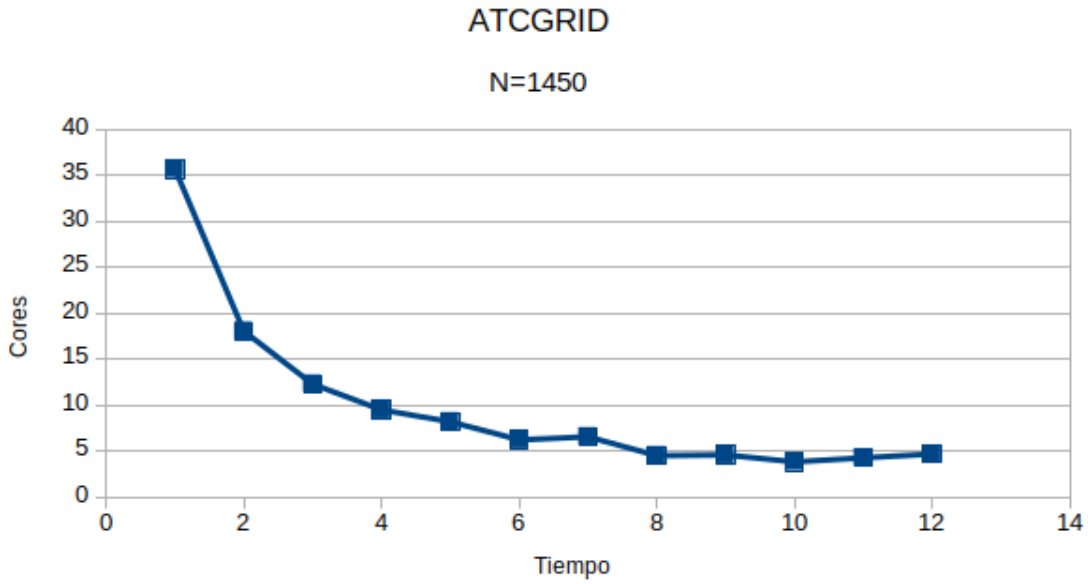
SCRIPT: pmm-OpenMP_atcgrid.sh

```
#!/bin/bash

for((N=1;N<13;N=N+1)) do
  export OMP_NUM_THREADS=$N
  echo "N=160;OMP_NUM_THREADS=$N"
  ./pmm-openmp 160
done
```

```
#!/bin/bash
for((N=1;N<13;N=N+1)) do
  export OMP_NUM_THREADS=$N
  echo "N=1450;OMP_NUM_THREADS=$N"
  ./pmm-openmp 1450
done
```





ESTUDIO DE ESCALABILIDAD EN PCLOCAL:

SCRIPT: pmm-OpenMP_pclocal.sh

```
#!/bin/bash

for((N=1;N<5;N=N+1)) do
  export OMP_NUM_THREADS=$N
  echo "N=160;OMP_NUM_THREADS=$N"
  ./pmm-openmp 160
done

for((N=1;N<5;N=N+1)) do
  export OMP_NUM_THREADS=$N
  echo "N=1450;OMP_NUM_THREADS=$N"
  ./pmm-openmp 1450
done
```