2º curso / 2º cuatr.

Grado Ing. Inform.

Arquitectura de Computadores (AC)

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

Estudiante (nombre y apellidos): Daniel Monjas Miguélez
Grupo de prácticas: Miércoles
Fecha de entrega: 10 de mayo de 2020
Fecha evaluación en clase: 13 de mayo de 2020

Antes de comenzar a realizar el trabajo de este cuaderno consultar el fichero con los normas de prácticas que se encuentra en SWAD

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

        int i, n=20, tid, x = 1;
        int a[n],suma=0, sumalocal;
        double t;

        if(argc < 3){
                fprintf(stderr,"[ERROR]-Iteratciones - Nº threads\n");
                exit(-1);
        }

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

        if(n>20)
                n = 20;

        if(x > 8)
                x = 8;

        for(i=0; i < n; i++){
                a[i]=i;
        }
        t=omp_get_wtime();
        #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);
        }

        t=omp_get_wtime() - t;

        printf("Tiempo de ejecución -> %8.6f \n", t);
}
```

CAPTURAS DE PANTALLA:

```
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp3/ejer1] 2020-04-15 mi
érc oles
$gcc -O2 -fopenmp -o if-clausemodificado if-clausemodificado.c
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp3/ejer1] 2020-04-15 mi
érc oles
$./if-clausemodificado 10 8
thread 7 suma de a[9]=9 sumalocal = 9
thread 2 suma de a[4]=4 sumalocal = 4
thread 1 suma de a[2]=2 sumalocal = 2
thread 1 suma de a[3]=3 sumalocal = 5
thread 4 suma de a[6]=6 sumalocal = 6
thread 3 suma de a[5]=5 sumalocal = 5
thread 0 suma de a[0]=0 sumalocal = 0
thread 0 suma de a[1]=1 sumalocal = 1
thread 5 suma de a[7]=7 sumalocal = 7
thread 6 suma de a[8]=8 sumalocal = 8
thread master =0 imprime suma=45
Tiempo de ejecución -> 0.004710
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp3/ejer1] 2020-04-15 mi
érc oles
$./if-clausemodificado 10 4
thread 1 suma de a[3]=3 sumalocal = 3
thread 1 suma de a[4]=4 sumalocal = 7
thread 1 suma de a[5]=5 sumalocal = 12
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 2 suma de a[6]=6 sumalocal = 6
thread 2 suma de a[7]=7 sumalocal = 13
thread 3 suma de a[8]=8 sumalocal = 8
thread 3 suma de a[9]=9 sumalocal = 17
thread master =0 imprime suma=45
Tiempo de ejecución -> 0.001380
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp3/ejer1] 2020-04-15 mi
érc oles
$./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
Tiempo de ejecución -> 0.000386
```

RESPUESTA: Se puede observar claramente que las iteraciones del bucle for, el cual calcula las variables sumalocal, divide sus iteraciones en el número de hebras que pasamos como segundo argumento. También se observa que el tiempo de ejecución aumenta significativamente, ya que se tarda más tiempo en la creación y destrucción de las hebras y su comunicación y sincronización que el tiempo que se ahorra, es decir, no se pasa un tamaño de problema suficientemente grande como para observar una mejora al paralelizar el bucle.

2. (a) Rellenar la Tabla 1 (se debe poner en la tabla el id del *thread* que ejecuta cada iteración) ejecutando los ejemplos del seminario `schedule-clause.c`, `scheduled-clause.c` y `scheduleg-clause.c` con dos *threads* (0,1) y unas entradas de:

- iteraciones: 16 (0,…15)
- chunck= 1, 2 y 4

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

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

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

RESPUESTA:
Se puede observar que en schedule(static) la planificación se asigna en modo round-robin, asignando un chunk a una hebra, otra a la siguiente, hasta llegar a la última y vuelta a empezar.
Se puede observar que en schedule(dynamic), si bien si se asignan a las hebras iteraciones en función del tamaño del chunk, es completamente aleatorio que chunk le toca a cada hebra y cuantos chunks le corresponden a una misma hebra.
Por último para schedule(guided), se puede observar que en primer lugar se asigna a una hebra un bloque largo, luego se asigna un bloque más pequeño, y así reduciendo el tamaño del bloque, hasta llegar a la última iteración.

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`

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


        #pragma omp single
        {
        omp_get_schedule(&kind,&modifier);
        printf("Dentro de 'parallel for'\ndyn-var=%d\nnthreads-var=%d\nthread-limit-var=%d\nrun-sched-
var(kind:%d,modifier: %d)
\n",omp_get_dynamic(),omp_get_max_threads(),omp_get_thread_limit(),kind,modifier);
        }
        }

        omp_get_schedule(&kind,&modifier);
        printf("Fuera de 'parallel for' suma=%d\n",suma);
        printf("Fuera de 'parallel for'\ndyn-var=%d\nnthreads-var=%d\nthread-limit-var=%d\nrun-sched-
var(kind:%d,modifier: %d)
\n",omp_get_dynamic(),omp_get_max_threads(),omp_get_thread_limit(),kind,modifier);

}
```

CAPTURAS DE PANTALLA:

```
$export OMP_DYNAMIC=FALSE
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp3/ejer3] 2020-04-15 mi
ércoles
$export OMP_NUM_THREADS=2
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp3/ejer3] 2020-04-15 mi
ércoles
$export OMP_THREAD_LIMIT=20
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp3/ejer3] 2020-04-15 mi
ércoles
$export OMP_SCHEDULE="static,2"
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp3/ejer3] 2020-04-15 mi
ércoles
$./scheduled-clause 4 2
 thread 0 suma a[0]=0 suma=0
 thread 0 suma a[1]=1 suma=1
 thread 1 suma a[2]=2 suma=2
 thread 1 suma a[3]=3 suma=5
Dentro de 'parallel for'
dyn-var=0       nthreads-var=2  thread-limit-var=20     run-sched-var(kind:1,modifier: 2)
Fuera de 'parallel for' suma=5
Fuera de 'parallel for'
dyn-var=0       nthreads-var=2  thread-limit-var=20     run-sched-var(kind:1,modifier: 2)
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp3/ejer3] 2020-04-15 mi
ércoles
$export OMP_DYNAMIC=TRUE
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp3/ejer3] 2020-04-15 mi
ércoles
$export OMP_NUM_THREADS=4
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp3/ejer3] 2020-04-15 mi
ércoles
$export OMP_THREAD_LIMIT=10
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp3/ejer3] 2020-04-15 mi
ércoles
$export OMP_SCHEDULE="dynamic,4"
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp3/ejer3] 2020-04-15 mi
ércoles
$./scheduled-clause 8 4
 thread 1 suma a[0]=0 suma=0
 thread 1 suma a[1]=1 suma=1
 thread 1 suma a[2]=2 suma=3
 thread 1 suma a[3]=3 suma=6
 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
Dentro de 'parallel for'
dyn-var=1       nthreads-var=4  thread-limit-var=10     run-sched-var(kind:2,modifier: 4)
Fuera de 'parallel for' suma=22
Fuera de 'parallel for'
dyn-var=1       nthreads-var=4  thread-limit-var=10     run-sched-var(kind:2,modifier: 4)
```

FRANCISCO

RESPUESTA:
Como se puede apreciar en las capturas de pantalla se imprimen los mismo valores dentro y fuera de la región parallel. Se deberá a que si no se realiza ninguna modificación sobre dichas variables, no afecta el hilo de ejecución, ya que estas variables van a ser 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`

```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;

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


        #pragma omp single
        {
        printf("Dentro de 'parallel for'\nomp_get_num_threads(): %d\tomp_get_num_procs():
%d\tomp_in_parallel():%d\n",omp_get_num_threads(),omp_get_num_procs(),omp_in_parallel());
        }
        }

        omp_get_schedule(&kind,&modifier);
        printf("Fuera de 'parallel for' suma=%d\n",suma);
        printf("Fuera de 'parallel for'\nomp_get_num_threads(): %d\tomp_get_num_procs():
%d\tomp_in_parallel():%d\n",omp_get_num_threads(),omp_get_num_procs(),omp_in_parallel());

}
```

CAPTURAS DE PANTALLA:

```
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp3/ejer4] 2020-04-15 mi
ércoles
$./scheduled-clause 8 4
 thread 4 suma a[0]=0 suma=0
 thread 4 suma a[1]=1 suma=1
 thread 4 suma a[2]=2 suma=3
 thread 4 suma a[3]=3 suma=6
 thread 3 suma a[4]=4 suma=4
 thread 3 suma a[5]=5 suma=9
 thread 3 suma a[6]=6 suma=15
 thread 3 suma a[7]=7 suma=22
Dentro de 'parallel for'
omp_get_num_threads(): 8        omp_get_num_procs():8   omp_in_parallel():1
Fuera de 'parallel for' suma=22
Fuera de 'parallel for'
omp_get_num_threads(): 1        omp_get_num_procs():8   omp_in_parallel():0
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp3/ejer4] 2020-04-15 mi
ércoles
$./scheduled-clause 4 1
 thread 4 suma a[0]=0 suma=0
 thread 3 suma a[2]=2 suma=2
 thread 7 suma a[3]=3 suma=3
 thread 2 suma a[1]=1 suma=1
Dentro de 'parallel for'
omp_get_num_threads(): 8        omp_get_num_procs():8   omp_in_parallel():1
Fuera de 'parallel for' suma=3
Fuera de 'parallel for'
omp_get_num_threads(): 1        omp_get_num_procs():8   omp_in_parallel():0
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp3/ejer4] 2020-04-15 mi
ércoles
$./scheduled-clause 6 2
 thread 5 suma a[0]=0 suma=0
 thread 6 suma a[4]=4 suma=4
 thread 6 suma a[5]=5 suma=9
 thread 3 suma a[2]=2 suma=2
 thread 3 suma a[3]=3 suma=5
 thread 5 suma a[1]=1 suma=1
Dentro de 'parallel for'
omp_get_num_threads(): 8        omp_get_num_procs():8   omp_in_parallel():1
Fuera de 'parallel for' suma=9
Fuera de 'parallel for'
omp_get_num_threads(): 1        omp_get_num_procs():8   omp_in_parallel():0
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp3/ejer4] 2020-04-15 mi
ércoles
$
```

RESPUESTA: Como se observa en la ejecución se obtienen valores distintos para omp_get_num_threads() y omp_in_parallel(), dentro y fuera del bucle. Esto se debe en primer lugar a que en en el single al ejecutar dicho bloque una sola hebra se detecta como código secuencial y por tanto el ordenador interpreta como que sólo una hebra está ejecutando. Por otro lado, es claro que en la sección de código donde hay un single no se ejecutará en paralelo y por tanto devolverá 0, es decir, false.

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


        #pragma omp single
        {
        omp_get_schedule(&kind,&modifier);
        printf("Dentro de 'parallel for' Antes de la modificación\ndyn-var=%d\tnthreads-var=%d\tthread-
limit-var=%d\trun-sched-var(kind:%d,modifier: %d)
\n",omp_get_dynamic(),omp_get_max_threads(),omp_get_thread_limit(),kind,modifier);

        kind=2;
        modifier=4;

        omp_set_dynamic(0);
        omp_set_num_threads(4);
        omp_set_schedule(kind,modifier);

        printf("Dentro de 'parallel for' Depués de la modificación\ndyn-var=%d\tnthreads-var=%d\tthread-
limit-var=%d\trun-sched-var(kind:%d,modifier: %d)
\n",omp_get_dynamic(),omp_get_num_threads(),omp_get_thread_limit(),kind,modifier);
        }
        }

        omp_get_schedule(&kind,&modifier);
        printf("Fuera de 'parallel for' suma=%d\n",suma);
        printf("Fuera de 'parallel for' Antes de la modificación\ndyn-var=%d\tnthreads-var=%d\tthread-
limit-var=%d\trun-sched-var(kind:%d,modifier: %d)
\n",omp_get_dynamic(),omp_get_max_threads(),omp_get_thread_limit(),kind,modifier);

        kind=1;
        modifier=2;

        omp_set_dynamic(0);
        omp_set_num_threads(4);
        omp_set_schedule(2,4);

        printf("Fuera de 'parallel for' Depués de la modificación\ndyn-var=%d\tnthreads-var=%d\tthread-
limit-var=%d\trun-sched-var(kind:%d,modifier: %d)
\n",omp_get_dynamic(),omp_get_num_threads(),omp_get_thread_limit(),kind,modifier);
```

CAPTURAS DE PANTALLA:

```
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp3/ejer5] 2020-04-15 mi
ércoles
$./scheduled-clause 8 4
 thread 1 suma a[0]=0 suma=0
 thread 1 suma a[1]=1 suma=1
 thread 1 suma a[2]=2 suma=3
 thread 1 suma a[3]=3 suma=6
 thread 7 suma a[4]=4 suma=4
 thread 7 suma a[5]=5 suma=9
 thread 7 suma a[6]=6 suma=15
 thread 7 suma a[7]=7 suma=22
Dentro de 'parallel for' Antes de la modificación
dyn-var=1       nthreads-var=8  thread-limit-var=2147483647     run-sched-var(kind:1,modifier: 4)
Dentro de 'parallel for' Depués de la modificación
dyn-var=0       nthreads-var=8  thread-limit-var=2147483647     run-sched-var(kind:2,modifier: 4)
Fuera de 'parallel for' suma=22
Fuera de 'parallel for' Antes de la modificación
dyn-var=1       nthreads-var=8  thread-limit-var=2147483647     run-sched-var(kind:1,modifier: 4)
Fuera de 'parallel for' Depués de la modificación
dyn-var=0       nthreads-var=1  thread-limit-var=2147483647     run-sched-var(kind:1,modifier: 2)
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp3/ejer5] 2020-04-15 mi
ércoles
$
```

RESPUESTA: Para esa ejecución en primer lugar habíamos fijado OMP_DYNAMIC=TRUE, OMP_NUM_THREADS=4 y OMP_SCHEDULE=”static,4”. Podemos apreciar como en el bucle se modifican run-sched-var y dyn-var cómo se ha indicado. Y de igual manera fuera del bucle con run-sched-var y dyn-var.

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`

```c
int main(int argc, char *argv[]){
    if(argc < 2){
        printf("Modo de ejecución: <programa> <filas/columnas>");
        exit(-1);
    }

    int filcol=atoi(argv[1]);
    int **matriz=NULL;
    int *vector=NULL;
    int *resultado=NULL;
    int acumulador=0;

    vector=(int *)malloc(sizeof(int)*filcol);
    resultado=(int *)malloc(sizeof(int)*filcol);
    matriz=(int **)malloc(sizeof(int*)*filcol);

    for(int i=0; i < filcol; i++)
        matriz[i]=(int *)malloc(sizeof(int)*filcol);

    srand((unsigned int) getpid());

    //Inicializo la matriz y el vector que se van a multiplicar
    for(int i=0; i < filcol; i++){
        for(int j=0; j < filcol; j++){
            if(j <= i)
                matriz[i][j]=rand() % 10;

            else
                matriz[i][j]=0;

        }

        vector[i] = rand()%10;
        resultado[i]=0;
    }

    //Realizo el producto
    for(int i=0; i < filcol; i ++){
        for(int j=0; j <= i; j++)
            acumulador += matriz[i][j]*vector[j];

        resultado[i]=acumulador;
        acumulador=0;
    }

    if(filcol < 5){
    //Muestro matriz, vector y resultado
    for(int i=0; i < filcol; i++){
        for(int j=0; j < filcol; j++)
            printf("M[%d][%d]=%d ",i,j,matriz[i][j]);

        printf("\n");

    }

    printf("\n");

    for(int i=0; i < filcol; i++)
        printf("v[%d]=%d ", i, vector[i]);

    printf("\n\n");

    for(int i=0; i < filcol; i++)
        printf("r[%d]=%d ", i, resultado[i]);

    printf("\n");
    }

    else{
        printf("r[0]=%d, r[%d]=%d\n", resultado[0],filcol-1, resultado[filcol-1]);
    }
```

CAPTURAS DE PANTALLA:

```
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp3/ejer6] 2020-04-25 sá
bado
$./pmtv-secuencial 4
M[0][0]=7 M[0][1]=0 M[0][2]=0 M[0][3]=0
M[1][0]=5 M[1][1]=3 M[1][2]=0 M[1][3]=0
M[2][0]=5 M[2][1]=6 M[2][2]=8 M[2][3]=0
M[3][0]=9 M[3][1]=8 M[3][2]=8 M[3][3]=4

v[0]=7 v[1]=3 v[2]=6 v[3]=9

r[0]=49 r[1]=44 r[2]=101 r[3]=171
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp3/ejer6] 2020-04-25 sá
bado
$./pmtv-secuencial 12
r[0]=8, r[11]=138
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp3/ejer6] 2020-04-25 sá
bado
$./pmtv-secuencial 20
r[0]=63, r[19]=447
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp3/ejer6] 2020-04-25 sá
bado
$
```

Respuesta: Para el producto de matriz por vector se tenía un orden de complejidad de $O(n^2)$, ya que se disponía de dos bucles anidados que recorrían el número total de componentes de la matriz para el producto. En este caso la complejidad sería O(n*(n-i)), ya que en el producto no se llegan a recorrer todas las columnas de la matriz (de ahí el n-i), mientras que si se recorren todas las filas. Recalcar que el programa implementado crea una matriz triangular inferior, por eso el número de columnas recorridas es igual al número de la fila en la que nos encontremos.

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) Para static usa chunk=1, para dynamic chunk=1, y para guided también usa chunk=1. Para esto lo que he hecho es que en la ejecución del programa aparte de mostrar el tiempo y la primera y última componente del resultado, llamo a omp_get_schedule(&kind,&modifier), y los muestro, pudiendo ver así el tipo de planificación que se usa y el tamaño del chunk. Para static se daba que el chunk=0, pero buscando en el seminario encontré que el valor por defecto para el chunk era 1. También lo comprobé poniendo que hebra realizaba cada iteración para un tamaño pequeño (12 en mi caso) y se ve que por default tienen chunk 1.

(b) En mi programa he utilizado como tamaño 17280, que es a la vez mayor que 15360, multiplo de 12, que

es el número de cores usado en atcgrid, y múltiplo de 64. Sabemos que el bucle realiza tantas iteraciones cómo se le pase el tamaño y como el tamaño es múltiplo del número de cores y en static la planificación se realiza siguiendo el método round-robin, sabemos que cada hebra realizará un total de 17280/12=1440 iteraciones. En cada una de estas iteraciones la hebra realiza tantos productos y tantas sumas como el número de la iteración mas uno, es decir, en la iteración cero la hebra que tenga esta iteración asignada hara 1 suma y un producto, y así sucesivamente. He creado un programa en C++ que realiza el cálculo de operaciones, al que se le pasa por parámetros la iteración inicial y el chunk, y para todas las iteraciones iniciales (1,…,12), he obtenido que el número de sumas y multiplicaciones es 12434400, para chunks de 1, que resulta ser también el que se toma por defecto. (En total hay 24868800 operaciones).

Para chunks de 64 también se da ese número de sumas y multiplicaciones 12434400.

El programa usado para calcular las operaciones viene adjunto en la carpeta ejer7.

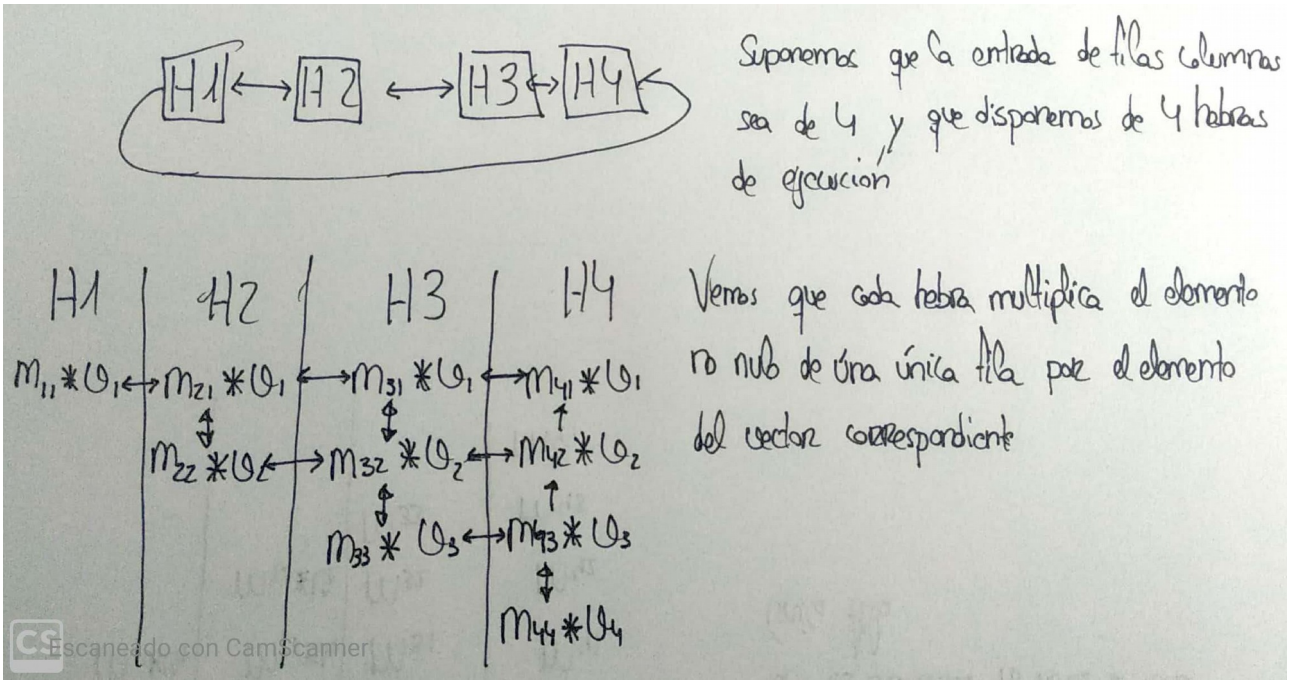
(c) Para dynamic, como se asignará iteraciones del bucle a cada thread en función de su disponibilidad, probablemente las cargas de trabajo, es decir, el número de operaciones y multiplicaciones que realiza un thread, será dispar, ya que puede darse que a una hebra puede que se le asignen más iteraciones un valor de i alto, lo que implicará que realizará más sumas y multiplicaciones viendo en como se ha diseñado el programa.

Por otro lado para guided, como el tamaño del bloque se va reduciendo las cargas de trabajo más o menos se irán equilibrando, pues si bien puede ser que una hebra en una iteración realiza más multiplicaciones y sumas simplemente por el número de iteración que le ha tocado, se dará que tendrá las misma carga de trabajo o incluso menor que otra hebra a la que schedule le asignó un número de iteraciones anteriormente, por el simple hecho de que al haberse planificado antes recibirá un bloque más grande, es decir, más iteraciones.

CAPTURA CÓDIGO FUENTE: `pmtv-OpenMP.c`

```
srand((unsigned int) getpid());

//Inicializo la matriz y el vector que se van a multiplicar
#pragma omp for private(i,j)
for(i=0; i < filcol; i++){
    for(j=0; j < filcol; j++){
        if(j <= i)
            matriz[i][j]=rand() % 10;

        else
            matriz[i][j]=0;

    }

    vector[i] = rand()%10;
    resultado[i]=0;
}

clock_gettime(CLOCK_REALTIME,&cgt1);
//Realizo el producto
#pragma omp parallel private(i,j)
{

#pragma omp for schedule(runtime) reduction(+:acumulador)
for( i=0; i < filcol; i ++){
    for( j=0; j <= i; j++){
        acumulador += matriz[i][j]*vector[j];
    }

    resultado[i]=acumulador;
    acumulador=0;
}
}

clock_gettime(CLOCK_REALTIME,&cgt2);

ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+ (double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e
```

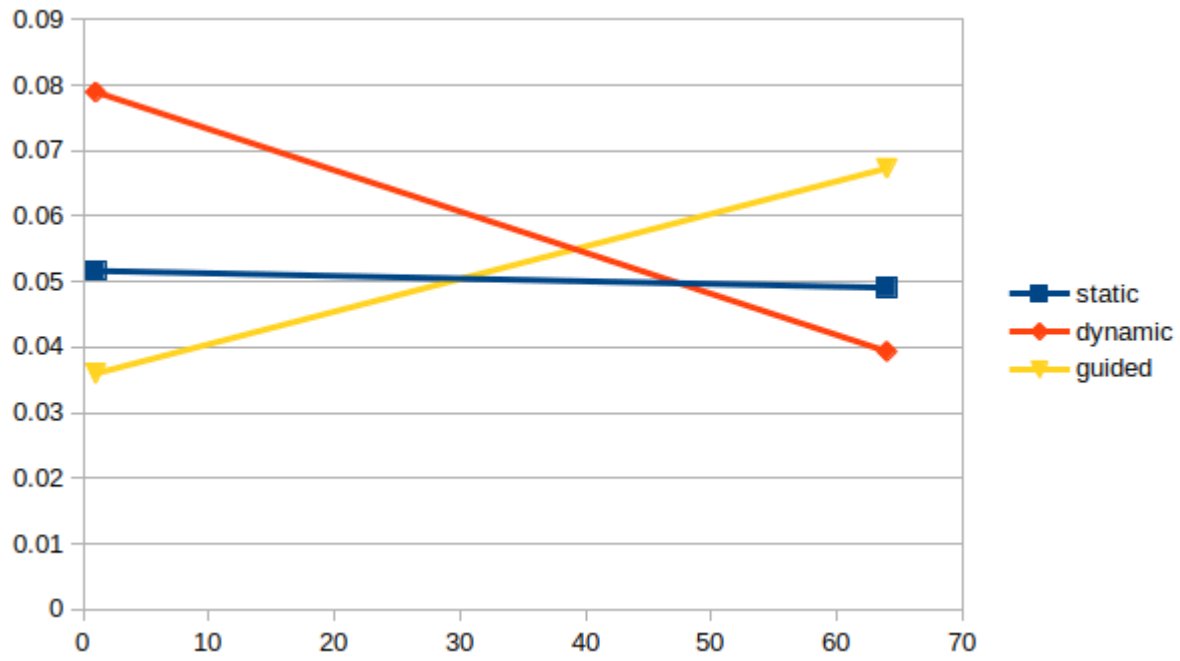
DESCOMPOSICIÓN DE DOMINIO:



CAPTURAS DE PANTALLA:

```
[DanielMonjasMiguelez e1estudiante18@atcgrid:~/bp3/ejer7] 2020-04-26 domingo
$bash pmvt-OpenMP_atcgrid.sh dynamic
Schedule: 2,1

r[0]=42, r[17279]=348649
Tiempo: 0.079754
Schedule: 2,1

r[0]=0, r[17279]=350082
Tiempo: 0.067225
Schedule: 2,64

r[0]=0, r[17279]=349683
Tiempo: 0.072086
[DanielMonjasMiguelez e1estudiante18@atcgrid:~/bp3/ejer7] 2020-04-26 domingo
$bash pmvt-OpenMP_atcgrid.sh guided
Schedule: 3,1

r[0]=35, r[17279]=350960
Tiempo: 0.038804
Schedule: 3,1

r[0]=30, r[17279]=350771
Tiempo: 0.058153
Schedule: 3,64

r[0]=0, r[17279]=349876
Tiempo: 0.083737
[DanielMonjasMiguelez e1estudiante18@atcgrid:~/bp3/ejer7] 2020-04-26 domingo
$
```

TABLA RESULTADOS, SCRIPT Y GRÁFICA atcgrid



SCRIPT: pmvt-OpenMP_atcgrid.sh

```
[DanielMonjasMiguelez e1estudiante18@atcgrid:~/bp3/ejer7] 2020-04-26 domingo
$cat pmvt-OpenMP_atcgrid.sh
#!/bin/bash

sched=$1
j=0

for i in 1 2 3
do
        if (($i == 1))
        then
                export OMP_SCHEDULE="$1"
                srun -n1 -p ac --account ac --cpus-per-task=12 --hint=nomultithread ./pmtv-OpenMP 17280
        fi

        if (( $i == 2))
        then
                export OMP_SCHEDULE="$1,1"
                srun -n1 -p ac --account ac --cpus-per-task=12 --hint=nomultithread ./pmtv-OpenMP 17280
        fi

        if (( $i == 3))
        then
                export OMP_SCHEDULE="$1,64"
                srun -n1 -p ac --account ac --cpus-per-task=12 --hint=nomultithread ./pmtv-OpenMP 17280
        fi
done
```

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	0.142624	0.078934	0.101316
1	0.051655	0.121551	0.036001
64	0.077323	0.048042	0.100479
Chunk	**Static**	**Dynamic**	**Guided**
por defecto	0.073407	0.127539	0.096096
1	0.062430	0.099715	0.115729
64	0.049090	0.039355	0.067314

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`

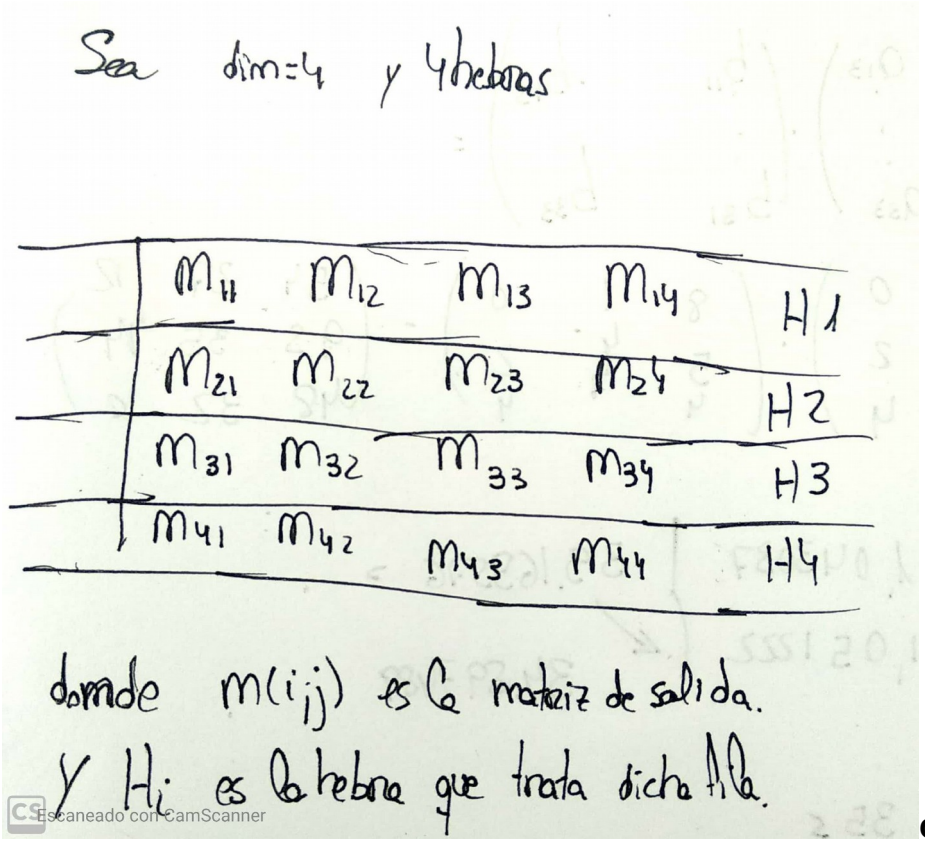
```c
int dimension = atoi(argv[1]);
int ** matriz_1;
int ** matriz_2;
int ** resultado;
int acumulador=0;

matriz_1 = (int **) malloc(sizeof(int *)*dimension);
matriz_2 = (int **) malloc(sizeof(int *)*dimension);
resultado = (int **) malloc(sizeof(int *)*dimension);

for(int i=0; i < dimension; i++){
    matriz_1[i]=(int *) malloc(sizeof(int)*dimension);
    matriz_2[i]=(int *) malloc(sizeof(int)*dimension);
    resultado[i]=(int *) malloc(sizeof(int)*dimension);
}

for(int i=0; i < dimension; i++){
    for(int j=0; j < dimension; j++){
        matriz_1[i][j]=rand()%10;
        matriz_2[i][j]=rand()%10;
    }
}


for(int i=0; i < dimension; i++){
    for(int j=0; j < dimension; j++){
        for(int k=0; k < dimension; k++){
            acumulador += matriz_1[i][k] * matriz_2[k][j];
        }

        resultado[i][j]=acumulador;

        acumulador = 0;
    }
}

printf("m[0][0]=%d --- m[%d][%d]=%d \n", resultado[0][0], dimension-1, dimension-1,


for(int i=0; i < dimension; i++){
    free(matriz_1[i]);
    free(matriz_2[i]);
    free(resultado[i]);
}
```

CAPTURAS DE PANTALLA:

```
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp3/ejer8] 2020-05-06 mi
ércoles
$./producto_matriz_secuencial 3
Tiempo = 0.000002
a[0][0]=7 a[0][1]=6 a[0][2]=7
a[1][0]=2 a[1][1]=1 a[1][2]=1
a[2][0]=6 a[2][1]=7 a[2][2]=4

b[0][0]=0 b[0][1]=0 b[0][2]=6
b[1][0]=5 b[1][1]=0 b[1][2]=3
b[2][0]=5 b[2][1]=6 b[2][2]=2

m[0][0]=65 m[0][1]=42 m[0][2]=74
m[1][0]=10 m[1][1]=6 m[1][2]=17
m[2][0]=55 m[2][1]=24 m[2][2]=65

[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp3/ejer8] 2020-05-06 mi
ércoles
$./producto_matriz_secuencial 100
Tiempo = 0.003105
m[0][0]=2049 --- m[99][99]=1989
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp3/ejer8] 2020-05-06 mi
ércoles
$./producto_matriz_secuencial 500
Tiempo = 0.096114
m[0][0]=9910 --- m[499][499]=9984
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp3/ejer8] 2020-05-06 mi
ércoles
$./producto_matriz_secuencial 1000
Tiempo = 0.853872
m[0][0]=19847 --- m[999][999]=20104
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp3/ejer8] 2020-05-06 mi
ércoles
$./producto_matriz_secuencial 2000
Tiempo = 33.431722
m[0][0]=40231 --- m[1999][1999]=40520
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp3/ejer8] 2020-05-06 mi
ércoles
$
```

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`

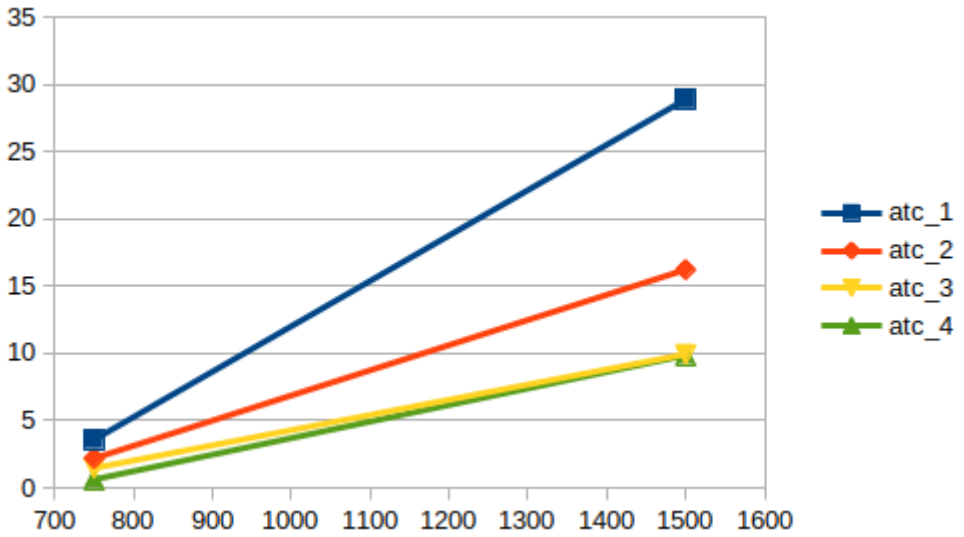
```
tantes = omp_get_wtime();

#pragma omp parallel for private(i,j)
    for(i=0; i < dimension; i++){
        for(int j=0; j < dimension; j++){
            matriz_1[i][j]=rand()%10;
            matriz_2[i][j]=rand()%10;
        }
    }

#pragma omp parallel for private(i,j,acumulador)
    for(i=0; i < dimension; i++){
        for(j=0; j < dimension; j++){
            for(k=0; k < dimension; k++){
                acumulador += matriz_1[i][k] * matriz_2[k][j];
            }

            resultado[i][j]=acumulador;

            acumulador = 0;
        }
    }

tdespues = omp_get_wtime();

ncgt = tdespues - tantes;

printf("Tiempo = %f \n",ncgt);
```

CAPTURAS DE PANTALLA:

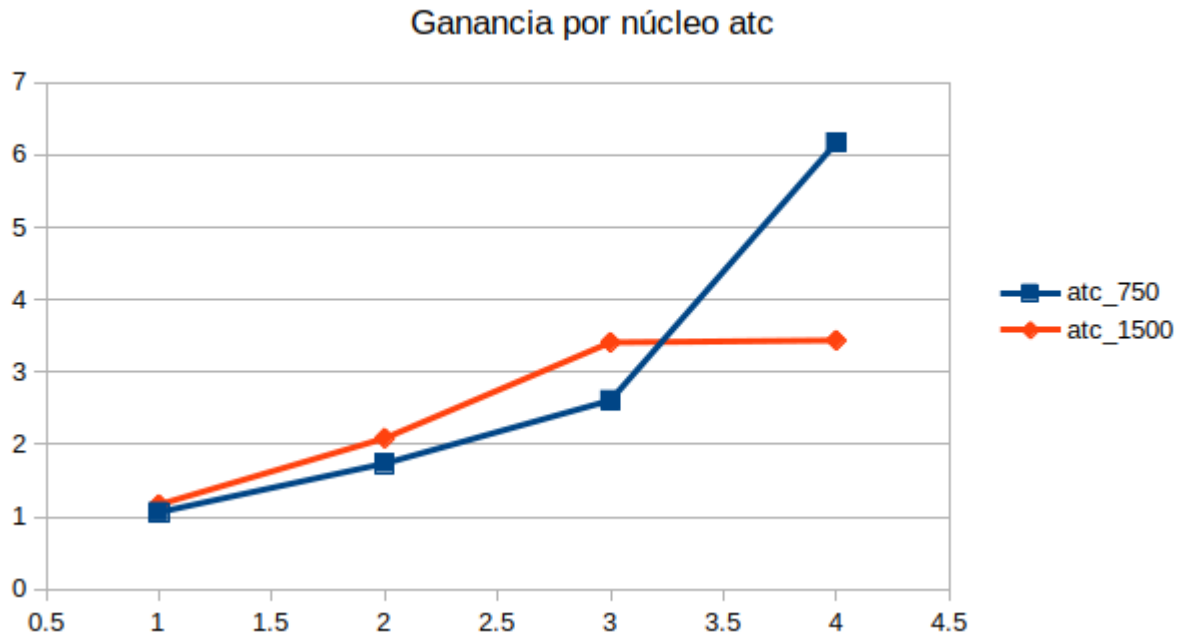
```
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp3/ejer9] 2020-05-06 mi
ércoles
$./producto_matriz_paralelo 3
Tiempo = 0.001582
a[0][0]=3 a[0][1]=5 a[0][2]=5
a[1][0]=5 a[1][1]=3 a[1][2]=3
a[2][0]=2 a[2][1]=3 a[2][2]=6

b[0][0]=7 b[0][1]=8 b[0][2]=6
b[1][0]=1 b[1][1]=3 b[1][2]=5
b[2][0]=9 b[2][1]=5 b[2][2]=0

m[0][0]=71 m[0][1]=64 m[0][2]=43
m[1][0]=65 m[1][1]=64 m[1][2]=45
m[2][0]=71 m[2][1]=55 m[2][2]=27

[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp3/ejer9] 2020-05-06 mi
ércoles
$./producto_matriz_paralelo 100
Tiempo = 0.005281
m[0][0]=1691 --- m[99][99]=2148
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp3/ejer9] 2020-05-06 mi
ércoles
$./producto_matriz_paralelo 500
Tiempo = 0.082283
m[0][0]=9440 --- m[499][499]=9340
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp3/ejer9] 2020-05-06 mi
ércoles
$./producto_matriz_paralelo 1000
Tiempo = 1.083444
m[0][0]=20165 --- m[999][999]=20645
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp3/ejer9] 2020-05-06 mi
ércoles
$./producto_matriz_paralelo 2000
Tiempo = 9.515859
m[0][0]=40967 --- m[1999][1999]=40933
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp3/ejer9] 2020-05-06 mi
ércoles
$
```

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:



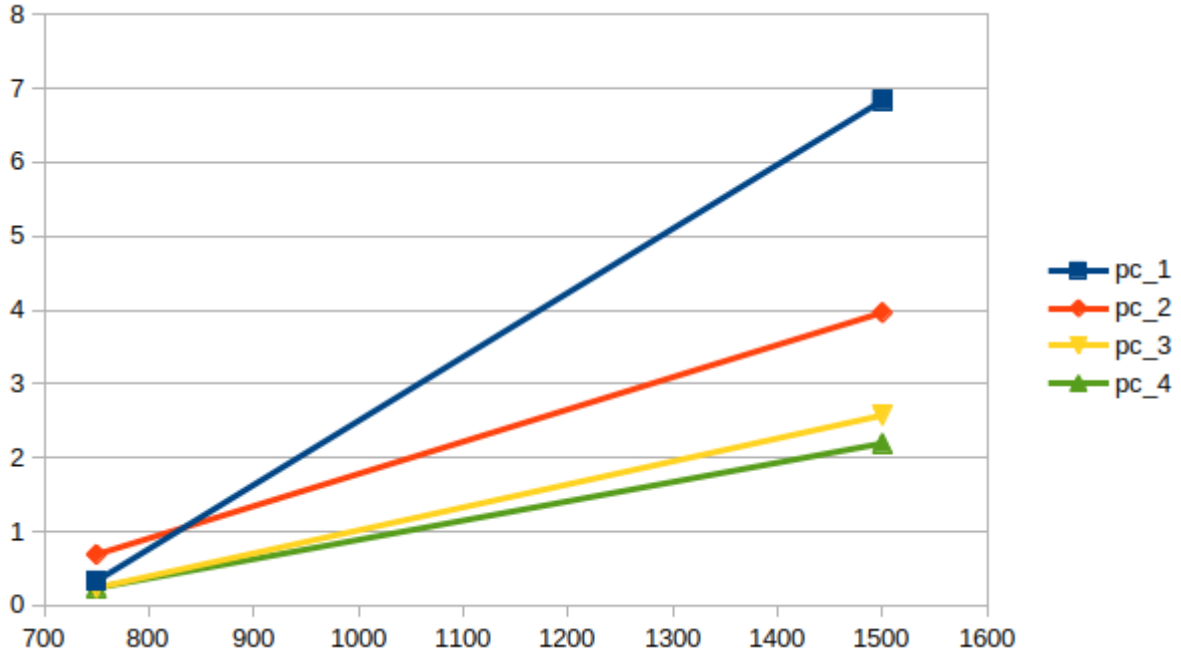
atcgrid	1	2	3	4	tsecuencial
750	3.570775	2.189739	1.458598	0.617293	3.808348
1500	28.884375	16.215836	9.925242	9.842699	33.846482
Ganancia atc	1	2	3	4	
750	1.066532615	1.739178962	2.610964776	6.169433316	
1500	1.171792085	2.087248662	3.410141738	3.438739923	



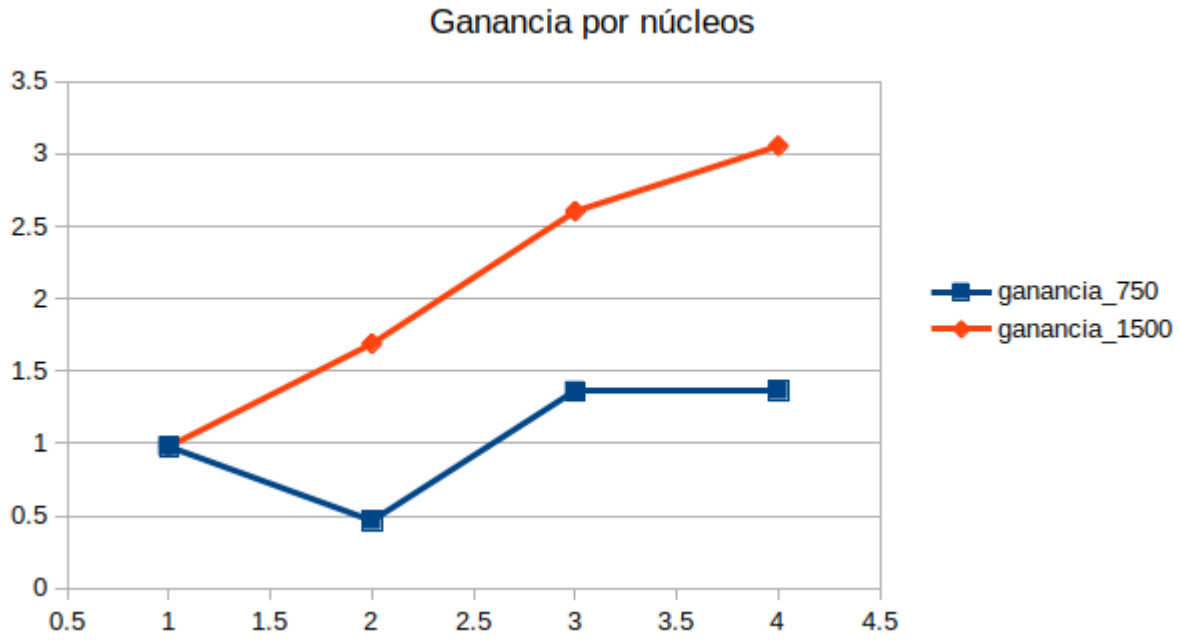
SCRIPT: pmm-OpenMP_atcgrid.sh

```
#!/bin/bash

for i in {1..4}
do
        export OMP_NUM_THREADS=$i

        for j in 1 2
        do
                if(($j == 1))
                then
                        srun -n1 -p ac --account ac --cpus-per-task=$i --hint=nomultithread ./
producto_matriz_paralelo 750
                fi

                if(($j == 2))
                then
                        srun -n1 -p ac --account ac --cpus-per-task=$i --hint=nomultithread ./
producto_matriz_paralelo 1500
                fi
        done
done
```

ESTUDIO DE ESCALABILIDAD EN PCLOCAL:



pclocal	1	2	3	4	tsecuencial
750	0.329741	0.689623	0.237392	0.236032	0.322757
1500	6.835866	3.964579	2.572687	2.192878	6.699087
Ganancia pcl	1	2	3	4	
750	0.97881974	0.468019483	1.3595951	1.367428993	
1500	0.979990977	1.689734774	2.603926167	3.054929184	



SCRIPT: pmm-OpenMP_pclocal.sh

```
#!/bin/bash

for i in {1..4}
do
        export OMP_NUM_THREADS=$i

        for j in 1 2
        do
                if(($j == 1))
                then
                        ./producto_matriz_paralelo 750
                fi

                if(($j == 2))
                then
                        ./producto_matriz_paralelo 1500
                fi
        done
done
```