

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): Bryan Moreno Picamán

Grupo de prácticas: A3

Fecha de entrega: 11/05/2015

Fecha evaluación en clase: 19/05/2015

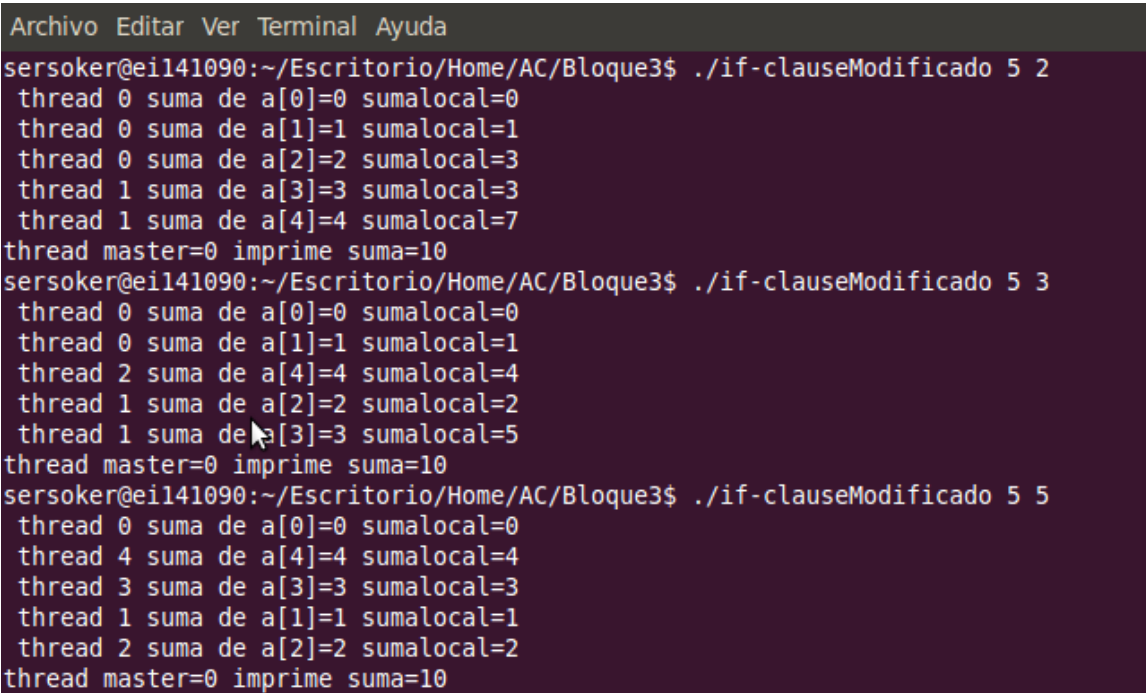
1. Usar la cláusula `num_threads(x)` en el ejemplo del seminario `if_clause.c`, y añadir un parámetro de entrada al programa que fije el valor `x` que se va a usar en la cláusula. Incorporar en el cuaderno de trabajo de esta práctica volcados de pantalla con ejemplos de ejecución que ilustren la funcionalidad de esta cláusula y explicar por qué lo ilustran.

CÓDIGO FUENTE: `if-clauseModificado.c`

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
int main(int argc, char **argv)
{
    int i, n=20, tid,x;
    int a[n], suma=0, sumalocal;
    if(argc < 3) {
        fprintf(stderr, "[ERROR]-Falta iteraciones-Falta threads\n");
        exit(-1);
    }
    n = atoi(argv[1]);
    x = atoi(argv[2]);
    if (n>20) n=20;

    for (i=0; i<n; i++) {
        a[i] = i;
    }
    omp_set_num_threads(5);
    #pragma omp parallel if(n>4) default(none) num_threads(x) \
    private(sumalocal,tid) shared(a,suma,n)
    { sumalocal=0;
        tid=omp_get_thread_num();
        #pragma omp for private(i) schedule(static) nowait
        for (i=0; i<n; i++)
        { sumalocal += a[i];
            printf(" thread %d suma de a[%d]=%d sumalocal=%d \n",
            tid,i,a[i],sumalocal);
        }
        #pragma omp atomic
        suma += sumalocal;
        #pragma omp barrier
        #pragma omp master
        printf("thread master=%d imprime suma=%d\n",tid,suma);
    }
}
```

CAPTURAS DE PANTALLA:



```
Archivo Editar Ver Terminal Ayuda
sersoker@eil41090:~/Escritorio/Home/AC/Bloque3$ ./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
sersoker@eil41090:~/Escritorio/Home/AC/Bloque3$ ./if-clauseModificado 5 3
thread 0 suma de a[0]=0 sumalocal=0
thread 0 suma de a[1]=1 sumalocal=1
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 master=0 imprime suma=10
sersoker@eil41090:~/Escritorio/Home/AC/Bloque3$ ./if-clauseModificado 5 5
thread 0 suma de a[0]=0 sumalocal=0
thread 4 suma de a[4]=4 sumalocal=4
thread 3 suma de a[3]=3 sumalocal=3
thread 1 suma de a[1]=1 sumalocal=1
thread 2 suma de a[2]=2 sumalocal=2
thread master=0 imprime suma=10
```

RESPUESTA: Al ponerle esto, se usa como numero de hebras del for el valor de x que metemos y reparte las iteraciones del for entre estos. Se puede ver en la captura el reparto de forma clara.

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
- c

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-clause.g.c		
	1	2	4	1	2	4	1	2	4
0	0	0	0	1	0	1	0	0	1
1	1	0	0	0	1	1	0	0	1
2	0	1	0	0	0	1	0	0	1
3	1	1	0	0	0	1	0	0	1
4	0	0	1	0	0	0	0	0	1
5	1	0	1	0	0	0	0	0	1
6	0	1	1	0	0	0	0	0	1
7	1	1	1	0	0	0	0	0	1
8	0	0	0	0	0	0	1	1	0
9	1	0	0	0	0	0	1	1	0
10	0	1	0	0	0	0	1	1	0
11	1	1	0	0	0	0	1	1	0
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).

Tabla2. 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-clause.g.c		
	1	2	4	1	2	4	1	2	4
0	0	0	0	2	2	3	0	2	3
1	1	0	0	0	2	3	0	2	3
2	2	1	0	3	3	3	0	2	3
3	3	1	0	1	3	3	0	2	3
4	0	2	1	0	1	1	3	1	2
5	1	2	1	0	1	1	3	1	2
6	2	3	1	0	0	1	3	1	2
7	3	3	1	0	0	1	2	3	2
8	0	0	2	0	0	0	2	3	0
9	1	0	2	0	0	0	2	3	0
10	2	1	2	0	0	0	1	0	0
11	3	1	2	0	0	0	1	0	0
12	0	2	3	0	0	2	0	0	1
13	1	2	3	0	0	2	0	0	1
14	2	3	3	0	0	2	0	0	1
15	3	3	3	0	0	2	0	0	1

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

RESPUESTA:

- `Schedule(static,chunk)`-> En tiempo de ejecución, iteraciones de chunk en chunk
- `Schedule(dynamic,chunk)`-> Se dividen en chunk, y se van dando conforme los threads quedan libres (los threads más rápidos ejecutan más unidades)
- `Schedule(guided,chunk)`-> Reparte el trabajo entre el número de hebras, quedando n iteraciones, las cuales reparte de nuevo a la siguiente hebra ($8/4=2$ quedan 6 $6/4=1,5...$)

3. Añadir al programa `scheduled-clause.c` lo necesario para que imprima el valor de las variables de control `dyn-var`, `nthreads-var`, `thread-limit-var` y `run-sched-var` dentro (debe imprimir sólo un thread) y fuera de la región paralela. Realizar varias ejecuciones usando variables de entorno para modificar estas variables de control antes de la ejecución. Incorporar en su cuaderno de prácticas volcados de pantalla de estas ejecuciones. ¿Se imprimen valores distintos dentro y fuera de la región paralela?

CÓDIGO FUENTE: `scheduled-clauseModificado.c`

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

main(int argc, char **argv) {
    int i, n=200, chunk, a[n], suma=0;
    int modifier=0, modifier2=0;
    omp_sched_t kind, kind2;

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

        #pragma omp single
        {
            omp_get_schedule(&kind, &modifier);
            printf("DENTRO: dyn-var: %d nthreads-var:%d thread-limit-var:%d run-
sched-var: Modifier:%d Kind:%d\n",
              omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit(),
              modifier, kind);
        }

        omp_get_schedule(&kind2, &modifier2);
        printf("FUERA: dyn-var: %d nthreads-var:%d thread-limit-var:%d run-
sched-var: Modifier:%d Kind:%d\n",
          omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit(),
          modifier2, kind2);

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

CAPTURAS DE PANTALLA:**1:**

```

sersoker@toshiba14:~/Dropbox/Universidad/AC/Practicas/Bloque3$ export OMP_NUM_THREADS=2
sersoker@toshiba14:~/Dropbox/Universidad/AC/Practicas/Bloque3$ ./3 10 2
thread 0 suma a[0]=0 suma=0
thread 0 suma a[1]=1 suma=1
thread 0 suma a[4]=4 suma=5
thread 0 suma a[5]=5 suma=10
thread 0 suma a[6]=6 suma=16
thread 0 suma a[7]=7 suma=23
thread 0 suma a[8]=8 suma=31
thread 0 suma a[9]=9 suma=40
thread 1 suma a[2]=2 suma=2
thread 1 suma a[3]=3 suma=5
DENTRO: dyn-var: 0 nthreads-var:2 thread-limit-var:2147483647 run-sched-var: Modifier:1 Kind:2
FUERA: dyn-var: 0 nthreads-var:2 thread-limit-var:2147483647 run-sched-var: Modifier:1 Kind:2
Fuera de 'parallel for' suma=40
sersoker@toshiba14:~/Dropbox/Universidad/AC/Practicas/Bloque3$ export OMP_NUM_THREADS=3
sersoker@toshiba14:~/Dropbox/Universidad/AC/Practicas/Bloque3$ ./3 10 2
thread 0 suma a[0]=0 suma=0
thread 0 suma a[1]=1 suma=1
thread 0 suma a[6]=6 suma=7
thread 0 suma a[7]=7 suma=14
thread 0 suma a[8]=8 suma=22
thread 0 suma a[9]=9 suma=31
thread 1 suma a[4]=4 suma=4
thread 1 suma a[5]=5 suma=9
thread 2 suma a[2]=2 suma=2
thread 2 suma a[3]=3 suma=5
DENTRO: dyn-var: 0 nthreads-var:3 thread-limit-var:2147483647 run-sched-var: Modifier:1 Kind:2
FUERA: dyn-var: 0 nthreads-var:3 thread-limit-var:2147483647 run-sched-var: Modifier:1 Kind:2
Fuera de 'parallel for' suma=31

```

2:

```

sersoker@toshiba14:~/Dropbox/Universidad/AC/Practicas/Bloque3$ export OMP_DYNAMIC=true
sersoker@toshiba14:~/Dropbox/Universidad/AC/Practicas/Bloque3$ ./3 10 2
thread 0 suma a[0]=0 suma=0
thread 0 suma a[1]=1 suma=1
thread 0 suma a[4]=4 suma=5
thread 0 suma a[5]=5 suma=10
thread 0 suma a[6]=6 suma=16
thread 0 suma a[7]=7 suma=23
thread 0 suma a[8]=8 suma=31
thread 0 suma a[9]=9 suma=40
thread 1 suma a[2]=2 suma=2
thread 1 suma a[3]=3 suma=5
DENTRO: dyn-var: 1 nthreads-var:3 thread-limit-var:2147483647 run-sched-var: Modifier:1 Kind:2
FUERA: dyn-var: 1 nthreads-var:3 thread-limit-var:2147483647 run-sched-var: Modifier:1 Kind:2
Fuera de 'parallel for' suma=40
sersoker@toshiba14:~/Dropbox/Universidad/AC/Practicas/Bloque3$ export OMP_DYNAMIC=false
sersoker@toshiba14:~/Dropbox/Universidad/AC/Practicas/Bloque3$ ./3 10 2
thread 0 suma a[0]=0 suma=0
thread 0 suma a[1]=1 suma=1
thread 0 suma a[4]=4 suma=5
thread 0 suma a[5]=5 suma=10
thread 0 suma a[6]=6 suma=16
thread 0 suma a[7]=7 suma=23
thread 0 suma a[8]=8 suma=31
thread 0 suma a[9]=9 suma=40
thread 1 suma a[2]=2 suma=2
thread 1 suma a[3]=3 suma=5
DENTRO: dyn-var: 0 nthreads-var:3 thread-limit-var:2147483647 run-sched-var: Modifier:1 Kind:2
FUERA: dyn-var: 0 nthreads-var:3 thread-limit-var:2147483647 run-sched-var: Modifier:1 Kind:2
Fuera de 'parallel for' suma=40

```

3:

```

sersoker@toshiba14:~/Dropbox/Universidad/AC/Practicas/Bloque3$ export OMP_SCHEDULE="static,4"
sersoker@toshiba14:~/Dropbox/Universidad/AC/Practicas/Bloque3$ ./3 10 2
thread 0 suma a[0]=0 suma=0
thread 0 suma a[1]=1 suma=1
thread 0 suma a[6]=6 suma=7
thread 0 suma a[7]=7 suma=14
thread 0 suma a[8]=8 suma=22
thread 0 suma a[9]=9 suma=31
thread 2 suma a[2]=2 suma=2
thread 2 suma a[3]=3 suma=5
thread 1 suma a[4]=4 suma=4
thread 1 suma a[5]=5 suma=9
DENTRO: dyn-var: 0 nthreads-var:3 thread-limit-var:2147483647 run-sched-var: Modifier:1 Kind:2
FUERA: dyn-var: 0 nthreads-var:3 thread-limit-var:2147483647 run-sched-var: Modifier:1 Kind:2
Fuera de 'parallel for' suma=31
sersoker@toshiba14:~/Dropbox/Universidad/AC/Practicas/Bloque3$ export OMP_SCHEDULE="static,5"
sersoker@toshiba14:~/Dropbox/Universidad/AC/Practicas/Bloque3$ ./3 10 2
thread 0 suma a[0]=0 suma=0
thread 0 suma a[1]=1 suma=1
thread 0 suma a[6]=6 suma=7
thread 0 suma a[7]=7 suma=14
thread 0 suma a[8]=8 suma=22
thread 0 suma a[9]=9 suma=31
thread 1 suma a[4]=4 suma=4
thread 1 suma a[5]=5 suma=9
thread 2 suma a[2]=2 suma=2
thread 2 suma a[3]=3 suma=5
DENTRO: dyn-var: 0 nthreads-var:3 thread-limit-var:2147483647 run-sched-var: Modifier:1 Kind:2
FUERA: dyn-var: 0 nthreads-var:3 thread-limit-var:2147483647 run-sched-var: Modifier:1 Kind:2
Fuera de 'parallel for' suma=31

```

RESPUESTA: Si se modifican las variables de entorno por ejemplo el `num_threads` se ve claramente reflejado en las ejecuciones, ya que la variable `nthreads-var` se modifica. En la segunda captura se puede ver como la variable `dyn_var` también se modifica, cosa que no ocurre con la variable `run_sched_var` que al modificarse en el propio código tiene prioridad sobre lo que se especifica modificando las variables fuera.

4. Usar en el ejemplo anterior las funciones `omp_get_num_threads()`, `omp_get_num_procs()` y `omp_in_parallel()` dentro y fuera de la región paralela. Imprimir los valores que obtienen estas funciones dentro (lo debe imprimir sólo uno de los threads) y fuera de la región paralela. Incorporar en su cuaderno de prácticas volcados de pantalla con los resultados de ejecución obtenidos. Indicar en qué funciones se obtienen valores distintos dentro y fuera de la región paralela.

CÓDIGO FUENTE: `scheduled-clauseModificado4.c`

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

main(int argc, char **argv) {
    int i, n=200, chunk, a[n], suma=0;
    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 schedule(dynamic, chunk) lastprivate(suma)
        firstprivate(suma)
            for (i=0; i<n; i++)
            { suma = suma + a[i];
              printf(" thread %d suma a[%d]=%d suma=%d \n",
                    omp_get_thread_num(), i, a[i], suma);
            }

        #pragma omp single
        {
            printf("DENTRO: num_threads: %d num_procs:%d in_parallel:%d\n",
                  omp_get_num_threads(), omp_get_num_procs(), omp_in_parallel());
        }
        printf("FUERA: num_threads: %d num_procs:%d in_parallel:%d\n",
              omp_get_num_threads(), omp_get_num_procs(), omp_in_parallel());
        printf("Fuera de 'parallel for' suma=%d\n", suma);
    }
}
```

CAPTURAS DE PANTALLA:

```

sersoker@toshiba14:~/Dropbox/Universidad/AC/Practicas/Bloque3$ gcc -fopenmp sche
dled-clauseModificado4.c -o 4
sersoker@toshiba14:~/Dropbox/Universidad/AC/Practicas/Bloque3$ ./4 10 3
thread 2 suma a[3]=3 suma=3
thread 2 suma a[4]=4 suma=7
thread 2 suma a[5]=5 suma=12
thread 2 suma a[9]=9 suma=21
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 0 suma a[6]=6 suma=6
thread 0 suma a[7]=7 suma=13
thread 0 suma a[8]=8 suma=21
DENTRO: num_threads: 3 num_procs:2 in_parallel:1
FUERA: num_threads: 1 num_procs:2 in_parallel:0
Fuera de 'parallel for' suma=21

```


RESPUESTA: Los valores de dentro y fuera son distintos, menos el num_procs que es la cantidad de procesadores disponible, esto no se modifica, ya que es algo físico del equipo.

In_parallel indica si estamos o no en una región paralelizable, por lo que fuera y dentro tiene valores true o false (1 y 0)

Con respecto a omp_get_num_threads(), obtiene el nº de threads que se están usando en una región Paralela, en caso de encontrarse en código secuencial devuelve 1.

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

CÓDIGO FUENTE: scheduled-clauseModificado5.c

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

main(int argc, char **argv) {
    int i, n=200, chunk, a[n], suma=0;
    int modifier=0, modifier2=0;
    omp_sched_t kind, kind2;

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

        #pragma omp single
        {
            omp_get_schedule(&kind, &modifier);
            printf("Antes: dyn-var: %d  nthreads-var:%d run-sched-var: Modifier:%d
Kind:%d\n",
              omp_get_dynamic(), omp_get_max_threads(), modifier, kind);

            omp_set_dynamic(1);
            omp_set_num_threads(15);
            omp_set_schedule(omp_sched_auto, 0);

            omp_get_schedule(&kind2, &modifier2);
            printf("Despues: dyn-var: %d  nthreads-var:%d run-sched-var: Modifier:%d
Kind:%d\n",
              omp_get_dynamic(), omp_get_max_threads(), modifier2, kind2);
        }
        printf("Fuera de 'parallel for' suma=%d\n", suma);
    }
}
```

CAPTURAS DE PANTALLA:

```

sersoker@toshiba14:~/Dropbox/Universidad/AC/Practicas/Bloque3$ gcc -fopenmp scheduled-clauseModificado5.c -o 5
sersoker@toshiba14:~/Dropbox/Universidad/AC/Practicas/Bloque3$ ./5 10 3
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[9]=9 suma=12
thread 2 suma a[3]=3 suma=3
thread 2 suma a[4]=4 suma=7
thread 2 suma a[5]=5 suma=12
thread 1 suma a[6]=6 suma=6
thread 1 suma a[7]=7 suma=13
thread 1 suma a[8]=8 suma=21
Antes: dyn-var: 0 nthreads-var:3 run-sched-var: Modifier:1 Kind:2
Despues: dyn-var: 1 nthreads-var:15 run-sched-var: Modifier:1 Kind:4
Fuera de 'parallel for' suma=12

```

RESPUESTA: Como se puede ver en las capturas, los valores se modifican correctamente con las llamadas correctas, ignorando los que tiene pro defecto el sistema ya que tienen menor prioridad que los de dentro del programa.

6. Implementar un programa secuencial en C que multiplique una matriz triangular por un vector.

NOTAS: (1) el número de filas/columnas debe ser un argumento de entrada; (2) se debe inicializar las matrices antes del cálculo; (3) se debe imprimir siempre la primera y última componente del resultado antes de que termine el programa.

CÓDIGO FUENTE: pmtv-secuencial.c

```

#include <stdio.h>
#include <stdlib.h>
#ifdef _OPENMP
#include <omp.h>
#else
#define omp_get_thread_num() 0
#endif
#include <time.h> // biblioteca donde se encuentra la función clock_gettime()

main(int argc, char **argv){
    struct timespec cgt1,cgt2; double ncgt; //para tiempo de ejecución

    if(argc < 2) {
        fprintf(stderr,"Falta numero de filas/columnas N de la matriz.\n");
        exit(-1);
    }
    int i,j,k,n;
    n = atoi(argv[1]);
    double v1[n], v2[n],M[n][n];
    double suma=0;

    // Inicio vector 1 v1[i]=i
    for (i=0; i<n; i++) v1[i] = i;
    // Inicio vector 2 a 0
    for (i=0; i<n; i++) v2[i] = 0;
    // Inicio Matriz M m[i][j]=i+j
    for (i=0; i<n; i++)
        for (j=0; j<n; j++)
            if(i>=j)
                M[i][j]=i+j;
            else

```

```

        M[i][j]=0;

clock_gettime(CLOCK_REALTIME,&cgt1);
    for (i=0; i<n; i++){
        for (k=0;k<=i;k++)
            v2[i] += M[i][k]*v1[k];
    }
clock_gettime(CLOCK_REALTIME,&cgt2);
    ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+(double) ((cgt2.tv_nsec-
cgt1.tv_nsec)/(1.e+9));

printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\n",ncgt,n);
printf("Valor v2[0]= %f v2[n-1]=%f\n",v2[0],v2[n-1]);

    printf("Vector resultante tras calculo:");
    for (i=0; i<n; i++)
        printf(" %f ", v2[i]);
        printf("\n");

}

```

CAPTURAS DE PANTALLA:

```

sersoker@toshiba14:~/Dropbox/Universidad/AC/Practicas/Bloque3$ gcc -fopenmp pmtv
-secuencial.c -o s
sersoker@toshiba14:~/Dropbox/Universidad/AC/Practicas/Bloque3$ ./s 2
Tiempo(seg.):0.000001956 / Tamaño Vectores:2
Valor v2[0]= 0.000000 v2[n-1]=2.000000
Vector resultante tras calculo: 0.000000 2.000000
sersoker@toshiba14:~/Dropbox/Universidad/AC/Practicas/Bloque3$ ./s 3
Tiempo(seg.):0.000002445 / Tamaño Vectores:3
Valor v2[0]= 0.000000 v2[n-1]=11.000000
Vector resultante tras calculo: 0.000000 2.000000 11.000000
sersoker@toshiba14:~/Dropbox/Universidad/AC/Practicas/Bloque3$ ./s 4
Tiempo(seg.):0.000002934 / Tamaño Vectores:4
Valor v2[0]= 0.000000 v2[n-1]=32.000000
Vector resultante tras calculo: 0.000000 2.000000 11.000000 32.000000

```

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. 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 que multiplica una matriz triangular por un vector con las alternativas de planificación `static`, `dynamic` y `guided` para `chunk` de 2, 64, 128, 1024 y el `chunk` por defecto para la alternativa. No use vectores mayores de 32768 componentes ni menores de 4096 componentes. El número de threads en las ejecuciones debe coincidir con el número de cores. Rellenar la Ocon los tiempos obtenidos, ponga en la tabla el número de threads que utilizan las ejecuciones. Representar el tiempo para `static`, `dynamic` y `guided` en función del tamaño del `chunk` en una gráfica. Rellenar la tabla y realizar la gráfica también para el PC local. ¿Qué alternativa ofrece mejores prestaciones? Razone por qué.

RESPUESTA: Se puede ver que en un entorno local con 2 hebras, la mejor alternativa de principio a fin (independientemente del `chunk`) sería la `guided`, no obstante no supone una gran diferencia con respecto a las otras una vez se aumenta el `chunk`, siendo el cambio casi imperceptible.

Si hablamos de ATCgrid se puede ver como `Static` y `Guided` empiezan muy bien para `chunk` por defecto, manteniéndose para `guided` y con mucha variación en `static` dependiendo del `chunk`. El modo `Dynamic` empieza algo peor que los otros dos pero se mantiene independientemente del `chunk`, siendo el que más rápido trabaja en números altos del mismo.

Comparando los dos se puede ver que para pocas hebras de ejecución `Guided` podría ser la forma mas eficiente, ya que casi no sufre variaciones por el `chunk`, y en entornos con varias hebras `Dynamic` es el que ofrece resultados con menos variación, siendo no obstante peor en algunos casos como `chunk` pequeños o intermedios (2 y 32), y mejorando respecto a los demás conforme este parámetro se hace mayor..

CÓDIGO FUENTE: `pmtv-OpenMP.c`

```
#include <stdio.h>
#include <stdlib.h>
#ifdef _OPENMP
#include <omp.h>
#else
#define omp_get_thread_num() 0
#endif
#include <time.h> // biblioteca donde se encuentra la función clock_gettime()

int v1[5000], v2[5000], M[5000][5000];

main(int argc, char **argv){
    struct timespec cgt1, cgt2; double ncgt; //para tiempo de ejecución

    int i, j, k, n;
    n=5000;
    int sumalocal=0;

    // Inicio vector 1 v1[i]=i
    #pragma omp parallel for default(none) private(i) shared(v1, n)
    for (i=0; i<n; i++) v1[i] = i;
    // Inicio vector 2 a 0
    #pragma omp parallel for default(none) private(i) shared(v2, n)
    for (i=0; i<n; i++) v2[i] = 0;
    // Inicio Matriz M m[i][j]=i+j
    #pragma omp parallel for default(none) private(i, j) shared(M, n)
    for (i=0; i<n; i++)
        for (j=0; j<n; j++)
```

```

        if(i>=j)
            M[i][j]=i+j;
        else
            M[i][j]=0;

clock_gettime(CLOCK_REALTIME,&cgt1);
#pragma omp parallel for default(none) private(i,k) shared(M,v1,v2,n) schedule(runtime)
    for (i=0; i<n; i++)
    {
        for (k=0;k<=i;k++)
            v2[i] += M[i][k]*v1[k];
    }

clock_gettime(CLOCK_REALTIME,&cgt2);
    ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+(double) ((cgt2.tv_nsec-
cgt1.tv_nsec)/(1.e+9));

printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%d\n",ncgt,n);
printf("Valor v2[0]= %d v2[n-1]=%d\n",v2[0],v2[n-1]);

    /*printf("Vector resultante tras calculo:");
    for (i=0; i<n; i++)
        printf(" %f ", v2[i]);
        printf("\n")*/;

    // 2= 1 2
    // 3= 5 - 11
    // 4= 14 - - 32
}

```

CAPTURAS DE PANTALLA:**STATIC**

```

sersoker@toshiba14:~/Dropbox/Universidad/AC/Practicas/Bloque3$ ./7
Tiempo(seg.):0.118467570 / Tamaño Vectores:5000
Valor v2[0]= 0 v2[n-1]=1049954896
sersoker@toshiba14:~/Dropbox/Universidad/AC/Practicas/Bloque3$ export OMP_SCHEDULE="static,2"
sersoker@toshiba14:~/Dropbox/Universidad/AC/Practicas/Bloque3$ ./7
Tiempo(seg.):0.122596238 / Tamaño Vectores:5000
Valor v2[0]= 0 v2[n-1]=1049954896
sersoker@toshiba14:~/Dropbox/Universidad/AC/Practicas/Bloque3$ export OMP_SCHEDULE="static,32"
sersoker@toshiba14:~/Dropbox/Universidad/AC/Practicas/Bloque3$ ./7
Tiempo(seg.):0.093263879 / Tamaño Vectores:5000
Valor v2[0]= 0 v2[n-1]=1049954896
sersoker@toshiba14:~/Dropbox/Universidad/AC/Practicas/Bloque3$ export OMP_SCHEDULE="static,64"
sersoker@toshiba14:~/Dropbox/Universidad/AC/Practicas/Bloque3$ ./7
Tiempo(seg.):0.093504901 / Tamaño Vectores:5000
Valor v2[0]= 0 v2[n-1]=1049954896
sersoker@toshiba14:~/Dropbox/Universidad/AC/Practicas/Bloque3$ export OMP_SCHEDULE="static,1024"
sersoker@toshiba14:~/Dropbox/Universidad/AC/Practicas/Bloque3$ ./7
Tiempo(seg.):0.108767036 / Tamaño Vectores:5000
Valor v2[0]= 0 v2[n-1]=1049954896

```

DYNAMIC

```

sersoker@toshiba14:~/Dropbox/Universidad/AC/Practicas/Bloque3$ ./7
Tiempo(seg.):0.223967361 / Tamaño Vectores:5000
Valor v2[0]= 0 v2[n-1]=1049954896
sersoker@toshiba14:~/Dropbox/Universidad/AC/Practicas/Bloque3$ export OMP_SCHEDULE="dynamic,2"
sersoker@toshiba14:~/Dropbox/Universidad/AC/Practicas/Bloque3$ ./7
Tiempo(seg.):0.208795182 / Tamaño Vectores:5000
Valor v2[0]= 0 v2[n-1]=1049954896
sersoker@toshiba14:~/Dropbox/Universidad/AC/Practicas/Bloque3$ export OMP_SCHEDULE="dynamic,32"
sersoker@toshiba14:~/Dropbox/Universidad/AC/Practicas/Bloque3$ ./7
Tiempo(seg.):0.094489523 / Tamaño Vectores:5000
Valor v2[0]= 0 v2[n-1]=1049954896
sersoker@toshiba14:~/Dropbox/Universidad/AC/Practicas/Bloque3$ export OMP_SCHEDULE="dynamic,64"
sersoker@toshiba14:~/Dropbox/Universidad/AC/Practicas/Bloque3$ ./7
Tiempo(seg.):0.093513212 / Tamaño Vectores:5000
Valor v2[0]= 0 v2[n-1]=1049954896
sersoker@toshiba14:~/Dropbox/Universidad/AC/Practicas/Bloque3$ export OMP_SCHEDULE="dynamic,1024"
sersoker@toshiba14:~/Dropbox/Universidad/AC/Practicas/Bloque3$ ./7
Tiempo(seg.):0.109555614 / Tamaño Vectores:5000
Valor v2[0]= 0 v2[n-1]=1049954896

```

GUIDED

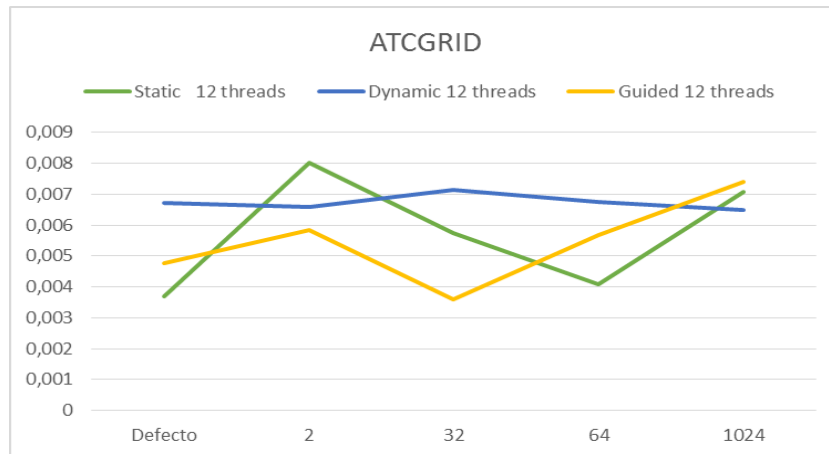
```

sersoker@toshiba14:~/Dropbox/Universidad/AC/Practicas/Bloque3$ ./7
Tiempo(seg.):0.094476323 / Tamaño Vectores:5000
Valor v2[0]= 0 v2[n-1]=1049954896
sersoker@toshiba14:~/Dropbox/Universidad/AC/Practicas/Bloque3$ export OMP_SCHEDULE="guided,2"
sersoker@toshiba14:~/Dropbox/Universidad/AC/Practicas/Bloque3$ ./7
Tiempo(seg.):0.094012367 / Tamaño Vectores:5000
Valor v2[0]= 0 v2[n-1]=1049954896
sersoker@toshiba14:~/Dropbox/Universidad/AC/Practicas/Bloque3$ export OMP_SCHEDULE="guided,32"
sersoker@toshiba14:~/Dropbox/Universidad/AC/Practicas/Bloque3$ ./7
Tiempo(seg.):0.094004057 / Tamaño Vectores:5000
Valor v2[0]= 0 v2[n-1]=1049954896
sersoker@toshiba14:~/Dropbox/Universidad/AC/Practicas/Bloque3$ export OMP_SCHEDULE="guided,64"
sersoker@toshiba14:~/Dropbox/Universidad/AC/Practicas/Bloque3$ ./7
Tiempo(seg.):0.093865212 / Tamaño Vectores:5000
Valor v2[0]= 0 v2[n-1]=1049954896
sersoker@toshiba14:~/Dropbox/Universidad/AC/Practicas/Bloque3$ export OMP_SCHEDULE="guided,1024"
sersoker@toshiba14:~/Dropbox/Universidad/AC/Practicas/Bloque3$ ./7
Tiempo(seg.):0.106818324 / Tamaño Vectores:5000
Valor v2[0]= 0 v2[n-1]=1049954896

```

TABLA RESULTADOS Y GRÁFICA ATCGRID:**Tabla1.-** Tiempos de ejecución de la versión paralela del producto de una matriz triangular por un vector

Chunk	Static 12 threads	Dynamic 12 threads	Guided 12 threads
Defecto	0,003709411	0,006717054	0,004766628
2	0,008013167	0,006575531	0,005831035
32	0,005754961	0,007144432	0,003591669
64	0,004092928	0,006762006	0,005691277
1024	0,007083256	0,006483235	0,007416602

**TABLA RESULTADOS Y GRÁFICA PC LOCAL:****Tabla2.-** Tiempos de ejecución de la versión paralela del producto de una matriz triangular por un vector

Chunk	Static 2 threads	Dynamic 2 threads	Guided 2 threads
Defecto	0,11846757	0,223967361	0,094476323
2	0,122596238	0,208795182	0,094012367
32	0,093263879	0,094489523	0,094004057
64	0,093504901	0,093513212	0,093865212
1024	0,108767036	0,109555614	0,105620041

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

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

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

CÓDIGO FUENTE: pmm-secuencial.c

```
#include <stdio.h>
#include <stdlib.h>
#ifdef _OPENMP
#include <omp.h>
#else
#define omp_get_thread_num() 0
#endif
#include <time.h> // biblioteca donde se encuentra la función clock_gettime()

main(int argc, char **argv){
    struct timespec cgt1,cgt2; double ncgt; //para tiempo de ejecución

    if(argc < 2) {
        fprintf(stderr,"Falta numero de filas/columnas N de la matriz.\n");
        exit(-1);
    }
    int i,j,k,n,x;
    n = atoi(argv[1]);

    int **M1,**M2,**M3;
    M1 = (int **)malloc (n*sizeof(int *));
    M2 = (int **)malloc (n*sizeof(int *));
    M3 = (int **)malloc (n*sizeof(int *));
    for (i=0;i<n;i++){
        M1[i] = (int *) malloc (n*sizeof(int));
        M2[i] = (int *) malloc (n*sizeof(int));
        M3[i] = (int *) malloc (n*sizeof(int));
    }

    // Inicio Matriz M1 m[i][j]=i+j
    for (i=0; i<n; i++)
        for (j=0; j<n; j++)
            M1[i][j]=1;
    // Inicio Matriz M2 m[i][j]=i+j
    for (i=0; i<n; i++)
        for (j=0; j<n; j++)
            M2[i][j]=1;
    // Inicio Matriz M3 m[i][j]=i+j
    for (i=0; i<n; i++)
        for (j=0; j<n; j++)
            M3[i][j]=0 ;

    clock_gettime(CLOCK_REALTIME,&cgt1);
    for (i=0; i<n; i++){
        for (j=0;j<n;j++){
            for (x=0;x<n;x++){

                M3[i][j]=M3[i][j]+(M1[i][x]*M2[x][j]);
            }
        }
    }
    clock_gettime(CLOCK_REALTIME,&cgt2);
    ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+(double) ((cgt2.tv_nsec-
cgt1.tv_nsec)/(1.e+9));

    printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\n",ncgt,n);
    printf("Valor M[0][0]= %f M[n-1][n-1]=%f\n",M3[0][0],M3[n-1][n-1]);
}
```


CAPTURAS DE PANTALLA:

```
sersoker@toshiba14:~/Dropbox/Universidad/AC/Practicas/Bloque3$ gcc -fopenmp pmn-  
secuencial.c -o 8  
sersoker@toshiba14:~/Dropbox/Universidad/AC/Practicas/Bloque3$ ./8 1  
Tiempo(seg.):0.000002444 / Tamaño Vectores:1  
Valor M[0][0]= 1.000000 M[n-1][n-1]=1.000000  
sersoker@toshiba14:~/Dropbox/Universidad/AC/Practicas/Bloque3$ ./8 2  
Tiempo(seg.):0.000002444 / Tamaño Vectores:2  
Valor M[0][0]= 2.000000 M[n-1][n-1]=2.000000  
sersoker@toshiba14:~/Dropbox/Universidad/AC/Practicas/Bloque3$ ./8 3  
Tiempo(seg.):0.000003911 / Tamaño Vectores:3  
Valor M[0][0]= 3.000000 M[n-1][n-1]=3.000000
```

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).

CÓDIGO FUENTE: pmm-OpenMP.c

```

#include <stdio.h>
#include <stdlib.h>
#ifdef _OPENMP
#include <omp.h>
#else
#define omp_get_thread_num() 0
#endif
#include <time.h> // biblioteca donde se encuentra la función clock_gettime()

int v1[5000], v2[5000], M[5000][5000];

main(int argc, char **argv){
struct timespec cgt1,cgt2; double ncgt; //para tiempo de ejecución

int i,j,k,n;
n=5000;
int sumalocal=0;

// Inicio vector 1 v1[i]=i
#pragma omp parallel for default(none) private(i) shared(v1,n)
for (i=0; i<n; i++) v1[i] = i;
// Inicio vector 2 a 0
#pragma omp parallel for default(none) private(i) shared(v2,n)
for (i=0; i<n; i++) v2[i] = 0;
// Inicio Matriz M m[i][j]=i+j
#pragma omp parallel for default(none) private(i,j) shared(M,n)
for (i=0; i<n; i++)
    for (j=0; j<n; j++)
        if(i>=j)
            M[i][j]=i+j;
        else
            M[i][j]=0;

clock_gettime(CLOCK_REALTIME,&cgt1);
#pragma omp parallel for default(none) private(i,k) shared(M,v1,v2,n) schedule(runtime)
for (i=0; i<n; i++)
{
    for (k=0; k<n; k++)
        if(i>=k)
            v2[i] += M[i][k]*v1[k];
}

clock_gettime(CLOCK_REALTIME,&cgt2);
ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+(double) ((cgt2.tv_nsec-
cgt1.tv_nsec)/(1.e+9));

printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%d\n",ncgt,n);
printf("Valor v2[0]= %d v2[n-1]=%d\n",v2[0],v2[n-1]);

}

```

CAPTURAS DE PANTALLA:

```

sersoker@toshiba14:~/Dropbox/Universidad/AC/Practicas/Bloque3$ gcc -fopenmp pmn-
OpenMP.c -o 9
sersoker@toshiba14:~/Dropbox/Universidad/AC/Practicas/Bloque3$ ./9 1
Tiempo(seg.):0.000106578          / Tamaño Vectores:1
Valor M[0][0]= 1 M[n-1][n-1]=1
sersoker@toshiba14:~/Dropbox/Universidad/AC/Practicas/Bloque3$ ./9 5
Tiempo(seg.):0.0004181479        / Tamaño Vectores:5
Valor M[0][0]= 5 M[n-1][n-1]=5
sersoker@toshiba14:~/Dropbox/Universidad/AC/Practicas/Bloque3$ ./9 10
Tiempo(seg.):0.000154489         / Tamaño Vectores:10
Valor M[0][0]= 10 M[n-1][n-1]=10
sersoker@toshiba14:~/Dropbox/Universidad/AC/Practicas/Bloque3$ ./9 20
Tiempo(seg.):0.000587643         / Tamaño Vectores:20
Valor M[0][0]= 20 M[n-1][n-1]=20
sersoker@toshiba14:~/Dropbox/Universidad/AC/Practicas/Bloque3$ ./9 30
Tiempo(seg.):0.001212930         / Tamaño Vectores:30
Valor M[0][0]= 30 M[n-1][n-1]=30

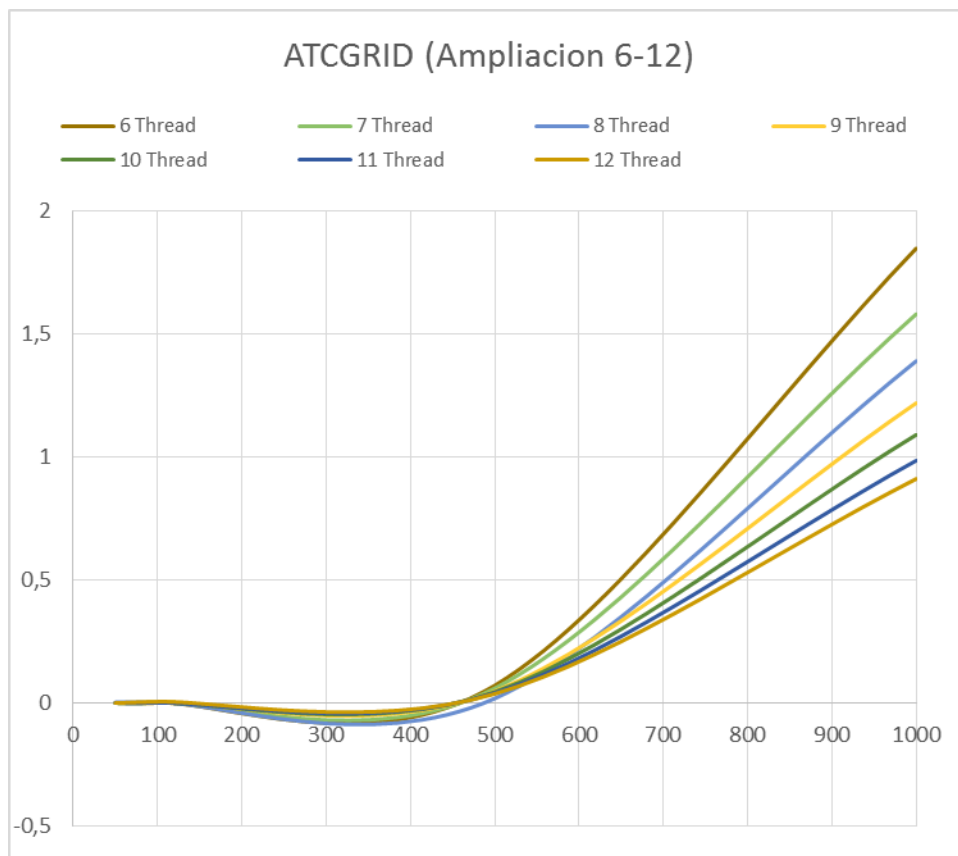
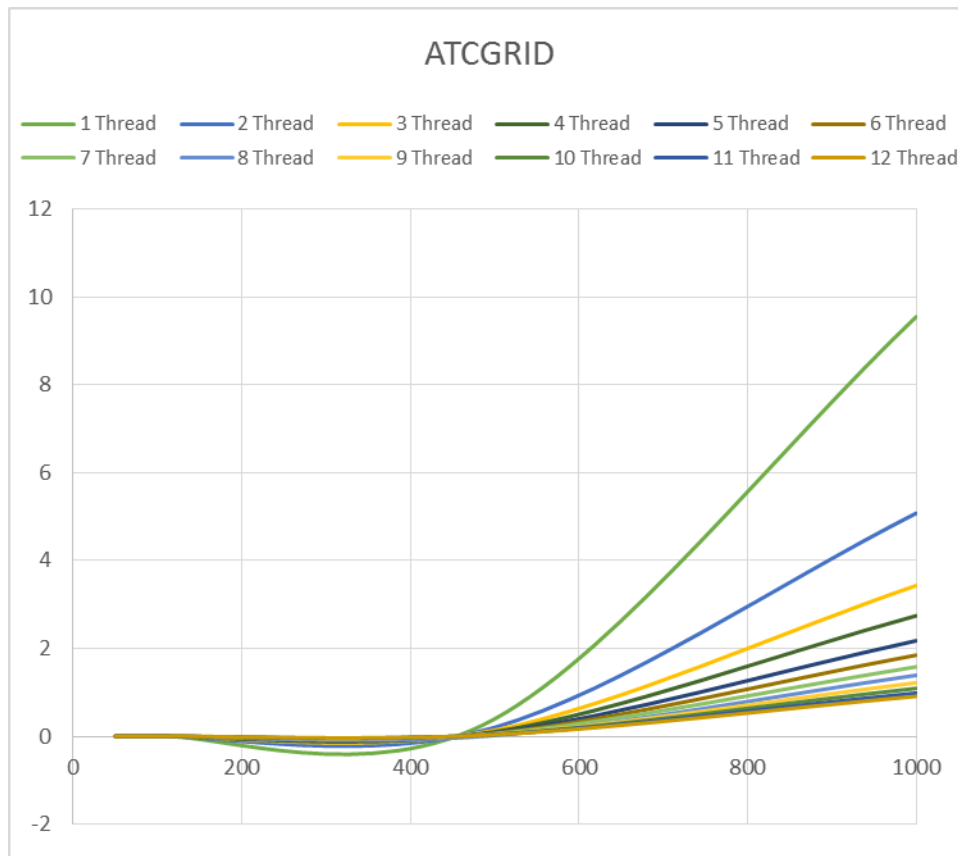
```

10. Hacer un estudio de escalabilidad (ganancia en velocidad en función del número de cores) en atcgrid y en el PC local del código paralelo implementado para tres tamaños de las matrices. Debe recordar usar `-O2` al compilar. Presente los resultados del estudio en tablas de valores y en gráficas. Escoger los tamaños de manera que se observe diferentes curvas de escalabilidad en las gráficas que entregue en su cuaderno de prácticas. Consulte la Lección 6/Tema 2.

ESTUDIO ESCALABILIDAD ATCGRID:

Tamaño	1 Thread	2 Thread	3 Thread	4 Thread	5 Thread	6 Thread
50	0,00025298	0,00041167	0,00051387	0,0026669	0,0006437	0,00076655
100	0,00277131	0,00167244	0,00142546	0,0014402	0,00135914	0,00149551
500	0,39812708	0,20233894	0,13937127	0,10560677	0,08509395	0,07200383
1000	9,55266783	5,08005999	3,43418333	2,7457367	2,17756467	1,84891016

Tamaño	7 Thread	8 Thread	9 Thread	10 Thread	11 Thread	12 Thread
50	0,001126	0,00337971	0,00081752	0,00062685	0,00087702	0,00074676
100	0,00443417	0,00432031	0,00501368	0,00159749	0,00156186	0,00577718
500	0,06067883	0,01937354	0,04931762	0,04588527	0,04155439	0,03802308
1000	1,58193494	1,3911492	1,22022593	1,09132771	0,98642747	0,91205335



ESTUDIO ESCALABILIDAD PC LOCAL:

LOCAL

<i>Tamaño</i>	1 núcleo	2 nucleos
50	0,00149037	0,00116249
100	0,011296	0,01266029
500	1,66329899	0,65903631
1000	17,3968962	9,77387722

