

2º curso / 2º cuatr.  
Grado Ing. Inform.  
Doble Grado Ing.  
Inform. y Mat.

## Arquitectura de Computadores (AC)

### Cuaderno de prácticas.

### Bloque Práctico 3. Programación paralela III: Interacción con el entorno en OpenMP

Estudiante (nombre y apellidos): Daniel Bolaños Martínez

Grupo de prácticas: A1

Fecha de entrega: 11/05/17

Fecha evaluación en clase: 12/05/17

#### Ejercicios basados en los ejemplos del seminario práctico

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

**CÓDIGO FUENTE:** `if-clauseModificado.c`

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

int main(int argc, char **argv)
{
    int i, n=20, x, tid;
    int a[n], suma=0, sumalocal;
    if(argc < 3){
        fprintf(stderr, "[ERROR]-Falta iteraciones y nºthreads\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;
    }

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

```

danibolanos@Aspire-E5-575G:~/Escritorio/Codigo_AC_3$ gcc -O2 -fopenmp -o if-clauseModificado if-clauseModificado.c
danibolanos@Aspire-E5-575G:~/Escritorio/Codigo_AC_3$ ./if-clauseModificado 4 3
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
danibolanos@Aspire-E5-575G:~/Escritorio/Codigo_AC_3$ ./if-clauseModificado 5 4
thread 0 suma de a[0]=0 sumalocal=0
thread 0 suma de a[1]=1 sumalocal=1
thread 3 suma de a[4]=4 sumalocal=4
thread 1 suma de a[2]=2 sumalocal=2
thread 2 suma de a[3]=3 sumalocal=3
thread master=0 imprime suma=10
danibolanos@Aspire-E5-575G:~/Escritorio/Codigo_AC_3$ ./if-clauseModificado 6 5
thread 1 suma de a[2]=2 sumalocal=2
thread 0 suma de a[0]=0 sumalocal=0
thread 0 suma de a[1]=1 sumalocal=1
thread 3 suma de a[4]=4 sumalocal=4
thread 2 suma de a[3]=3 sumalocal=3
thread 4 suma de a[5]=5 sumalocal=5
thread master=0 imprime suma=15

```

### RESPUESTA:

La cláusula `num_threads` solo fijará el número de threads que se utilizarán en paralelo cuando el valor de iteraciones sea mayor que 4, ya que el `if`, hace que para valores de `n` menores de 4 el código se ejecute secuencialmente.

Como podemos ver en la imagen, para valores de `n= 5` o `6` podemos ver que el segundo parámetro pasado como argumento es el que fija el número de threads creando en cada caso 4 y 5 threads respectivamente.

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

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

### RESPUESTA:

En el `schedule static`, las tareas se asignan utilizando Round-Robin tal y como se puede observar en las tablas en tiempo de compilación, por lo que podemos saber cómo se repartirán las tareas entre las `threads`.

Los casos de `dynamic` y `guided` asignan las `threads` en tiempo de ejecución (`threads` más rápidos ejecutan más iteraciones) por lo que no podemos saber la asignación de iteraciones que recibirá cada `thread`. Además `guided` tiene la característica de que mengua el tamaño de bloque en relación a las iteraciones restantes y del número de `threads`.

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

int 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 single
        {
            printf("Dentro del parallel\n");
            printf("dyn-var: %d \n", omp_get_dynamic());
            printf("nthreads-var: %d \n", omp_get_max_threads());
        }
    }
}
```

```

printf("thread-limit-var: %d \n",omp_get_thread_limit());
omp_get_schedule(&kind,&modifier);
printf("run-sched-var: Kind-> %d, Modifier-> %d \n",kind,modifier);
}
#pragma 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);
}
}
}

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

```

### CAPTURAS DE PANTALLA:

```

danibolanos@Aspire-E5-575G:~/Escritorio/Codigo_AC_3$ export OMP_NUM_THR
EADS=2
danibolanos@Aspire-E5-575G:~/Escritorio/Codigo_AC_3$ export OMP_SCHEDUL
E="static,4"
danibolanos@Aspire-E5-575G:~/Escritorio/Codigo_AC_3$ export OMP_DYNAMIC
=FALSE
danibolanos@Aspire-E5-575G:~/Escritorio/Codigo_AC_3$ ./scheduled-clause
Modificado 3 3
Dentro del parallel
dyn-var: 0
nthreads-var: 2
thread-limit-var: 2147483647
run-sched-var: Kind-> 1, Modifier-> 4
  thread 0 suma a[0]=0 suma=0
  thread 0 suma a[1]=1 suma=1
  thread 0 suma a[2]=2 suma=3
  thread 1 suma a[0]=0 suma=0
Fuera de 'parallel for' suma=3
Fuera del parallel
dyn-var: 0
nthreads-var: 2
thread-limit-var: 2147483647
run-sched-var: Kind-> 1, Modifier-> 4

```

```

danibolanos@Aspire-E5-575G:~/Escritorio/Codigo_AC_3$ export OMP_NUM_THR
EADS=4
danibolanos@Aspire-E5-575G:~/Escritorio/Codigo_AC_3$ export OMP_SCHEDUL
E="dynamic,3"
danibolanos@Aspire-E5-575G:~/Escritorio/Codigo_AC_3$ export OMP_DYNAMIC
=TRUE
danibolanos@Aspire-E5-575G:~/Escritorio/Codigo_AC_3$ ./scheduled-clause
Modificado 4 4
Dentro del parallel
dyn-var: 1
nthreads-var: 4
thread-limit-var: 2147483647
run-sched-var: Kind-> 2, Modifier-> 3
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 2 suma a[0]=0 suma=0
thread 0 suma a[0]=0 suma=0
thread 0 suma a[1]=1 suma=7
thread 0 suma a[2]=2 suma=9
thread 0 suma a[3]=3 suma=12
thread 3 suma a[0]=0 suma=6
Fuera de 'parallel for' suma=12
Fuera del parallel
dyn-var: 1
nthreads-var: 4
thread-limit-var: 2147483647
run-sched-var: Kind-> 2, Modifier-> 3

```

**RESPUESTA:**

No. Los valores tanto dentro como fuera de la región paralela son equivalentes.

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

int 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 single
    {
        printf("Dentro del parallel\n");
        printf("omp_get_num_threads: %d \n",omp_get_num_threads());
        printf("omp_get_num_procs: %d \n",omp_get_num_procs());
        printf("omp_in_parallel: %d \n",omp_in_parallel());
    }
    #pragma 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);
        }
    }
}

printf("Fuera de 'parallel for' suma=%d\n",suma);
printf("Fuera del parallel\n");
printf("omp_get_num_threads: %d \n",omp_get_num_threads());
printf("omp_get_num_procs: %d \n",omp_get_num_procs());
printf("omp_in_parallel: %d \n",omp_in_parallel());
}

```

**CAPTURAS DE PANTALLA:**

```

danibolanos@Aspire-E5-575G:~/Escritorio/Codigo_AC_3$ export OMP_NUM_THR
EADS=2
danibolanos@Aspire-E5-575G:~/Escritorio/Codigo_AC_3$ ./scheduled-clause
Modificado4 5 3
Dentro del parallel
omp_get_num_threads: 2
omp_get_num_procs: 4
omp_in_parallel: 1
 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 1 suma a[4]=4 suma=10
 thread 0 suma a[0]=0 suma=0
Fuera de 'parallel for' suma=10
Fuera del parallel
omp_get_num_threads: 1
omp_get_num_procs: 4
omp_in_parallel: 0

```

```

dantibolanos@Aspire-E5-575G:~/Escritorio/Codigo_AC_3$ export OMP_NUM_THREADS=4
dantibolanos@Aspire-E5-575G:~/Escritorio/Codigo_AC_3$ ./scheduled-clause
Modificado4 4 2
Dentro del parallel
omp_get_num_threads: 4
omp_get_num_procs: 4
omp_in_parallel: 1
thread 3 suma a[0]=0 suma=0
thread 3 suma a[1]=1 suma=1
thread 3 suma a[2]=2 suma=3
thread 3 suma a[3]=3 suma=6
thread 2 suma a[0]=0 suma=0
thread 0 suma a[0]=0 suma=6
thread 0 suma a[1]=1 suma=7
thread 0 suma a[2]=2 suma=9
thread 0 suma a[3]=3 suma=12
thread 1 suma a[0]=0 suma=0
Fuera de 'parallel for' suma=12
Fuera del parallel
omp_get_num_threads: 1
omp_get_num_procs: 4
omp_in_parallel: 0

```

**RESPUESTA:**

En la función `omp_get_num_threads` obtenemos valores distintos, ya que dentro de la región paralela se ejecutarán tantas hebras como se hayan especificado con las variables de entorno mientras que fuera del mismo al ejecutarse secuencialmente, siempre habrá una sola hebra.

En la función `omp_in_parallel` los valores también cambian pues esta función devuelve `true` si se llama dentro de una región paralela y `false` en caso contrario.

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

int 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;

    omp_set_dynamic(1);
    omp_set_num_threads(4);
    omp_set_schedule(2,3); //El 2 indica schedule_dynamic

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

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

```

**CAPTURAS DE PANTALLA:**

```

danibolanos@Aspire-E5-575G:~/Escritorio/Codigo_AC_3$ export OMP_NUM_THR
EADS=2
danibolanos@Aspire-E5-575G:~/Escritorio/Codigo_AC_3$ export OMP_DYNAMIC
=FALSE
danibolanos@Aspire-E5-575G:~/Escritorio/Codigo_AC_3$ export OMP_SCHEDUL
E="static,2"
danibolanos@Aspire-E5-575G:~/Escritorio/Codigo_AC_3$ ./scheduled-clang
Modificado5 5 3
Dentro del parallel
dyn-var: 1
nthreads-var: 4
run-sched-var: Kind-> 2, Modifier-> 3
 thread 2 suma a[0]=0 suma=0
 thread 2 suma a[1]=1 suma=1
 thread 2 suma a[2]=2 suma=3
 thread 2 suma a[3]=3 suma=6
 thread 2 suma a[4]=4 suma=10
 thread 0 suma a[0]=0 suma=0
 thread 3 suma a[0]=0 suma=0
 thread 1 suma a[0]=0 suma=0
Fuera de 'parallel for' suma=10
dyn-var: 1
nthreads-var: 4
run-sched-var: Kind-> 2, Modifier-> 3

```

**RESPUESTA:**

Como se puede observar a pesar de utilizar variables de entorno, el programa se ejecuta teniendo en cuenta los setters de las funciones `dyn-var`, `nthreads-var` y `run-sched-var`, dando como resultado los valores modificados.

Utilizamos las funciones:

`dyn-var`: `omp_set_dynamic(int dynamic_threads)`

`nthreads-var`: `omp_set_num_threads(int threads)`

`run-sched-var`: `omp_set_schedule(omp_sched_t kind, int modifier)`

## Resto de ejercicios

**6.** Implementar un programa secuencial en C que multiplique una matriz triangular por un vector (use variables dinámicas). Compare el orden de complejidad del código que ha implementado con el código que implementó para el producto matriz por vector.

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

### CÓDIGO FUENTE: `pmtv-secuencial.c`

```
/* pmtv-secuencial.c
Multiplicación de matriz por vector: v2 = T1*v1, T1 matriz triang. sup
Para compilar usar (-lrt: real time library):
gcc -O2 -fopenmp pmtv-secuencial.c -o pmtv-secuencial -lrt

Para ejecutar use: pmtv-secuencial longitud
*/

#include <stdlib.h> // biblioteca con funciones atoi(), malloc() y free()
#include <stdio.h> // biblioteca donde se encuentra la función printf()
#include <time.h> // biblioteca donde se encuentra la función clock_gettime()
#include <omp.h>

#define PRINTF_ALL // comentar para quitar el printf ...
                    // que imprime todos los componentes

//Sólo puede estar definida uno de los dos constantes PMV_ (sólo uno de
los ...
//un defines siguientes puede estar descomentado):
//#define PMV_GLOBAL // descomentar para que los vector/matriz sean variables
...
                    // globales (su longitud no estará limitada
por el ...
                    // tamaño de la pila del programa)
#define PMV_DYNAMIC // descomentar para que los vector/matriz sean
variables ...
                    // dinámicas (memoria reutilizable durante la
ejecución)
#ifdef PMV_GLOBAL
#define MAX 16350 // aprox 2^14
double T1[MAX], v1[MAX], v2[MAX];
#endif

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

```

int i, k, s;

double cgt1, cgt2, ncgt; //para tiempo de ejecución

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

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

#ifdef PMV_GLOBAL
if (N>MAX) N=MAX;
#endif
#ifdef PMV_DYNAMIC
double **T1, *v1, *v2;
T1 = (double**) malloc(N*sizeof(double*)); // malloc necesita el tamaño en
bytes
v1 = (double*) malloc(N*sizeof(double)); //si no hay espacio suficiente
malloc devuelve NULL
v2 = (double*) malloc(N*sizeof(double));
if ( (T1==NULL) || (v1==NULL) || (v2==NULL) ){
    printf("Error en la reserva de espacio para los vectores/matriz\n");
    exit(-2);
}
for(i=0; i<N;i++){
    T1[i] = (double*) malloc((N-i)*sizeof(double));
    if ( T1[i]==NULL ){
        printf("Error en la reserva de espacio para la matriz\n");
        exit(-2);
    }
}
#endif

//Inicializar Matriz Triangular Superior y vector
for(i=0; i<N; i++){
    v1[i] = N*0.1-i*0.1; //los valores dependen de N
    v2[i]=0;
}

for(i=0; i<N; i++){
    for(k=0; k<N-i; k++){
        T1[i][k] = N*0.1-i*0.1-k*0.1; //los valores dependen de N
    }
}

cgt1 = omp_get_wtime();
//Calcular producto Matriz Triangular*vector
for(i=0; i<N; i++){
    s = i;
    for(k=0; k<N-i; k++){
        v2[i] += T1[i][k] * v1[s];
        s++;
    }
}

cgt2 = omp_get_wtime();
ncgt= cgt2-cgt1;

//Imprimir resultado de la suma y el tiempo de ejecución

```

```

#ifdef PRINTF_ALL
printf("Tiempo(seg.):%11.9f\t / Tamaño Matriz-Vector:%u\n",ncgt,N);
for(i=0; i<N; i++)
    printf("V2[%d]=%8.6f\n", i, v2[i]);

#else
    printf("Tiempo(seg.):%11.9f\t / Tamaño Matriz-Vector:%u\t/ V2[0]=%8.6f / / V2[%d]=%8.6f /\n", ncgt,N, v2[0], N-1, v2[N-1]);
#endif

#ifdef PMV_DYNAMIC
for(i=0; i<N;i++)
    free(T1[i]);
free(T1); // libera el espacio reservado para T1
free(v1); // libera el espacio reservado para v1
free(v2); // libera el espacio reservado para v2
#endif
return 0;
}

```

### CAPTURAS DE PANTALLA:

```

danibolanos@Aspire-E5-575G:~/Escritorio/Codigo_AC_3$ gcc -O2 -fopenmp pmtv-secuencial.c -o pmtv-secuencial -lrt
danibolanos@Aspire-E5-575G:~/Escritorio/Codigo_AC_3$ ./pmtv-secuencial
8
Tiempo(seg.):0.000001570          / Tamaño Matriz-Vector:8
V2[0]=2.040000
V2[1]=1.400000
V2[2]=0.910000
V2[3]=0.550000
V2[4]=0.300000
V2[5]=0.140000
V2[6]=0.050000
V2[7]=0.010000
danibolanos@Aspire-E5-575G:~/Escritorio/Codigo_AC_3$ ./pmtv-secuencial
11
Tiempo(seg.):0.000001484          / Tamaño Matriz-Vector:11
V2[0]=5.060000
V2[1]=3.850000
V2[2]=2.850000
V2[3]=2.040000
V2[4]=1.400000
V2[5]=0.910000
V2[6]=0.550000
V2[7]=0.300000
V2[8]=0.140000
V2[9]=0.050000
V2[10]=0.010000

```

### (ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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:

Static es mejor para el valor por defecto de chunk y dynamic es mejor conforme el valor del chunk aumenta. Debido a que el Round-Robin utilizado por la asignación static es fácil de implementar pero a mayor número de divisiones (chunk), se hace más costoso a diferencia de los otros métodos de asignación.

a) static es 0 (reparte las tareas de forma equitativa asignando iteraciones consecutivas a cada thread), dynamic y guided es 1.

Esta información se puede ver tanto en el seminario o consultando la variable modifier de la función `omp_get_schedule(&kind, &modifier)` en el código del programa.

b) Fijemos un tamaño de matriz triangular (4x4 por ejemplo). Por cada operación de multiplicación se realiza una de suma. Fijando  $n.^{\circ}$  de threads = 2.

Para chunk=0 (reparto consecutivo de iteraciones), la thread 0: hará  $4*2+3*2=14$  operaciones, la thread 1: hará  $2*2+2*1=6$ .

Para chunk=1, la thread 0 ejecutará la primera y tercera fila hará por tanto  $4*2+2*2=12$  operaciones y la thread 1 ejecutará la segunda y cuarta fila, haciendo  $3*2+1*2=8$  operaciones.

Para chunk=64, la thread 0 hará los cálculos de toda la matriz ya que  $4<64$  por tanto  $4*2+3*2+2*2+1*2=20$  operaciones y la thread 1 no ejecutará ninguna fila.

Podemos concluir entonces que el número de operaciones por thread, dependerá del número de chunk, el tamaño de la matriz y el número de elementos de la fila que le toque ejecutar.

c) El número de operaciones de los threads ya no se adjudicará con el método Round-Robin, por lo que dependerá del tipo de asignación dinámica de las filas que le de el computador en cada caso a cada thread durante ejecución.

Suponiendo que los cores del computador son igual de rápido los threads que acabarán antes serán las que ejecuten las filas con menos elementos (al ser una matriz triangular superior).

Por tanto la carga de trabajo se repartirá de forma bastante similar, ya que los threads a los que se les asignen filas con más elementos tardarán más que a los que se le asignen menos. Al ser una matriz triangular superior, los threads con menos elementos terminarán antes y les tocará trabajar con las filas siguientes que tendrán menos elementos que la que

hayan terminado de ejecutar, pero más que las que ejecutará la thread que aún no ha acabado y que tenía más operaciones que la actual.

Podemos concluir que el número de operaciones por thread dependerá del número de chunk y de hebras, así como el número de elementos de cada fila que le toque ejecutar a cada hebra.

### **CÓDIGO FUENTE:** pmtv-OpenMP.c

```
/* pmtv-OpenMP.c
Multiplicación de matriz por vector:  $v2 = T1 \cdot v1$ , T1 matriz triang. sup
Para compilar usar (-lrt: real time library):
gcc -O2 -fopenmp pmtv-OpenMP1.c -o pmtv-OpenMP -lrt

Para ejecutar use: pmtv-OpenMP longitud
*/

#include <stdlib.h> // biblioteca con funciones atoi(), malloc() y free()
#include <stdio.h> // biblioteca donde se encuentra la función printf()
#include <time.h> // biblioteca donde se encuentra la función clock_gettime()
#include <omp.h>

// #define PRINTF_ALL // comentar para quitar el printf ...
// que imprime todos los componentes

// Sólo puede estar definida uno de los dos constantes PMV_ (sólo uno de
los ...
// un defines siguientes puede estar descomentado):
// #define PMV_GLOBAL // descomentar para que los vector/matriz sean variables
...
// globales (su longitud no estará limitada
por el ...
// tamaño de la pila del programa)
#define PMV_DYNAMIC // descomentar para que los vector/matriz sean
variables ...
// dinámicas (memoria reutilizable durante la
ejecución)
#ifdef PMV_GLOBAL
#define MAX 16350 // aprox  $2^{14}$ 
double T1[MAX], v1[MAX], v2[MAX];
#endif

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

    int i, k, s;
    double suma_fila;
    double cgt1, cgt2, ncgt; // para tiempo de ejecución

    int modifier;
    omp_sched_t kind;

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

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

    #ifdef PMV_GLOBAL
```



```

    if (N>MAX) N=MAX;
    #endif
    #ifdef PMV_DYNAMIC
    double **T1, *v1, *v2;
    T1 = (double**) malloc(N*sizeof(double*)); // malloc necesita el tamaño en
bytes
    v1 = (double*) malloc(N*sizeof(double)); //si no hay espacio suficiente
malloc devuelve NULL
    v2 = (double*) malloc(N*sizeof(double));
    if ( (T1==NULL) || (v1==NULL) || (v2==NULL) ){
        printf("Error en la reserva de espacio para los vectores/matriz\n");
        exit(-2);
    }
    for(i=0; i<N;i++){
        T1[i] = (double*) malloc((N-i)*sizeof(double));
        if ( T1[i]==NULL ){
            printf("Error en la reserva de espacio para la matriz\n");
            exit(-2);
        }
    }
    #endif

    //Inicializar Matriz Triangular Superior y vector
    #pragma omp parallel
    {
        #pragma omp for
        for(i=0; i<N; i++){
            v1[i] = N*0.1-i*0.1; //los valores dependen de N
            v2[i]=0;
        }

        #pragma omp for private(k)
        for(i=0; i<N; i++){
            for(k=0; k<N-i; k++){
                T1[i][k] = N*0.1-i*0.1-k*0.1; //los valores dependen de N
            }
        }

        cgt1 = omp_get_wtime();
        //Calcular producto Matriz Triangular*vector
        //Con runtime el usuario pueda fijar la planificación de tareas usando la
variable de entorno OMP_SCHEDULE
        #pragma omp parallel for private(suma_fila, k, s) schedule(runtime)
        for(i=0; i<N; i++){
            s = i;
            suma_fila = 0;
            for(k=0; k<N-i; k++){
                suma_fila += T1[i][k] * v1[s];
                s++;
            }
            v2[i] = suma_fila;
        }

        cgt2 = omp_get_wtime();
        ncgt= cgt2-cgt1;

        omp_get_schedule(&kind,&modifier);
        printf("Número de threads usando: %d", omp_get_max_threads());
        printf("\nAsignación usada: %d", kind);
        printf("\nChunk usado: %d", modifier);

        //Imprimir resultado de la suma y el tiempo de ejecución

```

```

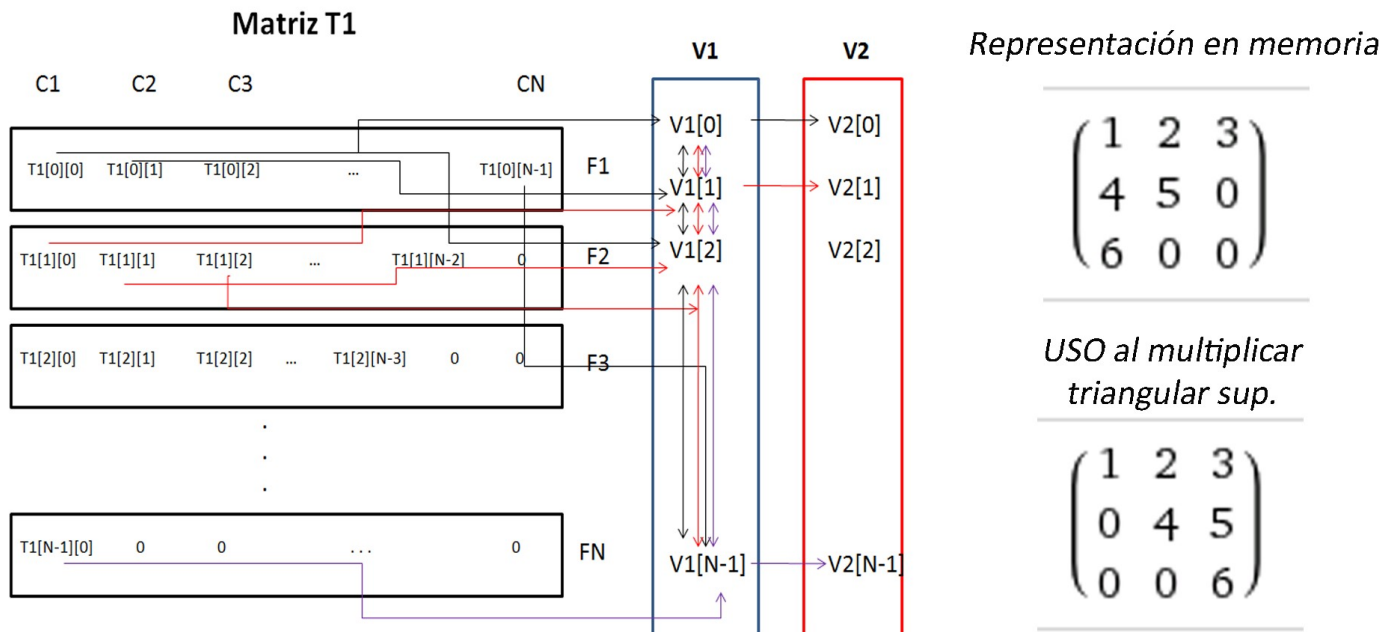
#ifdef PRINTF_ALL
printf("\nTiempo(seg.):%11.9f\t / Tamaño Matriz-Vector:%u\n",ncgt,N);
for(i=0; i<N; i++)
    printf("V2[%d]=%8.6f\n", i, v2[i]);

#else
printf("\nTiempo(seg.):%11.9f\t / Tamaño Matriz-Vector:%u\t/ V2[0]=%8.6f / / V2[%d]=%8.6f /\n", ncgt,N, v2[0], N-1, v2[N-1]);
#endif

#ifdef PMV_DYNAMIC
for(i=0; i<N;i++)
    free(T1[i]);
free(T1); // libera el espacio reservado para T1
free(v1); // libera el espacio reservado para v1
free(v2); // libera el espacio reservado para v2
#endif
return 0;
}

```

### DESCOMPOSICIÓN DE DOMINIO:



## CAPTURAS DE PANTALLA:

```

danibolanos@Aspire-E5-575G:~/Escritorio/Codigo_AC_3$ gcc -O2 -fopenmp p
mtv-OpenMP.c -o pmtv-OpenMP -lrt
danibolanos@Aspire-E5-575G:~/Escritorio/Codigo_AC_3$ ./pmtv-OpenMP 8
Tiempo(seg.):0.000005720 / Tamaño Matriz-Vector:8
V2[0]=2.040000
V2[1]=1.400000
V2[2]=0.910000
V2[3]=0.550000
V2[4]=0.300000
V2[5]=0.140000
V2[6]=0.050000
V2[7]=0.010000
danibolanos@Aspire-E5-575G:~/Escritorio/Codigo_AC_3$ ./pmtv-OpenMP 11
Tiempo(seg.):0.000004864 / Tamaño Matriz-Vector:11
V2[0]=5.060000
V2[1]=3.850000
V2[2]=2.850000
V2[3]=2.040000
V2[4]=1.400000
V2[5]=0.910000
V2[6]=0.550000
V2[7]=0.300000
V2[8]=0.140000
V2[9]=0.050000
V2[10]=0.010000

```

## TABLA RESULTADOS, SCRIPT Y GRÁFICA ATCGRID

**SCRIPT:** pmtv-OpenMP\_atcgrid.sh

```

#!/bin/bash
#Se asigna al trabajo el nombre pmtv-OpenMP
#PBS -N pmtv-OpenMP
#Se asigna al trabajo la cola ac
#PBS -q ac
#Se imprime información del trabajo usando variables de entorno de PBS
echo "Id. usuario del trabajo: $PBS_O_LOGNAME"
echo "Id. del trabajo: $PBS_JOBID"
echo "Nombre del trabajo especificado por usuario: $PBS_JOBNAME"
echo "Nodo que ejecuta qsub: $PBS_O_HOST"
echo "Directorio en el que se ha ejecutado qsub: $PBS_O_WORKDIR"
echo "Cola: $PBS_QUEUE"
echo "Nodos asignados al trabajo:"
cat $PBS_NODEFILE
#Se ejecuta pmtv-OpenMP
export OMP_NUM_THREADS=12

export OMP_SCHEDULE="static"
$PBS_O_WORKDIR/pmtv-OpenMP 76800

export OMP_SCHEDULE="static,1"
$PBS_O_WORKDIR/pmtv-OpenMP 76800

export OMP_SCHEDULE="static,64"
$PBS_O_WORKDIR/pmtv-OpenMP 76800

export OMP_SCHEDULE="dynamic"
$PBS_O_WORKDIR/pmtv-OpenMP 76800

export OMP_SCHEDULE="dynamic,1"
$PBS_O_WORKDIR/pmtv-OpenMP 76800

```

```

export OMP_SCHEDULE="dynamic,64"
$PBS_O_WORKDIR/pmtv-OpenMP 76800

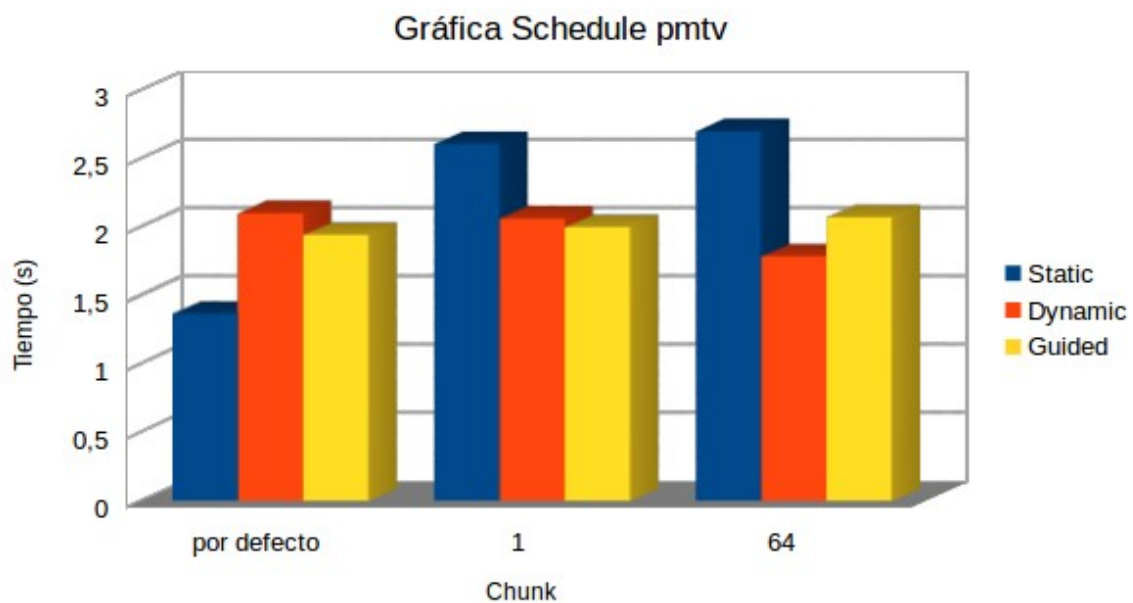
export OMP_SCHEDULE="guided"
$PBS_O_WORKDIR/pmtv-OpenMP 76800

export OMP_SCHEDULE="guided,1"
$PBS_O_WORKDIR/pmtv-OpenMP 76800

export OMP_SCHEDULE="guided,64"
$PBS_O_WORKDIR/pmtv-OpenMP 76800

```

**Nota:** He utilizado la media de los valores de ambas tablas para representar los datos en la gráfica.



**Tabla 3.** Tiempos de ejecución de la versión paralela del producto de una matriz triangular por un vector  $r$  para vectores de tamaño  $N=76800$ , 12 threads

Chunk	Static	Dynamic	Guided
por defecto	1.361285850	2.095312160	1.550302073
1	2.437089629	1.994670678	2.030737881
64	2.626403175	1.780157283	1.745690607
Chunk	Static	Dynamic	Guided
por defecto	1.370379806	2.105185982	2.330846678
1	2.775476683	2.128905144	1.966849491
64	2.772170253	1.789409146	2.393865705

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

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

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

**CÓDIGO FUENTE:** pmm-secuencial.c

```
/* pmm-secuencial.c
Multiplicación de matriz por matriz (cuadradas): v2 = M1*M2
Para compilar usar (-lrt: real time library):
gcc -O2 -fopenmp pmm-secuencial.c -o pmm-secuencial -lrt

Para ejecutar use: pmm-secuencial longitud
*/

#include <stdlib.h> // biblioteca con funciones atoi(), malloc() y free()
#include <stdio.h> // biblioteca donde se encuentra la función printf()
#include <time.h> // biblioteca donde se encuentra la función clock_gettime()
#include <omp.h>

#define PRINTF_ALL // comentar para quitar el printf ...
// que imprime todos los componentes

//Sólo puede estar definida uno de los dos constantes PMV_ (sólo uno de
los ...
//un defines siguientes puede estar descomentado):
//#define PMV_GLOBAL // descomentar para que los vector/matriz sean variables
...
// globales (su longitud no estará limitada
por el ...
// tamaño de la pila del programa)
#define PMV_DYNAMIC // descomentar para que los vector/matriz sean
variables ...
// dinámicas (memoria reutilizable durante la
ejecución)
#ifdef PMV_GLOBAL
#define MAX 16350 // aprox 2^14
double M1[MAX], M2[MAX], M3[MAX];
#endif

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

    int i, j, k;

    double cgt1, cgt2, ncgt; //para tiempo de ejecución

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

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

```

(sizeof(unsigned int) = 4 B)

#ifdef PMV_GLOBAL
if (N>MAX) N=MAX;
#endif
#ifdef PMV_DYNAMIC
double **M1, **M2, **M3;
M1 = (double**) malloc(N*sizeof(double*)); // malloc necesita el tamaño en
bytes
M2 = (double**) malloc(N*sizeof(double*)); //si no hay espacio suficiente
malloc devuelve NULL
M3 = (double**) malloc(N*sizeof(double*));
if ( (M1==NULL) || (M2==NULL) || (M3==NULL) ){
printf("Error en la reserva de espacio para los vectores/matriz\n");
exit(-2);
}
for(i=0; i<N;i++){
M1[i] = (double*) malloc(N*sizeof(double));
M2[i] = (double*) malloc(N*sizeof(double));
M3[i] = (double*) malloc(N*sizeof(double));
if ( (M1[i]==NULL) || (M2[i]==NULL) || (M3[i]==NULL)){
printf("Error en la reserva de espacio para la matriz\n");
exit(-2);
}
}
}
#endif

//Inicializar Matrices
for(i=0; i<N; i++){
for(k=0; k<N; k++){
M1[i][k] = N*0.1-i*0.1-k*0.1; //los valores dependen de N
M2[i][k] = N*0.1-i*0.1-k*0.1; //los valores dependen de N
M3[i][k] = 0; //los valores dependen de N
}
}

cgt1 = omp_get_wtime();
//Calcular producto de matrices
for(i=0; i<N; i++){
for(j=0; j<N; j++){
for(k=0; k<N; k++){
M3[i][j] += M1[i][k] * M2[k][j];
}
}
}

cgt2 = omp_get_wtime();
ncgt= cgt2-cgt1;

//Imprimir resultado de la suma y el tiempo de ejecución
#ifdef PRINTF_ALL
printf("Tiempo(seg.):%11.9f\t / Tamaño Matriz:%u\n",ncgt,N);
for(i=0; i<N; i++)
for(j=0; j<N; j++)
printf("M3[%d][%d]=%8.6f\n", i, j, M3[i][j]);

#else
printf("Tiempo(seg.):%11.9f\t / Tamaño Matriz:%u\t/ M3[0][0]=%8.6f / /
M3[%d][%d]=%8.6f /\n", ncgt,N, M3[0][0], N-1, N-1, M3[N-1][N-1]);
#endif

#ifdef PMV_DYNAMIC

```

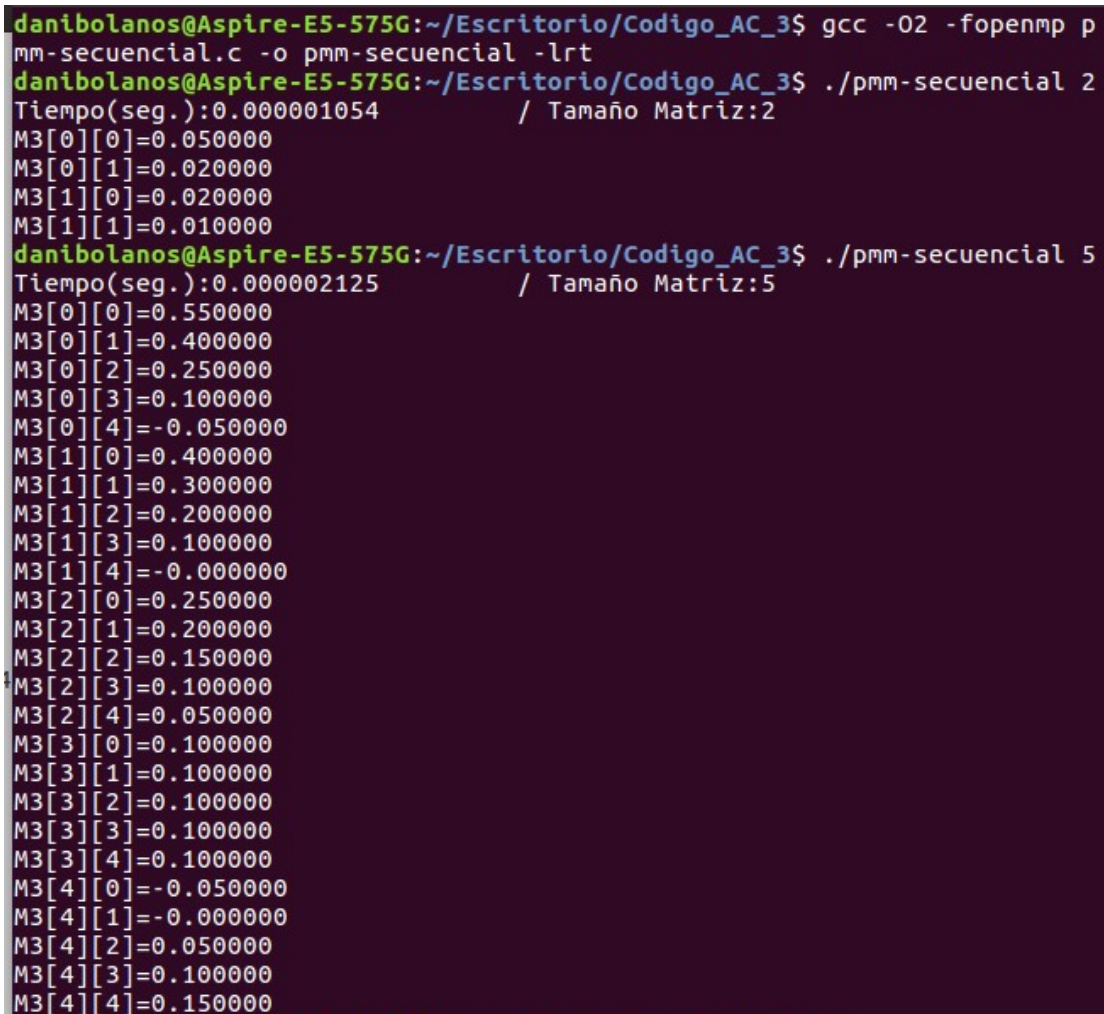


```

    for(i=0; i<N;i++){
        free(M1[i]);
        free(M2[i]);
        free(M3[i]);
    }
    free(M1); // libera el espacio reservado para T1
    free(M2); // libera el espacio reservado para v1
    free(M3); // libera el espacio reservado para v2
    #endif
    return 0;
}

```

### CAPTURAS DE PANTALLA:



```

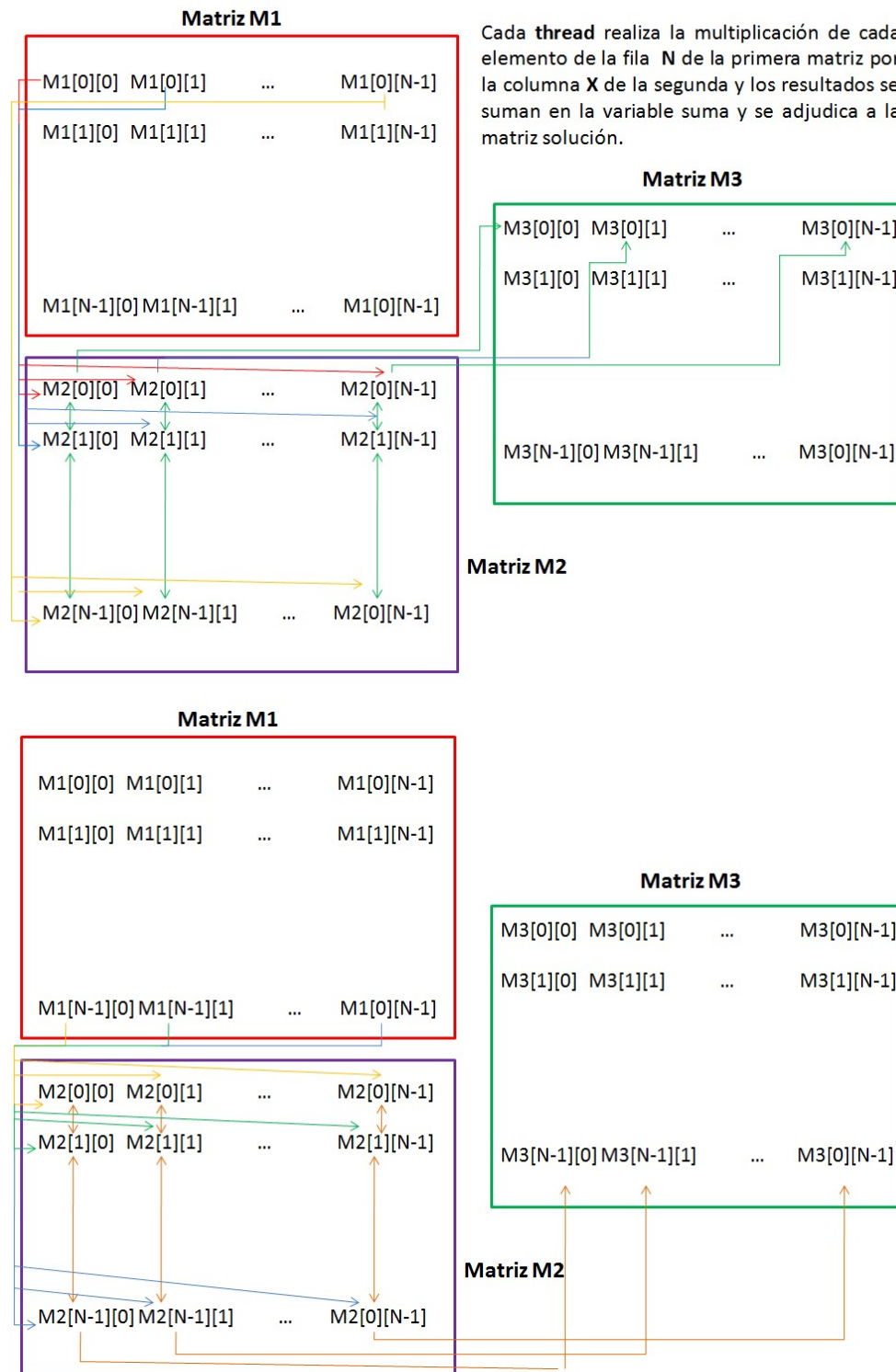
danibolanos@Aspire-E5-575G:~/Escritorio/Codigo_AC_3$ gcc -O2 -fopenmp p
mm-secuencial.c -o pmm-secuencial -lrt
danibolanos@Aspire-E5-575G:~/Escritorio/Codigo_AC_3$ ./pmm-secuencial 2
Tiempo(seg.):0.000001054      / Tamaño Matriz:2
M3[0][0]=0.050000
M3[0][1]=0.020000
M3[1][0]=0.020000
M3[1][1]=0.010000
danibolanos@Aspire-E5-575G:~/Escritorio/Codigo_AC_3$ ./pmm-secuencial 5
Tiempo(seg.):0.000002125      / Tamaño Matriz:5
M3[0][0]=0.550000
M3[0][1]=0.400000
M3[0][2]=0.250000
M3[0][3]=0.100000
M3[0][4]=-0.050000
M3[1][0]=0.400000
M3[1][1]=0.300000
M3[1][2]=0.200000
M3[1][3]=0.100000
M3[1][4]=-0.000000
M3[2][0]=0.250000
M3[2][1]=0.200000
M3[2][2]=0.150000
M3[2][3]=0.100000
M3[2][4]=0.050000
M3[3][0]=0.100000
M3[3][1]=0.100000
M3[3][2]=0.100000
M3[3][3]=0.100000
M3[3][4]=0.100000
M3[4][0]=-0.050000
M3[4][1]=-0.000000
M3[4][2]=0.050000
M3[4][3]=0.100000
M3[4][4]=0.150000

```

(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

9. Implementar en paralelo la multiplicación de matrices cuadradas con OpenMP a partir del código escrito en el ejercicio anterior. Use las directivas, las cláusulas y las funciones de entorno que considere oportunas. Se debe paralelizar también la inicialización de las matrices. Dibuje en su cuaderno de prácticas la descomposición de dominio que ha utilizado en el código paralelo implementado para asignar tareas a los threads (Lección 4/Tema 2, Lección 5/Tema 2).

### DESCOMPOSICIÓN DE DOMINIO:



**CÓDIGO FUENTE:** pmm-OpenMP.c

```

/* pmm-OpenMP.c
Multiplicación de matriz por matriz (cuadradas): v2 = M1*M2
Para compilar usar (-lrt: real time library):
gcc -O2 -fopenmp pmm-OpenMP.c -o pmm-OpenMP -lrt

Para ejecutar use: pmm-OpenMP longitud
*/

#include <stdlib.h> // biblioteca con funciones atoi(), malloc() y free()
#include <stdio.h> // biblioteca donde se encuentra la función printf()
#include <time.h> // biblioteca donde se encuentra la función clock_gettime()
#include <omp.h>

// #define PRINTF_ALL // comentar para quitar el printf ...
// que imprime todos los componentes

// Sólo puede estar definida uno de los dos constantes PMV_ (sólo uno de
los ...
// un defines siguientes puede estar descomentado):
// #define PMV_GLOBAL // descomentar para que los vector/matriz sean variables
...
// globales (su longitud no estará limitada
por el ...
// tamaño de la pila del programa)
#define PMV_DYNAMIC // descomentar para que los vector/matriz sean
variables ...
// dinámicas (memoria reutilizable durante la
ejecución)
#ifdef PMV_GLOBAL
#define MAX 16350 // aprox 2^14
double M1[MAX], M2[MAX], M3[MAX];
#endif

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

    int i, j, k;
    double suma;

    double cgt1, cgt2, ncgt; // para tiempo de ejecución

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

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

    #ifdef PMV_GLOBAL
    if (N > MAX) N = MAX;
    #endif
    #ifdef PMV_DYNAMIC
    double **M1, **M2, **M3;
    M1 = (double**) malloc(N*sizeof(double*)); // malloc necesita el tamaño en
bytes
    M2 = (double**) malloc(N*sizeof(double*)); // si no hay espacio suficiente
malloc devuelve NULL
    M3 = (double**) malloc(N*sizeof(double*));
    if ( (M1==NULL) || (M2==NULL) || (M3==NULL) ){

```

```

        printf("Error en la reserva de espacio para los vectores/matriz\n");
        exit(-2);
    }
    for(i=0; i<N;i++){
        M1[i] = (double*) malloc(N*sizeof(double));
        M2[i] = (double*) malloc(N*sizeof(double));
        M3[i] = (double*) malloc(N*sizeof(double));
        if ( (M1[i]==NULL) || (M2[i]==NULL) || (M3[i]==NULL)){
            printf("Error en la reserva de espacio para la matriz\n");
            exit(-2);
        }
    }
}
#endif

//Inicializar Matrices
#pragma omp parallel for private(k)
for(i=0; i<N; i++){
    for(k=0; k<N; k++){
        M1[i][k] = N*0.1-i*0.1-k*0.1; //los valores dependen de N
        M2[i][k] = N*0.1-i*0.1-k*0.1; //los valores dependen de N
        M3[i][k] = 0; //los valores dependen de N
    }
}

cgt1 = omp_get_wtime();
//Calcular producto de matrices
#pragma omp parallel for private(k,j, suma)
for(i=0; i<N; i++){
    for(j=0; j<N; j++){
        suma = 0;
        for(k=0; k<N; k++){
            suma += M1[i][k] * M2[k][j];
        }
        M3[i][j] = suma;
    }
}

cgt2 = omp_get_wtime();
ncgt= cgt2-cgt1;

printf("Número de threads usando: %d\n", omp_get_max_threads());

//Imprimir resultado de la suma y el tiempo de ejecución
#ifdef PRINTF_ALL
printf("Tiempo(seg.):%11.9f\t / Tamaño Matriz:%u\n",ncgt,N);
for(i=0; i<N; i++){
    for(j=0; j<N; j++){
        printf("M3[%d][%d]=%8.6f\n", i, j, M3[i][j]);
    }
}
#else
printf("Tiempo(seg.):%11.9f\t / Tamaño Matriz:%u\t/ M3[0][0]=%8.6f / /
M3[%d][%d]=%8.6f /\n", ncgt,N, M3[0][0], N-1, N-1, M3[N-1][N-1]);
#endif

#ifdef PMV_DYNAMIC
for(i=0; i<N;i++){
    free(M1[i]);
    free(M2[i]);
    free(M3[i]);
}
free(M1); // libera el espacio reservado para T1
free(M2); // libera el espacio reservado para v1
free(M3); // libera el espacio reservado para v2

```

```
#endif
return 0;
}
```

**CAPTURAS DE PANTALLA:**

```
danibolanos@Aspire-E5-575G:~/Escritorio/Codigo_AC_3$ gcc -O2 -fopenmp p
mm-OpenMP.c -o pmm-OpenMP -lrt
danibolanos@Aspire-E5-575G:~/Escritorio/Codigo_AC_3$ ./pmm-OpenMP 2
Tiempo(seg.):0.000005632          / Tamaño Matriz:2
M3[0][0]=0.050000
M3[0][1]=0.020000
M3[1][0]=0.020000
M3[1][1]=0.010000
danibolanos@Aspire-E5-575G:~/Escritorio/Codigo_AC_3$ ./pmm-OpenMP 5
Tiempo(seg.):0.000002050          / Tamaño Matriz:5
M3[0][0]=0.550000
M3[0][1]=0.400000
M3[0][2]=0.250000
M3[0][3]=0.100000
M3[0][4]=-0.050000
M3[1][0]=0.400000
M3[1][1]=0.300000
M3[1][2]=0.200000
```

```
M3[1][3]=0.100000
M3[1][4]=-0.000000
M3[2][0]=0.250000
M3[2][1]=0.200000
M3[2][2]=0.150000
M3[2][3]=0.100000
M3[2][4]=0.050000
M3[3][0]=0.100000
M3[3][1]=0.100000
M3[3][2]=0.100000
M3[3][3]=0.100000
M3[3][4]=0.100000
M3[4][0]=-0.050000
M3[4][1]=-0.000000
M3[4][2]=0.050000
M3[4][3]=0.100000
M3[4][4]=0.150000
```

**(ADJUNTAR CÓDIGO FUENTE AL .ZIP)**

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

**ESTUDIO DE ESCALABILIDAD EN ATCGRID:****SCRIPT:** pmm-OpenMP\_atcgrid\_1.sh

//Existe otra versión para el tamaño 1200, pmm-OpenMP\_atcgrid\_2.sh incluso he tenido que hacer por separada la versión de 12 threads para tamaño 1200 porque la tarea sobrepasaba el tiempo de atcgrid.

```
#!/bin/bash
#Se asigna al trabajo el nombre pmm-OpenMP
#PBS -N pmm-OpenMP
#Se asigna al trabajo la cola ac
#PBS -q ac
#Se imprime información del trabajo usando variables de entorno de PBS
echo "Id. usuario del trabajo: $PBS_O_LOGNAME"
echo "Id. del trabajo: $PBS_JOBID"
echo "Nombre del trabajo especificado por usuario: $PBS_JOBNAME"
echo "Nodo que ejecuta qsub: $PBS_O_HOST"
echo "Directorio en el que se ha ejecutado qsub: $PBS_O_WORKDIR"
echo "Cola: $PBS_QUEUE"
echo "Nodos asignados al trabajo:"
cat $PBS_NODEFILE
#Se ejecuta pmm-OpenMP

$PBS_O_WORKDIR/pmm-secuencial 200

export OMP_NUM_THREADS=1

$PBS_O_WORKDIR/pmm-OpenMP 200

export OMP_NUM_THREADS=2

$PBS_O_WORKDIR/pmm-OpenMP 200

export OMP_NUM_THREADS=3

$PBS_O_WORKDIR/pmm-OpenMP 200

export OMP_NUM_THREADS=4

$PBS_O_WORKDIR/pmm-OpenMP 200

export OMP_NUM_THREADS=5

$PBS_O_WORKDIR/pmm-OpenMP 200

export OMP_NUM_THREADS=6

$PBS_O_WORKDIR/pmm-OpenMP 200

export OMP_NUM_THREADS=7

$PBS_O_WORKDIR/pmm-OpenMP 200

export OMP_NUM_THREADS=8

$PBS_O_WORKDIR/pmm-OpenMP 200

export OMP_NUM_THREADS=9

$PBS_O_WORKDIR/pmm-OpenMP 200
```



```
export OMP_NUM_THREADS=10

$PBS_O_WORKDIR/pmm-OpenMP 200

export OMP_NUM_THREADS=11

$PBS_O_WORKDIR/pmm-OpenMP 200

export OMP_NUM_THREADS=12

$PBS_O_WORKDIR/pmm-OpenMP 200
```

## ESTUDIO DE ESCALABILIDAD EN PCLOCAL:

**SCRIPT:** pmm-OpenMP\_pclocal.sh

```
#!/bin/bash

echo "TAMANIO 200"

echo "Programa secuencial"
./pmm-secuencial 200

echo "Programa paralelo 1 threads"
export OMP_NUM_THREADS=1

./pmm-OpenMP 200

echo "Programa paralelo 2 threads"
export OMP_NUM_THREADS=2

./pmm-OpenMP 200

echo "Programa paralelo 3 threads"
export OMP_NUM_THREADS=3

./pmm-OpenMP 200

echo "Programa paralelo 4 threads"
export OMP_NUM_THREADS=4

./pmm-OpenMP 200

echo "TAMANIO 1200"

echo "Programa secuencial"
./pmm-secuencial 1200

echo "Programa paralelo 1 threads"
export OMP_NUM_THREADS=1

./pmm-OpenMP 1200

echo "Programa paralelo 2 threads"
export OMP_NUM_THREADS=2

./pmm-OpenMP 1200

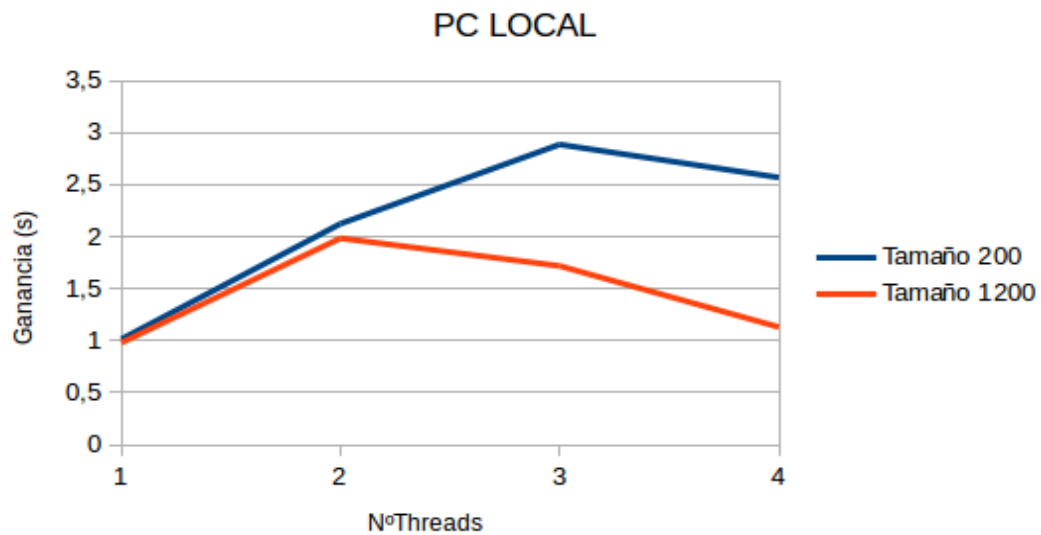
echo "Programa paralelo 3 threads"
export OMP_NUM_THREADS=3
```

```
./pmm-OpenMP 1200

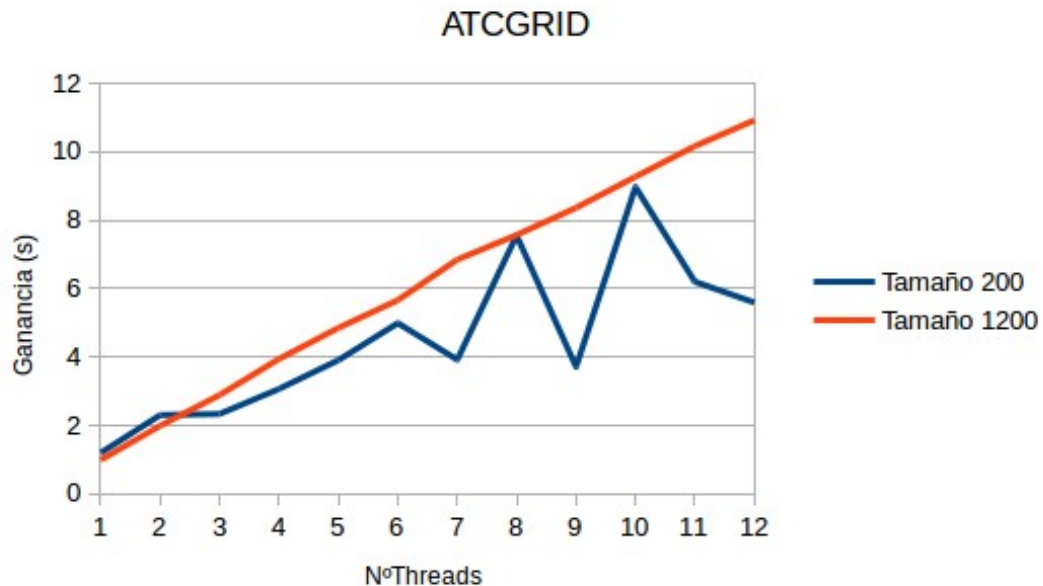
echo "Programa paralelo 4 threads"
export OMP_NUM_THREADS=4

./pmm-OpenMP 1200
```

PC LOCAL				
Tamaño	N,º Threads	T,secuencial	T,paralelo	Ganancia
200	1	0,013929447	0,013740134	1,0137781043
200	2	0,013929447	0,00655928	2,1236243917
200	3	0,013929447	0,004828992	2,8845454704
200	4	0,013929447	0,005424635	2,5678127653
Tamaño	N,º Threads	T,secuencial	T,paralelo	Ganancia
1200	1	5,005052893	5,109642694	0,9795308973
1200	2	5,005052893	2,522439532	1,9842112485
1200	3	5,005052893	2,912427362	1,7185159562
1200	4	5,005052893	4,426992308	1,1305763699



ATCGRID				
Tamaño	N,º Threads	T,secuencial	T,paralelo	Ganancia
200	1	0,018066913	0,015094057	1,1969553977
200	2	0,018066913	0,007842343	2,303764704
200	3	0,018066913	0,007737692	2,3349227392
200	4	0,018066913	0,005891513	3,06659987
200	5	0,018066913	0,0046218	3,9090642174
200	6	0,018066913	0,003621001	4,989480257
200	7	0,018066913	0,004606932	3,9216799814
200	8	0,018066913	0,002392091	7,5527699406
200	9	0,018066913	0,004864357	3,7141420747
200	10	0,018066913	0,002013803	8,9715394207
200	11	0,018066913	0,002912689	6,202829413
200	12	0,018066913	0,003229968	5,5935269328
Tamaño	N,º Threads	T,secuencial	T,paralelo	Ganancia
1200	1	20,942679755	21,118914716	0,9916551128
1200	2	20,942679755	10,530177474	1,9888249563
1200	3	20,942679755	7,253701448	2,8871714538
1200	4	20,942679755	5,310121503	3,9439172424
1200	5	20,942679755	4,316575766	4,8516882108
1200	6	20,942679755	3,697840385	5,6634893815
1200	7	20,942679755	3,058232665	6,8479681074
1200	8	20,942679755	2,766869817	7,5690875033
1200	9	20,942679755	2,50384295	8,3642145986
1200	10	20,942679755	2,258818321	9,2715202282
1200	11	20,942679755	2,059835978	10,167158929
1200	12	20,942679755	1,917243212	10,923329718



### COMENTARIOS SOBRE LOS RESULTADOS:

He realizado el ejercicio para dos tamaños de matrices, 200 y 1200 y me he dispuesto a calcular los tiempos de ejecución para cada N.º de hebras en el nodo de atcgrid y en mi PC local.

En general y como ya comentamos en la práctica anterior, vemos que a mayor número de threads la ganancia aumenta ya que al repartirse el trabajo en esta versión del programa, a mayor número de componentes, el computador tarda menos.

En mi PC local no se da este caso, posiblemente sea debido a que al tener 2 cores físicos, mi ordenador no puede ejecutar las 4 threads simultáneamente, por tanto, la versión de 4 hebras, tarda más que la de 2.

En cambio en ATCGRID, se puede observar que con 200 componentes se obtiene mejor escalabilidad cuanto menor sea el número de threads (mejor escalabilidad con 8 o 10 threads que con 12), con 1200 sin embargo ocurre lo contrario. Esto ocurre porque la programación en paralelo es mejor cuanto más carga de trabajo tengamos.

Los picos que se ven en las gráfica de ATCGRID para tamaño de 200 pueden ser debidos como ya comentamos en la BP2, a que la creación y destrucción de threads también afecta al tiempo de ejecución y puede causar que para valores de threads utilizadas cercanos, existan oscilaciones en los resultados obtenidos.