


<p>2º curso / 2º cuatr. Grado Ing. Inform. Doble Grado Ing. Inform. y Mat.</p>	<h2>Arquitectura de Computadores (AC)</h2> <h3>Cuaderno de prácticas.</h3> <h3>Bloque Práctico 3. Programación paralela III: Interacción con el entorno en OpenMP</h3> <p>Estudiante (nombre y apellidos): Carlos de la Torre</p> <p>Grupo de prácticas: A2</p> <p>Fecha de entrega: 19/05</p> <p>Fecha evaluación en clase: 28/05</p>	
--	--	---

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`

```

/*
 * if-clauseModificado.c
 *
 * Created on: 28/04/2014
 * Author: Carlos de la Torre
 */
#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, h = 1, n = 20, tid;
    int a[n], suma = 0, sumalocal;
    if (argc < 3) {
        if (argv[1] == NULL) {
            printf("[USAGE]-%s [num iteracciones] [num hebras]\n", argv[0]);
            fprintf(stderr, "[ERROR]-Falta iteraciones\n");
            exit(-1);
        } else {
            printf("[USAGE]-%s [num iteracciones] [num hebras]\n", argv[0]);
            fprintf(stderr, "[ERROR]-Falta numero de hebras\n");
            exit(-1);
        }
    }
    h = atoi(argv[2]);
    n = atoi(argv[1]);
    if (n > 20)
        n = 20;
    for (i = 0; i < n; i++) {
        a[i] = i;
    }
    #pragma omp parallel num_threads(h) private(sumalocal,tid) shared(a,suma,n){
        printf("Esto es lo que tiene omp_num_threads():\n", omp_get_num_threads());
        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);
}
return 0;
}

```

CAPTURAS DE PANTALLA:

```

[usuario@portatil Practica03_AC]$ ./bin/if-clauseModificado 8 1
Esto es lo que tiene omp_num_threads(): 1
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 0 suma de a[5]=5 sumalocal=15
thread 0 suma de a[6]=6 sumalocal=21
thread 0 suma de a[7]=7 sumalocal=28
thread master=0 imprime suma=28
[usuario@portatil Practica03_AC]$ ./bin/if-clauseModificado 8 2
Esto es lo que tiene omp_num_threads(): 2
thread 1 suma de a[4]=4 sumalocal=4
thread 1 suma de a[5]=5 sumalocal=9
thread 1 suma de a[6]=6 sumalocal=15
thread 1 suma de a[7]=7 sumalocal=22
Esto es lo que tiene omp_num_threads(): 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 master=0 imprime suma=28
[usuario@portatil Practica03_AC]$ ./bin/if-clauseModificado 8 4
Esto es lo que tiene omp_num_threads(): 4
thread 0 suma de a[0]=0 sumalocal=0
thread 0 suma de a[1]=1 sumalocal=1
Esto es lo que tiene omp_num_threads(): 4
thread 3 suma de a[6]=6 sumalocal=6
thread 3 suma de a[7]=7 sumalocal=13
Esto es lo que tiene omp_num_threads(): 4
thread 1 suma de a[2]=2 sumalocal=2
thread 1 suma de a[3]=3 sumalocal=5
Esto es lo que tiene omp_num_threads(): 4
thread 2 suma de a[4]=4 sumalocal=4
thread 2 suma de a[5]=5 sumalocal=9
thread master=0 imprime suma=28
[usuario@portatil Practica03_AC]$ █

```

RESPUESTA:

Con el segundo parámetro que introducimos mediante consola le estamos diciendo a nuestro ejecutable cual es el valor para la cantidad de hebras en las que queremos que se ejecuten la paralelización de tareas, al fijarnos en los resultados obtenidos podemos ver claramente que si utilizamos una sola hebra la suma de todos los componentes del vector la realiza la hebra 0 por lo tanto la salida de nuestro programa es correcta, y para asegurarnos simplemente ponemos que la cantidad de hebras sean 4 y vemos que las sumas parciales se realizan en 4 hebras diferentes.

2. (a) Rellenar la Tabla 1 (se debe poner en la tabla el *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

Iteración	Schedule-clause.c			Scheduled-clause.c			Scheduleg-clause.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	1	1	0	0	0	0
3	1	1	0	1	1	0	0	0	0
4	0	0	1	1	0	1	0	0	0
5	1	0	1	1	0	1	0	0	0
6	0	1	1	1	0	1	0	0	0
7	1	1	1	1	0	1	0	0	0
8	0	0	0	1	0	0	1	1	1
9	1	0	0	1	0	0	1	1	1
10	0	1	0	1	0	0	1	1	1
11	1	1	0	1	0	0	1	1	1
12	0	0	1	1	0	1	1	0	1
13	1	0	1	1	0	1	1	0	1
14	0	1	1	1	0	1	1	0	1
15	1	1	1	1	0	1	1	0	1

(b) Rellenar la Tabla 2 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

Iteración	Schedule-clause.c			Scheduled-clause.c			Scheduleg-clause.c		
	1	2	4	1	2	4	1	2	4
0	0	0	0	0	2	3	3	2	2
1	1	0	0	2	2	3	3	2	2
2	2	1	0	3	3	3	3	2	2
3	3	1	0	1	3	3	3	2	2
4	0	2	1	2	1	0	0	0	1
5	1	2	1	0	1	0	0	0	1
6	2	3	1	0	0	0	0	0	1
7	3	3	1	0	0	0	2	1	1
8	0	0	2	0	0	1	2	1	0
9	1	0	2	0	0	1	2	1	0
10	2	1	2	0	0	1	3	3	0
11	3	1	2	0	0	1	3	3	0
12	0	2	3	0	0	2	3	2	3
13	1	2	3	0	0	2	3	2	3
14	2	3	3	0	0	2	3	1	3
15	3	3	3	0	0	2	3	1	3

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`

```

/*
 * scheduled-clause.c
 *
 * Created on: 28/04/2014
 * Author: Carlos de la Torre
 */

#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, hebras = 1, a[n], suma = 0;

    if (argc < 4) {
        if (argv[1]==NULL){
            fprintf(stderr, "[ERROR]-Falta iteraciones\n");
            exit(-1);
        }else if (argv[2]==NULL){
            fprintf(stderr, "[ERROR]-Falta chunk\n");
            exit(-1);
        }else if (argv[3]==NULL){
            fprintf(stderr, "[ERROR]-Falta numero de hebras\n");
            exit(-1);
        }else{
            printf("[USAGE]-%s [num iteraciones] [num chunk] [num
hebras]\n",argv[0]);
        }
    }
    n = atoi(argv[1]);
    if (n > 200)
        n = 200;
    chunk = atoi(argv[2]);
    hebras = atoi(argv[3]);

    for (i = 0; i < n; i++)
        a[i] = i;
#pragma omp parallel num_threads(hebras)
    {
        #pragma omp for firstprivate(suma) lastprivate(suma) schedule(dynamic,chunk)
        for (i = 0; i < n; i++) {
            suma = suma + a[i];
            if (argv[4]==NULL)
                printf(" La iteración %d la realiza thread %d suma a[%d]=
%d suma=%d \n", i, omp_get_thread_num(), i, a[i], suma);
            else if (atoi(argv[4])==1)
                printf("%d %d\n", i, omp_get_thread_num());
        }
        #pragma omp master
        if (omp_in_parallel())
            printf("Valores de las variables de control dentro del
parallel:\n"
                " dyn-var: %s\n"
                " nthreads-var: %s\n"
                " thread-limit-var: %s\n"
                " nest-var: %s\n"
                " run-sched-var:
%s\n",getenv("OMP_DYNAMIC"),getenv("OMP_NUM_THREADS"),getenv("OMP_THREAD_LIMIT"),getenv(
"OMP_NESTED"),getenv("OMP_SCHEDULE"));
    }
}

```

```

printf("Valores de las variables de control fuera del parallel:\n"
      " dyn-var: %s\n"
      " nthreads-var: %s\n"
      " thread-limit-var: %s\n"
      " nest-var: %s\n"
      " run-sched-var: %s\n",
      getenv("OMP_DYNAMIC"),getenv("OMP_NUM_THREADS"),getenv("OMP_THREAD_LIMIT"),getenv(
"OMP_NESTED"),getenv("OMP_SCHEDULE"));
printf("Fuera de 'parallel for' suma=%d\n", suma);
return 0;
}

```

CAPTURAS DE PANTALLA:

```

[usuario@portatil Practica03_AC]$ ./bin/scheduled-clauseModificado 10 4 4
La iteración 8 la realiza thread 1 suma a[8]=8 suma=8
La iteración 9 la realiza thread 1 suma a[9]=9 suma=17
La iteración 4 la realiza thread 3 suma a[4]=4 suma=4
La iteración 5 la realiza thread 3 suma a[5]=5 suma=9
La iteración 6 la realiza thread 3 suma a[6]=6 suma=15
La iteración 7 la realiza thread 3 suma a[7]=7 suma=22
La iteración 0 la realiza thread 0 suma a[0]=0 suma=0
La iteración 1 la realiza thread 0 suma a[1]=1 suma=1
La iteración 2 la realiza thread 0 suma a[2]=2 suma=3
La iteración 3 la realiza thread 0 suma a[3]=3 suma=6
Valores de las variables de control dentro del parallel:
dyn-var: TRUE
nthreads-var: 4
thread-limit-var: (null)
nest-var: TRUE
run-sched-var: dynamic
Valores de las variables de control fuera del parallel:
dyn-var: TRUE
nthreads-var: 4
thread-limit-var: (null)
nest-var: TRUE
run-sched-var: dynamic
Fuera de 'parallel for' suma=17
[usuario@portatil Practica03_AC]$ ./bin/scheduled-clauseModificado 5 4 4
La iteración 4 la realiza thread 3 suma a[4]=4 suma=4
La iteración 0 la realiza thread 2 suma a[0]=0 suma=0
La iteración 1 la realiza thread 2 suma a[1]=1 suma=1
La iteración 2 la realiza thread 2 suma a[2]=2 suma=3
La iteración 3 la realiza thread 2 suma a[3]=3 suma=6
Valores de las variables de control dentro del parallel:
dyn-var: TRUE
nthreads-var: 4
thread-limit-var: (null)
nest-var: TRUE
run-sched-var: dynamic
Valores de las variables de control fuera del parallel:
dyn-var: TRUE
nthreads-var: 4
thread-limit-var: (null)
nest-var: TRUE
run-sched-var: dynamic
Fuera de 'parallel for' suma=4
[usuario@portatil Practica03_AC]$

```

RESPUESTA:

Si, se imprimen siempre los mismos valores, ya que los valores de las variables de entorno no cambian durante la ejecución del programa.

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`

```

/*
 * scheduled-clauseModificado4.c
 *
 * Created on: 28/04/2014
 * Author: Carlos de la Torre
 */

#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, hebras = 1, a[n], suma = 0;

    if (argc < 4) {
        if (argv[1]==NULL){
            fprintf(stderr, "[ERROR]-Falta iteraciones\n");
            exit(-1);
        }else if (argv[2]==NULL){
            fprintf(stderr, "[ERROR]-Falta chunk\n");
            exit(-1);
        }else if (argv[3]==NULL){
            fprintf(stderr, "[ERROR]-Falta numero de hebras\n");
            exit(-1);
        }else{
            printf("[USAGE]-%s [num iteraciones] [num chunk] [num
hebras]\n",argv[0]);
        }
    }
    n = atoi(argv[1]);
    if (n > 200)
        n = 200;
    chunk = atoi(argv[2]);
    hebras = atoi(argv[3]);
    for (i = 0; i < n; i++)
        a[i] = i;
#pragma omp parallel num_threads(hebras)
    {
        #pragma omp for firstprivate(suma) lastprivate(suma) schedule(dynamic,chunk)
        for (i = 0; i < n; i++) {
            suma = suma + a[i];
            if (argv[4]==NULL)
                printf(" La iteración %d la realiza thread %d suma a[%d]=
%d suma=%d \n", i, omp_get_thread_num(), i, a[i], suma);
            else if (atoi(argv[4])==1)
                printf("%d %d\n", i, omp_get_thread_num());
        }
        #pragma omp master
        if (omp_in_parallel())
            printf("El valor de las funciones dentro de la region paralela
son:\n"
                " omp_get_num_threads() %d\n"
                " omp_get_num_procs(): %d\n"
                " omp_in_parallel():
%d\n",omp_get_num_threads(),omp_get_num_procs(),omp_in_parallel());
        printf("El valor de las funciones fuera de la region paralela son:\n"
            " omp_get_num_threads() %d\n"
            " omp_get_num_procs(): %d\n"
            " omp_in_parallel():

```

```
%d\n",omp_get_num_threads(),omp_get_num_procs(),omp_in_parallel());
    printf("Fuera de 'parallel for' suma=%d\n", suma);
    return 0;
}
```

CAPTURAS DE PANTALLA:

```
[A2estudiante7@atcgrid ~]$ echo '03_Practica3_OMP/scheduled-clauseModificado4 10 4 4' | qsub -q ac
40759.atcgrid
[A2estudiante7@atcgrid ~]$ cat STDIN.o40759
La iteración 4 la realiza thread 3 suma a[4]=4 suma=4
La iteración 5 la realiza thread 3 suma a[5]=5 suma=9
La iteración 6 la realiza thread 3 suma a[6]=6 suma=15
La iteración 7 la realiza thread 3 suma a[7]=7 suma=22
La iteración 0 la realiza thread 0 suma a[0]=0 suma=0
La iteración 1 la realiza thread 0 suma a[1]=1 suma=1
La iteración 2 la realiza thread 0 suma a[2]=2 suma=3
La iteración 3 la realiza thread 0 suma a[3]=3 suma=6
La iteración 8 la realiza thread 2 suma a[8]=8 suma=8
La iteración 9 la realiza thread 2 suma a[9]=9 suma=17
El valor de las funciones dentro de la region paralela son:
omp_get_num_threads() 4
omp_get_num_procs(): 24
omp_in_parallel(): 1
El valor de las funciones fuera de la region paralela son:
omp_get_num_threads() 1
omp_get_num_procs(): 24
omp_in_parallel(): 0
Fuera de 'parallel for' suma=17
[A2estudiante7@atcgrid ~]$
```

RESPUESTA:

Como se puede ver en la captura de pantalla los valores de las hebras que están dentro de la región paralela serán aquellas que nosotros le indiquemos al programa y fuera de dicha región solo habrá 1 y la función `omp_in_parallel()` por supuesto tendrá diferente valor según donde se ejecute.

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

```
/*
 * sheduled-clauseModificado5.c
 *
 * Created on: 28/04/2014
 * Author: Carlos de la Torre
 */

#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, hebras = 1, a[n], suma = 0;

    if (argc < 4) {
        if (argv[1]==NULL){
            fprintf(stderr, "[ERROR]-Falta iteraciones\n");
            exit(-1);
        }else if (argv[2]==NULL){
            fprintf(stderr, "[ERROR]-Falta chunk\n");
            exit(-1);
        }else if (argv[3]==NULL){
            fprintf(stderr, "[ERROR]-Falta numero de hebras\n");
            exit(-1);
        }else{
            printf("[USAGE]-%s [num iteraciones] [num chunk] [num
```

```

hebras]\n",argv[0]);
    }
    }
    n = atoi(argv[1]);
    if (n > 200)
        n = 200;
    chunk = atoi(argv[2]);
    hebras = atoi(argv[3]);
    printf("Valores de las variables de control antes de la modificacion valen:\n"
           " dyn-var: %s\n"
           " nthreads-var: %s\n"
           " run-sched-var: %s\n",
           getenv("OMP_DYNAMIC"),getenv("OMP_NUM_THREADS"),getenv("OMP_SCHEDULE"));

    /* Las siguientes cuatro líneas setean las diferentes variables de entorno de
    usuario
    * normalmente esto se tendría que hacer desde la consola para que cada usuario
    * modificase la forma de ejecución del programa a su antojo, pero en esta
    practica
    * para no tener que estar continuamente modificando el entorno del usuario
    * se ha implementado directamente desde el programa.
    */
    setenv("OMP_DYNAMIC","TRUE",1); // Seteamos a true el ajuste dinámico del nº de
    threads
    setenv("OMP_NUM_THREADS","4",1); // Seteamos el nº de threads en la siguiente
    ejecución paralela
    setenv("OMP_NESTED","TRUE",1); // Seteamos a true el paralelismo anidado
    setenv("OMP_SCHEDULE","dynamic",1); // Elegimos como queremos la planificación
    de bucles

    printf("Valores de las variables de control después de la modificación y fuera
    del parallel valen:\n"
           " dyn-var: %s\n"
           " nthreads-var: %s\n"
           " run-sched-var: %s\n",
           getenv("OMP_DYNAMIC"),getenv("OMP_NUM_THREADS"),getenv("OMP_SCHEDULE"));
    for (i = 0; i < n; i++)
        a[i] = i;
    #pragma omp parallel num_threads(hebras)
    {
        #pragma omp for firstprivate(suma) lastprivate(suma) schedule(dynamic,chunk)
        for (i = 0; i < n; i++) {
            suma = suma + a[i];
            if (argv[4]==NULL)
                printf(" La iteración %d la realiza thread %d suma a[%d]=
            %d suma=%d \n", i, omp_get_thread_num(), i, a[i], suma);
            else if (atoi(argv[4])==1)
                printf("%d %d\n", i, omp_get_thread_num());
        }
        #pragma omp master
        if (omp_in_parallel())
            printf("Valores de las variables de control dentro del
            parallel:\n"
                   " dyn-var: %s\n"
                   " nthreads-var: %s\n"
                   " thread-limit-var: %s\n"
                   " nest-var: %s\n"
                   " run-sched-var: %s\n",
                   getenv("OMP_DYNAMIC"),getenv("OMP_NUM_THREADS"),getenv("OMP_THREAD_LIMIT"),getenv(
                   "OMP_NESTED"),getenv("OMP_SCHEDULE"));
    }
    printf("Fuera de 'parallel for' suma=%d\n", suma);
    return 0;
}

```


CAPTURAS DE PANTALLA:

```
[usuario@portatil Practica03_AC]$ ./bin/scheduled-clauseModificado5 10 3 4
Valores de las variables de control antes de la modificacion valen:
dyn-var: (null)
nthreads-var: (null)
run-sched-var: (null)
Valores de las variables de control después de la modificación y fuera del parallel valen:
dyn-var: TRUE
nthreads-var: 4
run-sched-var: dynamic
La iteración 0 la realiza thread 0 suma a[0]=0 suma=0
La iteración 1 la realiza thread 0 suma a[1]=1 suma=1
La iteración 2 la realiza thread 0 suma a[2]=2 suma=3
La iteración 6 la realiza thread 1 suma a[6]=6 suma=6
La iteración 7 la realiza thread 1 suma a[7]=7 suma=13
La iteración 8 la realiza thread 1 suma a[8]=8 suma=21
La iteración 3 la realiza thread 2 suma a[3]=3 suma=3
La iteración 4 la realiza thread 2 suma a[4]=4 suma=7
La iteración 5 la realiza thread 2 suma a[5]=5 suma=12
La iteración 9 la realiza thread 3 suma a[9]=9 suma=9
Valores de las variables de control dentro del parallel:
dyn-var: TRUE
nthreads-var: 4
thread-limit-var: (null)
nest-var: TRUE
run-sched-var: dynamic
Fuera de 'parallel for' suma=9
[usuario@portatil Practica03_AC]$ ./bin/scheduled-clauseModificado5 5 2 4
Valores de las variables de control antes de la modificacion valen:
dyn-var: (null)
nthreads-var: (null)
run-sched-var: (null)
Valores de las variables de control después de la modificación y fuera del parallel valen:
dyn-var: TRUE
nthreads-var: 4
run-sched-var: dynamic
La iteración 0 la realiza thread 0 suma a[0]=0 suma=0
La iteración 1 la realiza thread 0 suma a[1]=1 suma=1
La iteración 2 la realiza thread 3 suma a[2]=2 suma=2
La iteración 3 la realiza thread 3 suma a[3]=3 suma=5
La iteración 4 la realiza thread 1 suma a[4]=4 suma=4
Valores de las variables de control dentro del parallel:
dyn-var: TRUE
nthreads-var: 4
thread-limit-var: (null)
nest-var: TRUE
run-sched-var: dynamic
Fuera de 'parallel for' suma=4
[usuario@portatil Practica03_AC]$
```

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 las componentes (0,0) y (N-1, N-1) del resultado antes de que termine el programa.

CÓDIGO FUENTE: pmtv-secuencial.c

```
/*
 * pmtv-secuencial.c
 *
 * Created on: 04/05/2014
 * Author: Carlos de la Torre
 */

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define PRINT_ALL_MIN 15
// Ponemos que los elementos mínimos para que se
// impriman todos los valores de la matriz sea 15

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

```

    int f,c,N,TIME;
    double tr;
    struct timespec t1, t2;
    // con este switch nos aseguramos de que la entrada de parámetros sea correcta
    switch (argc){
        case 1:
            printf("Faltan las filas/columnas de la Matriz, y el tamaño del
vector\n");
            printf("\nUso: %s [numero] [0/1]\n",argv[0]);
            printf("\nDonde numero es el tamaño de las filas y las columnas
de la matriz y el tamaño del vector\n");
            printf("y el 0 o el 1 especifica si queremos solo los tiempos (1)
o no\n");
            exit(-1);
            break;
        case 2:
            // Este sera el tamaño del vector y de las filas/columnas de la matriz
            N = atoi(argv[1]);
            TIME = 0;
            break;
        case 3:
            N = atoi(argv[1]);
            TIME = atoi(argv[2]);
            // si tiene un valor de 0 se imprime toda la info si tiene un valor de 1 se imprime solo
            el tiempo
            break;
        default:
            printf("La cantidad de parametros es incorrecta\n");
            exit(-1);
            break;
    }

    int *vector, *Vresultado;
    int **MatrizTri;
    MatrizTri = (int**) malloc(N * sizeof(int*));
    for (f = 0; f < N; f++)
        MatrizTri[f] = (int*) malloc(N * sizeof(int));
    vector = (int*) malloc(N * sizeof(int)); //si no hay espacio suficiente malloc
devuelve NULL
    Vresultado = (int*) malloc(N * sizeof(int));
    if ((MatrizTri == NULL) || (vector == NULL) || (Vresultado == NULL)) {
        printf("Error en la reserva de espacio para los Vectores o MatrizTri\n");
        exit(-2);
    }

    srand(time(NULL)); // esta es la semilla que se usa para los random
    // Inicializamos la Matriz y el vector
    for(f = 0; f < N; f++){
        for(c = 0; c < N; c++){
            if(f > c) // <---- Cambiando el sentido del símbolo la matriz es
superior o inferior
                MatrizTri[f][c]=rand()%10;
            else
                MatrizTri[f][c]=0;
        }
        vector[f] = rand()%10;
    }

    // imprimimos la matriz y el vector si el tamaño de N < PRINT_ALL_MIN
    if (N <= PRINT_ALL_MIN && TIME!=1){
        printf ("\nEsta es la matriz: \n");
        for (f = 0; f < N; f++){
            for (c = 0; c < N; c++){
                printf ("%d ",MatrizTri[f][c]);
            }
            printf ("\n");
        }
        printf ("\nEste es el vector: \n");
        for (f = 0; f < N; f++){
            printf ("%d ",vector[f]);
        }
        printf("\n\n");
    }

    // Calcular la multiplicación de una matriz por un vector
    clock_gettime(CLOCK_REALTIME, &t1);

```

```

    for (f = 0; f < N; f++)
        for (c = 0; c < N; c++)
            Vresultado[f] += MatrizTri[f][c] * vector[c];

    clock_gettime(CLOCK_REALTIME, &t2);

    // calculamos el tiempo que hemos tardado en calcular la multiplicación
    tr = (double) (t2.tv_sec - t1.tv_sec) + (double) ((t2.tv_nsec - t1.tv_nsec) /
(1.e+9));

    // Ahora imprimimos por pantalla los resultados obtenidos según las
restricciones del problema
    if (N <= PRINT_ALL_MIN){
        printf("Tiempo(seg.):%11.9f\nTamaño Matriz y Vector:%u\n",tr,N); // si
queremos imprimir datos completos y N < PRINT_ALL_MIN
        printf("Este es el vector resultante: \n");
        printf("{");
        for (f = 0; f < N; f++){
            if (f==N-1)
                printf ("VR[%d]=%d",f,Vresultado[f]);
            else
                printf ("VR[%d]=%d, ",f,Vresultado[f]);
        }
        printf("}\n");
    }else if (TIME==1) // si queremos imprimir unicamente el tiempo de cálculo
        printf("%11.9f\n",tr); //
    else{ // y si queremos imprimir el tiempo la primera y la ultima multiplicación
        printf("Tiempo(seg.):%11.9f\n",tr);
        printf("Tamaño Matriz y Vector:%u\n",N);
        printf("(Matriz[0][0]=%d)*(Vector[0]=%d)=%d\n",MatrizTri[0]
[0],vector[0],MatrizTri[0][0]*vector[0]);
        printf("(Matriz[%d][%d]=%d)*(Vector[%d]=%d)=%d\n",N-1,N-1,MatrizTri[N-1]
[N-1],N-1,vector[N-1],MatrizTri[N-1][N-1]*vector[N-1]);
    }
    free(vector);
    free(Vresultado);
    for(f=0; f<N; f++)
        free(MatrizTri[f]);
    free(MatrizTri);
    return 0;
}

```

CAPTURAS DE PANTALLA:

```

[A2estudiante7@atcgrid ~]$ echo './03_Practica3_OMP/pmtv-secuencial 1000' | qsub -q ac
39496.atcgrid
[A2estudiante7@atcgrid ~]$ cat STDIN.o39496
Tiempo(seg.):0.001273105
Tamaño Matriz y Vector:1000
(Matriz[0][0]=0)*(Vector[0]=2)=0
(Matriz[999][999]=0)*(Vector[999]=6)=0
[A2estudiante7@atcgrid ~]$ echo './03_Practica3_OMP/pmtv-secuencial 5' | qsub -q ac
39497.atcgrid
[A2estudiante7@atcgrid ~]$ cat STDIN.o39497

Esta es la matriz:
0 0 0 0 0
3 0 0 0 0
3 7 0 0 0
4 7 4 0 0
5 6 7 4 0

Este es el vector:
8 4 0 0 1

Tiempo(seg.):0.000000208
Tamaño Matriz y Vector:5
Este es el vector resultante:
{VR[0]=0, VR[1]=24, VR[2]=52, VR[3]=60, VR[4]=64}

```

- 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 (Lec-

ción 4/Tema 2) en el código paralelo implementado para asignar tareas a los threads (Leción 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 Tabla 3 con 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.

CÓDIGO FUENTE: pmtv-OpenMP.c

```
/*
 * pmtv-OpenMP.c
 *
 * Created on: 04/05/2014
 * Author: Carlos de la Torre
 */

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
#ifdef _OPENMP
#include <omp.h> // biblioteca para programas paralelos
#else
#define omp_get_thread_num() 0
#endif
#define PRINT_ALL_MIN 15
// Ponemos que los elementos mínimos para que se
// impriman todos los valores de la matriz sea 15
#define NELEMENTOS(x) ((sizeof(x) / sizeof(x[0])))
// Con esto lo que hacemos es saber cual es el numero
// de elementos de cualquier vector de C solo tenemos
// que poner donde pone x el nombre del vector
#define DEBUGMODE 0
// con esta definición nos aseguramos que solo
// salgan las cifras de tiempo en cada ejecución
// así de esa manera es mas fácil realizar el
// estudio empírico del programa

void error(char* param[]){
    printf("\n [USAGE]-%s [num iteraciones] [planificación] [num chunk] [num
    hebras]\n"
           " para mas información %s --help\n\n", param[0],param[0]);
    if (param[1]==NULL){
        fprintf(stderr, " [ERROR]-Falta iteraciones\n");
        exit(-1);
    }else if (param[2]==NULL){
        fprintf(stderr, " [ERROR]-Falta modo de planificación: static, dynamic,
    guided\n");
        exit(-1);
    }else if (param[3]==NULL){
        fprintf(stderr, " [ERROR]-Falta chunk\n");
        exit(-1);
    }
    exit(-1);
}

int main(int argc, char* argv[]) {
    int f,c,N;
    char planificacion[10], chunk[2]="", hebras[2], tmp[6], help[6]="--help";
    char statics[9]="static", dynamic[10]="dynamic", guided[9]="guided";
    double tr, t1, t2;

    if (argc!=2 && argc!=4 && argc!=5){
```

```

        error(argv);
    }else if (argc==2){
        if (getenv("OMP_SCHEDULE")!=NULL){
            N = atoi(argv[1]); // Este sera el tamaño del vector y de las
filas/columnas de la matriz
            snprintf(hebras, sizeof(int), "%d", omp_get_num_procs());
            if (!strcmp(dynamic,getenv("OMP_SCHEDULE"),7))
                setenv("OMP_DYNAMIC","TRUE",1);
            strcpy(planificacion,getenv("OMP_SCHEDULE"));
        }else{
            strcpy(tmp,argv[1]);
            if (!strcmp(tmp,help,6)){
                printf("\n [USAGE]-%s [num iteraciones] [planificación]
[num chunk] [num hebras]\n\n"
                                " Tambien se puede utilizar la variable de
entorno\n"
                                " OMP_SCHEDULE para modificar la
planificación\n\n"
                                " Ejemplos:\n"
                                " export OMP_SCHEDULE=\"static,4\"\n"
                                " %s 10 la cantidad de hebras se asignará
según el número de cores\n\n 0\n"
                                " %s 10 dynamic 4
4\n\n",argv[0],argv[0],argv[0]);
                exit(-1);
            }else
                error(argv);
        }
    }else if (argc==4){
        N = atoi(argv[1]); // Este sera el tamaño del vector y de las
filas/columnas de la matriz
        strcpy(planificacion,argv[2]); // Esto es para poder capturar el texto
desde consola
        strcpy(chunk,argv[3]); // Cual es el chunk del programa
        sprintf(hebras, "%d",omp_get_num_procs());
    }else if (argc==5){
        N = atoi(argv[1]); // Este sera el tamaño del vector y de las
filas/columnas de la matriz
        strcpy(planificacion,argv[2]); // Esto es para poder capturar el texto
desde consola
        strcpy(chunk,argv[3]); // Cual es el chunk del programa
        strcpy(hebras,argv[4]);
    }

    /* He elegido esta manera de asignar los valores al programa por que en OMP
    * en la escala de prioridad esta es la segunda opción osea que la unica
    * manera de poder cambiar la planificación del programa sería editando
    * el codigo y utilizando un if.
    */
    if (!strcmp(statics,planificacion)){ // ponemos ! por que si las cadenas son
iguales el valor es 0
        if (strcmp(chunk,"")){
            strcat(statics,"");
            strcat(statics,chunk);
        }
        setenv("OMP_SCHEDULE",statics,1); // Elegimos como queremos la
planificación de bucles
    }else if (!strcmp(dynamic,planificacion)){
        if (strcmp(chunk,"")){
            strcat(dynamic,"");
            strcat(dynamic,chunk);
        }
        setenv("OMP_SCHEDULE",dynamic,1); // Elegimos como queremos la
planificación de bucles
        setenv("OMP_DYNAMIC","TRUE",1); // Seteamos a true el ajuste dinámico del
nº de threads
    }else if (!strcmp(guided,planificacion)){
        if (strcmp(chunk,"")){
            strcat(guided,"");
            strcat(guided,chunk);
        }
        setenv("OMP_SCHEDULE",guided,1); // Elegimos como queremos la
planificación de bucles
    }
    setenv("OMP_NUM_THREADS",hebras,1); // Seteamos el nº de threads en la siguiente

```

ejecución paralela

```

int *vector, *Vresultado;
int **MatrizTri;
MatrizTri = (int**) malloc(N * sizeof(int*));
for (f = 0; f < N; f++)
    MatrizTri[f] = (int*) malloc(N * sizeof(int));
vector = (int*) malloc(N * sizeof(int)); //si no hay espacio suficiente malloc
devuelve NULL
Vresultado = (int*) malloc(N * sizeof(int));
if ((MatrizTri == NULL) || (vector == NULL) || (Vresultado == NULL)) {
    printf("Error en la reserva de espacio para los Vectores o MatrizTri\n");
    exit(-2);
}

srand(time(NULL)); // esta es la semilla que se usa para los random
#pragma omp parallel for schedule(runtime) private(f,c) // Inicializamos la Matriz y el
vector
for(f = 0; f < N; f++){
    for(c = 0; c < N; c++){
        if(f > c) // <---- Cambiando el sentido del simbolo la matriz es
superior o inferior
            MatrizTri[f][c]=rand()%10;
        else
            MatrizTri[f][c]=0;
    }
    vector[f] = rand()%10;
}

// imprimimos la matriz y el vector si el tamaño de N < PRINT_ALL_MIN
if (N <= PRINT_ALL_MIN && DEBUGMODE!=1){
    printf("\nEsta es la matriz: \n");
    for (f = 0; f < N; f++){
        for (c = 0; c < N; c++){
            printf ("%d ",MatrizTri[f][c]);
        }
        printf ("\n");
    }
    printf ("\nEste es el vector: \n");
    for (f = 0; f < N; f++)
        printf ("%d ",vector[f]);
    printf("\n\n");
}

t1 = omp_get_wtime(); // Calcular la multiplicación de una matriz por un vector
#pragma omp parallel
{
    #pragma omp for schedule(runtime) private (f,c)
    for (f = 0; f < N; f++){
        for (c = 0; c < N; c++)
            Vresultado[f] += MatrizTri[f][c]*vector[c];
    }
}
#pragma omp master
    if (omp_in_parallel())
        printf("Valores de las variables de control dentro del
parallel:\n"
                " dyn-var: %s\n"
                " nthreads-var: %s\n"
                " thread-limit-var: %s\n"
                " nest-var: %s\n"
                " run-sched-var:
%s\n",getenv("OMP_DYNAMIC"),getenv("OMP_NUM_THREADS"),
getenv("OMP_THREAD_LIMIT"),getenv("OMP_NESTED"),getenv("OMP_SCHEDULE"));
}
t2 = omp_get_wtime();
tr = t2 - t1; // Calculo el tiempo que he tardado en multiplicarlo

// Ahora imprimimos por pantalla los resultados obtenidos segun las
restricciones del problema
if (N <= PRINT_ALL_MIN){
    // si queremos imprimir datos completos y N < PRINT_ALL_MIN
    printf("Tiempo(seg.):%11.9f\nTamaño Matriz y Vector:%u\n",tr,N);
    printf ("Este es el vector resultante: \n");
    printf("{");

```

```

        for (f = 0; f < N; f++){
            if (f==N-1)
                printf ("VR[%d]=%d",f,Vresultado[f]);
            else
                printf ("VR[%d]=%d, ",f,Vresultado[f]);
        }
        printf("\n");
    }else if (DEBUGMODE==1) // si queremos imprimir unicamente el tiempo de cálculo
        printf("%11.9f\n",tr);//
    else{ // y si queremos imprimir el tiempo la primera y la ultima multiplicación
        printf("Tiempo(seg.):%11.9f\n",tr);
        printf("Tamaño Matriz y Vector:%u\n",N);
        printf("(Matriz[0][0]=%d)*(Vector[0]=%d)=%d\n",MatrizTri[0]
[0],vector[0],MatrizTri[0][0]*vector[0]);
        printf("(Matriz[%d][%d]=%d)*(Vector[%d]=%d)=%d\n",N-1,N-1,MatrizTri[N-1]
[N-1],N-1,vector[N-1],MatrizTri[N-1][N-1]*vector[N-1]);
    }
    free(vector);
    free(Vresultado);
    for(f=0; f<N; f++)
        free(MatrizTri[f]);
    free(MatrizTri);
    return 0;
}

```

CAPTURAS DE PANTALLA:

```

[usuario@portatil Practica03_AC]$ ./bin/pmtv-OpenMP --help

[USAGE] ./bin/pmtv-OpenMP [num iteraciones] [planificación] [num chunk] [num hebras]

Tambien se puede utilizar la variable de entorno
OMP_SCHEDULE para modificar la planificación

Ejemplos:
export OMP_SCHEDULE="static,4"
./bin/pmtv-OpenMP 10 la cantidad de hebras se asignará según el número de cores

0
./bin/pmtv-OpenMP 10 dynamic 4 4

```

```

[usuario@portatil Practica03_AC]$ export OMP_SCHEDULE=static,4
[usuario@portatil Practica03_AC]$ printenv | grep OMP
OMP_SCHEDULE=static,4
[usuario@portatil Practica03_AC]$ ./bin/pmtv-OpenMP 10

Esta es la matriz:
0 0 0 0 0 0 0 0 0 0
2 0 0 0 0 0 0 0 0 0
0 7 0 0 0 0 0 0 0 0
7 1 4 0 0 0 0 0 0 0
8 1 6 3 0 0 0 0 0 0
9 5 9 6 3 0 0 0 0 0
8 5 4 9 0 6 0 0 0 0
8 2 2 2 9 1 3 0 0 0
8 4 4 4 4 9 9 3 0 0
6 3 7 3 9 4 3 0 4 0

Este es el vector:
8 4 2 9 2 2 5 5 7 5

Valores de las variables de control dentro del parallel:
dyn-var: (null)
nthreads-var: 4
thread-limit-var: (null)
nest-var: (null)
run-sched-var: static,4
Tiempo(seg.):0.000097542
Tamaño Matriz y Vector:10
Este es el vector resultante:
{VR[0]=0, VR[1]=16, VR[2]=28, VR[3]=68, VR[4]=107, VR[5]=170, VR[6]=185, VR[7]=129, VR[8]=210, VR[9]=170}
[usuario@portatil Practica03_AC]$ █

```

Tabla 3 . Gráfica de Descomposición y Asignación de Tareas

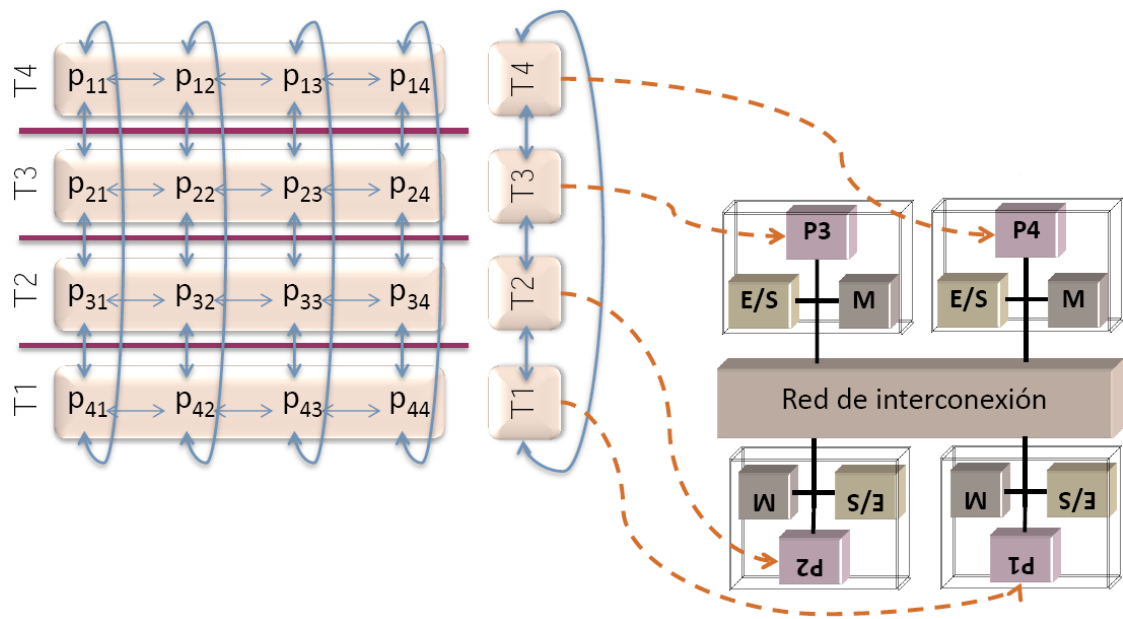
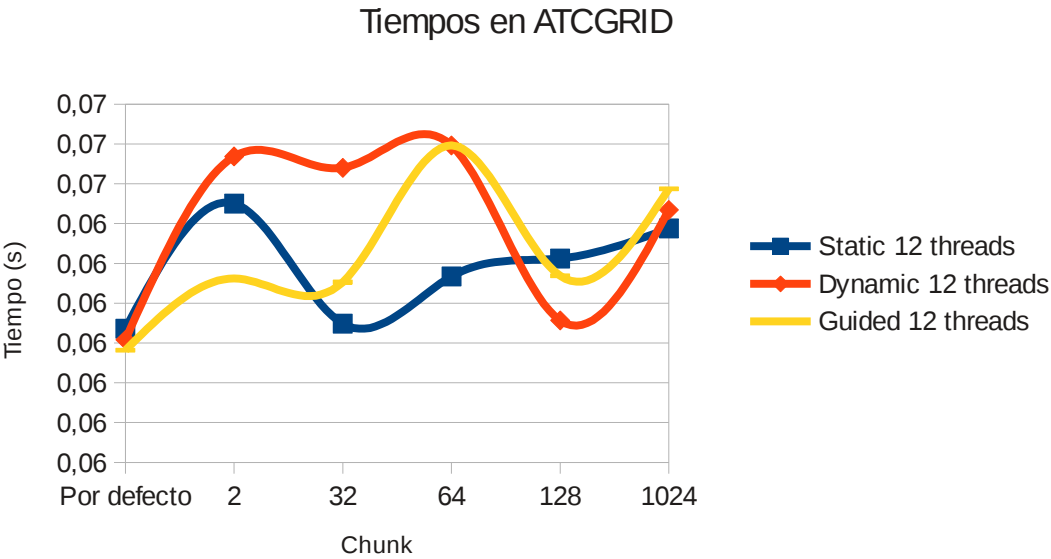


Tabla 4 . Tiempos de ejecución de la versión paralela en sus tres planificaciones diferentes ATCGRID

chunk	Static 12 threads	Dynamic 12 threads	Guided 12 threads
Por defecto	0,061357807	0,061079897	0,060818653
2	0,064503436	0,065687405	0,062618909
32	0,061485105	0,065399652	0,062531109
64	0,062670807	0,065961193	0,065961193
128	0,063123811	0,061564824	0,062687051
1024	0,063876363	0,064341138	0,064874376

Tabla 5 . Gráfica de tiempo de ejecución en programa paralelo en sus tres planificaciones con valores de chunk diferentes en ATCGRID

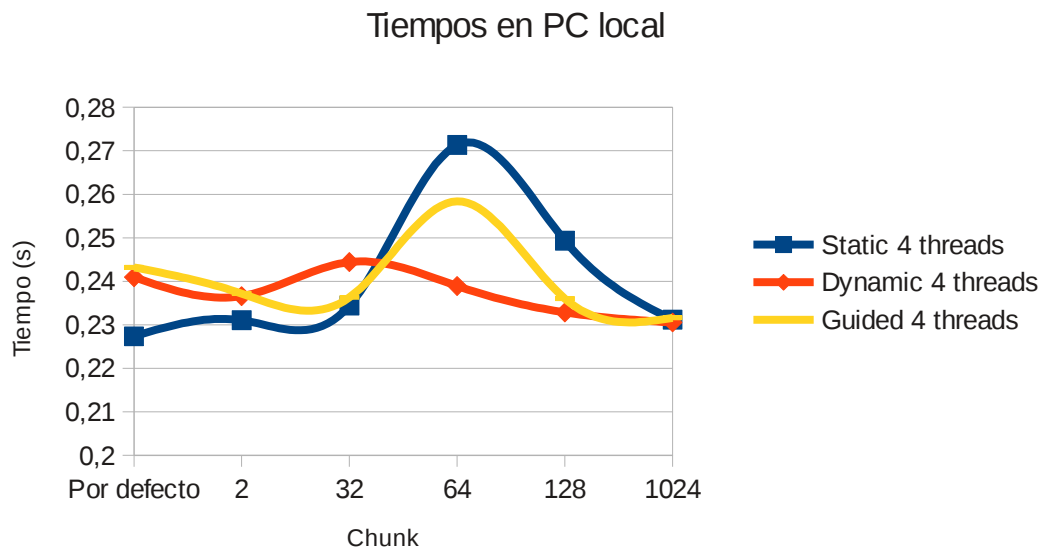


El PC Local es un Intel(R) Core(TM)2 Quad CPU Q6600 2.40GHz con 4GB de Memoria DDR2 a 1333Mhz

Tabla 6 . Tiempos de ejecución de la versión paralela en sus tres planificaciones diferentes PC local

Chunk	Static 4 threads	Dynamic 4 threads	Guided 4 threads
Por defecto	0,227371657	0,240940109	0,243203249
2	0,231053543	0,23660928	0,237191986
32	0,234436326	0,244427119	0,236295484
64	0,271345491	0,238939543	0,258382488
128	0,249343529	0,232857828	0,236038489
1024	0,231191634	0,230549015	0,231614964

Tabla 7. Gráfica de tiempo de ejecución en programa paralelo en sus tres planificaciones con valores de chunk diferentes en PC local



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

```

/*
 * pmtv-secuencial.c
 *
 * Created on: 04/05/2014
 * Author: Carlos de la Torre
 */

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define PRINT_ALL_MIN 15
// Ponemos que los elementos mínimos para que se
// impriman todos los valores de la matriz sea 15
#define DEBUGMODE 0
// con esta definición nos aseguramos que solo
// salgan las cifras de tiempo en cada ejecución
// así de esa manera es mas fácil realizar el
// estudio empírico del programa

int main(int argc, char* argv[]) {
    int f,c,k,N;
    double tr;
    struct timespec t1, t2;

    switch (argc){ // coneste switch nos aseguramos de que la entrada de parametros
sea correcta
        case 1:
            printf("Faltan las filas/columnas de la Matrices\n");
            printf("\nUso: %s [numero]\n",argv[0]);

```

```

        printf("\nDonde numero es el tamaño de las filas y las columnas
de la matrices\n");
        exit(-1);
        break;
    case 2:
        N = atoi(argv[1]); // Este sera el tamaño del vector y de las
filas/columnas de la matriz
        break;
    default:
        printf("La cantidad de parametros es incorrecta\n");
        exit(-1);
        break;
}

int **M1, **M2, **MR;
M1 = (int**) malloc(N * sizeof(int*));
for (f = 0; f < N; f++)
    M1[f] = (int*) malloc(N * sizeof(int));
M2 = (int**) malloc(N * sizeof(int*));
for (f = 0; f < N; f++)
    M2[f] = (int*) malloc(N * sizeof(int));
MR = (int**) malloc(N * sizeof(int*));
for (f = 0; f < N; f++)
    MR[f] = (int*) malloc(N * sizeof(int));
if ((M1 == NULL) || (M2 == NULL) || (MR == NULL)) {
    printf("Error en la reserva de espacio para los Vectores o MatrizTri\n");
    exit(-2);
}

srand(time(NULL)); // esta es la semilla que se usa para los random
// Inicializamos la Matriz y el vector
for(f = 0; f < N; f++)
    for(c = 0; c < N; c++){
        M1[f][c]=rand()%10;
        M2[f][c]=rand()%10;
    }

// imprimimos la matriz y el vector si el tamaño de N < PRINT_ALL_MIN
if (N <= PRINT_ALL_MIN && DEBUGMODE!=1){
    printf("\nEsta es la matriz 1: \n");
    for (f = 0; f < N; f++){
        for (c = 0; c < N; c++){
            printf ("%d ",M1[f][c]);
        }
        printf ("\n");
    }
    printf ("\nEsta es la matriz 2: \n");
    for (f = 0; f < N; f++){
        for (c = 0; c < N; c++){
            printf ("%d ",M2[f][c]);
        }
        printf ("\n");
    }
    printf ("\n");
}

// Calcular la multiplicación de una matriz por un vector
clock_gettime(CLOCK_REALTIME, &t1);
for (f = 0; f < N; ++f) {
    for (c = 0; c < N; ++c) {
        for (k = 0; k < N; ++k) {
            MR[f][c] += M1[f][k] * M2[k][c];
        }
    }
}
clock_gettime(CLOCK_REALTIME, &t2);

// calculamos el tiempo que hemos tardado en calcular la multiplicación
tr = (double) (t2.tv_sec - t1.tv_sec) + (double) ((t2.tv_nsec - t1.tv_nsec) /
(1.e+9));

// Ahora imprimimos por pantalla los resultados obtenidos segun las
restricciones del problema
if (N <= PRINT_ALL_MIN && DEBUGMODE == 0){
    printf("Tiempo(seg.):%11.9f\nTamaño Matriz y Vector:%u\n",tr,N);// si

```

```

queremos imprimir datos completos y N < PRINT_ALL_MIN
printf ("Este es la matriz resultante: \n");
for (f = 0; f < N; f++){
    for (c = 0; c < N; c++){
        printf ("%d ",MR[f][c]);
    }
    printf ("\n");
}
printf("\n");
}else if (DEBUGMODE==1) // si queremos imprimir unicamente el tiempo de cálculo
    printf("%11.9f\n",tr);//
else{ // y si queremos imprimir el tiempo la primera y la ultima multiplicación
    printf("Tiempo(seg.):%11.9f\n",tr);
    printf("Tamaño Matriz 1, Matriz 2 y Matriz resultante: %u\n",N);
    printf("(M1[0][0]=%d)*(M2[0][0]=%d)=%d\n",M1[0][0],M2[0][0],MR[0][0]);
    printf("(M1[%d][%d]=%d)*(M2[%d][%d]=%d)=%d\n",N-1,N-1,M1[N-1][N-1],N-1,N-1,
1,M2[N-1][N-1],MR[N-1][N-1]);
}
for(f=0; f<N; f++){
    free(M1[f]);
    free(M2[f]);
    free(MR[f]);
}
free(M1);
free(M2);
free(MR);
return 0;
}

```

CAPTURAS DE PANTALLA:

```

[usuario@portatil Practica03_AC]$ ./bin/pmm-secuencial 10

Esta es la matriz 1:
4 3 9 0 7 5 0 7 2 0
7 8 1 0 4 2 9 8 1 2
0 4 0 2 1 4 6 2 7 1
9 4 2 9 5 3 0 7 0 5
2 5 7 7 1 8 6 1 5 4
3 8 9 5 5 8 3 9 4 7
0 8 8 7 3 8 6 9 5 1
9 2 6 5 0 8 8 1 9 6
4 8 3 0 5 1 3 9 6 2
7 3 8 8 5 9 1 2 0 7

Esta es la matriz 2:
3 7 4 1 7 4 3 0 8 9
1 1 0 2 4 1 7 8 0 8
7 0 1 2 2 9 4 7 2 1
5 8 3 3 1 5 9 0 9 4
1 8 7 4 1 2 2 1 0 1
0 3 6 2 0 9 5 1 9 5
5 6 1 5 4 6 7 7 6 0
9 2 1 0 3 2 6 8 8 2
6 0 9 0 1 4 5 9 4 9
0 8 4 7 4 9 9 6 4 7

Tiempo(seg.):0.000001495
Tamaño Matriz y Vector:10
Este es la matriz resultante:
160 116 129 66 88 181 160 173 159 133
163 181 103 104 156 163 233 225 206 181
105 88 112 63 60 129 166 164 138 135
158 242 145 109 138 199 256 140 260 220
165 177 159 125 104 279 283 217 245 209
230 225 195 152 152 323 354 311 293 268
243 174 162 118 118 276 327 296 283 213
199 227 219 138 156 329 311 247 317 288
178 131 134 77 124 148 218 250 169 199
148 249 180 145 128 307 283 159 275 230

[usuario@portatil Practica03_AC]$ █

```

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 utilizada (Lección 4/Tema 2) en el código paralelo implementado para asignar tareas a los threads (Lección 5/Tema 2).

CÓDIGO FUENTE: pmm-OpenMP.c

```

/*
 * pmm-OpenMP.c
 *
 * Created on: 04/05/2014
 * Author: Carlos de la Torre
 */

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
#ifdef _OPENMP
    #include <omp.h> // biblioteca para programas paralelos
#else
    #define omp_get_thread_num() 0
#endif
#define PRINT_ALL_MIN 15
// Ponemos que los elementos mínimos para que se
// impriman todos los valores de la matriz sea 15
#define NELEMENTOS(x) (sizeof(x) / sizeof(x[0]))
// Con esto lo que hacemos es saber cual es el numero
// de elementos de cualquier vector de C solo tenemos
// que poner donde pone x el nombre del vector
#define DEBUGMODE 0
// con esta definición nos aseguramos que solo
// salgan las cifras de tiempo en cada ejecución
// así de esa manera es mas fácil realizar el
// estudio empírico del programa

void error(char* param[]){
    printf("\n [USAGE]-%s [num iteraciones] [planificación] [num chunk] [num
hebras]\n"
           " para mas información %s --help\n\n", param[0],param[0]);
    if (param[1]==NULL){
        fprintf(stderr, " [ERROR]-Falta iteraciones\n");
        exit(-1);
    }else if (param[2]==NULL){
        fprintf(stderr, " [ERROR]-Falta modo de planificación: static, dynamic,
guided\n");
        exit(-1);
    }else if (param[3]==NULL){
        fprintf(stderr, " [ERROR]-Falta chunk\n");
        exit(-1);
    }
    }else if (argv[4]==NULL){
        fprintf(stderr, " [ERROR]-Falta numero de hebras\n");
        exit(-1);
    }
    }
    exit(-1);
}

int main(int argc, char* argv[]) {
    int f,c,k,N;
    char planificacion[10], chunk[2]="", hebras[2], tmp[6], help[6]="--help";
    char statics[9]="static", dynamic[10]="dynamic", guided[9]="guided";
    double tr, t1, t2;

    if (argc!=2 && argc!=3 && argc!=4 && argc!=5){
        error(argv);
    }else if (argc==2){
        if (getenv("OMP_SCHEDULE")!=NULL){
            N = atoi(argv[1]); // Este sera el tamaño del vector y de las
filas/columnas de la matriz

```

```

        snprintf(hebras, sizeof(int), "%d", omp_get_num_procs());
        if (!strcmp(dynamic, getenv("OMP_SCHEDULE"), 7))
            setenv("OMP_DYNAMIC", "TRUE", 1);
        strcpy(planificacion, getenv("OMP_SCHEDULE"));
    }else{
        strcpy(tmp, argv[1]);
        if (!strcmp(tmp, help, 6)){
            printf("\n [USAGE]-%s [num iteraciones] [planificación]
[num chunk] [num hebras]\n\n"
                                " Tambien se puede utilizar la variable de
entorno\n"
                                " OMP_SCHEDULE para modificar la
planificación\n\n"
                                " Ejemplos:\n"
                                " export OMP_SCHEDULE=\"static,4\"\n"
                                " %s 10 la cantidad de hebras se asignará
según el número de cores\n\n 0\n"
                                " %s 10 dynamic 4
4\n\n", argv[0], argv[0], argv[0]);
            exit(-1);
        }else
            error(argv);
    }
}
}else if (argc==3){
    N = atoi(argv[1]); // Este sera el tamaño del vector y de las
    filas/columnas de la matriz
    strcpy(planificacion, argv[2]); // Esto es para poder capturar el texto
    desde consola
    sprintf(hebras, "%d", 12); // para atcgrid
    //sprintf(hebras, "%d", omp_get_num_procs()); // para cualquier procesador
}else if (argc==4){
    N = atoi(argv[1]); // Este sera el tamaño del vector y de las
    filas/columnas de la matriz
    strcpy(planificacion, argv[2]); // Esto es para poder capturar el texto
    desde consola
    strcpy(chunk, argv[3]); // Cual es el chunk del programa
    sprintf(hebras, "%d", 12); // para atcgrid
    sprintf(hebras, "%d", omp_get_num_procs()); // para cualquier procesador
}else if (argc==5){
    N = atoi(argv[1]); // Este sera el tamaño del vector y de las
    filas/columnas de la matriz
    strcpy(planificacion, argv[2]); // Esto es para poder capturar el texto
    desde consola
    strcpy(chunk, argv[3]); // Cual es el chunk del programa
    strcpy(hebras, argv[4]);
}

/* He elegido esta manera de asignar los valores al programa por que en OMP
 * en la escala de prioridad esta es la segunda opción osea que la unica
 * manera de poder cambiar la planificación del programa sería editando
 * el codigo y utilizando un if.
 */
if (!strcmp(statics, planificacion)){ // ponemos ! por que si las cadenas son
iguales el valor es 0
    if (strcmp(chunk, "")){
        strcat(statics, ",");
        strcat(statics, chunk);
    }
    setenv("OMP_SCHEDULE", statics, 1); // Elegimos como queremos la
planificación de bucles
}else if (!strcmp(dynamic, planificacion)){
    if (strcmp(chunk, "")){
        strcat(dynamic, ",");
        strcat(dynamic, chunk);
    }
    setenv("OMP_SCHEDULE", dynamic, 1); // Elegimos como queremos la
planificación de bucles
    setenv("OMP_DYNAMIC", "TRUE", 1); // Seteamos a true el ajuste dinámico del
nº de threads
}else if (!strcmp(guided, planificacion)){
    if (strcmp(chunk, "")){
        strcat(guided, ",");
        strcat(guided, chunk);
    }
    setenv("OMP_SCHEDULE", guided, 1); // Elegimos como queremos la

```

```

planificación de bucles
}
setenv("OMP_NUM_THREADS",hebras,1); // Seteamos el nº de threads en la siguiente
ejecución paralela

int **M1, **M2, **MR;
M1 = (int**) malloc(N * sizeof(int*));
for (f = 0; f < N; f++)
    M1[f] = (int*) malloc(N * sizeof(int));
M2 = (int**) malloc(N * sizeof(int*));
for (f = 0; f < N; f++)
    M2[f] = (int*) malloc(N * sizeof(int));
MR = (int**) malloc(N * sizeof(int*));
for (f = 0; f < N; f++)
    MR[f] = (int*) malloc(N * sizeof(int));
if ((M1 == NULL) || (M2 == NULL) || (MR == NULL)) {
    printf("Error en la reserva de espacio para los Vectores o MatrizTri\n");
    exit(-2);
}

srand(time(NULL)); // esta es la semilla que se usa para los random
#pragma omp parallel for schedule(runtime) private(f,c) shared(M1,M2) // Inicializamos la
Matrices
for(f = 0; f < N; f++)
    for(c = 0; c < N; c++){
        M1[f][c]=rand()%10;
        M2[f][c]=rand()%10;
    }

// imprimimos la matriz y el vector si el tamaño de N < PRINT_ALL_MIN
if (N <= PRINT_ALL_MIN && DEBUGMODE!=1){
    printf ("\nEsta es la matriz 1: \n");
    for (f = 0; f < N; f++){
        for (c = 0; c < N; c++){
            printf ("%d ",M1[f][c]);
        }
        printf ("\n");
    }
    printf ("\nEsta es la matriz 2: \n");
    for (f = 0; f < N; f++){
        for (c = 0; c < N; c++){
            printf ("%d ",M2[f][c]);
        }
        printf ("\n");
    }
    printf ("\n");
}

t1 = omp_get_wtime(); // Calcular la multiplicación de una matriz por un vector
#pragma omp parallel shared(M1,M2,MR)
{
    #pragma omp for schedule(runtime) private(f,c)
    for (f = 0; f < N; ++f) {
        for (c = 0; c < N; ++c) {
            for (k = 0; k < N; ++k) {
                MR[f][c] += M1[f][k] * M2[k][c];
            }
        }
    }
}
#pragma omp master
    if (omp_in_parallel())
        printf("Valores de las variables de control dentro del
parallel:\n"
              " dyn-var: %s\n"
              " nthreads-var: %s\n"
              " thread-limit-var: %s\n"
              " nest-var: %s\n"
              " run-sched-var:
%s\n",getenv("OMP_DYNAMIC"),getenv("OMP_NUM_THREADS"),
getenv("OMP_THREAD_LIMIT"),getenv("OMP_NESTED"),getenv("OMP_SCHEDULE"));
}
t2 = omp_get_wtime();
tr = t2 - t1; // Calculo el tiempo que he tardado en multiplicarlo

```

```

// Ahora imprimimos por pantalla los resultados obtenidos segun las
restricciones del problema
if (N <= PRINT_ALL_MIN && DEBUGMODE == 0){
    printf("Tiempo(seg.):%11.9f\nTamaño Matriz y Vector:%u\n",tr,N);// si
    queremos imprimir datos completos y N < PRINT_ALL_MIN
    printf ("Este es la matriz resultante: \n");
    for (f = 0; f < N; f++){
        for (c = 0; c < N; c++){
            printf ("%d ",MR[f][c]);
        }
        printf ("\n");
    }
    printf("\n");
}else if (DEBUGMODE==1) // si queremos imprimir unicamente el tiempo de cálculo
    printf("%11.9f\n",tr);//
else{ // y si queremos imprimir el tiempo la primera y la ultima multiplicación
    printf("Tiempo(seg.):%11.9f\n",tr);
    printf("Tamaño Matriz 1, Matriz 2 y Matriz resultante: %u\n",N);
    printf("(M1[0][0]=%d)*(M2[0][0]=%d)=%d\n",M1[0][0],M2[0][0],MR[0][0]);
    printf("(M1[%d][%d]=%d)*(M2[%d][%d]=%d)=%d\n",N-1,N-1,M1[N-1][N-1],N-1,N-1,
1,M2[N-1][N-1],MR[N-1][N-1]);
}

for(f=0; f<N; f++){
    free(M1[f]);
    free(M2[f]);
    free(MR[f]);
}
free(M1);
free(M2);
free(MR);
return 0;
}

```

CAPTURAS DE PANTALLA:

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 (N = 100, 1000 y 5000). 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.

El PC Local es un Intel(R) Core(TM)2 Quad CPU Q6600 2.40GHz con 4GB de Memoria DDR2 a 1333Mhz

Tabla 8 . Mediciones del estudio de Escalabilidad para PC local

Tamaño del Vector	Static Chunk 4 threads 4	Dynamic Chunk 4 threads 4
100	0,032563035	0,042390868
250	0,600749296	0,591736639
750	14,872883611	14,356715368
1500	121,25719312	120,110795253

Tabla 9 . Mediciones del estudio de Escalabilidad para ATCGRID

Tamaño del Vector	Static Chunk 12 threads 12	Dynamic Chunk 12 threads 12
100	0,01229889	0,013697952
250	0,143082251	0,179912577
750	5,864492989	4,571545386
1500	53,777203282	50,347473479

Tabla 10. Gráfica de Escalabilidad para PC local

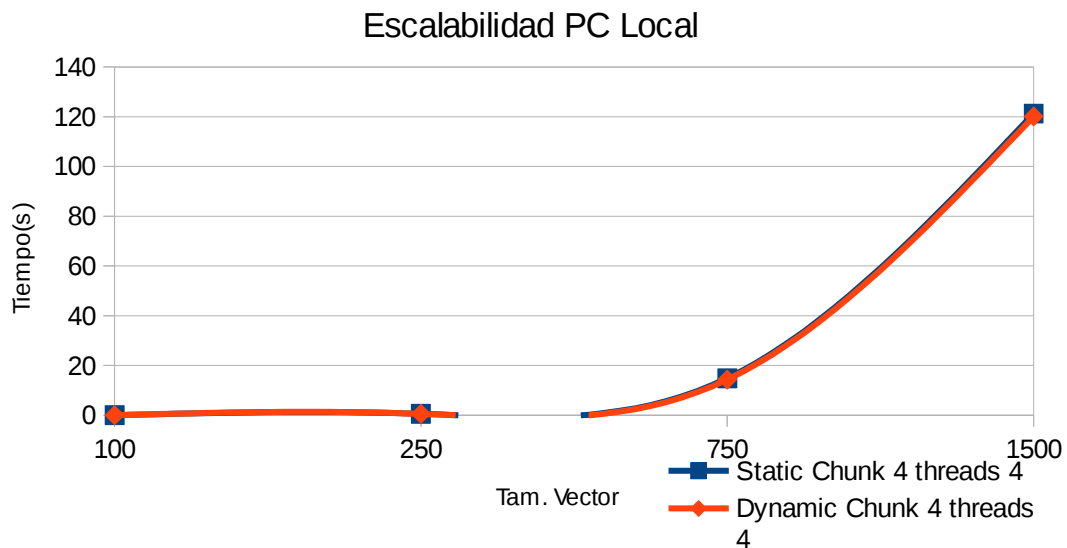
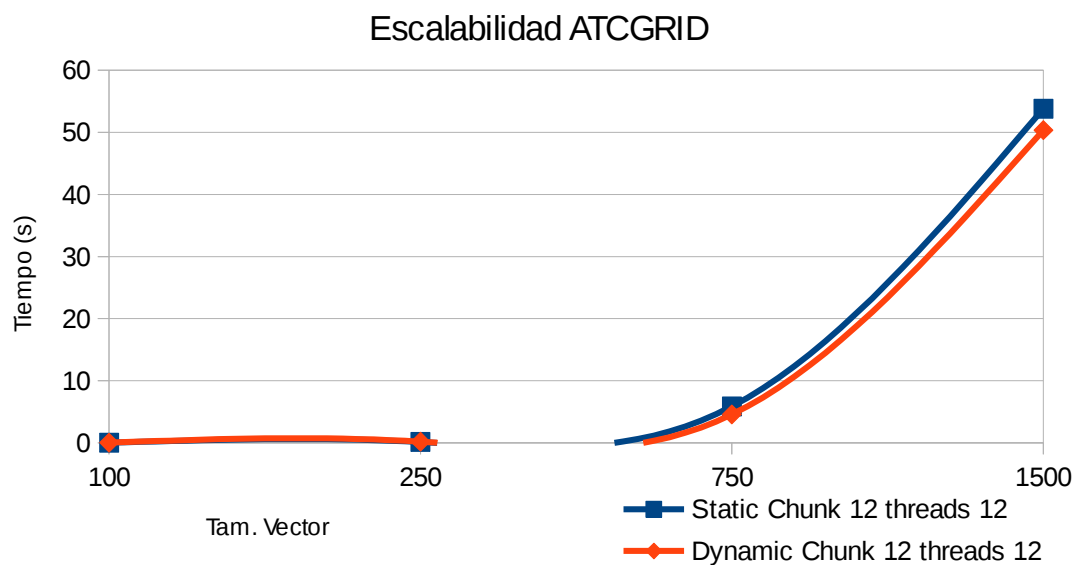


Tabla 11. Gráfica de Escalabilidad para ATCGRID



RESPUESTA:

Con este estudio de escalabilidad podemos confirmar lo que ya sabíamos de forma natural y es que por supuesto al utilizar mayor numero de procesadores el proceso de cálculo de la multiplicación de dos matrices se hace mas rápidamente, la particularidad que tiene este estudio empírico es que se puede ver fácilmente que la velocidad que conseguimos con el triple de procesadores es solamente el doble, con esto quiero decir que no por poner el doble de procesadores tendremos que conseguir el doble de velocidad