

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 OpenMP

Estudiante (nombre y apellidos): José María Martín Luque

Grupo de prácticas: 2

Fecha de entrega: 10-5-17

Fecha de evaluación en clase: 11-5-17

Ejercicios basados en los ejemplos del seminario práctico

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

Código fuente 1: `if-clauseModificado.c`

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <omp.h>
4 int main(int argc, char **argv)
5 {
6     int i, n=20, x, tid;
7     int a[n], suma=0, sumalocal;
8     if (argc < 3) {
9         fprintf(stderr, "[ERROR]-Falta iteraciones y/o número de threads\n");
10        exit(-1);
11    }
12    n = atoi(argv[1]);
13    x = atoi(argv[2]);
14
15    if (n > 20) n=20;
16    for (i = 0; i < n; i++) {
17        a[i] = i;
18    }
19
20    #pragma omp parallel if(n > 4) default(none) num_threads(x) private(sumalocal, tid) shared(a,
        suma, n)
21    {
22        sumalocal=0;
23        tid=omp_get_thread_num();
24
25        #pragma omp for private(i) schedule(static) nowait
```

```

26     for (i = 0; i < n; i++)
27     { sumalocal += a[i];
28         printf(" thread %d suma de a[%d]=%d sumalocal=%d \n", tid, i, a[i], sumalocal);
29     }
30
31     #pragma omp atomic
32     suma += sumalocal;
33
34     #pragma omp barrier
35
36     #pragma omp master
37     printf("thread master=%d imprime suma=%d\n", tid, suma);
38 }
39 }

```

Figura 1: Resultados

```

2.jmml@jose-torre:~/universidad/2/ac/p/s3 (ssh)
[jmml@jose-torre s3]$ ./bin/if-clauseModificado 10 2
thread 0 suma de a[0]=0 sumalocal=0
thread 0 suma de a[1]=1 sumalocal=1
thread 0 suma de a[2]=2 sumalocal=3
thread 0 suma de a[3]=3 sumalocal=6
thread 0 suma de a[4]=4 sumalocal=10
thread 1 suma de a[5]=5 sumalocal=5
thread 1 suma de a[6]=6 sumalocal=11
thread 1 suma de a[7]=7 sumalocal=18
thread 1 suma de a[8]=8 sumalocal=26
thread 1 suma de a[9]=9 sumalocal=35
thread master=0 imprime suma=45
[jmml@jose-torre s3]$ ./bin/if-clauseModificado 10 3
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 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 2 suma de a[7]=7 sumalocal=7
thread 2 suma de a[8]=8 sumalocal=15
thread 2 suma de a[9]=9 sumalocal=24
thread master=0 imprime suma=45
[jmml@jose-torre s3]$

```

Respuesta. La captura de pantalla ilustra que esta cláusula permite limitar el número de *threads* que utiliza el programa. Efectivamente, en la primera ejecución al poner el valor 2 el programa sólo utiliza 2 *threads* (0, 1), y en la segunda, con el valor 3, los *threads* 0, 1 y 2.

Ejercicio 2.

a) Rellena 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

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

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

Figura 2: Tablas `schedule`. 1, 2 y 4 representan el tamaño del *chunk*. Se han utilizado dos *threads*

schedule-clause.c				scheduled-clause.c				scheduleg-clause.c			
Iteración	1	2	4	Iteración	1	2	4	Iteración	1	2	4
1	0	0	0	1	0	0	1	1	1	1	0
2	0	0	0	2	0	0	1	2	1	1	0
3	0	0	0	3	0	0	1	3	1	1	0
4	0	0	0	4	0	0	1	4	1	1	0
5	0	0	0	5	0	0	1	5	1	1	0
6	0	0	0	6	0	0	1	6	1	1	0
7	0	0	0	7	0	0	1	7	1	1	0
8	0	1	0	8	0	0	1	8	1	1	0
9	1	1	1	9	0	0	1	9	0	0	0
10	1	1	1	10	1	0	1	10	0	0	0
11	1	1	1	11	1	0	1	11	0	0	0
12	1	1	1	12	1	0	1	12	0	0	0
13	1	1	1	13	1	1	1	13	0	0	1
14	1	1	1	14	1	1	1	14	0	0	1
15	1	1	1	15	1	1	0	15	0	0	1
16	1	0	1	16	0	1	0	16	0	0	1

Figura 3: Tablas schedule. 1, 2 y 4 representan el tamaño del *chunk*. Se han utilizado cuatro *threads*

schedule-clause.c					scheduled-clause.c					scheduleg-clause.c				
Iteración	1	2	4		Iteración	1	2	4		Iteración	1	2	4	
1	0	0	0		1	0	0	2		1	0	0	0	
2	0	0	0		2	0	0	2		2	0	0	0	
3	0	0	0		3	0	0	2		3	0	0	0	
4	0	0	0		4	3	0	2		4	0	0	0	
5	3	1	2		5	3	2	2		5	3	2	1	
6	3	1	2		6	3	2	2		6	0	2	1	
7	3	1	2		7	3	2	2		7	0	2	1	
8	3	1	2		8	3	2	2		8	0	2	1	
9	1	2	3		9	3	1	2		9	3	1	1	
10	1	2	3		10	3	1	2		10	3	1	2	
11	1	2	3		11	3	1	0		11	0	1	2	
12	1	2	3		12	3	1	0		12	1	1	2	
13	2	3	1		13	3	3	3		13	1	3	3	
14	2	3	1		14	1	3	3		14	2	3	3	
15	2	3	1		15	2	3	1		15	2	3	0	
16	2	3	1		16	0	3	1		16	2	3	0	

Respuesta: Las diferencias son las siguientes:

- *static* asigna los *threads* correspondientes a cada iteración mediante un algoritmo *round-robin*. La ventaja que tiene es que si se tienen dos cargas de trabajo que se distribuyen entre el mismo número de *threads* siempre le corresponderá el mismo rango de iteraciones a cada *thread*. Los bucles se dividen en *chunks* del mismo tamaño o similar en el caso de que haya un resto al dividir.
- *dynamic* asigna los *threads* por orden de llegada. De nuevo, los bucles se dividen en *chunks* del mismo tamaño.
- *guided* es similar a *dynamic* pero el *chunk* va reduciendo su tamaño en cada asignación. El parámetro *chunk* indica el tamaño mínimo que puede tener.

Ejercicio 3. Añade 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. Realiza varias ejecuciones usando variables de entorno para modificar estas variables de control antes de la ejecución. Incorpora en tu cuaderno de prácticas volcados de pantalla de estas ejecuciones. ¿Se imprimen valores distintos dentro y fuera de la región paralela?

Código fuente 2: `scheduled-clauseModificado.c`

```
1 #include <stdio.h>
2 #include <stdlib.h>
```

```

3  #ifdef _OPENMP
4  #include <omp.h>
5  #else
6  #define omp_get_thread_num() 0
7  #endif
8
9  int main(int argc, char **argv)
10 {
11     int i, n=200, chunk, a[n], suma=0;
12
13     omp_sched_t schedule_type;
14     int nchunk;
15
16     if(argc < 3) {
17         fprintf(stderr, "\nFalta iteraciones o chunk \n");
18         exit(-1);
19     }
20
21     n = atoi(argv[1]);
22     if (n>200)
23         n=200;
24     chunk = atoi(argv[2]);
25
26     for (i=0; i<n; i++)
27         a[i]=i;
28
29     #pragma omp parallel for firstprivate(suma) lastprivate(suma) schedule(dynamic,chunk)
30     for (i=0; i<n; i++) {
31         suma = suma + a[i];
32         printf(" thread %d suma a[%d]=%d suma=%d \n",
33             omp_get_thread_num(), i, a[i], suma);
34
35         if (omp_get_thread_num() == 0)
36         {
37             printf("Estoy dentro de parallel for \n");
38             omp_get_schedule(&schedule_type, &nchunk);
39             printf("dyn-var = %d, nthreads-var = %d, thread-limit-var: %d, run-sched-var = %d, chunk
40                 = %d \n", \
41                 omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit(), schedule_type,
42                 nchunk);
43         }
44
45         printf("Fuera de 'parallel for' suma=%d\n", suma);
46
47         omp_get_schedule(&schedule_type, &nchunk);
48         printf("dyn-var = %d, nthreads-var = %d, thread-limit-var: %d, run-sched-var = %d, chunk = %d \n"
49             , \
50             omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit(), schedule_type, nchunk);
51     }

```

Figura 4: Sin modificar variables de entorno

```
2. jmml@jose-torre:~/universidad/2/ac/p/s3 (ssh)
[jmml@jose-torre s3]$ ./bin/scheduled-clauseModificado 5 2
thread 1 suma a[0]=0 suma=0
thread 1 suma a[1]=1 suma=1
thread 0 suma a[2]=2 suma=2
thread 2 suma a[4]=4 suma=4
Estoy dentro de parallel for
dyn-var = 0, nthreads-var = 4, thread-limit-var: 2147483647, run-sched-var = 2, chunk = 1
thread 0 suma a[3]=3 suma=5
Estoy dentro de parallel for
dyn-var = 0, nthreads-var = 4, thread-limit-var: 2147483647, run-sched-var = 2, chunk = 1
Fuera de 'parallel for' suma=4
dyn-var = 0, nthreads-var = 4, thread-limit-var: 2147483647, run-sched-var = 2, chunk = 1
[jmml@jose-torre s3]$
```

Figura 5: Modificando OMP_NUM_THREADS

```
2. jmml@jose-torre:~/universidad/2/ac/p/s3 (ssh)
[jmml@jose-torre s3]$ export OMP_NUM_THREADS=2
[jmml@jose-torre s3]$ ./bin/scheduled-clauseModificado 5 2
thread 0 suma a[0]=0 suma=0
Estoy dentro de parallel for
thread 1 suma a[2]=2 suma=2
thread 1 suma a[3]=3 suma=5
dyn-var = 0, nthreads-var = 2, thread-limit-var: 2147483647, run-sched-var = 2, chunk = 1
thread 0 suma a[1]=1 suma=1
Estoy dentro de parallel for
dyn-var = 0, nthreads-var = 2, thread-limit-var: 2147483647, run-sched-var = 2, chunk = 1
thread 1 suma a[4]=4 suma=9
Fuera de 'parallel for' suma=9
dyn-var = 0, nthreads-var = 2, thread-limit-var: 2147483647, run-sched-var = 2, chunk = 1
[jmml@jose-torre s3]$
```

Figura 6: Modificando OMP_THREAD_LIMIT

```
2. jmm1@jose-torre:~/universidad/2/ac/p/s3 (ssh)
[jmm1@jose-torre s3]$ export OMP_THREAD_LIMIT=1
[jmm1@jose-torre s3]$ ./bin/scheduled-clauseModificado 5 2
thread 0 suma a[0]=0 suma=0
Estoy dentro de parallel for
dyn-var = 0, nthreads-var = 2, thread-limit-var: 1, run-sched-var = 2, chunk = 1
thread 0 suma a[1]=1 suma=1
Estoy dentro de parallel for
dyn-var = 0, nthreads-var = 2, thread-limit-var: 1, run-sched-var = 2, chunk = 1
thread 0 suma a[2]=2 suma=3
Estoy dentro de parallel for
dyn-var = 0, nthreads-var = 2, thread-limit-var: 1, run-sched-var = 2, chunk = 1
thread 0 suma a[3]=3 suma=6
Estoy dentro de parallel for
dyn-var = 0, nthreads-var = 2, thread-limit-var: 1, run-sched-var = 2, chunk = 1
thread 0 suma a[4]=4 suma=10
Estoy dentro de parallel for
dyn-var = 0, nthreads-var = 2, thread-limit-var: 1, run-sched-var = 2, chunk = 1
Fuera de 'parallel for' suma=10
dyn-var = 0, nthreads-var = 2, thread-limit-var: 1, run-sched-var = 2, chunk = 1
[jmm1@jose-torre s3]$
```

Respuesta: Como se puede observar, los resultados son los mismos dentro de la región paralela y fuera de la misma.

Ejercicio 4. Usa 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. Imprime los valores que obtienen estas funciones dentro (lo debe imprimir sólo uno de los threads) y fuera de la región paralela. Incorpora en tu cuaderno de prácticas volcados de pantalla con los resultados de ejecución obtenidos. Indica en qué funciones se obtienen valores distintos dentro y fuera de la región paralela.

Código fuente 3: scheduled-clauseModificado4.c

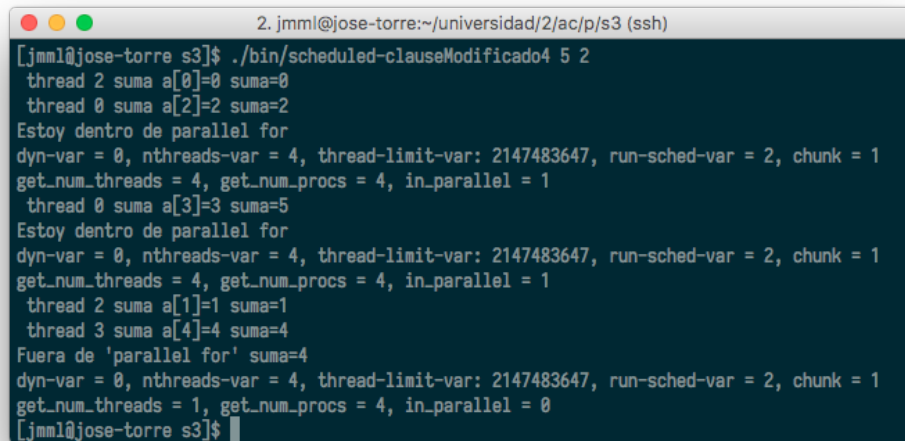
```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #ifdef _OPENMP
4 #include <omp.h>
5 #else
6 #define omp_get_thread_num() 0
7 #endif
8
9 int main(int argc, char **argv)
10 {
11     int i, n=200, chunk, a[n], suma=0;
12
13     omp_sched_t schedule_type;
14     int nchunk;
15 }
```

```

16     if (argc < 3) {
17         fprintf(stderr, "\nFalta iteraciones o chunk \n");
18         exit(-1);
19     }
20
21     n = atoi(argv[1]);
22     if (n > 200)
23         n=200;
24     chunk = atoi(argv[2]);
25
26     for (i = 0; i < n; i++)
27         a[i] = i;
28
29 #pragma omp parallel for firstprivate(suma) lastprivate(suma) schedule(dynamic, chunk)
30     for (i = 0; i < n; i++) {
31         suma = suma + a[i];
32         printf(" thread %d suma a[%d]=%d suma=%d \n",
33             omp_get_thread_num(), i, a[i], suma);
34
35         if (omp_get_thread_num() == 0)
36         {
37             printf("Estoy dentro de parallel for \n");
38             omp_get_schedule(&schedule_type, &nchunk);
39             printf("dyn-var = %d, nthreads-var = %d, thread-limit-var: %d, run-sched-var = %d, chunk
               = %d \n", \
40                 omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit(), schedule_type,
               nchunk);
41             printf("get_num_threads = %d, get_num_procs = %d, in_parallel = %d \n",
               omp_get_num_threads(), omp_get_num_procs(), omp_in_parallel());
42         }
43     }
44
45     printf("Fuera de 'parallel for' suma=%d\n", suma);
46
47     omp_get_schedule(&schedule_type, &nchunk);
48     printf("dyn-var = %d, nthreads-var = %d, thread-limit-var: %d, run-sched-var = %d, chunk = %d \n"
       , \
49         omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit(), schedule_type, nchunk);
50     printf("get_num_threads = %d, get_num_procs = %d, in_parallel = %d \n", omp_get_num_threads(),
       omp_get_num_procs(), omp_in_parallel());
51 }

```


Figura 7: Valores de las funciones dentro y fuera



```
2. jmm1@jose-torre:~/universidad/2/ac/p/s3 (ssh)
[jmm1@jose-torre s3]$ ./bin/scheduled-clauseModificado4 5 2
thread 2 suma a[0]=0 suma=0
thread 0 suma a[2]=2 suma=2
Estoy dentro de parallel for
dyn-var = 0, nthreads-var = 4, thread-limit-var: 2147483647, run-sched-var = 2, chunk = 1
get_num_threads = 4, get_num_procs = 4, in_parallel = 1
thread 0 suma a[3]=3 suma=5
Estoy dentro de parallel for
dyn-var = 0, nthreads-var = 4, thread-limit-var: 2147483647, run-sched-var = 2, chunk = 1
get_num_threads = 4, get_num_procs = 4, in_parallel = 1
thread 2 suma a[1]=1 suma=1
thread 3 suma a[4]=4 suma=4
Fuera de 'parallel for' suma=4
dyn-var = 0, nthreads-var = 4, thread-limit-var: 2147483647, run-sched-var = 2, chunk = 1
get_num_threads = 1, get_num_procs = 4, in_parallel = 0
[jmm1@jose-torre s3]$
```

Respuesta: Las variables que cambian son `get_num_threads` e `in_parallel`. Era de esperar ya que al salir de la región paralela el número de *threads* será 1 y el indicador, 0.

Ejercicio 5. Añade 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. Incorpora en tu cuaderno de prácticas volcados de pantalla con los resultados de ejecución obtenidos.

Código fuente 4: `scheduled-clauseModificado5.c`

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #ifdef _OPENMP
4 #include <omp.h>
5 #else
6 #define omp_get_thread_num() 0
7 #endif
8
9 int main(int argc, char **argv)
10 {
11     int i, n=200, chunk, a[n], suma=0;
12
13     omp_sched_t schedule_type;
14     int nchunk;
15
16     if (argc < 3) {
17         fprintf(stderr, "\nFalta iteraciones o chunk \n");
18         exit(-1);
19     }
20 }
```

```

21     n = atoi(argv[1]);
22     if (n > 200)
23         n=200;
24     chunk = atoi(argv[2]);
25
26     for (i = 0; i < n; i++)
27         a[i] = i;
28
29     printf("Antes de cambiar: \n");
30     omp_get_schedule(&schedule_type, &nchunk);
31     printf("dyn-var = %d, nthreads-var = %d, thread-limit-var: %d, run-sched-var = %d, chunk = %d \n"
32           , \
33           omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit(), schedule_type, nchunk);
34     printf("get_num_threads = %d, get_num_procs = %d, in_parallel = %d \n", omp_get_num_threads(),
35           omp_get_num_procs(), omp_in_parallel());
36
37     omp_set_dynamic(2);
38     omp_set_num_threads(2);
39     omp_set_schedule(1, 1);
40 #pragma omp parallel for firstprivate(suma) lastprivate(suma) schedule(dynamic, chunk)
41     for (i = 0; i < n; i++) {
42         suma = suma + a[i];
43         printf(" thread %d suma a[%d]=%d suma=%d \n",
44               omp_get_thread_num(),i,a[i],suma);
45     }
46
47     printf("Fuera de 'parallel for' suma=%d\n",suma);
48
49     printf("Los valores ya han cambiado: \n");
50     omp_get_schedule(&schedule_type, &nchunk);
51     printf("dyn-var = %d, nthreads-var = %d, thread-limit-var: %d, run-sched-var = %d, chunk = %d \n"
52           , \
53           omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit(), schedule_type, nchunk);
54     printf("get_num_threads = %d, get_num_procs = %d, in_parallel = %d \n", omp_get_num_threads(),
55           omp_get_num_procs(), omp_in_parallel());
56 }

```

Figura 8: Valores de las variables



```
2. jmm1@jose-torre:~/universidad/2/ac/p/s3 (ssh)
[jmm1@jose-torre s3]$ ./bin/scheduled-clauseModificado5 5 2
Antes de cambiar:
dyn-var = 0, nthreads-var = 4, thread-limit-var: 2147483647, run-sched-var = 2, chunk = 1
get_num_threads = 1, get_num_procs = 4, in_parallel = 0
thread 0 suma a[0]=0 suma=0
thread 0 suma a[1]=1 suma=1
thread 0 suma a[4]=4 suma=5
thread 1 suma a[2]=2 suma=2
thread 1 suma a[3]=3 suma=5
Fuera de 'parallel for' suma=5
Los valores ya han cambiado:
dyn-var = 1, nthreads-var = 2, thread-limit-var: 2147483647, run-sched-var = 1, chunk = 1
get_num_threads = 1, get_num_procs = 4, in_parallel = 0
[jmm1@jose-torre s3]$
```

Resto de ejercicios

Ejercicio 6. Implementa un programa secuencial en C que multiplique una matriz triangular por un vector (use variables dinámicas). Compara el orden de complejidad del código que has implementado con el código que implementaste para el producto matriz por vector.

Notas.

- El número de filas/columnas debe ser un argumento de entrada
- Se debe inicializar las matrices antes del cálculo
- Se debe imprimir siempre la primera y última componente del resultado antes de que termine el programa.

Código fuente 5: pmtv-secuencial.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #ifdef _OPENMP
4     #include <omp.h>
5 #else
6     #define omp_get_thread_num() 0
7 #endif
8
9 int main(int argc, char **argv)
10 {
11     int i, j;
12
13     if (argc < 2)
14     {
15         printf("Falta indicar el tamaño\n");
```

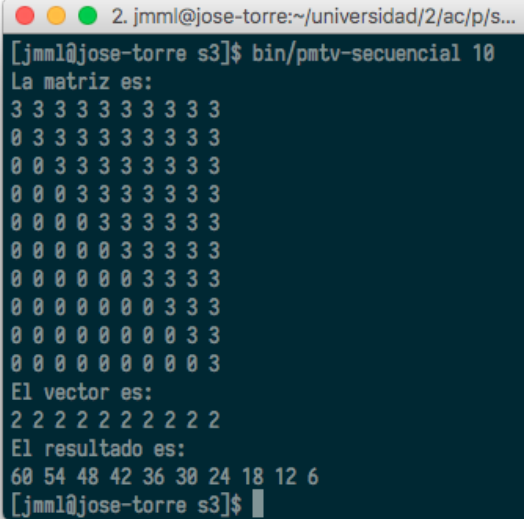
```

16     }
17
18     unsigned int N = atoi(argv[1]);
19
20     int *vector, *res, **matriz;
21     vector = (int*) malloc(N * sizeof(int));
22     res = (int*) malloc(N * sizeof(int));
23     matriz = (int**) malloc(N * sizeof(int*));
24
25     for (int i = 0; i < N; i++)
26         matriz[i] = (int*) malloc (N * sizeof(int));
27
28     for (int i = 0; i < N; i++)
29     {
30         for (j = i; j < N; j++)
31             matriz[i][j] = 3;
32         vector[i] = 2;
33         res[i] = 0;
34     }
35
36     printf("La matriz es: \n");
37     for (i = 0; i < N; i++)
38     {
39         for (j = 0; j < N; j++)
40         {
41             if (j >= i)
42                 printf("%d ", matriz[i][j]);
43             else
44                 printf("0 ");
45         }
46         printf("\n");
47     }
48
49     printf("El vector es: \n");
50     for (i = 0; i < N; ++i) {
51         printf("%d ", vector[i]);
52     }
53     printf("\n");
54
55     for (i = 0; i < N; ++i) {
56         for (j = i; j < N; ++j) {
57             res[i] += matriz[i][j] * vector[j];
58         }
59     }
60
61     printf("El resultado es: \n");
62     for (i = 0; i < N; ++i) {
63         printf("%d ", res[i]);
64     }
65     printf("\n");
66
67     for (i = 0; i < N; ++i) {
68         free(matriz[i]);
69     }
70     free(matriz);
71     free(vector);

```

```
72 | free(res);  
73 | }
```

Figura 9: Resultados para matriz 10×10



```
2. jmml@jose-torre:~/universidad/2/ac/p/s...  
[jmml@jose-torre s3]$ bin/pmtv-secuencial 10  
La matriz es:  
3 3 3 3 3 3 3 3 3 3  
0 3 3 3 3 3 3 3 3 3  
0 0 3 3 3 3 3 3 3 3  
0 0 0 3 3 3 3 3 3 3  
0 0 0 0 3 3 3 3 3 3  
0 0 0 0 3 3 3 3 3 3  
0 0 0 0 0 3 3 3 3 3  
0 0 0 0 0 0 3 3 3 3  
0 0 0 0 0 0 0 3 3 3  
0 0 0 0 0 0 0 0 3 3  
0 0 0 0 0 0 0 0 0 3  
El vector es:  
2 2 2 2 2 2 2 2 2 2  
El resultado es:  
60 54 48 42 36 30 24 18 12 6  
[jmml@jose-torre s3]$
```

Figura 10: Resultados para matriz 20×20

```
2. jmm1@jose-torre:~/universidad/2/ac/p/s3 (ssh)
[jmm1@jose-torre s3]$ bin/pmtv-secuencial 20
La matriz es:
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
0 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
0 0 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
0 0 0 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
0 0 0 0 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
0 0 0 0 0 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
0 0 0 0 0 0 3 3 3 3 3 3 3 3 3 3 3 3 3 3
0 0 0 0 0 0 0 3 3 3 3 3 3 3 3 3 3 3 3 3
0 0 0 0 0 0 0 0 3 3 3 3 3 3 3 3 3 3 3 3
0 0 0 0 0 0 0 0 0 3 3 3 3 3 3 3 3 3 3 3
0 0 0 0 0 0 0 0 0 0 3 3 3 3 3 3 3 3 3 3
0 0 0 0 0 0 0 0 0 0 0 3 3 3 3 3 3 3 3 3
0 0 0 0 0 0 0 0 0 0 0 0 3 3 3 3 3 3 3 3
0 0 0 0 0 0 0 0 0 0 0 0 0 3 3 3 3 3 3 3
0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 3 3 3 3 3
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 3 3 3 3
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 3 3 3
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 3 3
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 3
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3
El vector es:
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
El resultado es:
120 114 108 102 96 90 84 78 72 66 60 54 48 42 36 30 24 18 12 6
[jmm1@jose-torre s3]$
```

Respuesta: El orden de complejidad de ambos algoritmos es $O(n^2)$.

Código fuente 6: Producto de matriz por vector

```
1   for (i = 0; i < N; i++)
2       for (j = 0; j < N; j++)
3           v2[i] += m[i][j] * v1[j];
```

Tenemos dos bucles de tamaño N que realizan una operación $O(1)$. Luego el orden de complejidad es $O(1) \cdot N \cdot N = O(N^2)$.

Código fuente 7: Producto de matriz triangular por vector

```
1   for (i = 0; i < N; ++i)
2       for (j = i; j < N; ++j)
3           res[i] += matriz[i][j] * vector[j];
```

De nuevo tenemos dos bucles, pero en este caso uno es de tamaño N y el otro de tamaño $\frac{N}{2}$ dado que la matriz es triangular. Aún así el orden de complejidad es $O(1) \cdot N \cdot \frac{N}{2} = O(N^2)$.

Ejercicio 7. Implementa 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. Dibuja 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). Añade lo necesario para que el usuario pueda fijar la planificación de tareas usando la variable de entorno `OMP_SCHEDULE`. Obtén 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. Usa 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. Representa el tiempo para `static`, `dynamic` y `guided` en función del tamaño del chunk en una gráfica. ¿Qué alternativa ofrece mejores prestaciones? Razona por qué. Incluye los scripts utilizado en el cuaderno de prácticas.

Nota: Nunca ejecutes en `atcgrid` código que imprima todos los componentes del resultado.

Contesta a las siguientes preguntas:

- ¿Qué valor por defecto usa OpenMP para chunk con `static`, `dynamic` y `guided`? Indica qué ha hecho para obtener este valor por defecto para cada alternativa.
- ¿Qué número de operaciones de multiplicación y suma realizan cada uno de los threads en la asignación `static` para cada uno de los chunks?
- Con la asignación `dynamic` y `guided`, ¿qué crees que debe ocurrir con el número de operaciones de multiplicación y suma que realizan cada uno de los threads?

Código fuente 8: `pmtv-OpenMP.c`

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #ifdef _OPENMP
4  #include <omp.h>
5  #else
6  #define omp_get_thread_num() 0
7  #define omp_get_num_threads() 1
8  #define omp_set_num_threads(int)
9  #define omp_in_parallel() 0
10 #define omp_set_dynamic(int)
11 #endif
12
13 int main(int argc, char **argv)
14 {
15     int i, j, t=1;
16
17     if (argc < 2) {
18         printf("Falta el tamaño\n");
19         exit(-1);
20     }
21
22     unsigned int N = atoi(argv[1]);

```

```

23
24     if (argc == 3) {
25         t = atoi(argv[2]);
26     }
27
28     int *vector, *res, **matriz;
29     vector = (int*) malloc(N * sizeof(int));
30     res = (int*) malloc(N * sizeof(int));
31     matriz = (int**) malloc(N * sizeof(int*));
32
33     for (int i = 0; i < N; i++)
34         matriz[i] = (int*) malloc (N * sizeof(int));
35
36     for (int i = 0; i < N; i++)
37     {
38         for (j = i; j < N; j++)
39             matriz[i][j] = 3;
40         vector[i] = 2;
41         res[i] = 0;
42     }
43
44     if (t == 0) {
45         printf("La matriz es: \n");
46         for (i = 0; i < N; i++)
47         {
48             for (j = 0; j < N; j++)
49             {
50                 if (j >= i)
51                     printf("%d ", matriz[i][j]);
52                 else
53                     printf("0 ");
54             }
55             printf("\n");
56         }
57
58         printf("El vector es: \n");
59         for (i = 0; i < N; ++i) {
60             printf("%d ", vector[i]);
61         }
62         printf("\n");
63
64     }
65
66     double inicio, fin, total;
67     inicio = omp_get_wtime();
68
69 #pragma omp parallel for private(j) schedule(runtime)
70     for (i = 0; i < N; ++i) {
71         for (j = i; j < N; ++j) {
72             res[i] += matriz[i][j] * vector[j];
73         }
74     }
75
76     fin = omp_get_wtime();
77     total = fin - inicio;
78

```



```

79     if (t == 0) {
80         printf("El resultado es: \n");
81         for (i = 0; i < N; ++i) {
82             printf("%d ", res[i]);
83         }
84         printf("\n");
85     } else if (t == 1) {
86         printf("%.4f\n", total);
87     }
88
89     for (i = 0; i < N; ++i) {
90         free(matriz[i]);
91     }
92     free(matriz);
93     free(vector);
94     free(res);
95
96 }

```

Figura 11: Resultados para matriz 10×10

```

2. jmm1@jose-torre:~/universidad/2/ac/p/...
[jmm1@jose-torre s3]$ ./bin/pmtv-OpenMP 10 0
La matriz es:
3 3 3 3 3 3 3 3 3 3
0 3 3 3 3 3 3 3 3 3
0 0 3 3 3 3 3 3 3 3
0 0 0 3 3 3 3 3 3 3
0 0 0 0 3 3 3 3 3 3
0 0 0 0 0 3 3 3 3 3
0 0 0 0 0 0 3 3 3 3
0 0 0 0 0 0 0 3 3 3
0 0 0 0 0 0 0 0 3 3
0 0 0 0 0 0 0 0 0 3
0 0 0 0 0 0 0 0 0 3
El vector es:
2 2 2 2 2 2 2 2 2 2
El resultado es:
60 54 48 42 36 30 24 18 12 6
[jmm1@jose-torre s3]$

```

Figura 12: Resultados para matriz 20×20

```
2. jmm1@jose-torre:~/universidad/2/ac/p/s3 (ssh)
[jmm1@jose-torre s3]$ ./bin/pmtv-OpenMP 20 0
La matriz es:
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
0 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
0 0 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
0 0 0 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
0 0 0 0 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
0 0 0 0 0 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
0 0 0 0 0 0 3 3 3 3 3 3 3 3 3 3 3 3 3 3
0 0 0 0 0 0 0 3 3 3 3 3 3 3 3 3 3 3 3 3
0 0 0 0 0 0 0 0 3 3 3 3 3 3 3 3 3 3 3 3
0 0 0 0 0 0 0 0 0 3 3 3 3 3 3 3 3 3 3 3
0 0 0 0 0 0 0 0 0 0 3 3 3 3 3 3 3 3 3 3
0 0 0 0 0 0 0 0 0 0 0 3 3 3 3 3 3 3 3 3
0 0 0 0 0 0 0 0 0 0 0 0 3 3 3 3 3 3 3 3
0 0 0 0 0 0 0 0 0 0 0 0 0 3 3 3 3 3 3 3
0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 3 3 3 3 3
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 3 3 3 3
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 3 3 3
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 3 3
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 3
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3
El vector es:
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
El resultado es:
120 114 108 102 96 90 84 78 72 66 60 54 48 42 36 30 24 18 12 6
[jmm1@jose-torre s3]$
```

Código fuente 9: pmtv-OpenMP-atcgrid.sh

```
1 #!/bin/bash
2
3 #Se asigna al trabajo el nombre
4 #PBS -N pmtv-OpenMP
5 #Se asigna al trabajo la cola ac
6 #PBS -q ac
7
8 #Se imprime información del trabajo usando variables de entorno de PBS
9 echo "Id. usuario del trabajo: $PBS_O_LOGNAME"
10 echo "Id. del trabajo: $PBS_JOBID"
11 echo "Nombre del trabajo especificado por usuario: $PBS_JOBNAME"
12 echo "Nodo que ejecuta qsub: $PBS_O_HOST"
13 echo "Cola: $PBS_QUEUE"
14 echo "Nodos asignados al trabajo:"
15 cat $PBS_NODEFILE
16 echo "Directorio de trabajo: $PBS_O_WORKDIR"
17
18
19 echo -e "Por-defecto\n1\n64" > $PBS_O_WORKDIR/dat/tmp0.dat
20
```

```

21 export OMP_SCHEDULE="static"
22 echo "static con chunk por defecto"
23 $PBS_O_WORKDIR/bin/pmtv-OpenMP 15360 > /dev/null > $PBS_O_WORKDIR/dat/tmp1.dat
24
25 export OMP_SCHEDULE="static,1"
26 echo "static con chunk 1"
27 $PBS_O_WORKDIR/bin/pmtv-OpenMP 15360 > /dev/null >> $PBS_O_WORKDIR/dat/tmp1.dat 2>81
28
29 export OMP_SCHEDULE="static,64"
30 echo "static con chunk 64"
31 $PBS_O_WORKDIR/bin/pmtv-OpenMP 15360 > /dev/null >> $PBS_O_WORKDIR/dat/tmp1.dat 2>81
32
33 export OMP_SCHEDULE="dynamic"
34 echo "dynamic con chunk por defecto"
35 $PBS_O_WORKDIR/bin/pmtv-OpenMP 15360 > /dev/null > $PBS_O_WORKDIR/dat/tmp2.dat 2>81
36
37 export OMP_SCHEDULE="dynamic,1"
38 echo "dynamic con chunk 1"
39 $PBS_O_WORKDIR/bin/pmtv-OpenMP 15360 > /dev/null >> $PBS_O_WORKDIR/dat/tmp2.dat 2>81
40
41 export OMP_SCHEDULE="dynamic,64"
42 echo "dynamic con chunk 64"
43 $PBS_O_WORKDIR/bin/pmtv-OpenMP 15360 > /dev/null >> $PBS_O_WORKDIR/dat/tmp2.dat 2>81
44
45 export OMP_SCHEDULE="guided"
46 echo "guided con chunk por defecto"
47 $PBS_O_WORKDIR/bin/pmtv-OpenMP 15360 > /dev/null > $PBS_O_WORKDIR/dat/tmp3.dat 2>81
48
49 export OMP_SCHEDULE="guided,1"
50 echo "guided con chunk 1"
51 $PBS_O_WORKDIR/bin/pmtv-OpenMP 15360 > /dev/null >> $PBS_O_WORKDIR/dat/tmp3.dat 2>81
52
53 export OMP_SCHEDULE="guided,64"
54 echo "guided con chunk 64"
55 $PBS_O_WORKDIR/bin/pmtv-OpenMP 15360 > /dev/null >> $PBS_O_WORKDIR/dat/tmp3.dat 2>81
56
57 paste $PBS_O_WORKDIR/dat/tmp0.dat $PBS_O_WORKDIR/dat/tmp1.dat $PBS_O_WORKDIR/dat/tmp2.dat
58     $PBS_O_WORKDIR/dat/tmp3.dat > $PBS_O_WORKDIR/dat/pmtv-OpenMP1.dat
59
60 echo -e "Chunk\tStatic\tDynamic\tGuided\n$(cat $PBS_O_WORKDIR/dat/pmtv-OpenMP1.dat)" > $PBS_O_WORKDIR
61 /dat/pmtv-OpenMP1.dat
62
63 rm $PBS_O_WORKDIR/dat/tmp0.dat $PBS_O_WORKDIR/dat/tmp1.dat $PBS_O_WORKDIR/dat/tmp2.dat $PBS_O_WORKDIR
64 /dat/tmp3.dat

```

Figura 13: Tiempos de ejecución (primera vez) de la versión paralela del producto de una matriz triangular por un vector r para vectores de tamaño $N = 12$ threads

Chunk	Static	Dynamic	Guided
Por-defecto	0.047s	0.046s	0.049s
1	0.052s	0.044s	0.052s
64	0.055s	0.043s	0.054s

Figura 14: Tiempos de ejecución (segunda vez) de la versión paralela del producto de una matriz triangular por un vector r para vectores de tamaño $N = 12$ threads

Chunk	Static	Dynamic	Guided
Por-defecto	0.056s	0.043s	0.046s
1	0.050s	0.043s	0.046s
64	0.046s	0.042s	0.048s

Figura 15: Primera ejecución

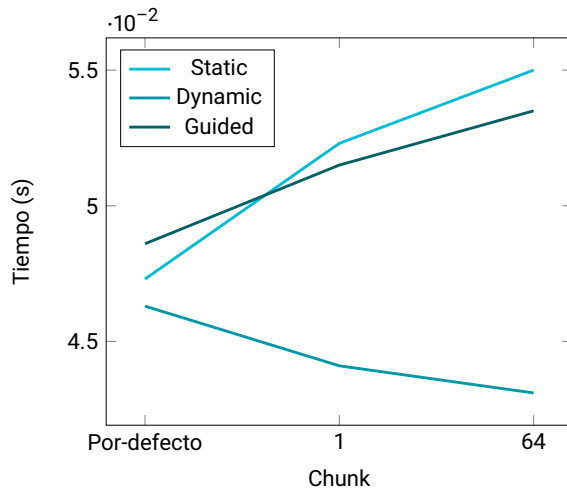
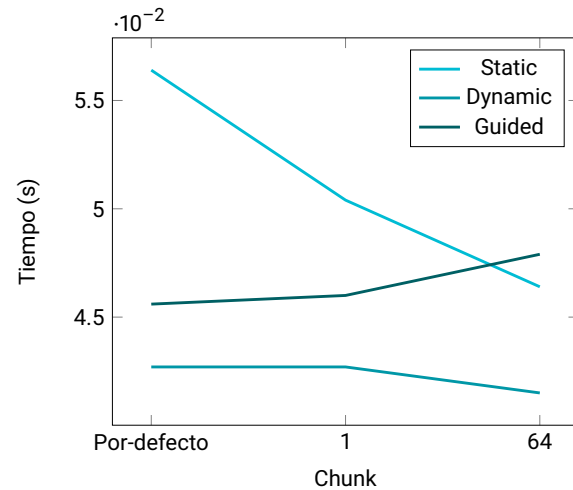


Figura 16: Segunda ejecución



Respuestas:

- El valor por defecto de chunk en static es $\frac{\text{número de iteraciones}}{\text{número de threads}}$, el de dynamic es 1 y el de guided, de nuevo $\frac{\text{número de iteraciones}}{\text{número de threads}}$. La información la he obtenido en la documentación de OpenMP.
- Para el chunk por defecto, cada thread hace $\frac{N}{12} \cdot 2$ (2 es el número de sumas y multiplicaciones por iteración). Con el chunk = 1, de nuevo $\frac{N}{12} \cdot 2$ operaciones y con chunk = 64, realizará como mínimo $\frac{N}{64} \cdot \frac{1}{12} \cdot 2$ operaciones. En este último caso lo que se ha hecho es dividir $\frac{\text{número de iteraciones}}{\text{tamaño de chunk}}$ para saber cuántos chunks habrá en total. Entonces como mínimo, cada thread ejecutará $\frac{\text{número de chunks}}{\text{número de threads}}$. Los chunks sobrantes se repartirán entre el total de threads.
- Variará en distintas ejecuciones ya que no están asignados de forma fija y depende del tiempo que tarde cada thread en realizar las operaciones.

A la vista de los datos la alternativa que ofrece mejores prestaciones es dynamic. Puede deberse al hecho de que a cada thread se le asigna una iteración en cuanto acaba la que tenía asignada en lugar de tenerlas asignadas desde un principio. De esta forma, si algún thread acaba más rápido que otro, puede pasar a la siguiente iteración.

Ejercicio 8. Implementa un programa secuencial en C que calcule la multiplicación de matrices cuadra-

das, B y C:

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

Notas.

- El número de filas/columnas debe ser un argumento de entrada.
- Se debe inicializar las matrices antes del cálculo.
- Se debe imprimir siempre las componentes (0,0) y (N - 1, N - 1) del resultado antes de que termine el programa.

Código fuente 10: pmm-secuencial.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4
5 int main(int argc, char **argv) {
6     unsigned i, j, k, d = 1;
7
8     if (argc < 2) {
9         fprintf(stderr, "falta size\n");
10        exit(-1);
11    } else if (argc == 3) {
12        d = atoi(argv[2]);
13    }
14
15    unsigned int N = atoi(argv[1]);
16
17    int **a, **b, **c;
18    a = (int **) malloc(N*sizeof(int*));
19    b = (int **) malloc(N*sizeof(int*));
20    c = (int **) malloc(N*sizeof(int*));
21
22    for (i=0; i<N; i++) {
23        a[i] = (int *) malloc(N*sizeof(int));
24        b[i] = (int *) malloc(N*sizeof(int));
25        c[i] = (int *) malloc(N*sizeof(int));
26    }
27
28    for (i=0; i<N; i++) {
29        for (j=0; j<N; j++) {
30            a[i][j] = 2;
31            b[i][j] = 3;
32            c[i][j] = 4;
33        }
34    }
35
36    struct timespec cgt1, cgt2;
37    double ncgt;
38
39    clock_gettime(CLOCK_REALTIME, &cgt1);
40
```

```

41     for (i=0; i<N; i++)
42         for (j=0; j<N; j++)
43             for (k=0; k<N; k++)
44                 a[i][j] += b[i][k] * c[k][j];
45
46     clock_gettime(CLOCK_REALTIME,&cgt2);
47
48     ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec) + (double) ((cgt2.tv_nsec-cgt1.tv_nsec) / (1.e+9));
49
50     if (d) {
51         printf("%.4f\n", ncgt);
52     } else {
53         printf("Tiempo de ejecución = %.4f\t (0, 0) = %d\t (N-1, N-1) = %d\n", ncgt, a[0][0], a[N-1][
54             N-1]);
55     }
56
57     for (i=0; i<N; i++) {
58         free(a[i]);
59         free(b[i]);
60         free(c[i]);
61     }
62
63     free(a);
64     free(b);
65     free(c);
66 }

```

Figura 17: Resultados para varios tamaños

```

2. jmml@jose-torre:~/universidad/2/ac/p/s3 (ssh)
[jmml@jose-torre s3]$ ./bin/pmm-secuencial 50 0
Tiempo de ejecución = 0.0002      (0, 0) = 602      (N-1, N-1) =602
[jmml@jose-torre s3]$ ./bin/pmm-secuencial 100 0
Tiempo de ejecución = 0.0017      (0, 0) = 1202     (N-1, N-1) =1202
[jmml@jose-torre s3]$ ./bin/pmm-secuencial 500 0
Tiempo de ejecución = 0.6706      (0, 0) = 6002     (N-1, N-1) =6002
[jmml@jose-torre s3]$ ./bin/pmm-secuencial 1000 0
Tiempo de ejecución = 7.0314      (0, 0) = 12002    (N-1, N-1) =12002
[jmml@jose-torre s3]$

```

Ejercicio 9. Implementa en paralelo la multiplicación de matrices cuadradas con OpenMP a partir del código escrito en el ejercicio anterior. Usa las directivas, las cláusulas y las funciones de entorno que consideres oportunas. Se debe paralelizar también la inicialización de las matrices. Dibuja en su cua-

dermo de prácticas la descomposición de dominio que ha utilizado en el código paralelo implementado para asignar tareas a los threads (Lección 4/Tema 2, Lección 5/Tema 2).

Código fuente 11: pmm-OpenMP.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #ifdef _OPENMP
5      #include <omp.h>
6  #else
7      #define omp_get_thread_num() 0
8      #define omp_get_num_threads() 1
9      #define omp_set_num_threads(int)
10     #define omp_in_parallel() 0
11     #define omp_set_dynamic(int)
12 #endif
13
14 int main(int argc, char **argv)
15 {
16     unsigned i, j, k, d = 1;
17
18     if(argc < 2) {
19         fprintf(stderr, "falta size\n");
20         exit(-1);
21     } else if (argc == 3) {
22         d = atoi(argv[2]);
23     }
24
25     unsigned int N = atoi(argv[1]);
26
27     int **a, **b, **c;
28     a = (int **) malloc(N*sizeof(int*));
29     b = (int **) malloc(N*sizeof(int*));
30     c = (int **) malloc(N*sizeof(int*));
31
32     for (i=0; i<N; i++) {
33         a[i] = (int *) malloc(N*sizeof(int));
34         b[i] = (int *) malloc(N*sizeof(int));
35         c[i] = (int *) malloc(N*sizeof(int));
36     }
37
38     #pragma omp parallel for private(j)
39     for (i=0; i<N; i++) {
40         for (j=0; j<N; j++) {
41             a[i][j] = 2;
42             b[i][j] = 3;
43             c[i][j] = 4;
44         }
45     }
46
47     double start, end, total;
48     start = omp_get_wtime();
49
50     #pragma omp parallel for private(k,j)
```

```

51     for (i=0; i<N; i++)
52         for (j=0; j<N; j++)
53             for (k=0; k<N; k++)
54                 a[i][j] += b[i][k] * c[k][j];
55
56     end = omp_get_wtime();
57
58     total = end - start;
59
60     if (d) {
61         printf("%.4f\n", total);
62     } else {
63         printf("Tiempo de ejecución = %.4f\t (0, 0) = %d\t (N-1, N-1)=%d\n", total, a[0][0], a[N-1][
64             N-1]);
65     }
66
67     for (i=0; i<N; i++) {
68         free(a[i]);
69         free(b[i]);
70         free(c[i]);
71     }
72
73     free(a);
74     free(b);
75     free(c);
76 }

```

Figura 18: Resultados para varios tamaños

```

2. jmm1@jose-torre:~/universidad/2/ac/p/s3 (ssh)
[jmm1@jose-torre s3]$ ./bin/pmm-OpenMP 50 0
Tiempo de ejecución = 0.0043      (0, 0) = 602      (N-1, N-1)=602
[jmm1@jose-torre s3]$ ./bin/pmm-OpenMP 100 0
Tiempo de ejecución = 0.0005      (0, 0) = 1202     (N-1, N-1)=1202
[jmm1@jose-torre s3]$ ./bin/pmm-OpenMP 500 0
Tiempo de ejecución = 0.1728      (0, 0) = 6002     (N-1, N-1)=6002
[jmm1@jose-torre s3]$ ./bin/pmm-OpenMP 1000 0
Tiempo de ejecución = 1.8714      (0, 0) = 12002    (N-1, N-1)=12002
[jmm1@jose-torre s3]$

```

Ejercicio 10. Haz 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. Debes recordar usar `-O2` al compilar. Presenta los resultados del estudio en tablas de valores y en gráficas. Escoge los tamaños de manera que se observen diferentes curvas de escalabilidad en las gráficas que entregues en tu cuaderno de prácticas (prueba con valores de N entre 100 y 1500). Consulta la Lección

6/Tema 2. Incluye los scripts utilizado en el cuaderno de prácticas.

Nota: Nunca ejecute en atcgrid código que imprima todos los componentes del resultado.

Figura 19: Estudio de escalabilidad en el PC local

Elementos	Secuencial	2 threads	4 threads
100	0.002s	0.001s	0.000s
300	0.051s	0.029s	0.014s
500	0.673s	0.339s	0.172s
700	1.675s	0.850s	0.429s
900	4.794s	2.476s	1.260s
1100	9.506s	4.966s	2.604s
1300	16.141s	8.249s	4.248s
1500	28.974s	14.288s	7.791s

Figura 20: Estudio de escalabilidad en atcgrid

Elementos	Secuencial	2 threads	4 threads
100	0.002s	0.001s	0.001s
200	0.017s	0.011s	0.006s
300	0.051s	0.029s	0.014s
400	0.124s	0.067s	0.035s
500	0.349s	0.182s	0.120s
600	1.858s	0.910s	0.459s
700	2.856s	1.397s	0.734s
800	5.430s	2.657s	1.337s

Figura 21: Estudio de escalabilidad en el PC local

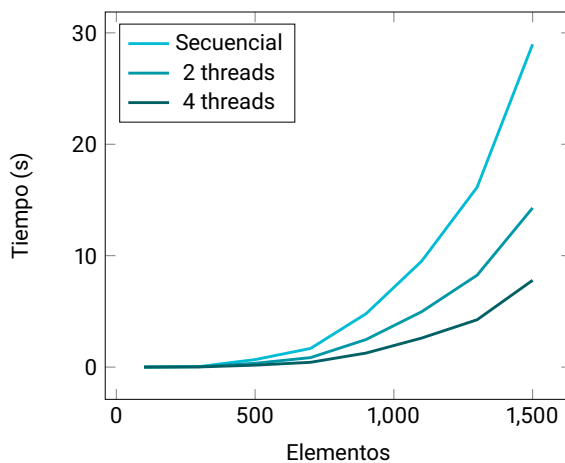
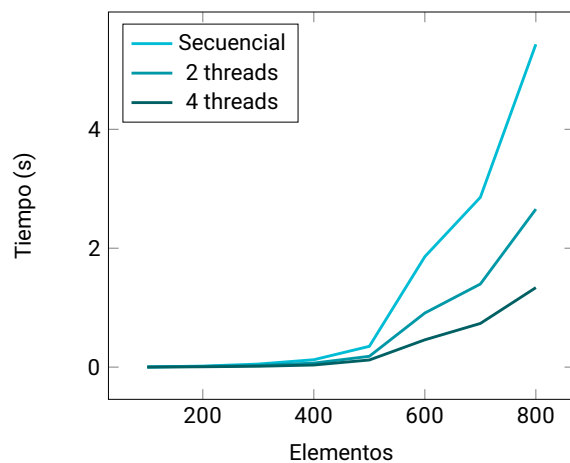


Figura 22: Estudio de escalabilidad en atcgrid



Código fuente 12: pmm-OpenMP-atcgrid.sh

```

1 #!/bin/bash
2
3 #Se asigna al trabajo el nombre
4 #PBS -N pmm-escalabilidad-atcgrid
5 #Se asigna al trabajo la cola ac
6 #PBS -q ac
7
8 #Se imprime información del trabajo usando variables de entorno de PBS
9 echo "Id. usuario del trabajo: $PBS_O_LOGNAME"
10 echo "Id. del trabajo: $PBS_JOBID"
11 echo "Nombre del trabajo especificado por usuario: $PBS_JOBNAME"
12 echo "Nodo que ejecuta qsub: $PBS_O_HOST"
13 echo "Cola: $PBS_QUEUE"
14 echo "Nodos asignados al trabajo:"
15 cat $PBS_NODEFILE

```

```

16 echo "Directorio de trabajo: $PBS_O_WORKDIR"
17
18 echo -e "100\n200\n300\n400\n500\n600\n700\n800" > $PBS_O_WORKDIR/dat/tmp0.dat
19
20 $PBS_O_WORKDIR/bin/pmm-secuencial 100 > $PBS_O_WORKDIR/dat/tmp1.dat
21 $PBS_O_WORKDIR/bin/pmm-secuencial 200 >> $PBS_O_WORKDIR/dat/tmp1.dat
22 $PBS_O_WORKDIR/bin/pmm-secuencial 300 >> $PBS_O_WORKDIR/dat/tmp1.dat
23 $PBS_O_WORKDIR/bin/pmm-secuencial 400 >> $PBS_O_WORKDIR/dat/tmp1.dat
24 $PBS_O_WORKDIR/bin/pmm-secuencial 500 >> $PBS_O_WORKDIR/dat/tmp1.dat
25 $PBS_O_WORKDIR/bin/pmm-secuencial 600 >> $PBS_O_WORKDIR/dat/tmp1.dat
26 $PBS_O_WORKDIR/bin/pmm-secuencial 700 >> $PBS_O_WORKDIR/dat/tmp1.dat
27 $PBS_O_WORKDIR/bin/pmm-secuencial 800 >> $PBS_O_WORKDIR/dat/tmp1.dat
28
29 export OMP_NUM_THREADS="2"
30 echo "num_threads=2"
31 $PBS_O_WORKDIR/bin/pmm-OpenMP 100 > $PBS_O_WORKDIR/dat/tmp2.dat
32 $PBS_O_WORKDIR/bin/pmm-OpenMP 200 >> $PBS_O_WORKDIR/dat/tmp2.dat
33 $PBS_O_WORKDIR/bin/pmm-OpenMP 300 >> $PBS_O_WORKDIR/dat/tmp2.dat
34 $PBS_O_WORKDIR/bin/pmm-OpenMP 400 >> $PBS_O_WORKDIR/dat/tmp2.dat
35 $PBS_O_WORKDIR/bin/pmm-OpenMP 500 >> $PBS_O_WORKDIR/dat/tmp2.dat
36 $PBS_O_WORKDIR/bin/pmm-OpenMP 600 >> $PBS_O_WORKDIR/dat/tmp2.dat
37 $PBS_O_WORKDIR/bin/pmm-OpenMP 700 >> $PBS_O_WORKDIR/dat/tmp2.dat
38 $PBS_O_WORKDIR/bin/pmm-OpenMP 800 >> $PBS_O_WORKDIR/dat/tmp2.dat
39
40 export OMP_NUM_THREADS="4"
41 echo "num_threads=4"
42 $PBS_O_WORKDIR/bin/pmm-OpenMP 100 > $PBS_O_WORKDIR/dat/tmp3.dat
43 $PBS_O_WORKDIR/bin/pmm-OpenMP 200 >> $PBS_O_WORKDIR/dat/tmp3.dat
44 $PBS_O_WORKDIR/bin/pmm-OpenMP 300 >> $PBS_O_WORKDIR/dat/tmp3.dat
45 $PBS_O_WORKDIR/bin/pmm-OpenMP 400 >> $PBS_O_WORKDIR/dat/tmp3.dat
46 $PBS_O_WORKDIR/bin/pmm-OpenMP 500 >> $PBS_O_WORKDIR/dat/tmp3.dat
47 $PBS_O_WORKDIR/bin/pmm-OpenMP 600 >> $PBS_O_WORKDIR/dat/tmp3.dat
48 $PBS_O_WORKDIR/bin/pmm-OpenMP 700 >> $PBS_O_WORKDIR/dat/tmp3.dat
49 $PBS_O_WORKDIR/bin/pmm-OpenMP 800 >> $PBS_O_WORKDIR/dat/tmp3.dat
50
51
52 paste $PBS_O_WORKDIR/dat/tmp0.dat $PBS_O_WORKDIR/dat/tmp1.dat $PBS_O_WORKDIR/dat/tmp2.dat
    $PBS_O_WORKDIR/dat/tmp3.dat > $PBS_O_WORKDIR/dat/pmm-escalabilidad-atcgrid.dat
53
54 echo -e "Elementos\tSecuencial\t2 threads\t4 threads\n$(cat $PBS_O_WORKDIR/dat/pmm-escalabilidad-
    atcgrid.dat)" > $PBS_O_WORKDIR/dat/pmm-escalabilidad-atcgrid.dat
55
56 rm $PBS_O_WORKDIR/dat/tmp0.dat $PBS_O_WORKDIR/dat/tmp1.dat $PBS_O_WORKDIR/dat/tmp2.dat $PBS_O_WORKDIR
    /dat/tmp3.dat

```

Código fuente 13: pmm-OpenMP-pclocal.sh

```

1 #!/bin/bash
2
3 echo -e "100\n300\n500\n700\n900\n1100\n1300\n1500" > ./dat/tmp0.dat
4
5 ./bin/pmm-secuencial 100 > ./dat/tmp1.dat
6 ./bin/pmm-secuencial 300 >> ./dat/tmp1.dat
7 ./bin/pmm-secuencial 500 >> ./dat/tmp1.dat
8 ./bin/pmm-secuencial 700 >> ./dat/tmp1.dat
9 ./bin/pmm-secuencial 900 >> ./dat/tmp1.dat

```

```

10 ./bin/pmm-secuencial 1100 >> ./dat/tmp1.dat
11 ./bin/pmm-secuencial 1300 >> ./dat/tmp1.dat
12 ./bin/pmm-secuencial 1500 >> ./dat/tmp1.dat
13
14 export OMP_NUM_THREADS="2"
15 echo "num_threads=2"
16 ./bin/pmm-OpenMP 100 > ./dat/tmp2.dat
17 ./bin/pmm-OpenMP 300 >> ./dat/tmp2.dat
18 ./bin/pmm-OpenMP 500 >> ./dat/tmp2.dat
19 ./bin/pmm-OpenMP 700 >> ./dat/tmp2.dat
20 ./bin/pmm-OpenMP 900 >> ./dat/tmp2.dat
21 ./bin/pmm-OpenMP 1100 >> ./dat/tmp2.dat
22 ./bin/pmm-OpenMP 1300 >> ./dat/tmp2.dat
23 ./bin/pmm-OpenMP 1500 >> ./dat/tmp2.dat
24
25 export OMP_NUM_THREADS="4"
26 echo "num_threads=4"
27 ./bin/pmm-OpenMP 100 > ./dat/tmp3.dat
28 ./bin/pmm-OpenMP 300 >> ./dat/tmp3.dat
29 ./bin/pmm-OpenMP 500 >> ./dat/tmp3.dat
30 ./bin/pmm-OpenMP 700 >> ./dat/tmp3.dat
31 ./bin/pmm-OpenMP 900 >> ./dat/tmp3.dat
32 ./bin/pmm-OpenMP 1100 >> ./dat/tmp3.dat
33 ./bin/pmm-OpenMP 1300 >> ./dat/tmp3.dat
34 ./bin/pmm-OpenMP 1500 >> ./dat/tmp3.dat
35
36
37 paste ./dat/tmp0.dat ./dat/tmp1.dat ./dat/tmp2.dat ./dat/tmp3.dat > ./dat/pmm-escalabilidad-pclocal.
    dat
38
39 echo -e "Elementos\tSecuencial\t2 threads\t4 threads\n$(cat ./dat/pmm-escalabilidad-pclocal.dat)" >
    ./dat/pmm-escalabilidad-pclocal.dat
40
41 rm ./dat/tmp0.dat ./dat/tmp1.dat ./dat/tmp2.dat ./dat/tmp3.dat

```