

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

Arquitectura de Computadores (AC)

Cuaderno de prácticas.

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

Estudiante (nombre y apellidos):

Grupo de prácticas:

Fecha de entrega:

Fecha evaluación en clase:

Ejercicios basados en los ejemplos del seminario práctico

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

CAPTURA CÓDIGO FUENTE: `if-clauseModificado.c`

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <omp.h>
4
5 int main(int argc, char **argv)
6 {
7     int i, n=20, tid;
8     int a[n], suma=0, sumalocal;
9     int x; //a usar por la cláusula num_threads
10
11     if(argc < 3) {
12         fprintf(stderr, "[ERROR]-Falta iteraciones\n");
13         exit(-1);
14     }
15
16     n = atoi(argv[1]); if (n>20) n=20;
17     x = atoi(argv[2]); if (n>8) n=20; if (n<=0) n=1;
18     for (i=0; i<n; i++) {
19         a[i] = i;
20     }
21
22     #pragma omp parallel num_threads(x) if(n>4) default(none) private(sumalocal,tid) shared
23     (a,suma,n)
24     {
25         sumalocal=0;
26         tid = omp_get_thread_num();
27         #pragma omp for private(i) schedule(static) nowait
28         for(i=0; i<n; i++){
29             sumalocal+= a[i];
30             printf(" thread %d suma de a[%d]=%d sumalocal=%d \n", tid, i, a[i], sumalocal);
31         }
32
33         #pragma omp atomic
34         suma+=sumalocal;
35         #pragma omp barrier
36         #pragma omp master
37         printf("thread master=%d imprime suma=%d \n",tid, suma);
38     }
39 }
```

CAPTURAS DE PANTALLA:

```

guillesiesta@guillesiesta:~$ gcc -O2 -fopenmp if-clauseModificado.c -o icM
guillesiesta@guillesiesta:~$ ./icM 5 4
thread 0 suma de a[0]=0 sumalocal=0
thread 0 suma de a[1]=1 sumalocal=1
thread 2 suma de a[3]=3 sumalocal=3
thread 3 suma de a[4]=4 sumalocal=4
thread 1 suma de a[2]=2 sumalocal=2
thread master=0 imprime suma=10
guillesiesta@guillesiesta:~$ ./icM 5 3
thread 0 suma de a[0]=0 sumalocal=0
thread 0 suma de a[1]=1 sumalocal=1
thread 2 suma de a[4]=4 sumalocal=4
thread 1 suma de a[2]=2 sumalocal=2
thread 1 suma de a[3]=3 sumalocal=5
thread master=0 imprime suma=10
guillesiesta@guillesiesta:~$ ./icM 5 5
thread 0 suma de a[0]=0 sumalocal=0
thread 1 suma de a[1]=1 sumalocal=1
thread 2 suma de a[2]=2 sumalocal=2
thread 3 suma de a[3]=3 sumalocal=3
thread 4 suma de a[4]=4 sumalocal=4
thread master=0 imprime suma=10
guillesiesta@guillesiesta:~$

```

RESPUESTA:

Como se puede observar en el código, solo se paraleliza cuando el número de iteraciones es mayor que 4, es entonces, cuando actúa la cláusula `num_threads`, creando el número de hebras que le pasamos por parámetro.

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

- iteraciones: 16 (0,...15)
- chunk= 1, 2 y 4

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

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

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

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

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

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

RESPUESTA:

Usando `static`, todas las tareas se reparte equitativamente usando round-robin. Con `dynamic`, no controlamos la asignación de estas tareas, es el sistema operativo el que se encarga de repartirlas, nosotros solamente podemos especificar el tamaño de éstas con el valor definido en `chunk`. Con `guided`, el reparto de las hebras sigue siendo dinámico, pero la principal diferencia con `Dynamic` es que el `chunk` indica el valor mínimo del reparto. Se puede observar en la tabla 1, cuando el `chunk` es 4, vemos que la hebra 0 ejecuta un bloque de 7, y mientras se avanzan en las iteraciones, los bloques tienden a tener el tamaño especificado por `chunk`.

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

CAPTURA CÓDIGO FUENTE: `scheduled-clauseModificado.c`

```

2
3 #include <stdio.h>
4 #include <stdlib.h>
5 #ifdef _OPENMP
6     #include <omp.h>
7 #else
8     #define omp_get_thread_num() 0
9     #define omp_get_thread_num() 1
10    #define omp_get_thread_num(int)
11 #endif
12
13 void main(int argc, char **argv) {
14
15     int i, n=200, chunk, a[n], suma=0;
16     omp_sched_t schedule_type;
17     /*
18      omp_sched_t {
19          omp_sched_static =1,
20          omp_sched_dynamic=2,
21          omp_sched_guided=3,
22          omp_sched_auto=4
23      }
24     */
25     int chunk_value;
26
27     if(argc<3){
28         fprintf(stderr, "\nFalta chunk o iteraciones\n");
29         exit(-1);
30     }
31
32     n = atoi(argv[1]); if (n>200) n=200;
33     chunk = atoi(argv[2]);
34
35     for(i=0; i<n; i++) a[i]=i;
36
37     #pragma omp parallel for firstprivate(suma) lastprivate(suma) schedule(dynamic,chunk)
38     for(i=0; i<n; i++){
39         suma=suma + a[i];
40         printf("thread %d suma a[%d]=%d suma=%d\n", omp_get_thread_num(), i, a[i], suma);
41
42         if(omp_get_thread_num()==0){
43             printf("Dentro del parallel for:\n");
44             printf("static=1, dynamic=2, guided=3, auto=4\n");
45             omp_get_schedule(&schedule_type, &chunk_value);
46             printf("dyn-var: %d, nthreads-var: %d, thread-limit: %d, run-sched-var: %d, chunk-
47 value: %d\n", omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit(), schedule_type,
48 chunk_value);
49         }
50         printf("Fuera de 'parallel for' suma=%d \n", suma);
51         printf("static=1, dynamic=2, guided=3, auto=4\n");
52         omp_get_schedule(&schedule_type, &chunk_value);
53         printf("dyn-var: %d, nthreads-var: %d, thread-limit: %d, run-sched-var: %d, chunk-value: %d\n",
54 omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit(), schedule_type, chunk_value);
55     }

```

CAPTURAS DE PANTALLA:

```

guillesiest@guillesiest:~$ ./scdm 5 1
thread 1 suma a[1]=1 suma=1
thread 7 suma a[4]=4 suma=4
thread 2 suma a[3]=3 suma=3
thread 6 suma a[2]=2 suma=2
thread 0 suma a[0]=0 suma=0
Dentro del parallel for:
static=1, dynamic=2, guided=3, auto=4
dyn-var: 0, nthreads-var: 8, thread-limit: 2147483647, run-sched-var: 2, chunk-value: 1
Fuera de 'parallel for' suma=4
static=1, dynamic=2, guided=3, auto=4
dyn-var: 0, nthreads-var: 8, thread-limit: 2147483647, run-sched-var: 2, chunk-value: 1

```

```

guillesiesta@guillesiesta:~$ ./scdM 5 1
thread 0 suma a[0]=0 suma=0
Dentro del parallel for:
static=1, dynamic=2, guided=3, auto=4
dyn-var: 0, nthreads-var: 2, thread-limit: 2147483647, run-sched-var: 2, chunk-value: 1
thread 0 suma a[2]=2 suma=2
Dentro del parallel for:
static=1, dynamic=2, guided=3, auto=4
dyn-var: 0, nthreads-var: 2, thread-limit: 2147483647, run-sched-var: 2, chunk-value: 1
thread 0 suma a[3]=3 suma=5
Dentro del parallel for:
static=1, dynamic=2, guided=3, auto=4
dyn-var: 0, nthreads-var: 2, thread-limit: 2147483647, run-sched-var: 2, chunk-value: 1
thread 1 suma a[1]=1 suma=1
thread 0 suma a[4]=4 suma=9
Dentro del parallel for:
static=1, dynamic=2, guided=3, auto=4
dyn-var: 0, nthreads-var: 2, thread-limit: 2147483647, run-sched-var: 2, chunk-value: 1
Fuera de 'parallel for' suma=9
static=1, dynamic=2, guided=3, auto=4
dyn-var: 0, nthreads-var: 2, thread-limit: 2147483647, run-sched-var: 2, chunk-value: 1

```

```

guillesiesta@guillesiesta:~$ export OMP_NUM_THREADS=10
guillesiesta@guillesiesta:~$ ./scdM 5 1
thread 4 suma a[0]=0 suma=0
thread 6 suma a[1]=1 suma=1
thread 7 suma a[2]=2 suma=2
thread 5 suma a[3]=3 suma=3
thread 3 suma a[4]=4 suma=4
Fuera de 'parallel for' suma=4
static=1, dynamic=2, guided=3, auto=4
dyn-var: 0, nthreads-var: 10, thread-limit: 2147483647, run-sched-var: 1, chunk-value: 2
guillesiesta@guillesiesta:~$

```

```

guillesiesta@guillesiesta:~$ export OMP_THREAD_LIMIT=4
guillesiesta@guillesiesta:~$ ./scdM 5 1
thread 0 suma a[0]=0 suma=0
Dentro del parallel for:
static=1, dynamic=2, guided=3, auto=4
dyn-var: 0, nthreads-var: 10, thread-limit: 4, run-sched-var: 1, chunk-value: 2
thread 0 suma a[4]=4 suma=4
Dentro del parallel for:
static=1, dynamic=2, guided=3, auto=4
dyn-var: 0, nthreads-var: 10, thread-limit: 4, run-sched-var: 1, chunk-value: 2
thread 1 suma a[2]=2 suma=2
thread 3 suma a[3]=3 suma=3
thread 2 suma a[1]=1 suma=1
Fuera de 'parallel for' suma=4
static=1, dynamic=2, guided=3, auto=4
dyn-var: 0, nthreads-var: 10, thread-limit: 4, run-sched-var: 1, chunk-value: 2
guillesiesta@guillesiesta:~$

```

```

guillesiesta@guillesiesta:~$ export OMP_SCHEDULE="static,2"
guillesiesta@guillesiesta:~$ ./scdM 5 1
thread 0 suma a[0]=0 suma=0
Dentro del parallel for:
static=1, dynamic=2, guided=3, auto=4
thread 1 suma a[1]=1 suma=1
thread 1 suma a[2]=2 suma=3
thread 1 suma a[3]=3 suma=6
thread 1 suma a[4]=4 suma=10
dyn-var: 0, nthreads-var: 2, thread-limit: 2147483647, run-sched-var: 1, chunk-value: 2
Fuera de 'parallel for' suma=10
static=1, dynamic=2, guided=3, auto=4
dyn-var: 0, nthreads-var: 2, thread-limit: 2147483647, run-sched-var: 1, chunk-value: 2
guillesiesta@guillesiesta:~$

```

```
guillesiesta@guillesiesta:~$ export OMP_DYNAMIC=TRUE
guillesiesta@guillesiesta:~$ ./scdm 5 1
thread 0 suma a[1]=1 suma=1
Dentro del parallel for:
static=1, dynamic=2, guided=3, auto=4
thread 2 suma a[2]=2 suma=2
thread 2 suma a[4]=4 suma=6
thread 3 suma a[3]=3 suma=3
thread 1 suma a[0]=0 suma=0
dyn-var: 1, nthreads-var: 10, thread-limit: 4, run-sched-var: 1, chunk-value: 2
Fuera de 'parallel for' suma=6
static=1, dynamic=2, guided=3, auto=4
dyn-var: 1, nthreads-var: 10, thread-limit: 4, run-sched-var: 1, chunk-value: 2
guillesiesta@guillesiesta:~$
```

RESPUESTA:

Se puede observar que los valores dentro del parallel y fuera del parallel son los mismos.

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

CAPTURA CÓDIGO FUENTE: `scheduled-clauseModificado4.c`

```

12
13 void main(int argc, char **argv) {
14
15     int i, n=200, chunk, a[n], suma=0;
16     omp_sched_t schedule_type;
17     /*
18         omp_sched_t {
19             omp_sched_static =1,
20             omp_sched_dynamic=2,
21             omp_sched_guided=3,
22             omp_sched_auto=4
23         }
24     */
25     int chunk_value;
26
27     if(argc<3){
28         fprintf(stderr, "\nFalta chunk o iteraciones\n");
29         exit(-1);
30     }
31
32     n = atoi(argv[1]); if (n>200) n=200;
33     chunk = atoi(argv[2]);
34
35     for(i=0; i<n; i++) a[i]=i;
36
37     #pragma omp parallel for firstprivate(suma) lastprivate(suma) schedule(dynamic,chunk)
38     for(i=0;i<n;i++){
39         suma=suma + a[i];
40         printf("thread %d suma a[%d]=%d suma=%d\n", omp_get_thread_num(), i, a[i], suma);
41
42         if(omp_get_thread_num()==0){
43             printf("Dentro del parallel for:\n");
44             printf("static=1, dynamic=2, guided=3, auto=4\n");
45             omp_get_schedule(&schedule_type, &chunk_value);
46             printf("dyn-var: %d, nthreads-var: %d, thread-limit: %d, run-sched-var: %d, chunk-
value: %d\n", omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit(), schedule_type,
chunk_value);
47
48             //añadido nuevo
49             printf("get_num_threads: %d, get_num_procs: %d, in_parallel: %d\n",omp_get_num_threads
(), omp_get_num_procs(),omp_in_parallel());
50         }
51     }//fin del parallel
52
53     printf("Fuera de 'parallel for' suma=%d \n",suma);
54     printf("static=1, dynamic=2, guided=3, auto=4\n");
55     omp_get_schedule(&schedule_type, &chunk_value);
56     printf("dyn-var: %d, nthreads-var: %d, thread-limit: %d, run-sched-var: %d, chunk-value: %d\n",
omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit(), schedule_type, chunk_value);
57
58     //añadido nuevo
59     printf("get_num_threads: %d, get_num_procs: %d, in_parallel: %d\n",omp_get_num_threads
(), omp_get_num_procs(),omp_in_parallel());
60 }

```

CAPTURAS DE PANTALLA:

```

guillesiesta@guillesiesta:~$ ./scdM4 5 1
thread 0 suma a[4]=4 suma=4
Dentro del parallel for:
static=1, dynamic=2, guided=3, auto=4
dyn-var: 1, nthreads-var: 8, thread-limit: 50000, run-sched-var: 2, chunk-value: 1
get_num_threads: 8, get_num_procs: 8, in_parallel: 1
thread 6 suma a[3]=3 suma=3
thread 1 suma a[0]=0 suma=0
thread 7 suma a[2]=2 suma=2
thread 2 suma a[1]=1 suma=1
Fuera de 'parallel for' suma=4
static=1, dynamic=2, guided=3, auto=4
dyn-var: 1, nthreads-var: 8, thread-limit: 50000, run-sched-var: 2, chunk-value: 1
get_num_threads: 1, get_num_procs: 8, in_parallel: 0
guillesiesta@guillesiesta:~$

```

RESPUESTA:

La única función que no varía es `omp_get_num_procs()`, las demás si varían dependiendo de la región en la que se encuentren.

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

CAPTURA CÓDIGO FUENTE: `scheduled-clauseModificado5.c`

```

3 #include <stdio.h>
4 #include <stdlib.h>
5 #ifdef _OPENMP
6 #include <omp.h>
7 #else
8 #define omp_get_thread_num() 0
9 #define omp_get_thread_num() 1
10 #define omp_get_thread_num(int)
11 #define omp_in_parallel() 0
12 #define omp_set_dynamic(int)
13 #endif
14
15 void main(int argc, char **argv) {
16
17     int i, n=200, chunk, a[n], suma=0;
18     omp_sched_t schedule_type;
19     /*
20      omp_sched_t {
21          omp_sched_static=1,
22          omp_sched_dynamic=2,
23          omp_sched_guided=3,
24          omp_sched_auto=4
25      }
26     */
27     int chunk_value;
28
29     if(argc<3){
30         fprintf(stderr, "\nFalta chunk o iteraciones\n");
31         exit(-1);
32     }
33
34     n = atoi(argv[1]); if (n>200) n=200;
35     chunk = atoi(argv[2]);
36
37     for(i=0; i<n; i++) a[i]=i;
38
39     //IMPRESIÓN ANTES DEL CAMBIO
40
41     printf("-----ANTES DEL CAMBIO-----\n");
42     printf("INFO--->static=1, dynamic=2, guided=3, auto=4\n");
43     omp_get_schedule(&schedule_type, &chunk_value);
44     printf("dyn-var: %d, nthreads-var: %d, thread-limit: %d, run-sched-var: %d, chunk-
value: %d\n", omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit(), schedule_type,
chunk_value);
45
46     //añadido nuevo
47     printf("get_num_threads: %d, get_num_procs: %d, in_parallel: %d\n", omp_get_num_threads
(), omp_get_num_procs(), omp_in_parallel());
48
49     omp_set_dynamic(2);
50     omp_set_num_threads(2);
51     omp_set_schedule(1,1);
52     printf("-----FIN DEL CAMBIO-----\n");
53
54     #pragma omp parallel for firstprivate(suma) lastprivate(suma) schedule(dynamic,chunk)
55     for(i=0; i<n; i++){
56         suma=suma + a[i];
57         printf("thread %d suma a[%d]=%d suma=%d\n", omp_get_thread_num(), i, a[i], suma);
58
59         if(omp_get_thread_num()==0){
60             printf("Dentro del parallel for:\n");
61         }
62     } //fin del parallel
63
64     printf("Fuera de 'parallel for' suma=%d \n", suma);
65     printf("-----DESPUES DEL CAMBIO-----\n");
66     printf("INFO--->static=1, dynamic=2, guided=3, auto=4\n");
67     omp_get_schedule(&schedule_type, &chunk_value);
68     printf("dyn-var: %d, nthreads-var: %d, thread-limit: %d, run-sched-var: %d, chunk-
value: %d\n", omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit(), schedule_type,
chunk_value);
69
70     //añadido nuevo
71     printf("get_num_threads: %d, get_num_procs: %d, in_parallel: %d\n", omp_get_num_threads
(), omp_get_num_procs(), omp_in_parallel());
72
73     printf("-----FIN DESPUES DEL CAMBIO-----\n");
74 }

```


CAPTURAS DE PANTALLA:

```
guillesiesta@guillesiesta:~$ gcc -O2 scheduled-clauseModificado5.c -o scdM5 -fopenmp
guillesiesta@guillesiesta:~$ ./scdM5 5 1
-----ANTES DEL CAMBIO-----
INFO--->static=1, dynamic=2, guided=3, auto=4
dyn-var: 1, nthreads-var: 8, thread-limit: 50000, run-sched-var: 2, chunk-value: 1
get_num_threads: 1, get_num_procs: 8, in_parallel: 0
-----FIN DEL CAMBIO-----
thread 0 suma a[0]=0 suma=0
Dentro del parallel for:
thread 1 suma a[1]=1 suma=1
thread 1 suma a[3]=3 suma=4
thread 1 suma a[4]=4 suma=8
thread 0 suma a[2]=2 suma=2
Dentro del parallel for:
Fuera de 'parallel for' suma=8
-----DESPUES DEL CAMBIO-----
INFO--->static=1, dynamic=2, guided=3, auto=4
dyn-var: 1, nthreads-var: 2, thread-limit: 50000, run-sched-var: 1, chunk-value: 1
get_num_threads: 1, get_num_procs: 8, in_parallel: 0
-----FIN DESPUES DEL CAMBIO-----
```

RESPUESTA:

Se puede observar que al cambiar los valores, éstos son devueltos en las funciones finales.

Resto de ejercicios

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

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

CAPTURA CÓDIGO FUENTE: pmtv-secuencial.c

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4
5 int main(int argc, char **argv)
6 {
7     int i, j;
8
9     //Argumento de entrada
10    if(argc < 2){
11        fprintf(stderr, "Falta tamaño de filas/columnas\n");
12        exit(-1);
13    }
14
15    unsigned int N = atoi(argv[1]); // Máximo N =2^32-1=4294967295 (sizeof(unsigned int) = 4 B)
16
17    // Inicializamos la matriz triangular (superior)
18    int *vector, *result, **matriz;
19    vector = (int *) malloc(N*sizeof(int)); // malloc necesita el tamaño en bytes
20    result = (int *) malloc(N*sizeof(int)); //si no hay espacio suficiente malloc devuelve NULL
21    matriz = (int **) malloc(N*sizeof(int*));
22
23    for (i=0; i<N; i++)
24        matriz[i] = (int*) malloc(N*sizeof(int));
25
26    for (i=0; i<N; i++){
27        for (j=i; j<N; j++){
28            matriz[i][j] = 0;
29            vector[i] = 4;
30            result[i]=0;
31        }
32    }
33
34    // Pintamos la matriz
35    printf("Matriz:\n");
36    for (i=0; i<N; i++){
37        for (j=0; j<N; j++){
38            if (j >= i){
39                printf("%d ", matriz[i][j]);
40            }else{
41                printf("0 ");
42            }
43        }
44        printf("\n");
45    }
46
47    // Pintamos el vector
48    printf("Vector:\n");
49    for (i=0; i<N; i++){
50        printf("%d ", vector[i]);
51    }
52    printf("\n");
53

```

```

53
54    //MULTIPLICACION
55    for (i=0; i<N; i++){
56        for (j=i; j<N; j++){
57            result[i] += matriz[i][j] * vector[j];
58        }
59    }
60
61    // Pintamos los resultados
62    printf("Resultado:\n");
63    for (i=0; i<N; i++){
64        printf("%d ", result[i]);
65    }
66    printf("\n");
67
68    // Liberamos la memoria
69    for (i=0; i<N; i++)
70        free(matriz[i]);
71    free(matriz);
72    free(vector);
73    free(result);
74    return 0;
75 }

```

CAPTURAS DE PANTALLA:

```

guillesiesta@guillesiesta:~$ gcc -O2 -fopenmp pmtv-secuencial.c -o pmtv
guillesiesta@guillesiesta:~$ ./pmtv 4
Matriz:
6 6 6 6
0 6 6 6
0 0 6 6
0 0 0 6
Vector:
4 4 4 4
Resultado:
96 72 48 24
guillesiesta@guillesiesta:~$ ./pmtv 10
Matriz:
6 6 6 6 6 6 6 6 6 6
0 6 6 6 6 6 6 6 6 6
0 0 6 6 6 6 6 6 6 6
0 0 0 6 6 6 6 6 6 6
0 0 0 0 6 6 6 6 6 6
0 0 0 0 0 6 6 6 6 6
0 0 0 0 0 0 6 6 6 6
0 0 0 0 0 0 0 6 6 6
0 0 0 0 0 0 0 0 6 6
0 0 0 0 0 0 0 0 0 6
Vector:
4 4 4 4 4 4 4 4 4 4
Resultado:
240 216 192 168 144 120 96 72 48 24
guillesiesta@guillesiesta:~$

```

7. Implementar en paralelo la multiplicación de una matriz triangular por un vector a partir del código secuencial realizado para el ejercicio anterior utilizando la directiva `for` de OpenMP. El código debe repartir entre los threads las iteraciones del bucle que recorre las filas. Dibujar en el cuaderno de prácticas la descomposición de dominio utilizada (Lección 4/Tema 2) en el código paralelo implementado para asignar tareas a los threads (Lección 5/Tema 2). Añadir lo necesario para que el usuario pueda fijar la planificación de tareas usando la variable de entorno `OMP_SCHEDULE`. Obtener en `atcgrid` los tiempos de ejecución del código paralelo (usando, como siempre, `-O2` al compilar) que multiplica una matriz triangular por un vector con las alternativas de planificación `static`, `dynamic` y `guided` para `chunk` de 1, 64 y el `chunk` por defecto para la alternativa. Use un tamaño de vector `N` múltiplo del número de cores y de 64 que no sea inferior a 15360. El número de threads en las ejecuciones debe coincidir con el número de cores. Rellenar la Tabla 3 dos veces con los tiempos obtenidos. Representar el tiempo para `static`, `dynamic` y `guided` en función del tamaño del `chunk` en una gráfica. ¿Qué alternativa ofrece mejores prestaciones? Razone por qué. Incluya los scripts utilizado en el cuaderno de prácticas. NOTA: Nunca ejecute en `atcgrid` código que imprima todos los componentes del resultado.

Conteste a las siguientes preguntas: (a) ¿Qué valor por defecto usa OpenMP para `chunk` con `static`, `dynamic` y `guided`? Indique qué ha hecho para obtener este valor por defecto para cada alternativa. (b) ¿Qué número de operaciones de multiplicación y suma realizan cada uno de los threads en la asignación `static` para cada uno de los `chunks`? (c) Con la asignación `dynamic` y `guided`, ¿qué cree que debe ocurrir con el número de operaciones de multiplicación y suma que realizan cada uno de los threads?

RESPUESTA:

Con respecto al tiempo de ejecución, el tiempo es más o menos igual en todas las `SCHEDULE`. `Static` puede funcionar un poco mejor, al poder ser menos costoso el reparto `round-robin`.

a) Static no tiene chunk. Dynamic y guided tienen por defecto chunk 1. Visto aquí:

<https://computing.llnl.gov/tutorials/openMP/#RunTimeLibrary>

b) Se hará el número de chunk * número de la fila

c) En guided el chunk tiende a ser muy alto al principio y después a llegar al tamaño chunk especificado como mínimo. En Dynamic, pues el chunk seria 1 operación o 64 operaciones por hebra.

CAPTURA CÓDIGO FUENTE: pmtv-OpenMP.c

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4
5 #ifdef _OPENMP
6     #include <omp.h>
7 #else
8     #define omp_get_thread_num() 0
9     #define omp_get_num_threads() 1
10    #define omp_set_num_threads(int)
11    #define omp_in_parallel() 0
12    #define omp_set_dynamic(int)
13 #endif
14
15
16 int main(int argc, char **argv)
17 {
18     int i, j, debug=0; //si ponemos debug a 1 es para ver la matriz y el vector pintados
19
20     //Argumento de entrada
21     if(argc < 2){
22         fprintf(stderr, "Falta tamaño de filas/columnas [optional debug]\n");
23         exit(-1);
24     }
25
26     unsigned int N = atoi(argv[1]); // Máximo N =2^32-1=4294967295 (sizeof(unsigned int) = 4 B)
27
28     if(argc == 3){
29         debug = atoi(argv[2]);
30     }
31
32     // Inicializamos la matriz triangular (superior)
33     int *vector, *result, **matriz;
34     vector = (int *) malloc(N*sizeof(int)); // malloc necesita el tamaño en bytes
35     result = (int *) malloc(N*sizeof(int)); //si no hay espacio suficiente malloc devuelve NULL
36     matriz = (int **) malloc(N*sizeof(int*));
37
38     for (i=0; i<N; i++){
39         matriz[i] = (int*) malloc(N*sizeof(int));
40     }
41
42     for (i=0; i<N; i++){
43         for (j=i; j<N; j++){
44             matriz[i][j] = 6;
45             vector[i] = 4;
46             result[i]=0;
47         }
48     }

```

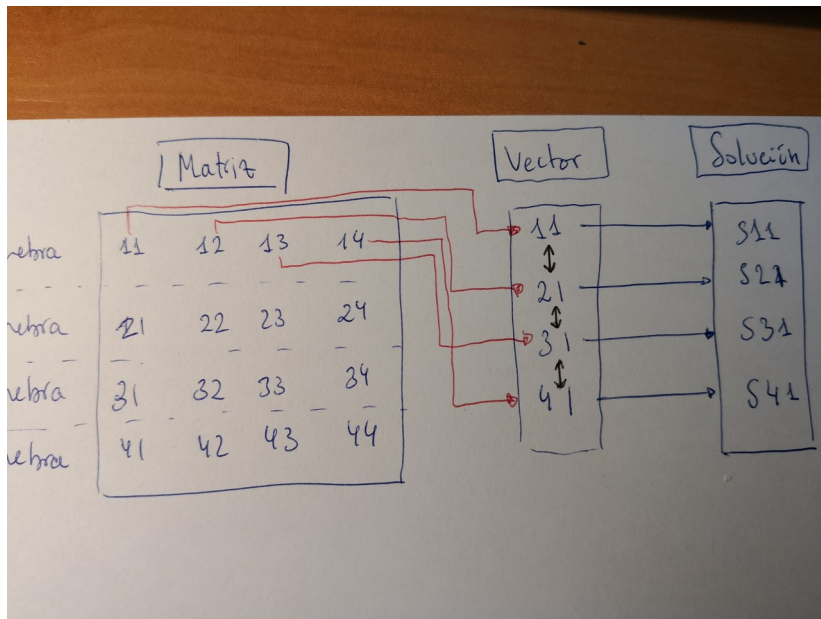
```

48
49     if(debug==1){
50         // Pintamos la matriz
51         printf("Matriz:\n");
52         for (i=0; i<N; i++){
53             for (j=0; j<N; j++){
54                 if (j >= i){
55                     printf("%d ", matriz[i][j]);
56                 }else{
57                     printf("0 ");
58                 }
59             }
60             printf("\n");
61         }
62
63         // Pintamos el vector
64         printf("Vector:\n");
65         for (i=0; i<N; i++){
66             printf("%d ", vector[i]);
67         }
68         printf("\n");
69     }
70
71     double t1, t2, t_total;
72     t1 = omp_get_wtime();
73
74     //A por los resultados!!
75     //uso runtime para poder variarlo luego con la variable OMP_SCHEDULE
76
77
78     #pragma omp parallel for private(j) schedule(runtime)
79     for (i=0; i<N; i++){
80         for (j=i; j<N; j++){
81             result[i] += matriz[i][j] * vector[j];
82         }
83     }
84
85     t2 = omp_get_wtime();
86     t_total = t2 - t1;
87
88     if(debug==1){
89         // Pintamos los resultados
90         printf("Resultado:\n");
91         for (i=0; i<N; i++){
92             printf("%d ", result[i]);
93         }
94         printf("\n");
95     }
96
97     //Se imprime el primer y el último valor del vector resultado :- )
98     printf("Tiempo = %11.9f\t Primera = %d\t Ultima=%d\n",t_total,result[0],result[N-1]);
99
100
101     // Liberamos la memoria
102     for (i=0; i<N; i++)
103         free(matriz[i]);
104     free(matriz);
105     free(vector);
106     free(result);
107     return 0;

```

DESCOMPOSICIÓN DE DOMINIO:

Se busca que haya un reparto en el que las filas de la matriz en cada thread. Después cada thread va calculando un valor del vector final, recorriendo cada una una fila distinta.



CAPTURAS DE PANTALLA:

```
guillesiestra@guillesiestra:~$ gcc -O2 -fopenmp pmtv-OpenMP.c -o pmtvO
guillesiestra@guillesiestra:~$ ./pmtvO 10 1
Matriz:
6 6 6 6 6 6 6 6 6 6
0 6 6 6 6 6 6 6 6 6
0 0 6 6 6 6 6 6 6 6
0 0 0 6 6 6 6 6 6 6
0 0 0 0 6 6 6 6 6 6
0 0 0 0 0 6 6 6 6 6
0 0 0 0 0 0 6 6 6 6
0 0 0 0 0 0 0 6 6 6
0 0 0 0 0 0 0 0 6 6
0 0 0 0 0 0 0 0 0 6
0 0 0 0 0 0 0 0 0 6
Vector:
4 4 4 4 4 4 4 4 4 4
Resultado:
240 216 192 168 144 120 96 72 48 24
Tiempo = 0.007722216    Primera = 240    Ultima=24
guillesiestra@guillesiestra:~$
```

TABLA RESULTADOS, SCRIPT Y GRÁFICA atcgrid

SCRIPT: pmtv-OpenMP_PCaula.sh

```
1 #!/bin/bash
2
3 #PBS -N ej7_atcgrid
4 #PBS -q ac
5 echo "Id$PBS_O_WORKDIR usuario del trabajo: $PBS_O_LOGNAME"
6 echo "Id$PBS_O_WORKDIR del trabajo: $PBS_JOBID"
7 echo "Nombre del trabajo especificado por usuario: $PBS_JOBNAME"
8 echo "Nodo que ejecuta qsub: $PBS_O_HOST"
9 echo "Directorio en el que se ha ejecutado qsub: $PBS_O_WORKDIR"
10 echo "Cola: $PBS_QUEUE"
11 echo "Nodos asignados al trabajo:"
12 cat $PBS_NODEFILE
13
14 export OMP_SCHEDULE="static"
15 echo "static y chunk por defecto"
16 $PBS_O_WORKDIR/pmtv-OpenMP 15360
17
18 export OMP_SCHEDULE="static,1"
19 echo "static y chunk 1"
20 $PBS_O_WORKDIR/pmtv-OpenMP 15360
21
22 export OMP_SCHEDULE="static,64"
23 echo "static y chunk 64"
24 $PBS_O_WORKDIR/pmtv-OpenMP 15360
25
26 export OMP_SCHEDULE="dynamic"
27 echo "dynamic y chunk por defecto"
28 $PBS_O_WORKDIR/pmtv-OpenMP 15360
29
30 export OMP_SCHEDULE="dynamic,1"
31 echo "dynamic y chunk 1"
32 $PBS_O_WORKDIR/pmtv-OpenMP 15360
33
34 export OMP_SCHEDULE="dynamic,64"
35 echo "dynamic y chunk 64"
36 $PBS_O_WORKDIR/pmtv-OpenMP 15360
37
38 export OMP_SCHEDULE="guided"
39 echo "guided y chunk por defecto"
40 $PBS_O_WORKDIR/pmtv-OpenMP 15360
41
42 export OMP_SCHEDULE="guided,1"
43 echo "guided y chunk 1"
44 $PBS_O_WORKDIR/pmtv-OpenMP 15360
45
46 export OMP_SCHEDULE="guided,64"
47 echo "guided y chunk 64"
48 $PBS_O_WORKDIR/pmtv-OpenMP 15360
```

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

Chunk	Static	Dynamic	Guided
por defecto	0,046711916	1,046711916	2,046711916
1	0,035047011	0,037187167	0,034859763
64	0,033244486	0,034451319	0,034451054
Chunk	Static	Dynamic	Guided
por defecto	0,05897941	0,036976165	0,041331375
1	0,034590594	0,037256078	0,037069255
64	0,033459864	0,033246293	0,040510096

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.

CAPTURA CÓDIGO FUENTE: pmm-secuencial.c


```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4
5 int main(int argc, char **argv)
6 {
7     unsigned i, j, k;
8
9     if(argc < 2)
10     {
11         fprintf(stderr, "falta size\n");
12         exit(-1);
13     }
14
15     unsigned int N = atoi(argv[1]); // Máximo N = 2^32-1=4294967295 (sizeof(unsigned int) = 4 B)
16
17     int **ma, **mb, **mc;
18     ma = (int **) malloc(N*sizeof(int*));
19     mb = (int **) malloc(N*sizeof(int*));
20     mc = (int **) malloc(N*sizeof(int*));
21
22     //reserva de memoria
23     for (i=0; i<N; i++){
24         ma[i] = (int *) malloc(N*sizeof(int));
25         mb[i] = (int *) malloc(N*sizeof(int));
26         mc[i] = (int *) malloc(N*sizeof(int));
27     }
28
29     // Inicialización
30     for (i=0; i<N; i++){
31         for (j=0; j<N; j++){
32             ma[i][j] = 0; //aquí es donde almacenaremos el resultado
33             mb[i][j] = 6;
34             mc[i][j] = 4;
35         }
36     }
37
38     struct timespec cgt1,cgt2; double ncgt;
39
40     clock_gettime(CLOCK_REALTIME,&cgt1);
41     // Multiplicación
42     for (i=0; i<N; i++){
43         for (j=0; j<N; j++){
44             for (k=0; k<N; k++){
45                 ma[i][j] += mb[i][k] * mc[k][j];
46             }
47         }
48     }
49     clock_gettime(CLOCK_REALTIME,&cgt2);
50
51     ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+( double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));
52
53     // Pitamos la primera y la ultima linea de la matriz resultante
54     printf("Tiempo = %11.9f\t Primera = %d\t Ultima=%d\n",ncgt,ma[0][0],ma[N-1][N-1]);
55
56     // Liberamos la memoria
57     for (i=0; i<N; i++)
58     {
59         free(ma[i]);
60         free(mb[i]);
61         free(mc[i]);
62     }
63     free(ma);
64     free(mb);
65     free(mc);
66
67     return 0;
68 }
69

```

CAPTURAS DE PANTALLA:

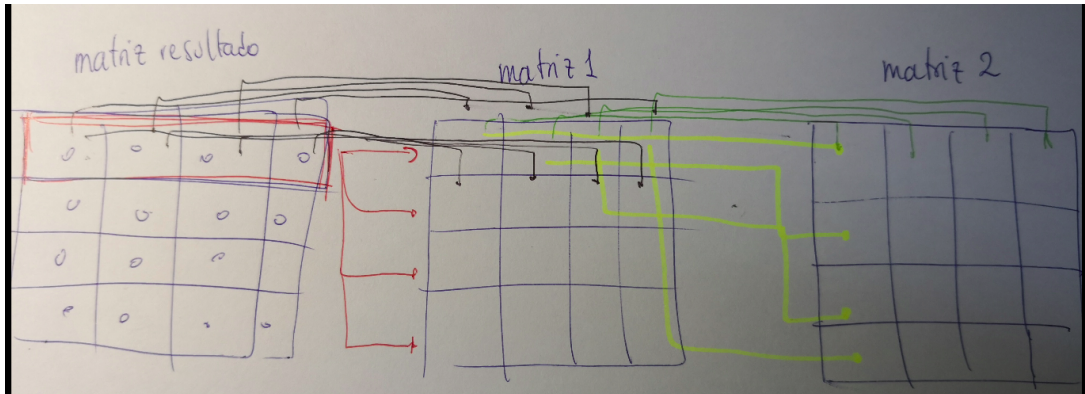
```

guillesiesta@guillesiesta:~$ gcc -O2 -fopenmp pmm-secuencial.c -o pmm
guillesiesta@guillesiesta:~$ ./pmm
falta size
guillesiesta@guillesiesta:~$ ./pmm 5
Tiempo = 0.000000980    Primera = 120    Ultima=120
guillesiesta@guillesiesta:~$ ./pmm 10
Tiempo = 0.000003556    Primera = 240    Ultima=240
guillesiesta@guillesiesta:~$ ./pmm 20
Tiempo = 0.000031197    Primera = 480    Ultima=480
guillesiesta@guillesiesta:~$ ./pmm 2000
Tiempo = 47.871238197    Primera = 48000    Ultima=48000
guillesiesta@guillesiesta:~$

```

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

DESCOMPOSICIÓN DE DOMINIO: Repartimos las filas de la matriz resultado, cada thread va recorriendo diferentes filas de la primera matriz, pero todas recorren todas sus columnas y todas las filas de la segunda matriz.

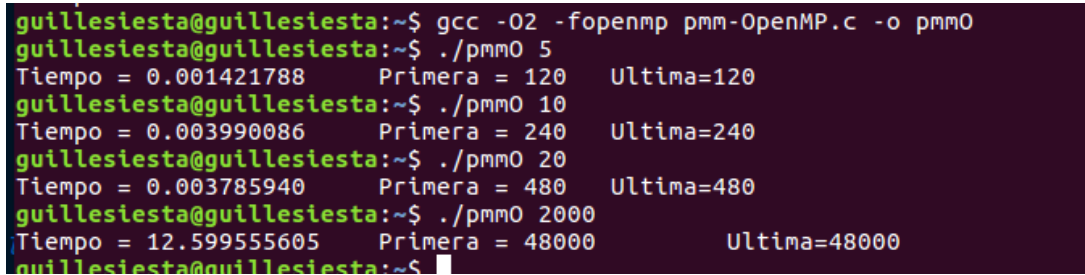


CAPTURA CÓDIGO FUENTE: pmm-OpenMP.c

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4
5 #ifdef _OPENMP
6     #include <omp.h>
7 #else
8     #define omp_get_thread_num() 0
9     #define omp_get_num_threads() 1
10    #define omp_set_num_threads(int)
11    #define omp_in_parallel() 0
12    #define omp_set_dynamic(int)
13 #endif
14
15
16 int main(int argc, char **argv)
17 {
18     int i, j, k;
19
20     //Argumento de entrada
21     if(argc < 2){
22         fprintf(stderr, "Falta tamaño de filas/columnas\n");
23         exit(-1);
24     }
25
26     unsigned int N = atoi(argv[1]); // Máximo N = 2^32-1=4294967295 (sizeof(unsigned int) = 4 B)
27
28     // Reserva de espacio de las matrices
29     int **ma, **mb, **mc;
30     ma = (int **) malloc(N*sizeof(int*)); // malloc necesita el tamaño en bytes
31     mb = (int **) malloc(N*sizeof(int*)); //si no hay espacio suficiente malloc devuelve NULL
32     mc = (int **) malloc(N*sizeof(int*));
33
34     for (i=0; i<N; i++){
35         ma[i] = (int*) malloc(N*sizeof(int));
36         mb[i] = (int*) malloc(N*sizeof(int));
37         mc[i] = (int*) malloc(N*sizeof(int));
38     }
39
40     //inicialización de las matrices
41     for (i=0; i<N; i++){
42         for (j=0; j<N; j++){
43             ma[i][j] = 0; //aquí es donde se almacenará el resultado
44             mb[i][j] = 6;
45             mc[i][j] = 4;
46         }
47     }
48
49     double t1, t2, t_total;
50     t1 = omp_get_wtime();
51
52     //MULTIPLICACION
53     #pragma omp parallel for private(j,k)
54     for (i=0; i<N; i++){
55         for (j=0; j<N; j++){
56             for(k=0; k<N; k++){
57                 ma[i][j] += mb[i][k] * mc[k][j];
58             }
59         }
60     }
61
62     t2 = omp_get_wtime();
63     t_total = t2 - t1;
64
65     // Imprimir por pantalla el resultado de la primera y última línea de la matriz resultado (ma)
66     printf("Tiempo = %11.9f\t Primera = %d\t Ultima=%d\n",t_total,ma[0][0],ma[N-1][N-1]);
67
68     // Liberar la memoria
69     for (i=0; i<N; i++)
70     {
71         free(ma[i]);
72         free(mb[i]);
73         free(mc[i]);
74     }
75     free(ma);
76     free(mb);
77     free(mc);
78
79     return 0;
80 }

```

CAPTURAS DE PANTALLA:


```

guillesiesta@guillesiesta:~$ gcc -O2 -fopenmp pmm-OpenMP.c -o pmm0
guillesiesta@guillesiesta:~$ ./pmm0 5
Tiempo = 0.001421788      Primera = 120      Ultima=120
guillesiesta@guillesiesta:~$ ./pmm0 10
Tiempo = 0.003990086      Primera = 240      Ultima=240
guillesiesta@guillesiesta:~$ ./pmm0 20
Tiempo = 0.003785940      Primera = 480      Ultima=480
guillesiesta@guillesiesta:~$ ./pmm0 2000
Tiempo = 12.599555605     Primera = 48000     Ultima=48000
guillesiesta@guillesiesta:~$

```

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

ESTUDIO DE ESCALABILIDAD EN atcgrid:

SCRIPT: pmm-OpenMP_atcgrid.sh

```
1 #!/bin/bash
2
3 #PBS -N ej10_atcgrid
4 #PBS -q ac
5 echo "Id$PBS_O_WORKDIR usuario del trabajo: $PBS_O_LOGNAME"
6 echo "Id$PBS_O_WORKDIR del trabajo: $PBS_JOBID"
7 echo "Nombre del trabajo especificado por usuario: $PBS_JOBNAME"
8 echo "Nodo que ejecuta qsub: $PBS_O_HOST"
9 echo "Directorio en el que se ha ejecutado qsub: $PBS_O_WORKDIR"
10 echo "Cola: $PBS_QUEUE"
11 echo "Nodos asignados al trabajo:"
12 cat $PBS_NODEFILE
13
14 export OMP_SCHEDULE="static"
15 echo "static y chunk por defecto"
16 $PBS_O_WORKDIR/pmtv-OpenMP 15360
17
18
19 echo "secuencial"
20 $PBS_O_WORKDIR/pmm-secuencial 100
21 $PBS_O_WORKDIR/pmm-secuencial 500
22 $PBS_O_WORKDIR/pmm-secuencial 1000
23 $PBS_O_WORKDIR/pmm-secuencial 1500
24
25 echo "paralelo 2 threads"
26 export OMP_NUM_THREADS=2
27
28 $PBS_O_WORKDIR/pmm-OpenMP 100
29 $PBS_O_WORKDIR/pmm-OpenMP 500
30 $PBS_O_WORKDIR/pmm-OpenMP 1000
31 $PBS_O_WORKDIR/pmm-OpenMP 1500
32
33 echo "paralelo 4 threads"
34 export OMP_NUM_THREADS=4
35
36 $PBS_O_WORKDIR/pmm-OpenMP 100
37 $PBS_O_WORKDIR/pmm-OpenMP 500
38 $PBS_O_WORKDIR/pmm-OpenMP 1000
39 $PBS_O_WORKDIR/pmm-OpenMP 1500
```

ESTUDIO DE ESCALABILIDAD EN PCLOCAL:

SCRIPT: pmm-OpenMP_pcllocal.sh

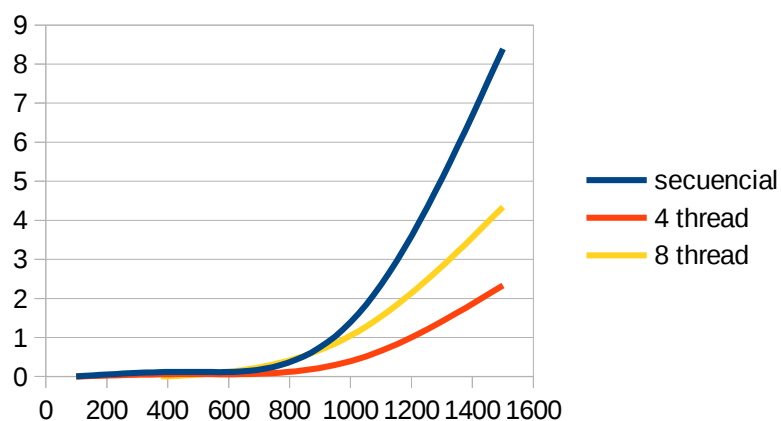
```

1 #!/./bash
2
3 echo "secuencial"
4 ./pmm 100
5 ./pmm 500
6 ./pmm 1000
7 ./pmm 1500
8
9 echo "paralelo 4 threads"
10 export OMP_NUM_THREADS=4
11
12 ./pmm0 100
13 ./pmm0 500
14 ./pmm0 1000
15 ./pmm0 1500
16
17 echo "paralelo 8 threads"
18 export OMP_NUM_THREADS=8
19
20 ./pmm0 100
21 ./pmm0 500
22 ./pmm0 1000
23 ./pmm0 1500

```

pcclocal

	secuencial	4 thread	8 thread
100	0,002928363	0,000350211	0,003100471
500	0,118825599	0,061507425	0,047626234
1000	1,388651759	0,393446265	1,04621322
1500	8,389792255	2,334081648	4,337541477



Se puede observar perfectamente que la versión secuencial es mucho peor que las demás. El porqué de cómo es mejor la versión de 4 threads que la de 8 puede ser debido a procesos de E/S en mi pc o al recargo por el reparto de trabajo en las hebras.