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): David Carrasco Chicharro
Grupo de prácticas: B1
Fecha de entrega: 13/05/2018
Fecha evaluación en clase: 14/05/2018

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`

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

CAPTURAS DE PANTALLA:

```
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP3:
2018-04-30 lunes
$ ./if-clauseModificado 4 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=6
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP3:
2018-04-30 lunes
$ ./if-clauseModificado 4 7
 thread 0 suma de a[0]=0 sumalocal=0
 thread 0 suma de a[1]=1 sumalocal=1
 thread 0 suma de a[2]=2 sumalocal=3
 thread 0 suma de a[3]=3 sumalocal=6
thread master=0 imprime suma=6
```

Ejecución con 4 iteraciones y 2 hebras, y 4 iteraciones y 7 hebras

```
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP3:
2018-04-30 lunes
$ ./if-clauseModificado 5 4
 thread 1 suma de a[2]=2 sumalocal=2
 thread 3 suma de a[4]=4 sumalocal=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 master=0 imprime suma=10
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP3:
2018-04-30 lunes
$ ./if-clauseModificado 10 8
 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 3 suma de a[5]=5 sumalocal=5
 thread 0 suma de a[0]=0 sumalocal=0
 thread 0 suma de a[1]=1 sumalocal=1
 thread 4 suma de a[6]=6 sumalocal=6
 thread 6 suma de a[8]=8 sumalocal=8
 thread 5 suma de a[7]=7 sumalocal=7
 thread 7 suma de a[9]=9 sumalocal=9
thread master=0 imprime suma=45
```

Ejecución con 5 iteraciones y 4 hebras, y 10 iteraciones y 8 hebras

RESPUESTA:

Con el primer parámetro de entrada se indican el número de iteraciones (máximo 20), y con el segundo, el de hebras (máximo 8), que paralelizan el bucle. De este modo se puede indicar el número de threads a utilizar sin necesidad de recompilar el programa.

Para que se paralelice el bucle `for` el número de iteraciones debe ser mayor que 4, ya que de usar menos no sería eficiente la creación, sincronización y destrucción de hebras.

El número de iteraciones se distribuye de forma más o menos equitativa entre el número de hebras establecido.

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)
- chunck= 1, 2 y 4

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

Iteración	schedule-clause.c			scheduled-claused.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	1	1	0	0	0
7	1	1	1	1	1	1	0	0	0
8	0	0	0	1	1	0	1	1	1
9	1	0	0	1	1	0	1	1	1
10	0	1	0	1	1	0	1	1	1
11	1	1	0	1	1	0	1	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	0	0
15	1	1	1	1	1	0	0	0	0

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

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

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

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

RESPUESTA:
La cláusula schedule(kind,chunk) distribuye y asigna las iteraciones del bucle entre las distintas hebras, siendo kind la forma de asignación y chunk la granularidad de la distribución.

- **static:** La distribución de hebras se realiza en tiempo de compilación, las iteraciones se dividen en unidades de *chunk iteraciones* y las unidades se asignan en round-robin. De este modo el número de iteraciones que realizará cada hebra está predefinido y será equitativo (si el tamaño es múltiplo el número de hebras estas realizarán el mismo número de iteraciones).
- **dynamic:** La distribución se realiza en tiempo de ejecución, luego no se puede saber en principio qué iteraciones realizará cada hebra. Sí se sabe que la unidad de distribución tiene *chunk iteraciones* → nº uds=O(n/chunk). Los threads más rápidos ejecutan más unidades, pero siempre ejecutarán como mínimo *chunk iteraciones* de las que le corresponden.
- **guided:** La distribución se realiza en tiempo de ejecución. Comienza con bloque largo → el tamaño del bloque va menguando (nº iteraciones que restan / nº threads), no más pequeño que chunk (excepto la última). Aquí también realizarán más ejecuciones las hebras más ociosas.

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`

```
 9 int main(int argc, char **argv) {
10   int i, n=200,chunk,chunk_value ,a[n],suma=0;
11   omp_sched_t kind;
12
13    if(argc < 3) {
14       fprintf(stderr,"\nFalta iteraciones o chunk \n");
15       exit(-1);
16    }
17    n = atoi(argv[1]); if (n>200) n=200;
18    chunk = atoi(argv[2]);
19
20    for (i=0; i<n; i++)       a[i] = i;
21
22    #pragma omp parallel for firstprivate(suma) \
23             lastprivate(suma) schedule(dynamic,chunk)
24    for (i=0; i<n; i++)
25    {    suma = suma + a[i];
26       printf(" thread %d suma a[%d]=%d suma=%d \n",
27          omp_get_thread_num(),i,a[i],suma);
28
29       if(i == 0){
30          printf(" Dentro de la region 'paralel for':\n");
31          omp_get_schedule(&kind, &chunk_value);
32          printf(" dyn-var: %d, nthreads-var: %d, thread-limit-var: %d, \
33            run-sched-var:%d, chunk:%d \n", omp_get_dynamic(), \
34            omp_get_max_threads(), omp_get_thread_limit(), kind, chunk_value);
35       }
36    }
```

```
37
38    printf("Fuera de 'parallel for' suma=%d\n",suma);
39    printf(" dyn-var: %d, nthreads-var: %d, thread-limit-var: %d, \
40       run-sched-var:%d, chunk:%d \n", omp_get_dynamic(), \
41       omp_get_max_threads(), omp_get_thread_limit(), kind, chunk_value);
42
43    return(0);
44 }
```

CAPTURAS DE PANTALLA:

```
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP3:2018-05-01 martes
$ export OMP_DYNAMIC=FALSE
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP3:2018-05-01 martes
$ export OMP_NUM_THREADS=4
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP3:2018-05-01 martes
$ export OMP_SHEDULE="static,4"
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP3:2018-05-01 martes
$ ./scheduled-clauseModificado 6 5
 thread 0 suma a[5]=5 suma=5
 thread 3 suma a[0]=0 suma=0
 Dentro de la region 'paralel for':
 dyn-var: 0, nthreads-var: 4, thread-limit-var: 2147483647, run-sched-var:2, chunk:1
 thread 3 suma a[1]=1 suma=1
 thread 3 suma a[2]=2 suma=3
 thread 3 suma a[3]=3 suma=6
 thread 3 suma a[4]=4 suma=10
Fuera de 'parallel for' suma=5
 dyn-var: 0, nthreads-var: 4, thread-limit-var: 2147483647, run-sched-var:2, chunk:1
```

```
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP3:2018-05-01 martes
$ export OMP_DYNAMIC=TRUE
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP3:2018-05-01 martes
$ export OMP_NUM_THREADS=8
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP3:2018-05-01 martes
$ export OMP_SCHEDULE="dynamic,4"
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP3:2018-05-01 martes
$ ./scheduled-clauseModificado 12 4
 thread 0 suma a[0]=0 suma=0
 Dentro de la region 'paralel for':
 dyn-var: 1, nthreads-var: 8, thread-limit-var: 2147483647, run-sched-var:2, chunk:4
 thread 0 suma a[1]=1 suma=1
 thread 0 suma a[2]=2 suma=3
 thread 0 suma a[3]=3 suma=6
 thread 1 suma a[4]=4 suma=4
 thread 1 suma a[5]=5 suma=9
 thread 1 suma a[6]=6 suma=15
 thread 1 suma a[7]=7 suma=22
 thread 2 suma a[8]=8 suma=8
 thread 2 suma a[9]=9 suma=17
 thread 2 suma a[10]=10 suma=27
 thread 2 suma a[11]=11 suma=38
Fuera de 'parallel for' suma=38
 dyn-var: 1, nthreads-var: 8, thread-limit-var: 2147483647, run-sched-var:2, chunk:4
```

```
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP3:2018-05-01 martes
$ export OMP_DYNAMIC=TRUE && export OMP_NUM_THREADS=8 && export OMP_SCHEDULE="static,2"
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP3:2018-05-01 martes
$ ./scheduled-clauseModificado 12 4
 thread 0 suma a[8]=8 suma=8
 thread 0 suma a[9]=9 suma=17
 thread 0 suma a[10]=10 suma=27
 thread 0 suma a[11]=11 suma=38
 thread 2 suma a[4]=4 suma=4
 thread 2 suma a[5]=5 suma=9
 thread 2 suma a[6]=6 suma=15
 thread 2 suma a[7]=7 suma=22
 thread 1 suma a[0]=0 suma=0
 Dentro de la region 'paralel for':
 dyn-var: 1, nthreads-var: 8, thread-limit-var: 2147483647, run-sched-var:1, chunk:2
 thread 1 suma a[1]=1 suma=1
 thread 1 suma a[2]=2 suma=3
 thread 1 suma a[3]=3 suma=6
Fuera de 'parallel for' suma=38
 dyn-var: 1, nthreads-var: 8, thread-limit-var: 2147483647, run-sched-var:1, chunk:2
```

RESPUESTA:
Tanto dentro como fuera de la región paralela se imprimen los mismos valores, correspondiéndose estos con los valores establecidos previamente en las variables de entorno .

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`

```
 9 int main(int argc, char **argv) {
10   int i, n=200,chunk,chunk_value ,a[n],suma=0;
11   omp_sched_t kind;
12
13    if(argc < 3) {
14       fprintf(stderr,"\nFalta iteraciones o chunk \n");
15       exit(-1);
16    }
17    n = atoi(argv[1]); if (n>200) n=200;
18    chunk = atoi(argv[2]);
19
20    for (i=0; i<n; i++)       a[i] = i;
21
22    #pragma omp parallel for firstprivate(suma) \
23             lastprivate(suma) schedule(dynamic,chunk)
24    for (i=0; i<n; i++)
25    {   suma = suma + a[i];
26       printf(" thread %d suma a[%d]=%d suma=%d \n",
27          omp_get_thread_num(),i,a[i],suma);
28
29       if(i == 0){
30          printf(" Dentro de la region 'paralel for':\n");
31          omp_get_schedule(&kind, &chunk_value);
32          printf(" \tNúmero de threads usados: %d\n",
33             omp_get_num_threads() );
34          printf(" \tNúmero de procesadores disponibles: %d\n",
35             omp_get_num_procs() );
36          if(omp_in_parallel() == 0)
37             printf(" \t¿Region paralela?: false\n");
38          else
39             printf(" \t¿Region paralela?: true\n");
40          printf(" dyn-var: %d, nthreads-var: %d, thread-limit-var: %d, \
41             run-sched-var:%d, chunk:%d \n", omp_get_dynamic(), \
42             omp_get_max_threads(), omp_get_thread_limit(), kind, chunk_value);
43       }
44    }
45
46    printf("Fuera de 'parallel for' suma=%d\n",suma);
47    printf(" \tNúmero de threads usados: %d\n", omp_get_num_threads() );
48    printf(" \tNúmero de procesadores disponibles: %d\n", omp_get_num_procs() );
49    if(omp_in_parallel() == 0)
50       printf(" \t¿Region paralela?: false\n");
51    else
52       printf(" \t¿Region paralela?: true\n");
53    printf(" dyn-var: %d, nthreads-var: %d, thread-limit-var: %d, \
54          run-sched-var:%d, chunk:%d \n", omp_get_dynamic(), \
55          omp_get_max_threads(), omp_get_thread_limit(), kind, chunk_value);
```

CAPTURAS DE PANTALLA:

```
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP3:2018-05-01 martes
$ export OMP_DYNAMIC=TRUE && export OMP_NUM_THREADS=4 && export OMP_SCHEDULE="dynamic,2"
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP3:2018-05-01 martes
$ ./scheduled-clauseModificado4 6 3
 thread 0 suma a[3]=3 suma=3
 thread 0 suma a[4]=4 suma=7
 thread 0 suma a[5]=5 suma=12
 thread 2 suma a[0]=0 suma=0
 Dentro de la region 'paralel for':
        Número de threads usados: 4
        Número de procesadores disponibles: 4
        ¿Region paralela?: true
 dyn-var: 1, nthreads-var: 4, thread-limit-var: 2147483647, run-sched-var:2, chunk:2
 thread 2 suma a[1]=1 suma=1
 thread 2 suma a[2]=2 suma=3
Fuera de 'parallel for' suma=12
        Número de threads usados: 1
        Número de procesadores disponibles: 4
        ¿Region paralela?: false
 dyn-var: 1, nthreads-var: 4, thread-limit-var: 2147483647, run-sched-var:2, chunk:2
```

RESPUESTA:
En este caso las funciones `omp_get_num_threads()` y `omp_in_parallel()` devuelven valores distintos, puesto que la primera da el número de threads usados en la región paralela (4) y en la región no paralela (1, que es el valor que se obtendrá siempre al llamar a esta función fuera de una región `parallel`); la segunda función, que da un valor distinto, nos dice si la estamos llamando desde una región paralela o fuera de ella, luego, como era de esperar, devuelve `true` en la primera llamada y `false` en la segunda.

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`

```
 9 int main(int argc, char **argv) {
10   int i, n=200,chunk,chunk_value ,a[n],suma=0;
11   omp_sched_t kind;
12
13    if(argc < 3) {
14       fprintf(stderr,"\nFalta iteraciones o chunk \n");
15       exit(-1);
16    }
17    n = atoi(argv[1]); if (n>200) n=200;
18    chunk = atoi(argv[2]);
19
20    for (i=0; i<n; i++)       a[i] = i;
21
22    #pragma omp parallel for firstprivate(suma) \
23                lastprivate(suma) schedule(dynamic,chunk)
24       for (i=0; i<n; i++)
25       {   suma = suma + a[i];
26          printf(" thread %d suma a[%d]=%d suma=%d \n",
27             omp_get_thread_num(),i,a[i],suma);
28
```

```
29
30          if( i==(n-2) ){
31             omp_set_dynamic(0);
32             omp_set_num_threads(2);
33             omp_set_schedule(1,2);
34          }
35          if( i==0 || i==(n-2) ){
36             if (i==0)
37                printf(" Antes del cambio:\n");
38             else
39                printf(" Después del cambio:\n");
40
41             omp_get_schedule(&kind, &chunk_value);
42             printf(" dyn-var: %d, nthreads-var: %d, run-sched-var:%d, chunk:%d \n", \
43                omp_get_dynamic(), omp_get_max_threads(), kind, chunk_value);
44          }
45       }
46
47    printf(" \nFuera de 'parallel for' suma=%d\n",suma);
48    printf(" dyn-var: %d, nthreads-var: %d, run-sched-var:%d, chunk:%d \n", \
49             omp_get_dynamic(), omp_get_max_threads(), kind, chunk_value);
50
51    return(0);
52 }
```

CAPTURAS DE PANTALLA:

```
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP3:2018-05-01 martes
$ export OMP_DYNAMIC=TRUE && export OMP_NUM_THREADS=4 && export OMP_SCHEDULE="dynamic,3"
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP3:2018-05-01 martes
$ ./scheduled-clauseModificado5 16 4
 thread 0 suma a[0]=0 suma=0
 Antes del cambio:
 dyn-var: 1, nthreads-var: 4, run-sched-var:2, chunk:3
 thread 0 suma a[1]=1 suma=1
 thread 0 suma a[2]=2 suma=3
 thread 0 suma a[3]=3 suma=6
 thread 3 suma a[12]=12 suma=12
 thread 3 suma a[13]=13 suma=25
 thread 3 suma a[14]=14 suma=39
 Después del cambio:
 dyn-var: 0, nthreads-var: 2, run-sched-var:1, chunk:2
 thread 3 suma a[15]=15 suma=54
 thread 2 suma a[8]=8 suma=8
 thread 2 suma a[9]=9 suma=17
 thread 2 suma a[10]=10 suma=27
 thread 2 suma a[11]=11 suma=38
 thread 1 suma a[4]=4 suma=4
 thread 1 suma a[5]=5 suma=9
 thread 1 suma a[6]=6 suma=15
 thread 1 suma a[7]=7 suma=22

Fuera de 'parallel for' suma=54
 dyn-var: 1, nthreads-var: 4, run-sched-var:1, chunk:2
```

RESPUESTA:
Antes de modificar las variables de control, sus valores son los proporcionados por las variables de entorno. Tras el cambio, los valores son los proporcionados por las funciones en tiempo de ejecución. Hay que destacar que fuera de la región `parallel` las variables `dyn-var` y `nthreads-var` obtienen el valor que poseían antes de haber sido alteradas, pero las variables `run-sched-var` y `chunk` mantienen el último que se les proporcionó.

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`

```
15 int main(int argc, char **argv) {
16
17   struct timespec cgt1,cgt2;
18   double ncgt; //para tiempo de ejecución
19
20    if(argc < 2){
21       fprintf(stderr,"Falta el tamaño\n");
22       exit(-1);
23    }
24
25    unsigned int N = atoi(argv[1]);
26    int fil, col, suma;
27
28    long long unsigned int **M, *v1, *v_res;
29    M = (long long unsigned int**) malloc(N*sizeof(long long unsigned int*));
30    for(long long unsigned int i=0 ; i<N ; i++)
31       M[i] = (long long unsigned int*) malloc(N*sizeof(long long unsigned int));
32
33    v1 = (long long unsigned int*) malloc(N*sizeof(long long unsigned int));
34    v_res = (long long unsigned int*) malloc(N*sizeof(long long unsigned int));
35
36    if ( (M==NULL) || (v1==NULL) || (v_res==NULL) ){
37       printf("Error en la reserva de espacio para los vectores\n");
38       exit(-2);
39    }
40
41    // Inicialización de la matriz y los vectores
42    for(fil=0 ; fil<N ; fil++){
43       for(int col=0 ; col<N ; col++)
44          M[fil][col]=0;
45    for(fil=0 ; fil<N ; fil++){
46       for(int col=fil ; col<N ; col++)
47          M[fil][col]=fil+col;
48
49       v1[fil]=fil;
50       v_res[fil]=0;
51    }
52
53    clock_gettime(CLOCK_REALTIME,&cgt1);
54    // Cálculo de v_res
55    for (fil=0 ; fil<N; fil++){
56       suma=0;
57       for(col=fil ; col<N ; col++)
58          suma += M[fil][col] * v1[col];
59
60       v_res[fil] = suma;
61    }
62
63    clock_gettime(CLOCK_REALTIME,&cgt2);
64    ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+
65       (double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));
66
67    printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\n",ncgt,N);
68    if(N<20)
69       for(fil=0 ; fil<N ; fil++){
70          printf("v_res[%d]=%llu  ", fil, v_res[fil]);
71       }
72    else{
73       printf("v_res[0]=%llu  ", v_res[0]);
74       printf("v_res[%d]=%llu  ", N-1, v_res[N-1]);
75    }
76    printf("\n");
77
78    for(int i=0 ; i<N ; i++)
79       free(M[i]);
80    free(M);
81    free(v1); // libera el espacio reservado para v1
82    free(v_res); // libera el espacio reservado para v_res
83
84    return 0;
```

CAPTURAS DE PANTALLA:

```
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP3:2018-05-01 martes
$ ./pmv-secuencial 1000
Tiempo(seg.):0.001112953         / Tamaño Vectores:1000

David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP3:2018-05-01 martes
$ ./pmtv-secuencial 1000
Tiempo(seg.):0.000790748         / Tamaño Vectores:1000
v_res[0]=332833500  v_res[999]=1996002
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP3:2018-05-01 martes
$ ./pmv-secuencial 10000
Tiempo(seg.):0.099147976         / Tamaño Vectores:10000

David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP3:2018-05-01 martes
$ ./pmtv-secuencial 10000
Tiempo(seg.):0.055465957         / Tamaño Vectores:10000
v_res[0]=18446744071985437528  v_res[9999]=199960002
```

RESPUESTA:
La nueva versión, pmtv-secuencial, es más eficiente que la versión pmv-secuencial. Ambas tienen un orden de eficiencia $O\left(\frac{N^2+N}{2}\right)$ y $O(N^2)$ respectivamente.

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:

a) Por defecto el chunk es 0 para static y 1 para dynamic y guided. Para averiguar estos valores, habiendo utilizado previamente la asignación de la alternativa correspondiente a la variable de entorno `export OMP_SCHEDULE`, se han ejecutado las siguientes instrucciones dentro de la región paralela,:

```
omp_get_schedule(&kind, &chunk);
printf(" run-sched-var:%d, chunk:%d \n", kind, chunk);
```

b) Varía según la fila, ya que la matriz es triangular; por ejemplo para la fila 1 se realizan N operaciones, para la fila 2, N-1, y así sucesivamente hasta la última, en la

que sólo se realiza una.

c) Dependerá de la disponibilidad de las hebras, es decir, cada hebra hará un numero de multiplicaciones y sumas distintas, dependiendo de cuales de ellas se encuentren ocupadas y cuales libres.

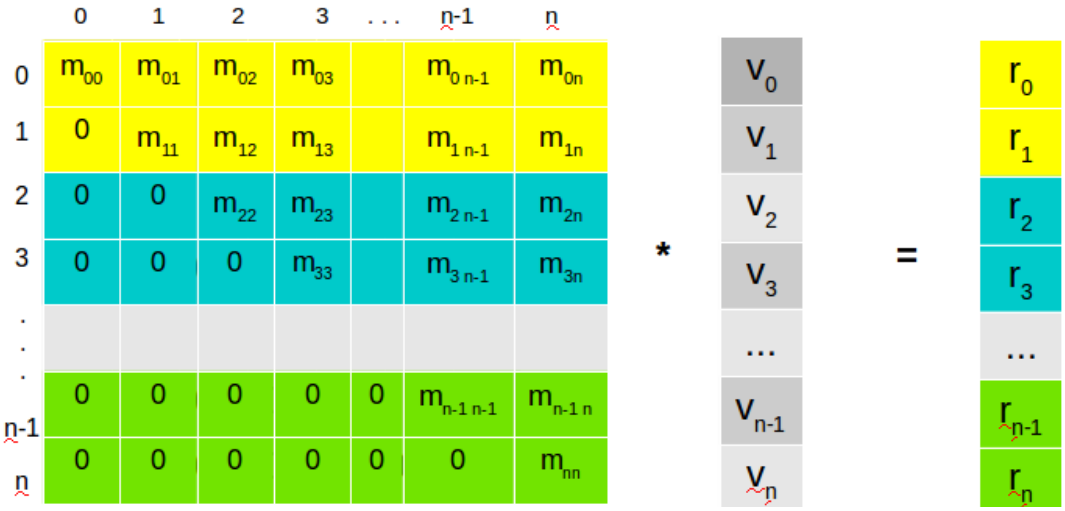
En cuanto a la alternativa que ofrece mejores prestaciones, tal y como se muestra en los resultados que se presentan a continuación, es mejor la que usa una planificación `dynamic` pues las iteraciones se dividen en unidades de chunk iteraciones y estas unidades se asignan en tiempo de ejecución. Los threads más rápidos ejecutan las que están libres, por lo que se acelera el proceso y por ello presenta mejores resultados en cuanto al tiempo de ejecución.

CAPTURA CÓDIGO FUENTE: `pmtv-OpenMP.c`

```
21 int main(int argc, char **argv) {
22
23    double t_ini, t_fin, t_elapsed; //para tiempo de ejecución
24    int fil, col, suma;
25    unsigned int N = atoi(argv[1]);
26
27    if(argc < 2) {
28       fprintf(stderr,"\nFalta iteraciones\n");
29       exit(-1);
30    }
31
32    long long unsigned int **M, *v1, *v_res;
33    M = (long long unsigned int**) malloc(N*sizeof(long long unsigned int*));
34    for(long long unsigned int i=0 ; i<N ; i++)
35       M[i] = (long long unsigned int*) malloc(N*sizeof(long long unsigned int));
36
37    v1 = (long long unsigned int*) malloc(N*sizeof(long long unsigned int));
38    v_res = (long long unsigned int*) malloc(N*sizeof(long long unsigned int));
39
40    if ( (M==NULL) || (v1==NULL) || (v_res==NULL) ){
41       printf("Error en la reserva de espacio para los vectores\n");
42       exit(-2);
43    }
44
45    // Inicialización de la matriz y los vectores
46    for(fil=0 ; fil<N ; fil++)
47       for(int col=0 ; col<N ; col++)
48          M[fil][col]=0;
49    for(fil=0 ; fil<N ; fil++){
50       for(int col=fil ; col<N ; col++)
51          M[fil][col]=fil+col;
52
53       v1[fil]=fil;
54       v_res[fil]=0;
55    }
56
57    #pragma omp parallel
58    {
59       #pragma omp single
60       t_ini = omp_get_wtime();
61
62       #pragma omp for firstprivate(suma) schedule(runtime)
63          for (fil=0 ; fil<N; fil++){
64             suma=0;
65             for(col=fil ; col<N ; col++)
66                suma += M[fil][col] * v1[col];
67
68             v_res[fil] = suma;
69          }
70
71       #pragma omp single
72       t_fin = omp_get_wtime();
73    }
74
```

```
75    t_elapsed = t_fin - t_ini;
76
77    printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\n",t_elapsed,N);
78    if(N<20)
79       for(fil=0 ; fil<N ; fil++){
80          printf("v_res[%d]=%llu  ", fil, v_res[fil]);
81       }
82    else{
83       printf("v_res[0]=%llu  ", v_res[0]);
84       printf("v_res[%d]=%llu  ", N-1, v_res[N-1]);
85    }
86    printf("\n");
87
88    for(int i=0 ; i<N ; i++)
89       free(M[i]);
90    free(M);
91    free(v1); // libera el espacio reservado para v1
92    free(v_res); // libera el espacio reservado para v_res
93
94    return 0;
95 }
```

DESCOMPOSICIÓN DE DOMINIO:



Supongamos que, por ejemplo, hacemos una asignación estática (round robin) y fijamos el chunk=2, para cualquier número de threads. Cada thread multiplicaría por el vector v dos filas de la matriz de forma consecutiva. En la representación gráfica se muestra la descomposición por colores.

CAPTURAS DE PANTALLA:

```
David Carrasco Chicharro B1estudiante4@atcgrid:~:2018-05-07 lunes
$ cat resultado_ej7.o77067
Id/home/B1estudiante4 usuario del trabajo: B1estudiante4
Id/home/B1estudiante4 del trabajo: 77067.atcgrid
Nombre del trabajo especificado por usuario: resultado_ej7
Nodo que ejecuta qsub: atcgrid
Directorio en el que se ha ejecutado qsub: /home/B1estudiante4
Cola: ac
```

```
static y chunk por defecto
Tiempo(seg.):0.066338748         / Tamaño Vectores:15360
v_res[0]=955779584  v_res[15359]=471797762


static y chunk 1
Tiempo(seg.):0.058685266         / Tamaño Vectores:15360
v_res[0]=955779584  v_res[15359]=471797762

static y chunk 64
Tiempo(seg.):0.058621149         / Tamaño Vectores:15360
v_res[0]=955779584  v_res[15359]=471797762


dynamic y chunk por defecto
Tiempo(seg.):0.055882150         / Tamaño Vectores:15360
v_res[0]=955779584  v_res[15359]=471797762


dynamic y chunk 1
Tiempo(seg.):0.055965117         / Tamaño Vectores:15360
v_res[0]=955779584  v_res[15359]=471797762


dynamic y chunk 64
Tiempo(seg.):0.053638433         / Tamaño Vectores:15360
v_res[0]=955779584  v_res[15359]=471797762


guided y chunk por defecto
Tiempo(seg.):0.070453160         / Tamaño Vectores:15360
v_res[0]=955779584  v_res[15359]=471797762


guided y chunk 1
Tiempo(seg.):0.058619324         / Tamaño Vectores:15360
v_res[0]=955779584  v_res[15359]=471797762


guided y chunk 64
Tiempo(seg.):0.072450520         / Tamaño Vectores:15360
v_res[0]=955779584  v_res[15359]=471797762
```

TABLA RESULTADOS, SCRIPT Y GRÁFICA atcgrid

SCRIPT: pmtv-OpenMP_PCaula.sh

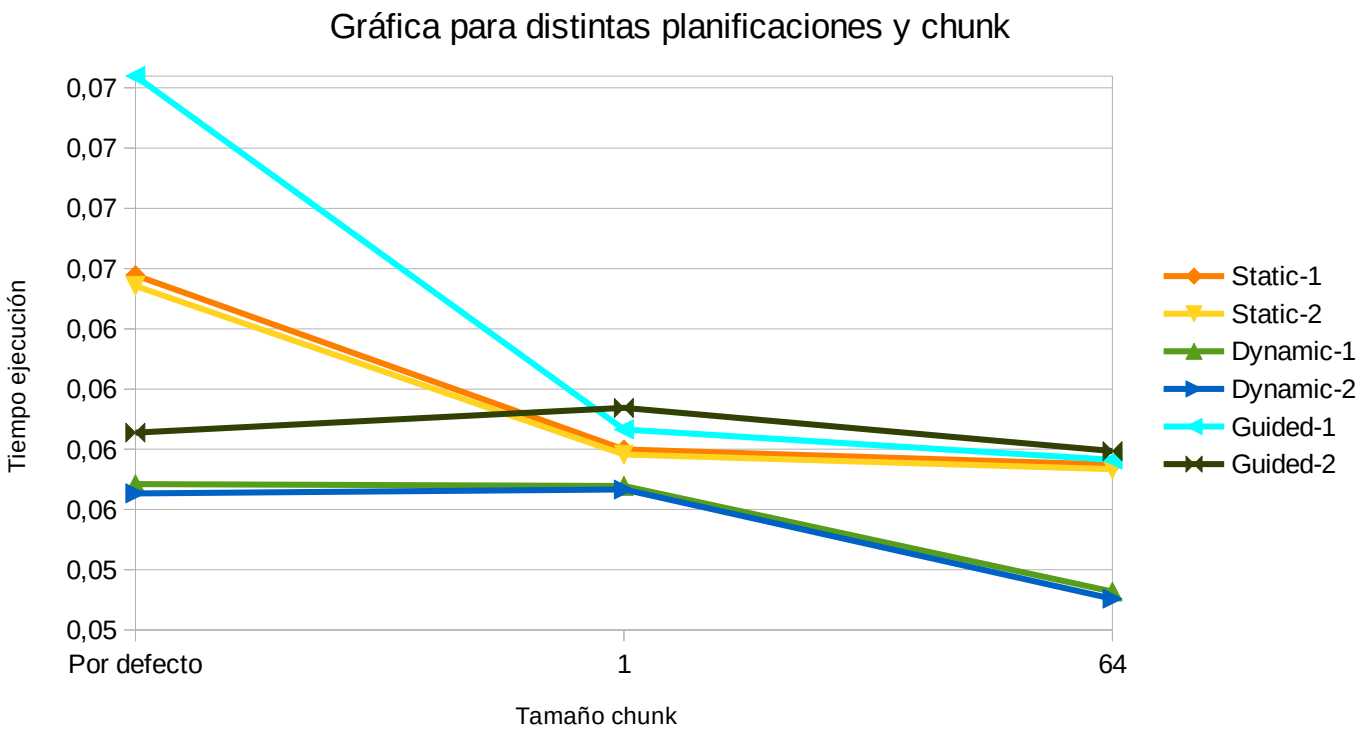
```
#!/bin/bash
#Se asigna al trabajo el nombre resultado_ej7
#PBS -N resultado_ej7
#Se asigna al trabajo la cola ac
#PBS -q ac
#Se imprime información del trabajo usando variables de entorno de PBS
echo "Id$PBS_O_WORKDIR usuario del trabajo: $PBS_O_LOGNAME"
echo "Id$PBS_O_WORKDIR del trabajo: $PBS_JOBID"
echo "Nombre del trabajo especificado por usuario: $PBS_JOBNAME"
echo "Nodo que ejecuta qsub: $PBS_O_HOST"
```

```
11 echo "Directorio en el que se ha ejecutado qsub: $PBS_O_WORKDIR"
12 echo "Cola: $PBS_QUEUE"
13 echo "Nodos asignados al trabajo:"
14 cat $PBS_NODEFILE
15
16 export OMP_NUM_THREADS=12
17
18 declare -a planificacion=("static" "dynamic" "guided")
19
20 for i in "${planificacion[@]}"
21 do
22    for (( j = 0; j < 3; ++j )); do
23       if [ $j -eq 0 ]
24       then
25          echo "$i y chunk por defecto"
26          export OMP_SCHEDULE=$i
27       elif [ $j -eq 1 ]
28       then
29          echo "$i y chunk 1"
30          export OMP_SCHEDULE="$i,1"
31       else
32          echo "$i y chunk 64"
33          export OMP_SCHEDULE="$i,64"
34       fi
35       $PBS_O_WORKDIR/pmtv-OpenMP 15360
36       echo -e "\n"
37    done
38 done
```

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	0.066727698	0.058056343	0.074995137
1	0.059496218	0.057982421	0.060328342
64	0.058865992	0.053595665	0.059065246
Chunk	**Static**	**Dynamic**	**Guided**
por defecto	0.066286693	0.057670952	0.070842884
1	0.059282904	0.057831816	0.064885012
64	0.058669628	0.053298571	0.077257145

GRÁFICA:



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

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

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`

```
15 int main(int argc, char **argv) {
16
17   struct timespec cgt1,cgt2;
18   double ncgt; //para tiempo de ejecución
19
20    if(argc < 2){
21       fprintf(stderr,"Falta el tamaño\n");
22       exit(-1);
23    }
24
25    unsigned int N = atoi(argv[1]);
26    int fil, col, suma;
27
28
29
30    long long unsigned int **M1, **M2, **M3;
31    M1 = (long long unsigned int**) malloc(N*sizeof(long long unsigned int*));
32    M2 = (long long unsigned int**) malloc(N*sizeof(long long unsigned int*));
33    M3 = (long long unsigned int**) malloc(N*sizeof(long long unsigned int*));
```

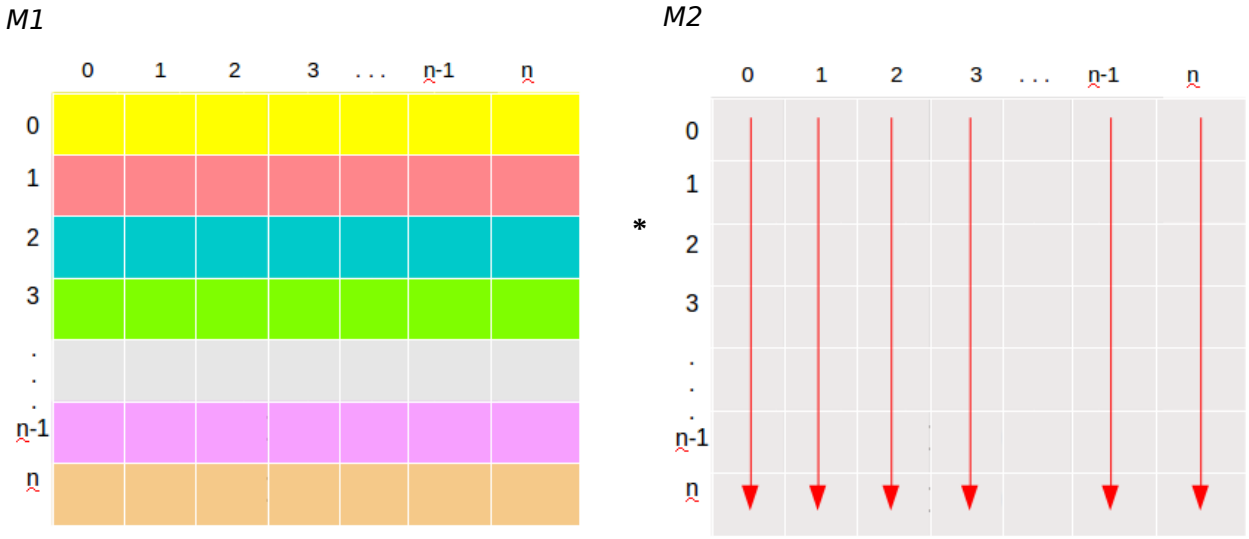
```
34    for(long long unsigned int i=0 ; i<N ; i++){
35       M1[i] = (long long unsigned int*) malloc(N*sizeof(long long unsigned int));
36       M2[i] = (long long unsigned int*) malloc(N*sizeof(long long unsigned int));
37       M3[i] = (long long unsigned int*) malloc(N*sizeof(long long unsigned int));
38    }
39
40    if ( (M1==NULL) || (M2==NULL) || (M3==NULL) ){
41       printf("Error en la reserva de espacio para las matrices\n");
42       exit(-2);
43    }
44
45    // Inicialización de las matrices
46    for(fil=0 ; fil<N ; fil++)
47       for(col=0 ; col<N ; col++){
48          M1[fil][col]=fil+col;
49          M2[fil][col]=fil+col;
50          M3[fil][col]=0;
51       }
52
53    clock_gettime(CLOCK_REALTIME,&cgt1);
54    // Cálculo de M3
55    for (int i=0 ; i<N; i++)
56       for(int j=0 ; j<N ; j++)
57          for(int k=0 ; k<N ; k++)
58             M3[i][j] += M1[i][k] * M2[k][j];
59
60
61    clock_gettime(CLOCK_REALTIME,&cgt2);
62    ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+
63      (double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));
64
65    printf("Tiempo(seg.):%11.9f\t / Tamaño Matrices:%u\n",ncgt,N);
66    printf("M3[0][0]=%llu , M3[%d][%d]=%llu \n", \
67       M3[0][0], N-1, N-1, M3[N-1][N-1]);
68
69    for(int i=0 ; i<N ; i++){
70       free(M1[i]);
71       free(M2[i]);
72       free(M3[i]);
73    }
74    free(M1);   free(M2);   free(M3);
75
76    return 0;
77 }
```

CAPTURAS DE PANTALLA:

```
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP3:
2018-05-07 lunes
$ ./pmm-secuencial 30
Tiempo(seg.):0.000066433         / Tamaño Matrices:30
M3[0][0]=8555 , M3[29][29]=59015
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP3:
2018-05-07 lunes
$ ./pmm-secuencial 100
Tiempo(seg.):0.002244431         / Tamaño Matrices:100
M3[0][0]=328350 , M3[99][99]=2288550
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP3:
2018-05-07 lunes
$ ./pmm-secuencial 1000
Tiempo(seg.):3.127944160         / Tamaño Matrices:1000
M3[0][0]=332833500 , M3[999][999]=2328835500
```

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:



La tarea que realiza cada thread es la de multiplicar cada casilla de cada fila de la matriz M1 por cada casilla de cada columna de la matriz M2 y sumar en la casilla correspondiente de la matriz resultado, M3, el resultado de la multiplicación realizada.
De este modo, cada thread se correspondería con cada color de M1, que recorre dicha fila, además de todas las columnas de M2 y escribiría resultados en la misma fila de la matriz resultado, siguiendo el mismo patrón que se refleja en los colores de M1.

CAPTURA CÓDIGO FUENTE: `pmm-OpenMP.c`

```
21 int main(int argc, char **argv) {
22
23    double t_ini, t_fin, t_elapsed; //para tiempo de ejecución
24
25    if(argc < 2){
26       fprintf(stderr,"Falta el tamaño\n");
27       exit(-1);
28    }
29
30    unsigned int N = atoi(argv[1]);
31    int fil, col, chunk;
32    omp_sched_t kind;
33
34    long long unsigned int **M1, **M2, **M3;
35    M1 = (long long unsigned int**) malloc(N*sizeof(long long unsigned int*));
36    M2 = (long long unsigned int**) malloc(N*sizeof(long long unsigned int*));
37    M3 = (long long unsigned int**) malloc(N*sizeof(long long unsigned int*));
38    for(long long unsigned int i=0 ; i<N ; i++){
39       M1[i] = (long long unsigned int*) malloc(N*sizeof(long long unsigned int));
40       M2[i] = (long long unsigned int*) malloc(N*sizeof(long long unsigned int));
41       M3[i] = (long long unsigned int*) malloc(N*sizeof(long long unsigned int));
42    }
43
44    if ( (M1==NULL) || (M2==NULL) || (M3==NULL) ){
45       printf("Error en la reserva de espacio para las matrices\n");
46       exit(-2);
47    }
48
49    #pragma omp parallel
50    {
51       // Inicialización de las matrices
52       #pragma omp for firstprivate(col) schedule(runtime)
53       for(fil=0 ; fil<N ; fil++)
54          for(col=0 ; col<N ; col++){
55             M1[fil][col]=fil+col;
56             M2[fil][col]=fil+col;
57             M3[fil][col]=0;
58          }
```

```
59
60       #pragma omp single
61       t_ini = omp_get_wtime();
62
63       #pragma omp for schedule(runtime)
64       for (int i=0 ; i<N; i++)
65          for(int j=0 ; j<N ; j++)
66             for(int k=0 ; k<N ; k++)
67                M3[i][j] += M1[i][k] * M2[k][j];
68
69       #pragma omp single
70       {
71          t_fin = omp_get_wtime();
72          omp_get_schedule(&kind, &chunk);
73          printf(" run-sched-var:%d, chunk:%d \n", kind, chunk);
74       }
75    }
76
77    t_elapsed = t_fin - t_ini;
78
79
80    printf("Tiempo(seg.):%11.9f\t / Tamaño Matrices:%u\n",t_elapsed,N);
81    printf("M3[0][0]=%llu , M3[%d][%d]=%llu \n", \
82       M3[0][0], N-1, N-1, M3[N-1][N-1]);
83
84    for(int i=0 ; i<N ; i++){
85       free(M1[i]);
86       free(M2[i]);
87       free(M3[i]);
88    }
89    free(M1);   free(M2);   free(M3);
90
91    return 0;
92 }
```

CAPTURAS DE PANTALLA:

```
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP3:
2018-05-07 lunes
$ ./pmm-OpenMP 30
 run-sched-var:2, chunk:1
Tiempo(seg.):0.000060079          / Tamaño Matrices:30
M3[0][0]=8555 , M3[29][29]=59015
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP3:
2018-05-07 lunes
$ ./pmm-OpenMP 100
 run-sched-var:2, chunk:1
Tiempo(seg.):0.001945997          / Tamaño Matrices:100
M3[0][0]=328350 , M3[99][99]=2288550
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP3:
2018-05-07 lunes
$ ./pmm-OpenMP 1000
 run-sched-var:2, chunk:1
Tiempo(seg.):1.771085626          / Tamaño Matrices:1000
M3[0][0]=332833500 , M3[999][999]=2328835500
```

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

```
#!/bin/bash
#Uso: ./pmm-OpenMP-pclocal.sh
#Se asigna al trabajo el nombre res_ej10_pclocal
#PBS -N res_ej10_pclocal
#Se asigna al trabajo la cola ac
#PBS -q ac
#Se imprime información del trabajo usando variables de entorno de PBS
echo "Id$PBS_O_WORKDIR usuario del trabajo: $PBS_O_LOGNAME"
echo "Id$PBS_O_WORKDIR del trabajo: $PBS_JOBID"
echo "Nombre del trabajo especificado por usuario: $PBS_JOBNAME"
echo "Nodo que ejecuta qsub: $PBS_O_HOST"
echo "Directorio en el que se ha ejecutado qsub: $PBS_O_WORKDIR"
echo "Cola: $PBS_QUEUE"

export OMP_DYNAMIC=FALSE
export OMP_NUM_THREADS=1
echo "Ejecucion: static"

for (( i = 100; i <= 1500; i +=200 )); do
   $PBS_O_WORKDIR/pmm-OpenMP $i
done

echo -e "\n"
export OMP_DYNAMIC=TRUE
declare -a threads=(2 6 12)

for i in "${threads[@]}"
do
   export OMP_NUM_THREADS=$i
   echo -e "\n Ejecucion: dynamic - Num. threads = $i"
   for (( j = 100; j <= 1500; j+=200 )); do
      $PBS_O_WORKDIR/pmm-OpenMP $j
   done
done
```

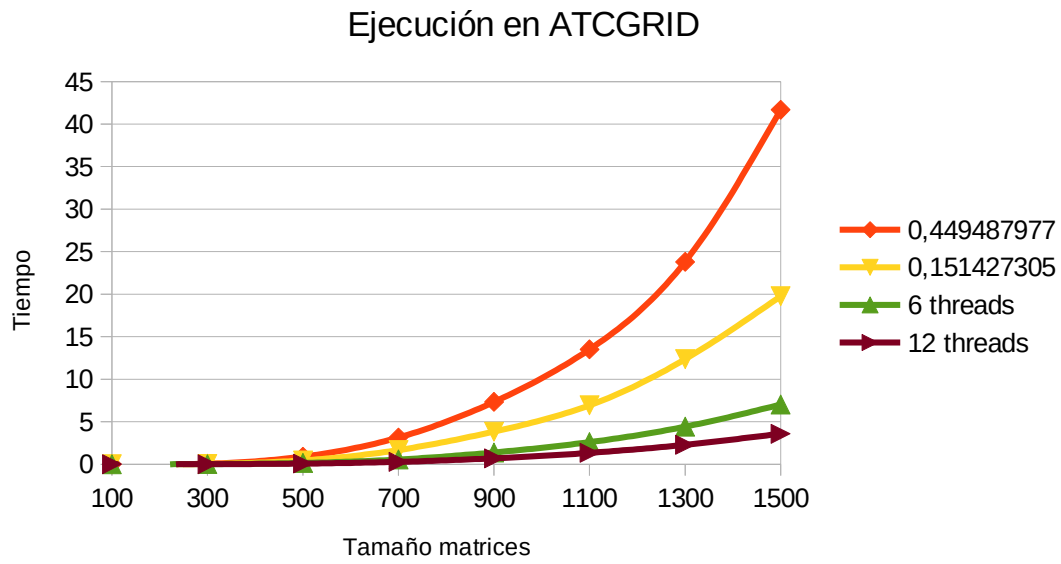
Tiempos de ejecución

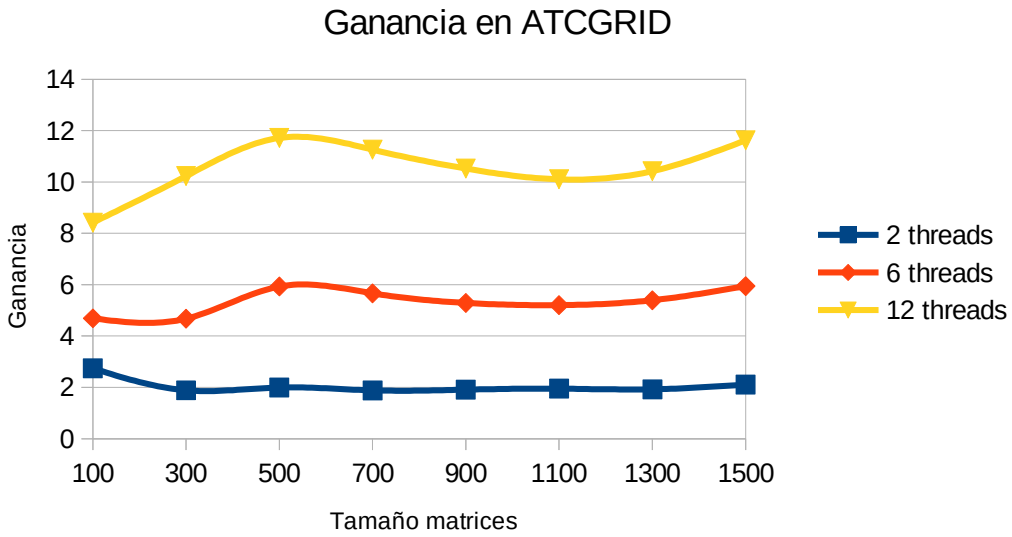
	1 thread	2 threads	6 threads	12 threads
100	0,002291116	0,000835726	0,000488585	0,000272223
300	0,061665767	0,032699398	0,013186405	0,006028771
500	0,897430269	0,449487977	0,151427305	0,076608783
700	3,139691563	1,667738588	0,554468854	0,278867828
900	7,328681355	3,837485665	1,385986549	0,69656516
1100	13,508488329	6,933593564	2,595726846	1,337295778
1300	23,799706214	12,354624058	4,413120911	2,283656209
1500	41,677947499	19,751226744	7,013371938	3,586483849

Ganancia

2 threads	6 threads	12 threads
2,74146789737306	4,68928845543764	8,41632044316608
1,88583799004495	4,6764654202567	10,2285800870526
1,99656123171455	5,9264758690647	11,7144566700661
1,88260413567885	5,66252106020007	11,2587084193878
1,90976123294522	5,28770020191588	10,5211712785061
1,94826653802403	5,20412552261287	373,888864547062
1,92638044688935	5,3929422497076	306,525998633799
2,11014475400425	5,94264041140891	11,6208379163957

Gráficas





ESTUDIO DE ESCALABILIDAD EN PCLOCAL:

SCRIPT: pmm-OpenMP_pclocal.sh

```
#!/bin/bash
#Uso: ./pmm-OpenMP-pclocal.sh <ejecutable>

export OMP_DYNAMIC=FALSE
export OMP_NUM_THREADS=1
echo "Ejecucion: static"

for (( i = 100; i <= 1500; i +=200 )); do
   ./$1 $i
done
```

```
12 echo -e "\n"
13 export OMP_DYNAMIC=TRUE
14
15 for (( i = 2 ; i <= 4 ; i+=2 )); do
16    export OMP_NUM_THREADS=$i
17    echo -e "\n Ejecucion: dynamic - Num. threads = $i"
18    for (( j = 100; j <= 1500; j+=200 )); do
19       ./$1 $j
20    done
21 done
```

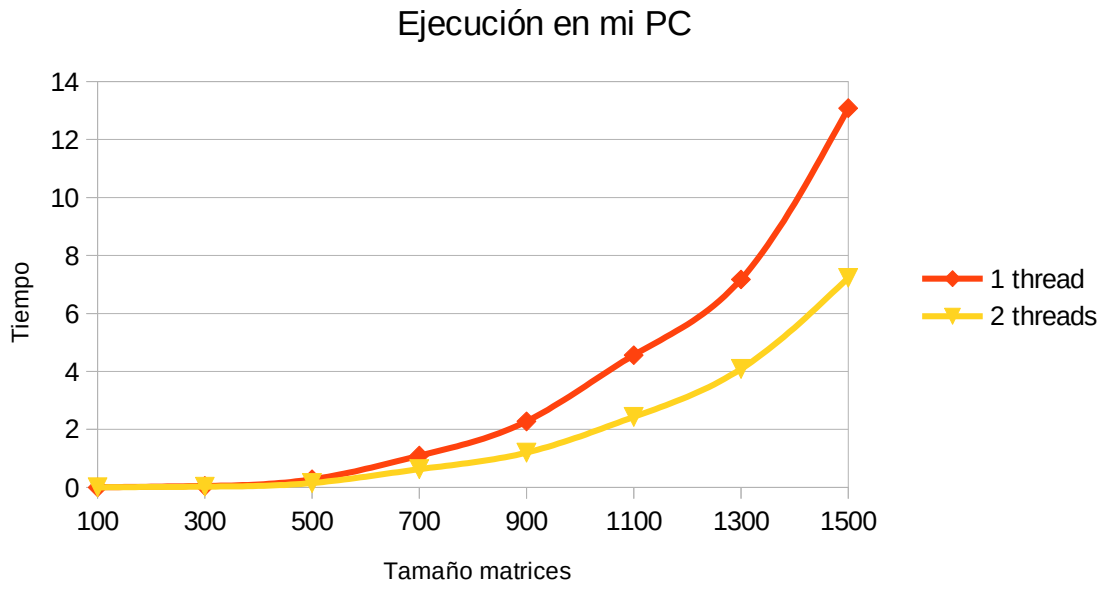
Tiempos de ejecución

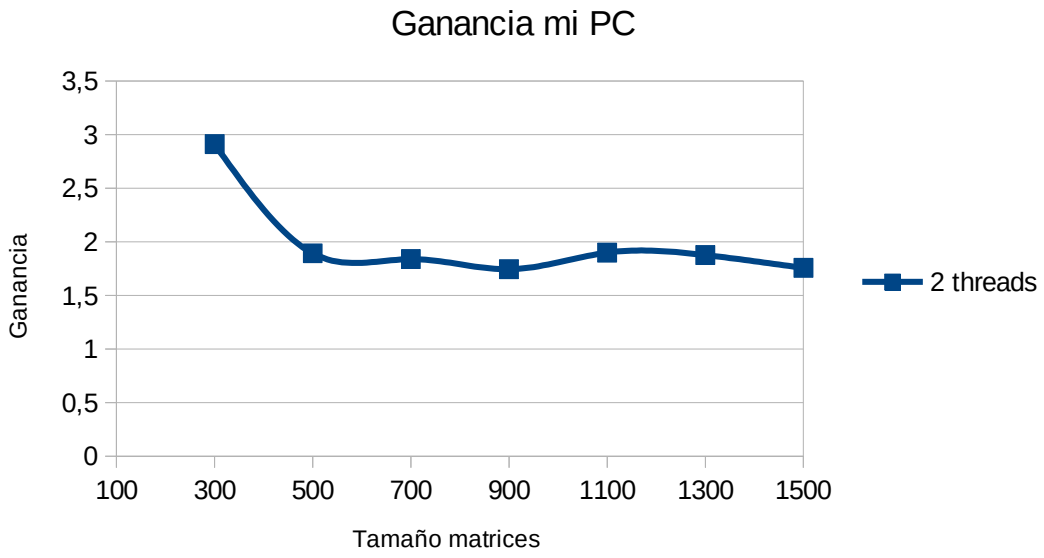
PC Local

	1 thread	2 threads
100	0,002003637	0,000688313
300	0,048423706	0,025583181
500	0,275202311	0,149687304
700	1,093235075	0,62671017
900	2,278071486	1,199253763
1100	4,560849023	2,430309419
1300	7,174370466	4,083503954
1500	13,082416398	7,229414201

Ganancia

2 threads
2,91093877349403
1,89279456686798
1,83851471464808
1,7444029590265
1,89957418211578
1,87665364226612
1,75691527345586




Ganancia mi PC
3,5
3
2,5
2
1,5
1
0,5
0
Ganancia
100
300
500
700
900
1100
1300
1500
Tamaño matrices
2 threads