2º curso / 2º cuatr.

Grado Ing. Inform.

Arquitectura de Computadores (AC)

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

Estudiante (nombre y apellidos): Jose Luis Pedraza Román
Grupo de prácticas: A2
Fecha de entrega: 6/5/2020
Fecha evaluación en clase: 7/5/2020

Antes de comenzar a realizar el trabajo de este cuaderno consultar el fichero con los normas de prácticas que se encuentra en SWAD

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`

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
int main(int argc, char **argv)
{
   int i, n=20, tid, x;
   int a[n],suma=0,sumalocal;
   if(argc < 3) {
      fprintf(stderr,"[ERROR]-Falta iteraciones y/o num_threads\n");
      exit(-1);
   }
   n = atoi(argv[1]); if (n>20) n=20;
   x = atoi(argv[2]);
   for (i=0; i<n; i++) {
      a[i] = i;
   }

   #pragma omp parallel if(n>4) default(none) \
         private(sumalocal,tid) shared(a,suma,n) num_threads(x)
   {
      sumalocal=0;
      tid=omp_get_thread_num();
      #pragma omp for private(i) schedule(static) nowait
         for (i=0; i<n; i++){
            sumalocal += a[i];
            printf(" thread %d suma de a[%d]=%d sumalocal=%d \n",
tid,i,a[i],sumalocal);
         }
      #pragma omp atomic
         suma += sumalocal;
```

```
      #pragma omp barrier
      #pragma omp master
         printf("thread master=%d imprime suma=%d\n",tid,suma);
   }
}
```

CAPTURAS DE PANTALLA:

```
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP3/ejer1] 2020-05-04 lunes
$gcc -O2 -fopenmp -o if-clauseModificado if-clauseModificado.c
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP3/ejer1] 2020-05-04 lunes
$./if-clauseModificado 4 3
 thread 0 suma de a[0]=0 sumalocal=0
 thread 0 suma de a[1]=1 sumalocal=1
 thread 0 suma de a[2]=2 sumalocal=3
 thread 0 suma de a[3]=3 sumalocal=6
thread master=0 imprime suma=6
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP3/ejer1] 2020-05-04 lunes
$./if-clauseModificado 4 4
 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
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP3/ejer1] 2020-05-04 lunes
$./if-clauseModificado 10 8
 thread 0 suma de a[0]=0 sumalocal=0
 thread 0 suma de a[1]=1 sumalocal=1
 thread 1 suma de a[2]=2 sumalocal=2
 thread 1 suma de a[3]=3 sumalocal=5
 thread 2 suma de a[4]=4 sumalocal=4
 thread 3 suma de a[5]=5 sumalocal=5
 thread 4 suma de a[6]=6 sumalocal=6
 thread 5 suma de a[7]=7 sumalocal=7
 thread 7 suma de a[9]=9 sumalocal=9
 thread 6 suma de a[8]=8 sumalocal=8
thread master=0 imprime suma=45
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP3/ejer1] 2020-05-04 lunes
$./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
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP3/ejer1] 2020-05-04 lunes
```

RESPUESTA:

Utilizando la cláusula *num_threads(x*) se fija el número de hebras que ejecutarán la región *parallel*.

En los dos primeros casos del ejemplo de ejecución siempre la ejecuta la hebra 0 debido al tamaño del problema y al número de hebras asignado, en las dos ejecuciones siguientes ya podemos distinguir esta utilidad, dado que se ejecuta con el número de hebras que fijamos con la cláusula *num_threads(x).*

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

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

RESPUESTA:

Con *static* → Las iteraciones se dividen en unidades de *chunk* iteraciones y se asignará con round-robin.
Con *dynamic* → La unidad de distribución tiene *chunk* iteraciones, pero si una hebra acaba su trabajo antes que otra (es más rápida), se le asignará antes una nueva unidad de *chunk* iteraciones.
Con *guided* → El *chunk* indicará el valor mínimo del tamaño del bloque.

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`

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

#ifdef _OPENMP
   #include <omp.h>
#else
   #define omp_get_thread_num() 0
#endif

int main(int argc, char **argv) {
   int i, n=200,chunk,a[n],suma=0;
   int *chunk_size;
   omp_sched_t *kind;
   if(argc < 3) {
      fprintf(stderr,"\nFalta iteraciones o chunk \n");
      exit(-1);
   }
   n = atoi(argv[1]); if (n>200) n=200; chunk = atoi(argv[2]);
   for (i=0; i<n; i++) a[i] = i;

   #pragma omp parallel
             {
                           #pragma omp single
        {
                                        printf("dyn-var: %d \n",omp_get_dynamic());
                                        printf("nthreads-var: %d \
n",omp_get_max_threads());
                                        printf("thread-limit-var: %d \
n",omp_get_thread_limit());

                                        omp_get_schedule(&kind,&chunk_size);
                                        printf("run-sched-var: (Kind: %d, Modifier: %d) \
n",kind,chunk_size);
                           }

             #pragma omp for firstprivate(suma) \
            lastprivate(suma) schedule(dynamic,chunk)
      for (i=0; i<n; i++)
      {
         suma = suma + a[i];
         printf(" thread %d suma a[%d]=%d suma=%d \n",
         omp_get_thread_num(),i,a[i],suma);
      }
   }
   printf("Fuera de 'parallel for':\n");
   printf("suma=%d\n",suma);
   printf("dyn-var: %d \n",omp_get_dynamic());
   printf("nthreads-var: %d \n",omp_get_max_threads());
   printf("thread-limit-var: %d \n",omp_get_thread_limit());
   omp_get_schedule(&kind,&chunk_size);
   printf("run-sched-var (Kind: %d. Modifier: %d) \n",kind,chunk_size);

   printf("Fuera de 'parallel for' suma=%d\n",suma);
}
```

CAPTURAS DE PANTALLA:

```
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP3/ejer3] 2020-05-04 lunes
$./scheduled-clauseModificado 10 4
dyn-var: 0
nthreads-var: 8
thread-limit-var: 2147483647
run-sched-var: (Kind: 2, Modifier: 1)
 thread 1 suma a[8]=8 suma=8
 thread 1 suma a[9]=9 suma=17
 thread 4 suma a[0]=0 suma=0
 thread 4 suma a[1]=1 suma=1
 thread 4 suma a[2]=2 suma=3
 thread 4 suma a[3]=3 suma=6
 thread 0 suma a[4]=4 suma=4
 thread 0 suma a[5]=5 suma=9
 thread 0 suma a[6]=6 suma=15
 thread 0 suma a[7]=7 suma=22
Fuera de 'parallel for':
suma=17
dyn-var: 0
nthreads-var: 8
thread-limit-var: 2147483647
run-sched-var (Kind: 2. Modifier: 1)
Fuera de 'parallel for' suma=17
```

```
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP3/ejer3] 2020-05-04 lunes
$export OMP_DYNAMIC=TRUE
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP3/ejer3] 2020-05-04 lunes
$./scheduled-clauseModificado 10 4
dyn-var: 1
nthreads-var: 8
thread-limit-var: 2147483647
run-sched-var: (Kind: 2, Modifier: 1)
 thread 4 suma a[4]=4 suma=4
 thread 4 suma a[5]=5 suma=9
 thread 4 suma a[6]=6 suma=15
 thread 4 suma a[7]=7 suma=22
 thread 2 suma a[8]=8 suma=8
 thread 2 suma a[9]=9 suma=17
 thread 3 suma a[0]=0 suma=0
 thread 3 suma a[1]=1 suma=1
 thread 3 suma a[2]=2 suma=3
 thread 3 suma a[3]=3 suma=6
Fuera de 'parallel for':
suma=17
dyn-var: 1
nthreads-var: 8
thread-limit-var: 2147483647
run-sched-var (Kind: 2. Modifier: 1)
Fuera de 'parallel for' suma=17
```

```
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP3/ejer3] 2020-05-04 lunes
$export OMP_NUM_THREADS=4
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP3/ejer3] 2020-05-04 lunes
$./scheduled-clauseModificado 10 4
dyn-var: 1
nthreads-var: 4
thread-limit-var: 2147483647
run-sched-var: (Kind: 2, Modifier: 1)
 thread 1 suma a[4]=4 suma=4
 thread 1 suma a[5]=5 suma=9
 thread 2 suma a[0]=0 suma=0
 thread 2 suma a[1]=1 suma=1
 thread 2 suma a[2]=2 suma=3
 thread 2 suma a[3]=3 suma=6
 thread 3 suma a[8]=8 suma=8
 thread 3 suma a[9]=9 suma=17
 thread 1 suma a[6]=6 suma=15
 thread 1 suma a[7]=7 suma=22
Fuera de 'parallel for':
suma=17
dyn-var: 1
nthreads-var: 4
thread-limit-var: 2147483647
run-sched-var (Kind: 2. Modifier: 1)
Fuera de 'parallel for' suma=17
```

```
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP3/ejer3] 2020-05-04 lunes
$export OMP_SCHEDULE="static,4"
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP3/ejer3] 2020-05-04 lunes
$./scheduled-clauseModificado 10 4
dyn-var: 1
nthreads-var: 4
thread-limit-var: 2147483647
run-sched-var: (Kind: 1, Modifier: 4)
 thread 3 suma a[4]=4 suma=4
 thread 3 suma a[5]=5 suma=9
 thread 3 suma a[6]=6 suma=15
 thread 3 suma a[7]=7 suma=22
 thread 0 suma a[8]=8 suma=8
 thread 0 suma a[9]=9 suma=17
 thread 1 suma a[0]=0 suma=0
 thread 1 suma a[1]=1 suma=1
 thread 1 suma a[2]=2 suma=3
 thread 1 suma a[3]=3 suma=6
Fuera de 'parallel for':
suma=17
dyn-var: 1
nthreads-var: 4
thread-limit-var: 2147483647
run-sched-var (Kind: 1. Modifier: 4)
Fuera de 'parallel for' suma=17
```

RESPUESTA:

Se imprimen los mismos valores dentro y fuera de la región paralela.

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`

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

#ifdef _OPENMP
   #include <omp.h>
#else
   #define omp_get_thread_num() 0
#endif

int main(int argc, char **argv) {
   int i, n=200,chunk,a[n],suma=0;
   if(argc < 3) {
      fprintf(stderr,"\nFalta iteraciones o chunk \n");
      exit(-1);
   }
   n = atoi(argv[1]); if (n>200) n=200; chunk = atoi(argv[2]);
   for (i=0; i<n; i++) a[i] = i;

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

              #pragma omp for firstprivate(suma) \
            lastprivate(suma) schedule(dynamic,chunk)
      for (i=0; i<n; i++)
      {
         suma = suma + a[i];
         printf(" thread %d suma a[%d]=%d suma=%d \n",
         omp_get_thread_num(),i,a[i],suma);
      }
   }

   printf("Fuera de Parallel: omp_get_num_threads: %d \n",omp_get_num_threads());
   printf("Fuera de Parallel: omp_get_num_procs: %d \n",omp_get_num_procs());
   printf("Fuera de Parallel: omp_in_parallel: %d \n",omp_in_parallel());

   printf("Fuera de 'parallel for' suma=%d\n",suma);
}
```

CAPTURAS DE PANTALLA:

```
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP3/ejer4] 2020-05-04 lunes
$export OMP_NUM_THREADS=4
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP3/ejer4] 2020-05-04 lunes
$./scheduled-clauseModificado4 10 4
Dentro de Parallel: omp_get_num_threads: 4
Dentro de Parallel: omp_get_num_procs: 8
Dentro de Parallel: omp_in_parallel: 1
 thread 0 suma a[4]=4 suma=4
 thread 0 suma a[5]=5 suma=9
 thread 0 suma a[6]=6 suma=15
 thread 2 suma a[8]=8 suma=8
 thread 2 suma a[9]=9 suma=17
 thread 3 suma a[0]=0 suma=0
 thread 3 suma a[1]=1 suma=1
 thread 3 suma a[2]=2 suma=3
 thread 3 suma a[3]=3 suma=6
 thread 0 suma a[7]=7 suma=22
Fuera de Parallel: omp_get_num_threads: 1
Fuera de Parallel: omp_get_num_procs: 8
Fuera de Parallel: omp_in_parallel: 0
Fuera de 'parallel for' suma=17
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP3/ejer4] 2020-05-04 lunes
$export OMP_NUM_THREADS=8
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP3/ejer4] 2020-05-04 lunes
$./scheduled-clauseModificado4 10 4
Dentro de Parallel: omp_get_num_threads: 8
Dentro de Parallel: omp_get_num_procs: 8
Dentro de Parallel: omp_in_parallel: 1
 thread 0 suma a[8]=8 suma=8
 thread 0 suma a[9]=9 suma=17
 thread 7 suma a[4]=4 suma=4
 thread 7 suma a[5]=5 suma=9
 thread 7 suma a[6]=6 suma=15
 thread 7 suma a[7]=7 suma=22
 thread 4 suma a[0]=0 suma=0
 thread 4 suma a[1]=1 suma=1
 thread 4 suma a[2]=2 suma=3
 thread 4 suma a[3]=3 suma=6
Fuera de Parallel: omp_get_num_threads: 1
Fuera de Parallel: omp_get_num_procs: 8
Fuera de Parallel: omp_in_parallel: 0
Fuera de 'parallel for' suma=17
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP3/ejer4] 2020-05-04 lunes
```

RESPUESTA:

Se producen resultados diferentes en *omp_get_num_threads* (ya que fuera siempre devolverá 1 al ser secuencial) y en *omp_in_parallel* (dado que solo devolverá 1 cuando se la llama dentro) dentro y fuera de la región paralela mientras que con *omp_get_num_procs* matiene su valor como es lógico debido a que el numero de procesadores disponibles no cambia.

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`

```
#include <stdio.h>
#include <stdlib.h>
#ifdef _OPENMP
#include <omp.h>
#else
#define omp_get_thread_num() 0
#endif
int main(int argc, char **argv) {
   int i, n=200,chunk,a[n],suma=0;
   int * modifier;
   omp_sched_t * kind;
   if(argc < 3)
   {
      fprintf(stderr,"\nFalta iteraciones o chunk \n");
      exit(-1);
   }
   n = atoi(argv[1]); if (n>200) n=200; chunk = atoi(argv[2]);
   for (i=0; i<n; i++)   a[i] = i;
   #pragma omp parallel
   {
      #pragma omp single
      {
         printf("DENTRO DEL PARALLEL\n\n");
         printf("dyn-var: %d \n",omp_get_dynamic());
         omp_set_dynamic(1);
         printf("Modificamos dyn-var con omp_set_dynamic(1) y el resultado es: %d\n",
omp_get_dynamic());
         printf("nthreads-var: %d \n",omp_get_max_threads());
         omp_set_num_threads(8);
         printf("Modificamos nthreads-var con omp_set_num_threads(8); y el resultado es:
%d\n",omp_get_max_threads());

         omp_get_schedule(&kind,&modifier);
         printf("run-sched-var: (Kind: %d, Modifier: %d)\n",kind,modifier);
         omp_set_schedule(2,2);
         omp_get_schedule(&kind,&modifier);
         printf("Modificamos run-sched-var con omp_set_schedule(2,2) y el resultado de
Kind es %d y el de Modifier es: %d \n",kind,modifier);
      }
      #pragma omp for firstprivate(suma) lastprivate(suma) schedule(dynamic,chunk)
         for (i=0; i<n; i++) {
            suma = suma + a[i];
            printf("thread %d suma a[%d]=%d suma=%d \n",
            omp_get_thread_num(),i,a[i],suma);
         }
   }
   printf("\n\nFUERA DEL PARALLEL\n");
   printf("suma=%d\n",suma);
   printf("dyn-var: %d \n",omp_get_dynamic());
   printf("nthreads-var: %d \n",omp_get_max_threads());
   printf("omp_get_num_threads: %d \n",omp_get_num_threads());
   omp_get_schedule(&kind,&modifier);
   printf("run-sched-var: (Kind: %d. Modifier: %d)\n",kind,modifier);
   return(0);
}
```

CAPTURAS DE PANTALLA:

```
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP3/ejer5] 2020-05-05 martes
$./scheduled-clauseModificado5 10 4
DENTRO DEL PARALLEL

dyn-var: 0
Modificamos dyn-var con omp_set_dynamic(1) y el resultado es: 1
nthreads-var: 4
Modificamos nthreads-var con omp_set_num_threads(8); y el resultado es: 8
run-sched-var: (Kind: 2, Modifier: 1)
Modificamos run-sched-var con omp_set_schedule(2,2) y el resultado de Kind es 2 y el de Modifier es: 2
thread 3 suma a[0]=0 suma=0
thread 3 suma a[1]=1 suma=1
thread 3 suma a[2]=2 suma=3
thread 3 suma a[3]=3 suma=6
thread 0 suma a[4]=4 suma=4
thread 0 suma a[5]=5 suma=9
thread 0 suma a[6]=6 suma=15
thread 0 suma a[7]=7 suma=22
thread 1 suma a[8]=8 suma=8
thread 1 suma a[9]=9 suma=17


FUERA DEL PARALLEL
suma=17
dyn-var: 0
nthreads-var: 4
omp_get_num_threads: 1
run-sched-var: (Kind: 2. Modifier: 1)
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP3/ejer5] 2020-05-05 martes
$
```

RESPUESTA:

Como vemos cambian correctamente los valores.

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`

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#ifdef _OPENMP
  #include <omp.h>
#else
  #define omp_get_thread_num() 0
#endif

//#define TIMES
#define PRINTF_ALL

int main(int argc, char **argv) {
  //Lectura valores de entrada
  if(argc < 2) {
    fprintf(stderr,"Falta num\n");
    exit(-1);
  }
```

```
  int n = atoi(argv[1]);

  int i,j;
  struct timespec ini,fin; double transcurrido;

  //Creación e inicialización de vector y matriz
  // Creación
  int *v1,*v2;
  v1 = (int*) malloc(n*sizeof(double));
  v2 = (int*) malloc(n*sizeof(double));

  int **M;
  M = (int**) malloc(n*sizeof(int*));
  for(i=0;i<n;i++)
    M[i] = (int*)malloc(n*sizeof(int));

  // Inicialización
  for(i=0;i<n;i++)
    v1[i]=i+1;

  int num=1;
  for(i=0;i<n;i++){
    for(j=0;j<n;j++){
      if(j>i) M[i][j]=0;
      else { M[i][j]=num; num++; }
    }
  }

  //Impresión de vector y matriz iniciales
  #ifndef TIMES
    #ifdef PRINTF_ALL
      printf("Vector inicial:\n");
      for (i=0; i<n; i++) printf("%d ",v1[i]);
      printf("\n");

      printf("Matriz inicial:\n");
      for (i=0; i<n; i++) {
        for (j=0; j<n; j++) {
          if(M[i][j]<10) printf(" %d ",M[i][j]);
          else printf("%d ",M[i][j]);
        }
        printf("\n");
      }
    #endif
  #endif

  //Cálculo resultado y toma de tiempos
  clock_gettime(CLOCK_REALTIME,&ini);
  for (i=0; i<n; i++) {
    v2[i]=0;
    for (j=0; j<=i; j++) {
      v2[i]+=M[i][j]*v1[j];
    }
  }
  clock_gettime(CLOCK_REALTIME,&fin);
  transcurrido=(double) (fin.tv_sec-ini.tv_sec)+(double)
((fin.tv_nsec-ini.tv_nsec)/(1.e+9));

  //Impresión del tiempo y matriz resultado
  #ifdef TIMES
    printf("%d %11.9f\n",n,transcurrido);
  #else
```

```
    #ifdef PRINTF_ALL
      printf("Tiempo: %11.9f\n",transcurrido);
      printf("Vector resultado (M x v1):\n");
      for (i=0; i<n; i++) printf("%d ",v2[i]);
      printf("\n");
    #else
      printf("Tiempo: %11.9f\n",transcurrido);
      printf("v2[0]: %d, v2[n-1]: %d\n",v2[0],v2[n-1]);
    #endif
  #endif

  //Vaciar memoria
  free(M);
  free(v1);
  free(v2);
return(0);
}
```

CAPTURAS DE PANTALLA:

```
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP3/ejer6] 2020-05-04 lunes
$gcc -O2 -o pmtv-secuencial pmtv-secuencial.c
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP3/ejer6] 2020-05-04 lunes
$./pmtv-secuencial 10
Vector inicial:
1 2 3 4 5 6 7 8 9 10
Matriz inicial:
 1  0  0  0  0  0  0  0  0  0
 2  3  0  0  0  0  0  0  0  0
 4  5  6  0  0  0  0  0  0  0
 7  8  9 10  0  0  0  0  0  0
11 12 13 14 15  0  0  0  0  0
16 17 18 19 20 21  0  0  0  0
22 23 24 25 26 27 28  0  0  0
29 30 31 32 33 34 35 36  0  0
37 38 39 40 41 42 43 44 45  0
46 47 48 49 50 51 52 53 54 55
Tiempo: 0.000000761
Vector resultado (M x v1):
1 8 32 90 205 406 728 1212 1905 2860
```

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) Información sacada del manual: https://computing.llnl.gov/tutorials/openMP/

- **SCHEDULE**: Describes how iterations of the loop are divided among the threads in the team. The default schedule is implementation dependent. For a discussion on how one type of scheduling may be more optimal than others, see http://openmp.org/forum/viewtopic.php?f=3&t=83.

STATIC

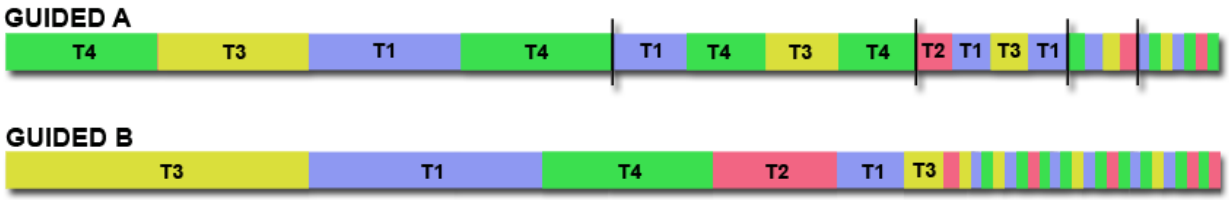
Loop iterations are divided into pieces of size *chunk* and then statically assigned to threads. If chunk is not specified, the iterations are evenly (if possible) divided contiguously among the threads.

DYNAMIC

Loop iterations are divided into pieces of size *chunk*, and dynamically scheduled among the threads; when a thread finishes one chunk, it is dynamically assigned another. The default chunk size is 1.

GUIDED

Iterations are dynamically assigned to threads in blocks as threads request them until no blocks remain to be assigned. Similar to DYNAMIC except that the block size decreases each time a parcel of work is given to a thread.
The size of the initial block is proportional to: **number_of_iterations / number_of_threads**
Subsequent blocks are proportional to **number_of_iterations_remaining / number_of_threads**
The chunk parameter defines the minimum block size. The default chunk size is 1.
Note: compilers differ in how GUIDED is implemented as shown in the "Guided A" and "Guided B" examples below.



Por tanto:

static → No está definido
dynamic → 1
guided → 1

b) num_op = chunk*num_fila

c) En *dynamic* se ejecutarán 64 operaciones ó 1 operación; mientras que con *guided* el último valor no será igual.

CAPTURA CÓDIGO FUENTE: `pmtv-OpenMP.c`

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

#ifdef _OPENMP
  #include <omp.h>
#else
  #define omp_get_thread_num() 0
#endif

//#define TIMES
//#define PRINTF_ALL

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

```
  //Lectura valores de entrada
  if(argc < 2) {
    fprintf(stderr,"Falta num\n");
    exit(-1);
  }
  int n = atoi(argv[1]);

  int i,j;
  struct timespec ini,fin; double transcurrido;

  //Creación e inicialización de vector y matriz
  // Creación
  int *v1,*v2;
  v1 = (int*) malloc(n*sizeof(double));
  v2 = (int*) malloc(n*sizeof(double));

  int **M;
  M = (int**) malloc(n*sizeof(int*));
  for(i=0;i<n;i++)
    M[i] = (int*)malloc(n*sizeof(int));

  // Inicialización
  for(i=0;i<n;i++)
    v1[i]=i+1;

  int num=1;
  for(i=0;i<n;i++){
    for(j=0;j<n;j++){
      if(j>i) M[i][j]=0;
      else { M[i][j]=num; num++; }
    }
  }

  //Impresión de vector y matriz iniciales
  #ifndef TIMES
    #ifdef PRINTF_ALL
      printf("Vector inicial:\n");
      for (i=0; i<n; i++) printf("%d ",v1[i]);
      printf("\n");

      printf("Matriz inicial:\n");
      for (i=0; i<n; i++) {
        for (j=0; j<n; j++) {
          if(M[i][j]<10) printf(" %d ",M[i][j]);
          else printf("%d ",M[i][j]);
        }
        printf("\n");
      }
    #endif
  #endif

  //Cálculo resultado y toma de tiempos
  clock_gettime(CLOCK_REALTIME,&ini);

  #pragma omp parallel for default(none) private(i,j) shared(n,v1,v2,M) schedule(runtime)
  for (i=0; i<n; i++) {
    v2[i]=0;
    for (j=0; j<=i; j++) {
      v2[i]+=M[i][j]*v1[j];
    }
  }
```

```
  clock_gettime(CLOCK_REALTIME,&fin);
  transcurrido=(double) (fin.tv_sec-ini.tv_sec)+(double)
((fin.tv_nsec-ini.tv_nsec)/(1.e+9));

  //Impresión del tiempo y matriz resultado
  #ifdef TIMES
    printf("%d %11.9f\n",n,transcurrido);
  #else
    #ifdef PRINTF_ALL
      printf("Tiempo: %11.9f\n",transcurrido);
      printf("Vector resultado (M x v1):\n");
      for (i=0; i<n; i++) printf("%d ",v2[i]);
      printf("\n");
    #else
      printf("Tiempo: %11.9f\n",transcurrido);
      printf("v2[0]: %d, v2[n-1]: %d\n",v2[0],v2[n-1]);
    #endif
  #endif

  //Vaciar memoria
  free(M);
  free(v1);
  free(v2);
return(0);
}
```

DESCOMPOSICIÓN DE DOMINIO:
Repartimos las filas de la matriz y cada hebra calculará un valor del vector final, recorriendo las diferentes filas de la matriz para realizar las operaciones.

CAPTURAS DE PANTALLA:

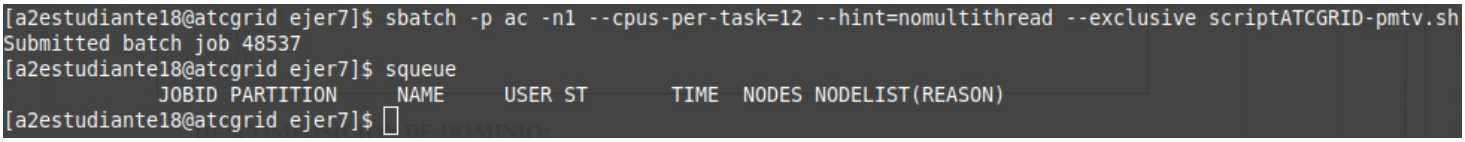


TABLA RESULTADOS, SCRIPT Y GRÁFICA atcgrid

SCRIPT: pmvt-OpenMP_atcgrid.sh

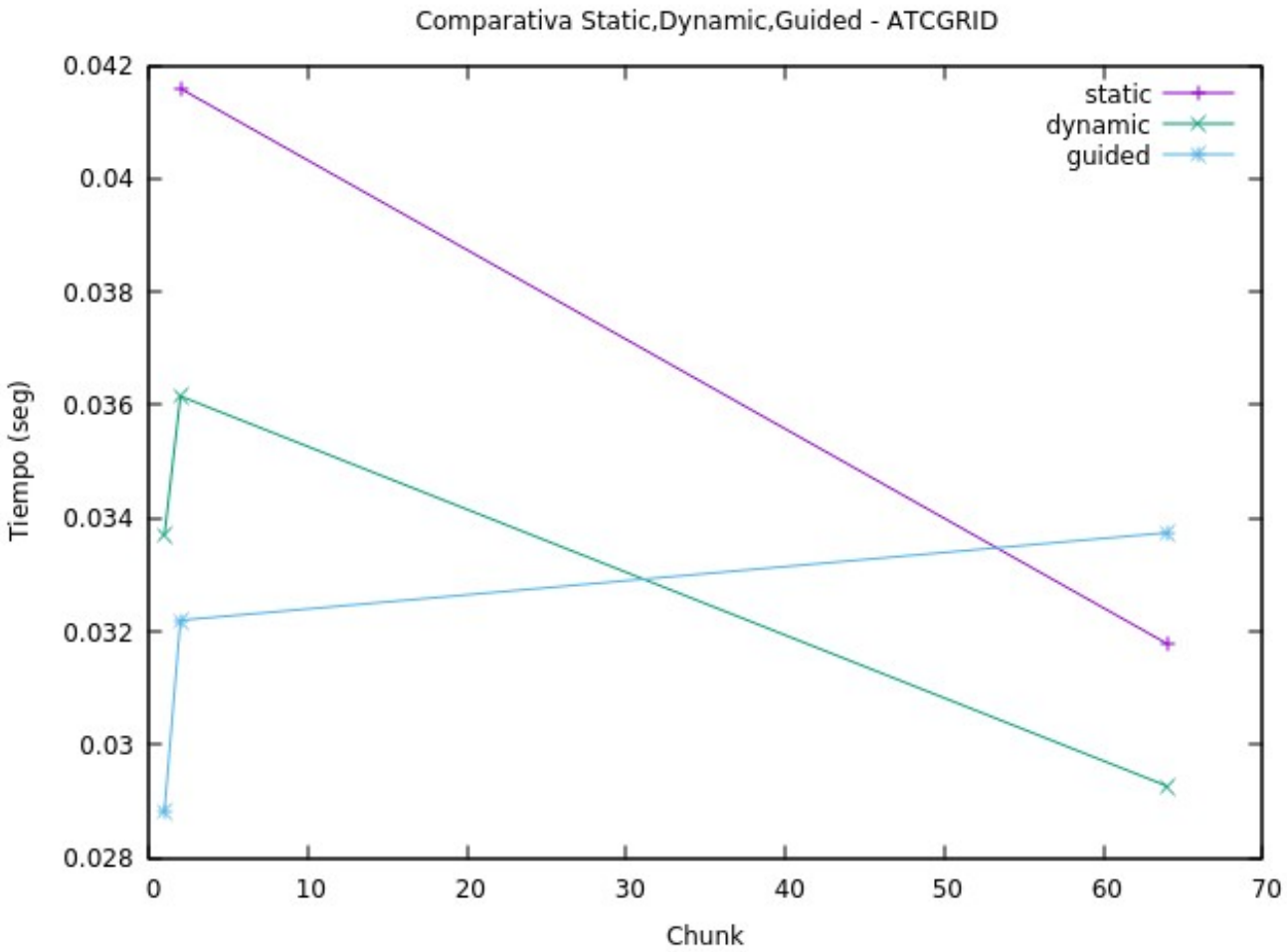
```
#!/bin/bash

#sbatch -p ac -n1 --cpus-per-task=12 --hint=nomultithread --exclusive scriptATCGRID-
pmtv.sh

#Órdenes para el sistema de colas:
#1. Asigna al trabajo un nombre
#SBATCH --job-name=pmv-OpenMP-a
#2. Asignar el trabajo a una cola (partición)
#SBATCH --partition=ac
#2. Asignar el trabajo a un account
#SBATCH --account=ac
#Obtener información de las variables del entorno del sistema de colas:

echo "Id. usuario del trabajo: $SLURM_JOB_USER"
```

```
23. echo "Id. del trabajo: $SLURM_JOBID"
24. echo "Nombre del trabajo especificado por usuario: $SLURM_JOB_NAME"
25. echo "Directorio de trabajo (en el que se ejecuta el script):
26. $SLURM_SUBMIT_DIR"
27. echo "Cola: $SLURM_JOB_PARTITION"
28. echo "Nodo que ejecuta este trabajo:$SLURM_SUBMIT_HOST"
29. echo "No de nodos asignados al trabajo: $SLURM_JOB_NUM_NODES"
30. echo "Nodos asignados al trabajo: $SLURM_JOB_NODELIST"
31. echo "CPUs por nodo: $SLURM_JOB_CPUS_PER_NODE"
32.
33.
34. export OMP_NUM_THREADS=12
35. N=15360
36.
37. export OMP_SCHEDULE="STATIC"
38. echo "STATIC,default";
39. srun -p ac ./pmtv-OpenMP $N
40. srun -p ac ./pmtv-OpenMP $N
41.
42. export OMP_SCHEDULE="STATIC,1"
43. echo "STATIC,1";
44. srun -p ac ./pmtv-OpenMP $N
45. srun -p ac ./pmtv-OpenMP $N
46.
47. export OMP_SCHEDULE="STATIC,64"
48. echo "STATIC,64";
49. srun -p ac ./pmtv-OpenMP $N
50. srun -p ac ./pmtv-OpenMP $N
51.
52. echo "--------------------------";
53.
54. export OMP_SCHEDULE="DYNAMIC"
55. echo "DYNAMIC,default";
56. srun -p ac ./pmtv-OpenMP $N
57. srun -p ac ./pmtv-OpenMP $N
58.
59. export OMP_SCHEDULE="DYNAMIC,1"
60. echo "DYNAMIC,1";
61. srun -p ac ./pmtv-OpenMP $N
62. srun -p ac ./pmtv-OpenMP $N
63.
64. export OMP_SCHEDULE="DYNAMIC,64"
65. echo "DYNAMIC,64";
66. srun -p ac ./pmtv-OpenMP $N
67. srun -p ac ./pmtv-OpenMP $N
68.
69. echo "--------------------------";
70.
71. export OMP_SCHEDULE="GUIDED"
72. echo "GUIDED,default";
73. srun -p ac ./pmtv-OpenMP $N
74. srun -p ac ./pmtv-OpenMP $N
75.
76. export OMP_SCHEDULE="GUIDED,1"
77. echo "GUIDED,1";
78. srun -p ac ./pmtv-OpenMP $N
79. srun -p ac ./pmtv-OpenMP $N
80.
81. export OMP_SCHEDULE="GUIDED,64"
82. echo "GUIDED,64";
83. srun -p ac ./pmtv-OpenMP $N
84. srun -p ac ./pmtv-OpenMP $N
```

Tabla 3 . Tiempos de ejecución de la versión paralela del producto de una matriz triangular por un vector r para vectores de tamaño N= 15360 , 12 threads

Chunk	Static	Dynamic	Guided
por defecto	0.041590462	0.036157070	0.032198212
1	0.029249676	0.033697883	0.028828295
64	0.031779824	0.029268046	0.033740976
Chunk	**Static**	**Dynamic**	**Guided**
por defecto	0.042784082	0.033533905	0.029106553
1	0.033732811	0.032542282	0.031143165
64	0.035454419	0.029916512	0.030451957

Gráfica:



Tanto en *static* como en *dynamic*, a mayor *chunk* obtendremos mayor eficiencia, ya que conseguiremos reducir la carga de asignación de trabajo, pero debemos tener en cuenta, que si el *chunk* es muy grande, no se paraleliza del todo y por tanto será menos eficiente.
En el caso de *guided*, cuanto menor sea el *chunk*, normalmente obtendremos mejor eficiencia.

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`

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

#ifdef _OPENMP
  #include <omp.h>
#else
  #define omp_get_thread_num() 0
#endif

//#define TIMES
#define PRINTF_ALL

int main(int argc, char **argv) {
  //Lectura valores de entrada
  if(argc < 2) {
    fprintf(stderr,"Falta num\n");
    exit(-1);
  }
  int n = atoi(argv[1]);
  int i,j,k;
  struct timespec ini,fin; double transcurrido;

  //Creación e inicialización de vector y matriz
  //Creación
  int **A, **B, **C;
  A = (int**) malloc(n*sizeof(int*));
  for(i=0;i<n;i++)
    A[i] = (int*)malloc(n*sizeof(int));

  B = (int**) malloc(n*sizeof(int*));
  for(i=0;i<n;i++)
    B[i] = (int*)malloc(n*sizeof(int));

  C = (int**) malloc(n*sizeof(int*));
  for(i=0;i<n;i++)
    C[i] = (int*)malloc(n*sizeof(int));

  //Inicialización
  for(i=0;i<n;i++){
    for(j=0;j<n;j++){
      B[i][j]=n*i+j;
      C[i][j]=n*i+j;
    }
  }

  // 3. Impresión de matrices iniciales
  #ifndef TIMES
    #ifdef PRINTF_ALL
      printf("Matriz inicial B:\n");
```

```
      for (i=0; i<n; i++) {
        for (j=0; j<n; j++) {
          if(B[i][j]<10) printf(" %d ",B[i][j]);
          else printf("%d ",B[i][j]);
        }
        printf("\n");
      }
      printf("Matriz inicial C:\n");
      for (i=0; i<n; i++) {
        for (j=0; j<n; j++) {
          if(C[i][j]<10) printf(" %d ",C[i][j]);
          else printf("%d ",C[i][j]);
        }
        printf("\n");
      }
    #endif
  #endif

  //Cálculo resultado y toma de tiempos
  clock_gettime(CLOCK_REALTIME,&ini);

  for (i=0; i<n; i++) {
    for (j=0; j<n; j++) {
      A[i][j]=0;
      for (k=0; k<n; k++) {
        A[i][j]+=B[i][k]*C[k][j];
      }
    }
  }

  clock_gettime(CLOCK_REALTIME,&fin);
  transcurrido=(double) (fin.tv_sec-ini.tv_sec)+(double)
((fin.tv_nsec-ini.tv_nsec)/(1.e+9));

  //Impresión del tiempo y vector resultado
  #ifdef TIMES
    printf("%d %11.9f\n",n,transcurrido);
  #else
    #ifdef PRINTF_ALL
      printf("Tiempo: %11.9f\n",transcurrido);
      printf("Matriz resultado A=B*C:\n");
      for (i=0; i<n; i++) {
        for (j=0; j<n; j++) {
          if(A[i][j]<10) printf(" %d ",A[i][j]);
          else printf("%d ",A[i][j]);
        }
        printf("\n");
      }
    #else
      printf("Tiempo: %11.9f\n",transcurrido);
      printf("A[0][0]: %d, A[n-1][n-1]: %d\n",A[0][0],A[n-1][n-1]);
    #endif
  #endif

  //Vaciar memoria
  free(A);
  free(B);
  free(C);
return(0);
}
```

CAPTURAS DE PANTALLA:

```
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP3/ejer8] 2020-05-05 martes
$gcc -O2 -o pmm-secuencial pmm-secuencial.c -lrt
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP3/ejer8] 2020-05-05 martes
$./pmm-secuencial 10
Matriz inicial B:
 0  1  2  3  4  5  6  7  8  9
10 11 12 13 14 15 16 17 18 19
20 21 22 23 24 25 26 27 28 29
30 31 32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47 48 49
50 51 52 53 54 55 56 57 58 59
60 61 62 63 64 65 66 67 68 69
70 71 72 73 74 75 76 77 78 79
80 81 82 83 84 85 86 87 88 89
90 91 92 93 94 95 96 97 98 99
Matriz inicial C:
 0  1  2  3  4  5  6  7  8  9
10 11 12 13 14 15 16 17 18 19
20 21 22 23 24 25 26 27 28 29
30 31 32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47 48 49
50 51 52 53 54 55 56 57 58 59
60 61 62 63 64 65 66 67 68 69
70 71 72 73 74 75 76 77 78 79
80 81 82 83 84 85 86 87 88 89
90 91 92 93 94 95 96 97 98 99
Tiempo: 0.000002153
Matriz resultado A=B*C:
2850 2895 2940 2985 3030 3075 3120 3165 3210 3255
7350 7495 7640 7785 7930 8075 8220 8365 8510 8655
11850 12095 12340 12585 12830 13075 13320 13565 13810 14055
16350 16695 17040 17385 17730 18075 18420 18765 19110 19455
20850 21295 21740 22185 22630 23075 23520 23965 24410 24855
25350 25895 26440 26985 27530 28075 28620 29165 29710 30255
29850 30495 31140 31785 32430 33075 33720 34365 35010 35655
34350 35095 35840 36585 37330 38075 38820 39565 40310 41055
38850 39695 40540 41385 42230 43075 43920 44765 45610 46455
43350 44295 45240 46185 47130 48075 49020 49965 50910 51855
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP3/ejer8] 2020-05-05 martes
```

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:

Pensando en la multiplicación de 2 matrices 2x2:

A00,A01 B00,B01

X

A10,A11 B10,B11

Entonces tenemos que para 2 hebras:

Hebra1 = A00*B00+A01*B10, A00*B01+A01*B11

Hebra2 = A10*B00+A11*B10, A10*B01+A11*B11

CAPTURA CÓDIGO FUENTE: `pmm-OpenMP.c`

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

#ifdef _OPENMP
  #include <omp.h>
#else
  #define omp_get_thread_num() 0
#endif

//#define TIMES
#define PRINTF_ALL

int main(int argc, char **argv) {
  //Lectura valores de entrada
  if(argc < 2) {
    fprintf(stderr,"Falta num\n");
    exit(-1);
  }
  int n = atoi(argv[1]);
  int i,j,k;
  struct timespec ini,fin; double transcurrido;

  //Creación e inicialización de matrices
  //Creación
  int **A, **B, **C;
  A = (int**) malloc(n*sizeof(int*));
  for(i=0;i<n;i++)
    A[i] = (int*)malloc(n*sizeof(int));

  B = (int**) malloc(n*sizeof(int*));
  for(i=0;i<n;i++)
    B[i] = (int*)malloc(n*sizeof(int));

  C = (int**) malloc(n*sizeof(int*));
  for(i=0;i<n;i++)
    C[i] = (int*)malloc(n*sizeof(int));

  //Inicialización
  #pragma omp parallel for default(none) private(i,j) shared(n,B,C)
  for(i=0;i<n;i++){
    for(j=0;j<n;j++){
      B[i][j]=n*i+j;
```

```
      C[i][j]=n*i+j;
    }
  }

  // 3. Impresión de matrices iniciales
  #ifndef TIMES
    #ifdef PRINTF_ALL
      printf("Matriz inicial B:\n");
      for (i=0; i<n; i++) {
        for (j=0; j<n; j++) {
          if(B[i][j]<10) printf(" %d ",B[i][j]);
          else printf("%d ",B[i][j]);
        }
        printf("\n");
      }
      printf("Matriz inicial C:\n");
      for (i=0; i<n; i++) {
        for (j=0; j<n; j++) {
          if(C[i][j]<10) printf(" %d ",C[i][j]);
          else printf("%d ",C[i][j]);
        }
        printf("\n");
      }
    #endif
  #endif

  //Cálculo resultado y toma de tiempos
  clock_gettime(CLOCK_REALTIME,&ini);
  #pragma omp parallel for default(none) private(i,j,k) shared(n,A,B,C)
  for (i=0; i<n; i++) {
    for (j=0; j<n; j++) {
      A[i][j]=0;
      for (k=0; k<n; k++) {
        A[i][j]+=B[i][k]*C[k][j];
      }
    }
  }

  clock_gettime(CLOCK_REALTIME,&fin);
  transcurrido=(double) (fin.tv_sec-ini.tv_sec)+(double)
((fin.tv_nsec-ini.tv_nsec)/(1.e+9));

  //Impresión del tiempo y vector resultado
  #ifdef TIMES
    printf("%d %11.9f\n",n,transcurrido);
  #else
    #ifdef PRINTF_ALL
      printf("Tiempo: %11.9f\n",transcurrido);
      printf("Matriz resultado A=B*C:\n");
      for (i=0; i<n; i++) {
        for (j=0; j<n; j++) {
          if(A[i][j]<10) printf(" %d ",A[i][j]);
          else printf("%d ",A[i][j]);
        }
        printf("\n");
      }
    #else
      printf("Tiempo: %11.9f\n",transcurrido);
      printf("A[0][0]: %d, A[n-1][n-1]: %d\n",A[0][0],A[n-1][n-1]);
    #endif
  #endif
```

```
  //Vaciar memoria
  free(A);
  free(B);
  free(C);
return(0);
}
```

CAPTURAS DE PANTALLA:

```
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP3/ejer9] 2020-05-05 martes
$gcc -O2 -fopenmp -o pmm-OpenMP pmm-OpenMP.c -lrt
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP3/ejer9] 2020-05-05 martes
$./pmm-OpenMP 10
Matriz inicial B:
 0  1  2  3  4  5  6  7  8  9
10 11 12 13 14 15 16 17 18 19
20 21 22 23 24 25 26 27 28 29
30 31 32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47 48 49
50 51 52 53 54 55 56 57 58 59
60 61 62 63 64 65 66 67 68 69
70 71 72 73 74 75 76 77 78 79
80 81 82 83 84 85 86 87 88 89
90 91 92 93 94 95 96 97 98 99
Matriz inicial C:
 0  1  2  3  4  5  6  7  8  9
10 11 12 13 14 15 16 17 18 19
20 21 22 23 24 25 26 27 28 29
30 31 32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47 48 49
50 51 52 53 54 55 56 57 58 59
60 61 62 63 64 65 66 67 68 69
70 71 72 73 74 75 76 77 78 79
80 81 82 83 84 85 86 87 88 89
90 91 92 93 94 95 96 97 98 99
Tiempo: 0.000010060
Matriz resultado A=B*C:
2850 2895 2940 2985 3030 3075 3120 3165 3210 3255
7350 7495 7640 7785 7930 8075 8220 8365 8510 8655
11850 12095 12340 12585 12830 13075 13320 13565 13810 14055
16350 16695 17040 17385 17730 18075 18420 18765 19110 19455
20850 21295 21740 22185 22630 23075 23520 23965 24410 24855
25350 25895 26440 26985 27530 28075 28620 29165 29710 30255
29850 30495 31140 31785 32430 33075 33720 34365 35010 35655
34350 35095 35840 36585 37330 38075 38820 39565 40310 41055
38850 39695 40540 41385 42230 43075 43920 44765 45610 46455
43350 44295 45240 46185 47130 48075 49020 49965 50910 51855
```

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

#sbatch -p ac -n1 --cpus-per-task=12 --hint=nomultithread --exclusive scriptATCGRID-pmtv.sh

#Órdenes para el sistema de colas:
#1. Asigna al trabajo un nombre
#SBATCH --job-name=pmv-OpenMP-a
#2. Asignar el trabajo a una cola (partición)
#SBATCH --partition=ac
#2. Asignar el trabajo a un account
#SBATCH --account=ac
#Obtener información de las variables del entorno del sistema de colas:

echo "Id. usuario del trabajo: $SLURM_JOB_USER"
echo "Id. del trabajo: $SLURM_JOBID"
echo "Nombre del trabajo especificado por usuario: $SLURM_JOB_NAME"
            echo "Directorio de trabajo (en el que se ejecuta el script):
            $SLURM_SUBMIT_DIR"
            echo "Cola: $SLURM_JOB_PARTITION"
            echo "Nodo que ejecuta este trabajo:$SLURM_SUBMIT_HOST"
            echo "No de nodos asignados al trabajo: $SLURM_JOB_NUM_NODES"
            echo "Nodos asignados al trabajo: $SLURM_JOB_NODELIST"
            echo "CPUs por nodo: $SLURM_JOB_CPUS_PER_NODE"

            export OMP_NUM_THREADS=1
            echo "1 PROCESADOR";
            N=100
            echo "TAMA = 100";
            srun -p ac ./pmm-OpenMP $N

            N=500
            echo "TAMA = 500";
            srun -p ac ./pmm-OpenMP $N

            N=1000
            echo "TAMA = 1000";
            srun -p ac ./pmm-OpenMP $N

            echo "-------------------------"

            export OMP_NUM_THREADS=2
            echo "2 PROCESADOR";
            N=100
            echo "TAMA = 100";
            srun -p ac ./pmm-OpenMP $N

            N=500
            echo "TAMA = 500";
```

```
132.            srun -p ac ./pmm-OpenMP $N
133.
134.            N=1000
135.            echo "TAMA = 1000";
136.            srun -p ac ./pmm-OpenMP $N
137.
138.            echo "-------------------------"
139.
140.            export OMP_NUM_THREADS=3
141.            echo "3 PROCESADOR";
142.            N=100
143.            echo "TAMA = 100";
144.            srun -p ac ./pmm-OpenMP $N
145.
146.            N=500
147.            echo "TAMA = 500";
148.            srun -p ac ./pmm-OpenMP $N
149.
150.            N=1000
151.            echo "TAMA = 1000";
152.            srun -p ac ./pmm-OpenMP $N
153.
154.            echo "-------------------------"
155.
156.            export OMP_NUM_THREADS=4
157.            echo "4 PROCESADOR";
158.            N=100
159.            echo "TAMA = 100";
160.            srun -p ac ./pmm-OpenMP $N
161.
162.            N=500
163.            echo "TAMA = 500";
164.            srun -p ac ./pmm-OpenMP $N
165.
166.            N=1000
167.            echo "TAMA = 1000";
168.            srun -p ac ./pmm-OpenMP $N
169.
170.            echo "-------------------------"
171.
172.            export OMP_NUM_THREADS=5
173.            echo "5 PROCESADOR";
174.            N=100
175.            echo "TAMA = 100";
176.            srun -p ac ./pmm-OpenMP $N
177.
178.            N=500
179.            echo "TAMA = 500";
180.            srun -p ac ./pmm-OpenMP $N
181.
182.            N=1000
183.            echo "TAMA = 1000";
184.            srun -p ac ./pmm-OpenMP $N
185.
186.            echo "-------------------------"
187.
188.            export OMP_NUM_THREADS=6
189.            echo "6 PROCESADOR";
190.            N=100
191.            echo "TAMA = 100";
192.            srun -p ac ./pmm-OpenMP $N
193.
```

```
194.            N=500
195.            echo "TAMA = 500";
196.            srun -p ac ./pmm-OpenMP $N
197.
198.            N=1000
199.            echo "TAMA = 1000";
200.            srun -p ac ./pmm-OpenMP $N
201.
202.            echo "-------------------------"
203.
204.            export OMP_NUM_THREADS=7
205.            echo "7 PROCESADOR";
206.            N=100
207.            echo "TAMA = 100";
208.            srun -p ac ./pmm-OpenMP $N
209.
210.            N=500
211.            echo "TAMA = 500";
212.            srun -p ac ./pmm-OpenMP $N
213.
214.            N=1000
215.            echo "TAMA = 1000";
216.            srun -p ac ./pmm-OpenMP $N
217.
218.            echo "-------------------------"
219.
220.            export OMP_NUM_THREADS=8
221.            echo "8 PROCESADOR";
222.            N=100
223.            echo "TAMA = 100";
224.            srun -p ac ./pmm-OpenMP $N
225.
226.            N=500
227.            echo "TAMA = 500";
228.            srun -p ac ./pmm-OpenMP $N
229.
230.            N=1000
231.            echo "TAMA = 1000";
232.            srun -p ac ./pmm-OpenMP $N
233.
234.            echo "-------------------------"
235.
236.            export OMP_NUM_THREADS=9
237.            echo "9 PROCESADOR";
238.            N=100
239.            echo "TAMA = 100";
240.            srun -p ac ./pmm-OpenMP $N
241.
242.            N=500
243.            echo "TAMA = 500";
244.            srun -p ac ./pmm-OpenMP $N
245.
246.            N=1000
247.            echo "TAMA = 1000";
248.            srun -p ac ./pmm-OpenMP $N
249.
250.            echo "-------------------------"
251.
252.            export OMP_NUM_THREADS=10
253.            echo "10 PROCESADOR";
254.            N=100
255.            echo "TAMA = 100";
```

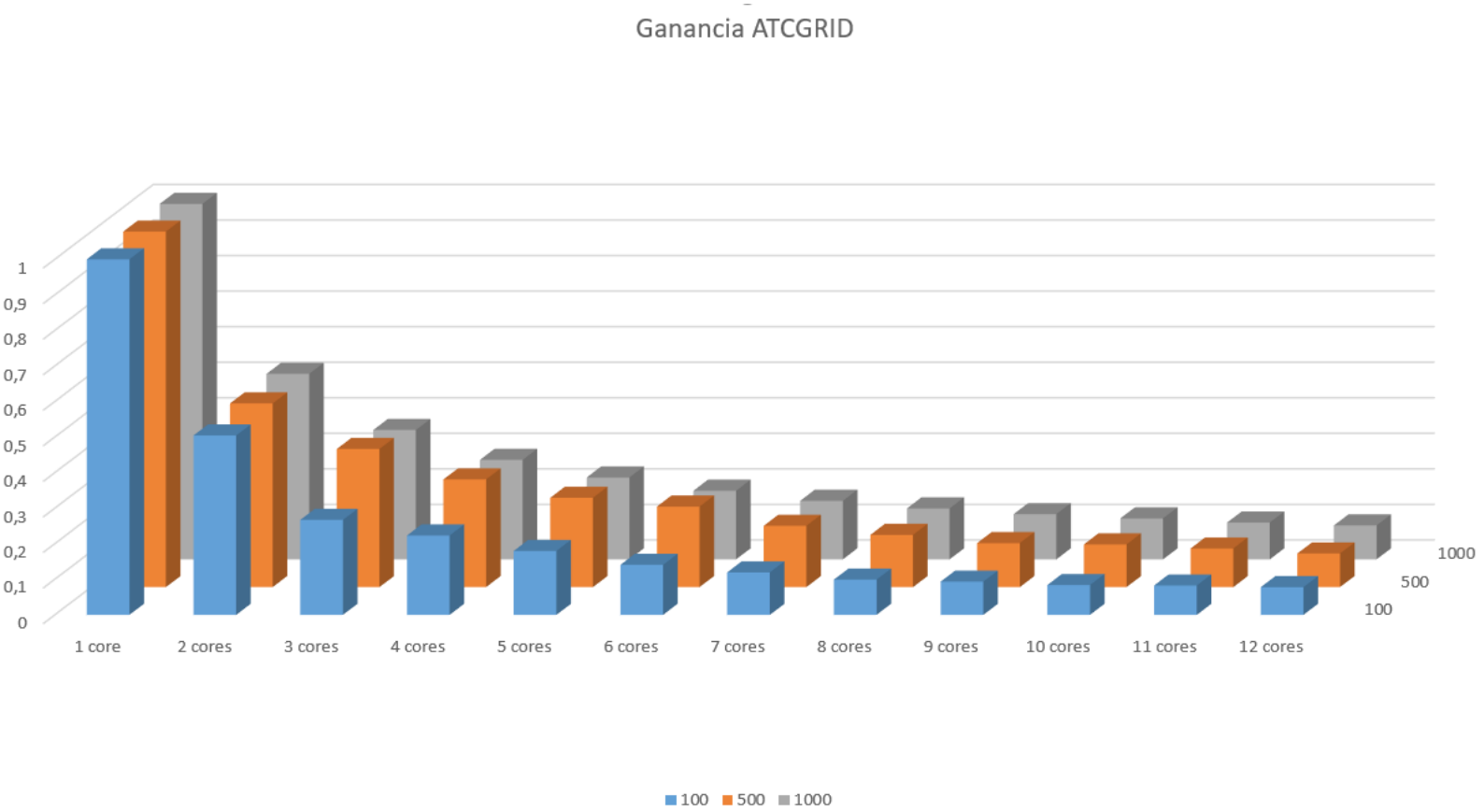
```
256.              srun -p ac ./pmm-OpenMP $N
257.
258.              N=500
259.              echo "TAMA = 500";
260.              srun -p ac ./pmm-OpenMP $N
261.
262.              N=1000
263.              echo "TAMA = 1000";
264.              srun -p ac ./pmm-OpenMP $N
265.
266.              echo "--------------------------"
267.
268.              export OMP_NUM_THREADS=11
269.              echo "11 PROCESADOR";
270.              N=100
271.              echo "TAMA = 100";
272.              srun -p ac ./pmm-OpenMP $N
273.
274.              N=500
275.              echo "TAMA = 500";
276.              srun -p ac ./pmm-OpenMP $N
277.
278.              N=1000
279.              echo "TAMA = 1000";
280.              srun -p ac ./pmm-OpenMP $N
281.
282.              echo "--------------------------"
283.
284.              export OMP_NUM_THREADS=12
285.              echo "12 PROCESADOR";
286.              N=100
287.              echo "TAMA = 100";
288.              srun -p ac ./pmm-OpenMP $N
289.
290.              N=500
291.              echo "TAMA = 500";
292.              srun -p ac ./pmm-OpenMP $N
293.
294.              N=1000
295.              echo "TAMA = 1000";
296.              srun -p ac ./pmm-OpenMP $N
```

tiempos

	cores											
tama	1 core	2 cores	3 cores	4 cores	5 cores	6 cores	7 cores	8 cores	9 cores	10 cores	11 cores	12 cores
100	0,002117037	0,001068349	0,0005655	0,00047253	0,00038056	0,00029804	0,00025282	0,000210111	0,000198116	0,00017768	0,00017476	0,00016509
500	0,34797226	0,179810324	0,13520483	0,10542838	0,08742501	0,07885397	0,06001625	0,051129003	0,04306351	0,04185323	0,03781802	0,03290598
1000	8,95542323	4,674557333	3,25930981	2,5061291	2,0633004	1,72904488	1,47474535	1,28268522	1,13909933	1,03265319	0,92779523	0,85564491

ganancia en velocidad

tamaño	cores											
	1 core	2 cores	3 cores	4 cores	5 cores	6 cores	7 cores	8 cores	9 cores	10 cores	11 cores	12 cores
100	1	0,504643518275779	0,26712051	0,22320252	0,1797602	0,1407812	0,11942257	0,09924767	0,09358174	0,08392626	0,08254839	0,07798163
500	1	0,516737523847447	0,38855061	0,30297925	0,25124131	0,22660992	0,17247425	0,14693413	0,12375559	0,12027748	0,10868113	0,09456495
1000	1	0,521980615873137	0,36394816	0,27984485	0,23039675	0,19307238	0,16467623	0,14322999	0,12719659	0,11531037	0,10360149	0,09554489



ESTUDIO DE ESCALABILIDAD EN PCLOCAL:

SCRIPT: pmm-OpenMP_pclocal.sh

```
    #!/bin/bash
297.
298.            export OMP_NUM_THREADS=1
299.            echo "1 PROCESADOR";
300.            N=100
301.            echo "TAMA = 100";
302.            ./pmm-OpenMP $N
303.
304.            N=500
305.            echo "TAMA = 500";
306.            ./pmm-OpenMP $N
307.
308.            N=1000
309.            echo "TAMA = 1000";
310.            ./pmm-OpenMP $N
311.
312.            N=1500
313.            echo "TAMA = 1500";
314.            ./pmm-OpenMP $N
315.
316.            echo "-------------------------"
317.
318.            export OMP_NUM_THREADS=2
319.            echo "2 PROCESADOR";
320.            N=100
321.            echo "TAMA = 100";
322.            ./pmm-OpenMP $N
```

```
323.
324.            N=500
325.            echo "TAMA = 500";
326.            ./pmm-OpenMP $N
327.
328.            N=1000
329.            echo "TAMA = 1000";
330.            ./pmm-OpenMP $N
331.
332.            N=1500
333.            echo "TAMA = 1500";
334.            ./pmm-OpenMP $N
335.
336.            echo "-------------------------"
337.
338.            export OMP_NUM_THREADS=3
339.            echo "3 PROCESADOR";
340.            N=100
341.            echo "TAMA = 100";
342.            ./pmm-OpenMP $N
343.
344.            N=500
345.            echo "TAMA = 500";
346.            ./pmm-OpenMP $N
347.
348.            N=1000
349.            echo "TAMA = 1000";
350.            ./pmm-OpenMP $N
351.
352.            N=1500
353.            echo "TAMA = 1500";
354.            ./pmm-OpenMP $N
355.
356.            echo "-------------------------"
357.
358.            export OMP_NUM_THREADS=4
359.            echo "4 PROCESADOR";
360.            N=100
361.            echo "TAMA = 100";
362.            ./pmm-OpenMP $N
363.
364.            N=500
365.            echo "TAMA = 500";
366.            ./pmm-OpenMP $N
367.
368.            N=1000
369.            echo "TAMA = 1000";
370.            ./pmm-OpenMP $N
371.
372.            N=1500
373.            echo "TAMA = 1500";
374.            ./pmm-OpenMP $N
```

tiempos

	cores			
tama	1 core	2 cores	3 cores	4 cores
100	0,00246033	0,00042408	0,00026217	0,00020292
500	0,10942019	0,053885521	0,03670986	0,02803964
1000	0,95164153	0,508147493	0,3549984	0,276281113
1500	5,81612495	3,142790177	2,01369217	1,59280872

ganancia en velocidad = Ts/Tp

tamaño	cores			
	1 core	2 cores	3 cores	4 cores
100	1	0,1723669814	0,10656001	0,08247586
500	1	0,4924641738	0,33549443	0,25625654
1000	1	0,5339694358	0,37303794	0,29032057
1500	1	0,5403580918	0,34622574	0,27386082