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): Juan Miguel Castro Guerrero Grupo de prácticas: B3 Fecha de entrega: Fecha evaluación en clase:

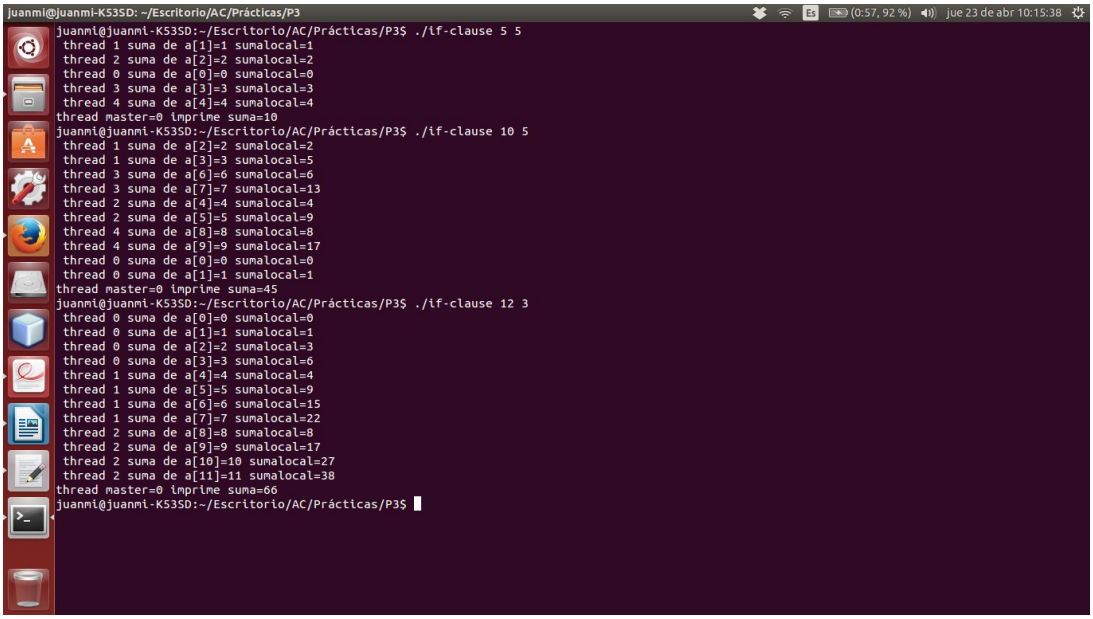
---

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

CÓDIGO FUENTE: if-clauseModificado.c

```
#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 numero threads \n");
      exit(-1);
     }
   n = atoi(argv[1]); if (n>20) n=20;
   x = atoi(argv[2]); if (x>20) x=20;
   for (i=0; i<n; i++) {
      a[i] = i;
   }

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

  return(0);
}
```

CAPTURAS DE PANTALLA:



RESPUESTA: Esta cláusula sirve para indicar el número de hebras que queremos que ejecuten un código seleccionado. En este caso indicamos el número de iteraciones del programa y el número de hebras que queremos que ejecuten el código correspondiente.

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			scheduled-clause.c			scheduleg-clause.c		
	1	2	4	1	2	4	1	2	4
0	0	0	0	0	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0
2	0	1	0	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	0	0	1	0	0	0
7	1	1	1	0	0	1	0	0	0
8	0	0	0	0	0	0	1	1	1
9	1	0	0	0	0	0	1	1	1
10	0	1	0	0	0	0	1	1	1
11	1	1	0	0	0	0	1	1	1

12	0	0	1	0	0	1	0	0	0
13	1	0	1	0	0	1	0	0	0
14	0	1	1	0	0	1	1	0	0
15	1	1	1	0	0	1	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	0	3	0	2	3	0
1	1	0	0	1	3	0	2	3	0
2	2	1	0	3	0	0	2	3	0
3	3	1	0	2	0	0	2	3	0
4	0	2	1	3	1	2	0	1	2
5	1	2	1	2	1	2	0	1	2
6	2	3	1	2	2	2	0	1	2
7	3	3	1	2	2	2	3	2	2
8	0	0	2	2	1	1	3	2	3
9	1	0	2	2	1	1	3	2	3
10	2	1	2	2	2	1	1	0	3
11	3	1	2	2	2	1	1	0	3
12	0	2	3	2	2	3	0	1	1
13	1	2	3	2	2	3	0	1	1
14	2	3	3	2	2	3	0	1	1
15	3	3	3	2	2	3	0	1	1

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

RESPUESTA: Static: “chunk” iteraciones se asignan de manera estática a los threads en round-robin. Dynamic: cada thread toma “chunk” iteraciones cada vez que está sin trabajo. Guided: cada thread toma iteraciones dinámicamente y progresivamente va tomando menos iteraciones.

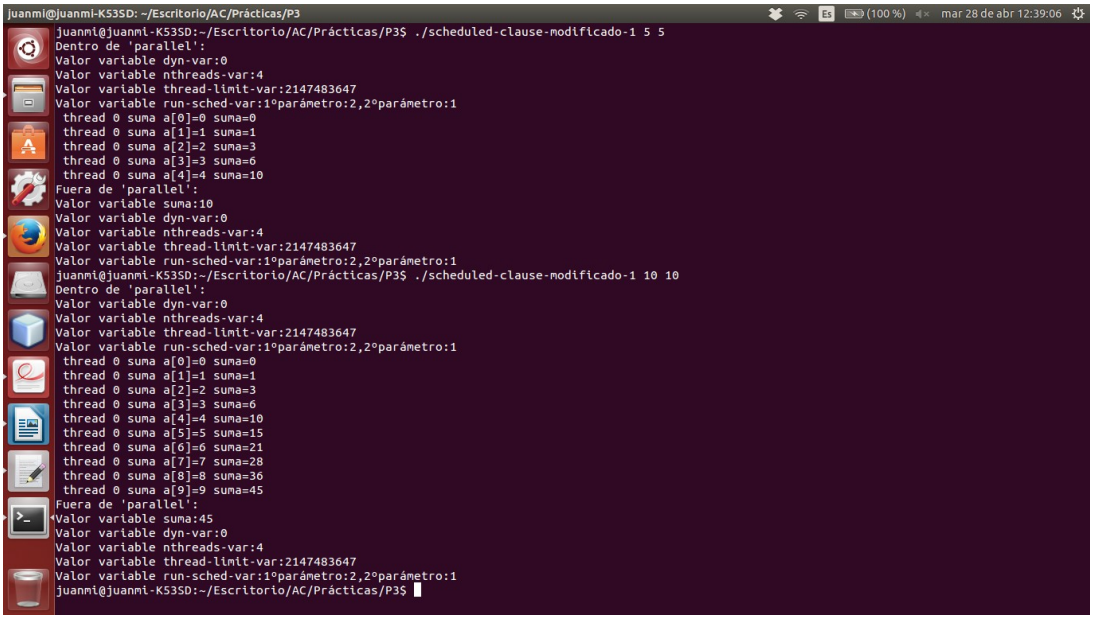
3. Añadir al programa scheduled-clause.c lo necesario para que imprima el valor de las variables de control dyn-var, nthreads-var, thread-limit-var y run-sched-var dentro (debe imprimir sólo un thread) y fuera de la región paralela. Realizar varias ejecuciones usando variables de entorno para modificar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

CÓDIGO FUENTE: scheduled-clauseModificado.c

```
#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, modifier;
  omp_sched_t kind;
   if(argc < 3)     {
      fprintf(stderr,"Falta 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 sections
                                    {
                                                printf("Dentro de 'parallel':\n");
                                                printf("Valor variable dyn-var:
%d\n",omp_get_dynamic());
                                                printf("Valor variable nthreads-var:
%d\n",omp_get_max_threads());
                                                printf("Valor variable thread-limit-var:
%d\n",omp_get_thread_limit());
                                                omp_get_schedule(&kind, &modifier);
                                                printf("Valor variable run-sched-
var:1ºparámetro:%d,2ºparámetro:%d\n",(int) 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("Fuera de 'parallel':\n");
  printf("Valor variable suma:%d\n",suma);
  printf("Valor variable dyn-var:%d\n",omp_get_dynamic());
  printf("Valor variable nthreads-var:%d\n",omp_get_max_threads());
  printf("Valor variable thread-limit-var:%d\n",omp_get_thread_limit());
  omp_get_schedule(&kind, &modifier);
  printf("Valor variable run-sched-var:1ºparámetro:%d,2ºparámetro:%d\n",(int)
kind,modifier);

  return(0);
}
```

CAPTURAS DE PANTALLA:



RESPUESTA: Se imprimen siempre los mismos valores tanto dentro como fuera del parallel.

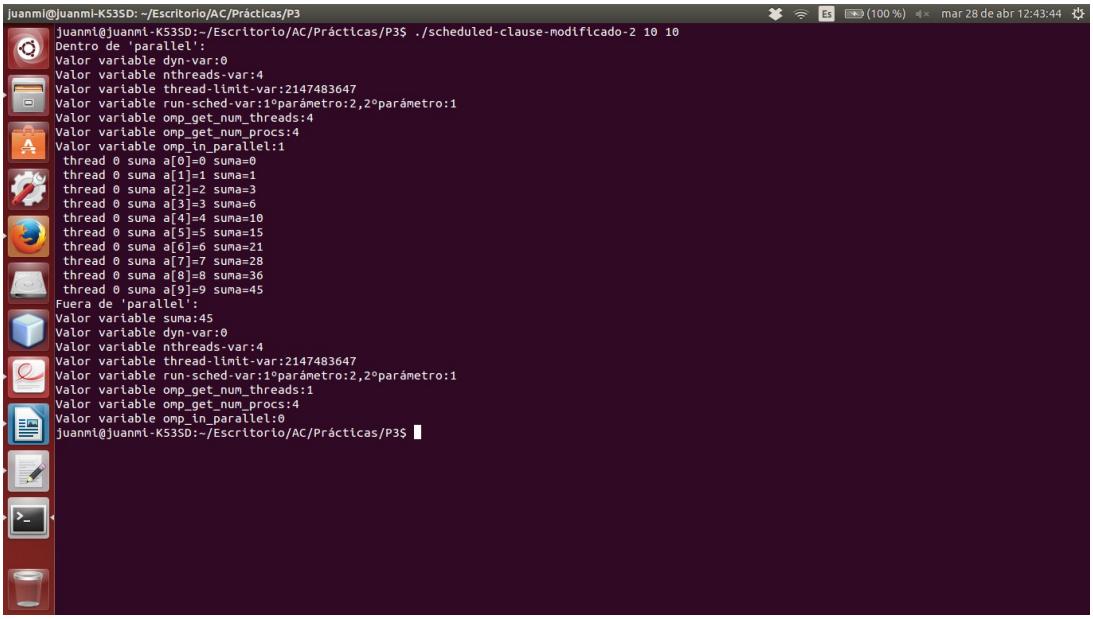
4. Usar en el ejemplo anterior las funciones omp_get_num_threads(), omp_get_num_procs() y omp_in_parallel() dentro y fuera de la región paralela. Imprimir los valores que obtienen estas funciones dentro (lo debe imprimir sólo uno de los threads) y fuera de la región paralela. Incorporar en su cuaderno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

CÓDIGO FUENTE: scheduled-clauseModificado4.c

```
#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, modifier;
  omp_sched_t kind;
   if(argc < 3)     {
      fprintf(stderr,"Falta 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 sections
                                    {
                                                printf("Dentro de 'parallel':\n");
                                                printf("Valor variable dyn-var:
%d\n",omp_get_dynamic());
                                                printf("Valor variable nthreads-var:
%d\n",omp_get_max_threads());
                                                printf("Valor variable thread-limit-var:
%d\n",omp_get_thread_limit());
                                                omp_get_schedule(&kind, &modifier);
                                                printf("Valor variable run-sched-
var:1ºparámetro:%d,2ºparámetro:%d\n",(int) kind,modifier);
                                                printf("Valor variable
omp_get_num_threads:%d\n",omp_get_num_threads());
                                                printf("Valor variable
omp_get_num_procs:%d\n",omp_get_num_procs());
                                                printf("Valor variable 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':\n");
  printf("Valor variable suma:%d\n",suma);
  printf("Valor variable dyn-var:%d\n",omp_get_dynamic());
  printf("Valor variable nthreads-var:%d\n",omp_get_max_threads());
  printf("Valor variable thread-limit-var:%d\n",omp_get_thread_limit());
  omp_get_schedule(&kind, &modifier);
  printf("Valor variable run-sched-var:1ºparámetro:%d,2ºparámetro:%d\n",(int)
kind,modifier);
  printf("Valor variable omp_get_num_threads:%d\n",omp_get_num_threads());
  printf("Valor variable omp_get_num_procs:%d\n",omp_get_num_procs());
  printf("Valor variable omp_in_parallel:%d\n",omp_in_parallel());

  return(0);
}
```

CAPTURAS DE PANTALLA:



RESPUESTA: Se obtienen siempre diferentes valores en las funciones omp_get_num_threads() ya que indica el número de threads que se están usando en una región paralela y omp_in_parallel() ya que devuelve 1 si se llama a la rutina dentro de una región parallel activa o 0 en caso contrario.

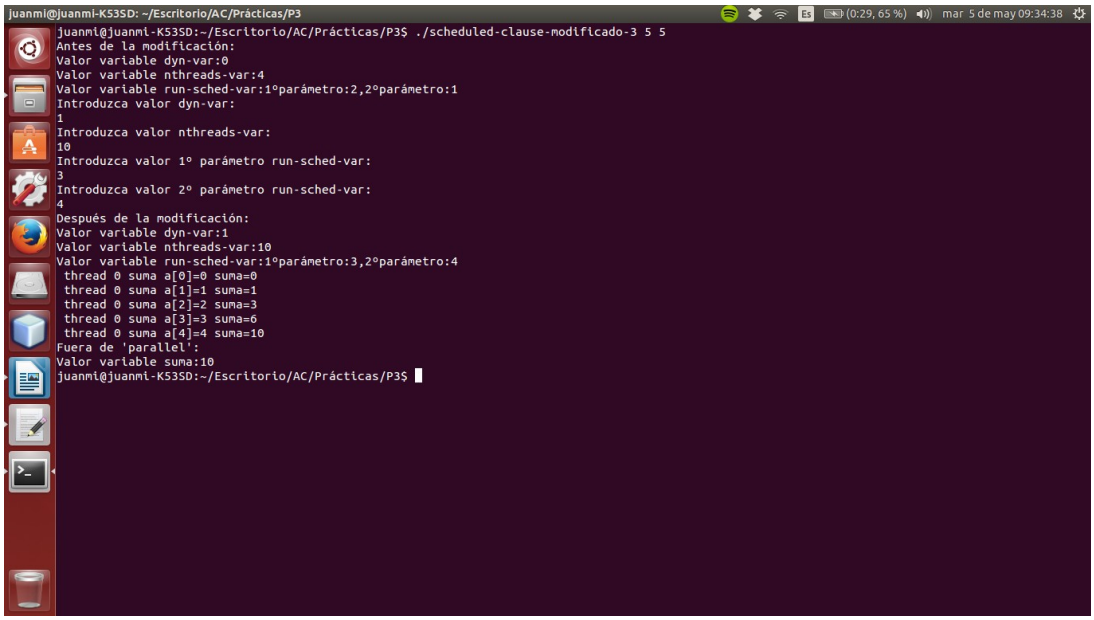
5. Añadir al programa scheduled-clause.c lo necesario para modificar las variables de control dyn-var, nthreads-var y run-sched-var y para poder imprimir el valor de estas variables antes y después de dicha modificación. Incorporar en su cuaderno de prácticas volcados de pantalla con los resultados de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

CÓDIGO FUENTE: scheduled-clauseModificado5.c

```
#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, dyn, nthreads, modifier;
  omp_sched_t kind;
   if(argc < 3)     {
      fprintf(stderr,"Falta 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;

   printf("Antes de la modificación:\n");
   printf("Valor variable dyn-var:%d\n",omp_get_dynamic());
   printf("Valor variable nthreads-var:%d\n",omp_get_max_threads());
   omp_get_schedule(&kind, &modifier);
   printf("Valor variable run-sched-var:1ºparámetro:%d,2ºparámetro:%d\n",(int)
kind,modifier);

   #pragma omp parallel
   {

                                 #pragma omp single
                                 {
                                                printf("Introduzca valor dyn-var:\n");
                                                scanf("%d", &dyn );
                                                if( dyn == 0 || dyn == 1 )
omp_set_dynamic(dyn);
                                                else { fprintf(stderr,"El valor de dyn debe
ser 0 o 1\n"); exit(-1); }

                                                printf("Introduzca valor nthreads-var:\n");
                                                scanf("%d", &nthreads );
```

```
                                        if( nthreads > 1 && nthreads <
omp_get_thread_limit() ) omp_set_num_threads(nthreads);
                                        else { fprintf(stderr,"El valor de nthreads
debe ser mayor que 0\n"); exit(-1); }

                                        printf("Introduzca valor 1º parámetro run-
sched-var:\n");
                                        scanf("%d", (int *) &kind );
                                        printf("Introduzca valor 2º parámetro run-
sched-var:\n");
                                        scanf("%d", &modifier );
                                        if( (int) kind >= 0 || modifier > 0 )
omp_set_schedule(kind, modifier);
                                        else { fprintf(stderr,"El valor de kind debe
ser mayor o igual que 0 y el valor de modifier mayor que 0\n"); exit(-1); }

                                }

                                #pragma omp sections
                                {
                                        printf("Después de la modificación:\n");
                                        printf("Valor variable dyn-var:
%d\n",omp_get_dynamic());
                                        printf("Valor variable nthreads-var:
%d\n",omp_get_max_threads());
                                        omp_get_schedule(&kind, &modifier);
                                        printf("Valor variable run-sched-
var:1ºparámetro:%d,2ºparámetro:%d\n",(int) 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("Fuera de 'parallel':\n");
  printf("Valor variable suma:%d\n",suma);

  return(0);
}
```

CAPTURAS DE PANTALLA:



6. Implementar un programa secuencial en C que multiplique una matriz triangular por un vector.

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

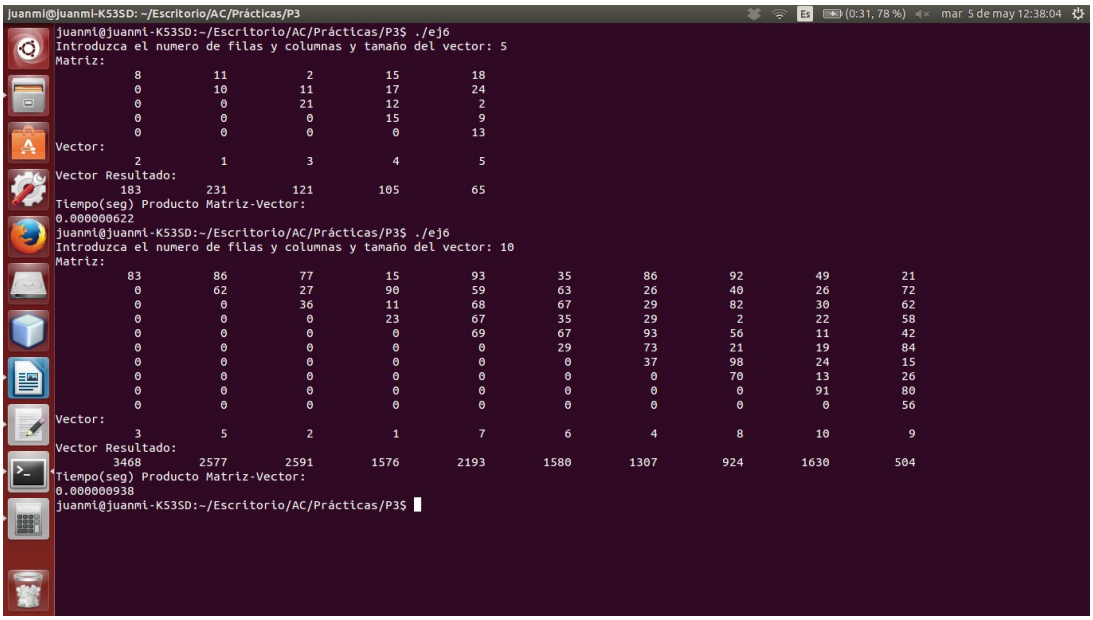
CÓDIGO FUENTE: pmtv-secuencial.c

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

int main(void)
{
   int z, y, x, i, g, temp, filascolumnas, auxfilascolumnas=0;
                  struct timespec cgt1,cgt2;
   double ncgt;

   printf ("Introduzca el numero de filas y columnas y tamaño del vector: ");
   scanf ("%i",&filascolumnas);

   int matriz[filascolumnas][filascolumnas];
   temp = filascolumnas*filascolumnas;

   for(i=0; i<filascolumnas; i++){
                  for(g=0; g<filascolumnas; g++)
```

```
                                matriz[i][g] = 0;
}

for(i=auxfilascolumnas; i<filascolumnas; i++){
                for(g=auxfilascolumnas; g<filascolumnas; g++)
                                matriz[i][g] = rand() % temp;
                auxfilascolumnas++;
}

printf("Matriz:\n");
for (i=0; i<filascolumnas; i++){
                                for(g=0; g<filascolumnas; g++)
                                                printf("%12i",matriz[i][g]);
                                printf("\n");
}

int v1[filascolumnas],v2[filascolumnas];

for(i=0; i<filascolumnas; i++)
                                v1[i] = 0;

for(i=0; i<filascolumnas; i++)
                                v2[i] = 0;

for(i=1; i<=filascolumnas; i++){
                do{
                                z = rand() % filascolumnas;

                }while(v1[z]);

                                v1[z] = i;
}

printf("Vector:\n");
for(i=0; i<filascolumnas; i++)
                                printf("%12i",v1[i]);

                printf("\n");

                clock_gettime(CLOCK_REALTIME,&cgt1);
for(i=0; i<filascolumnas; i++){
                for(g=0; g<filascolumnas; g++)
                                v2[i]+=matriz[i][g]*v1[g];
}
clock_gettime(CLOCK_REALTIME,&cgt2);
```

```
    ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+(double) ((cgt2.tv_nsec-cgt1.tv_nsec)/
(1.e+9));

    printf("Vector Resultado:\n");
    for(i=0; i<filascolumnas; i++)
                                        printf("%12i",v2[i]);

                    printf("\nTiempo(seg) Producto Matriz-Vector:\n");
                    printf("%12.9f\t",ncgt);

    printf("\n");
}
```

CAPTURAS DE PANTALLA:



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. 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 que multiplica una matriz triangular por un vector con las alternativas de planificación static, dynamic y guided para chunk de 2, 64, 128, 1024 y el chunk por defecto para la alternativa. No use vectores mayores de 32768 componentes ni menores de 4096 componentes. El número de threads en las ejecuciones debe coincidir con el número de cores. Rellenar la Tabla 3 con los tiempos obtenidos, ponga en la tabla el número de threads que utilizan las ejecuciones. Representar el tiempo para static, dynamic y guided en función del tamaño del chunk en una gráfica. Rellenar la tabla y realizar la gráfica también para el PC local. ¿Qué alternativa ofrece mejores prestaciones? Razone por qué.

RESPUESTA: La mejor alternativa sería utilizar la directiva schedule con el parámetro guided ya que cada thread toma iteraciones dinámicamente y progresivamente va tomando menos iteraciones sobrecargando menos la CPU.

CÓDIGO FUENTE: pmtv-OpenMP.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 z, y, x, i, g, temp, filascolumnas, auxfilascolumnas=0, chunk;
		double inicio,final,diferencia;

		if(argc < 3)	{
	fprintf(stderr,"\nFalta filas-columnas-tam y/o chunk\n");
	exit(-1);
    }
   filascolumnas= atoi(argv[1]);
		chunk = atoi(argv[2]);

   int matriz[filascolumnas][filascolumnas];
   temp = filascolumnas*filascolumnas;

   for(i=0; i<filascolumnas; i++){
		for(g=0; g<filascolumnas; g++)
			matriz[i][g] = 0;
   }

   for(i=auxfilascolumnas; i<filascolumnas; i++){
		for(g=auxfilascolumnas; g<filascolumnas; g++)
			matriz[i][g] = rand() % temp;
		auxfilascolumnas++;
   }
   /*
   printf("Matriz inicial:\n");
   for (i=0; i<filascolumnas; i++){
			for(g=0; g<filascolumnas; g++)
				printf("%12i",matriz[i][g]);
			printf("\n");
   }
		*/
```

```
int v1[filascolumnas],v2[filascolumnas];

for(i=0; i<filascolumnas; i++)
                        v1[i] = 0;

for(i=0; i<filascolumnas; i++)
                        v2[i] = 0;

for(i=1; i<=filascolumnas; i++){
        do{
                        z = rand() % filascolumnas;

        }while(v1[z]);

                        v1[z] = i;
}
/*
printf("Vector inicial:\n");
for(i=0; i<filascolumnas; i++)
                        printf("%12i",v1[i]);

printf("\n");
        */
        inicio = omp_get_wtime();
        #pragma omp parallel
        {
                        /*
                        #pragma omp sections
                        {
                                        printf("Dentro de 'parallel':\n");
                                        printf("\tValor variable dyn-var:
%d\n",omp_get_dynamic());
                                        printf("\tValor variable
omp_get_num_threads:%d\n",omp_get_num_threads());
        }
        */
        #pragma omp for schedule(guided,chunk)
        for(i=0; i<filascolumnas; i++){
                        for(g=0; g<filascolumnas; g++)
                                        v2[i]+=matriz[i][g]*v1[g];
        }
}
final = omp_get_wtime();

diferencia = final - inicio;
        /*
```

```
    printf("Vector Resultado:\n");
    for(i=0; i<filascolumnas; i++)
                                    printf("%12i",v2[i]);

                    printf("\n");
                    */
                    printf("Tiempo(seg) Producto Matriz-Vector:\n");
                    printf("%21.9f\t",diferencia);

    printf("\n");
}
```

CAPTURAS DE PANTALLA:

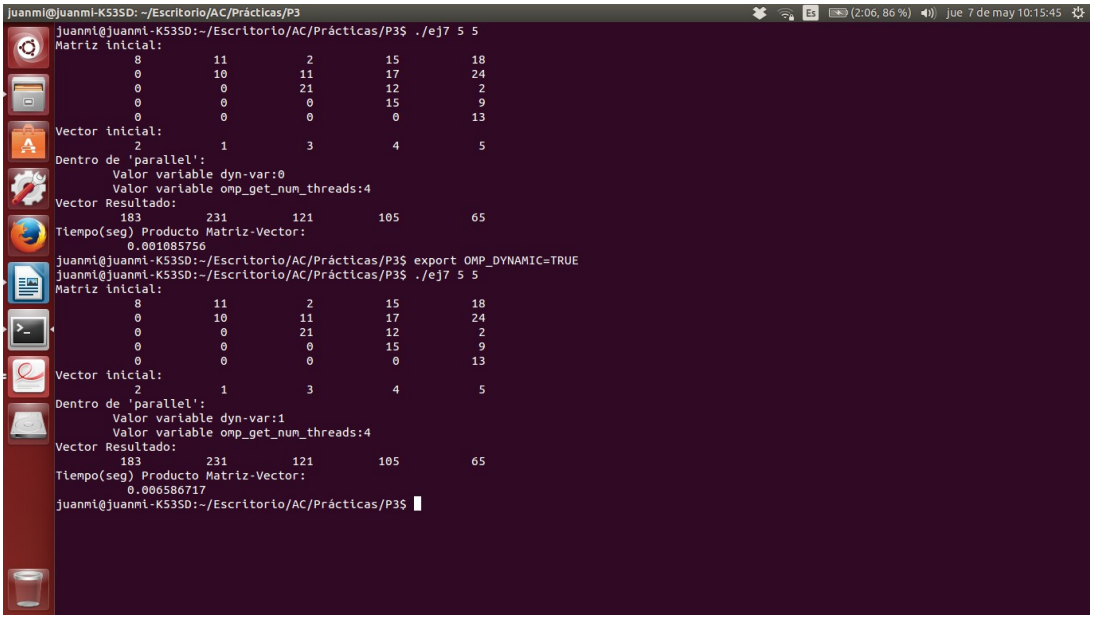


TABLA RESULTADOS Y GRÁFICA ATCGRID

Tabla 3 . Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones paralelas

Chunk	Static 1 thread	Dynamic 1 thread	Guided 1 thread
por defecto			
Tam 16 Chunk 2	0.013355811	0.011184466	0.010868773
Tam 64 Chunk 32	0.006535527	0.011992972	0.008297428
Tam 256 Chunk 64	0.002864886	0.006017303	0.009330367
Tam 1024 Chunk 1024	0.006239991	0.005576854	0.006892460

TABLA RESULTADOS Y GRÁFICA PC LOCAL

Tabla 4 . Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones paralelas

Chunk	Static 4 threads	Dynamic 4 threads	Guided 4 threads
por defecto			
Tam 16 Chunk 2	0.009165960	0.003717393	0.000428868
Tam 64 Chunk 32	0.002168381	0.002113696	0.000361648
Tam 256 Chunk 64	0.001265930	0.003127445	0.005502619
Tam 1024 Chunk 1024	0.006494135	0.005267736	0.007373912

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.

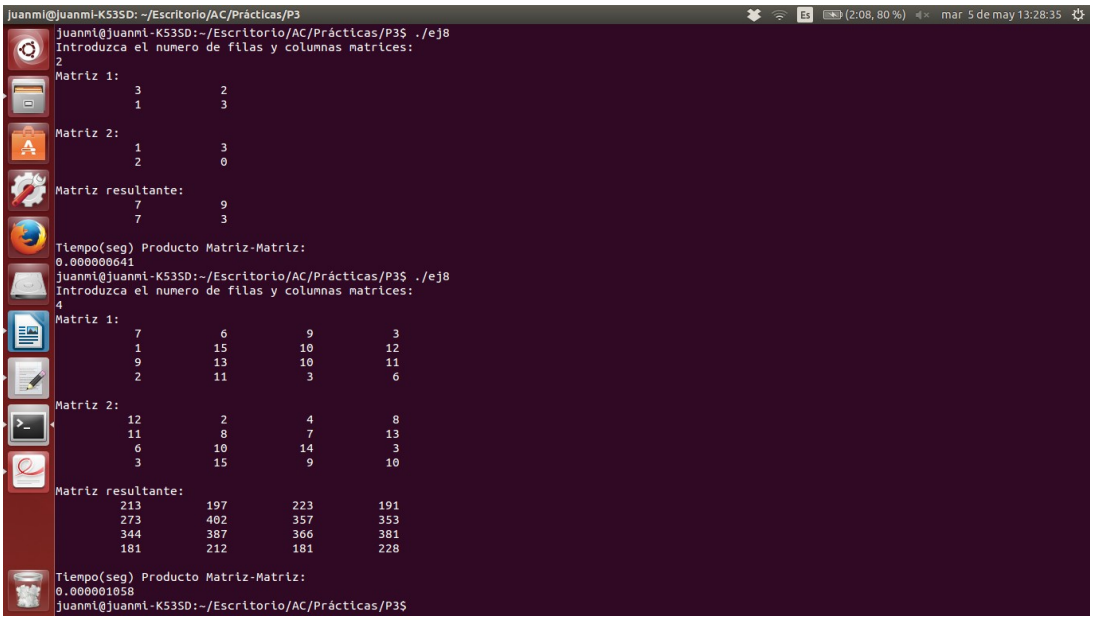
CÓDIGO FUENTE: pmm-secuencial.c

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

int main(void)
{
   int z, y, x, i, g, temp, filascolumnas, auxfilascolumnas=0, chunk;
                double inicio,final,diferencia;

                if(argc < 2)    {
      fprintf(stderr,"\nFalta chunk \n");
      exit(-1);
    }
                chunk = atoi(argv[1]);

   printf ("Introduzca el numero de filas y columnas y tamaño del vector: ");
   scanf ("%i",&filascolumnas);
```

```
int matriz[filascolumnas][filascolumnas];
temp = filascolumnas*filascolumnas;

for(i=0; i<filascolumnas; i++){
		for(g=0; g<filascolumnas; g++)
				matriz[i][g] = 0;
}

for(i=auxfilascolumnas; i<filascolumnas; i++){
		for(g=auxfilascolumnas; g<filascolumnas; g++)
				matriz[i][g] = rand() % temp;
		auxfilascolumnas++;
}

printf("Matriz:\n");
for (i=0; i<filascolumnas; i++){
				for(g=0; g<filascolumnas; g++)
						printf("%12i",matriz[i][g]);
				printf("\n");
}

int v1[filascolumnas],v2[filascolumnas];

for(i=0; i<filascolumnas; i++)
				v1[i] = 0;

for(i=0; i<filascolumnas; i++)
				v2[i] = 0;

for(i=1; i<=filascolumnas; i++){
		do{
				z = rand() % filascolumnas;

		}while(v1[z]);

				v1[z] = i;
}

printf("Vector:\n");
for(i=0; i<filascolumnas; i++)
				printf("%12i\t",v1[i]);

printf("\n");

		inicio = omp_get_wtime();
```

```
                #pragma omp parallel for schedule(static,chunk)
    for(i=0; i<filascolumnas; i++){
                for(g=0; g<filascolumnas; g++)
                                v2[i]+=matriz[i][g]*v1[g];
    }
    final = omp_get_wtime();

    diferencia = final - inicio;

    printf("Vector Resultado:\n");
    for(i=0; i<filascolumnas; i++)
                                printf("%12i\t",v2[i]);

                printf("\nTiempo(seg) Producto Matriz-Vector:\n");
                printf("%12f\t",diferencia);

    printf("\n");
}
```

CAPTURAS DE PANTALLA:



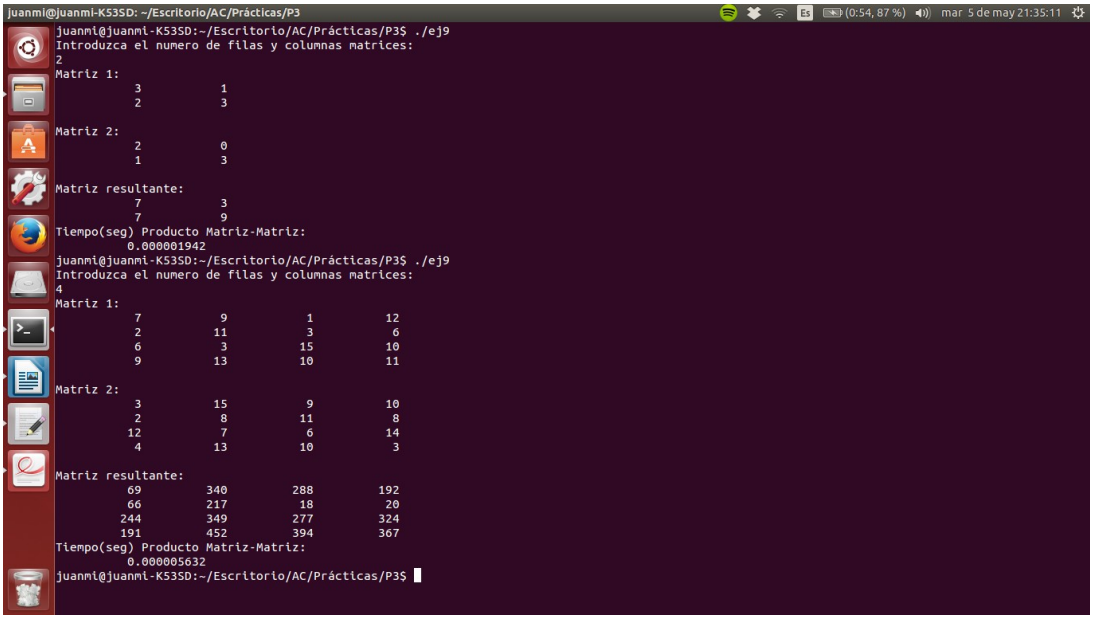
9. Implementar en paralelo la multiplicación de matrices cuadradas con OpenMP a partir del código escrito en el ejercicio anterior. Use las directivas, las cláusulas y las funciones de entorno que considere oportunas. Se debe paralelizar también la inicialización de las matrices. Dibuje en su cuaderno 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).

CÓDIGO FUENTE: pmm-OpenMP.c

```
#include <stdio.h>
#include <stdlib.h>
#ifdef _OPENMP
		#include <omp.h>
#else
		#define omp_get_thread_num() 0
#endif
int main()
{
		int i, j, k, N;
		double inicio,final,diferencia;

		printf("Introduzca el numero de filas y columnas matrices:\n");
		scanf("%d",&N);

		int mat1[N][N],mat2[N][N],mul[N][N],tam=N*N;

		// Inicializar y Almacenar los valores de las matrices
		#pragma omp parallel for private(j)
		for(i=0;i<N;i++)
				for(j=0;j<N;j++)
						mat1[i][j]=0;

		#pragma omp parallel for private(j)
		for(i=0;i<N;i++)
				for(j=0;j<N;j++)
						mat2[i][j]=0;

		#pragma omp parallel for private(j)
		for(i=0;i<N;i++)
				for(j=0;j<N;j++){
						mat1[i][j] = rand() % tam;
				}

		#pragma omp parallel for private(j)
		for(i=0;i<N;i++)
				for(j=0;j<N;j++){
						mat2[i][j] = rand() % tam;
				}

		//Imprimir los valores de las matrices en forma de filas y columas

		printf("Matriz 1:\n");
		for (i=0;i<N;i++){
				for(j=0;j<N;j++)
```

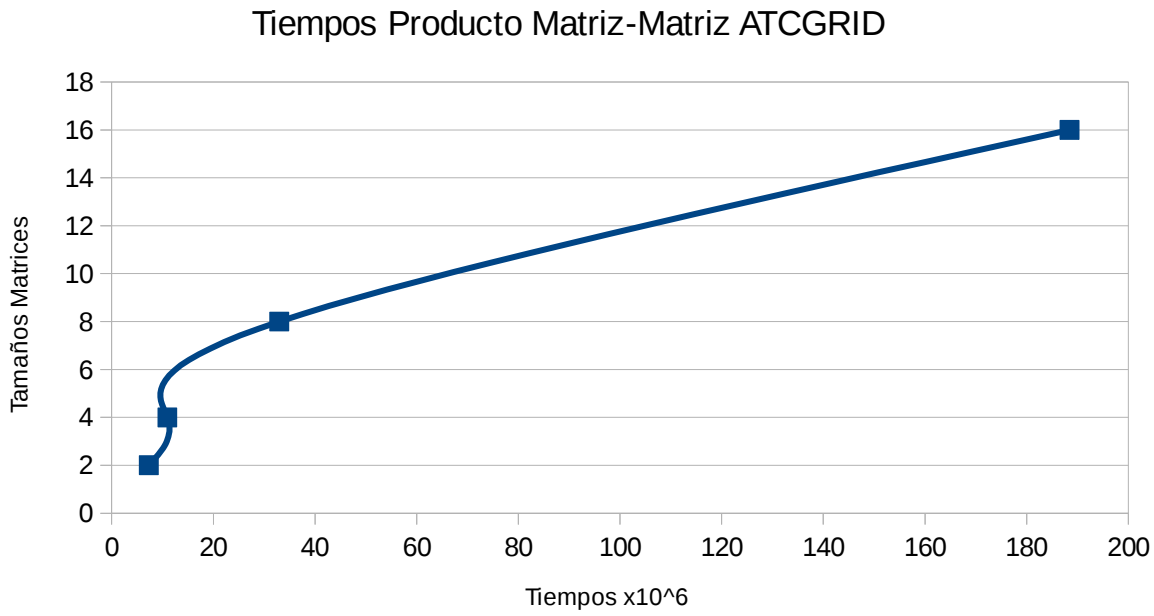
```
                                        printf("%12i",mat1[i][j]);
                        printf("\n");
                }
                printf("\n");

                printf("Matriz 2:\n");
                for (i=0;i<N;i++){
                        for(j=0;j<N;j++)
                                printf("%12i",mat2[i][j]);
                        printf("\n");
                }
                printf("\n");

                //Realiza la multiplicación e imprime el resultado
                #pragma omp parallel for private(j)
                for(i=0;i<N;i++)
                        for(j=0;j<N;j++)
                                mul[i][j]=0;

                inicio = omp_get_wtime();
                #pragma omp parallel for private(j)
                for(i=0;i<N;i++)
                        for(j=0;j<N;j++)
                                for(k=0;k<N;k++)
                                        mul[i][j]+=(mat1[i]
[k]*mat2[k][j]);
                final = omp_get_wtime();

                diferencia = final - inicio;

                printf("Matriz resultante:\n");
                for (i=0;i<N;i++){
                        for(j=0;j<N;j++)
                                printf("%12i",mul[i][j]);
                        printf("\n");
                }

                printf("Tiempo(seg) Producto Matriz-Matriz:\n");
                printf("%21.9f\t",diferencia);

   printf("\n");

}
```

CAPTURAS DE PANTALLA:

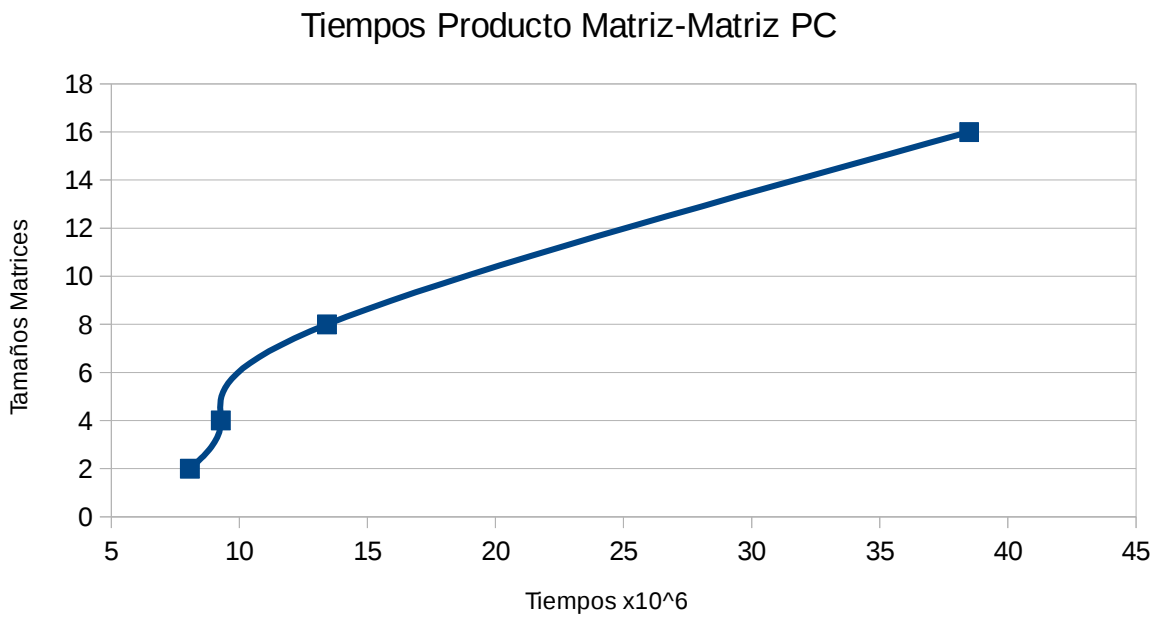


10. Hacer un estudio de escalabilidad (ganancia en velocidad en función del número de cores) en atcgrid y en el PC local del código paralelo implementado para tres tamaños de las matrices. Debe recordar usar –O2 al compilar. Presente los resultados del estudio en tablas de valores y en gráficas. Escoger los tamaños de manera que se observe diferentes curvas de escalabilidad en las gráficas que entregue en su cuaderno de prácticas . Consulte la Lección 6/Tema 2.

ESTUDIO DE ESCALABILIDAD EN ATCGRID:



ESTUDIO DE ESCALABILIDAD EN PCLOCAL:



11.