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): Antonio Martín Ruiz
Grupo de prácticas:
Fecha de entrega:
Fecha evaluación en clase:

Ejercicios basados en los ejemplos del seminario práctico

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

CÓDIGO FUENTE: `if-clauseModificado.c`

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

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

  if (argc != 2){
    printf("ERROR: Argumentos incorrectos. Modo de uso: ./if-clause
<numero_de_threads>\n");
    return -1;
  }

  int numero_de_threads = atoi(argv[1]);
  int i, n=20, tid;
  int a[n], suma=0, sumalocal;

  if(argc < 2) {
    fprintf(stderr,"[ERROR]-Falta iteraciones\n");
    exit(-1);
  }

  n = atoi(argv[1]);
  if (n>20)
    n=20;

  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(numero_de_threads)
  {
    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:

```
amrantonio@HAL9000:~/DGIIM/2DGIIM/AC/Prácticas/Práctica3/src/Ejercicio1$ gcc -O2 -fopenmp if-clause.c -o if-clause
amrantonio@HAL9000:~/DGIIM/2DGIIM/AC/Prácticas/Práctica3/src/Ejercicio1$ ./if-clause 4
 thread 0 suma de a[0]=0 sumalocal=0
 thread 2 suma de a[2]=2 sumalocal=2
 thread 1 suma de a[1]=1 sumalocal=1
 thread 3 suma de a[3]=3 sumalocal=3
thread master=0 imprime suma=6
amrantonio@HAL9000:~/DGIIM/2DGIIM/AC/Prácticas/Práctica3/src/Ejercicio1$ ./if-clause 4
 thread 2 suma de a[2]=2 sumalocal=2
 thread 3 suma de a[3]=3 sumalocal=3
 thread 0 suma de a[0]=0 sumalocal=0
 thread 1 suma de a[1]=1 sumalocal=1
thread master=0 imprime suma=6
amrantonio@HAL9000:~/DGIIM/2DGIIM/AC/Prácticas/Práctica3/src/Ejercicio1$ ./if-clause 4
 thread 2 suma de a[2]=2 sumalocal=2
 thread 0 suma de a[0]=0 sumalocal=0
 thread 3 suma de a[3]=3 sumalocal=3
 thread 1 suma de a[1]=1 sumalocal=1
thread master=0 imprime suma=6
```

RESPUESTA:
Como se puede comprobar, efectivamente la clausula thread_num modifica el número de threads que realizan el bloque de código.

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

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	1
1	1	0	0	1	0	0	0	0	1
2	0	1	0	0	1	0	0	0	1
3	1	1	0	0	1	0	0	0	1
4	0	0	1	0	0	1	0	0	1

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

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

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

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

RESPUESTA:

Utilizando static las tareas se reparten en bloques del tamaño del chunk siguiendo la planificación round-robin. Con dynamic y guided no se sigue esta planificación, pero sí se cumple que a cada thread se le asignan tareas de tamaño chunck.

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;
  omp_sched_t tipo;
  int chunk2;

  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 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);

     #pragma omp critical
     {
     omp_get_schedule(&tipo, &chunk2);
     printf("DENTRO DE PARALLEL: dyn-var: %d, nthreads-var: %d, thread-limit-
var: %d, run-sched-var: %d, chunk: %d\n", omp_get_dynamic(),
omp_get_max_threads(), omp_get_thread_limit(), tipo, chunk2);
    }
  }

  printf("Fuera de 'parallel for' suma=%d\n",suma);
  omp_get_schedule(&tipo, &chunk2);
  printf("FUERA DE PARALLEL: dyn-var: %d, nthreads-var: %d, thread-limit-var:
%d, run-sched-var: %d, chunk: %d\n", omp_get_dynamic(), omp_get_max_threads(),
omp_get_thread_limit(), tipo, chunk2);

}
```

CAPTURAS DE PANTALLA:

```
amrantonio@HAL9000:~/DGIIM/2DGIIM/AC/Prácticas/Práctica3/src/Ejercicio3$ gcc -O2 -fopenmp scheduled-clause.c -o scheduled-clause
amrantonio@HAL9000:~/DGIIM/2DGIIM/AC/Prácticas/Práctica3/src/Ejercicio3$ ./scheduled-clause 5 1
 thread 0 suma a[1]=1 suma=1
 thread 3 suma a[3]=3 suma=3
DENTRO DE PARALLEL: dyn-var: 0, nthreads-var: 4, thread-limit-var: 2147483647, run-sched-var: 3, chunk: 3
 thread 0 suma a[4]=4 suma=5
DENTRO DE PARALLEL: dyn-var: 0, nthreads-var: 4, thread-limit-var: 2147483647, run-sched-var: 3, chunk: 3
DENTRO DE PARALLEL: dyn-var: 0, nthreads-var: 4, thread-limit-var: 2147483647, run-sched-var: 3, chunk: 3
 thread 2 suma a[2]=2 suma=2
DENTRO DE PARALLEL: dyn-var: 0, nthreads-var: 4, thread-limit-var: 2147483647, run-sched-var: 3, chunk: 3
 thread 1 suma a[0]=0 suma=0
DENTRO DE PARALLEL: dyn-var: 0, nthreads-var: 4, thread-limit-var: 2147483647, run-sched-var: 3, chunk: 3
Fuera de 'parallel for' suma=5
FUERA DE PARALLEL: dyn-var: 0, nthreads-var: 4, thread-limit-var: 2147483647, run-sched-var: 3, chunk: 3
amrantonio@HAL9000:~/DGIIM/2DGIIM/AC/Prácticas/Práctica3/src/Ejercicio3$ export OMP_SCHEDULE="static, 2"
amrantonio@HAL9000:~/DGIIM/2DGIIM/AC/Prácticas/Práctica3/src/Ejercicio3$ ./scheduled-clause 5 1
 thread 1 suma a[0]=0 suma=0
 thread 0 suma a[3]=3 suma=3
 thread 3 suma a[2]=2 suma=2
 thread 2 suma a[1]=1 suma=1
DENTRO DE PARALLEL: dyn-var: 0, nthreads-var: 4, thread-limit-var: 2147483647, run-sched-var: 1, chunk: 2
DENTRO DE PARALLEL: dyn-var: 0, nthreads-var: 4, thread-limit-var: 2147483647, run-sched-var: 1, chunk: 2
DENTRO DE PARALLEL: dyn-var: 0, nthreads-var: 4, thread-limit-var: 2147483647, run-sched-var: 1, chunk: 2
DENTRO DE PARALLEL: dyn-var: 0, nthreads-var: 4, thread-limit-var: 2147483647, run-sched-var: 1, chunk: 2
 thread 1 suma a[4]=4 suma=4
DENTRO DE PARALLEL: dyn-var: 0, nthreads-var: 4, thread-limit-var: 2147483647, run-sched-var: 1, chunk: 2
Fuera de 'parallel for' suma=4
FUERA DE PARALLEL: dyn-var: 0, nthreads-var: 4, thread-limit-var: 2147483647, run-sched-var: 1, chunk: 2
amrantonio@HAL9000:~/DGIIM/2DGIIM/AC/Prácticas/Práctica3/src/Ejercicio3$ export OMP_SCHEDULE="guided, 3"
amrantonio@HAL9000:~/DGIIM/2DGIIM/AC/Prácticas/Práctica3/src/Ejercicio3$ ./scheduled-clause 5 1
 thread 0 suma a[0]=0 suma=0
DENTRO DE PARALLEL: dyn-var: 0, nthreads-var: 4, thread-limit-var: 2147483647, run-sched-var: 3, chunk: 3
 thread 0 suma a[4]=4 suma=4
DENTRO DE PARALLEL: dyn-var: 0, nthreads-var: 4, thread-limit-var: 2147483647, run-sched-var: 3, chunk: 3
 thread 3 suma a[2]=2 suma=2
DENTRO DE PARALLEL: dyn-var: 0, nthreads-var: 4, thread-limit-var: 2147483647, run-sched-var: 3, chunk: 3
 thread 1 suma a[3]=3 suma=3
DENTRO DE PARALLEL: dyn-var: 0, nthreads-var: 4, thread-limit-var: 2147483647, run-sched-var: 3, chunk: 3
 thread 2 suma a[1]=1 suma=1
DENTRO DE PARALLEL: dyn-var: 0, nthreads-var: 4, thread-limit-var: 2147483647, run-sched-var: 3, chunk: 3
Fuera de 'parallel for' suma=4
FUERA DE PARALLEL: dyn-var: 0, nthreads-var: 4, thread-limit-var: 2147483647, run-sched-var: 3, chunk: 3
```

RESPUESTA:

Los valores son iguales tanto dentro como 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.

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;
  omp_sched_t tipo;
  int chunk2;
```

```
  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 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);

     #pragma omp critical
     {
     omp_get_schedule(&tipo, &chunk2);
     printf("DENTRO DE PARALLEL: dyn-var: %d, nthreads-var: %d, thread-limit-
var: %d, run-sched-var: %d, chunk: %d\n", omp_get_dynamic(),
omp_get_max_threads(), omp_get_thread_limit(), tipo, chunk2);
     printf("num_threads: %d, num_procs: %d, in_parallel: %d\n",
omp_get_num_threads(), omp_get_num_procs(), omp_in_parallel());
    }
  }

  printf("Fuera de 'parallel for' suma=%d\n",suma);
  omp_get_schedule(&tipo, &chunk2);
  printf("FUERA DE PARALLEL: dyn-var: %d, nthreads-var: %d, thread-limit-var:
%d, run-sched-var: %d, chunk: %d\n", omp_get_dynamic(), omp_get_max_threads(),
omp_get_thread_limit(), tipo, chunk2);
  printf("num_threads: %d, num_procs: %d, in_parallel: %d\n",
omp_get_num_threads(), omp_get_num_procs(), omp_in_parallel());

}
```

CAPTURAS DE PANTALLA:

```
amrantonio@HAL9000:~/DGIIM/2DGIIM/AC/Practicas/Practica3/src/Ejercicio4$ ./scheduled-clause 5 1
 thread 2 suma a[0]=0 suma=0
DENTRO DE PARALLEL: dyn-var: 0, nthreads-var: 4, thread-limit-var: 2147483647, run-sched-var: 3, chunk: 3
num_threads: 4, num_procs: 4, in_parallel: 1
 thread 2 suma a[4]=4 suma=4
DENTRO DE PARALLEL: dyn-var: 0, nthreads-var: 4, thread-limit-var: 2147483647, run-sched-var: 3, chunk: 3
num_threads: 4, num_procs: 4, in_parallel: 1
 thread 1 suma a[1]=1 suma=1
DENTRO DE PARALLEL: dyn-var: 0, nthreads-var: 4, thread-limit-var: 2147483647, run-sched-var: 3, chunk: 3
 thread 0 suma a[3]=3 suma=3
 thread 3 suma a[2]=2 suma=2
num_threads: 4, num_procs: 4, in_parallel: 1
DENTRO DE PARALLEL: dyn-var: 0, nthreads-var: 4, thread-limit-var: 2147483647, run-sched-var: 3, chunk: 3
num_threads: 4, num_procs: 4, in_parallel: 1
DENTRO DE PARALLEL: dyn-var: 0, nthreads-var: 4, thread-limit-var: 2147483647, run-sched-var: 3, chunk: 3
num_threads: 4, num_procs: 4, in_parallel: 1
Fuera de 'parallel for' suma=4
FUERA DE PARALLEL: dyn-var: 0, nthreads-var: 4, thread-limit-var: 2147483647, run-sched-var: 3, chunk: 3
num_threads: 1, num_procs: 4, in_parallel: 0
```

RESPUESTA:

Las funciones en las que se obtienen valores distintos son `omp_get_num_threads()` y `omp_in_parallel()`.

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;
  omp_sched_t tipo;
  int chunk2;

  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;

    omp_get_schedule(&tipo, &chunk2);
    printf("ANTES DEL CAMBIO: dyn-var: %d, nthreads-var: %d, thread-limit-var:
%d, run-sched-var: %d, chunk: %d\n", omp_get_dynamic(), omp_get_max_threads(),
omp_get_thread_limit(), tipo, chunk2);
    printf("num_threads: %d, num_procs: %d, in_parallel: %d\n",
omp_get_num_threads(), omp_get_num_procs(), omp_in_parallel());


    omp_set_dynamic(2);
    omp_set_num_threads(2);
    omp_set_schedule(2,1);

  #pragma omp parallel 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);

     #pragma omp critical
     {
     omp_get_schedule(&tipo, &chunk2);
     printf("DENTRO DE PARALLEL: dyn-var: %d, nthreads-var: %d, thread-limit-
```

```
var: %d, run-sched-var: %d, chunk: %d\n", omp_get_dynamic(),
omp_get_max_threads(), omp_get_thread_limit(), tipo, chunk2);
     printf("num_threads: %d, num_procs: %d, in_parallel: %d\n",
omp_get_num_threads(), omp_get_num_procs(), omp_in_parallel());
    }
  }

  printf("Fuera de 'parallel for' suma=%d\n",suma);
  omp_get_schedule(&tipo, &chunk2);
  printf("DESPUÉS DEL CAMBIO: dyn-var: %d, nthreads-var: %d, thread-limit-var:
%d, run-sched-var: %d, chunk: %d\n", omp_get_dynamic(), omp_get_max_threads(),
omp_get_thread_limit(), tipo, chunk2);
  printf("num_threads: %d, num_procs: %d, in_parallel: %d\n",
omp_get_num_threads(), omp_get_num_procs(), omp_in_parallel());

}
```

CAPTURAS DE PANTALLA:

```
amrantonio@HAL9000:~/DGIIM/2DGIIM/AC/Prácticas/Práctica3/src/Ejercicio5$ gcc -O2 -fopenmp scheduled-clause.c -o scheduled-clause
amrantonio@HAL9000:~/DGIIM/2DGIIM/AC/Prácticas/Práctica3/src/Ejercicio5$ ./scheduled-clause 1 5
ANTES DEL CAMBIO: dyn-var: 0, nthreads-var: 4, thread-limit-var: 2147483647, run-sched-var: 2, chunk: 1
num_threads: 1, num_procs: 4, in_parallel: 0
 thread 0 suma a[0]=0 suma=0
DENTRO DE PARALLEL: dyn-var: 1, nthreads-var: 2, thread-limit-var: 2147483647, run-sched-var: 2, chunk: 1
num_threads: 1, num_procs: 4, in_parallel: 0
Fuera de 'parallel for' suma=0
DESPUÉS DEL CAMBIO: dyn-var: 1, nthreads-var: 2, thread-limit-var: 2147483647, run-sched-var: 2, chunk: 1
num_threads: 1, num_procs: 4, in_parallel: 0
```

RESPUESTA: Como se puede observar en la captura el cambio se produce entre la zona de antes del cambio y la zona de después con los valores que hemos elegido.

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.

CÓDIGO FUENTE: `pmtv-secuencial.c`

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

int main(int argc, char const *argv[]) {


  if (argc < 2){
      printf("ERROR. Introducir tamaño para matriz y vector.\n");
      return -1;
  }

  int tamanio = atoi(argv[1]);
  int i, j;
  int *vector, *vector_resultado, **matriz;

  vector = (int*) malloc(tamanio*sizeof(double));
```

```
  vector_resultado = (int*) malloc(tamanio*sizeof(double));
  matriz = (int**) malloc(tamanio*sizeof(double*));


  if ( (vector == NULL) || (vector_resultado == NULL) || (matriz == NULL)){
    printf("Error en la reserva de memoria");
    return -1;
  }

  for ( i = 0; i < tamanio; i++){
    matriz[i] = (int*) malloc(tamanio*sizeof(double));
    if (matriz[i] == NULL){
      printf("Error en la reserva de memoria para matriz");
      return -1;
    }
  }

  for ( i = 0; i < tamanio; i++){
    vector[i] = i;
    vector_resultado[i] = 0;
    for ( j = 0; j < tamanio; j++){
      if(i >= j)
      matriz[i][j] = i+j;
      else
      matriz[i][j] = 0;
    }
  }

  if (tamanio < 10){ //imprime la matriz si no es muy grande (tamaño 10)
    for ( i = 0; i < tamanio; i++){
      for ( j = 0; j < tamanio; j++){
        printf("%d ", matriz[i][j]);
      }
      printf("\n");
    }
  }

  for ( i = 0; i < tamanio; i++){
    for ( j = 0; j <= i; j++){
      vector_resultado[i] += matriz[i][j]*vector[j];
    }
  }

  int a;
  if (tamanio>15)
    a = tamanio-1;
  else
    a = 1;

  printf("resultado:\n" );
    for ( i = 0; i < tamanio; i+=a)
      printf("%d ",vector_resultado[i]);
  printf("\n");


  return 0;
}
```

CAPTURAS DE PANTALLA:
(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```
amrantonio@HAL9000:~/DGIIM/2DGIIM/AC/Prácticas/Práctica3/src/Ejercicio6$ gcc pmtv-secuencial.c -O2 -o pmtv-secuencial
amrantonio@HAL9000:~/DGIIM/2DGIIM/AC/Prácticas/Práctica3/src/Ejercicio6$ ./pmtv-secuencial 5
0 0 0 0 0
1 2 0 0 0
2 3 4 0 0
3 4 5 6 0
4 5 6 7 8
resultado:
0 2 11 32 70
amrantonio@HAL9000:~/DGIIM/2DGIIM/AC/Prácticas/Práctica3/src/Ejercicio6$ ./pmtv-secuencial 10
resultado:
0 2 11 32 70 130 217 336 492 690
amrantonio@HAL9000:~/DGIIM/2DGIIM/AC/Prácticas/Práctica3/src/Ejercicio6$ ./pmtv-secuencial 100
resultado:
0 818400
```

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) Según al documentación de OpenMP static no tiene valor por defecto y para dynamic y guided el valor es 1.
b)tamaño de chunk*2
c)En static y dynamic el número de operaciones que realiza cada thread depende del tamaño de chunk. En guided puede variar.

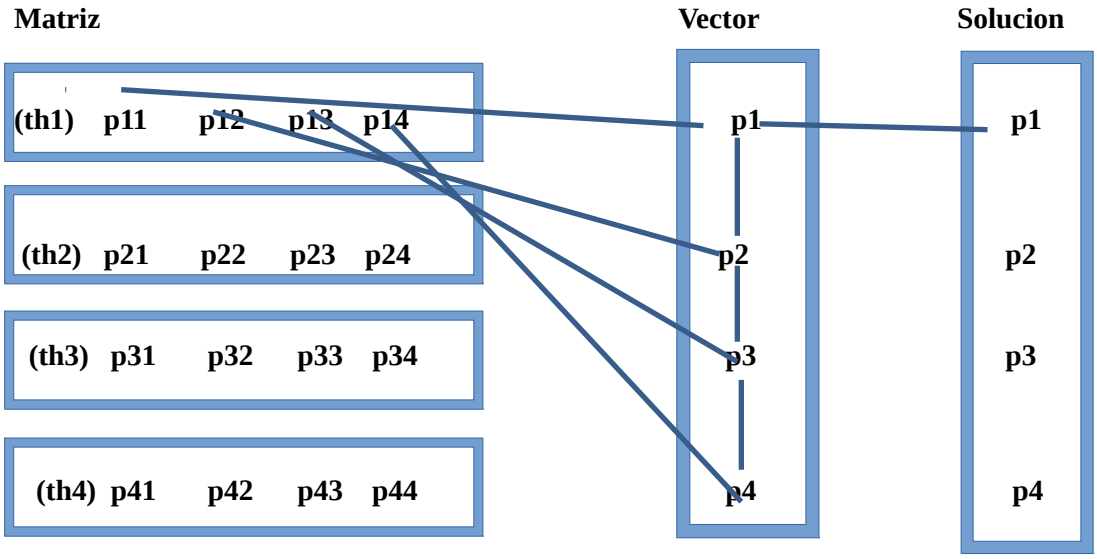
CÓDIGO FUENTE: `pmtv-OpenMP.c`

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


int main(int argc, char const *argv[]) {
```

```
  if (argc < 2){
      printf("ERROR. ./pmtv-paralelo <tamaño> \n");
      return -1;
  }


  int tamanio = atoi(argv[1]);
  int i, j;

  long int *vector, *vector_resultado, **matriz;


  vector = (long int*) malloc(tamanio*sizeof(long int));
  vector_resultado = (long int*) malloc(tamanio*sizeof(long int));
  matriz = (long int**) malloc(tamanio*sizeof(long int*));

  if ( (vector == NULL) || (vector_resultado == NULL) || (matriz == NULL)){
    printf("Error en la reserva de memoria");
    return -1;
  }

  for ( i = 0; i < tamanio; i++){
    matriz[i] = (long int*) malloc(tamanio*sizeof(long int));
    if (matriz[i] == NULL){
      printf("Error en la reserva de memoria para matriz");
      return -1;
    }
  }


  for ( i = 0; i < tamanio; i++){
    vector[i] = 2;
    vector_resultado[i] = 0;
    for ( j = 0; j < tamanio; j++){
      if(i >= j)
      matriz[i][j] = 5;
      else
      matriz[i][j] = 0;
    }
  }


  if (tamanio < 5){ //imprime la matriz si no es muy grande (tamaño 10)
    for ( i = 0; i < tamanio; i++){
      for ( j = 0; j < tamanio; j++){
        printf("%ld ", matriz[i][j]);
      }
      printf("\n");
    }
  }

  for ( i = 0; i < tamanio; i++){
    int acumulador = 0;
    #pragma omp parallel for reduction(+:acumulador) schedule(runtime)
    for ( j = 0; j <= i; j++){
      acumulador += matriz[i][j]*vector[j];
    }
    vector_resultado[i] += acumulador;
  }

  int a;
  if (tamanio>15)
```

```
    a = tamanio-1;
  else
    a = 1;

  printf("resultado:\n" );
    for ( i = 0; i < tamanio; i+=a)
      printf("%ld ",vector_resultado[i]);
  printf("\n");
  return 0;
}
```

DESCOMPOSICIÓN DE DOMINIO:



CAPTURAS DE PANTALLA:
(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```
amrantonio@HAL9000:~/DGIIM/2DGIIM/AC/Prácticas/Práctica3/src/Ejercicio7$ gcc -fopenmp -O2 pmtv-paralelo.c -o pmtv-paralelo
amrantonio@HAL9000:~/DGIIM/2DGIIM/AC/Prácticas/Práctica3/src/Ejercicio7$ export OMP_SCHEDULE="guided,3"
amrantonio@HAL9000:~/DGIIM/2DGIIM/AC/Prácticas/Práctica3/src/Ejercicio7$ ./pmtv-paralelo 20
resultado:
10 200
amrantonio@HAL9000:~/DGIIM/2DGIIM/AC/Prácticas/Práctica3/src/Ejercicio7$ export OMP_SCHEDULE="dynamic,2"
amrantonio@HAL9000:~/DGIIM/2DGIIM/AC/Prácticas/Práctica3/src/Ejercicio7$ ./pmtv-paralelo 20
resultado:
10 200
```

```
[E1estudiante13@atcgrid ~]$ qsub pmvt-OpenMP_atcgrid.sh -q ac
57676.atcgrid
[E1estudiante13@atcgrid ~]$ ls
hello          pmvt-OpenMP_atcgrid.sh         pmvt-OpenMP_atcgrid.sh.o57676
pmtv-paralelo  pmvt-OpenMP_atcgrid.sh.e57676
[E1estudiante13@atcgrid ~]$ cat pmvt-OpenMP_atcgrid.sh.o57676
Static por defecto
resultado:
10 50000
TIEMPO: 0.203188
Static chunk 1
resultado:
10 50000
TIEMPO: 0.158973
Static chunk 64
resultado:
10 50000
TIEMPO: 0.144849
Dynamic por defecto
resultado:
10 50000
TIEMPO: 0.570742
Dynamic chunk 1
resultado:
10 50000
TIEMPO: 0.507245
Dynamic chunk 64
resultado:
10 50000
TIEMPO: 0.119722
Guided por defecto
resultado:
10 50000
TIEMPO: 0.359632
Guided chunk 1
resultado:
10 50000
TIEMPO: 0.356686
Guided chunk 64
resultado:
10 50000
TIEMPO: 0.154356
```

TABLA RESULTADOS, SCRIPT Y GRÁFICA ATCGRID

SCRIPT: pmvt-OpenMP_atcgrid.sh

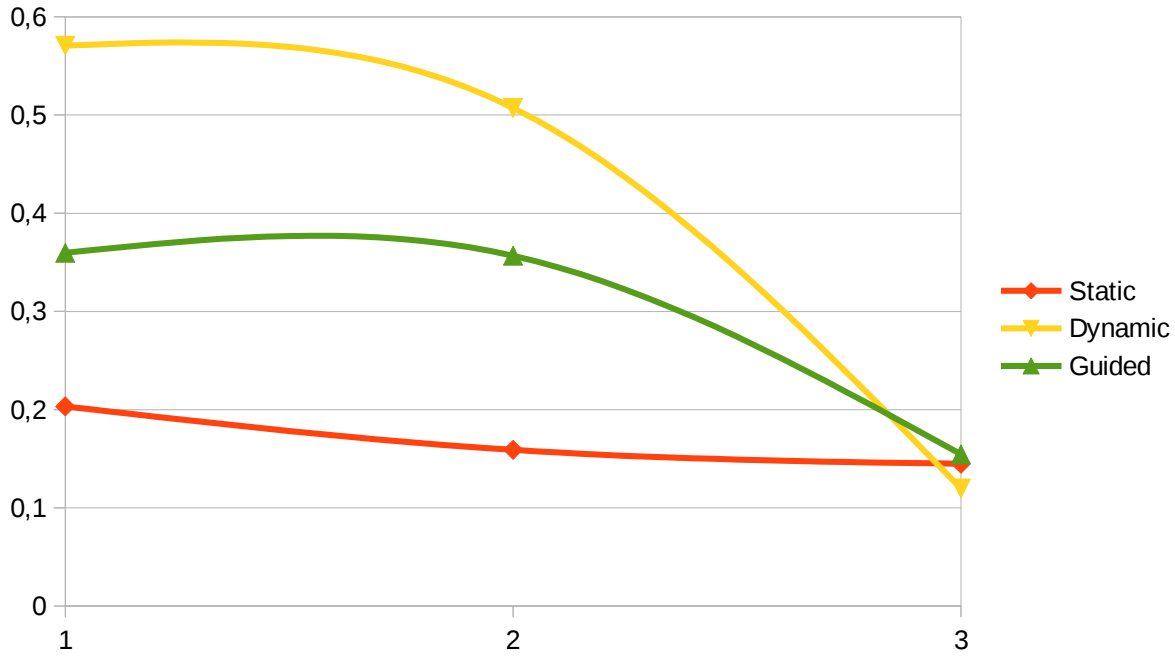
```
#!/bin/bash

echo "Static por defecto"
export OMP_SCHEDULE="static"
./pmtv-paralelo 5000
echo "Static chunk 1"
export OMP_SCHEDULE="static,1"
./pmtv-paralelo 5000
echo "Static chunk 64"
export OMP_SCHEDULE="static,64"
./pmtv-paralelo 5000
echo "Dynamic por defecto"
export OMP_SCHEDULE="dynamic"
./pmtv-paralelo 5000
echo "Dynamic chunk 1"
export OMP_SCHEDULE="dynamic,1"
./pmtv-paralelo 5000
echo "Dynamic chunk 64"
export OMP_SCHEDULE="dynamic,64"
./pmtv-paralelo 5000
echo "Guided por defecto"
export OMP_SCHEDULE="guided"
```

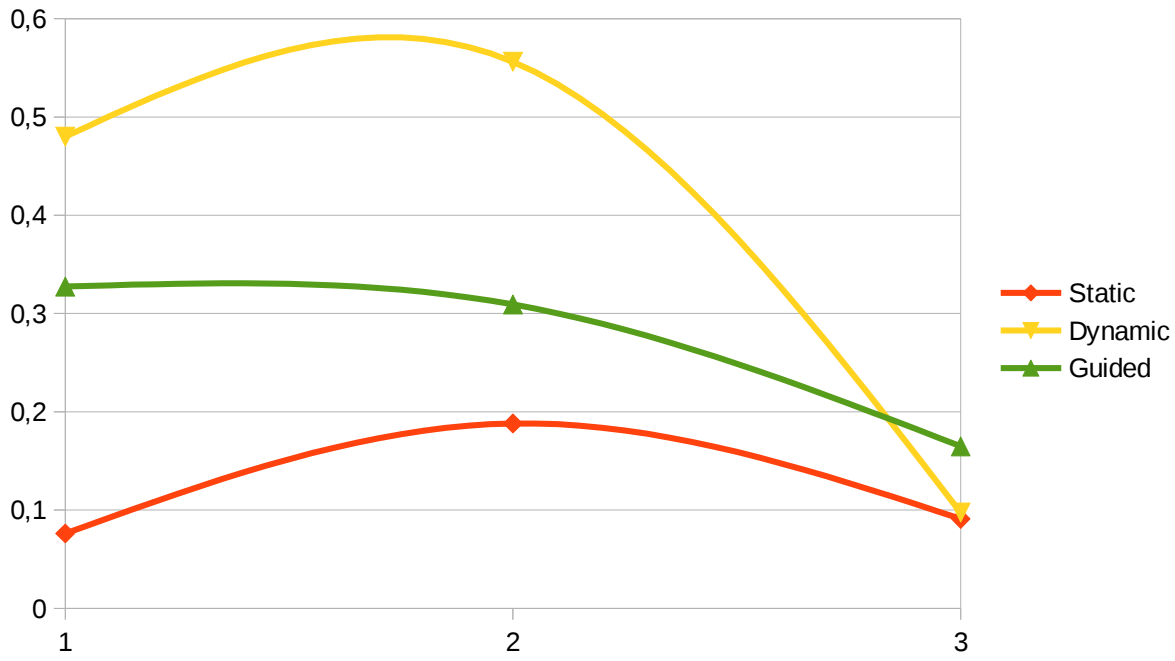
```
31. ./pmtv-paralelo 5000
32. echo "Guided chunk 1"
33. export OMP_SCHEDULE="guided,1"
34. ./pmtv-paralelo 5000
35. echo "Guided chunk 64"
36. export OMP_SCHEDULE="guided,64"
37. ./pmtv-paralelo 5000
```

Tabla 2 . Tiempos de ejecución de la versión paralela del producto de una matriz triangular por un vector r para vectores de tamaño N=5000 , 12 threads

Chunk	Static	Dynamic	Guided
por defecto	0,203188	0,570742	0,359632
1	0,158973	0,507245	0,356686
64	0,144849	0,119722	0,154356

Chunk	Static	Dynamic	Guided
por defecto	0,076158	0,480049	0,327435
1	0,188093	0,556238	0,309265
64	0,091019	0,097361	0,164909





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>

int main(int argc, char const *argv[]) {


  if (argc < 2){
      printf("ERROR. Introducir tamaño para matriz.\n");
      return -1;
  }

  int tamanio = atoi(argv[1]);
  int i, j, k;
  int **matriz_1, **matriz_2, **matriz_resultado;


  matriz_1 = (int**) malloc(tamanio*sizeof(int*));
  matriz_2 = (int**) malloc(tamanio*sizeof(int*));
  matriz_resultado = (int**) malloc(tamanio*sizeof(int*));



  if ( (matriz_1 == NULL) || (matriz_2 == NULL) || (matriz_resultado == NULL))
{
     printf("Error en la reserva de memoria");
```

```
    return -1;
  }

  for ( i = 0; i < tamanio; i++){
    matriz_1[i] = (int*) malloc(tamanio*sizeof(int));
    if (matriz_1[i] == NULL){
      printf("Error en la reserva de memoria para matriz");
      return -1;
    }
  }

  for ( i = 0; i < tamanio; i++){
    matriz_2[i] = (int*) malloc(tamanio*sizeof(int));
    if (matriz_2[i] == NULL){
      printf("Error en la reserva de memoria para matriz");
      return -1;
    }
  }

  for ( i = 0; i < tamanio; i++){
    matriz_resultado[i] = (int*) malloc(tamanio*sizeof(int));
    if (matriz_resultado[i] == NULL){
      printf("Error en la reserva de memoria para matriz");
      return -1;
    }
  }

  for ( i = 0; i < tamanio; i++){
    for ( j = 0; j < tamanio; j++){
      matriz_1[i][j] = 1;
      matriz_2[i][j] = 2;
      matriz_resultado[i][j] = 0;
    }
  }

  for ( i = 0; i < tamanio; i++){
    for ( j = 0; j < tamanio; j++){
      for ( k = 0; k < tamanio; k++){
        matriz_resultado[i][j] += matriz_1[i][k] * matriz_2[k][j];
      }
    }
  }

    printf("resultado:\n" );
    printf("%d, %d\n", matriz_resultado[0][0], matriz_resultado[tamanio-1]
[tamanio-1]);
  return 0;
}
```

CAPTURAS DE PANTALLA:
(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```
amrantonio@HAL9000:~/DGIIM/2DGIIM/AC/Prácticas/Práctica3/src/Ejercicio8$ gcc -O2 pmm-secuencial.c -o pmm-secuencial
amrantonio@HAL9000:~/DGIIM/2DGIIM/AC/Prácticas/Práctica3/src/Ejercicio8$ ./pmm-secuencial 5
resultado:
10, 10
amrantonio@HAL9000:~/DGIIM/2DGIIM/AC/Prácticas/Práctica3/src/Ejercicio8$ ./pmm-secuencial 4
resultado:
8, 8
amrantonio@HAL9000:~/DGIIM/2DGIIM/AC/Prácticas/Práctica3/src/Ejercicio8$ ./pmm-secuencial 3
resultado:
6, 6
amrantonio@HAL9000:~/DGIIM/2DGIIM/AC/Prácticas/Práctica3/src/Ejercicio8$ ./pmm-secuencial 2
resultado:
4, 4
```

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:

Se reparten las filas de la matriz resultado y cada thread recorre las filas y columnas correspondientes de las otras matrices para realizar las sumas y multiplicaciones de las posiciones de la matriz que le son asignadas.

CÓDIGO FUENTE: `pmm-OpenMP.c`

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

int main(int argc, char const *argv[]) {


  if (argc < 2){
      printf("ERROR. Introducir tamaño para matriz.\n");
      return -1;
  }

  int tamanio = atoi(argv[1]);
  int i, j, k;
  int **matriz_1, **matriz_2, **matriz_resultado;


  matriz_1 = (int**) malloc(tamanio*sizeof(int*));
  matriz_2 = (int**) malloc(tamanio*sizeof(int*));
  matriz_resultado = (int**) malloc(tamanio*sizeof(int*));



  if ( (matriz_1 == NULL) || (matriz_2 == NULL) || (matriz_resultado == NULL))
{
    printf("Error en la reserva de memoria");
    return -1;
  }

  for ( i = 0; i < tamanio; i++){
    matriz_1[i] = (int*) malloc(tamanio*sizeof(int));
    if (matriz_1[i] == NULL){
      printf("Error en la reserva de memoria para matriz");
      return -1;
```

```
    }
  }

  for ( i = 0; i < tamanio; i++){
    matriz_2[i] = (int*) malloc(tamanio*sizeof(int));
    if (matriz_2[i] == NULL){
      printf("Error en la reserva de memoria para matriz");
      return -1;
    }
  }

  for ( i = 0; i < tamanio; i++){
    matriz_resultado[i] = (int*) malloc(tamanio*sizeof(int));
    if (matriz_resultado[i] == NULL){
      printf("Error en la reserva de memoria para matriz");
      return -1;
    }
  }

  #pragma omp parallel for private(j)
  for ( i = 0; i < tamanio; i++){
    for ( j = 0; j < tamanio; j++){
      matriz_1[i][j] = 1;
      matriz_2[i][j] = 2;
      matriz_resultado[i][j] = 0;
    }
  }

  #pragma omp parallel for private(j,k)
  for ( i = 0; i < tamanio; i++){
    for ( j = 0; j < tamanio; j++){
      for ( k = 0; k < tamanio; k++){
        matriz_resultado[i][j] += matriz_1[i][k] * matriz_2[k][j];
      }
    }
  }

    printf("resultado:\n" );
    printf("%d, %d\n", matriz_resultado[0][0], matriz_resultado[tamanio-1]
[tamanio-1]);
  return 0;
}
```

CAPTURAS DE PANTALLA:
(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```
amrantonio@HAL9000:~/DGIIM/2DGIIM/AC/Prácticas/Práctica3/src/Ejercicio9$ gcc -O2 -fopenmp pmm-OpenMP.c -o pmm-OpenMP
amrantonio@HAL9000:~/DGIIM/2DGIIM/AC/Prácticas/Práctica3/src/Ejercicio9$ ./pmm-OpenMP 5
resultado:
10, 10
amrantonio@HAL9000:~/DGIIM/2DGIIM/AC/Prácticas/Práctica3/src/Ejercicio9$ ./pmm-OpenMP 4
resultado:
8, 8
amrantonio@HAL9000:~/DGIIM/2DGIIM/AC/Prácticas/Práctica3/src/Ejercicio9$ ./pmm-OpenMP 3
resultado:
6, 6
amrantonio@HAL9000:~/DGIIM/2DGIIM/AC/Prácticas/Práctica3/src/Ejercicio9$ ./pmm-OpenMP 2
resultado:
4, 4
```

38. 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 dos 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 (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.

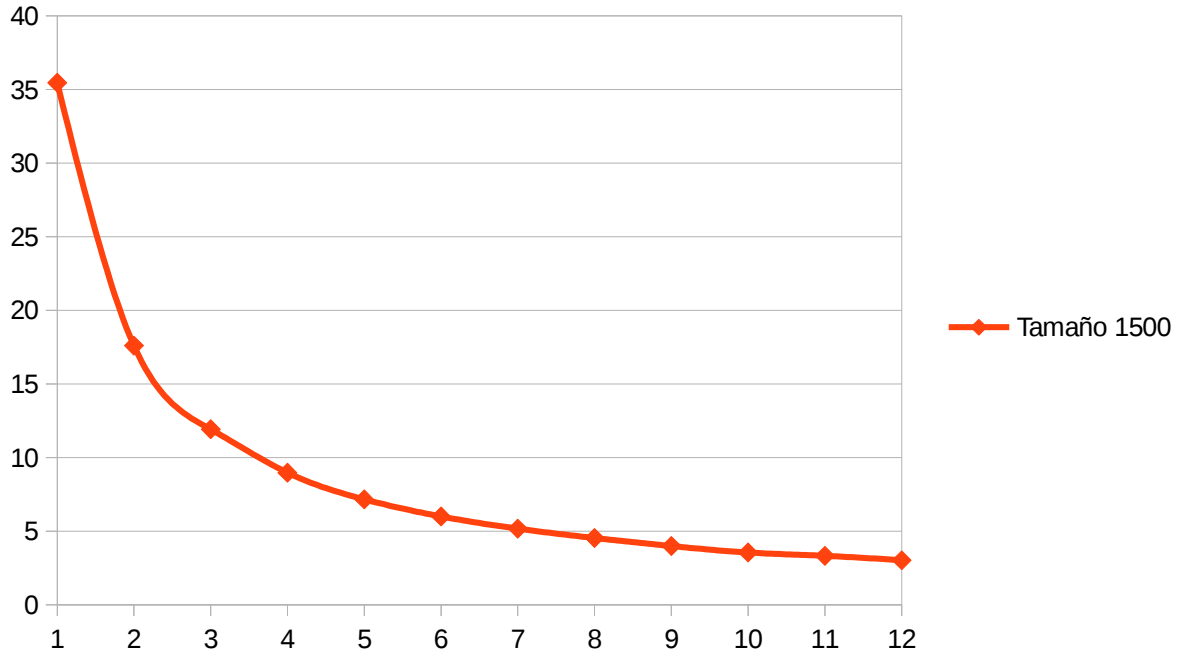
ESTUDIO DE ESCALABILIDAD EN ATCGRID:

SCRIPT: pmm-OpenMP_atcgrid.sh

```
#!/bin/bash

for ((i=1;i<=12;i=i+1))
do
    echo "NUMERO DE THREADS: $i"
    export OMP_NUM_THREADS=$i
    ./pmm-OpenMP 100
    ./pmm-OpenMP 1500

done
```

48.

	Tamaño 100	Tamaño 1500
1	0,00279	35,455054
2	0,01406	17,600267
3	0,001012	11,919513
4	0,000755	8,972131
5	0,000588	7,1608002
6	0,000492	5,999749
7	0,000428	5,1712
8	0,000586	4,53346
9	0,002128	3,99311
10	0,000428	3,54852
11	0,000402	3,3254585
12	0,000385	3,021457

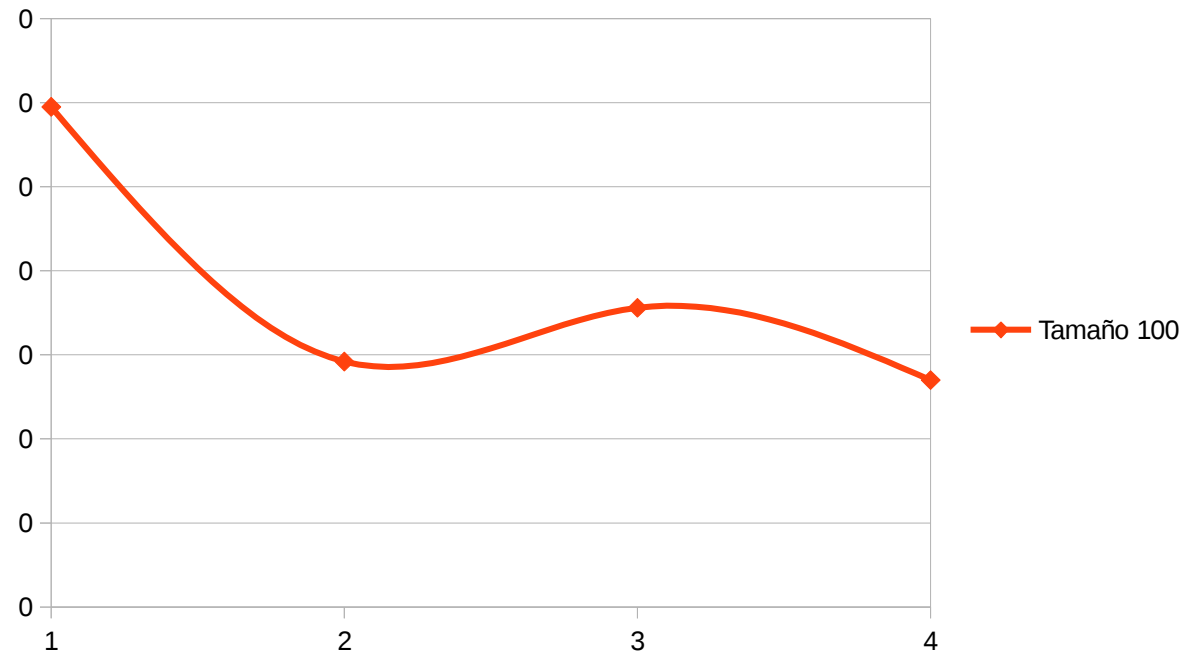


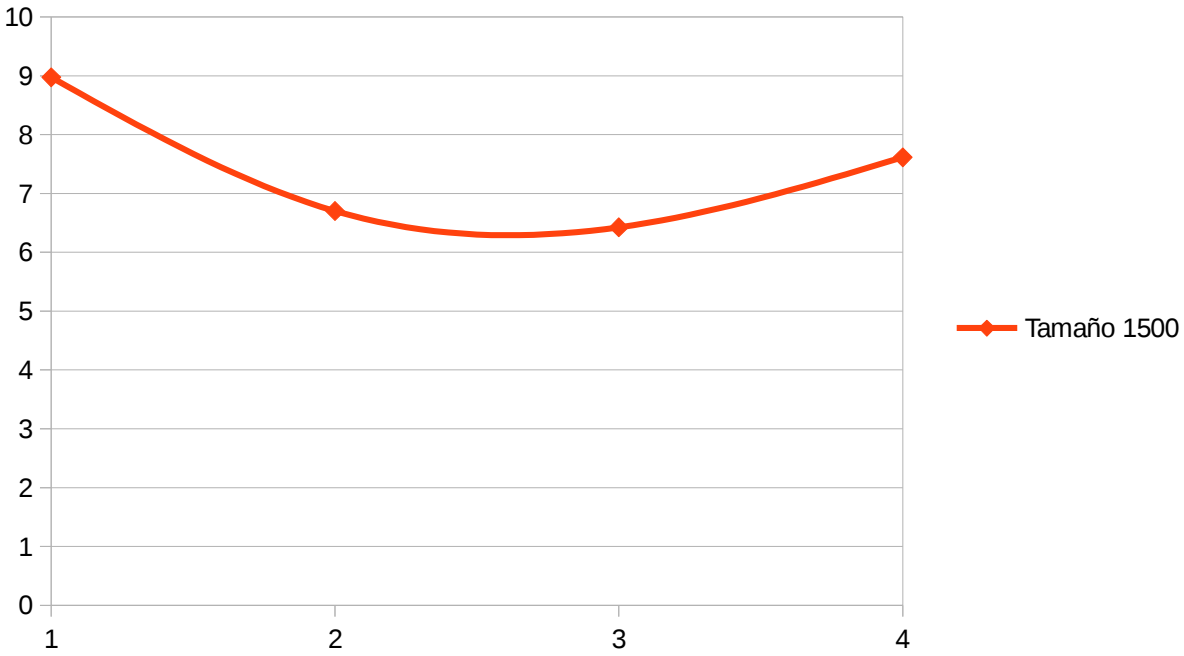
ESTUDIO DE ESCALABILIDAD EN PCLOCAL:

SCRIPT: pmm-OpenMP_pclocal.sh

```
    #!/bin/bash
49.
50. echo "1 thread"
51. export OMP_NUM_THREADS=1
52.
53. ./pmm-OpenMP 100
54. ./pmm-OpenMP 1500
55. echo "2 thread"
56. export OMP_NUM_THREADS=2
57.
58. ./pmm-OpenMP 100
59. ./pmm-OpenMP 1500
60. echo "3 thread"
61. export OMP_NUM_THREADS=3
62.
63. ./pmm-OpenMP 100
64. ./pmm-OpenMP 1500
65. echo "4 thread"
66. export OMP_NUM_THREADS=4
67.
68. ./pmm-OpenMP 100
69. ./pmm-OpenMP 1500
```

	Tamaño 100	Tamaño 1500
1	0,00119	8,974274
2	0,000584	6,700963
3	0,000712	6,424371
4	0,00054	7,61347


0
0
0
0
0
0
0
1
2
3
4
Tamaño 100


10
9
8
7
6
5
4
3
2
1
0
1
2
3
4
Tamaño 1500