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): Francisco Domínguez Lorente
Grupo de prácticas: B1
Fecha de entrega: 14/05/2019
Fecha evaluación en clase: 16/05/2019

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`

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

**CAPTURAS DE PANTALLA:**

```
[FranciscoDomínguezLorente d3vcho@d3vcho-PC:~/Escritorio/Uni/AC/bp3/ejer1] 2019-
05-02 jueves
$./if-clause 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
[FranciscoDomínguezLorente d3vcho@d3vcho-PC:~/Escritorio/Uni/AC/bp3/ejer1] 2019-
05-02 jueves
$./if-clause 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
[FranciscoDomínguezLorente d3vcho@d3vcho-PC:~/Escritorio/Uni/AC/bp3/ejer1] 2019-
05-02 jueves
$./if-clause 5 3
 thread 1 suma de a[2]=2 sumalocal=2
 thread 1 suma de a[3]=3 sumalocal=5
 thread 0 suma de a[0]=0 sumalocal=0
 thread 0 suma de a[1]=1 sumalocal=1
 thread 2 suma de a[4]=4 sumalocal=4
thread master=0 imprime suma=10
```

**RESPUESTA**: Como vemos en el último ejemplo, para 5 elementos del vector y 3 hebras, se hace el reparto según corresponda. La hebra con ID 0 realiza las dos primeras sumas, la hebra con ID 1 realiza las dos siguientes y la hebra con ID 2 realiza la última.

**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 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 2 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 3 | 0 | 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 | 0 | 0 | 1 |
| 11 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 12 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 13 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 14 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 |
| 15 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 |

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

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 *chunk* iteraciones para cada hebra y son asignadas siguiendo la política round-robin. Con *dynamic*, también se dividen en *chunk* iteraciones, pero si una hebra es más rápida y termina su tarea antes, se le asignarán otras *chunk* iteraciones. Por último, en *guided*, el tamaño *chunk* indica el tamaño mínimo de 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`

```
14    if(argc < 3)     {
15       fprintf(stderr,"\nFalta iteraciones o chunk \n");
16       exit(-1);
17      }
18    n = atoi(argv[1]); if (n>200) n=200; chunk = atoi(argv[2]);
19
20    for (i=0; i<n; i++)       a[i] = i;
21
22    #pragma omp parallel
23    {
24     #pragma omp single
25     {
26       printf("dyn-var: %d\n", omp_get_dynamic());
27       printf("nthreads-var: %d\n", omp_get_max_threads());
28       printf("thread-limit-var: %d\n", omp_get_thread_limit());
29       omp_get_schedule(&kind, &modifier);
30       printf("run-sched-var:\t Kind: %d \t Modifier: %d\n", kind, modifier);
31     }
32    }
33
34    #pragma omp parallel for firstprivate(suma) \
35              lastprivate(suma) schedule(dynamic,chunk)
36    for (i=0; i<n; i++)
37    {   suma = suma + a[i];
38        printf(" thread %d suma a[%d]=%d suma=%d \n",
39                omp_get_thread_num(),i,a[i],suma);
40    }
41
42    printf("Fuera de 'parallel for' suma=%d\n",suma);
43    printf("dyn-var FUERA del Parallel: %d\n", omp_get_dynamic());
44    printf("nthreads-var FUERA del Parallel: %d\n", omp_get_max_threads());
45    printf("thread-limit-var FUERA del Parallel: %d\n", omp_get_thread_limit());
46    omp_get_schedule(&kind, &modifier);
47    printf("run-sched-var FUERA del Parallel:\t Kind: %d \t Modifier: %d\n", kind, modifier);
```

**CAPTURAS DE PANTALLA:**

```
[FranciscoDominguezLorente d3vcho@d3vcho-PC:~/Escritorio/Uni/AC/bp3/ejer3] 2019-
05-02 jueves
$./scheduled-clauseModificado 5 3
dyn-var: 0
nthreads-var: 4
thread-limit-var: 2147483647
run-sched-var:   Kind: 2         Modifier: 1
 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 1 suma a[3]=3 suma=3
 thread 1 suma a[4]=4 suma=7
Fuera de 'parallel for' suma=7
dyn-var FUERA del Parallel: 0
nthreads-var FUERA del Parallel: 4
thread-limit-var FUERA del Parallel: 2147483647
run-sched-var FUERA del Parallel:        Kind: 2         Modifier: 1
```

**RESPUESTA**: No se imprimen valores distintos en las dos regiones.

**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`

```
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("num-threads: %d\n", omp_get_num_threads());
      printf("num-procs: %d\n", omp_get_num_procs());
      printf("in-parallel: %d\n", omp_in_parallel());
    }
   }

   #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);
   }

   printf("Fuera de 'parallel for' suma=%d\n",suma);
   printf("num-threads FUERA del Parallel: %d\n", omp_get_num_threads());
   printf("num-procs FUERA del Parallel: %d\n", omp_get_num_procs());
   printf("in-parallel FUERA del Parallel: %d\n", omp_in_parallel());

```

**CAPTURAS DE PANTALLA:**

```
[FranciscoDominguezLorente d3vcho@d3vcho-PC:~/Escritorio/Uni/AC/bp3/ejer4] 2019-
05-02 jueves
$./scheduled-clauseModificado4 5 3
num-threads: 4
num-procs: 4
in-parallel: 1
 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 2 suma a[3]=3 suma=3
 thread 2 suma a[4]=4 suma=7
Fuera de 'parallel for' suma=7
num-threads FUERA del Parallel: 1
num-procs FUERA del Parallel: 4
in-parallel FUERA del Parallel: 0
```

**RESPUESTA**: Se obtienen distintos valores para las funciones *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.

**CAPTURA CÓDIGO FUENTE**: `scheduled-clauseModificado5.c`

```c
int main(int argc, char **argv) {
  int i, n=200,chunk,a[n],suma=0;
  omp_sched_t * kind;
  int * modifier;

  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:\n\n");
     printf("dyn-var: %d\n", omp_get_dynamic());
     omp_set_dynamic(1);
     printf("dyn-var modificado con omp_set_dynamic(1): %d\n", omp_get_dynamic());

     printf("nthreads-var: %d\n", omp_get_max_threads());
     omp_set_num_threads(8);
     printf("nthreads-var modificado con omp_set_num_threads(8): %d\n", omp_get_max_threads());

     omp_get_schedule(&kind, &modifier);
     printf("run-sched-var:\t Kind: %d \t Modifier: %d\n", kind, modifier);
     omp_set_schedule(2,2);
     omp_get_schedule(&kind, &modifier);
     printf("run-sched-var modificado con omp_set_schedule(2,2);:\t Kind: %d \t Modifier: %d\n", kind, modifier);
   }
  }
```

```c
    #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);
    }

    printf("Fuera de 'parallel for' suma=%d\n",suma);
    printf("dyn-var: %d\n", omp_get_dynamic());
    printf("nthreads-var: %d\n", omp_get_max_threads());
    omp_get_schedule(&kind, &modifier);
    printf("run-sched-var:\t Kind: %d \t Modifier: %d\n", kind, modifier);

    return(0);
}
```

**CAPTURAS DE PANTALLA:**

```
[FranciscoDominguezLorente d3vcho@d3vcho-PC:~/Escritorio/Uni/AC/bp3/ejer5] 2019-
05-13 lunes
$./scheduled-clauseModificado5 4 3
Dentro de Parallel:

dyn-var: 0
dyn-var modificado con omp_set_dynamic(1): 1
nthreads-var: 4
nthreads-var modificado con omp_set_num_threads(8): 8
run-sched-var:   Kind: 2         Modifier: 1
run-sched-var modificado con omp_set_schedule(2,2);:     Kind: 2         Modifie
r: 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 2 suma a[3]=3 suma=3
Fuera de 'parallel for' suma=3
dyn-var: 0
nthreads-var: 4
run-sched-var:   Kind: 2         Modifier: 1
[FranciscoDominguezLorente d3vcho@d3vcho-PC:~/Escritorio/Uni/AC/bp3/ejer5] 2019-
05-13 lunes
```

**RESPUESTA**:

## 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 <omp.h>

int main(int argc, char **argv)  {
   int i, j;
   double t1, total;

   if(argc < 2){
      fprintf(stderr,"Falta tamaño de la matriz (debe de ser cuadrada)\n");
      exit(-1);
    }

    // Tamaño de la matriz pasado por parámetro
    unsigned int tam = atoi(argv[1]);

    if(tam < 2){
        fprintf(stderr, "El tamaño no puede ser menor a 2");
        exit(-1);
    }

    // Declaramos con memoria dinámica
    double *v1, *v2, **m;

    v1 = (double*) malloc(tam*sizeof(double));
    v2 = (double*) malloc(tam*sizeof(double));
    m = (double**) malloc(tam*sizeof(double*));


    // Reservamos memoria ahora para las componentes de la matriz
    for(i=0; i<tam; i++){
        m[i] = (double*) malloc(tam*sizeof(double));
    }

```

```c
    // Inicializamos la matriz y los vectores
    for(i=0; i<tam; i++){
        v1[i] = 2;
    }

    for(i=0; i<tam; i++){
        for(j=0; j<tam; j++){
            // Si es superior, inicializamos a 1
            if(i <= j){
                m[i][j] = 1;
            }

            // En caso contrario, inicializamos a 0
            else{
                m[i][j] = 0;
            }
        }
    }

    // Inicializamos la primera variable de tiempo
    t1 = omp_get_wtime();

    // Calculamos el producto
    for(i=0; i<tam; i++){
        double producto_local = 0;

        for(j=0; j<tam; j++){
            if(i <= j){
                producto_local = producto_local + (m[i][j]*v1[j]);
            }
        }

        v2[i] = producto_local;
    }

```

```c
    // Obtenemos el tiempo total transcurrido
    total = omp_get_wtime() - t1;

    // Para tamaños pequeños (hasta tam=11), imprimimos todas las componentes del vector resultante
    // y el tiempo de ejecución
    if(tam <= 11){
        printf("Tamaño vectores: %i\n Tiempo de ejecución: %f\n", tam, total);

        for(i=0; i<tam; i++){
            printf("v2[%i] = %f\n", i, v2[i]);
        }
    }

    // Para tamaños superiores, imprimimos el tiempo de ejecución y la primera y última componente del vector
    else{
        printf("Tamaño vectores: %i\n Tiempo de ejecución: %f\n Primera componente: %f\n Última componente: %f\n",
        tam, total, v2[0], v2[tam-1]);
    }

    // Liberamos memoria
    free(v1);
    free(v2);

    for(i=0; i<tam; i++){
        free(m[i]);
    }

    free(m);

    return 0;
}
```

**CAPTURAS DE PANTALLA:**

```
[FranciscoDominguezLorente d3vcho@d3vcho-PC:~/Escritorio/Uni/AC/bp3/ejer6] 2019-
05-13 lunes
$./pmvt-secuencial 3
Tamaño vectores: 3
 Tiempo de ejecución: 0.000001
v2[0] = 6.000000
v2[1] = 4.000000
v2[2] = 2.000000
```

```
[FranciscoDominguezLorente d3vcho@d3vcho-PC:~/Escritorio/Uni/AC/bp3/ejer6] 2019-
05-13 lunes
$./pmvt-secuencial 3000
Tamaño vectores: 3000
 Tiempo de ejecución: 0.012096
 Primera componente: 6000.000000
 Última componente: 2.000000
```

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) Para *static:* Kind = 1 y Modifier = 0
Para *dynamic:* Kind = 2 y Modifier = 1
Para *guided:* Kind = 3 y Modifier = 1

Los he obtenido llamando a la función *omp_get_schedule()*, antes de realizar la multiplicación, dentro de una directiva *single*.

b) Para static, se debería realizar un reparto equitativo de las operaciones a realizar entre todas las hebras disponibles.

c) Si una hebra es más rápida que otra, debería realizar mayor número de operaciones que las demás para el caso de *dynamic* y *guided*.

## CAPTURA CÓDIGO FUENTE: `pmtv-OpenMP.c`

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

int main(int argc, char **argv)  {
   int i, j;
   double t1, total;
   int modifier;
   omp_sched_t kind;

   if(argc < 2){
      fprintf(stderr,"Falta tamaño de la matriz (debe de ser cuadrada)\n");
      exit(-1);
   }

   // Tamaño de la matriz pasado por parámetro
   unsigned int tam = atoi(argv[1]);

   if(tam < 2){
       fprintf(stderr, "El tamaño no puede ser menor a 2");
       exit(-1);
   }

   // Declaramos con memoria dinámica
   double *v1, *v2, **m;

   v1 = (double*) malloc(tam*sizeof(double));
   v2 = (double*) malloc(tam*sizeof(double));
   m = (double**) malloc(tam*sizeof(double*));


   // Reservamos memoria ahora para las componentes de la matriz
   #pragma omp parallel for
   for(i=0; i<tam; i++)
       m[i] = (double*) malloc(tam*sizeof(double));
```

```c
#pragma omp parallel shared(m, v1, v2) private(i, j)
{
    // Inicializamos la matriz y los vectores
    #pragma omp for
    for(i=0; i<tam; i++)
        v1[i] = 2;

    #pragma omp for private(j) // Para paralelizar el for de dentro
    for(i=0; i<tam; i++){
        for(j=0; j<tam; j++){
            // Si es superior, inicializamos a 1
            if(i <= j){
                m[i][j] = 1;
            }

            // En caso contrario, inicializamos a 0
            else{
                m[i][j] = 0;
            }
        }
    }

    // Inicializamos la primera variable de tiempo
    #pragma omp single
    {
        omp_get_schedule(&kind, &modifier);
        printf("run-sched-var:\t Kind: %d \t Modifier: %d\n", kind, modifier);
        t1 = omp_get_wtime();
    }

    // Calculamos el producto
    #pragma omp for schedule (runtime)
    for(i=0; i<tam; i++){
        v2[i] = 0;
```

```c
        for(j=i; j<tam; j++){
            v2[i] = v2[i] + (m[i][j]*v1[j]);
        }
    }
}

    // Obtenemos el tiempo total transcurrido
    #pragma omp single
    {
        total = omp_get_wtime() - t1;
    }

    // Para tamaños pequeños (hasta tam=11), imprimimos todas las componentes del vector resultante
    // y el tiempo de ejecución
    if(tam <= 11){
        printf("Tamaño vectores: %i\n Tiempo de ejecución: %f\n", tam, total);

        for(i=0; i<tam; i++){
            printf("v2[%i] = %f\n", i, v2[i]);
        }
    }

    // Para tamaños superiores, imprimimos el tiempo de ejecución y la primera y última componente del vector
    else{
        printf("Tamaño vectores: %i\n Tiempo de ejecución: %f\n Primera componente: %f\n Última componente: %f\n",
        tam, total, v2[0], v2[tam-1]);
    }

    // Liberamos memoria
    free(v1);
    free(v2);

    for(i=0; i<tam; i++){
        free(m[i]);
    }

    free(m);
```

## DESCOMPOSICIÓN DE DOMINIO:

## CAPTURAS DE PANTALLA:

```
[B1estudiante9@atcgrid ejer7]$ cat script-atc.sh.o20971
Id. usuario del trabajo: B1estudiante9
Id. del trabajo: 20971.atcgrid
Nombre  del trabajo especificado por usuario: script-atc.sh
Nodo que ejecuta qsub: atcgrid
Directorio en el que se ha ejecutado qsub: /home/B1estudiante9/bp3/eje
r7
Cola: ac
Nodos asignados al trabajo:
atcgrid3
run-sched-var:   Kind: 1         Modifier: 0
Tamaño vectores: 15360
 Tiempo de ejecución: 0.049862
 Primera componente: 30720.000000
 Última componente: 2.000000
run-sched-var:   Kind: 1         Modifier: 1
Tamaño vectores: 15360
 Tiempo de ejecución: 0.049015
 Primera componente: 30720.000000
 Última componente: 2.000000
run-sched-var:   Kind: 1         Modifier: 64
Tamaño vectores: 15360
 Tiempo de ejecución: 0.049431
 Primera componente: 30720.000000
 Última componente: 2.000000
run-sched-var:   Kind: 2         Modifier: 1
Tamaño vectores: 15360
```

## TABLA RESULTADOS, SCRIPT Y GRÁFICA atcgrid

**SCRIPT:** pmvt-OpenMP_atcgrid.sh

```
#!/bin/bash
export OMP_DYNAMIC=FALSE
export OMP_NUM_THREADS=12
#Todos los scripts que se hagan para atcgrid deben incluir lo siguiente:
#Se asigna al trabajo el nombre SumaVectoresC_vlocales
#PBS -N pmtv-OpenMP
#Se asigna al trabajo la cola ac
#PBS -q ac
#Se imprime información del trabajo usando variables de entorno de PBS

echo "Id. usuario del trabajo: $PBS_O_LOGNAME"
echo "Id. del trabajo: $PBS_JOBID"
echo "Nombre  del trabajo especificado por usuario: $PBS_JOBNAME"
echo "Nodo que ejecuta qsub: $PBS_O_HOST"
echo "Directorio en el que se ha ejecutado qsub: $PBS_O_WORKDIR"
echo "Cola: $PBS_QUEUE"
echo "Nodos asignados al trabajo:"
cat $PBS_NODEFILE

# FIN del trozo que deben incluir todos los scripts

export OMP_SCHEDULE="static"
$PBS_O_WORKDIR/pmtv-OpenMP 15360
export OMP_SCHEDULE="static, 1"
$PBS_O_WORKDIR/pmtv-OpenMP 15360
export OMP_SCHEDULE="static, 64"
$PBS_O_WORKDIR/pmtv-OpenMP 15360

export OMP_SCHEDULE="dynamic"
$PBS_O_WORKDIR/pmtv-OpenMP 15360
export OMP_SCHEDULE="dynamic, 1"
$PBS_O_WORKDIR/pmtv-OpenMP 15360
export OMP_SCHEDULE="dynamic, 64"
$PBS_O_WORKDIR/pmtv-OpenMP 15360
```

```
export OMP_SCHEDULE="guided"
$PBS_O_WORKDIR/pmtv-OpenMP 15360
export OMP_SCHEDULE="guided, 1"
$PBS_O_WORKDIR/pmtv-OpenMP 15360
export OMP_SCHEDULE="guided, 64"
$PBS_O_WORKDIR/pmtv-OpenMP 15360
```

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

| Chunk | Static | Dynamic | Guided |
|---|---|---|---|
| **por defecto** | 0,049862 | 0,059560 | 0,051381 |
| **1** | 0,049015 | 0,058375 | 0,052065 |
| **64** | 0,049431 | 0,049618 | 0,056765 |
| **Chunk** | **Static** | **Dynamic** | **Guided** |
| **por defecto** | 0,044466 | 0,058623 | 0,056409 |
| **1** | 0,050880 | 0,058982 | 0,046681 |
| **64** | 0,049765 | 0,052509 | 0,057992 |

**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`

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

int main(int argc, char **argv)  {
   int i, j, k;
   double t1, total;

   if(argc < 2){
      fprintf(stderr,"Falta tamaño de la matriz (debe de ser cuadrada)\n");
      exit(-1);
    }

    // Tamaño de la matriz pasado por parámetro
    unsigned int tam = atoi(argv[1]);

    if(tam < 2){
        fprintf(stderr, "El tamaño no puede ser menor a 2");
        exit(-1);
    }

    // Declaramos con memoria dinámica
    double **m1, **m2, **m3;

    m1 = (double**) malloc(tam*sizeof(double*));
    m2 = (double**) malloc(tam*sizeof(double*));
    m3 = (double**) malloc(tam*sizeof(double*));


    // Reservamos memoria ahora para las componentes de las matrices
    for(i=0; i<tam; i++){
        m1[i] = (double*) malloc(tam*sizeof(double));
        m2[i] = (double*) malloc(tam*sizeof(double));
        m3[i] = (double*) malloc(tam*sizeof(double));
    }
```

```
    // Inicializamos la matriz y los vectores
    for(i=0; i<tam; i++){
        for(j=0; j<tam; j++){
            m1[i][j] = 0;
            m2[i][j] = 2;
            m3[i][j] = 2;
        }
    }

    // Inicializamos la primera variable de tiempo
    t1 = omp_get_wtime();

    // Calculamos el producto m1 = m2*m3
    for(i=0; i<tam; i++){
        for(j=0; j<tam; j++){
            for(k=0; k<tam; k++){
                m1[i][j] += (m2[i][k]*m3[k][j]);
            }
        }
    }

    // Obtenemos el tiempo total transcurrido
    total = omp_get_wtime() - t1;

    // Para tamaños pequeños (hasta tam=11), imprimimos todas las componentes del vector resultante
    // y el tiempo de ejecución
    if(tam <= 11){
        printf("Tamaño matriz: %i\n Tiempo de ejecución: %f\n", tam, total);

        for(i=0; i<tam; i++){
            for(j=0; j<tam; j++){
                printf("m1[%i][%i] = %f\n", i, j, m1[i][j]);
            }
        }
    }
```

```
    // Para tamaños superiores, imprimimos el tiempo de ejecución y la primera y última componente del vector
    else{
        printf("Tamaño vectores: %i\n Tiempo de ejecución: %f\n Primera componente: %f\n Última componente: %f\n",
        tam, total, m1[0][0], m1[tam-1][tam-1]);
    }

    // Liberamos memoria
    for(i=0; i<tam; i++){
        free(m1[i]);
        free(m2[i]);
        free(m3[i]);
    }

    free(m1);
    free(m2);
    free(m3);

    return 0;
}
```

## CAPTURAS DE PANTALLA:

```
[FranciscoDomínguezLorente d3vcho@d3vcho-PC:~/Escritorio/Uni/AC/bp3/ejer8] 2019-
05-14 martes
$./pmm-secuencial 5
Tamaño matriz: 5
 Tiempo de ejecución: 0.000001
m1[0][0] = 20.000000
m1[0][1] = 20.000000
m1[0][2] = 20.000000
m1[0][3] = 20.000000
m1[0][4] = 20.000000
m1[1][0] = 20.000000
m1[1][1] = 20.000000
m1[1][2] = 20.000000
m1[1][3] = 20.000000
m1[1][4] = 20.000000
m1[2][0] = 20.000000
m1[2][1] = 20.000000
m1[2][2] = 20.000000
m1[2][3] = 20.000000
m1[2][4] = 20.000000
m1[3][0] = 20.000000
m1[3][1] = 20.000000
m1[3][2] = 20.000000
```

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).

10.

**DESCOMPOSICIÓN DE DOMINIO:**

**CAPTURA CÓDIGO FUENTE**: `pmm-OpenMP.c`

```
5  int main(int argc, char **argv)  {
6     int i, j, k;
7     double t1, total, suma;
8
9     if(argc < 2){
10       fprintf(stderr,"Falta tamaño de la matriz (debe de ser cuadrada)\n");
11       exit(-1);
12    }
13
14    // Tamaño de la matriz pasado por parámetro
15    unsigned int tam = atoi(argv[1]);
16
17    if(tam < 2){
18        fprintf(stderr, "El tamaño no puede ser menor a 2");
19        exit(-1);
20    }
21
22    // Declaramos con memoria dinámica
23    double **m1, **m2, **m3;
24
25    m1 = (double**) malloc(tam*sizeof(double*));
26    m2 = (double**) malloc(tam*sizeof(double*));
27    m3 = (double**) malloc(tam*sizeof(double*));
28
29
30    // Reservamos memoria ahora para las componentes de la matriz
31    #pragma omp parallel for
32    for(i=0; i<tam; i++){
33        m1[i] = (double*) malloc(tam*sizeof(double));
34        m2[i] = (double*) malloc(tam*sizeof(double));
35        m3[i] = (double*) malloc(tam*sizeof(double));
36    }
```

```
38  #pragma omp parallel shared(m1, m2, m3) private(i, j, k)
39  {
40      // Inicializamos la matriz y los vectores
41      #pragma omp for schedule(runtime)
42      for(i=0; i<tam; i++){
43          for(j=0; j<tam; j++){
44              m1[i][j] = 0;
45          }
46      }
47
48      #pragma omp for schedule(runtime)
49      for(i=0; i<tam; i++){
50          for(j=0; j<tam; j++){
51              m2[i][j] = 2;
52          }
53      }
54
55      #pragma omp for schedule(runtime)
56      for(i=0; i<tam; i++){
57          for(j=0; j<tam; j++){
58              m3[i][j] = 2;
59          }
60      }
61
62      // Inicializamos la primera variable de tiempo
63      #pragma omp single
64      {
65          t1 = omp_get_wtime();
66      }
```

```
68      // Calculamos el producto m1 = m2*m3
69      #pragma omp for schedule (runtime)
70      for(i=0; i<tam; i++){
71          for(j=0; j<tam; j++){
72              for(k=0; k<tam; k++){
73                  m1[i][j] += (m2[i][k]*m3[k][j]);
74              }
75          }
76      }
77  }
78
79      // Obtenemos el tiempo total transcurrido
80      #pragma omp single
81      {
82          total = omp_get_wtime() - t1;
83      }
84
85      // Para tamaños pequeños (hasta tam=11), imprimimos todas las componentes del vector resultante
86      // y el tiempo de ejecución
87      if(tam <= 11){
88          printf("Tamaño matriz: %i\n Tiempo de ejecución: %f\n", tam, total);
89
90          for(i=0; i<tam; i++){
91              for(j=0; j<tam; j++){
92                  printf("m1[%i][%i] = %f\n", i, j, m1[i][j]);
93              }
94          }
95      }
96
97      // Para tamaños superiores, imprimimos el tiempo de ejecución y la primera y última componente del vector
98      else{
99          printf("Tamaño vectores: %i\n Tiempo de ejecución: %f\n Primera componente: %f\n Última componente: %f\n",
100         tam, total, m1[0][0], m1[tam-1][tam-1]);
101     }
102
```

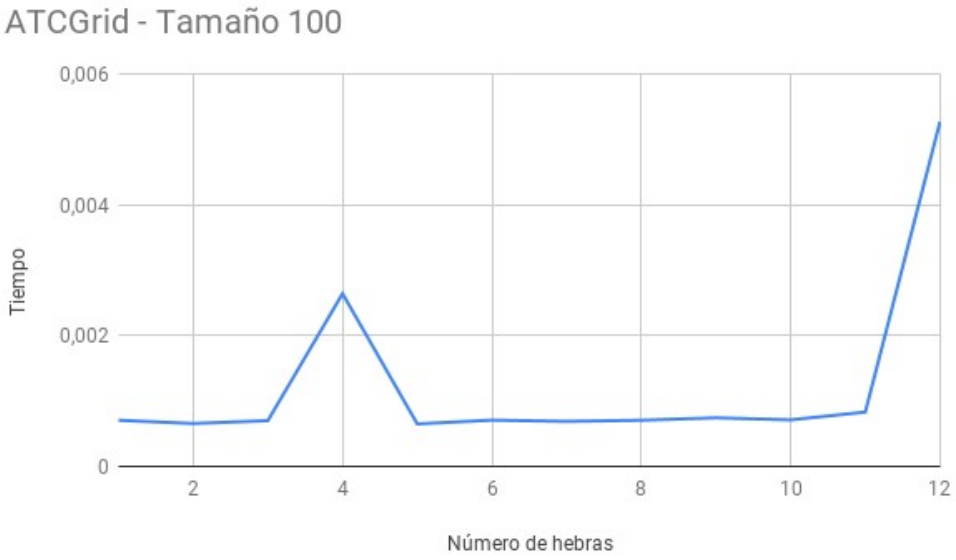
**CAPTURAS DE PANTALLA:**

```
[FranciscoDominguezLorente d3vcho@d3vcho-PC:~/Escritorio/Uni/AC/bp3/ejer9] 2019-
05-14 martes
$./pmm-OpenMP 5
Tamaño matriz: 5
 Tiempo de ejecución: 0.000009
m1[0][0] = 20.000000
m1[0][1] = 20.000000
m1[0][2] = 20.000000
m1[0][3] = 20.000000
m1[0][4] = 20.000000
m1[1][0] = 20.000000
m1[1][1] = 20.000000
m1[1][2] = 20.000000
m1[1][3] = 20.000000
m1[1][4] = 20.000000
m1[2][0] = 20.000000
m1[2][1] = 20.000000
m1[2][2] = 20.000000
m1[2][3] = 20.000000
m1[2][4] = 20.000000
m1[3][0] = 20.000000
m1[3][1] = 20.000000
m1[3][2] = 20.000000
```

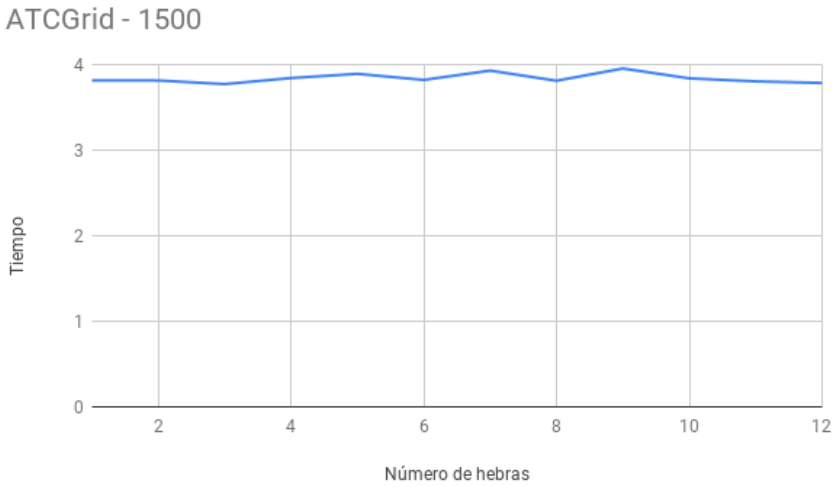
10. Hacer un estudio de escalabilidad (ganancia en velocidad en función del número de cores) en atcgrid y en su PC del código paralelo implementado para dos tamaños de las matrices. Debe recordar usar `-O2` al compilar. El número de núcleos máximo en este estudio debe ser el igual al de núcleos físicos 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:**

| Número de hebras | Tamaño | Tiempo |
|---|---|---|
| 1 | 100 | 0,000706 |
| 2 | 100 | 0,000656 |
| 3 | 100 | 0,000700 |
| 4 | 100 | 0,002641 |
| 5 | 100 | 0,000650 |
| 6 | 100 | 0,000708 |
| 7 | 100 | 0,000689 |
| 8 | 100 | 0,000706 |
| 9 | 100 | 0,000743 |
| 10 | 100 | 0,000712 |
| 11 | 100 | 0,000831 |
| 12 | 100 | 0,005269 |



| Número de hebras | Tamaño | Tiempo |
|---|---|---|
| 1 | 1500 | 3,815572 |
| 2 | 1500 | 3,814225 |
| 3 | 1500 | 3,772068 |
| 4 | 1500 | 3,842819 |
| 5 | 1500 | 3,892057 |
| 6 | 1500 | 3,821171 |
| 7 | 1500 | 3,929852 |
| 8 | 1500 | 3,811524 |
| 9 | 1500 | 3,954340 |
| 10 | 1500 | 3,839285 |
| 11 | 1500 | 3,803886 |
| 12 | 1500 | 3,785653 |

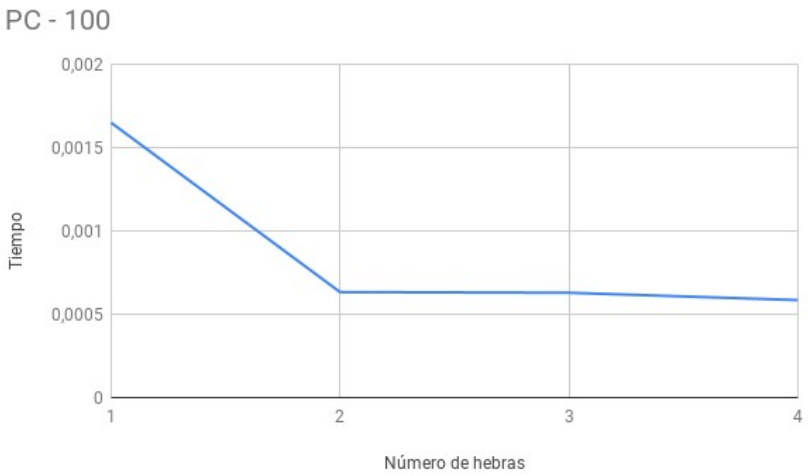


**SCRIPT:** pmm-OpenMP_atcgrid.sh

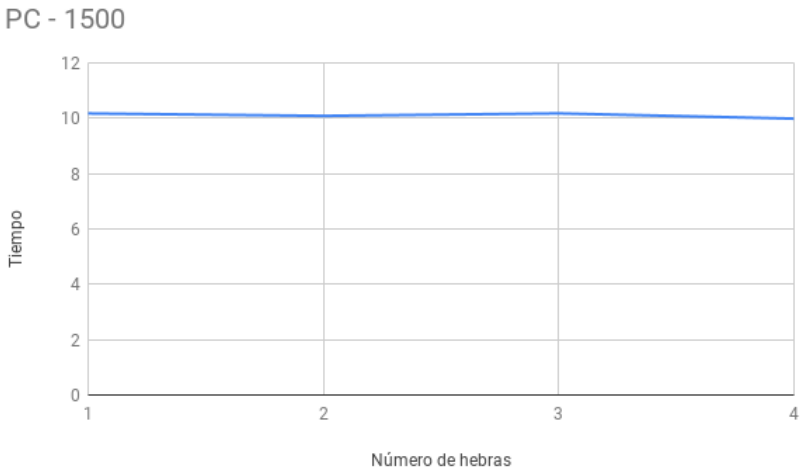
```
#!/bin/bash
export OMP_DYNAMIC=FALSE
#Todos los scripts que se hagan para atcgrid deben incluir lo siguiente:
#Se asigna al trabajo el nombre SumaVectoresC_vlocales
#PBS -N pmm-OpenMP
#Se asigna al trabajo la cola ac
#PBS -q ac
#Se imprime información del trabajo usando variables de entorno de PBS

echo "Id. usuario del trabajo: $PBS_O_LOGNAME"
echo "Id. del trabajo: $PBS_JOBID"
echo "Nombre  del trabajo especificado por usuario: $PBS_JOBNAME"
echo "Nodo que ejecuta qsub: $PBS_O_HOST"
echo "Directorio en el que se ha ejecutado qsub: $PBS_O_WORKDIR"
echo "Cola: $PBS_QUEUE"
echo "Nodos asignados al trabajo:"
cat $PBS_NODEFILE

# FIN del trozo que deben incluir todos los scripts

echo "Tamaño 100"
for ((N=1;N<13;N++))
do
    export OMP_NUM_THREADS=N
    $PBS_O_WORKDIR/pmm-OpenMP 100
done

echo "Tamaño 1500"
for ((N=1;N<13;N++))
do
    export OMP_NUM_THREADS=N
    $PBS_O_WORKDIR/pmm-OpenMP 1500
done
```

**ESTUDIO DE ESCALABILIDAD EN PCLOCAL:**

| Número de hebras | Tamaño | Tiempo |
|---|---|---|
| 1 | 100 | 0,001649 |
| 2 | 100 | 0,000633 |
| 3 | 100 | 0,000629 |
| 4 | 100 | 0,000585 |



| Número de hebras | Tamaño | Tiempo |
|---|---|---|
| 1 | 1500 | 10,173245 |
| 2 | 1500 | 10,085856 |
| 3 | 1500 | 10,175469 |
| 4 | 1500 | 9,986479 |




**SCRIPT:** pmm-OpenMP_pclocal.sh

```
#!/bin/bash
export OMP_DYNAMIC=FALSE

echo "Tamaño 100"
for ((N=1;N<5;N++))
do
    export OMP_NUM_THREADS=N
    ./pmm-OpenMP 100
done

echo "Tamaño 1500"
for ((N=1;N<5;N++))
do
    export OMP_NUM_THREADS=N
    ./pmm-OpenMP 1500
done
```