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): David Gómez Hernández
Grupo de prácticas: C3
Fecha de entrega:
Fecha evaluación en clase:

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`

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.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 o Num_threads\n");
      exit(-1);
     }
   n = atoi(argv[1]); if (n>20) n=20;
   for (i=0; i<n; i++) {
      a[i] = i;
   }

   //Cogemos el valor de dicha x
   x = atoi(argv[2]);

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

CAPTURAS DE PANTALLA:

```
[DavidGómezHernández david@david-HP-Pavilion-Notebook:/home/david/Escritorio/Carrera/AC/b
p3/ejer1] 04/21/20 10:44:09
$ export OMP_NUM_THREADS=4
[DavidGómezHernández david@david-HP-Pavilion-Notebook:/home/david/Escritorio/Carrera/AC/b
p3/ejer1] 04/21/20 10:44:11
$ gcc -O2 -fopenmp if-clauseModificado.c -o if-clauseModificado
[DavidGómezHernández david@david-HP-Pavilion-Notebook:/home/david/Escritorio/Carrera/AC/b
p3/ejer1] 04/21/20 10:44:13
$ ./if-clauseModificado 3 2
 thread 0 suma de a[0]=0 sumalocal=0
 thread 0 suma de a[1]=1 sumalocal=1
 thread 0 suma de a[2]=2 sumalocal=3
thread master=0 imprime suma=3
[DavidGómezHernández david@david-HP-Pavilion-Notebook:/home/david/Escritorio/Carrera/AC/b
p3/ejer1] 04/21/20 10:44:22
$ ./if-clauseModificado 4 3
 thread 0 suma de a[0]=0 sumalocal=0
 thread 0 suma de a[1]=1 sumalocal=1
 thread 0 suma de a[2]=2 sumalocal=3
 thread 0 suma de a[3]=3 sumalocal=6
thread master=0 imprime suma=6
[DavidGómezHernández david@david-HP-Pavilion-Notebook:/home/david/Escritorio/Carrera/AC/b
p3/ejer1] 04/21/20 10:44:28
$ ./if-clauseModificado 5 4
 thread 0 suma de a[0]=0 sumalocal=0
 thread 0 suma de a[1]=1 sumalocal=1
 thread 1 suma de a[2]=2 sumalocal=2
 thread 2 suma de a[3]=3 sumalocal=3
 thread 3 suma de a[4]=4 sumalocal=4
thread master=0 imprime suma=10
```

RESPUESTA:
Vemos que si no hay más de 4 iteraciones solo trabaja la hebra 0, ya que no hay paralelización. En el último ejemplo vemos que al haber más de 4 iteraciones y habiendo 4 hebras según le hemos especificado, vemos que el reparto se hace siguiendo la cláusula schedule.static (que veremos más adelante).

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

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

RESPUESTA:
Las iteraciones se dividen en unidades de chunk.
Static: Las unidades se asignan en round-robin. Se asigna un único chunk a cada thread (comportamiento usual por defecto).

Dynamic: Las unidades se asignan en tiempo de ejecución. Los threads más rápidos ejecutan más unidades. Si no se especifica el chunk, se usan unidades de una iteración. Es apropiado si se desconoce el tiempo de ejecución de las iteraciones, además de que añade sobrecarga adicional.

Guided: Las unidades se asignan en tiempo de ejecución. Es apropiado si se desconoce el tiempo de ejecución de las iteraciones o su número. Comienza con bloque largo. El tamaño del bloque va menguando (n.º de iteraciones que restan dividido por n.º threads), no más pequeño que un chunk (excepto la última). Añade sobrecarga extra, pero menos que dynamic para el mismo chunk.

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

CAPTURA CÓDIGO FUENTE: `scheduled-clauseModificado.c`

```
 9  int main(int argc, char **argv) {
10    int i, n=200,chunk,a[n],suma=0, *modifier;
11    omp_sched_t *kind;
12
13
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("DENTRO DEL PARALLEL\n");
27       printf("DIN-VAR: %d\n", omp_get_dynamic());
28       printf("NTHREADS-VAR: %d\n", omp_get_max_threads());
29       printf("THREAD-LIMIT-VAR: %d\n", omp_get_thread_limit());
30       omp_get_schedule(&kind, &modifier);
31       printf("RUN-SCHED-VAR:  Kind: %d    Modifier: %d\n", kind, modifier);
32     }
33    }
34
35    #pragma omp parallel for firstprivate(suma) \
36                lastprivate(suma) schedule(dynamic,chunk)
37    for (i=0; i<n; i++)
38    {   suma = suma + a[i];
39        printf(" thread %d suma a[%d]=%d suma=%d \n",
40                omp_get_thread_num(),i,a[i],suma);
41    }
42    printf("FUERA DEL PARALLEL\n");
43    printf("RESULTADO SUMA: %d\n", suma);
44    printf("DIN-VAR: %d\n", omp_get_dynamic());
45    printf("NTHREADS-VAR: %d\n", omp_get_max_threads());
46    printf("THREAD-LIMIT-VAR: %d\n", omp_get_thread_limit());
47    omp_get_schedule(&kind, &modifier);
48    printf("RUN-SCHED-VAR:  Kind: %d    Modifier: %d\n", kind, modifier);
49  }
```

CAPTURAS DE PANTALLA:

```
[DavidGómezHernández david@david-HP-Pavilion-Notebook:/home/david/Escritorio/Carrera/A
C/bp3/ejer3] 04/26/20 16:25:38
$ ./scheduled-clauseModificado 5 3
DENTRO DEL PARALLEL
DIN-VAR: 0
NTHREADS-VAR: 8
THREAD-LIMIT-VAR: 2147483647
RUN-SCHED-VAR:  Kind: 2    Modifier: 1
 thread 0 suma a[0]=0 suma=0
 thread 0 suma a[1]=1 suma=1
 thread 0 suma a[2]=2 suma=3
 thread 5 suma a[3]=3 suma=3
 thread 5 suma a[4]=4 suma=7
FUERA DEL PARALLEL
RESULTADO SUMA: 7
DIN-VAR: 0
NTHREADS-VAR: 8
THREAD-LIMIT-VAR: 2147483647
RUN-SCHED-VAR:  Kind: 2    Modifier: 1
```

RESPUESTA:
Los valores no cambian, son los mismos 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`

```
10    int i, n=200,chunk,a[n],suma=0, *modifier;
11    omp_sched_t *kind;
12
13
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("DENTRO DEL PARALLEL\n");
27       printf("DIN-VAR: %d\n", omp_get_dynamic());
28       printf("NTHREADS-VAR: %d\n", omp_get_max_threads());
29       printf("THREAD-LIMIT-VAR: %d\n", omp_get_thread_limit());
30       omp_get_schedule(&kind, &modifier);
31       printf("RUN-SCHED-VAR:  Kind: %d    Modifier: %d\n", kind, modifier);
32       printf("NUM_THREADS: %d\n", omp_get_num_threads());
33       printf("NUM_PROCS: %d\n", omp_get_num_procs());
34       printf("IN_PARALLEL: %d\n", omp_in_parallel());
35     }
36    }
37
38    #pragma omp parallel for firstprivate(suma) \
39                lastprivate(suma) schedule(dynamic,chunk)
40    for (i=0; i<n; i++)
41    {   suma = suma + a[i];
42        printf(" thread %d suma a[%d]=%d suma=%d \n",
43                omp_get_thread_num(),i,a[i],suma);
44    }
45    printf("FUERA DEL PARALLEL\n");
46    printf("RESULTADO SUMA: %d\n", suma);
47    printf("DIN-VAR: %d\n", omp_get_dynamic());
48    printf("NTHREADS-VAR: %d\n", omp_get_max_threads());
49    printf("THREAD-LIMIT-VAR: %d\n", omp_get_thread_limit());
50    omp_get_schedule(&kind, &modifier);
51    printf("RUN-SCHED-VAR:  Kind: %d    Modifier: %d\n", kind, modifier);
52    printf("NUM_THREADS: %d\n", omp_get_num_threads());
53    printf("NUM_PROCS: %d\n", omp_get_num_procs());
54    printf("IN_PARALLEL: %d\n", omp_in_parallel());
```

CAPTURAS DE PANTALLA:

```
[DavidGómezHernández david@david-HP-Pavilion-Notebook:/home/david/Escritorio/Carrera/A
C/bp3/ejer4] 04/26/20 16:27:29
$ ./scheduled-clauseModificado4 5 3
DENTRO DEL PARALLEL
DIN-VAR: 0
NTHREADS-VAR: 8
THREAD-LIMIT-VAR: 2147483647
RUN-SCHED-VAR:  Kind: 2    Modifier: 1
NUM_THREADS: 8
NUM_PROCS: 8
IN_PARALLEL: 1
 thread 6 suma a[3]=3 suma=3
 thread 6 suma a[4]=4 suma=7
 thread 7 suma a[0]=0 suma=0
 thread 7 suma a[1]=1 suma=1
 thread 7 suma a[2]=2 suma=3
FUERA DEL PARALLEL
RESULTADO SUMA: 7
DIN-VAR: 0
NTHREADS-VAR: 8
THREAD-LIMIT-VAR: 2147483647
RUN-SCHED-VAR:  Kind: 2    Modifier: 1
NUM_THREADS: 1
NUM_PROCS: 8
IN_PARALLEL: 0
```

RESPUESTA:

Para omp_get_num_threads() y omp_in_parallel() los valores son distintos.

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`

```
10  int i, n=200,chunk,a[n],suma=0, *modifier;
11  omp_sched_t *kind;
12
13
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("DENTRO DEL PARALLEL\n");
27     printf("DIN-VAR: %d\n", omp_get_dynamic());
28     printf("NTHREADS-VAR: %d\n", omp_get_max_threads());
29     printf("THREAD-LIMIT-VAR: %d\n", omp_get_thread_limit());
30     omp_get_schedule(&kind, &modifier);
31     printf("RUN-SCHED-VAR:  Kind: %d    Modifier: %d\n", kind, modifier);
32     printf("NUM_THREADS: %d\n", omp_get_num_threads());
33     printf("NUM_PROCS: %d\n", omp_get_num_procs());
34     printf("IN_PARALLEL: %d\n", omp_in_parallel());
35   }
36  }
37
38  #pragma omp parallel for firstprivate(suma) \
39              lastprivate(suma) schedule(dynamic,chunk)
40  for (i=0; i<n; i++)
41  {   suma = suma + a[i];
42      printf(" thread %d suma a[%d]=%d suma=%d \n",
43              omp_get_thread_num(),i,a[i],suma);
44  }
45  printf("FUERA DEL PARALLEL\n");
46  printf("RESULTADO SUMA: %d\n", suma);
47  printf("DIN-VAR: %d\n", omp_get_dynamic());
48  printf("NTHREADS-VAR: %d\n", omp_get_max_threads());
49  printf("THREAD-LIMIT-VAR: %d\n", omp_get_thread_limit());
50  omp_get_schedule(&kind, &modifier);
51  printf("RUN-SCHED-VAR:  Kind: %d    Modifier: %d\n", kind, modifier);
52  printf("NUM_THREADS: %d\n", omp_get_num_threads());
53  printf("NUM_PROCS: %d\n", omp_get_num_procs());
54  printf("IN_PARALLEL: %d\n", omp_in_parallel());
```

CAPTURAS DE PANTALLA:

```
[DavidGómezHernández david@david-HP-Pavilion-Notebook:/home/david/Escritorio/Carrera/A
C/bp3/ejer5] 04/26/20 16:51:58
$ ./scheduled-clauseModificado5 5 3
DENTRO DEL PARALLEL
DIN-VAR: 0
DIN-VAR MODIFICADO: 1
NTHREADS-VAR: 8
NTHREADS-VAR MODIFICADO: 4
RUN-SCHED-VAR:  Kind: 2    Modifier: 1
RUN-SCHED-VAR MODIFICADO:  Kind: 4    Modifier: 1
 thread 5 suma a[3]=3 suma=3
 thread 5 suma a[4]=4 suma=7
 thread 3 suma a[0]=0 suma=0
 thread 3 suma a[1]=1 suma=1
 thread 3 suma a[2]=2 suma=3
FUERA DEL PARALLEL
RESULTADO SUMA: 7
DIN-VAR: 0
NTHREADS-VAR: 8
RUN-SCHED-VAR:  Kind: 2    Modifier: 1
```

RESPUESTA:

Cambian cuando son modificadas.

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
int main(int argc, char **argv)  {
   double inicio, final;

   if(argc < 2){
     printf(stderr,"Error de introducción de parámetros: Falta el tamaño de la matriz (tiene que ser cuadrada)\n");
     exit(-1);
    }

    int tamanio = atoi(argv[1]);

    if(tamanio < 2){
        printf(stderr, "El tamaño tiene que ser igual o superior a 2");
        exit(-1);
    }

    double *v1, *v2, **m;

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

    for(int i=0; i<tamanio; i++){
        m[i] = (double*) malloc(tamanio*sizeof(double));
    }

    //Comprobamos que si la fila es mayor a la columna, eso quiere decir que está por debajo de la diagonal principal, con lo cual
    //lo igualamos a 0
    for(int i=0; i<tamanio; i++){
        for(int j=0; j<tamanio; j++){
            if(i > j)
                m[i][j] = 0;
            else
                m[i][j] = 1;

        }
    }

    for(int i=0; i<tamanio; i++){
        v1[i] = 1;
        v2[i] = 0;
    }
```

```c
    inicio = omp_get_wtime();

    for(int i=0; i<tamanio; i++){
        double local = 0;
        for(int j=0; j<tamanio; j++){
            local = local + (m[i][j]*v1[j]);
        }
        v2[i] = local;
    }

    final = omp_get_wtime() - inicio;

    if(tamanio <= 8){
        printf("Tamaño: %i\n Tiempo de ejecución: %f\n", tamanio, final);
        for(int i=0; i<tamanio; i++) printf("v2[%i] = %f\n", i, v2[i]);
    }

    else{
        printf("Tamaño: %i\n Tiempo de ejecución: %f\n Primer elemento: %f\n Último elemento: %f\n", tamanio, final,
         v2[0], v2[tamanio-1]);
    }

    free(v1);
    free(v2);

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

    free(m);

    return 0;
}
```

CAPTURAS DE PANTALLA:

```
[DavidGómezHernández david@david-HP-Pavilion-Notebook:/home/david/Escritorio/Carrera/A
C/bp3/ejer6] 04/26/20 17:06:48
$ ./pmtv-secuencial 8
Tamaño: 8
 Tiempo de ejecución: 0.000002
v2[0] = 8.000000
v2[1] = 7.000000
v2[2] = 6.000000
v2[3] = 5.000000
v2[4] = 4.000000
v2[5] = 3.000000
v2[6] = 2.000000
v2[7] = 1.000000
[DavidGómezHernández david@david-HP-Pavilion-Notebook:/home/david/Escritorio/Carrera/A
C/bp3/ejer6] 04/26/20 17:06:50
$ ./pmtv-secuencial 12
Tamaño: 12
 Tiempo de ejecución: 0.000001
 Primer elemento: 12.000000
 Último elemento: 1.000000
```

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) Utilizando la función omp_get_schedule() para obtener la variable de control RUN-TIME podemos sacar el kind y el modifier.
Static → Kind: 1 | Modifier: 0
Dynamic → Kind: 2 | Modifier: 1
Guided → Kind: 3 | Modifier: 1

b) Se realiza un reparto equitativo de las operaciones entre todas las hebras que no estén ocupadas con otra tarea.

c) En función de la rapidez con las que las hebras acaben las tareas, algunas realizarán más operaciones que otras.

CAPTURA CÓDIGO FUENTE: `pmtv-OpenMP.c`

```c
int main(int argc, char **argv)  {
    double inicio, final;
    int i,j;
    omp_sched_t * kind;
    int * modifier;

   if(argc < 2){
     printf(stderr,"Error de introducción de parámetros: Falta el tamaño de la matriz (tiene que ser cuadrada)\n");
     exit(-1);
    }

    int tamanio = atoi(argv[1]);

    if(tamanio < 2){
        printf(stderr, "El tamaño tiene que ser igual o superior a 2");
        exit(-1);
    }

    double *v1, *v2, **m;

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

    for(i=0; i<tamanio; i++){
        m[i] = (double*) malloc(tamanio*sizeof(double));
    }

#pragma omp parallel private(i,j)
{
    #pragma omp for private(j)
    for(i=0; i<tamanio; i++){
        for(j=0; j<tamanio; j++){
            if(i > j)
                m[i][j] = 0;
            else
                m[i][j] = 1;

        }
    }
```

```c
    #pragma omp for
    for(i=0; i<tamanio; i++){
        v1[i] = 1;
        v2[i] = 0;
    }


    #pragma omp single
    {
        omp_get_schedule(&kind, &modifier);
        printf("RUN-SCHED-VAR:  Kind: %d    Modifier: %d\n", kind, modifier);
        inicio = omp_get_wtime();
    }

    #pragma omp for schedule(runtime)
    for(i=0; i<tamanio; i++){
        double local = 0;
        for(j=0; j<tamanio; j++){
            local = local + (m[i][j]*v1[j]);
        }
        v2[i] = local;
    }
}

    final = omp_get_wtime() - inicio;


    if(tamanio <= 8){
        printf("Tamaño: %i\n Tiempo de ejecución: %f\n", tamanio, final);

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

    else{
        printf("Tamaño: %i\n Tiempo de ejecución: %f\n Primer elemento: %f\n Último elemento: %f\n", tamanio, final,
         v2[0], v2[tamanio-1]);
    }

    free(v1);
    free(v2);
```

```
87      for(int i=0; i<tamanio; i++){
88          free(m[i]);
89      }
90
91      free(m);
92
93      return 0;
94  }
```

DESCOMPOSICIÓN DE DOMINIO:

CAPTURAS DE PANTALLA:

```
Id. usuario del trabajo:
Id. del trabajo:
Nombre  del trabajo especificado por usuario:
Nodo que ejecuta qsub:
Directorio en el que se ha ejecutado qsub:
Cola:
Nodos asignados al trabajo:
STATIC DEFECTO
RUN-SCHED-VAR:  Kind: -2147483647    Modifier: 0
Tamaño: 15360
 Tiempo de ejecución: 0.317247
 Primer elemento: 15360.000000
 Último elemento: 1.000000
STATIC CHUNK 1
RUN-SCHED-VAR:  Kind: -2147483647    Modifier: 1
Tamaño: 15360
 Tiempo de ejecución: 0.321648
 Primer elemento: 15360.000000
 Último elemento: 1.000000
STATIC CHUNK 64
RUN-SCHED-VAR:  Kind: -2147483647    Modifier: 64
Tamaño: 15360
 Tiempo de ejecución: 0.315095
 Primer elemento: 15360.000000
 Último elemento: 1.000000
DYNAMIC DEFECTO
RUN-SCHED-VAR:  Kind: 2    Modifier: 1
Tamaño: 15360
 Tiempo de ejecución: 0.320374
 Primer elemento: 15360.000000
 Último elemento: 1.000000
DYNAMIC CHUNK 1
RUN-SCHED-VAR:  Kind: 2    Modifier: 1
Tamaño: 15360
 Tiempo de ejecución: 0.319451
 Primer elemento: 15360.000000
 Último elemento: 1.000000
DYNAMIC CHUNK 64
RUN-SCHED-VAR:  Kind: 2    Modifier: 64
Tamaño: 15360
 Tiempo de ejecución: 0.318113
 Primer elemento: 15360.000000
```

TABLA RESULTADOS, SCRIPT Y GRÁFICA atcgrid

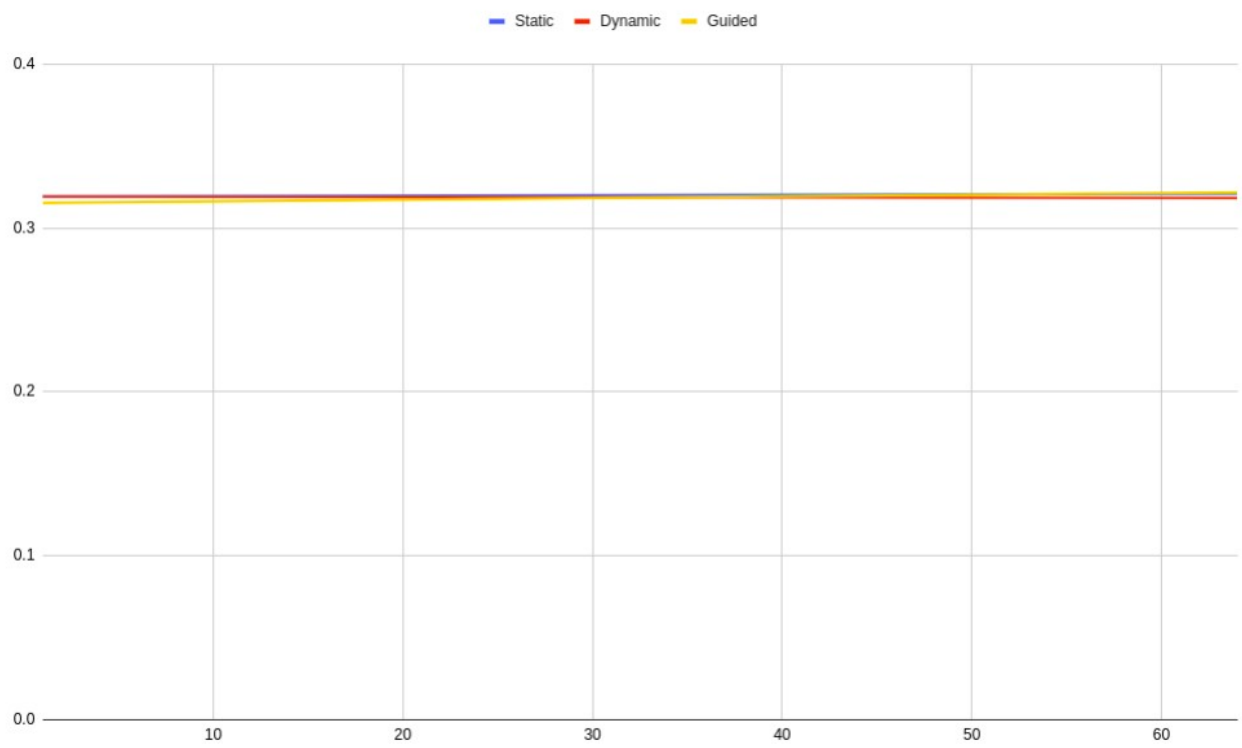
SCRIPT: pmvt-OpenMP_atcgrid.sh

```
#!/bin/bash
export OMP_DYNAMIC=FALSE
export OMP_NUM_THREADS=12
#Todos los scripts que se hagan para atcgrid deben incluir lo siguiente:
#PBS -N suma
#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"
srun -p ac ./pmtv-OpenMP 15360
export OMP_SCHEDULE="static, 1"
srun -p ac ./pmtv-OpenMP 15360
export OMP_SCHEDULE="static, 64"
srun -p ac ./pmtv-OpenMP 15360

export OMP_SCHEDULE="dynamic"
srun -p ac ./pmtv-OpenMP 15360
export OMP_SCHEDULE="dynamic, 1"
srun -p ac ./pmtv-OpenMP 15360
export OMP_SCHEDULE="dynamic, 64"
srun -p ac ./pmtv-OpenMP 15360

export OMP_SCHEDULE="guided"
srun -p ac ./pmtv-OpenMP 15360
export OMP_SCHEDULE="guided, 1"
srun -p ac ./pmtv-OpenMP 15360
export OMP_SCHEDULE="guided, 64"
srun -p ac ./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= , 12 threads

Chunk	Static	Dynamic	Guided
por defecto	0.323448	0.319136	0.318233
1	0.319162	0.317043	0.315251
64	0.320781	0.318233	0.321545

Chunk	Static	Dynamic	Guided
por defecto	0.317247	0.320374	0.318073
1	0.321648	0.319451	0.317778
64	0.315095	0.318113	0.317227



Se ve que quién tiene mejores prestaciones es la que usa dynamic.

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
int main(int argc, char **argv)  {
   double inicio, final;

   if(argc < 2){
      printf(stderr,"Error de introducción de parámetros: Falta el tamaño de las matrices (tienen que ser cuadradas)\n");
      exit(-1);
    }

    int tamanio = atoi(argv[1]);

    if(tamanio < 2){
        printf(stderr, "El tamaño tiene que ser igual o superior a 2");
        exit(-1);
    }

    double **m1, **m2, **m;

    m1 = (double**) malloc(tamanio*sizeof(double*));
    m2 = (double**) malloc(tamanio*sizeof(double*));
    m = (double**) malloc(tamanio*sizeof(double*));

    for(int i=0; i<tamanio; i++){
        m1[i] = (double*) malloc(tamanio*sizeof(double));
        m2[i] = (double*) malloc(tamanio*sizeof(double));
        m[i] = (double*) malloc(tamanio*sizeof(double));
    }

    for(int i=0; i<tamanio; i++){
        for(int j=0; j<tamanio; j++){
            m1[i][j] = 3;
            m2[i][j] = 3;
            m[i][j] = 0;
        }
    }

    inicio = omp_get_wtime();
```

```c

    for(int i=0; i<tamanio; i++){
        for(int j=0; j<tamanio; j++){
            double local = 0;
            for(int k=0;k<tamanio;k++){
                local += m1[i][j]*m2[i][j];
            }
            m[i][j] = local;
        }
    }

    final = omp_get_wtime() - inicio;

    if(tamanio <= 8){
        printf("Tamaño: %i\n Tiempo de ejecución: %f\n", tamanio, final);
        for(int i=0; i<tamanio; i++)
            for(int j=0;j<tamanio;j++)
                printf("m[%i][%i] = %f\n", i,j, m[i][j]);
    }

    else{
        printf("Tamaño: %i\n Tiempo de ejecución: %f\n Primer elemento: %f\n Último elemento: %f\n", tamanio, final,
         m[0][0], m[tamanio-1][tamanio-1]);
    }

    for(int i=0; i<tamanio; i++){
        free(m[i]);
        free(m1[i]);
        free(m2[i]);
    }

    free(m);
    free(m1);
    free(m2);

    return 0;
}
```

CAPTURAS DE PANTALLA:

```
[DavidGómezHernández david@david-HP-Pavilion-Notebook:/home/david/Escritorio/Carrera/AC/bp3/ejer8] 04/2
7/20 11:02:03
$ ./pmm-secuencial 3
Tamaño: 3
 Tiempo de ejecución: 0.000001
m[0][0] = 27.000000
m[0][1] = 27.000000
m[0][2] = 27.000000
m[1][0] = 27.000000
m[1][1] = 27.000000
m[1][2] = 27.000000
m[2][0] = 27.000000
m[2][1] = 27.000000
m[2][2] = 27.000000
```

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).

DESCOMPOSICIÓN DE DOMINIO:

CAPTURA CÓDIGO FUENTE: `pmm-OpenMP.c`

```
5  int main(int argc, char **argv)  {
6     double inicio, final;
7     int i,j,k;
8
9     if(argc < 2){
10       printf(stderr,"Error de introducción de parámetros: Falta el tamaño de las matrices (tienen que ser cuadradas)\n");
11       exit(-1);
12    }
13
14    int tamanio = atoi(argv[1]);
15
16    if(tamanio < 2){
17        printf(stderr, "El tamaño tiene que ser igual o superior a 2");
18        exit(-1);
19    }
20
21    double **m1, **m2, **m;
22
23    m1 = (double**) malloc(tamanio*sizeof(double*));
24    m2 = (double**) malloc(tamanio*sizeof(double*));
25    m = (double**) malloc(tamanio*sizeof(double*));
26
27    #pragma omp parallel for
28    for(i=0; i<tamanio; i++){
29        m1[i] = (double*) malloc(tamanio*sizeof(double));
30        m2[i] = (double*) malloc(tamanio*sizeof(double));
31        m[i] = (double*) malloc(tamanio*sizeof(double));
32    }
```

```
34      #pragma omp parallel private(i,j,k)
35      {
36          #pragma omp for schedule(dynamic)
37              for(i=0; i<tamanio; i++){
38                  for(j=0; j<tamanio; j++){
39                      m1[i][j] = 3;
40                      m2[i][j] = 3;
41                      m[i][j] = 0;
42                  }
43              }
44          #pragma omp critical
45              inicio = omp_get_wtime();
46
47          #pragma omp for schedule(dynamic)
48              for(i=0; i<tamanio; i++){
49                  for(j=0; j<tamanio; j++){
50                      double local = 0;
51                      for(k=0;k<tamanio;k++){
52                          local += m1[i][j]*m2[i][j];
53                      }
54                      m[i][j] = local;
55                  }
56              }
57
58          #pragma omp critical
59              final = omp_get_wtime() - inicio;
60      }
```

```
61      if(tamanio <= 8){
62          printf("Tamaño: %i\n Tiempo de ejecución: %f\n", tamanio, final);
63          for(int i=0; i<tamanio; i++)
64              for(int j=0;j<tamanio;j++)
65                  printf("m[%i][%i] = %f\n", i,j, m[i][j]);
66      }
67
68      else{
69          printf("Tamaño: %i\n Tiempo de ejecución: %f\n Primer elemento: %f\n Último elemento: %f\n", tamanio, final,
70           m[0][0], m[tamanio-1][tamanio-1]);
71      }
72
73      for(int i=0; i<tamanio; i++){
74          free(m[i]);
75          free(m1[i]);
76          free(m2[i]);
77      }
78
79      free(m);
80      free(m1);
81      free(m2);
82
83      return 0;
84  }
```

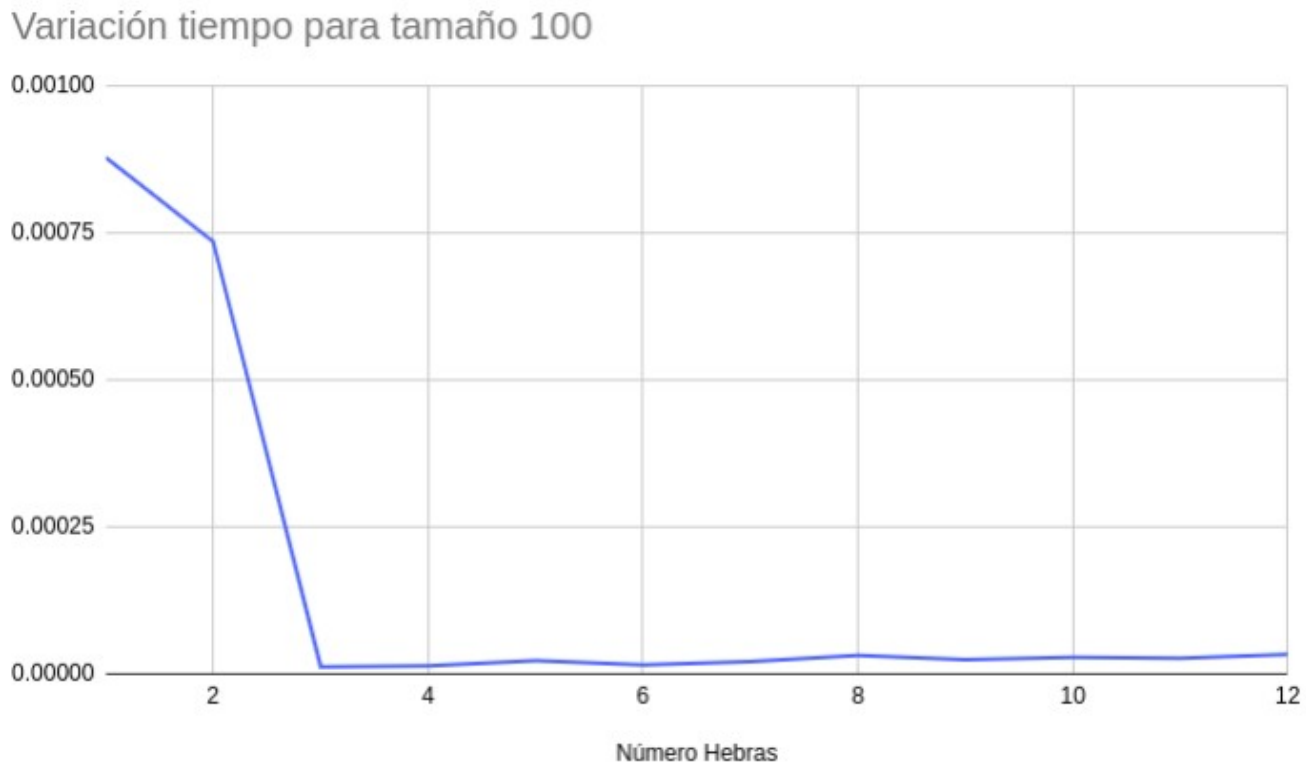
CAPTURAS DE PANTALLA:

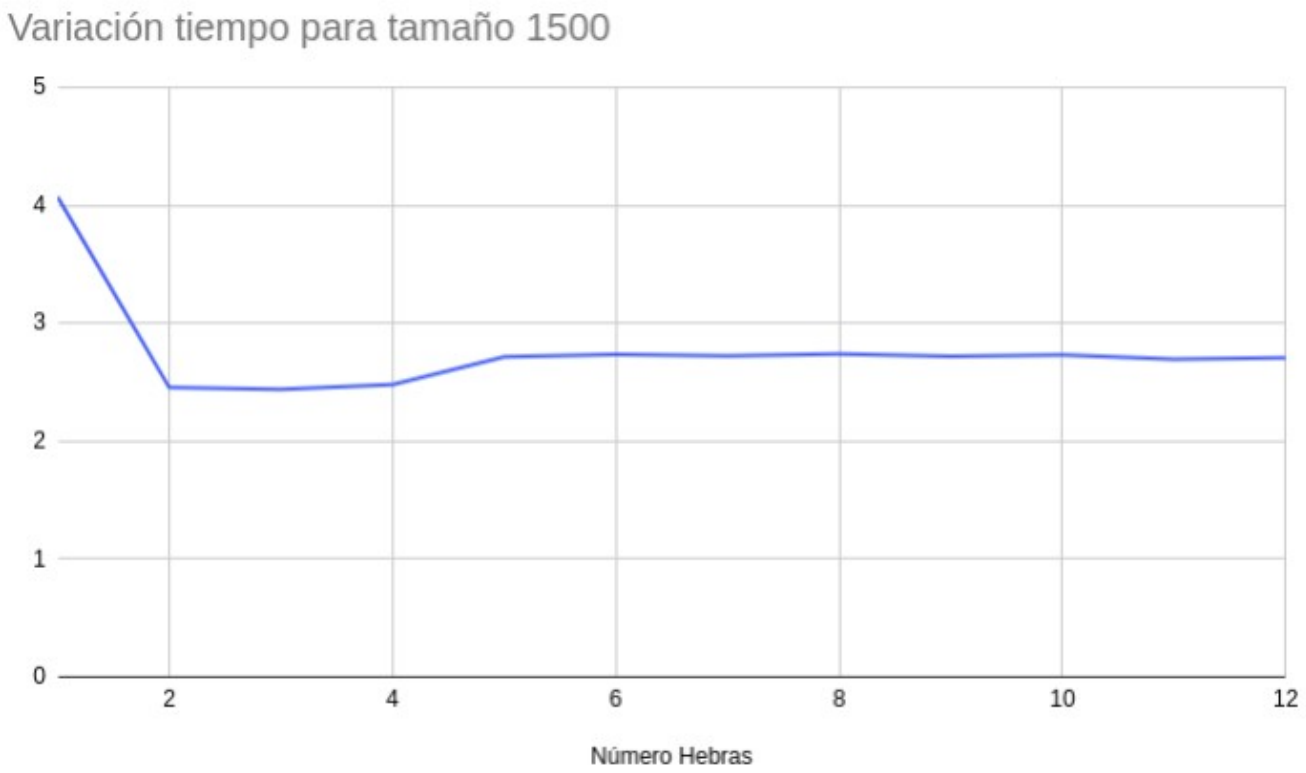
```
[DavidGómezHernández david@david-HP-Pavilion-Notebook:/home/david/Escritorio/Carrera/AC/bp3/ejer9] 04/2
7/20 11:20:38
$ ./pmm-OpenMP 3
Tamaño: 3
 Tiempo de ejecución: 0.000002
m[0][0] = 27.000000
m[0][1] = 27.000000
m[0][2] = 27.000000
m[1][0] = 27.000000
m[1][1] = 27.000000
m[1][2] = 27.000000
m[2][0] = 27.000000
m[2][1] = 27.000000
m[2][2] = 27.000000
```

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 Hebras	Tiempos para tamaño 100	Tiempos para tamaño 1500
1	0.000879	4.075176
2	0.000736	2.456459
3	0.000012	2.441078
4	0.000014	2.479383
5	0.000023	2.717792
6	0.000015	2.735243
7	0.000021	2.724303
8	0.000032	2.743055
9	0.000024	2.720993
10	0.000028	2.733759
11	0.000027	2.69786
12	0.000033	2.709644


Variación tiempo para tamaño 100
0.00100
0.00075
0.00050
0.00025
0.00000
2
4
6
8
10
12
Número Hebras

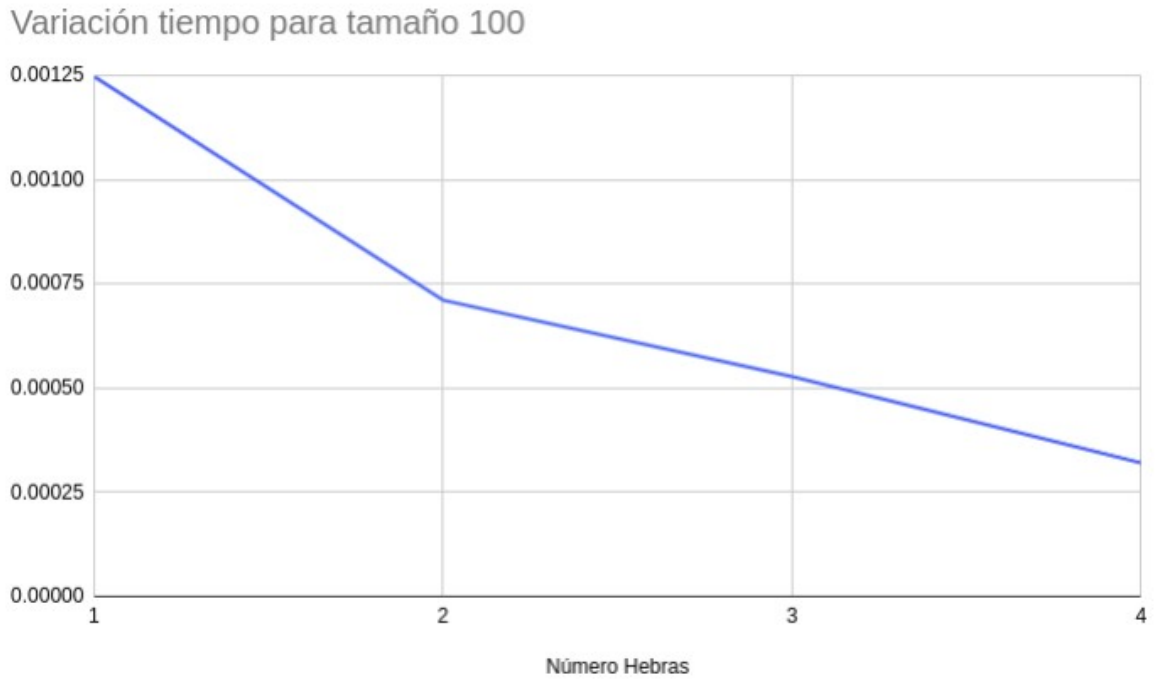

Variación tiempo para tamaño 1500
5
4
3
2
1
0
2
4
6
8
10
12
Número Hebras

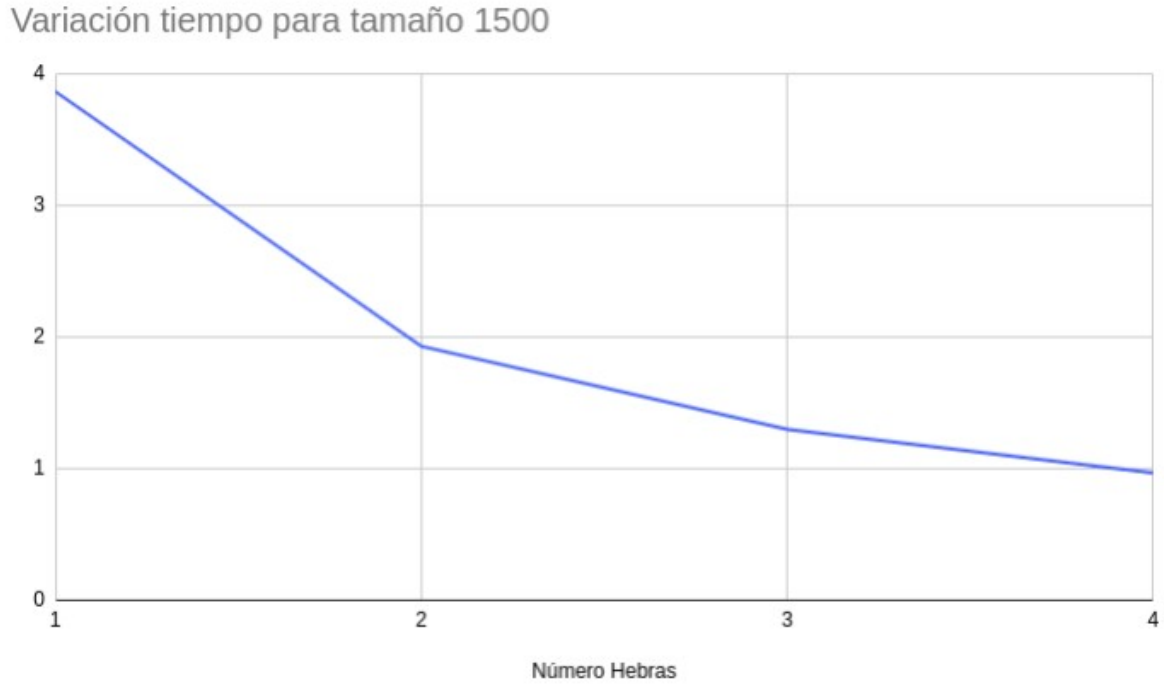
SCRIPT: pmm-OpenMP_atcgrid.sh

```
#!/bin/bash
export OMP_DYNAMIC=FALSE
#Todos los scripts que se hagan para atcgrid deben incluir lo siguiente:
#PBS -N suma
#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 ((i=1;i<=12;i++))
do
    export OMP_NUM_THREADS=$i
    echo "Número hebras $i"
    srun -p ac ./pmm-OpenMP 100
done

echo "TAMAÑO 1500"
for ((i=1;i<=12;i++))
do
    export OMP_NUM_THREADS=$i
    echo "Número hebras $i"
    srun -p ac ./pmm-OpenMP 1500
done
```

ESTUDIO DE ESCALABILIDAD EN PCLOCAL:

Número Hebras	Tiempos para tamaño 100	Tiempos para tamaño 1500
1	0.001248	3.869328
2	0.000711	1.931122
3	0.000528	1.300038
4	0.000321	0.968391





SCRIPT: pmm-OpenMP_pclocal.sh

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

for ((i=1;i<=4;i++))
do
    export OMP_NUM_THREADS=$i
    echo "Número hebras $i"
    ./pmm-OpenMP 100
done

for ((i=1;i<=4;i++))
do
    export OMP_NUM_THREADS=$i
    echo "Número hebras $i"
    ./pmm-OpenMP 1500
done
```