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): Francisco José González García
Grupo de prácticas: A2
Fecha de entrega: 15-05-18
Fecha evaluación en clase: 15-05-18

---

### Ejercicios basados en los ejemplos del seminario práctico

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

**CAPTURA CÓDIGO FUENTE**: `if-clauseModificado.c`

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char **argv) {
  int i, n = 20, tid, x;
  int a[n], suma = 0, sumalocal;
  if (argc < 2) {
    fprintf(stderr, "[ERROR]-Falta iteraciones y numero de hebras\n");
    exit(-1);
  }
  if (argc < 3) {
    fprintf(stderr, "[ERROR]-Falta numero de hebras\n");
    exit(-1);
  }
  n = atoi(argv[1]);
  x = atoi(argv[2]);
  if (x > 4) x = 4;
  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(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:**

```
/media/fran/windows/Users/fjose/Google Drive/Universidad/AC/Practicas/P3
[Francisco Jose Gonzalez Garcia fran@fran-Aspire-V3-572G] 18-05-12 sábado
❯ ./if_clause 12 2
 thread 0 suma de a[0]=0 sumalocal=0
 thread 0 suma de a[1]=1 sumalocal=1
 thread 0 suma de a[2]=2 sumalocal=3
 thread 0 suma de a[3]=3 sumalocal=6
 thread 0 suma de a[4]=4 sumalocal=10
 thread 0 suma de a[5]=5 sumalocal=15
 thread 1 suma de a[6]=6 sumalocal=6
 thread 1 suma de a[7]=7 sumalocal=13
 thread 1 suma de a[8]=8 sumalocal=21
 thread 1 suma de a[9]=9 sumalocal=30
 thread 1 suma de a[10]=10 sumalocal=40
 thread 1 suma de a[11]=11 sumalocal=51
thread master=0 imprime suma=66

/media/fran/windows/Users/fjose/Google Drive/Universidad/AC/Practicas/P3
[Francisco Jose Gonzalez Garcia fran@fran-Aspire-V3-572G] 18-05-12 sábado
❯ ./if_clause 23 5
 thread 0 suma de a[0]=0 sumalocal=0
 thread 0 suma de a[1]=1 sumalocal=1
 thread 0 suma de a[2]=2 sumalocal=3
 thread 0 suma de a[3]=3 sumalocal=6
 thread 0 suma de a[4]=4 sumalocal=10
 thread 3 suma de a[15]=15 sumalocal=15
 thread 3 suma de a[16]=16 sumalocal=31
 thread 3 suma de a[17]=17 sumalocal=48
 thread 3 suma de a[18]=18 sumalocal=66
 thread 3 suma de a[19]=19 sumalocal=85
 thread 2 suma de a[10]=10 sumalocal=10
 thread 2 suma de a[11]=11 sumalocal=21
 thread 2 suma de a[12]=12 sumalocal=33
 thread 2 suma de a[13]=13 sumalocal=46
 thread 2 suma de a[14]=14 sumalocal=60
 thread 1 suma de a[5]=5 sumalocal=5
 thread 1 suma de a[6]=6 sumalocal=11
 thread 1 suma de a[7]=7 sumalocal=18
 thread 1 suma de a[8]=8 sumalocal=26
 thread 1 suma de a[9]=9 sumalocal=35
thread master=0 imprime suma=190
```

**RESPUESTA**:
En el primer ejemplo solo trabajan 2 hebras porque es el número que le hemos pasado como parámetro y en la segunda ejecución trabajan las 4 hebras del pc aunque le hayamos pasado 5 como parámetro, debido a que en el código se ha contemplado de esa manera para que no trabajen más hebras de las que posee el pc.

**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)
- chunk= 1, 2 y 4

**Tabla 1 .** Tabla schedule. En la segunda fila, 1, 2 4 representan el tamaño del chunk (consulte seminario)

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

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

**Tabla 2 .** Tabla schedule. En la segunda fila, 1, 2 4 representan el tamaño del chunk (consulte seminario)

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

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

**RESPUESTA**:
*Static* asigna las iteraciones a cada hebra en tiempo de compilación, dividiendo las iteraciones en unidades de chunk iteraciones y asignandolas siguiendo una planificación round-robin. En el caso de *dynamic* y *guided*, las iteraciones se dividen en tiempo de ejecución y, al igual que ocurre en *static*, las iteraciones se dividen en unidades de chunk iteraciones. Lo que diferencia a *guided* del resto es que con este tipo las asignaciones comienzan con bloque largo (nº iteraciones restantes / nº threads) que va menguando.

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

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

```
#include <stdio.h>
#include <stdlib.h>
#ifdef _OPENMP
#include <omp.h>
#else
#define omp_get_thread_num() 0
#endif
int main(int argc, char **argv) {
  int i, n = 200, chunk, a[n], suma = 0;
  int nchunk = 0;
  omp_sched_t kind = 0;
  static char *sched_kind[] = {"static", "dynamic", "guided", "auto"};
  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 for schedule(dynamic, chunk) firstprivate(suma) lastprivate(suma)
    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 single
    {
      omp_get_schedule(&kind, &nchunk);
      printf(
          "\nDentro de la region parallel\n "
          "dyn-var:%d\n nthreads-var:%d\n "
          "thread-limit-var:%d\n "
          "run-sched-var:\n\tKind:%s  chunk:%d\n",
          omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit(),
          sched_kind[kind - 1], nchunk);
    }
  }

  omp_get_schedule(&kind, &nchunk);
  printf(
      "\nFuera de la region parallel\n "
      "dyn-var:%d\n nthreads-var:%d\n "
      "thread-limit-var:%d\n "
      "run-sched-var:\n\tKind:%s  chunk:%d\n",
      omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit(),
      sched_kind[kind - 1], nchunk);

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

**CAPTURAS DE PANTALLA:**

```
Variables POR DEFECTO

Dentro de la region parallel
dyn-var:0
 nthreads-var:4
 thread-limit-var:2147483647
 run-sched-var:
        Kind:dynamic  chunk:1

Fuera de la region parallel
dyn-var:0
 nthreads-var:4
 thread-limit-var:2147483647
 run-sched-var:
        Kind:dynamic  chunk:1
```

```
export OMP_SCHEDULE="static,4"
export OMP_NUM_THREADS=8
export OMP_DYNAMIC=true

Dentro de la region parallel
dyn-var:1
 nthreads-var:8
 thread-limit-var:2147483647
 run-sched-var:
        Kind:static  chunk:4

Fuera de la region parallel
dyn-var:1
 nthreads-var:8
 thread-limit-var:2147483647
 run-sched-var:
        Kind:static  chunk:4
```

**RESPUESTA**:
Como podemos ver en los ejemplos se imprimen los mismos valores tanto fuera como dentro de la región paralela ya que estos no dependen de ello, si no del valor que tengan las variables de control y si son o no modificadas.

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

**CAPTURA CÓDIGO FUENTE**: `scheduled-clauseModificado4.c`

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

```
int main(int argc, char **argv) {
  int i, n = 200, chunk, a[n], suma = 0;
  int nchunk = 0;
  omp_sched_t kind = 0;
  static char *sched_kind[] = {"static", "dynamic", "guided", "auto"};
  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 for schedule(dynamic, chunk) firstprivate(suma) lastprivate(suma)
    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 single
    {
      omp_get_schedule(&kind, &nchunk);
      printf(
          "\nDentro de la region parallel\n "
          "dyn-var:%d\n nthreads-var:%d\n "
          "thread-limit-var:%d\n "
          "run-sched-var:\n\tKind:%s  chunk:%d\n "
          "num_threads:%d\n num_procs:%d\n in_parallel:%d\n ",
          omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit(),
          sched_kind[kind - 1], nchunk, omp_get_num_threads(),
          omp_get_num_procs(), omp_in_parallel());
    }
  }

  omp_get_schedule(&kind, &nchunk);
  printf(
      "\nDentro de la region parallel\n "
      "dyn-var:%d\n nthreads-var:%d\n "
      "thread-limit-var:%d\n "
      "run-sched-var:\n\tKind:%s  chunk:%d\n "
      "num_threads:%d\n num_procs:%d\n in_parallel:%d\n ",
      omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit(),
      sched_kind[kind - 1], nchunk, omp_get_num_threads(),
omp_get_num_procs(),
      omp_in_parallel());

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

**CAPTURAS DE PANTALLA:**

**Mismos valores para las variables de control que en la ejecución anterior**

Dentro de la region parallel
dyn-var:1
nthreads-var:8
thread-limit-var:2147483647
run-sched-var:
        Kind:static  chunk:4
num_threads:4
num_procs:4

```
 in_parallel:1

Dentro de la region parallel
 dyn-var:1
 nthreads-var:8
 thread-limit-var:2147483647
 run-sched-var:
          Kind:static  chunk:4
 num_threads:1
 num_procs:4
 in_parallel:0
```

**RESPUESTA**:

Como vemos las funciones con las que obtenemos un valor diferente en función de si están dentro o fuera de una región paralela son *omp_get_num_threads() y omp_in_parallel*, mientras que el número de procesadores disponibles para la ejecución del programa no varía, es decir, el valor que da la función *omp_get_num_procs()* es el mismo.

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

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

```
#include <stdio.h>
#include <stdlib.h>
#ifdef _OPENMP
#include <omp.h>
#else
#define omp_get_thread_num() 0
#endif
int main(int argc, char **argv) {
  int i, n = 200, chunk, a[n], suma = 0;
  int nchunk = 0;
  omp_sched_t kind = 0;
  static char *sched_kind[] = {"static", "dynamic", "guided", "auto"};
  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 for schedule(dynamic, chunk) firstprivate(suma) lastprivate(suma)
    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 single
   {
     omp_get_schedule(&kind, &nchunk);
     printf(
        "\nAntes de modificar %d\n "
        "dyn-var:%d\n nthreads-var:%d\n "
        "thread-limit-var:%d\n "
        "run-sched-var:\n\tKind:%s  chunk:%d\n "
        "num_threads:%d\n num_procs:%d\n in_parallel:%d\n ",
        omp_get_thread_num(), omp_get_dynamic(), omp_get_max_threads(),
        omp_get_thread_limit(), sched_kind[kind - 1], nchunk,
        omp_get_num_threads(), omp_get_num_procs(), omp_in_parallel());
   }
  }
  omp_set_schedule(2, 2);
  omp_set_dynamic(0);
  omp_set_num_threads(4);
  omp_get_schedule(&kind, &nchunk);
  printf(
     "\nDespues de modificar\n "
     "dyn-var:%d\n nthreads-var:%d\n "
     "thread-limit-var:%d\n "
     "run-sched-var:\n\tKind:%s  chunk:%d\n "
     "num_threads:%d\n num_procs:%d\n in_parallel:%d\n ",
     omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit(),
     sched_kind[kind - 1], nchunk, omp_get_num_threads(), omp_get_num_procs(),
     omp_in_parallel());

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

**CAPTURAS DE PANTALLA:**

```
Antes de modificar
 dyn-var:1
 nthreads-var:2
 thread-limit-var:2147483647
 run-sched-var:
          Kind:static  chunk:4
 num_threads:1
 num_procs:4
 in_parallel:0

Despues de modificar
 dyn-var:0
 nthreads-var:4
 thread-limit-var:2147483647
 run-sched-var:
          Kind:dynamic  chunk:2
 num_threads:1
 num_procs:4
 in_parallel:0
```

## 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`

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

int main(int argc, char** argv) {
  if (argc < 2) {
    perror("Falta tamanio del vector");
    exit(EXIT_FAILURE);
  }
  int N = atoi(argv[1]);
  int** matriz = malloc(N * sizeof(int*));
  for (int i = 0; i < N; i++) matriz[i] = malloc(N * sizeof(int));
  int* vector = malloc(N * sizeof(int));
  int* solucion = malloc(N * sizeof(int));

  // Inicializacion
  for (int i = 0; i < N; i++)
    for (int j = 0; j < N; j++)
      if (j >= i)
        matriz[i][j] = i + j;
      else
        matriz[i][j] = 0;
  for (int i = 0; i < N; i++) vector[i] = i;

  // Calculo
  for (int i = 0; i < N; i++) {
    int aux = 0;
    for (int j = i; j < N; j++) aux += matriz[i][j] * vector[j];
    solucion[i] = aux;
  }

  // Salida
  printf("resultado: {");
  if (N < 12) {
    for (int i = 0; i < N; i++) printf("%d, ", solucion[i]);
    printf("\b\b}");
  } else {
    printf("%d,..., %d}", solucion[0], solucion[N - 1]);
  }

  // Liberacion memoria
  for (int i = 0; i < N; i++) free(matriz[i]);
  free(matriz);
  free(vector);
  free(solucion);
}
```

**Respuesta:**
El orden de complejidad que teniamos en la implementación anterior del producto de una matriz por un vector era de orden $n^2$, siendo n el tamaño de la matriz, mientras que en esta nueva implementación el segundo bucle solo recorre las columnas desde j=i, con i=nº de fila, hasta j<N. Reduciendo de esta forma el orden de complejidad en tiempo.

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) Static: 0 Dynamic: 1 Guided: 1

Lo he averiguado asignando a la variable de entorno OMP_SCHEDULE tan solo el tipo planificación, obteniendo de esa forma el chunk por defecto e imprimiendolo dentro del programa.

b) Para chunk = 1

filas para cada thread: 30720/12 = 2560

multiplicaciones y sumas por thread: (30720 - i) * 2560 , siendo i la fila correspondiente

Para chunk = 64

filas para cada thread: 30720/64 = 480; 480*64 = 2560

multiplicaciones y sumas por thread: (30720 - i) * 2560 , siendo i la fila correspondiente

c) Dependerá de la velocidad de procesamiento de cada thread, y por tanto de cuantas iteraciones ejecute cada uno.

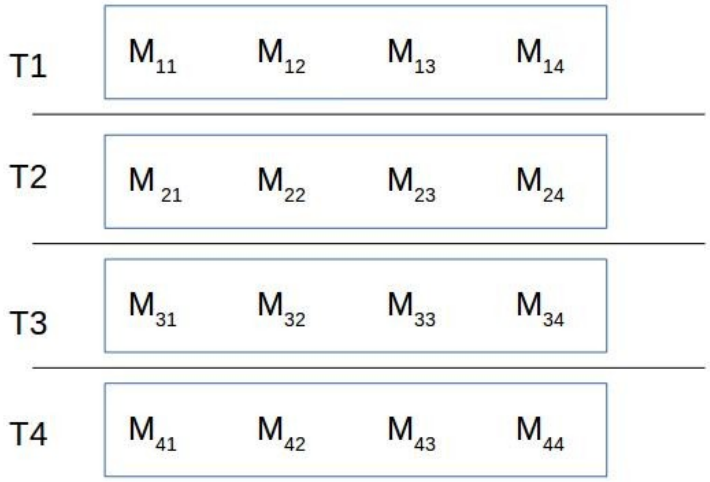
**CAPTURA CÓDIGO FUENTE**: `pmtv-OpenMP.c`

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#ifdef _OPENMP
#include <omp.h>
#else
typedef int omp_sched_t;
inline void omp_get_schedule(omp_sched_t* kind, int* chunk) {
  *kind = 0;
  *chunk = 0;
}
#endif
#define tiempo 1

int main(int argc, char** argv) {
  if (argc < 2) {
    perror("Falta tamanio del vector");
    exit(EXIT_FAILURE);
  }

  int N = atoi(argv[1]);
  int** matriz = malloc(N * sizeof(int*));
  for (int i = 0; i < N; i++) matriz[i] = malloc(N * sizeof(int));
  int* vector = malloc(N * sizeof(int));
  int* solucion = malloc(N * sizeof(int));
  omp_sched_t kind = 0;
  int chunk = 0;
  static char* sched_kind[] = {"static", "dynamic", "guided", "auto"};
  struct timespec cgt1, cgt2;
  double ncgt;

  // Inicializacion
  for (int i = 0; i < N; i++)
    for (int j = 0; j < N; j++)
      if (j >= i)
        matriz[i][j] = i + j;
      else
        matriz[i][j] = 0;
  for (int i = 0; i < N; i++) vector[i] = i;

  // Calculo
  clock_gettime(CLOCK_REALTIME, &cgt1);
#pragma omp parallel for schedule(runtime)
  for (int i = 0; i < N; i++) {
    int aux = 0;
    // printf("Thread: %d\t IT: %d\n", omp_get_thread_num(), i);
    for (int j = i; j < N; j++) aux += matriz[i][j] * vector[j];
    solucion[i] = aux;
  }
  clock_gettime(CLOCK_REALTIME, &cgt2);
  ncgt = (double)(cgt2.tv_sec - cgt1.tv_sec) +
         (double)((cgt2.tv_nsec - cgt1.tv_nsec) / (1.e+9));

  // Salida
  omp_get_schedule(&kind, &chunk);
  printf("Kind:%s  chunk:%d\n", sched_kind[kind - 1], chunk);
  printf("resultado: {");
  if (N < 12) {
    for (int i = 0; i < N; i++) printf("%d, ", solucion[i]);
    printf("\b\b}\n");
  } else {
```

```
    printf("%d,..., %d}\n", solucion[0], solucion[N - 1]);
  }
  printf("%f\n", ncgt);
  // Liberacion memoria
  for (int i = 0; i < N; i++) free(matriz[i]);
  free(matriz);
  free(vector);
  free(solucion);
}
```

**DESCOMPOSICIÓN DE DOMINIO:**

Para un tamaño N=4 como el usado en el ejemplo cada fila de la matriz se reparte entre las hebras, en nuestro caso 4 también. Cada una de ellas se encarga de hacer la multiplicación de su fila correspondiente y de asignarla al vector solución.




**CAPTURAS DE PANTALLA:**

```
/media/fran/windows/Users/fjose/Google Drive/Universidad/AC/Practicas/P3
[Francisco Jose Gonzalez Garcia fran@fran-Aspire-V3-572G] 18-05-13 domingo
❯ gcc src/pmtv-OpenMP.c -o pmtv-OpenMP -O2 -fopenmp -lrt

/media/fran/windows/Users/fjose/Google Drive/Universidad/AC/Practicas/P3
[Francisco Jose Gonzalez Garcia fran@fran-Aspire-V3-572G] 18-05-13 domingo
❯ ./pmtv-OpenMP 16
Kind:static  chunk:0
resultado: {1240,..., 450}
0.000223
```

**TABLA RESULTADOS, SCRIPT Y GRÁFICA atcgrid**

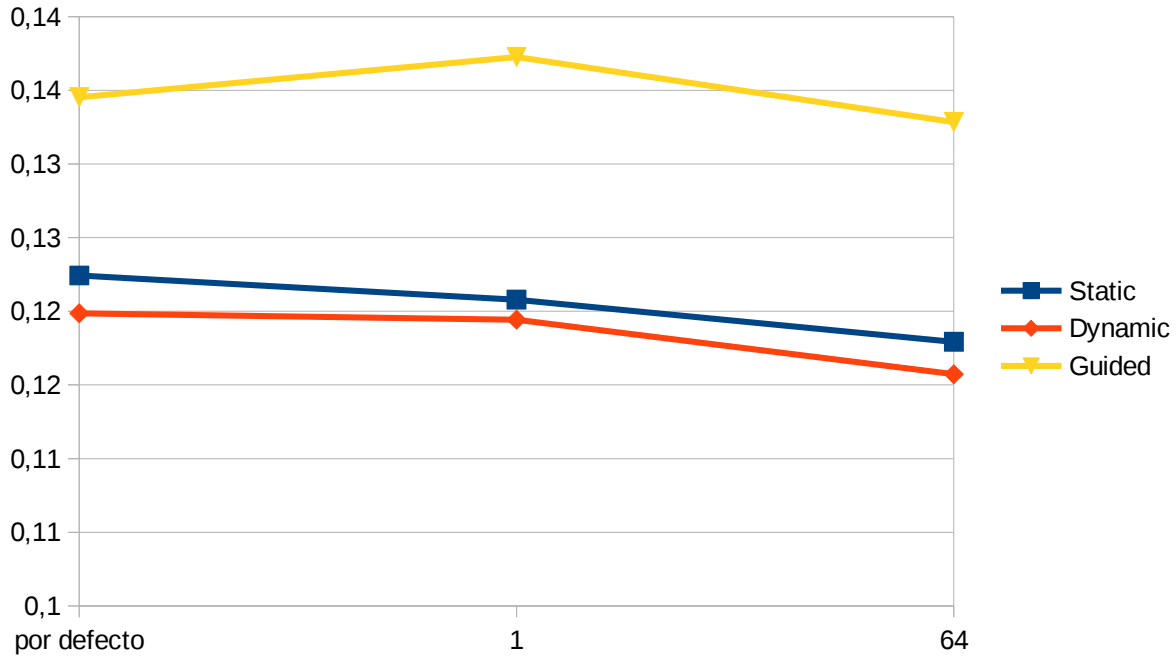
**SCRIPT:** pmvt-OpenMP_PCaula.sh

```
#!/bin/bash
#PBS -N salida
#PBS -q ac


export OMP_NUM_THREADS=12
for((i=0;i<2;i=i+1))
do
  echo "Iteracion $i"
  export OMP_SCHEDULE="static"
  $PBS_O_WORKDIR/pmtv-OpenMP 30720
  export OMP_SCHEDULE="dynamic"
```

```
  $PBS_O_WORKDIR/pmtv-OpenMP 30720
  export OMP_SCHEDULE="guided"
  $PBS_O_WORKDIR/pmtv-OpenMP 30720
  export OMP_SCHEDULE="static,1"
  $PBS_O_WORKDIR/pmtv-OpenMP 30720
  export OMP_SCHEDULE="dynamic,1"
  $PBS_O_WORKDIR/pmtv-OpenMP 30720
  export OMP_SCHEDULE="guided,1"
  $PBS_O_WORKDIR/pmtv-OpenMP 30720
  export OMP_SCHEDULE="static,64"
  $PBS_O_WORKDIR/pmtv-OpenMP 30720
  export OMP_SCHEDULE="dynamic,64"
  $PBS_O_WORKDIR/pmtv-OpenMP 30720
  export OMP_SCHEDULE="guided,64"
  $PBS_O_WORKDIR/pmtv-OpenMP 30720
  printf "\n\n"
done
```

**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** | 0.122431 | 0.119868 | 0.134532 |
| **1** | 0.120790 | 0.119438 | 0.137265 |
| **64** | 0.117932 | 0.115728 | 0.132850 |
| **Chunk** | **Static** | **Dynamic** | **Guided** |
| **por defecto** | 0.120905 | 0.119282 | 0.135547 |
| **1** | 0.117339 | 0.120183 | 0.124000 |
| **64** | 0.118248 | 0.115995 | 0.136029 |



Aunque cabría esperar que fuese la planificación static la más veloz vemos que para estos tamaños de problema la diferencia entre static y dynamic es mínima, siendo la sobrecarga propia de este último inapreciable.

**8.** Implementar un programa secuencial en C que calcule la multiplicación de matrices cuadradas, B y C:

$$A = B \bullet C;\ A(i,j) = \sum_{k=0}^{N-1} B(i,k) \bullet C(k,j),\ i,j = 0,\ldots N-1$$

NOTAS: (1) el número de filas/columnas debe ser un argumento de entrada; (2) se deben inicializar las matrices antes del cálculo; (3) se debe imprimir siempre las componentes (0,0) y (N-1, N-1) del resultado antes de que termine el programa.

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

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

int main(int argc, char** argv) {
  if (argc < 2) {
    perror("Falta tamanio del vector");
    exit(EXIT_FAILURE);
  }
  int N = atoi(argv[1]);
  int** matriz1 = malloc(N * sizeof(int*));
  int** matriz2 = malloc(N * sizeof(int*));
  int** solucion = malloc(N * sizeof(int*));
  for (int i = 0; i < N; i++) {
    matriz1[i] = malloc(N * sizeof(int));
    matriz2[i] = malloc(N * sizeof(int));
    solucion[i] = malloc(N * sizeof(int));
  }

  // Inicializacion
  for (int i = 0; i < N; i++)
    for (int j = 0; j < N; j++) {
      matriz1[i][j] = 2;
      matriz2[i][j] = 2;
      solucion[i][j] = 0;
    }

  // Calculo
  for (int i = 0; i < N; i++) {
    for (int j = 0; j < N; j++) {
      for (int k = 0; k < N; k++)
        solucion[i][j] += matriz1[i][k] * matriz2[k][j];
    }
  }

  // Salida
  printf("resultado: {");
  if (N < 12) {
    for (int i = 0; i < N; i++)
      for (int j = 0; j < N; j++) printf("%d, ", solucion[i][j]);
    printf("\b\b}");
  } else {
    printf("%d,..., %d}", solucion[0][0], solucion[N - 1][N - 1]);
  }

  // Liberacion memoria
  for (int i = 0; i < N; i++) {
    free(matriz1[i]);
```

```
    free(matriz2[i]);
    free(solucion[i]);
  }
  free(matriz1);
  free(matriz2);
  free(solucion);
}
```

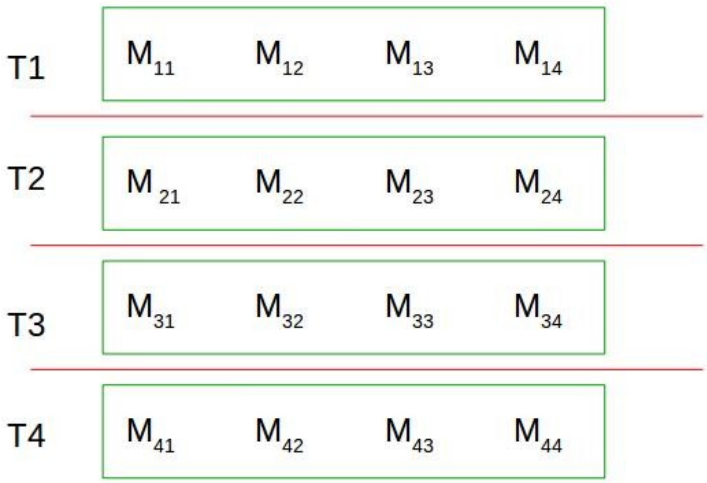
**CAPTURAS DE PANTALLA:**

```
/media/fran/windows/Users/fjose/Google Drive/Universidad/AC/Practicas/P3
[Francisco Jose Gonzalez Garcia fran@fran-Aspire-V3-572G] 18-05-13 domingo
❯ gcc src/pmm-secuencial.c  -o pmm-secuencial -O2

/media/fran/windows/Users/fjose/Google Drive/Universidad/AC/Practicas/P3
[Francisco Jose Gonzalez Garcia fran@fran-Aspire-V3-572G] 18-05-13 domingo
❯ ./pmm-secuencial 20
resultado: {80,..., 80}%

/media/fran/windows/Users/fjose/Google Drive/Universidad/AC/Practicas/P3
[Francisco Jose Gonzalez Garcia fran@fran-Aspire-V3-572G] 18-05-13 domingo
❯ ./pmm-secuencial 16
resultado: {64,..., 64}%
```

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:**

Como en el ejercicio anterior, la descomposición de datos que resulta se trata de dividir las filas de la primera matriz entre todas las hebras siguiendo una planificación static, mientras que las columnas y las filas y columnas de la matriz 2 son recorridas por todas las hebras por igual.




**MATRIZ 1**

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

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#ifdef _OPENMP
#include <omp.h>
#else
#endif
int main(int argc, char** argv) {
  if (argc < 2) {
    perror("Falta tamanio del vector");
    exit(EXIT_FAILURE);
  }
  int N = atoi(argv[1]);
  int** matriz1 = malloc(N * sizeof(int*));
  int** matriz2 = malloc(N * sizeof(int*));
  int** solucion = malloc(N * sizeof(int*));
  for (int i = 0; i < N; i++) {
    matriz1[i] = malloc(N * sizeof(int));
    matriz2[i] = malloc(N * sizeof(int));
    solucion[i] = malloc(N * sizeof(int));
  }

  // Inicializacion
#pragma omp parallel for
  for (int i = 0; i < N; i++)
    for (int j = 0; j < N; j++) {
      matriz1[i][j] = 2;
      matriz2[i][j] = 2;
      solucion[i][j] = 0;
    }

  // Calculo
#pragma omp parallel for schedule(static)
  for (int i = 0; i < N; i++) {
    for (int j = 0; j < N; j++) {
      for (int k = 0; k < N; k++)
        solucion[i][j] += matriz1[i][k] * matriz2[k][j];
    }
  }

  // Salida
  printf("resultado: {");
  if (N < 12) {
    for (int i = 0; i < N; i++)
      for (int j = 0; j < N; j++) printf("%d, ", solucion[i][j]);
    printf("\b\b}");
  } else {
    printf("%d,..., %d}", solucion[0][0], solucion[N - 1][N - 1]);
  }

  // Liberacion memoria
  for (int i = 0; i < N; i++) {
    free(matriz1[i]);
    free(matriz2[i]);
    free(solucion[i]);
  }
  free(matriz1);
  free(matriz2);
  free(solucion);
}
```

**CAPTURAS DE PANTALLA:**

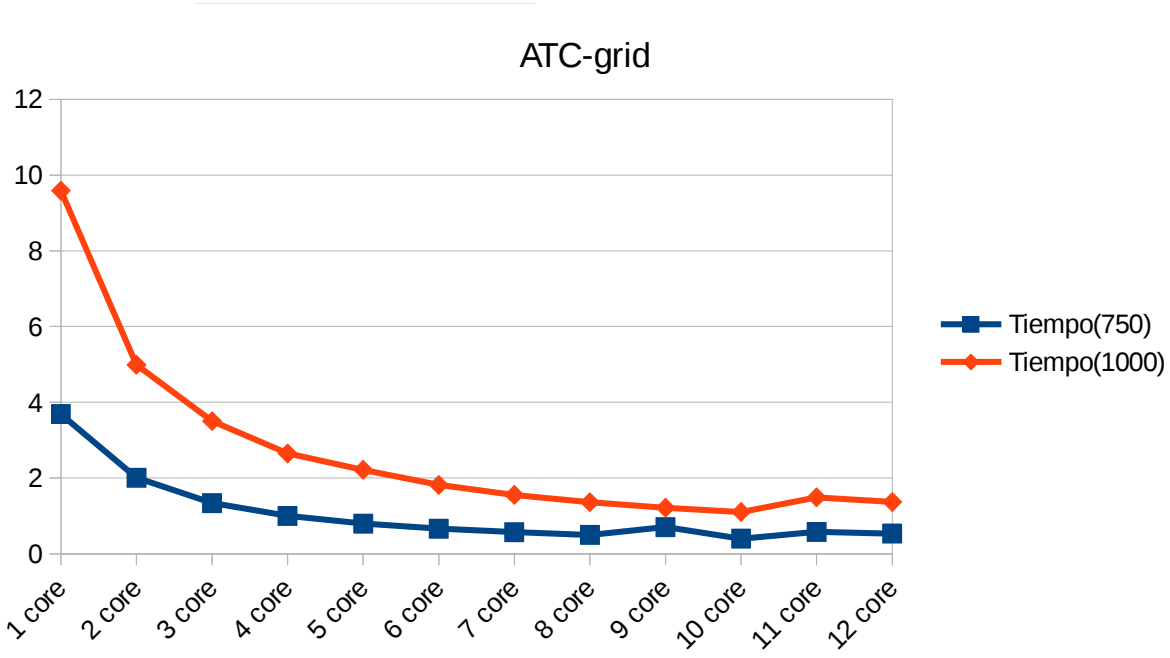
```
/media/fran/windows/Users/fjose/Google Drive/Universidad/AC/Practicas/P3
[Francisco Jose Gonzalez Garcia fran@fran-Aspire-V3-572G] 18-05-13 domingo
❯ gcc src/pmm-OpenMP.c  -o pmm-OpenMP -O2 -fopenmp

/media/fran/windows/Users/fjose/Google Drive/Universidad/AC/Practicas/P3
[Francisco Jose Gonzalez Garcia fran@fran-Aspire-V3-572G] 18-05-13 domingo
❯ ./pmm-OpenMP 20
resultado: {80,..., 80}

/media/fran/windows/Users/fjose/Google Drive/Universidad/AC/Practicas/P3
[Francisco Jose Gonzalez Garcia fran@fran-Aspire-V3-572G] 18-05-13 domingo
❯ ./pmm-OpenMP 16
resultado: {64,..., 64}
```

10. Hacer un estudio de escalabilidad (ganancia en velocidad en función del número de cores) en atcgrid y en su PC del código paralelo implementado para dos tamaños de las matrices. Debe recordar usar `-O2` al compilar. El número de núcleos máximo en este estudio debe ser el igual al de núcleos físicos del computador. Presente los resultados del estudio en tablas de valores y en gráficas. Escoger los tamaños de manera que se observe diferentes curvas de escalabilidad en las gráficas que entregue en su cuaderno de prácticas (pruebe con valores de N entre 100 y 1500). Consulte la Lección 6/Tema 2. Incluya los scripts utilizado en el cuaderno de prácticas. NOTA: Nunca ejecute en atcgrid código que imprima todos los componentes del resultado.

**ESTUDIO DE ESCALABILIDAD EN atcgrid:**

**SCRIPT:** pmm-OpenMP_atcgrid.sh

```
#! /bin/bash
#PBS -N escalabilidad
#PBS -q ac
for((i=1;i<13;i=i+1))
do
export OMP_NUM_THREADS=$i
./pmm-OpenMP 750
printf '\n'
done
printf "\n\n"

for((i=1;i<13;i=i+1))
do
export OMP_NUM_THREADS=$i
./pmm-OpenMP 1000
printf '\n'
done
```

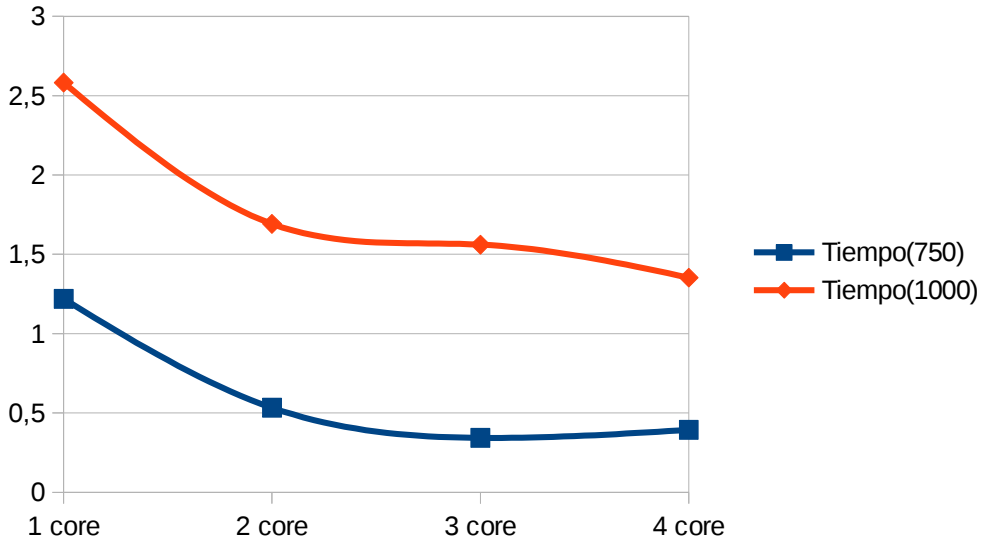



**ESTUDIO DE ESCALABILIDAD EN PCLOCAL:**

**SCRIPT:** pmm-OpenMP_pclocal.sh

```
#! /bin/bash

export OMP_NUM_THREADS=1
printf "$OMP_NUM_THREADS\n"
./pmm-OpenMP 750
printf '\n'
./pmm-OpenMP 1500
printf "\n\n"
export OMP_NUM_THREADS=2
printf "$OMP_NUM_THREADS\n"
./pmm-OpenMP 750
printf '\n'
./pmm-OpenMP 1500
printf "\n\n"
export OMP_NUM_THREADS=3
printf "$OMP_NUM_THREADS\n"
./pmm-OpenMP 750
printf '\n'
./pmm-OpenMP 1500
printf "\n\n"
export OMP_NUM_THREADS=4
printf "$OMP_NUM_THREADS\n"
./pmm-OpenMP 750
printf '\n'
./pmm-OpenMP 1500
```

| | Tiempo(750) | Tiempo(1000) |
|---|---|---|
| 1 core | 1,219606 | 2,582006 |
| 2 core | 0,53277 | 1,691633 |
| 3 core | 0,342932 | 1,560671 |
| 4 core | 0,393452 | 1,352545 |


3
2,5
2
1,5
1
0,5
0
1 core
2 core
3 core
4 core
Tiempo(750)
Tiempo(1000)