

2º curso / 2º cuatr.  
Grado Ing. Inform.  
Doble Grado Ing.  
Inform. y Mat.

## Arquitectura de Computadores (AC)

### Cuaderno de prácticas.

### Bloque Práctico 3. Programación paralela III: Interacción con el entorno en OpenMP

Estudiante (nombre y apellidos): Juan Carlos Ruiz García

Grupo de prácticas: C2

Fecha de entrega: 14/05/2017

Fecha evaluación en clase:

#### Ejercicios basados en los ejemplos del seminario práctico

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

#### CÓDIGO FUENTE: `if-clauseModificado.c`

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

int main(int argc, char **argv)
{
    int i, n=20, tid;
    int a[n], suma=0, sumalocal;
    int x;
    if(argc < 3) {
        fprintf(stderr, "[ERROR]-Falta iteraciones o numero de hebras\n");
        exit(-1);
    }
    n = atoi(argv[1]); if (n>20) n=20;
    x = atoi(argv[2]);

    for (i=0; i<n; i++) {
        a[i] = i;
    }

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

```
return(0);
}
```

**CAPTURAS DE PANTALLA:**

```
juanka1995@juanka-laptop ~/practicas/AC/Practicas/practica_3/ejercicio1 $ gcc -fopenmp if-clauseModificado.c -o if-clauseModificado
juanka1995@juanka-laptop ~/practicas/AC/Practicas/practica_3/ejercicio1 $ ./if-clauseModificado 5 5
thread 2 suma de a[2]=2 sumalocal=2
thread 3 suma de a[3]=3 sumalocal=3
thread 0 suma de a[0]=0 sumalocal=0
thread 1 suma de a[1]=1 sumalocal=1
thread 4 suma de a[4]=4 sumalocal=4
thread master=0 imprime suma=10
juanka1995@juanka-laptop ~/practicas/AC/Practicas/practica_3/ejercicio1 $ ./if-clauseModificado 4 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 master=0 imprime suma=6
juanka1995@juanka-laptop ~/practicas/AC/Practicas/practica_3/ejercicio1 $ ./if-clauseModificado 8 7
thread 1 suma de a[2]=2 sumalocal=2
thread 3 suma de a[4]=4 sumalocal=4
thread 2 suma de a[3]=3 sumalocal=3
thread 0 suma de a[0]=0 sumalocal=0
thread 4 suma de a[5]=5 sumalocal=5
thread 5 suma de a[6]=6 sumalocal=6
thread 0 suma de a[1]=1 sumalocal=1
thread 6 suma de a[7]=7 sumalocal=7
thread master=0 imprime suma=28
```

**RESPUESTA:**

Esta clausula especifica el número de hebras que queremos que se creen para ejecutar la región paralela. Como también usamos la cláusula **if(n>4)**, solo se ejecutará la cláusula **num\_threads** siempre y cuando haya mas de 4 iteraciones ha realizar.

En los ejemplos podemos ver como cuando el valor de n es >4 se crean las hebras especificadas y si no se realiza de forma secuencial (lo realiza todo la hebra master).

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-clause.d.c			schedule-clauseg.c		
	1	2	4	1	2	4	1	2	4
0	0	0	0	3	0	1	2	2	2
1	1	0	0	1	0	1	2	2	2
2	2	1	0	0	3	1	2	2	2
3	3	1	0	2	3	1	2	2	2
4	0	2	1	0	3	1	1	1	1
5	1	2	1	0	3	1	1	1	1
6	2	3	1	0	3	1	1	1	1
7	3	3	1	0	3	1	0	0	1

8	0	0	2	0	2	1	0	0	3
9	1	0	2	0	2	1	0	0	3
10	2	1	2	0	2	1	1	3	3
11	3	1	2	0	2	1	1	3	3
12	0	2	3	0	2	1	1	0	2
13	1	2	3	0	2	1	1	0	2
14	2	3	3	0	3	1	1	0	2
15	3	3	3	0	3	1	1	0	2

(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- clausd.c			schedule- clauseg.c		
	1	2	4	1	2	4	1	2	4
0	0	0	0	3	3	0	3	3	1
1	1	0	0	0	3	0	3	3	1
2	2	1	0	1	0	0	3	3	1
3	3	1	0	0	0	0	3	3	1
4	0	2	1	0	1	1	0	2	3
5	1	2	1	0	1	1	0	2	3
6	2	3	1	0	3	1	0	2	3
7	3	3	1	0	3	1	1	1	3
8	0	0	2	0	0	2	1	1	2
9	1	0	2	0	0	2	1	1	2
10	2	1	2	0	1	2	0	3	2
11	3	1	2	0	1	2	0	3	2
12	0	2	3	0	1	3	0	0	0
13	1	2	3	0	1	3	0	0	0
14	2	3	3	0	1	3	0	0	0
15	3	3	3	0	1	3	0	0	0

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

### RESPUESTA:

Diferencias:

- **Static** → Los bloques creados son de un chunk de tamaño fijo y se asignan/distribuyen en tiempo de compilación a las distintas hebras.
- **Dynamic** → Los bloques creados son de un chunk de tamaño fijo y en tiempo de ejecución se le asigna un bloque a cada hebra, cuando uno de ellos termine se le asigna otro bloque nuevo y así hasta terminar.
- **Guided** → Similar a *dynamic* con la diferencia de que el tamaño del bloque (chunk) decrece cada vez que un nuevo bloque es asignado a una hebra. El tamaño inicial del bloque es proporcional a:

$$\text{numero\_iteraciones} / \text{numero\_de\_hebras}$$

Los bloques siguientes tendrán un tamaño proporcional a:

$$\text{numero\_iteraciones\_restantes} / \text{numero\_de\_hebras}$$

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

#### CÓDIGO FUENTE: `scheduled-clauseModificado.c`

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

int main(int argc, char **argv) {
    int i, n=200, chunk, a[n], suma=0;
    omp_sched_t 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 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);
        if(omp_get_thread_num() == 0)
        {
            omp_get_schedule(&kind, &modifier);
            printf("Dentro de la region paralela:\ndyn-var: %d\nnthreads-var:
%d\nthread-limit-var: %d\nrun-sched-var: \n\tkind: %d\n\tmodifier:
%d\n", omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit(), kind, modi
fier);
        }
    }

    omp_get_schedule(&kind, &modifier);
    printf("Fuera de la region paralela:\ndyn-var: %d\nnthreads-var:
%d\nthread-limit-var: %d\nrun-sched-var: \n\tkind: %d\n\tmodifier:
%d\n", omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit(), kind, modi
fier);

    // printf("\nFuera de 'parallel for' suma=%d\n", suma);

    return(0);
}
```

### CAPTURAS DE PANTALLA:

```
cio3 $ export OMP_DYNAMIC=FALSE
cio3 $ export OMP_NUM_THREADS=8
cio3 $ export OMP_THREAD_LIMIT=10
cio3 $ export OMP_SCHEDULE="dynamic"
cio3 $
```

```
juanka1995@juanka-laptop ~/practicass/AC/Practicass/practica_3/ejercicio3 $ ./scheduled-clauseModificado 16 1
Dentro de la region paralela:
dyn-var: 0
threads-var: 8
thread-limit-var: 10
run-sched-var:
  kind: 2
  modifier: 1
Fuera de la region paralela:
dyn-var: 0
threads-var: 8
thread-limit-var: 10
run-sched-var:
  kind: 2
  modifier: 1
```

```
/* INTERLINEADO SENCILLO */
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

int main(int argc, char ** argv)
{
    ...
}
```

### CAPTURAS DE PANTALLA:

```
ejercicio3 $ export OMP_DYNAMIC=TRUE
ejercicio3 $ export OMP_NUM_THREADS=6
ejercicio3 $ export OMP_THREAD_LIMIT=12
ejercicio3 $ export OMP_SCHEDULE="dynamic"
ejercicio3 $
```

```
juanka1995@juanka-laptop ~/practicass/AC/Practicass/practica_3/ejercicio3 $ ./scheduled-clauseModificado 16 1
Dentro de la region paralela:
dyn-var: 1
threads-var: 6
thread-limit-var: 12
run-sched-var:
  kind: 2
  modifier: 1
Fuera de la region paralela:
dyn-var: 1
threads-var: 6
thread-limit-var: 12
run-sched-var:
  kind: 2
  modifier: 1
```

**RESPUESTA:**

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 threads) y fuera de la región paralela. Incorporar en su cuaderno de prácticas v

Depto. Arquitectura y Tecnología de Computadores

**RESPUESTA:** Como podemos comprobar el resultado es el mismo tanto dentro como fuera de la región paralela.

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

### CÓDIGO FUENTE: scheduled-clauseModificado4.c

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

```

int main(int argc, char **argv) {
    int i, n=200, chunk, a[n], suma=0;
    if(argc < 3) {
        fprintf(stderr, "\nFalta iteraciones o chunk \n");
        exit(-1);
    }
    n = atoi(argv[1]); if (n>200) n=200; chunk = atoi(argv[2]);

    for (i=0; i<n; i++)        a[i] = i;

    #pragma omp parallel for firstprivate(suma) \
        lastprivate(suma) schedule(dynamic, chunk)
    for (i=0; i<n; i++)
    {
        // suma = suma + a[i];
        // printf(" thread %d suma a[%d]=%d suma=%d \n",
        //         omp_get_thread_num(), i, a[i], suma);
        if(omp_get_thread_num() == 0)
            printf("Dentro de la region paralela:\nnum_threads: %d\nnum_proc:
%d\nin_parallel:
%d\n", omp_get_num_threads(), omp_get_num_procs(), omp_in_parallel());

    }

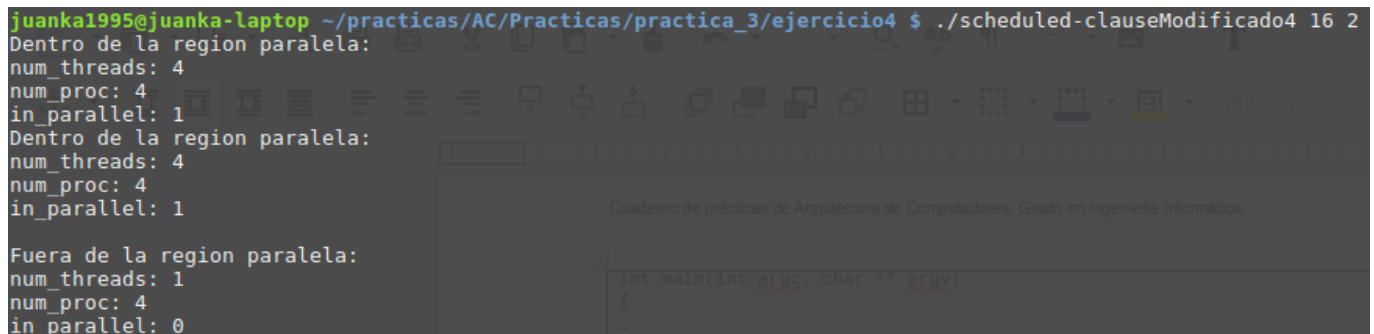
        printf("\nFuera de la region paralela:\nnum_threads: %d\nnum_proc:
%d\nin_parallel:
%d\n", omp_get_num_threads(), omp_get_num_procs(), omp_in_parallel());

        // printf("\nFuera de 'parallel for' suma=%d\n", suma);

    return(0);
}

```

### CAPTURAS DE PANTALLA:

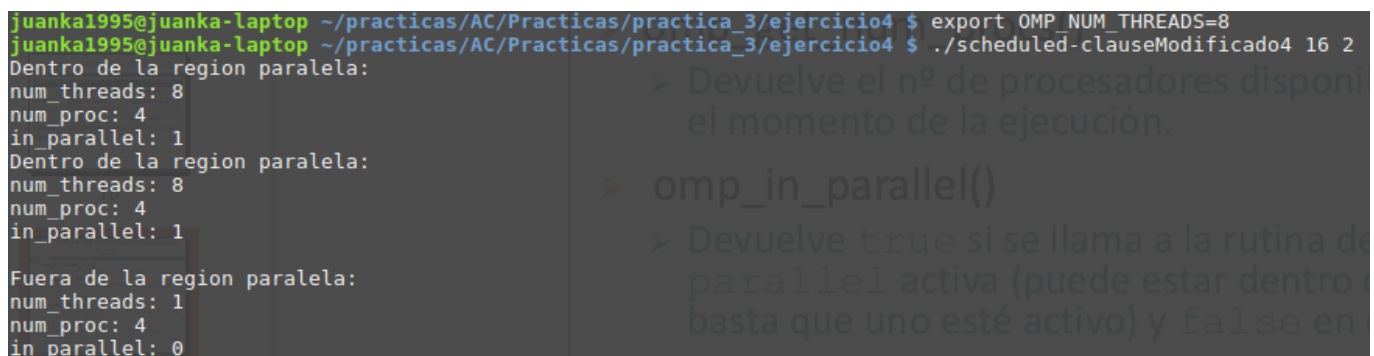


```

juanka1995@juanka-laptop ~/practicass/AC/Practicass/practica_3/ejercicio4 $ ./scheduled-clauseModificado4 16 2
Dentro de la region paralela:
num_threads: 4
num_proc: 4
in_parallel: 1
Dentro de la region paralela:
num_threads: 4
num_proc: 4
in_parallel: 1

Fuera de la region paralela:
num_threads: 1
num_proc: 4
in_parallel: 0

```



```

juanka1995@juanka-laptop ~/practicass/AC/Practicass/practica_3/ejercicio4 $ export OMP_NUM_THREADS=8
juanka1995@juanka-laptop ~/practicass/AC/Practicass/practica_3/ejercicio4 $ ./scheduled-clauseModificado4 16 2
Dentro de la region paralela:
num_threads: 8
num_proc: 4
in_parallel: 1
Dentro de la region paralela:
num_threads: 8
num_proc: 4
in_parallel: 1

Fuera de la region paralela:
num_threads: 1
num_proc: 4
in_parallel: 0

```

**RESPUESTA:**

Las funciones que obtienen distintos valores dependiendo de si están dentro o fuera de la región paralela son **omp\_get\_num\_threads()** y **omp\_in\_parallel()**. Sin embargo la función **omp\_get\_num\_procs()** se mantiene siempre igual.

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

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

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

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

    // Antes de modificar
    omp_get_schedule(&kind, &modifier);
    printf("Antes de modificar:\n\tdyn-var: %d\n\tnthreads-var: %d\n\trun-
    sched-var:\n\tkind: %d\n\tmodifier:
    %d\n", omp_get_dynamic(), omp_get_max_threads(), kind, modifier);

    // static = 1 ; dynmic = 2 ; guided = 3 ; auto = 4

    // Realizamos los cambios
    omp_set_dynamic(3);
    omp_set_num_threads(8);
    omp_set_schedule(3, 2);

    // Despues de modificar
    omp_get_schedule(&kind, &modifier);
    printf("\nDespues de modificar:\n\tdyn-var: %d\n\tnthreads-var: %d\n\trun-
    sched-var:\n\tkind: %d\n\tmodifier:
    %d\n\n", omp_get_dynamic(), omp_get_max_threads(), kind, modifier);

    #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);

return(0);
}
```

### CAPTURAS DE PANTALLA:

```
juanka1995@juanka-laptop ~/practicass/AC/Practicass/practica_3/ejercicio5 $ ./scheduled-clauseModificado5 16 1
Antes de modificar:
  dyn-var: 0
  nthreads-var: 6
  run-sched-var:
  kind: 2
  modifier: 1

Despues de modificar:
  dyn-var: 1
  nthreads-var: 8
  run-sched-var:
  kind: 3
  modifier: 2

thread 1 suma a[1]=1 suma=1
thread 1 suma a[3]=3 suma=4
thread 1 suma a[4]=4 suma=8
thread 1 suma a[5]=5 suma=13
thread 1 suma a[6]=6 suma=19
thread 1 suma a[7]=7 suma=26
thread 1 suma a[8]=8 suma=34
thread 1 suma a[9]=9 suma=43
thread 1 suma a[10]=10 suma=53
thread 1 suma a[11]=11 suma=64
thread 1 suma a[12]=12 suma=76
thread 1 suma a[13]=13 suma=89
thread 1 suma a[14]=14 suma=103
thread 1 suma a[15]=15 suma=118
thread 0 suma a[2]=2 suma=2
thread 2 suma a[0]=0 suma=0
Fuera de 'parallel for' suma=118
```

Las funciones que obtienen distintos valores dependiendo de si están dentro o fuera de paralela son `omp_get_num_threads()` y `omp_in_parallel()`. Sin embargo la `omp_get_num_procs()` se mantiene siempre igual.

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

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

```
/* Tipo de letra Courier new o Liberation Mono. Tamaño 8 o 9 */
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ */
/* INTERLINEADO SENCILLO */

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

int main(int argc, char ** argv)
{
    ...
}
```

```
juanka1995@juanka-laptop ~/practicass/AC/Practicass/practica_3/ejercicio5 $ ./scheduled-clauseModificado5 20 2
Antes de modificar:
  dyn-var: 0
  nthreads-var: 6
  run-sched-var:
  kind: 2
  modifier: 1

Despues de modificar:
  dyn-var: 1
  nthreads-var: 8
  run-sched-var:
  kind: 3
  modifier: 2

thread 3 suma a[4]=4 suma=4
thread 3 suma a[5]=5 suma=9
thread 3 suma a[8]=8 suma=17
thread 0 suma a[6]=6 suma=6
thread 2 suma a[2]=2 suma=2
thread 1 suma a[0]=0 suma=0
thread 1 suma a[1]=1 suma=1
thread 0 suma a[7]=7 suma=13
thread 0 suma a[12]=12 suma=25
thread 0 suma a[13]=13 suma=38
thread 0 suma a[14]=14 suma=52
thread 0 suma a[15]=15 suma=67
thread 0 suma a[16]=16 suma=83
thread 0 suma a[17]=17 suma=100
thread 0 suma a[18]=18 suma=118
thread 0 suma a[19]=19 suma=137
thread 3 suma a[9]=9 suma=26
thread 1 suma a[10]=10 suma=11
thread 1 suma a[11]=11 suma=22
thread 2 suma a[3]=3 suma=5
Fuera de 'parallel for' suma=137
```

**RESPUESTA:**

Resto de ejercicios

6. Implementar un programa secuencial en C que multiplique una matriz triangular vector (use variables dinámicas). Compare el orden de complejidad del código implementado con el código que implementó para el producto matriz por vector.



**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.

**CÓDIGO FUENTE:** pmtv-secuencial.c

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

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

    int **matriz;
    int *vector, *resultado;
    int tam, suma_aux;
    time_t t;

    // Inicializamos la semilla del rand
    srand((unsigned) time(&t));

    if(argc < 2){
        fprintf(stderr, "\nFalta tamaño de filas/columnas\n");
        exit(-1);
    }

    tam = atoi(argv[1]);

    // Reservo memoria
    vector = (int *) malloc(tam*sizeof(int));
    resultado = (int *) malloc(tam*sizeof(int));
    matriz = (int **) malloc(tam*sizeof(int*));

    // Inicializar vector y reservar memoria para matriz
    for (int i = 0; i < tam; i++) {
        matriz[i] = (int *) malloc(tam*sizeof(int));
    }

    // Inicializar matriz a 0
    for (int i = 0; i < tam; i++) {
        for (int j = 0; j < tam; j++) {
            matriz[i][j] = 0;
        }
    }

    // Inicializar valores por encima de diagonal principal de la matriz
    // e inicializar valores del vector
    for (int i = 0; i < tam; i++) {
        vector[i] = rand() % 20;
        for (int j = 0 + i; j < tam; j++) {
```

```

        matriz[i][j] = rand() % 20;
    }
}

// Realizar calculo
for (int i = 0; i < tam; i++) {
    suma_aux = 0;
    for (int j = 0 + i; j < tam; j++) {
        suma_aux += vector[j] * matriz[i][j];
    }
    resultado[i] = suma_aux;
}

// Mostrar resultado
printf("Primer elemento resultado: %d\n", resultado[0]);
printf("Ultimo elemento resultado: %d\n", resultado[tam-1]);

//Liberar memoria
for (int i = 0; i < tam; i++) {
    free(matriz[i]);
}

free(matriz);
free(vector);
free(resultado);

return 0;
}

```

**CAPTURAS DE PANTALLA:**

```

juanka1995@juanka-laptop ~/practicass/AC/Practicass/practica_3/ejercicio6 $ ./ejercicio6 4
Primer elemento resultado: 180
Ultimo elemento resultado: 30
juanka1995@juanka-laptop ~/practicass/AC/Practicass/practica_3/ejercicio6 $ ./ejercicio6 4
Primer elemento resultado: 270
Ultimo elemento resultado: 0
juanka1995@juanka-laptop ~/practicass/AC/Practicass/practica_3/ejercicio6 $ ./ejercicio6 4
Primer elemento resultado: 645
Ultimo elemento resultado: 25
juanka1995@juanka-laptop ~/practicass/AC/Practicass/practica_3/ejercicio6 $ ./ejercicio6 4
Primer elemento resultado: 363
Ultimo elemento resultado: 135
juanka1995@juanka-laptop ~/practicass/AC/Practicass/practica_3/ejercicio6 $ ./ejercicio6 4
Primer elemento resultado: 363
Ultimo elemento resultado: 135
juanka1995@juanka-laptop ~/practicass/AC/Practicass/practica_3/ejercicio6 $ ./ejercicio6 4
Primer elemento resultado: 310
Ultimo elemento resultado: 108

```

**RESPUESTA:** Para no inicializar la matriz y el vector a siempre los mismos valores, he usado la función `rand() % 20` para que me le asigne número comprendidos entre 0 y 19 de forma aleatoria en cada ejecución.

**(ADJUNTAR CÓDIGO FUENTE AL .ZIP)**

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:

La alternativa que ofrece mejores prestaciones tras las pruebas realizadas parece ser el **dynamic**. Esto se debe a que al no ir disminuyendo el tamaño del `chunk` y siempre asignar bloques de un tamaño múltiplo del n.º de iteraciones a las hebras que van terminando el reparto se hace de una forma muy equilibrada, lo que se obtiene como menor tiempo de ejecución en forma de resultados.

a)

`static` → iteraciones divididas de forma equitativa entre el n.º de hebras

`dynamic` → valor 1 por defecto

`guided` → valor 1 por defecto

Esta información ha sido obtenida del siguiente enlace:

<https://computing.llnl.gov/tutorials/openMP/>

b)

$(n.^{\circ} \text{ filas} / n.^{\circ} \text{ threads} * \text{chunk}) * n.^{\circ} \text{ columnas}$

c) En **dynamic** que el trabajo estaría mejor repartido, por lo que cada hebra estaría realizando multiplicaciones más o menos la misma cantidad de tiempo y en **guided** igual que en el `dynamic` pero con la diferencia de que cada vez el tamaño de los bloques será menor.

#### CÓDIGO FUENTE: `pmtv-OpenMP.c`

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

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

    int **matriz;
    int *vector, *resultado;
    int tam, suma_aux, j;
    time_t t;
```

```

double inicio, tiempo;

// Inicializamos la semilla del rand
srand((unsigned) time(&t));

if(argc < 2){
    fprintf(stderr, "\nFalta tamaño de filas/columnas\n");
    exit(-1);
}

tam = atoi(argv[1]);

// Reservo memoria
vector = (int *) malloc(tam*sizeof(int));
resultado = (int *) malloc(tam*sizeof(int));
matriz = (int **) malloc(tam*sizeof(int*));

// Inicializar vector y reservar memoria para matriz
for (int i = 0; i < tam; i++) {
    matriz[i] = (int *) malloc(tam*sizeof(int));
}

// Inicializar matriz a 0
for (int i = 0; i < tam; i++) {
    for (int j = 0; j < tam; j++) {
        matriz[i][j] = 0;
    }
}

// Inicializar valores por encima de diagonal principal de la matriz
// e inicializar valores del vector
for (int i = 0; i < tam; i++) {
    // vector[i] = rand() % 20;
    vector[i] = i + 1;
    for (int j = 0 + i; j < tam; j++) {
        matriz[i][j] = j + 1;
        // matriz[i][j] = rand() % 20;
    }
}

inicio = omp_get_wtime();
// Realizar calculo
#pragma omp parallel for private(suma_aux,j) schedule(runtime)
for (int i = 0; i < tam; i++) {
    suma_aux = 0;
    for (j = 0 + i; j < tam; j++) {
        suma_aux += vector[j] * matriz[i][j];
    }
    resultado[i] = suma_aux;
}
tiempo = omp_get_wtime() - inicio;

// Mostrar resultado
printf("Tiempo empleado: %11.9f\n", tiempo);
printf("Primer elemento resultado: %d\n", resultado[0]);
printf("Ultimo elemento resultado: %d\n", resultado[tam-1]);

//Liberar memoria
for (int i = 0; i < tam; i++) {
    free(matriz[i]);
}

```

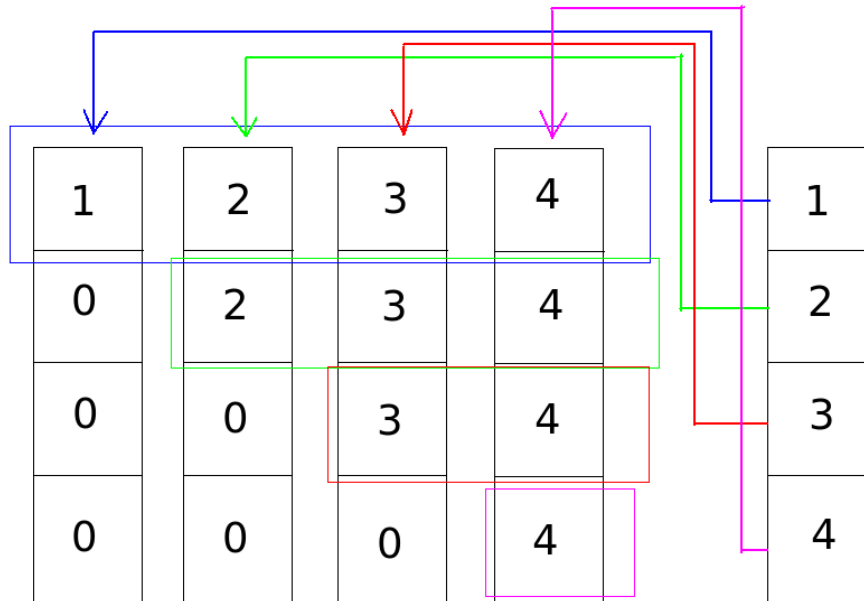
```

free(matriz);
free(vector);
free(resultado);

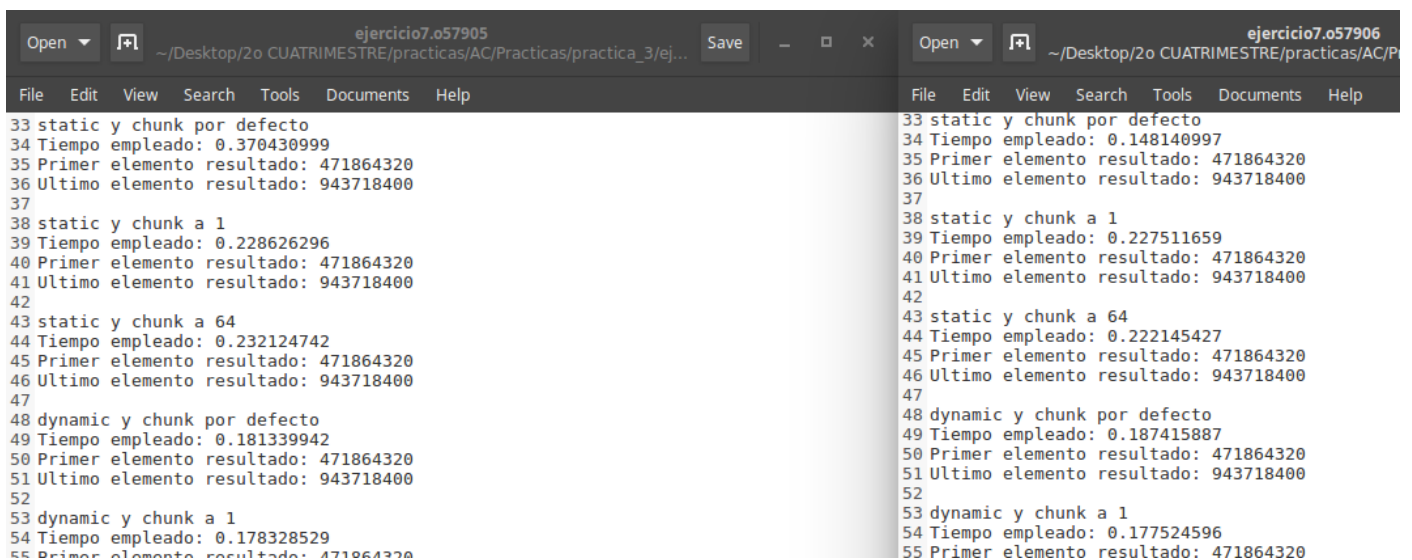
return 0;
}

```

**DESCOMPOSICIÓN DE DOMINIO:** Las distintas hebras se reparten las filas de la matriz y realizan las operaciones necesarias pero evitando las multiplicaciones por 0, ya que son absurdas.



### CAPTURAS DE PANTALLA:



```
[C2estudiante17@atcgrid ~]$ ls -l
total 20
-rwxr-xr-x 1 C2estudiante17 C2estudiante17 13464 May 14 03:05 ejercicio7
-rwxr-xr-x 1 C2estudiante17 C2estudiante17 1487 May 14 03:07 pmvt-OpenMP_atcgrid.sh
[C2estudiante17@atcgrid ~]$ qsub pmvt-OpenMP_atcgrid.sh -q ac
57907.atcgrid
[C2estudiante17@atcgrid ~]$ qstat
Job ID          Name          User          Time Use S Queue
-----
57907.atcgrid   ejercicio7     C2estudiante17  0 R ac
[C2estudiante17@atcgrid ~]$ qstat
Job ID          Name          User          Time Use S Queue
-----
57907.atcgrid   ejercicio7     C2estudiante17  0 R ac
[C2estudiante17@atcgrid ~]$ qstat
Job ID          Name          User          Time Use S Queue
-----
57907.atcgrid   ejercicio7     C2estudiante17  00:00:12 C ac
[C2estudiante17@atcgrid ~]$ ls -l
total 24
-rwxr-xr-x 1 C2estudiante17 C2estudiante17 13464 May 14 03:05 ejercicio7
-rw----- 1 C2estudiante17 C2estudiante17 0 May 14 03:16 ejercicio7.e57907
-rw----- 1 C2estudiante17 C2estudiante17 1603 May 14 03:16 ejercicio7.o57907
-rwxr-xr-x 1 C2estudiante17 C2estudiante17 1487 May 14 03:07 pmvt-OpenMP_atcgrid.sh
```

(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

## TABLA RESULTADOS, SCRIPT Y GRÁFICA ATCGRID

**SCRIPT:** pmvt-OpenMP\_atcgrid.sh

```
#!/bin/bash
#Se asigna al trabajo el nombre ejercicio7
#PBS -N ejercicio7
#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

tamano=30720

export OMP_SCHEDULE="static"
echo "static y chunk por defecto"
$PBS_O_WORKDIR/ejercicio7 $tamano

export OMP_SCHEDULE="static,1"
echo "static y chunk a 1"
$PBS_O_WORKDIR/ejercicio7 $tamano

export OMP_SCHEDULE="static,64"
echo "static y chunk a 64"
$PBS_O_WORKDIR/ejercicio7 $tamano

export OMP_SCHEDULE="dynamic"
echo "dynamic y chunk por defecto"
$PBS_O_WORKDIR/ejercicio7 $tamano
```

```

export OMP_SCHEDULE="dynamic,1"
echo "dynamic y chunk a 1"
$PBS_O_WORKDIR/ejercicio7 $tamanio

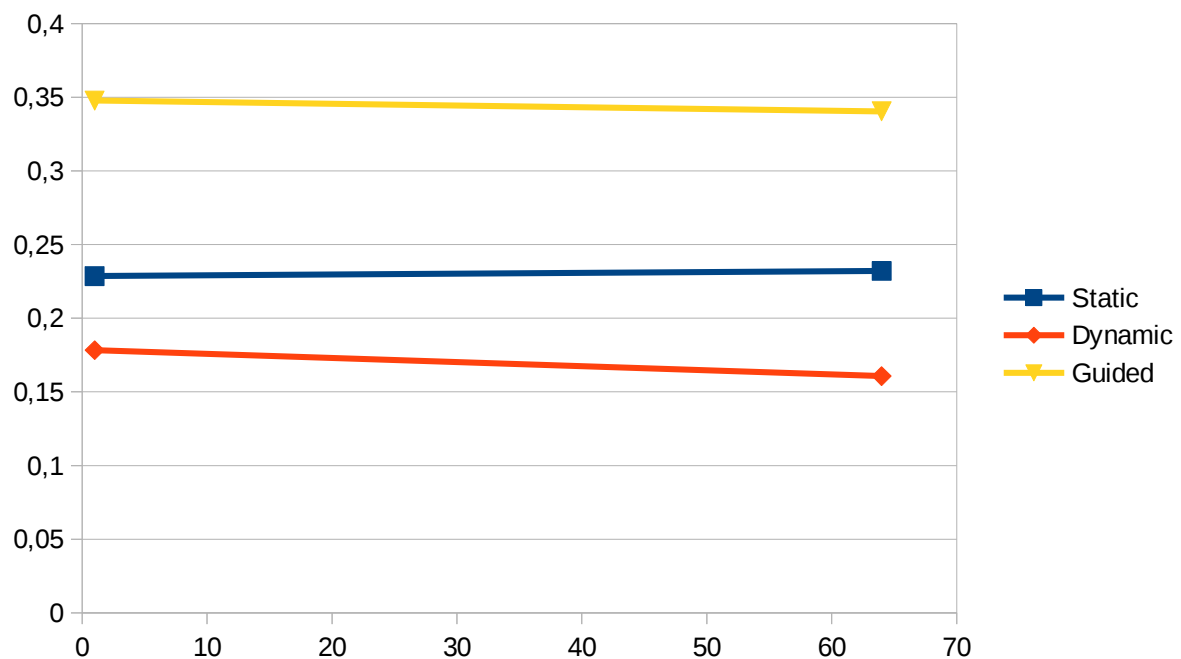
export OMP_SCHEDULE="dynamic,64"
echo "dynamic y chunk a 64"
$PBS_O_WORKDIR/ejercicio7 $tamanio

export OMP_SCHEDULE="guided"
echo "guided y chunk por defecto"
$PBS_O_WORKDIR/ejercicio7 $tamanio

export OMP_SCHEDULE="guided,1"
echo "guided y chunk a 1"
$PBS_O_WORKDIR/ejercicio7 $tamanio

export OMP_SCHEDULE="guided,64"
echo "guided y chunk a 64"
$PBS_O_WORKDIR/ejercicio7 $tamanio

```



**Tabla 3.** Tiempos de ejecución de la versión paralela del producto de una matriz triangular por un vector  $r$  para vectores de tamaño  $N=30720$ , 12 threads

Chunk	Static	Dynamic	Guided
por defecto	0.370430999	0.181339942	0.340246208
1	0.228626296	0.178328529	0.347764440
64	0.232124742	0.160726827	0.340446655
Chunk	Static	Dynamic	Guided
por defecto	0.148140997	0.187415887	0.330741532

1	0.227511659	0.177524596	0.368986163
64	0.222145427	0.166358888	0.345936902

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

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

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

**CÓDIGO FUENTE:** pmm-secuencial.c

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

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

    int **matrizA, **matrizB, **resultado;
    int tam;
    time_t t;

    // Inicializamos la semilla del rand
    srand((unsigned) time(&t));

    if(argc < 2){
        fprintf(stderr, "\nFalta tamaño de filas/columnas\n");
        exit(-1);
    }

    tam = atoi(argv[1]);

    // Reservo memoria
    matrizA = (int **) malloc(tam*sizeof(int*));
    matrizB = (int **) malloc(tam*sizeof(int*));
    resultado = (int **) malloc(tam*sizeof(int*));

    // Reservo memoria para matriz
    for (int i = 0; i < tam; i++) {
        matrizA[i] = (int *) malloc(tam*sizeof(int));
        matrizB[i] = (int *) malloc(tam*sizeof(int));
        resultado[i] = (int *) malloc(tam*sizeof(int));
    }

    // Inicializar matrices
    for (int i = 0; i < tam; i++) {
        for (int j = 0; j < tam; j++) {
            matrizA[i][j] = rand() % 10;
            matrizB[i][j] = rand() % 10;
        }
    }

    // Realizar calculo matrizA x matrizB
    for (int i = 0; i < tam; i++) {
        for (int j = 0; j < tam; j++) {
            for (int k = 0; k < tam; k++) {
                resultado[i][j] += matrizA[i][k] * matrizB[k][j];
            }
        }
    }
}
```



```

    }
}
}

// Mostrar resultado
printf("Primer elemento resultado: %d\n", resultado[0][0]);
printf("Ultimo elemento resultado: %d\n", resultado[tam-1][tam-1]);

// Liberar memoria
for (int i = 0; i < tam; i++) {
    free(matrizA[i]);
    free(matrizB[i]);
    free(resultado[i]);
}

free(matrizA);
free(matrizB);
free(resultado);

return 0;
}

```

**RESPUESTA:** Para no inicializar la matriz y el vector a siempre los mismos valores, he usado la función **rand() % 10** para que me le asigne número comprendidos entre 0 y 9 de forma aleatoria en cada ejecución.

#### CAPTURAS DE PANTALLA:

```

juanka1995@juanka-laptop ~/practicas/AC/Practicas/practica_3/ejercicio8 $ ./ejercicio8 3
Primer elemento resultado: 113
Ultimo elemento resultado: 27
juanka1995@juanka-laptop ~/practicas/AC/Practicas/practica_3/ejercicio8 $ ./ejercicio8 3
Primer elemento resultado: 144
Ultimo elemento resultado: 40
juanka1995@juanka-laptop ~/practicas/AC/Practicas/practica_3/ejercicio8 $ ./ejercicio8 3
Primer elemento resultado: 62
Ultimo elemento resultado: 124
juanka1995@juanka-laptop ~/practicas/AC/Practicas/practica_3/ejercicio8 $ ./ejercicio8 3
Primer elemento resultado: 70
Ultimo elemento resultado: 118

```

#### (ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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:

##### CÓDIGO FUENTE: pmm-OpenMP.c

```

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#ifdef _OPENMP
#include <omp.h>

```

```

#else
    #define omp_get_thread_num() 0
#endif

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

    int **matrizA, **matrizB, **resultado;
    int tam,j,k;
    time_t t;
    double inicio, tiempo;

    // Inicializamos la semilla del rand
    srand((unsigned) time(&t));

    if(argc < 2){
        fprintf(stderr, "\nFalta tamaño de filas/columnas\n");
        exit(-1);
    }

    tam = atoi(argv[1]);

    // Reservo memoria
    matrizA = (int **) malloc(tam*sizeof(int*));
    matrizB = (int **) malloc(tam*sizeof(int*));
    resultado = (int **) malloc(tam*sizeof(int*));

    // Reservo memoria para matriz
    for (int i = 0; i < tam; i++) {
        matrizA[i] = (int *) malloc(tam*sizeof(int));
        matrizB[i] = (int *) malloc(tam*sizeof(int));
        resultado[i] = (int *) malloc(tam*sizeof(int));
    }

    // Inicializar matrices
    for (int i = 0; i < tam; i++) {
        for (int j = 0; j < tam; j++) {
            matrizA[i][j] = rand() % 10;
            matrizB[i][j] = rand() % 10;
        }
    }

    inicio = omp_get_wtime();
    // Realizar calculo matrizA x matrizB
    #pragma omp parallel for private(j,k) schedule(runtime)
    for (int i = 0; i < tam; i++) {
        for (j = 0; j < tam; j++) {
            for (k = 0; k < tam; k++) {
                resultado[i][j] += matrizA[i][k] * matrizB[k][j];
            }
        }
    }
    tiempo = omp_get_wtime() - inicio;

    // Mostrar resultado
    printf("Tiempo empleado: %11.9f\n",tiempo);
    printf("Primer elemento resultado: %d\n",resultado[0][0]);
    printf("Ultimo elemento resultado: %d\n",resultado[tam-1][tam-1]);

    // Liberar memoria
    for (int i = 0; i < tam; i++) {
        free(matrizA[i]);
        free(matrizB[i]);
    }
}

```

```
    free(resultado[i]);  
}  
  
free(matrizA);  
free(matrizB);  
free(resultado);  
  
return 0;  
}
```

### CAPTURAS DE PANTALLA:

#### SECUENCIAL

```
juanka1995@juanka-desktop ~/practicas/AC/Practicas/practica_3/ejercicio9 $ ./ejercicio8 1000  
Tiempo empleado: 19.957009898  
Primer elemento resultado: 19507  
Ultimo elemento resultado: 20537  
juanka1995@juanka-desktop ~/practicas/AC/Practicas/practica_3/ejercicio9 $ ./ejercicio8 500  
Tiempo empleado: 1.641338126  
Primer elemento resultado: 9267  
Ultimo elemento resultado: 10887  
juanka1995@juanka-desktop ~/practicas/AC/Practicas/practica_3/ejercicio9 $ ./ejercicio8 1500  
Tiempo empleado: 66.033045670  
Primer elemento resultado: 29735  
Ultimo elemento resultado: 30366
```

#### PARALELO

```
juanka1995@juanka-desktop ~/practicas/AC/Practicas/practica_3/ejercicio9 $ ./ejercicio9 1000  
Tiempo empleado: 7.572825053  
Primer elemento resultado: 19507  
Ultimo elemento resultado: 20537  
juanka1995@juanka-desktop ~/practicas/AC/Practicas/practica_3/ejercicio9 $ ./ejercicio9 500  
Tiempo empleado: 0.688657416  
Primer elemento resultado: 9267  
Ultimo elemento resultado: 10887  
juanka1995@juanka-desktop ~/practicas/AC/Practicas/practica_3/ejercicio9 $ ./ejercicio9 1500  
Tiempo empleado: 23.321415213  
Primer elemento resultado: 29735  
Ultimo elemento resultado: 30366
```

(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

## ESTUDIO DE ESCALABILIDAD EN ATCGRID:

### SCRIPT: pmm-OpenMP\_atcgridA.sh

```
#!/bin/bash
#Se asigna al trabajo el nombre ejercicio10
#PBS -N ejercicio10
#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

tamano=350

echo -e "TAMAÑO $tamano:"
for ((N=1;N<25;N=N+1))
do
    export OMP_NUM_THREADS=$N
    echo -e "\nNº hebras: $N\n"
    $PBS_O_WORKDIR/ejercicio10 $tamano
done
```

### SCRIPT: pmm-OpenMP\_atcgridB.sh

```
#!/bin/bash
#Se asigna al trabajo el nombre ejercicio10
#PBS -N ejercicio10
#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

tamano=950

echo -e "TAMAÑO $tamano:"
for ((N=1;N<25;N=N+1))
do
    export OMP_NUM_THREADS=$N
    echo -e "\nNº hebras: $N\n"
```

```
$PBS_O_WORKDIR/ejercicio10 $tamanio
done
```

**CAPTURAS DE PANTALLA EN ATCGRID:**

TAMAÑO 350x350

```
ejercicio10 pmm-OpenMP_atcgridA.sh pmm-OpenMP_atcgridB.sh
[C2estudiante17@atcgrid ~]$ qsub pmm-OpenMP_atcgridA.sh -q ac
57972.atcgrid
[C2estudiante17@atcgrid ~]$ qstat
Job ID          Name          User          Time Use S Queue
-----
57972.atcgrid    ejercicio10    C2estudiante17 00:00:00 C ac
[C2estudiante17@atcgrid ~]$ ls -l
total 28
-rwxr-xr-x 1 C2estudiante17 C2estudiante17 13552 May 14 22:44 ejercicio10
-rw----- 1 C2estudiante17 C2estudiante17      0 May 14 22:44 ejercicio10.e57972
-rw----- 1 C2estudiante17 C2estudiante17  3120 May 14 22:44 ejercicio10.o57972
-rw-r--r-- 1 C2estudiante17 C2estudiante17   737 May 14 22:43 pmm-OpenMP_atcgridA.sh
-rw-r--r-- 1 C2estudiante17 C2estudiante17   738 May 14 22:43 pmm-OpenMP_atcgridB.sh
[C2estudiante17@atcgrid ~]$ cat ejercicio10.e57972
[C2estudiante17@atcgrid ~]$ cat ejercicio10.o57972
Id. usuario del trabajo: C2estudiante17
Id. del trabajo: 57972.atcgrid
Nombre del trabajo especificado por usuario: ejercicio10
Nodo que ejecuta qsub: atcgrid
Directorio en el que se ha ejecutado qsub: /home/C2estudiante17
Cola: ac
```

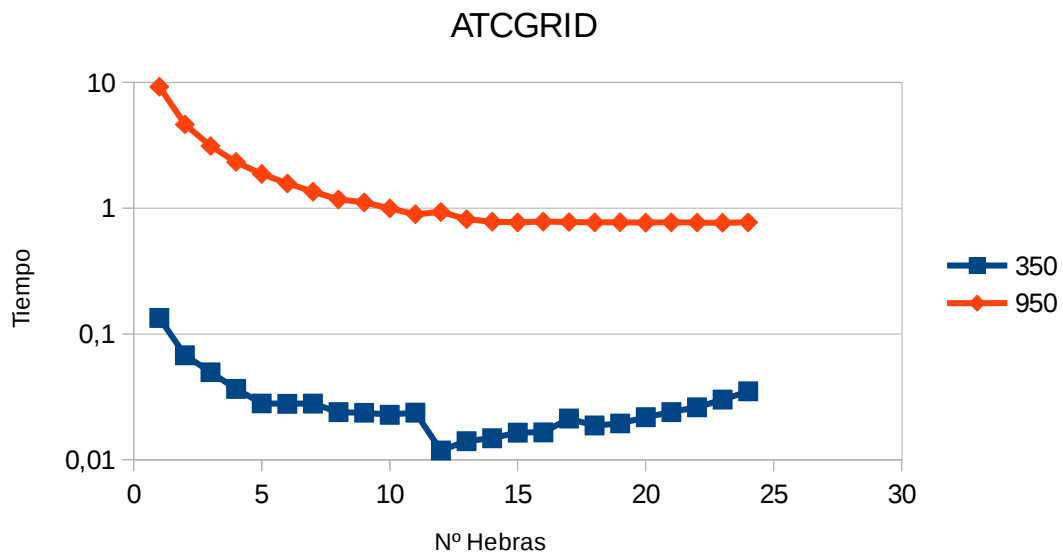
TAMAÑO 950x950

```
ejercicio10 pmm-OpenMP_atcgridA.sh pmm-OpenMP_atcgridB.sh
[C2estudiante17@atcgrid ~]$ qsub pmm-OpenMP_atcgridB.sh -q ac
57979.atcgrid
[C2estudiante17@atcgrid ~]$ qstat
Job ID          Name          User          Time Use S Queue
-----
57979.atcgrid    ejercicio10    C2estudiante17      0 R ac
[C2estudiante17@atcgrid ~]$ qstat
Job ID          Name          User          Time Use S Queue
-----
57979.atcgrid    ejercicio10    C2estudiante17      0 R ac
[C2estudiante17@atcgrid ~]$ qstat
Job ID          Name          User          Time Use S Queue
-----
57979.atcgrid    ejercicio10    C2estudiante17      0 R ac
[C2estudiante17@atcgrid ~]$ ls
ejercicio10.e57979 ejercicio10.o57979 pmm-OpenMP_atcgridA.sh pmm-OpenMP_atcgridB.sh
[C2estudiante17@atcgrid ~]$ cat ejercicio10.e57979
[C2estudiante17@atcgrid ~]$ cat ejercicio10.o57979
Id. usuario del trabajo: C2estudiante17
Id. del trabajo: 57979.atcgrid
Nombre del trabajo especificado por usuario: ejercicio10
Nodo que ejecuta qsub: atcgrid
Directorio en el que se ha ejecutado qsub: /home/C2estudiante17
Cola: ac
```

**TABLA DE TIEMPOS EN ATCGRID:**

N,º Hebras	350	950
1	0,133971568	9,241073657
2	0,067684256	4,634154126
3	0,049634565	3,118838429
4	0,03662036	2,333572313
5	0,028055873	1,871431243
6	0,027845208	1,572279904
7	0,027997985	1,350710604
8	0,023931827	1,171902381
9	0,023624238	1,111560985
10	0,022795338	0,995754037
11	0,02370885	0,895811066
12	0,011838716	0,932002671
13	0,014067359	0,817097154
14	0,014837272	0,780149419
15	0,016449094	0,771458391
16	0,016524721	0,781760022
17	0,021279063	0,775228854
18	0,018728584	0,771689586
19	0,019428264	0,773524232
20	0,021785956	0,766549058
21	0,023968939	0,769867454
22	0,026144251	0,768001765
23	0,030101694	0,766097877
24	0,034991164	0,771005128

**GRAFICA EN ATCGRID:**



**ESTUDIO DE ESCALABILIDAD EN PCLOCAL:****SCRIPT:** miMacro.sh

```
#!/bin/bash

tamano1=350
tamano2=950

salida1="tamaño350.dat"
salida2="tamaño950.dat"

rm -f salida1 salida2

echo -e "TAMAÑO $tamano1:" >> $salida1
for ((N=1;N<5;N=N+1))
do
    export OMP_NUM_THREADS=$N
    echo -e "\nNº hebras: $N\n" >> $salida1
    ./ejercicio10 $tamano1 >> $salida1
done

echo -e "TAMAÑO $tamano2:" >> $salida2
for ((N=1;N<5;N=N+1))
do
    export OMP_NUM_THREADS=$N
    echo -e "\nNº hebras: $N\n" >> $salida2
    ./ejercicio10 $tamano2 >> $salida2
done
```

**CAPTURAS DE PANTALLA EN PC\_LOCAL:**

```
juanka1995@juanka-desktop ~/practicass/AC/Practicass/practica_3/ejercicio10 $ chmod +x miMacro.sh
juanka1995@juanka-desktop ~/practicass/AC/Practicass/practica_3/ejercicio10 $ ls -l Aprendizaje
total 52
drwxr-xr-x 4 juanka1995 juanka1995 4096 May 14 23:19 atcgrid
-rwxr-xr-x 1 juanka1995 juanka1995 13552 May 14 22:42 ejercicio10
-rw-r--r-- 1 juanka1995 juanka1995 1770 May 14 22:21 ejercicio10.c
drwxr-xr-x 2 juanka1995 juanka1995 4096 May 14 23:34 local
-rwxr-xr-x 1 juanka1995 juanka1995 475 May 14 23:37 miMacro.sh
juanka1995@juanka-desktop ~/practicass/AC/Practicass/practica_3/ejercicio10 $ ./miMacro.sh
juanka1995@juanka-desktop ~/practicass/AC/Practicass/practica_3/ejercicio10 $ ls -l tamaño*
-rw-r--r-- 1 juanka1995 juanka1995 449 May 14 23:38 tamaño350.dat
-rw-r--r-- 1 juanka1995 juanka1995 457 May 14 23:39 tamaño950.dat
```

**TABLA DE TIEMPOS EN PC\_LOCAL:**

N,º Hebras	350	950
1	0,132934449	3,843261609
2	0,069644761	2,079071824
3	0,046497681	2,242754834
4	0,036425575	2,490022734



**GRAFICA EN PC\_LOCAL:**

