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): Jorge Gangoso Klock
Grupo de prácticas: D2
Fecha de entrega: 04/05/2020
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

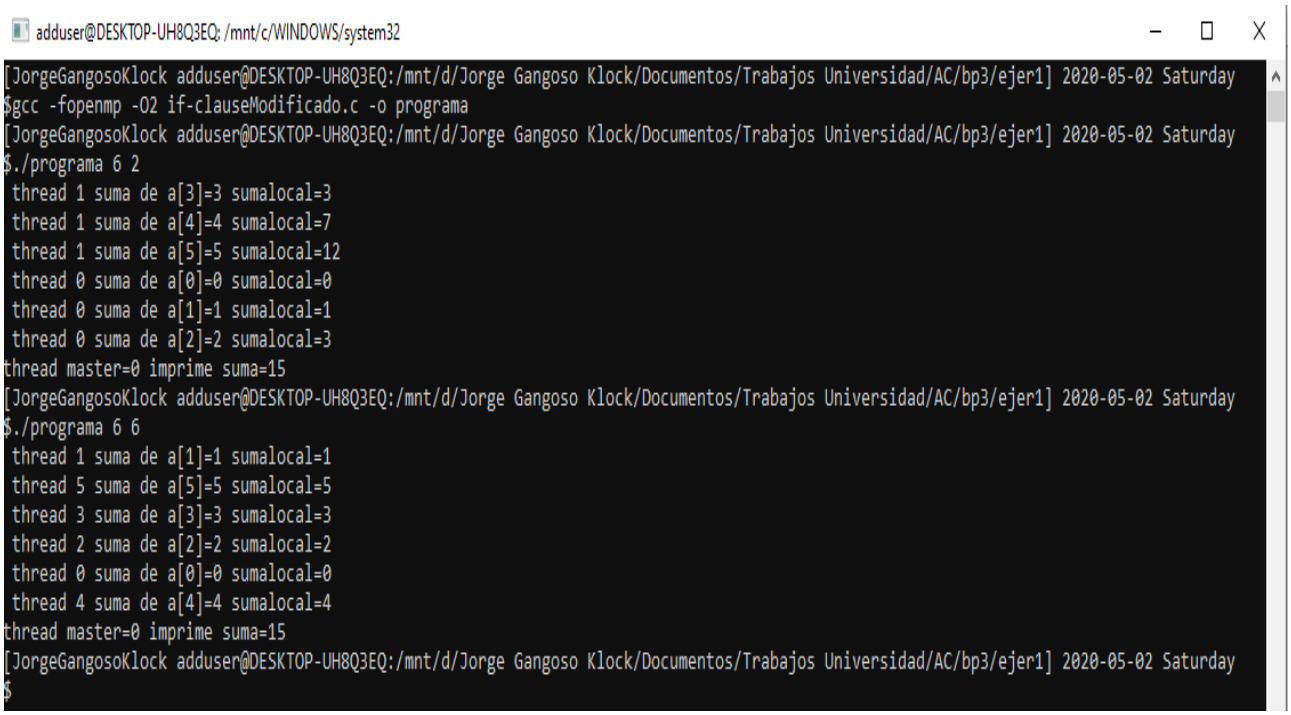
1. Usar la cláusula `num_threads(x)` en el ejemplo del seminario `if_clause.c`, y añadir un parámetro de entrada al programa que fije el valor `x` que se va a usar en la cláusula. Incorporar en el cuaderno de trabajo de esta práctica volcados de pantalla con ejemplos de ejecución que ilustren la funcionalidad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

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

if-clauseModificado.c

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

    n = atoi(argv[1]); if (n>20) n=20;
    for (i=0; i<n; i++) {
        a[i] = i;
    }
#pragma omp parallel num_threads(atoi(argv[2])) if(n>4) 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);
    }
}
```

**CAPTURAS DE PANTALLA:**




**RESPUESTA**: Como se puede observar, mediante el uso de la cláusula, podemos ejecutar un mismo código con un distinto número de threads, permitiendo al usuario elegir entre mayor rendimiento o menor consumo de threads por ejemplo.

**2. (a)** Rellenar la Tabla 1 (se debe poner en la tabla el id del *thread* que ejecuta cada iteración) ejecutando los ejemplos del seminario `schedule-clause.c`, `scheduled-clause.c` y `scheduleg-clause.c` con dos *threads* (0,1) y unas entradas de:

- iteraciones: 16 (0,…15)
- chunck= 1, 2 y 4

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

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

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

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

**RESPUESTA**: Mientras que static siempre va asignando iteraciones según la granularidad a los threads en orden, la versión dynamic asigna al thread más rápido, se ve claramente en la tabla 1 por ejemplo como el thread 1 realiza muchas más iteraciones que el thread 0. Finalmente guided asigna una mayor cantidad de iteraciones al primer thread que llega. En la tabla 2 vemos como (en la granularidad 1) el primer thread en iterar es el #2 y recibe 8 iteraciones, más del doble que realiza el thread #0 que es el segundo en llegar y sólo recibe 3.

**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`

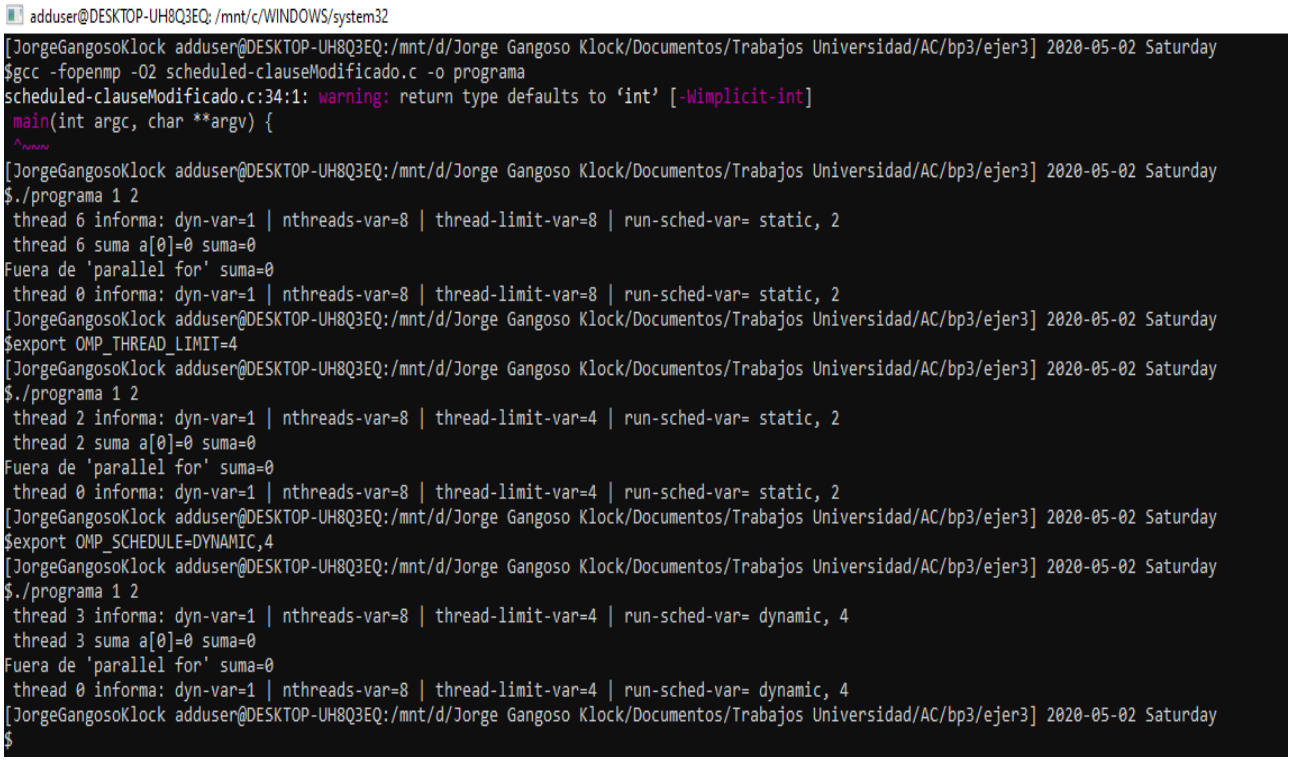
scheduled-clauseModificado.c

```c
#include <stdio.h>
#include <stdlib.h>
#ifdef _OPENMP
...
#else
#define omp_get_thread_num() 0
#endif
void mostrarDatos() {...}
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++)
    {
        if(i==0)
        {
            mostrarDatos();
        }
        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);
    mostrarDatos();

}
```

scheduled-clauseModificado.c

```c
void mostrarDatos()
{
    omp_sched_t tipo;
    int modificador;
    omp_get_schedule(&tipo, &modificador);
    printf(" thread %d informa: dyn-var=%d | nthreads-var=%d | thread-limit-var=%d | run-sched-var= ",
           omp_get_thread_num(), omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit());
    switch(tipo)
    {
        case omp_sched_static:
            printf("static, %d\n", modificador);
            break;
        case omp_sched_dynamic:
            printf("dynamic, %d\n", modificador);
            break;
        case omp_sched_guided:
            printf("guided, %d\n", modificador);
            break;
        case omp_sched_auto:
            printf("auto, %d\n", modificador);
            break;
        default:
            printf("other (implementation specific)\n");
            break;
    }
```

**CAPTURAS DE PANTALLA:**




**RESPUESTA**: Se puede ver como se imprime lo mismo, dentro y fuera del parallel, la causa de ésto es que schedule() no modifica las variables de control, únicamente se altera el funcionamiento de forma temporal dentro del bucle en el que se utilizan.

**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`

```c
#include <stdio.h>
#include <stdlib.h>
#ifdef _OPENMP
#include <omp.h>
#else
#define omp_get_thread_num() 0
#endif
void mostrarDatos()
{
    printf(" thread %d informa: num-threads=%d | num-procs=%d | in-parallel=%d\n",
           omp_get_thread_num(), omp_get_num_threads(), omp_get_num_procs(), omp_in_parallel());
}
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++)
    {
        if(i==0)
        {
            mostrarDatos();
        }
        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);
    mostrarDatos();
}
```

**CAPTURAS DE PANTALLA:**

```
adduser@DESKTOP-UH8Q3EQ: /mnt/c/WINDOWS/system32
[JorgeGangosoKlock adduser@DESKTOP-UH8Q3EQ:/mnt/d/Jorge Gangoso Klock/Documentos/Trabajos Universidad/AC/bp3/ejer4] 2020-05-02 Saturday
$gcc -fopenmp -O2 scheduled-clauseModificado4.c -o programa
scheduled-clauseModificado4.c:13:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
 main(int argc, char **argv) {
 ^~~~
[JorgeGangosoKlock adduser@DESKTOP-UH8Q3EQ:/mnt/d/Jorge Gangoso Klock/Documentos/Trabajos Universidad/AC/bp3/ejer4] 2020-05-02 Saturday
$./programa 2 1
 thread 3 informa: num-threads=4 | num-procs=12 | in-parallel=1
 thread 3 suma a[0]=0 suma=0
 thread 0 suma a[1]=1 suma=1
Fuera de 'parallel for' suma=1
 thread 0 informa: num-threads=1 | num-procs=12 | in-parallel=0
[JorgeGangosoKlock adduser@DESKTOP-UH8Q3EQ:/mnt/d/Jorge Gangoso Klock/Documentos/Trabajos Universidad/AC/bp3/ejer4] 2020-05-02 Saturday
$
```

**RESPUESTA**: Como es de esperar, la cantidad de threads activo pasa a ser, 1 tras la región paralela, mientras que dentro de ésta tendrá tantas como estén especificadas. Evidentemente, dentro de la región parallel la función in_parallel() devuelve true, y fuera devuelve false. La cantidad de procesos disponibles para el programa es la misma en todo momento.

**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`

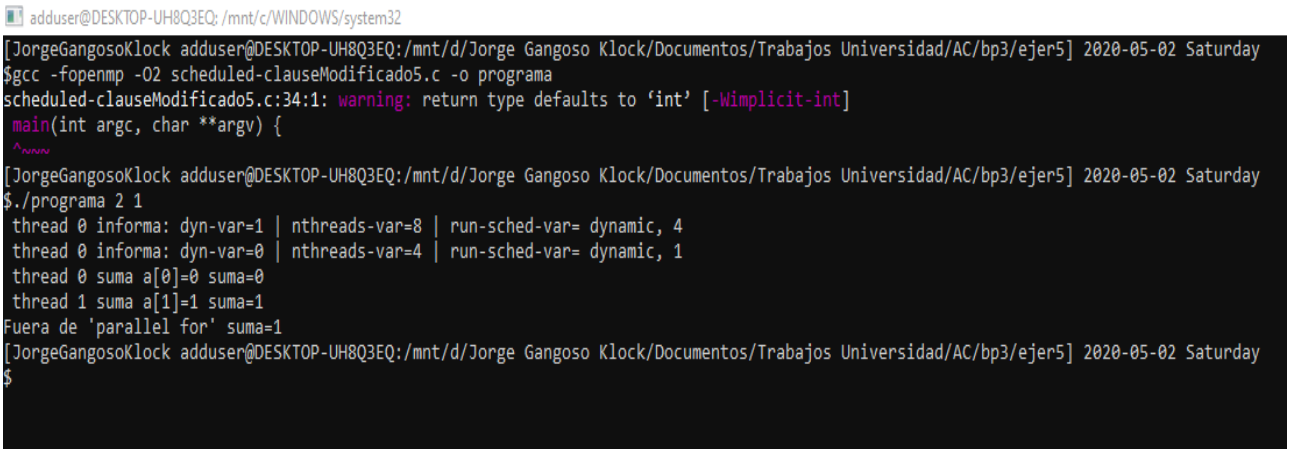
scheduled-clauseModificado5.c

```c
#include <stdio.h>
#include <stdlib.h>
#ifdef _OPENMP
#include <omp.h>
#else
#define omp_get_thread_num() 0
#endif
void mostrarDatos() {...}
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;
    mostrarDatos();
    omp_set_dynamic(0);
    omp_set_num_threads(4);
    omp_set_schedule(omp_sched_dynamic, chunk);
    mostrarDatos();
#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);

}
```

scheduled-clauseModificado5.c

```c
void mostrarDatos()
{
    omp_sched_t tipo;
    int modificador;
    omp_get_schedule(&tipo, &modificador);
    printf(" thread %d informa: dyn-var=%d | nthreads-var=%d | run-sched-var= ",
           omp_get_thread_num(), omp_get_dynamic(), omp_get_max_threads());
    switch(tipo)
    {
        case omp_sched_static:
            printf("static, %d\n", modificador);
            break;
        case omp_sched_dynamic:
            printf("dynamic, %d\n", modificador);
            break;
        case omp_sched_guided:
            printf("guided, %d\n", modificador);
            break;
        case omp_sched_auto:
            printf("auto, %d\n", modificador);
            break;
        default:
            printf("other (implementation specific)\n");
            break;
    }
}
```

**CAPTURAS DE PANTALLA:**




**RESPUESTA**: Podemos ver cómo se modifican, sin embargo, el cambio sólo dura en tiempo de ejecución, si realizamos la ejecución múltiples veces podemos comprobar como al comienzo de cada una, las variables siempre toman el mismo valor, que es el asignado mediante export X=value

## 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.

Complejidad: El programa inicial que multiplicaba una matriz por un vector realizaba un doble bucle anidado de N iteraciones cada uno, siendo la complejidad $O(n^2)$. Esta versión, aunque también tiene dos bucles, en el segundo sólo realiza N/2 iteraciones de media (realiza N en la primera iteración y 0 en la última, gradualmente reduciéndose en 1 la cantidad de iteraciones que debe realizar)

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

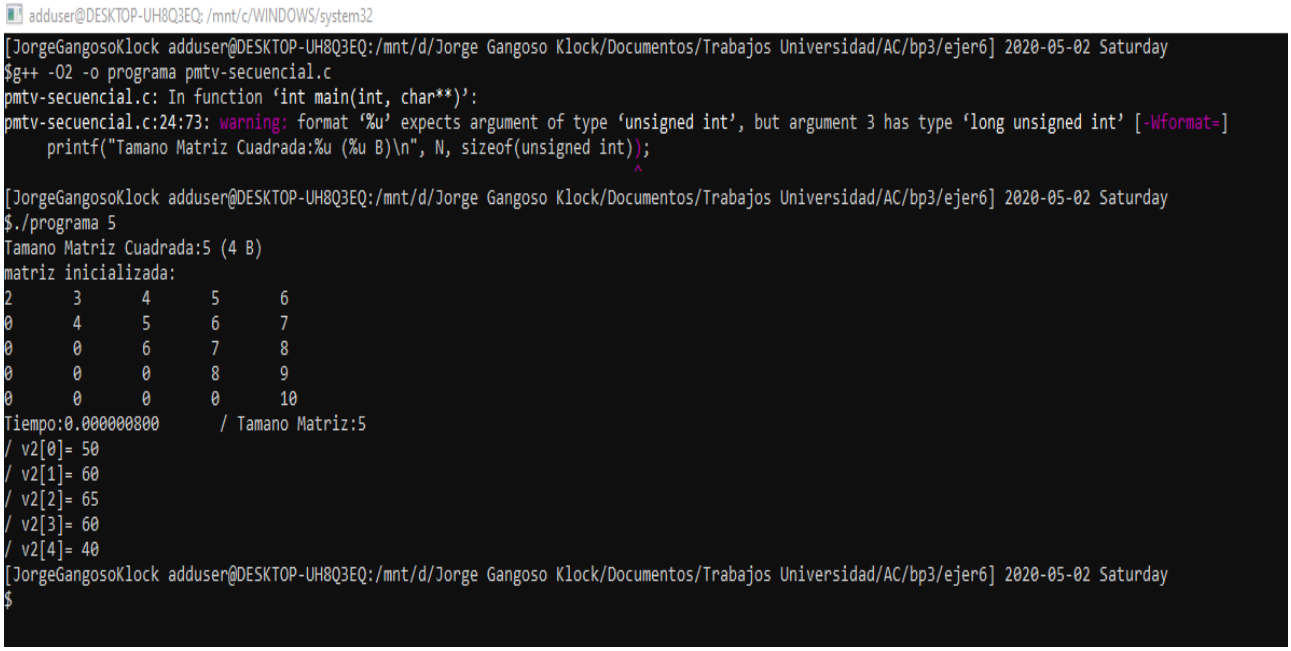
pmtv-secuencial.c

```c
//gcc -fopenmp -O2 pmtv-secuencial.c -o programa //Si usara OpenMP
//g++ -O2 -o programa pmtv-secuencial.c
#include <stdlib.h> // biblioteca con funciones atoi(), malloc() y free()
#include <stdio.h>  // biblioteca donde se encuentra la funcion printf()
#include <time.h>   // biblioteca donde se encuentra la funcion clock_gettime()

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

    if (argc < 2) {
        printf("Usar ./programa <num_filas>\n");
        exit(-1);
    }
    unsigned int N = atoi(argv[1]); //Matriz sera de tamaño N*N

    //Crear vector que multiplica a la matriz
    int *v1;
    v1 = (int*) malloc(N*sizeof(int)); // malloc necesita el tamano en bytes
     if (v1 == NULL)
     {
       printf("No hay suficiente espacio para el vector \n");
       exit(-2);
     }

    printf("Tamano Matriz Cuadrada:%u (%u B)\n", N, sizeof(unsigned int));
    //Inicializar vector
    for (int i = 0; i < N; i++) {
        v1[i] = i;
    }
    //Inicializar vector solucion
    int v2[N];
    for(int i=0; i<N; i++)
    {
        v2[i]=0;
    }
    //Inicializar Matriz
    int **M = (int **)malloc(N*sizeof(int*));
```

pmtv-secuencial.c

```
37      for(int i=0; i<N; i++)
38      M[i] = (int*) malloc(N*sizeof(int)); // malloc necesita el tamano en bytes
39      for (int i = 0; i < N; i++) {
40          for (int j = 0; j < N; j++) {
41              if(j<i)
42                  M[i][j] = 0;
43              else
44                  M[i][j] = (i+1) + (j+1);
45          }
46      }
47      //Mostrar matriz
48      if(N<=12)
49      {
50          printf("matriz inicializada: \n");
51          for (int i = 0; i < N; i++) {
52              for (int j = 0; j < N; j++) {
53                  printf("%d \t", M[i][j]);
54              }
55              printf("\n");
56          }
57      }
58      struct timespec cgt1, cgt2;
59      double ncgt; //para tiempo de ejecucion
60      clock_gettime(CLOCK_REALTIME,&cgt1);
61      //Inicio operacion
62      for(int i=0; i<N; i++)
63      {
64          for(int k=i; k<N; k++)
65          {
66              v2[i]+=M[i][k]*v1[k];
67          }
68      }
69      //Fin operacion
70      clock_gettime(CLOCK_REALTIME,&cgt2);
71      ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+(double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));
72      //Imprimir resultado del producto y el tiempo de ejecucion
73      if (N < 12) {
74          printf("Tiempo:%11.9f\t / Tamano Matriz:%u\n", ncgt, N);
75          for (int i = 0; i < N; i++) {
76              printf("/ v2[%d]= %d\n", i, v2[i]);
77          }
78      }
79      else
80      {
81          printf("Tiempo:%11.9f\t / Tamano Matriz:%u", ncgt,N);
82          printf("v2[0]=%d / / v2[%d]=%d /\n", v2[0],N,v2[N]);
83      }
84  }
```

**CAPTURAS DE PANTALLA:**




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) Por defecto, el tamaño del chunk para static es tal que se le asigne máximo un chunk a cada thread, es decir mínimo: N/n_threads y es 1 para los otros dos tipos de schedule

b)Con tamaño de chunk 1: Los threads que reciban las primeras filas realizarán más trabajo que las demás, difuminándose esta diferencia para valores más altos. (para tamaño de matriz 15.360 la diferencia es de 14.000 operaciones)

Con tamaño de chunk 64: La diferencia se amplia, al repartirse las filas de mayor peso en los primeros threads ya que cada thread recibe 64 filas, la primera thread recibe las 64 primeras filas, que son las que más peso tienen. (para tamaño de matriz 15360 la diferencia es de 901.120 operaciones, la diferencia es 60 veces mayor que con el chunk=1)

c) Se vuelve mucho más dispar, sobre todo con tamaños de chunk superiores, ya que nada garantiza lo libre que pueda estar un thread con respecto a otro, la diferencia de velocidades a las que acabaran los chunks asignados no es previsible. La organización con más diferencia de operaciones entre un thread y otro es la gestión static con chunk=default, teniendo diferencia de varios millones de operaciones.

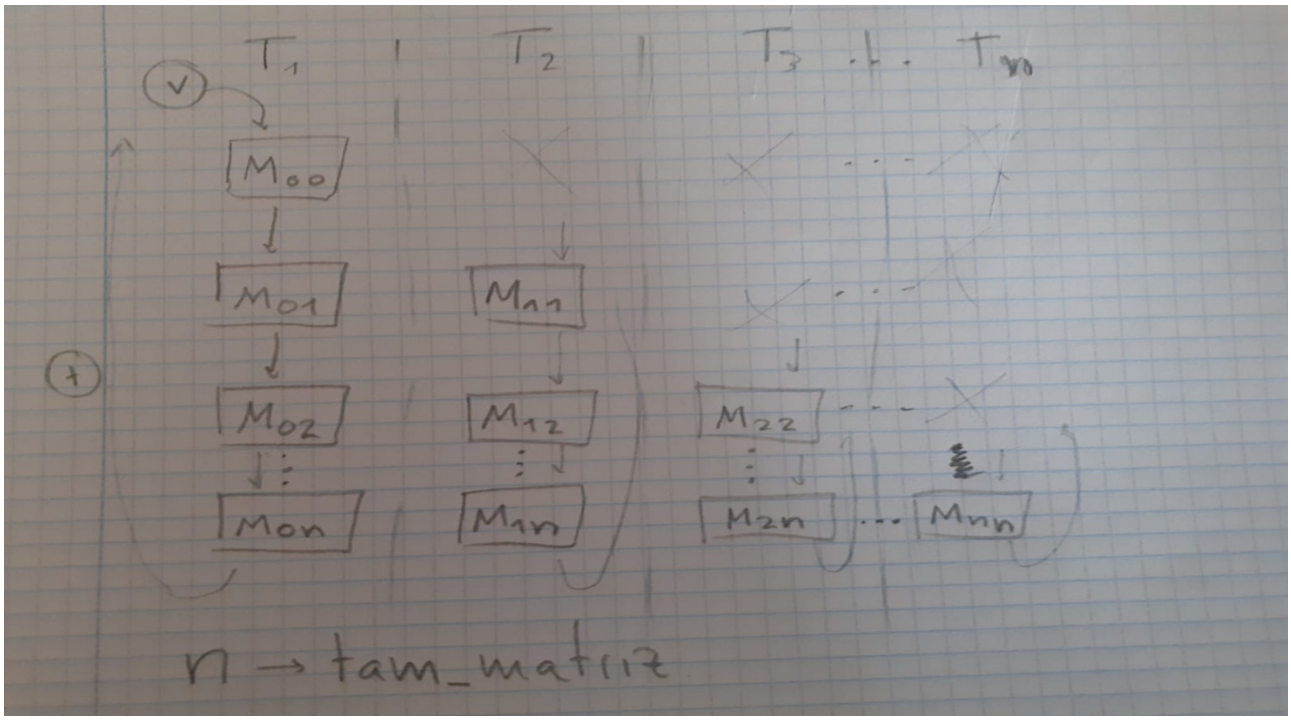
**CAPTURA CÓDIGO FUENTE**: `pmtv-OpenMP.c`

pmtv-OpenMP.c

```c
//gcc -fopenmp -O2 pmtv-secuencial.c -o programa //Si usara OpenMP
//g++ -O2 -o programa pmtv-secuencial.c //Si secuencial
#include <stdlib.h> // biblioteca con funciones atoi(), malloc() y free()
#include <stdio.h>  // biblioteca donde se encuentra la funcion printf()
#include <time.h>   // biblioteca donde se encuentra la funcion clock_gettime()
#include <omp.h>

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

    if (argc < 3) {
        printf("Usar ./programa <num_filas> <tipo_schedule(0-static/1-dynamic/2-guided)> (opt<chunk>)\n");
        exit(-1);
    }
    unsigned int N = atoi(argv[1]); //Matriz sera de tamaño N*N

    //Crear vector que multiplica a la matriz
    int *v1;
    v1 = (int*) malloc(N*sizeof(int)); // malloc necesita el tamano en bytes
      if (v1 == NULL)
      {
        printf("No hay suficiente espacio para el vector \n");
        exit(-2);
      }

    printf("Tamano Matriz Cuadrada:%u (%u B)\n", N, sizeof(unsigned int));
    //Inicializar vector
    for (int i = 0; i < N; i++) {
        v1[i] = i;
    }
    //Inicializar vector solucion
    int v2[N];
    for(int i=0; i<N; i++)
    {
        v2[i]=0;
    }
    //Inicializar Matriz
```

```
    int **M = (int **)malloc(N*sizeof(int*));
    for(int i=0; i<N; i++)
    M[i] = (int*) malloc(N*sizeof(int)); // malloc necesita el tamano en bytes
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            if(j<i)
                M[i][j] = 0;
            else
                M[i][j] = (i+1) + (j+1);
        }
    }
    //Mostrar matriz
    /.../
    struct timespec cgt1, cgt2;
    double ncgt; //para tiempo de ejecucion
    clock_gettime(CLOCK_REALTIME,&cgt1);
    //Inicio operacion

    omp_sched_t tipo = omp_sched_static;
        switch(atoi(argv[2]))
        {
            case 0:
                tipo = omp_sched_static;
                break;
            case 1:
                tipo = omp_sched_dynamic;
                break;
            case 2:
                tipo = omp_sched_guided;
                break;
        }
    if(argc==4) {
        int chunk = atoi(argv[3]);
        omp_set_schedule(tipo, chunk);
    }
    else omp_set_schedule(tipo, 0);
int k=0;
int operaciones_hechas[12]={0,0,0,0,0,0,0,0,0,0,0,0};
#pragma omp parallel for private(k) schedule(runtime)
    for(int i=0; i<N; i++)
    {
        for(k=i; k<N; k++)
        {
            operaciones_hechas[omp_get_thread_num()]++;
            v2[i]+=M[i][k]*v1[k];
        }
    }
    for(int i=0; i<12; i++)
    {
        printf("Thread %d ha realizado %d operaciones\n", i,operaciones_hechas[i]);
    }
    //Fin operacion
    clock_gettime(CLOCK_REALTIME,&cgt2);
    ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+(double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));
    //Imprimir resultado del producto y el tiempo de ejecucion
    if (N < 12) {
        printf("Tiempo:%11.9f\t / Tamano Matriz:%u\n", ncgt, N);
        for (int i = 0; i < N; i++) {
            printf("/ v2[%d]= %d\n", i, v2[i]);
        }
    }
    else
    {
        printf("Tiempo:%11.9f\t / Tamano Matriz:%u", ncgt,N);
        printf("v2[0]=%d / / v2[%d]=%d /\n", v2[0],N,v2[N]);
    }
}
```

## DESCOMPOSICIÓN DE DOMINIO:




## CAPTURAS DE PANTALLA:

Seleccionar d2estudiante8@atcgrid:~/bp3/ejer7

```
$srun -p ac pmtv-OpenMP_atcgrid.sh
Id. usuario del trabajo: d2estudiante8
Id. del trabajo: 47454
Nombre del trabajo especificado por usuario: pmtv-OpenMP_atcgrid.sh
Directorio de trabajo (en el que se ejecuta el script): /home/d2estudiante8/bp3/ejer7
Cola: ac
Nodo que ejecuta este trabajo:atcgrid
Nº de nodos asignados al trabajo: 1
Nodos asignados al trabajo: atcgrid1
CPUs por nodo: 2

 1. Ejecución distintos schedule cambiando chunk

  - Tipo de schedule: 0
Tamano Matriz Cuadrada:15360 (4 B)
Thread 0 ha realizado 18842240 operaciones
Thread 1 ha realizado 17203840 operaciones
Thread 2 ha realizado 15565440 operaciones
Thread 3 ha realizado 13927040 operaciones
Thread 4 ha realizado 12288640 operaciones
Thread 5 ha realizado 10650240 operaciones
Thread 6 ha realizado 9011840 operaciones
Thread 7 ha realizado 7373440 operaciones
Thread 8 ha realizado 5735040 operaciones
Thread 9 ha realizado 4096640 operaciones
Thread 10 ha realizado 6656189 operaciones
Thread 11 ha realizado 819840 operaciones
Tiempo:0.331920058        / Tamano Matriz:15360v2[0]=1191693824 / / v2[15360]=0 /
Tamano Matriz Cuadrada:15360 (4 B)
Thread 0 ha realizado 9838080 operaciones
Thread 1 ha realizado 9836800 operaciones
Thread 2 ha realizado 9835520 operaciones
Thread 3 ha realizado 9834240 operaciones
Thread 4 ha realizado 9832960 operaciones
Thread 5 ha realizado 9831680 operaciones
Thread 6 ha realizado 9830400 operaciones
Thread 7 ha realizado 9829120 operaciones
Thread 8 ha realizado 9827840 operaciones
Thread 9 ha realizado 9826560 operaciones
Thread 10 ha realizado 14023229 operaciones
Thread 11 ha realizado 9824000 operaciones
Tiempo:0.333305223        / Tamano Matriz:15360v2[0]=1191693824 / / v2[15360]=0 /
Tamano Matriz Cuadrada:15360 (4 B)
Thread 0 ha realizado 10281600 operaciones
```

**TABLA RESULTADOS, SCRIPT Y GRÁFICA atcgrid**

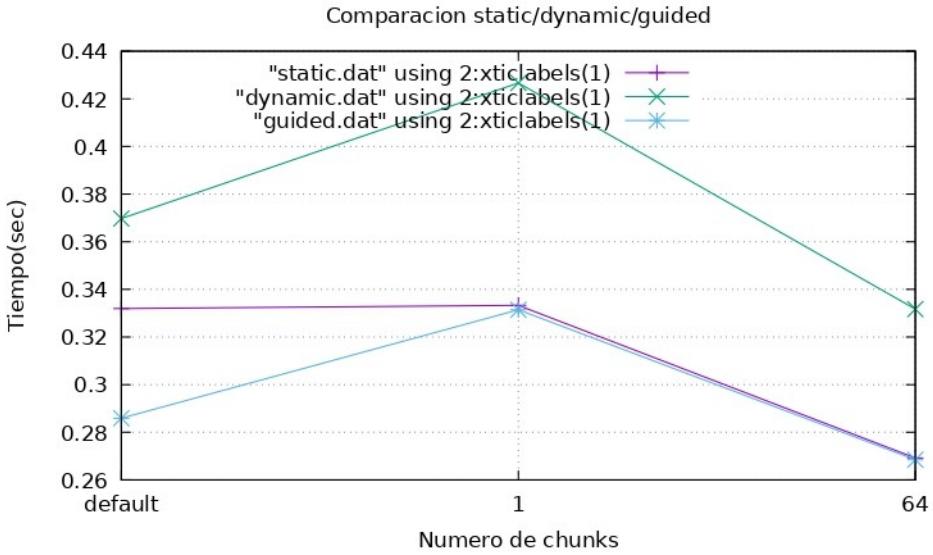
**SCRIPT:** pmvt-OpenMP_atcgrid.sh

v-OpenMP.c | pmtv-OpenMP_atcgrid.sh

```
#!/bin/bash
#Órdenes para el sistema de colas:
#1. Asigna al trabajo un nombre
#SBATCH --job-name=sumavectores
#2. Asignar el trabajo a una cola (partición)
#SBATCH --partition=ac
#2. Asignar el trabajo a un account
#SBATCH --account=ac

#Obtener información de las variables del entorno del sistema de colas:
echo "Id. usuario del trabajo: $SLURM_JOB_USER"
echo "Id. del trabajo: $SLURM_JOBID"
echo "Nombre del trabajo especificado por usuario: $SLURM_JOB_NAME"
echo "Directorio de trabajo (en el que se ejecuta el script): $SLURM_SUBMIT_DIR"
echo "Cola: $SLURM_JOB_PARTITION"
echo "Nodo que ejecuta este trabajo:$SLURM_SUBMIT_HOST"
echo "Nº de nodos asignados al trabajo: $SLURM_JOB_NUM_NODES"
echo "Nodos asignados al trabajo: $SLURM_JOB_NODELIST"
echo "CPUs por nodo: $SLURM_JOB_CPUS_PER_NODE"
#Instrucciones del script para ejecutar código:
echo -e "\n 1. Ejecución distintos schedule cambiando chunk"
for ((P=0;P<3;P++)) #tipo de schedule
do
  echo -e "\n  - Tipo de schedule: $P"
  ./programa 15360 $P
  ./programa 15360 $P 1
  ./programa 15360 $P 64
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** | 0,331478070 | 0,300526730 | 0,281221039 |
| **1** | 0,331503360 | 0,425920491 | 0,332638113 |
| **64** | 0,333492594 | 0,259726384 | 0,290548157 |
| **Chunk** | **Static** | **Dynamic** | **Guided** |
| **por defecto** | 0,331920058 | 0,369739752 | 0,285920342 |
| **1** | 0,333305223 | 0,426668324 | 0,331422759 |
| **64** | 0,269219934 | 0,331741498 | 0,268450616 |




Podemos ver como los mejores resultados los proporciona la schedule guided y los peores la dynamic, aunque como podemos observar en las tablas, el tiempo de ejecución tiene variaciones significativas entre distintas ejecuciones, probablemente porque hay schedules que intentarán asignar más trabajo a una hebra de menor rendimiento en ese momento y viceversa

**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`

pmm-secuencial.c

```c
//gcc -fopenmp -O2 pmm-secuencial.c -o programa //Si usara OpenMP
//g++ -O2 -o programa pmm-secuencial.c
#include <stdlib.h> // biblioteca con funciones atoi(), malloc() y free()
#include <stdio.h>  // biblioteca donde se encuentra la funcion printf()
#include <time.h>   // biblioteca donde se encuentra la funcion clock_gettime()

int main(int argc, char** argv) {
    if (argc < 2) {
        printf("Usar ./programa <num_filas>\n");
        exit(-1);
    }
    unsigned int N = atoi(argv[1]); //Matriz sera de tamaño N*N

    //Crear matriz solución e inicializar a 0
    int **S = (int **)malloc(N*sizeof(int*));
    for(int i=0; i<N; i++)
        S[i] = (int*) malloc(N*sizeof(int)); // malloc necesita el tamano en bytes
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
                S[i][j] = 0;
        }
    }
    printf("Tamano Matriz Cuadrada:%u (%u B)\n", N, sizeof(unsigned int));

    //Inicializar Matriz 1 y 2
    int **M1 = (int **)malloc(N*sizeof(int*));
    for(int i=0; i<N; i++)
    M1[i] = (int*) malloc(N*sizeof(int)); // malloc necesita el tamano en bytes
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            if(j<i)
                M1[i][j] = 0;
            else
                M1[i][j] = (i+1) + (j+1);
        }
    }
```

```
37      int **M2 = (int **)malloc(N*sizeof(int*));
38      for(int i=0; i<N; i++)
39          M2[i] = (int*) malloc(N*sizeof(int)); // malloc necesita el tamano en bytes
40      for (int i = 0; i < N; i++) {
41          for (int j = 0; j < N; j++) {
42              if(j<i)
43                  M2[i][j] = 0;
44              else
45                  M2[i][j] = (i+1) + (j+1);
46          }
47      }
48      //Mostrar matriz 1 que es igual a la 2
49      if(N<=12) {...}
59      struct timespec cgt1, cgt2;
60      double ncgt; //para tiempo de ejecucion
61      clock_gettime(CLOCK_REALTIME,&cgt1);
62      //Inicio operacion MULTIPLICAR MATRICES
63      for(int i=0; i<N; i++)
64      {
65          for(int j=0; j<N; j++)
66          {
67              for(int k=0; k<N; k++) {
68                  S[i][j] += M1[i][k] * M2[k][j];
69              }
70          }
71      }
72      //Fin operacion
73      clock_gettime(CLOCK_REALTIME,&cgt2);
74      ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+(double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));
75      //Imprimir resultado del producto y el tiempo de ejecucion
76      if (N < 12) {
77          printf("Tiempo:%11.9f\t / Tamano Matriz:%u\n", ncgt, N);
78          printf("Solucion: \n");
79          for (int i = 0; i < N; i++)
80          {
81              for(int j=0; j<N; j++)
82              {
83                  printf("%d\t", S[i][j]);
84              }
85              printf("\n");
86          }
87      }
88      else
89      {
90          printf("Tiempo:%11.9f\t / Tamano Matriz:%u", ncgt,N);
91          printf("S[0][0]=%d / / S[%d][%d]=%d /\n", S[0][0],N-1,N-1, S[N-1][N-1]);
92      }
93  }
```

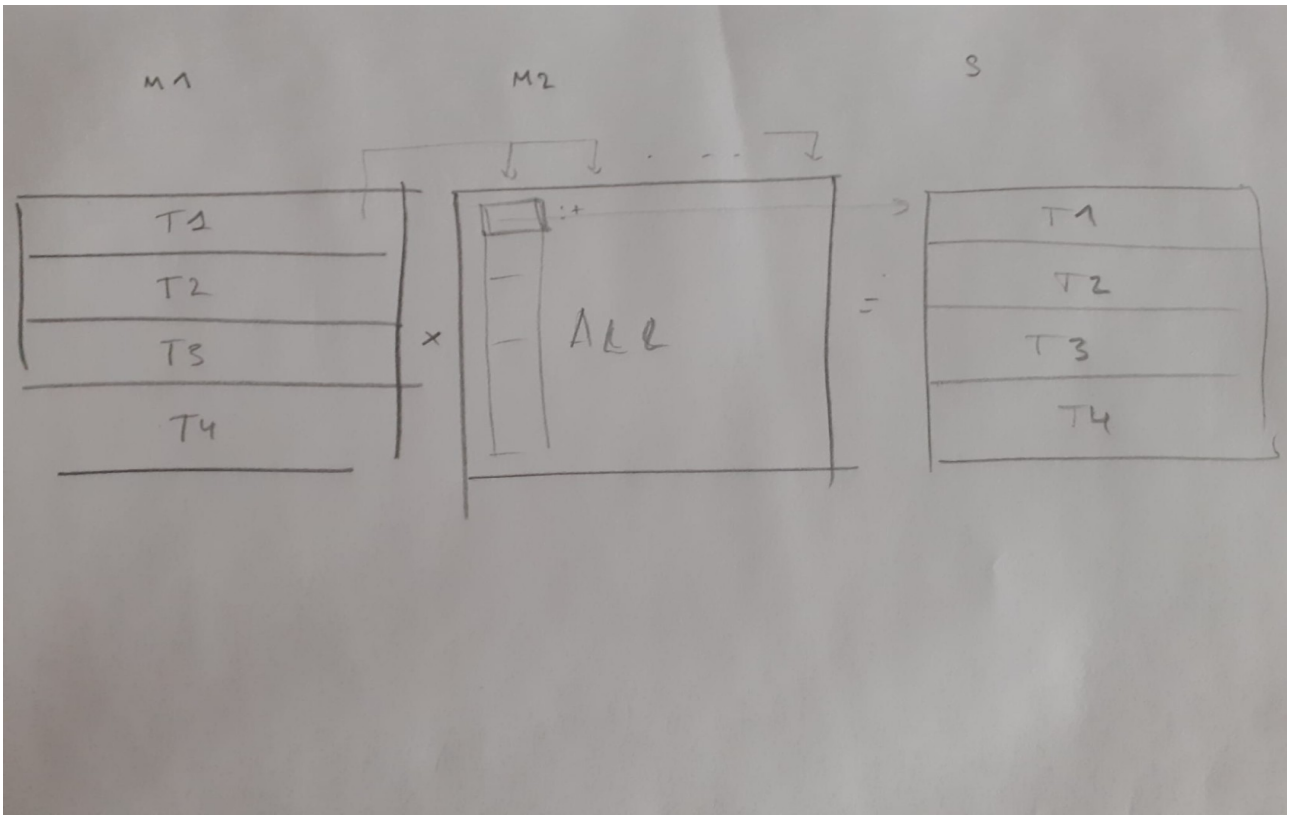
**CAPTURAS DE PANTALLA:**

```
adduser@DESKTOP-UH8Q3EQ: /mnt/d/Jorge Gangoso Klock/Documentos/Trabajos Universidad/AC/bp3/ejer8
adduser@DESKTOP-UH8Q3EQ:/mnt/d/Jorge Gangoso Klock/Documentos/Trabajos Universidad/AC/bp3/ejer8$ g++ -O2 -o programa pmm-secuencial.c
pmm-secuencial.c: In function ‘int main(int, char**)’:
pmm-secuencial.c:24:73: warning: format ‘%u’ expects argument of type ‘unsigned int’, but argument 3 has type ‘long unsigned int’ [-Wformat=]
     printf("Tamano Matriz Cuadrada:%u (%u B)\n", N, sizeof(unsigned int));
                                                                        ^

adduser@DESKTOP-UH8Q3EQ:/mnt/d/Jorge Gangoso Klock/Documentos/Trabajos Universidad/AC/bp3/ejer8$ ./programa 4
Tamano Matriz Cuadrada:4 (4 B)
matriz inicializada:
2       3       4       5
0       4       5       6
0       0       6       7
0       0       0       8
Tiempo:0.000000800       / Tamano Matriz:4
Solucion:
4       18      47      96
0       16      50      107
0       0       36      98
0       0       0       64
adduser@DESKTOP-UH8Q3EQ:/mnt/d/Jorge Gangoso Klock/Documentos/Trabajos Universidad/AC/bp3/ejer8$ ./programa 100
Tamano Matriz Cuadrada:100 (4 B)
Tiempo:0.000524300       / Tamano Matriz:100S[0][0]=4 / / S[99][99]=40000 /
adduser@DESKTOP-UH8Q3EQ:/mnt/d/Jorge Gangoso Klock/Documentos/Trabajos Universidad/AC/bp3/ejer8$
```

En este caso, puesto que la inicialización es indiferente, se ha mantenido las matrices triangulares del ejercicio anterior, para hacer matrices cuadradas normales sólo sería necesario cambiar la inicialización y eliminar el apartado if(j<i) M[i][j] = 0 como haremos para el siguiente ejercicio.

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`

pmm-OpenMP.c

```c
//gcc -fopenmp -O2 pmm-secuencial.c -o programa //Si usara OpenMP
//g++ -O2 -o programa pmm-secuencial.c
#include <stdlib.h> // biblioteca con funciones atoi(), malloc() y free()
#include <stdio.h>  // biblioteca donde se encuentra la funcion printf()
#include <time.h>   // biblioteca donde se encuentra la funcion clock_gettime()
#include <omp.h>

int main(int argc, char** argv)
{
    if (argc < 2) {
        printf("Usar ./programa <num_filas>\n");
        exit(-1);
    }
    unsigned int N = atoi(argv[1]); //Matriz sera de tamaño N*N

    //Crear matriz solución e inicializar a 0
    int **S = (int **)malloc(N*sizeof(int*));
    for(int i=0; i<N; i++)
        S[i] = (int*) malloc(N*sizeof(int)); // malloc necesita el tamano en bytes
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++) {
                S[i][j] = 0;
        }
    }
    printf("Tamano Matriz Cuadrada:%u (%u B)\n", N, sizeof(unsigned int));

    //Inicializar Matriz 1 y 2
    int **M1 = (int **)malloc(N*sizeof(int*));
    for(int i=0; i<N; i++)
     M1[i] = (int*) malloc(N*sizeof(int)); // malloc necesita el tamano en bytes
#pragma parallel for
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
                M1[i][j] = (i+1) + (j+1);
    }
    int **M2 = (int **)malloc(N*sizeof(int*));
    for(int i=0; i<N; i++)
        M2[i] = (int*) malloc(N*sizeof(int)); // malloc necesita el tamano en bytes
#pragma parallel for
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++)
            M2[i][j] = (i+1) + (j+1);
    }
    //Mostrar matriz 1 que es igual a la 2
    if(N<=12)
    {
        printf("matriz inicializada: \n");
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                printf("%d \t", M1[i][j]);
            }
            printf("\n");
        }
    }
    struct timespec cgt1, cgt2;
    double ncgt; //para tiempo de ejecucion
    clock_gettime(CLOCK_REALTIME,&cgt1);
    //Inicio operacion MULTIPLICAR MATRICES
    #pragma omp parallel for
    for(int i=0; i<N; i++)
    {
        for(int j=0; j<N; j++)
        {
            for(int k=0; k<N; k++) {
                S[i][j] += M1[i][k] * M2[k][j];
            }
        }
    }
    //Fin operacion
    clock_gettime(CLOCK_REALTIME,&cgt2);
```

```
73          ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+(double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));
74          //Imprimir resultado del producto y el tiempo de ejecucion
75          if (N < 12)
76          {
77              printf("Tiempo:%11.9f\t / Tamano Matriz:%u\n", ncgt, N);
78              printf("Solucion: \n");
79              for (int i = 0; i < N; i++)
80              {
81                  for(int j=0; j<N; j++)
82                  {
83                      printf("%d\t", S[i][j]);
84                  }
85                  printf("\n");
86              }
87          }
88          else
89          {
90              printf("Tiempo:%11.9f\t / Tamano Matriz:%u", ncgt,N);
91              printf("S[0][0]=%d / / S[%d][%d]=%d /\n", S[0][0],N-1,N-1, S[N-1][N-1]);
92          }
93      }
```

**CAPTURAS DE PANTALLA:**

```
adduser@DESKTOP-UH8Q3EQ: /mnt/d/Jorge Gangoso Klock/Documentos/Trabajos Universidad/AC/bp3/ejer9
[JorgeGangosoKlock adduser@DESKTOP-UH8Q3EQ:/mnt/d/Jorge Gangoso Klock/Documentos/Trabajos Universidad/AC/bp3/ejer9] 2020-05-04 Monday
$gcc -fopenmp -O2 pmm-OpenMP.c -o programa
pmm-OpenMP.c: In function 'main':
pmm-OpenMP.c:26:41: warning: format '%u' expects argument of type 'unsigned int', but argument 3 has type 'long unsigned int' [-Wformat
=]
     printf("Tamano Matriz Cuadrada:%u (%u B)\n", N, sizeof(unsigned int));
                                         ~^
                                         %lu
[JorgeGangosoKlock adduser@DESKTOP-UH8Q3EQ:/mnt/d/Jorge Gangoso Klock/Documentos/Trabajos Universidad/AC/bp3/ejer9] 2020-05-04 Monday
$./programa 5
Tamano Matriz Cuadrada:5 (4 B)
matriz inicializada:
2       3       4       5       6
3       4       5       6       7
4       5       6       7       8
5       6       7       8       9
6       7       8       9       10
Tiempo:0.000650200       / Tamano Matriz:5
Solucion:
90      110     130     150     170
110     135     160     185     210
130     160     190     220     250
150     185     220     255     290
170     210     250     290     330
[JorgeGangosoKlock adduser@DESKTOP-UH8Q3EQ:/mnt/d/Jorge Gangoso Klock/Documentos/Trabajos Universidad/AC/bp3/ejer9] 2020-05-04 Monday
$
```

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

**ESTUDIO DE ESCALABILIDAD EN PCLOCAL:**

**SCRIPT:** pmm-OpenMP_pclocal.sh