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): Alejandro Molina Criado
Grupo de prácticas: A2
Fecha de entrega: 6/05/2020
Fecha evaluación en clase: 6/05/2020

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`

```
/*
    *   Alejandro Molina Criado
    *   BP3 Arquitectura de Computadores
    *   06/05/2020
    *   Para compilar : gcc -O2 -fopenmp -o if-clausule-modificado if-clausule-
modificado.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/o num_threads\n");
 exit(-1);
 }

 n = atoi(argv[1]); if (n>20) n=20;
 for (i=0; i<n; i++) {
 a[i] = i;
 }

 int x = atoi(argv[2]);
 if (x > omp_get_max_threads()) x = omp_get_max_threads() ;
```

```
#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);
 }
}
```

CAPTURAS DE PANTALLA:

```
alex@alex-pc:~/UNI/AC/PRÁCTICAS/OPEN_MPI/bp3/ej1$ ./if-clausule 5
 thread 0 suma de a[0]=0 sumalocal=0
 thread 3 suma de a[3]=3 sumalocal=3
 thread 2 suma de a[2]=2 sumalocal=2
 thread 4 suma de a[4]=4 sumalocal=4
 thread 1 suma de a[1]=1 sumalocal=1
thread master=0 imprime suma=10
alex@alex-pc:~/UNI/AC/PRÁCTICAS/OPEN_MPI/bp3/ej1$ ./if-clausule 3
 thread 0 suma de a[0]=0 sumalocal=0
 thread 0 suma de a[1]=1 sumalocal=1
 thread 0 suma de a[2]=2 sumalocal=3
thread master=0 imprime suma=3
alex@alex-pc:~/UNI/AC/PRÁCTICAS/OPEN_MPI/bp3/ej1$ ./if-clausule-modificado 5 3
 thread 0 suma de a[0]=0 sumalocal=0
 thread 0 suma de a[1]=1 sumalocal=1
 thread 1 suma de a[2]=2 sumalocal=2
 thread 1 suma de a[3]=3 sumalocal=5
 thread 2 suma de a[4]=4 sumalocal=4
thread master=0 imprime suma=10
alex@alex-pc:~/UNI/AC/PRÁCTICAS/OPEN_MPI/bp3/ej1$ ./if-clausule-modificado 5 5
 thread 2 suma de a[2]=2 sumalocal=2
 thread 4 suma de a[4]=4 sumalocal=4
 thread 1 suma de a[1]=1 sumalocal=1
 thread 0 suma de a[0]=0 sumalocal=0
 thread 3 suma de a[3]=3 sumalocal=3
thread master=0 imprime suma=10
alex@alex-pc:~/UNI/AC/PRÁCTICAS/OPEN_MPI/bp3/ej1$ ./if-clausule-modificado 3 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 master=0 imprime suma=3
alex@alex-pc:~/UNI/AC/PRÁCTICAS/OPEN_MPI/bp3/ej1$
```

RESPUESTA: En la primera ejecución de if-clausule , observamos que se usan 5 hebras (ya que así está establecido por defecto en la variable exportada) , en la segunda se utiliza únicamente la hebras Master ya que no se cumple la sentencia if (que n > 4).

En la primera llamada a if-clausule-modificado , se estableció que el número de hebras fuera 3 y así se refleja en la ejecución. Ídem para la segunda ejecución , donde se establecieron 5 y 5 hebras fueron ejecutadas.

En la última ejecución ocurre lo mismo que en la segunda , como n < 4 , la hebras Master es la encargada de realizar la suma.

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

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

RESPUESTA:

Guided utilizan el valor del chunk como el tamaño mínimo de bloque , las otras dos (static y dinamic) , lo utiliza como el número de iteraciones que conforman el tamaño de bloque.
Con Static observamos que las iteraciones acorde con el id de hebra , de forma ordenada.
Con Dynamic y Guided , las iteraciones asignan iteraciones a hebras conforme van llegando.

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`

```
/*
    *   Alejandro Molina Criado
    *   BP3 Arquitectura de Computadores
    *   06/05/2020
    *   Para compilar : gcc -O2 -fopenmp -o if-clausule-modificado if-clausule-
modificado.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, modif;
    omp_sched_t tipo;

    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]);

    int dyn_var = omp_get_dynamic();
    int nthreads_var = omp_get_max_threads();
    int thread_limit_var = omp_get_thread_limit();
    omp_get_schedule(&tipo, &modif);

    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] suma=%d \n", omp_get_thread_num(), i,
suma);
        }

        #pragma omp single
            printf("\nthread %d imprime dyn-var %d \n", omp_get_thread_num(),
dyn_var);
```

```
printf("thread %d imprime nthreads-var %d \n", omp_get_thread_num(),
nthreads_var);
            printf("thread %d imprime thread-limit-var %d \n",
omp_get_thread_num(),thread_limit_var);
            printf("thread %d imprime run-sched-var mod %d \n",
omp_get_thread_num(), modif);
            if(tipo == omp_sched_static) printf("El tipo es estático \n");
            if(tipo == omp_sched_dynamic) printf("El tipo es dinámico \n");
            if(tipo == omp_sched_guided) printf("El tipo es guided \n");

        printf("\nFuera de 'parallel for' suma %d\n dyn-var %d\n nthreads-var %d\
n thread-limit-var %d\
n run-sched-var mod %d\n",suma,dyn_var, nthreads_var, thread_limit_var, modif);

        if(tipo == omp_sched_static) printf("El tipo es estático \n");
        if(tipo == omp_sched_dynamic) printf("El tipo es dinámico \n");
        if(tipo == omp_sched_guided) printf("El tipo es guided \n");
}
```

CAPTURAS DE PANTALLA:

```
alex@alex-pc:~/UNI/AC/PRÁCTICAS/OPEN_MPI/bp3/ej3$ gcc -O2 -fopenmp -o scheduled-modificado scheduled-modificado.c
alex@alex-pc:~/UNI/AC/PRÁCTICAS/OPEN_MPI/bp3/ej3$ ./scheduled-modificado 8 3
thread 0 suma a[0] suma=0
thread 0 suma a[1] suma=1
thread 0 suma a[2] suma=3
thread 3 suma a[3] suma=3
thread 3 suma a[4] suma=7
thread 3 suma a[5] suma=12
thread 1 suma a[6] suma=6
thread 1 suma a[7] suma=13

thread 0 imprime dyn-var 0
thread 0 imprime nthreads-var 5
thread 0 imprime thread-limit-var 2147483647
thread 0 imprime run-sched-var mod 1
El tipo es dinámico

Fuera de 'parallel for' suma 13
 dyn-var 0
 nthreads-var 5
 thread-limit-var 2147483647n run-sched-var mod 1
El tipo es dinámico
alex@alex-pc:~/UNI/AC/PRÁCTICAS/OPEN_MPI/bp3/ej3$
```

```
alex@alex-pc:~/UNI/AC/PRÁCTICAS/OPEN_MPI/bp3/ej3$ export OMP_DYNAMIC=TRUE
alex@alex-pc:~/UNI/AC/PRÁCTICAS/OPEN_MPI/bp3/ej3$ ./scheduled-modificado 15 3
thread 0 suma a[0] suma=0
thread 0 suma a[1] suma=1
thread 0 suma a[2] suma=3
thread 0 suma a[12] suma=15
thread 0 suma a[13] suma=28
thread 0 suma a[14] suma=42
thread 1 suma a[3] suma=3
thread 1 suma a[4] suma=7
thread 1 suma a[5] suma=12
thread 2 suma a[6] suma=6
thread 2 suma a[7] suma=13
thread 2 suma a[8] suma=21
thread 3 suma a[9] suma=9
thread 3 suma a[10] suma=19
thread 3 suma a[11] suma=30

thread 0 imprime dyn-var 1
thread 0 imprime nthreads-var 5
thread 0 imprime thread-limit-var 2147483647
thread 0 imprime run-sched-var mod 1
El tipo es dinámico

Fuera de 'parallel for' suma 42
 dyn-var 1
 nthreads-var 5
 thread-limit-var 2147483647n run-sched-var mod 1
El tipo es dinámico
```

```
alex@alex-pc:~/UNI/AC/PRÁCTICAS/OPEN_MPI/bp3/ej3$ export OMP_NUM_THREADS=5
alex@alex-pc:~/UNI/AC/PRÁCTICAS/OPEN_MPI/bp3/ej3$ export OMP_DYNAMIC=FALSE
alex@alex-pc:~/UNI/AC/PRÁCTICAS/OPEN_MPI/bp3/ej3$ ./scheduled-modificado 15 3
thread 0 suma a[0] suma=0
thread 0 suma a[1] suma=1
thread 0 suma a[2] suma=3
thread 0 suma a[12] suma=15
thread 0 suma a[13] suma=28
thread 0 suma a[14] suma=42
thread 3 suma a[3] suma=3
thread 3 suma a[4] suma=7
thread 3 suma a[5] suma=12
thread 4 suma a[6] suma=6
thread 4 suma a[7] suma=13
thread 4 suma a[8] suma=21
thread 1 suma a[9] suma=9
thread 1 suma a[10] suma=19
thread 1 suma a[11] suma=30

thread 0 imprime dyn-var 0
thread 0 imprime nthreads-var 5
thread 0 imprime thread-limit-var 2147483647
thread 0 imprime run-sched-var mod 1
El tipo es dinámico

Fuera de 'parallel for' suma 42
 dyn-var 0
 nthreads-var 5
 thread-limit-var 2147483647n run-sched-var mod 1
El tipo es dinámico
alex@alex-pc:~/UNI/AC/PRÁCTICAS/OPEN_MPI/bp3/ej3$
```

Modificando el numero de hebras a 5 y con ajuste no dinámico , con un vector de 15 elementos las hebras ejecutan bloques de 3 y son 5 hebras las que ejecutan el for.

Cambiando el ajuste a dinámico se observa que el número de hebras que ejecuta el for son 4 , decisión de omp.

```
alex@alex-pc:~/UNI/AC/PRÁCTICAS/OPEN_MPI/bp3/ej3$ export OMP_NUM_THREADS=3
alex@alex-pc:~/UNI/AC/PRÁCTICAS/OPEN_MPI/bp3/ej3$ ./scheduled-modificado 9 3
thread 0 suma a[0] suma=0
thread 0 suma a[1] suma=1
thread 0 suma a[2] suma=3
thread 1 suma a[3] suma=3
thread 1 suma a[4] suma=7
thread 1 suma a[5] suma=12
thread 2 suma a[6] suma=6
thread 2 suma a[7] suma=13
thread 2 suma a[8] suma=21

thread 0 imprime dyn-var 0
thread 0 imprime nthreads-var 3
thread 0 imprime thread-limit-var 2147483647
thread 0 imprime run-sched-var mod 1
El tipo es dinámico

Fuera de 'parallel for' suma 21
 dyn-var 0
 nthreads-var 3
 thread-limit-var 2147483647n run-sched-var mod 1
El tipo es dinámico
alex@alex-pc:~/UNI/AC/PRÁCTICAS/OPEN_MPI/bp3/ej3$
```

```
alex@alex-pc:~/UNI/AC/PRÁCTICAS/OPEN_MPI/bp3/ej3$ export OMP_NUM_THREADS=2
alex@alex-pc:~/UNI/AC/PRÁCTICAS/OPEN_MPI/bp3/ej3$ ./scheduled-modificado 9 3
thread 0 suma a[0] suma=0
thread 0 suma a[1] suma=1
thread 0 suma a[2] suma=3
thread 0 suma a[6] suma=9
thread 0 suma a[7] suma=16
thread 0 suma a[8] suma=24
thread 1 suma a[3] suma=3
thread 1 suma a[4] suma=7
thread 1 suma a[5] suma=12

thread 0 imprime dyn-var 0
thread 0 imprime nthreads-var 2
thread 0 imprime thread-limit-var 2147483647
thread 0 imprime run-sched-var mod 1
El tipo es dinámico

Fuera de 'parallel for' suma 24
 dyn-var 0
 nthreads-var 2
 thread-limit-var 2147483647n run-sched-var mod 1
El tipo es dinámico
```

En la primera ejecución , son 3 hebras las que ejecutan el for , en la segunda son 2 . También podemos observar que el valor de nthreads-var cambia en ambas ejecuciones acorde al valor dado.

Con RUN_SCHED_VAR , la variable de control modifica el tipo de reparto de hebras n el for , si es dinámico o bien estático.

```
alex@alex-pc:~/UNI/AC/PRÁCTICAS/OPEN_MPI/bp3/ej3$ export OMP_SCHEDULE="dynamic, 3"
alex@alex-pc:~/UNI/AC/PRÁCTICAS/OPEN_MPI/bp3/ej3$ ./scheduled-modificado 9 3
thread 0 suma a[0] suma=0
thread 0 suma a[1] suma=1
thread 0 suma a[2] suma=3
thread 0 suma a[6] suma=9
thread 0 suma a[7] suma=16
thread 0 suma a[8] suma=24
thread 1 suma a[3] suma=3
thread 1 suma a[4] suma=7
thread 1 suma a[5] suma=12

thread 0 imprime dyn-var 0
thread 0 imprime nthreads-var 4
thread 0 imprime thread-limit-var 2
thread 0 imprime run-sched-var mod 3
El tipo es dinámico

Fuera de 'parallel for' suma 24
 dyn-var 0
 nthreads-var 4
 thread-limit-var 2n run-sched-var mod 3
El tipo es dinámico
alex@alex-pc:~/UNI/AC/PRÁCTICAS/OPEN_MPI/bp3/ej3$
```

```
alex@alex-pc:~/UNI/AC/PRÁCTICAS/OPEN_MPI/bp3/ej3$ export OMP_SCHEDULE="static, 5"
alex@alex-pc:~/UNI/AC/PRÁCTICAS/OPEN_MPI/bp3/ej3$ ./scheduled-modificado 15 3
thread 0 suma a[0] suma=0
thread 0 suma a[1] suma=1
thread 0 suma a[2] suma=3
thread 0 suma a[6] suma=9
thread 0 suma a[7] suma=16
thread 0 suma a[8] suma=24
thread 0 suma a[9] suma=33
thread 0 suma a[10] suma=43
thread 0 suma a[11] suma=54
thread 1 suma a[3] suma=3
thread 1 suma a[4] suma=7
thread 1 suma a[5] suma=12
thread 0 suma a[12] suma=66
thread 0 suma a[13] suma=79
thread 0 suma a[14] suma=93

thread 0 imprime dyn-var 0
thread 0 imprime nthreads-var 4
thread 0 imprime thread-limit-var 2
thread 0 imprime run-sched-var mod 5
El tipo es estático

Fuera de 'parallel for' suma 93
 dyn-var 0
 nthreads-var 4
 thread-limit-var 2n run-sched-var mod 5
El tipo es estático
alex@alex-pc:~/UNI/AC/PRÁCTICAS/OPEN_MPI/bp3/ej3$
```

Con THREAD_LIMIT_VAR establecemos el máximo de hebras que se pueden crear . En la captura de abajo , se puede ver que al poner el límite a 2 , y efectivamente son 2 hebras las que trabajan.

Aún así podemos ver que realmente el numero de hebras son 4.

```
alex@alex-pc:~/UNI/AC/PRÁCTICAS/OPEN_MPI/bp3/ej3$ export OMP_THREAD_LIMIT=2
alex@alex-pc:~/UNI/AC/PRÁCTICAS/OPEN_MPI/bp3/ej3$ ./scheduled-modificado 5 3
thread 0 suma a[0] suma=0
thread 0 suma a[1] suma=1
thread 0 suma a[2] suma=3
thread 1 suma a[3] suma=3
thread 1 suma a[4] suma=7

thread 0 imprime dyn-var 0
thread 0 imprime nthreads-var 4
thread 0 imprime thread-limit-var 2
thread 0 imprime run-sched-var mod 1
El tipo es dinámico

Fuera de 'parallel for' suma 7
 dyn-var 0
 nthreads-var 4
 thread-limit-var 2n run-sched-var mod 1
El tipo es dinámico
alex@alex-pc:~/UNI/AC/PRÁCTICAS/OPEN_MPI/bp3/ej3$
```

RESPUESTA : En todas las ejecuciones de las variables de control , se imprime lo mismo dentro que fuera de la sección parallel

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
/*
    *   Alejandro Molina Criado
    *   BP3 Arquitectura de Computadores
    *   06/05/2020
    *   Para compilar : gcc -O2 -fopenmp -o if-clausule-modificado if-clausule-
modificado.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, modif;
    omp_sched_t tipo;

    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]);

    int dyn_var = omp_get_dynamic();
    int nthreads_var = omp_get_max_threads();
    int thread_limit_var = omp_get_thread_limit();
    omp_get_schedule(&tipo, &modif);

    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] suma=%d \n", omp_get_thread_num(), i, suma);
        }

        #pragma omp single
            printf("\nthread %d imprime dyn-var %d \n", omp_get_thread_num(), dyn_var);
            printf("thread %d imprime nthreads-var %d \n", omp_get_thread_num(),
nthreads_var);
            printf("thread %d imprime thread-limit-var %d \n",
omp_get_thread_num(),thread_limit_var);
            printf("thread %d imprime run-sched-var mod %d \n", omp_get_thread_num(),
modif);
            if(tipo == omp_sched_static) printf("El tipo es estático \n");
            if(tipo == omp_sched_dynamic) printf("El tipo es dinámico \n");
            if(tipo == omp_sched_guided) printf("El tipo es guided \n");

        printf("\nFuera de 'parallel for' suma %d\n dyn-var %d\n nthreads-var %d\n thread-
limit-var %d\
n run-sched-var mod %d\n",suma,dyn_var, nthreads_var, thread_limit_var, modif);

        if(tipo == omp_sched_static) printf("El tipo es estático \n");
        if(tipo == omp_sched_dynamic) printf("El tipo es dinámico \n");
        if(tipo == omp_sched_guided) printf("El tipo es guided \n");
}
```

CAPTURAS DE PANTALLA:

```
alex@alex-pc:~/UNI/AC/PRÁCTICAS/OPEN_MPI/bp3/ej4$ gcc -O2 -fopenmp -o scheduled-modificado4 scheduled-modificado4.c
alex@alex-pc:~/UNI/AC/PRÁCTICAS/OPEN_MPI/bp3/ej4$ ./scheduled-modificado4 4 2
thread 0 suma a[0] suma=0
thread 0 suma a[1] suma=1
thread 2 suma a[2] suma=2
thread 2 suma a[3] suma=5

thread 0 imprime dyn-var 0
thread 0 imprime nthreads-var 4
thread 0 imprime thread-limit-var 2147483647
thread 0 imprime run-sched-var mod 1
El tipo es dinámico

Fuera de 'parallel for' suma 5
 dyn-var 0
 nthreads-var 4
 thread-limit-var 2147483647n run-sched-var mod 1
El tipo es dinámico
alex@alex-pc:~/UNI/AC/PRÁCTICAS/OPEN_MPI/bp3/ej4$
```

RESPUESTA: El numero de threads dentro del for es 4 , fuera es 1 , aún estando dentro de la directiva parallel .
Las funciones omp_get_num_threads() y omp_in_parallel() , se mantiene igual dentro como fuera de la región paralell.
Fuera de la región paralela , el numero de threads disminuye a 1.

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`

```
/*
    *   Alejandro Molina Criado
    *   BP3 Arquitectura de Computadores
    *   06/05/2020
    *   Para compilar : gcc -O2 -fopenmp -o nombre nombre.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) {
  omp_sched_t kind;
  int dyn_var, nthreads_var;
  int i, n=16,chunk,a[n],suma=0;
  if(argc < 6) {
    fprintf(stderr, "Uso: %s n_iter kind chunk nthreads dyn\n", argv[0]);
    exit(-1);
  }
  omp_get_schedule(&kind, &chunk);
  printf("Antes:\ndyn-var: %d\nnthreads-var: %d\nthread-limit-var: %d\n\
run-sched-var: %d,%d\n", omp_get_dynamic(), omp_get_num_threads(),
    omp_get_thread_limit(), kind, chunk);

  n = atoi(argv[1]); if (n>200) n=200;

  chunk = atoi(argv[3]);
  kind = atoi(argv[2]); omp_set_schedule(kind,chunk);
  nthreads_var = atoi(argv[4]); omp_set_num_threads(nthreads_var);
  dyn_var = atoi(argv[5]); omp_set_dynamic(dyn_var);

  omp_get_schedule(&kind, &chunk);
  printf("\nDespues:\ndyn-var: %d\nnthreads-var: %d\nthread-limit-var: %d\n\
run-sched-var: %d,%d\n", omp_get_dynamic(), omp_get_num_threads(),
    omp_get_thread_limit(), kind, chunk);

  for (i=0; i<n; i++) a[i] = i;
  #pragma omp parallel
  {
    #pragma omp for 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);
    }
  }

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

}
```

CAPTURAS DE PANTALLA:

```
alex@alex-pc:~/UNI/AC/PRACTICAS/OPEN_MPI/bp3/ej5$ gcc -O2 -fopenmp -o scheduled-modificado5 scheduled-modificado5.c
alex@alex-pc:~/UNI/AC/PRÁCTICAS/OPEN_MPI/bp3/ej5$ ./scheduled-modificado5 9 3 3 3 4
Antes:
dyn-var: 0
nthreads-var: 1
thread-limit-var: 2147483647
run-sched-var: 2,1

Despues:
dyn-var: 1
nthreads-var: 1
thread-limit-var: 2147483647
run-sched-var: 3,3


Fuera de 'parallel for' suma=21
```

-- 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`

```
/*
    *   Alejandro Molina Criado
    *   BP3 Arquitectura de Computadores
    *   06/05/2020
    *   Para compilar : gcc -O2 -fopenmp -o pmtv-secuencia pmtv-secuencial.c
*/
#include <stdlib.h>
#include <stdio.h>
#include <time.h>

int main(int argc, char** argv){
    struct timespec cgt1,cgt2; double ncgt; //para tiempo de ejecución

    if (argc<2){
        printf("ERROR: Faltan numero de componentes del vector\n");
        exit(-1);
    }
```

```
    unsigned int N = atoi(argv[1]); //N =2^32-1=4294967295
    int i, j;

    int *v, *mv, **m;

    m = (int **)malloc(N*sizeof(int *));
    mv = (int*) malloc(N*sizeof(int));
    v = (int*) malloc(N*sizeof(int));

    for (i=0;i<N;i++)
        m[i] = (int *) malloc (N*sizeof(int));

    if ( v==NULL || mv==NULL || m==NULL ){
        printf("Error en la reserva de espacio para la matriz y el vector\n");
        exit(-2);
    }

    for(i=0; i<N; i++)
        if(m[i] == NULL){
            printf("Error en la reserva de espacio para la matriz y el vector\
n");
            exit(-3);
        }

    for(i=0; i<N; i++){
        for(j=i; j<N; j++){
            m[i][j] = 9;
        }
        v[i] = 2;
        mv[i] = 0;
    }

    clock_gettime(CLOCK_REALTIME,&cgt1);

    for(i=0; i<N; i++){
        for(j=i; j<N;j++){
            mv[i] += m[i][j] * v[i];
        }
    }

    clock_gettime(CLOCK_REALTIME,&cgt2);// tomamos el tiempo de finalización


    //VISUALIZACIÓN DE LA MATRIZ

    printf("--- Matriz --- \n");
    for(i=0; i<N; i++){
        for(j=0; j<N; j++)
            if(j >= i)
                printf("%d ", m[i][j]);
            else
                printf("0 ");
            printf("\n");
    }

    printf("Vector: \n");
```

```
    for(i=0; i<N; i++)
        printf("%d ", v[i]);
    printf("\n");

    printf("Resultado: \n");
    for(i=0; i<N; i++)
        printf("%d ", mv[i]);
    printf("\n");

    ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+(double) ((cgt2.tv_nsec-
cgt1.tv_nsec)/(1.e+9)); //Tiempo que ha tardado
    printf("Tamaño de la matriz y vector:%d\t / Tiempo(seg.) %11.9f\n", N, ncgt);


    //Liberacion de espacio
    for(i=0; i<N; i++)
        free(m[i]);

    free(m);
    free(mv);
    free(v);

    return 0;
}
```

CAPTURAS DE PANTALLA:

```
alex@alex-pc:~/UNI/AC/PRÁCTICAS/OPEN_MPI/bp3/ej6$ gcc -O2 -fopenmp -o pmtv-secuencial pmtv-secuencial.c
alex@alex-pc:~/UNI/AC/PRÁCTICAS/OPEN_MPI/bp3/ej6$ ./pmtv-secuencial 10
--- Matriz ---
9 9 9 9 9 9 9 9 9 9
0 9 9 9 9 9 9 9 9 9
0 0 9 9 9 9 9 9 9 9
0 0 0 9 9 9 9 9 9 9
0 0 0 0 9 9 9 9 9 9
0 0 0 0 0 9 9 9 9 9
0 0 0 0 0 0 9 9 9 9
0 0 0 0 0 0 0 9 9 9
0 0 0 0 0 0 0 0 9 9
0 0 0 0 0 0 0 0 0 9
Vector:
2 2 2 2 2 2 2 2 2 2
Resultado:
180 162 144 126 108 90 72 54 36 18
Tamaño de la matriz y vector:10  / Tiempo(seg.) 0.000000946
```

```
alex@alex-pc:~/UNI/AC/PRÁCTICAS/OPEN_MPI/bp3/ej6$ ./pmtv-secuencial 20
--- Matriz ---
9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
0 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
0 0 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
0 0 0 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
0 0 0 0 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
0 0 0 0 0 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
0 0 0 0 0 0 9 9 9 9 9 9 9 9 9 9 9 9 9 9
0 0 0 0 0 0 0 9 9 9 9 9 9 9 9 9 9 9 9 9
0 0 0 0 0 0 0 0 9 9 9 9 9 9 9 9 9 9 9 9
0 0 0 0 0 0 0 0 0 9 9 9 9 9 9 9 9 9 9 9
0 0 0 0 0 0 0 0 0 0 9 9 9 9 9 9 9 9 9 9
0 0 0 0 0 0 0 0 0 0 0 9 9 9 9 9 9 9 9 9
0 0 0 0 0 0 0 0 0 0 0 0 9 9 9 9 9 9 9 9
0 0 0 0 0 0 0 0 0 0 0 0 0 9 9 9 9 9 9 9
0 0 0 0 0 0 0 0 0 0 0 0 0 0 9 9 9 9 9 9
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 9 9 9 9 9
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 9 9 9 9
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 9 9 9
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 9 9
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 9
Vector:
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
Resultado:
360 342 324 306 288 270 252 234 216 198 180 162 144 126 108 90 72 54 36 18
Tamaño de la matriz y vector:20  / Tiempo(seg.) 0.000000566
alex@alex-pc:~/UNI/AC/PRÁCTICAS/OPEN_MPI/bp3/ej6$
```

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) El valor por defecto usado por OpenMP para chunk con static no está establecido , sin embargo , para dynamic es 1 , al igual que para guided.

b) n.º Chunk * n.º Fila

c) En static y dynamic se ejecutan las operaciones a la vez ó 1, en guided el último no será igual.

DESCOMPOSICIÓN DE DOMINIO: Recorriendo una fila diferente de la matriz , cada thread va calculando el valor del vector resultado.

CAPTURA CÓDIGO FUENTE: `pmtv-OpenMP.c`

```
#include <stdlib.h>
#include <stdio.h>
#ifdef _OPENMP
    #include <omp.h>
#else
    #define omp_get_thread_num() 0
    #define omp_get_num_threads() 1
    #define omp_set_num_threads(int)
    #define omp_in_parallel() 0
    #define omp_set_dynamic(int)
#endif

int main(int argc, char** argv){
    double t_ini, t_final, t_total;

    if (argc<2){
        printf("Faltan no componentes del vector\n");
        exit(-1);
    }

    int i, j;
    int *v, *mv, **m;
    unsigned int N = atoi(argv[1]);

    v = (int*) malloc(N*sizeof(int));
    mv = (int*) malloc(N*sizeof(int));
    m = (int **)malloc(N*sizeof(int *));

    for (i=0;i<N;i++)
        m[i] = (int *) malloc (N*sizeof(int));

    if ( v==NULL || mv==NULL || m==NULL ){
        printf("Error en la reserva de espacio para la matriz y el vector\n");
        exit(-2);
    }

    for(i=0; i<N; i++)
        if(m[i] == NULL){
            printf("Error en la reserva de espacio para la matriz y el vector\n");
            exit(-3);
        }

    #pragma omp parallel for schedule(runtime)
    for(i=0; i<N; i++){
        for(j=i; j<N; j++){
            m[i][j] = 9;
        }

        v[i] = 2;
        mv[i] = 0;
```

```
    }

    t_ini = omp_get_wtime();
    #pragma omp parallel for schedule(runtime)
    for(i=0; i<N; i++){
        for(j=i; j<N;j++){
            mv[i] += m[i][j] * v[i];
        }
    }

    t_final = omp_get_wtime();
    t_total = t_final - t_ini;


    printf("Tamaño de la matriz y vector:%d\t / Tiempo(seg.) %11.9f\n", N, t_total);
    for(i=0; i<N; i++)
        free(m[i]);
    free(mv);
    free(m);
    free(v);

    return 0;
}
```

CAPTURAS DE PANTALLA:

```
[AlejandroMolinaCriado a2estudiante13@atcgrid:~/bp3/ej7] 2020-05-06 miércoles$ gcc -O2 -fopenmp -o pmtv-OpenMP pmtv-OpenMP.c
[AlejandroMolinaCriado a2estudiante13@atcgrid:~/bp3/ej7] 2020-05-06 miércoles$ sh script.sh
static y chunk por defecto
Tamaño de la matriz y vector:15360      / Tiempo(seg.) 0.021829128
static y chunk 1
Tamaño de la matriz y vector:15360      / Tiempo(seg.) 0.017621994
static y chunk 64
Tamaño de la matriz y vector:15360      / Tiempo(seg.) 0.018184327
dynamic y chunk por defecto
Tamaño de la matriz y vector:15360      / Tiempo(seg.) 0.019483313
dynamic y chunk 1
Tamaño de la matriz y vector:15360      / Tiempo(seg.) 0.019515719
dynamic y chunk 64
Tamaño de la matriz y vector:15360      / Tiempo(seg.) 0.017026931
guided y chunk por defecto
Tamaño de la matriz y vector:15360      / Tiempo(seg.) 0.022156496
guided y chunk 1
Tamaño de la matriz y vector:15360      / Tiempo(seg.) 0.022154812
guided y chunk 64
Tamaño de la matriz y vector:15360      / Tiempo(seg.) 0.022132754
[AlejandroMolinaCriado a2estudiante13@atcgrid:~/bp3/ej7] 2020-05-06 miércoles$
```

TABLA RESULTADOS, SCRIPT Y GRÁFICA atcgrid

SCRIPT: pmvt-OpenMP_atcgrid.sh

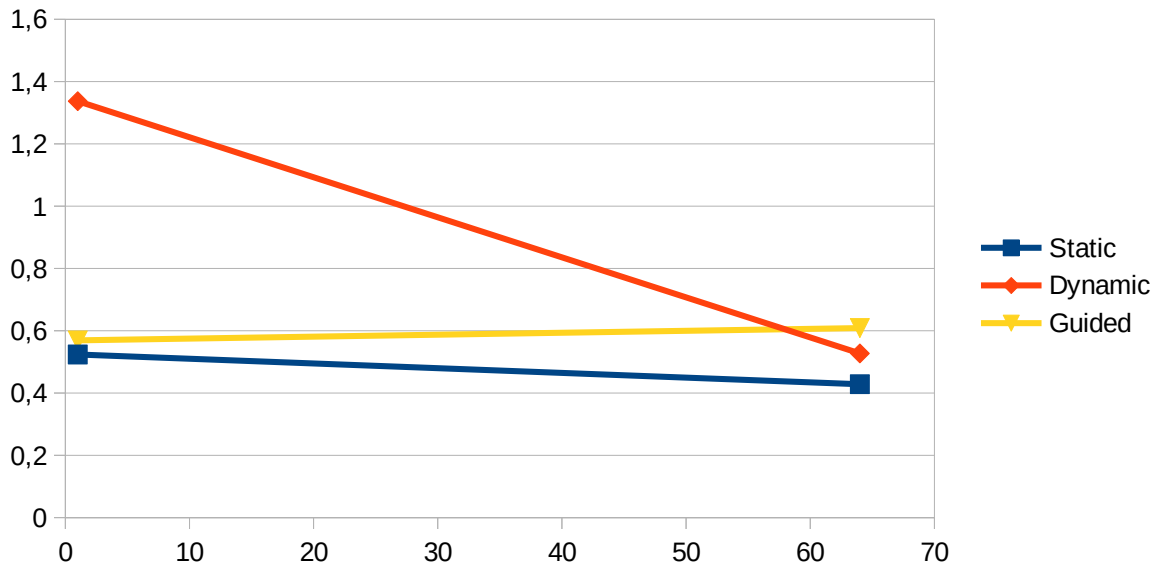
```
#!/bin/bash
export OMP_SCHEDULE="static"
echo "static y chunk por defecto"
./pmtv-OpenMP 15360
export OMP_SCHEDULE="static,1"
echo "static y chunk 1"
./pmtv-OpenMP 15360

export OMP_SCHEDULE="static,64"
echo "static y chunk 64"
./pmtv-OpenMP 15360
```

```
export OMP_SCHEDULE="dynamic"
echo "dynamic y chunk por defecto"
./pmtv-OpenMP 15360
export OMP_SCHEDULE="dynamic,1"
echo "dynamic y chunk 1"
./pmtv-OpenMP 15360
export OMP_SCHEDULE="dynamic,64"
echo "dynamic y chunk 64"
./pmtv-OpenMP 15360
export OMP_SCHEDULE="guided"
echo "guided y chunk por defecto"
./pmtv-OpenMP 15360
export OMP_SCHEDULE="guided,1"
echo "guided y chunk 1"
./pmtv-OpenMP 15360
export OMP_SCHEDULE="guided,64"
echo "guided y chunk 64"
./pmtv-OpenMP 15360
```

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

Chunk	Static	Dynamic	Guided
por defecto	0.760042655	1.073098731	0.575948207
1	0.523922035	1.337174697	0.568946962
64	0.428550951	0.527296656	0.608719275

Chunk	Static	Dynamic	Guided
por defecto	0.772004018	1.126207189	0.573844481
1	0.654055489	1.145340181	0.345992382
64	0.780204687	0.727934329	0.652265645



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`

```
/*
    *   Alejandro Molina Criado
    *   BP3 Arquitectura de Computadores
    *   06/05/2020
    *   Para compilar : gcc -O2 -fopenmp -o if-clausule-modificado if-clausule-
modificado.c
*/

#include <stdlib.h>
#include <stdio.h> // biblioteca donde se encuentra la función printf()
#ifdef _OPENMP
    #include <omp.h>
#else
    #define omp_get_thread_num() 0
    #define omp_get_num_threads() 1
    #define omp_set_num_threads(int)
    #define omp_in_parallel() 0
    #define omp_set_dynamic(int)
#endif

int main(int argc, char** argv){
    int i, j, k;
    double t_ini, t_fin, t_total;

    if (argc<2){
        printf("ERROR: Faltan no componentes del vector\n");
        exit(-1);
    }

    unsigned int N = atoi(argv[1]); // Máximo N =2^32-1=4294967295 (sizeof(unsigned int) =
4 B)

    int **mr, **m1, **m2;
    m1 = (int **) malloc(N*sizeof(int *));
    m2 = (int **) malloc(N*sizeof(int *));
    mr = (int **) malloc(N*sizeof(int *));

    for (i=0;i<N;i++){
        mr[i] = (int *) malloc (N*sizeof(int));
        m1[i] = (int *) malloc (N*sizeof(int));
        m2[i] = (int *) malloc (N*sizeof(int));
    }

    if ( mr==NULL || m1==NULL || m2==NULL ){
        printf("Error en la reserva de espacio para la matriz y el vector\n");
        exit(-2);
    }

    for(i=0; i<N; i++)
        if(mr[i] == NULL || m1[i] == NULL || m2[i] == NULL){
            printf("Error en la reserva de espacio para la matriz y el vector\n");
            exit(-3);
```

```
        }

    for(i=0; i<N; i++){
        for(j=i; j<N; j++){
            mr[i][j] = 0;
            m1[i][j] = 28+i;
            m2[i][j] = 2+i;
        }
    }

    t_ini = omp_get_wtime();
    for(i=0; i<N; i++){
        for(j=0; j<N;j++){
            for(k=0; k<N; k++){
                mr[i][j] += m1[i][k] * m2[k][j];
            }
        }
    }

    t_fin = omp_get_wtime();

    t_total = t_fin - t_ini;

    printf("Tamaño de la matriz y vector:%d\t / Tiempo(seg.) %11.9f\n", N, t_total);

    printf("t = %11.9f\t Primera línea= %d\t Última línea= %d\n", t_total, mr[0][0], mr[N-
1][N-1]);

    return 0;
}
```

CAPTURAS DE PANTALLA:

```
alex@alex-pc:~/UNI/AC/PRÁCTICAS/OPEN_MPI/bp3/ej8$ gcc -O2 -fopenmp -o pmm-secuencial pmm-secuencial.c
alex@alex-pc:~/UNI/AC/PRÁCTICAS/OPEN_MPI/bp3/ej8$ ./pmm-secuencial 8
Tamaño de la matriz y vector:8   / Tiempo(seg.) 0.000002100
t = 0.000002100  Primera linea= -2081855484      Última linea= 315
alex@alex-pc:~/UNI/AC/PRÁCTICAS/OPEN_MPI/bp3/ej8$ ./pmm-secuencial 15
Tamaño de la matriz y vector:15  / Tiempo(seg.) 0.000010276
t = 0.000010276  Primera linea= 56       Última linea= 672
alex@alex-pc:~/UNI/AC/PRÁCTICAS/OPEN_MPI/bp3/ej8$ ./pmm-secuencial 9
Tamaño de la matriz y vector:9   / Tiempo(seg.) 0.000002624
t = 0.000002624  Primera linea= -2081855484      Última linea= 360
alex@alex-pc:~/UNI/AC/PRÁCTICAS/OPEN_MPI/bp3/ej8$
```

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: Todas las hebras recorren las columnas de la primera matriz , aunque cada una itera en distintas filas dentro.
Todas las hebras recorren todas las filas y columnas de la matriz 2.

CAPTURA CÓDIGO FUENTE: pmm-OpenMP.c

```
/*
    *   Alejandro Molina Criado
    *   BP3 Arquitectura de Computadores
    *   06/05/2020
    *   Para compilar : gcc -O2 -fopenmp -o if-clausule-modificado if-clausule-
modificado.c
*/

#include <stdlib.h>
#include <stdio.h> // biblioteca donde se encuentra la función printf()
#ifdef _OPENMP
    #include <omp.h>
#else
    #define omp_get_thread_num() 0
    #define omp_get_num_threads() 1
    #define omp_set_num_threads(int)
    #define omp_in_parallel() 0
    #define omp_set_dynamic(int)
#endif

int main(int argc, char** argv){
    int i, j, k;
    double t_ini, t_fin, t_total;

    if (argc<2){
        printf("ERROR: Faltan no componentes del vector\n");
        exit(-1);
    }

    unsigned int N = atoi(argv[1]); // Máximo N =2^32-1=4294967295 (sizeof(unsigned int) =
4 B)

        int **mr, **m1, **m2;
    m1 = (int **) malloc(N*sizeof(int *));
    m2 = (int **) malloc(N*sizeof(int *));
    mr = (int **) malloc(N*sizeof(int *));

    for (i=0;i<N;i++){
        mr[i] = (int *) malloc (N*sizeof(int));
        m1[i] = (int *) malloc (N*sizeof(int));
        m2[i] = (int *) malloc (N*sizeof(int));
    }

    if ( mr==NULL || m1==NULL || m2==NULL ){
        printf("Error en la reserva de espacio para la matriz y el vector\n");
        exit(-2);
    }

    for(i=0; i<N; i++)
        if(mr[i] == NULL || m1[i] == NULL || m2[i] == NULL){
            printf("Error en la reserva de espacio para la matriz y el vector\n");
            exit(-3);
        }

    #pragma omp parallel for private(j)
    for(i=0; i<N; i++){
        for(j=i; j<N; j++){
            mr[i][j] = 0;
            m1[i][j] = 2+i;
            m2[i][j] = 2+i;
```

```
        }
    }


    t_ini = omp_get_wtime();

    #pragma omp parallel for private(j,k)
    for(i=0; i<N; i++){
        for(j=0; j<N;j++){
            for(k=0; k<N; k++){
                mr[i][j] += m1[i][k] * m2[k][j];
            }
        }
    }

    t_fin = omp_get_wtime();
    t_total = t_fin - t_ini;


    printf("Tamaño de la matriz y vector:%d\t / Tiempo(seg.) %11.9f\n", N, t_total);
    printf("Tiempo = %11.9f\t Primera línea= %d\t Última línea= %d\n", t_total, mr[0][0],
mr[N-1][N-1]);

    return 0;
}
```

CAPTURAS DE PANTALLA:

```
alex@alex-pc:~/UNI/AC/PRÁCTICAS/OPEN_MPI/bp3/ej9$ gcc -O2 -fopenmp -o pmm-openMP pmm-openMP.c
alex@alex-pc:~/UNI/AC/PRÁCTICAS/OPEN_MPI/bp3/ej9$ ./pmm-openMP 8
Tamaño de la matriz y vector:8   / Tiempo(seg.) 0.028103332
Tiempo = 0.028103332     Primera línea= -2081855484      Última línea= 315
alex@alex-pc:~/UNI/AC/PRÁCTICAS/OPEN_MPI/bp3/ej9$ ./pmm-openMP 15
Tamaño de la matriz y vector:15  / Tiempo(seg.) 0.000017742
Tiempo = 0.000017742     Primera línea= 56       Última línea= 672
alex@alex-pc:~/UNI/AC/PRÁCTICAS/OPEN_MPI/bp3/ej9$ ./pmm-openMP 9
Tamaño de la matriz y vector:9   / Tiempo(seg.) 0.022602725
Tiempo = 0.022602725     Primera línea= -2081855484      Última línea= 360
alex@alex-pc:~/UNI/AC/PRÁCTICAS/OPEN_MPI/bp3/ej9$
```

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:

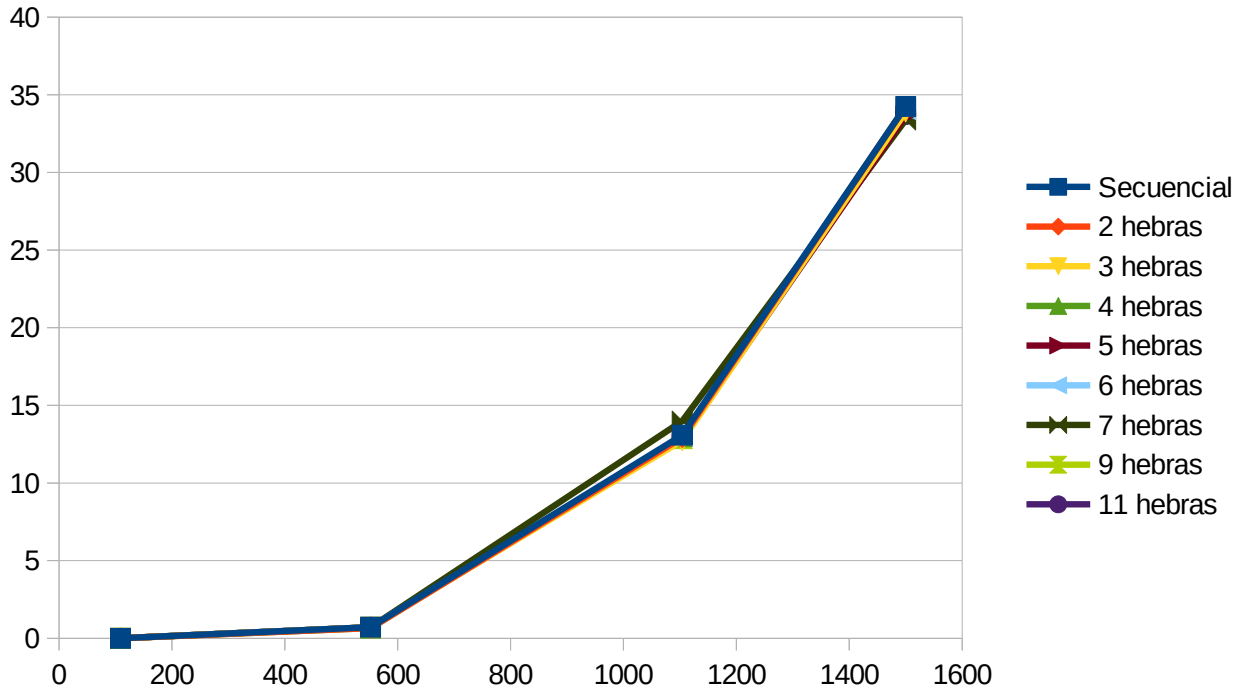
SCRIPT: pmm-OpenMP_atcgrid.sh

```
#!/bin/bash
./pmm-OpenMP 109
./pmm-OpenMP 552
./pmm-OpenMP 1104
./pmm-OpenMP 1500
```

EJECUCIÓN:

```
[AlejandroMolinaCriado a2estudiante13@atcgrid:~/bp3/ej10] 2020-05-06 miércoles$ ls
pmm-OpenMP  pmm-OpenMP_atcgrid.h  pmm-OpenMP.c  pmm-secuencial  pmm-secuencial.c
[AlejandroMolinaCriado a2estudiante13@atcgrid:~/bp3/ej10] 2020-05-06 miércoles$ sh pmm-OpenMP_atcgrid.h
Tamaño de la matriz y vector:109         / Tiempo(seg.) 0.000208396
Tiempo = 0.000208396     Primera linea= 4        Última linea= 12100
Tamaño de la matriz y vector:552         / Tiempo(seg.) 0.034279589
Tiempo = 0.034279589     Primera linea= 4        Última linea= 305809
Tamaño de la matriz y vector:1104        / Tiempo(seg.) 1.266986892
Tiempo = 1.266986892     Primera linea= 4        Última linea= 1221025
Tamaño de la matriz y vector:1500        / Tiempo(seg.) 3.504284047
Tiempo = 3.504284047     Primera linea= 4        Última linea= 2253001
[AlejandroMolinaCriado a2estudiante13@atcgrid:~/bp3/ej10] 2020-05-06 miércoles$
```

Tamaño	Secuencial	2 hebras	3 hebras	4 hebras	5 hebras	6 hebras	7 hebras	9 hebras	11 hebras
109	0,002483451	0,002812526	0,003597185	0,002702527	0,003643119	0,003603627	34,057229674	0,005678881	0,003729716
552	0,71985091	0,653338419	0,712661926	0,653438419	0,710088035	0,711024102	0,711144737	0,711690488	0,713123116
1104	13,081601447	12,92992969	12,734022955	12,97992969	12,86903108	12,969143968	13,989348485	12,856289809	12,983831757
1500	34,250348011	34,244362811	33,893699653	34,244362811	33,642514303	34,057229674	33,37246521	34,064056889	33,728118626



ESTUDIO DE ESCALABILIDAD EN PC local

SCRIPT: pmm-OpenMP_pclocal.sh

```
#!/bin/bash

echo "-- Ejecucion Secuencial --"
./pmm-secuencial 109
./pmm-secuencial 552
./pmm-secuencial 1104
./pmm-secuencial 1500

export OMP_NUM_THREADS=2
echo "\n-- Ejecucion paralela con 2 hebras: "

./pmm-OpenMP 109
./pmm-OpenMP 552
./pmm-OpenMP 1104
./pmm-OpenMP 1500

export OMP_NUM_THREADS=3
echo "\n-- Ejecucion paralela con 3 hebras: "

./pmm-OpenMP 109
./pmm-OpenMP 552
./pmm-OpenMP 1104
./pmm-OpenMP 1500

export OMP_NUM_THREADS=4
echo "\n-- Ejecucion paralela con 4 hebras: "

./pmm-OpenMP 109
```

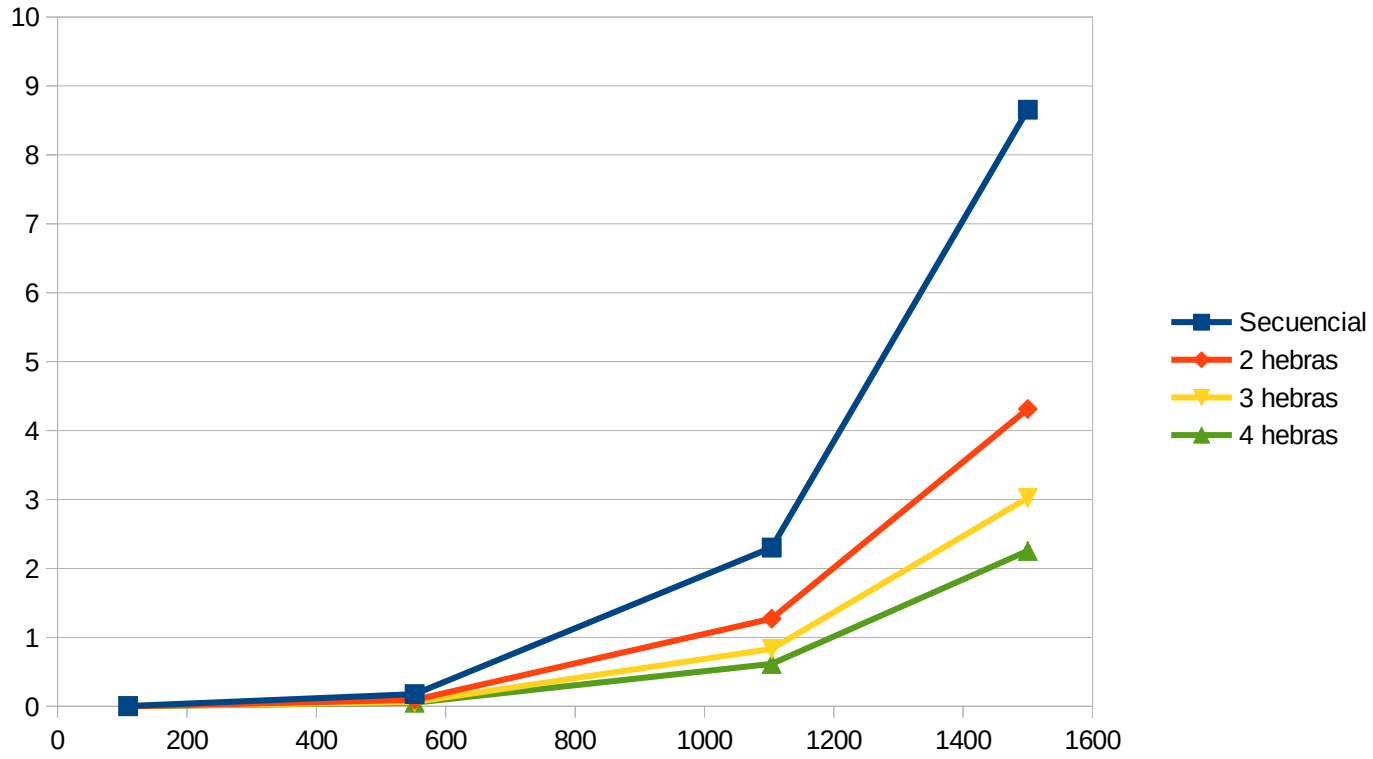
```
./pmm-OpenMP 552
./pmm-OpenMP 1104
./pmm-OpenMP 1500
```

EJECUCIÓN

```
alex@alex-pc:~/UNI/AC/PRÁCTICAS/OPEN_MPI/bp3/ej10$ gcc -O2 -fopenmp -o pmm-OpenMP pmm-OpenMP.c
alex@alex-pc:~/UNI/AC/PRÁCTICAS/OPEN_MPI/bp3/ej10$ gcc -O2 -fopenmp -o pmm-secuencial pmm-secuencial.c
alex@alex-pc:~/UNI/AC/PRÁCTICAS/OPEN_MPI/bp3/ej10$ sh pmm-OpenMP_pclocal.sh
-- Ejecucion Secuencial --
Tamaño de la matriz y vector:109         / Tiempo(seg.) 0.003591861
t = 0.003591861  Primera linea= 56       Última linea= 14960
Tamaño de la matriz y vector:552         / Tiempo(seg.) 0.176656448
t = 0.176656448  Primera linea= 56       Última linea= 320187
Tamaño de la matriz y vector:1104        / Tiempo(seg.) 2.302397709
t = 2.302397709  Primera linea= 56       Última linea= 1249755
Tamaño de la matriz y vector:1500        / Tiempo(seg.) 8.655371133
t = 8.655371133  Primera linea= 56       Última linea= 2292027

-- Ejecucion paralela con 2 hebras:
Tamaño de la matriz y vector:109         / Tiempo(seg.) 0.000569429
Tiempo = 0.000569429     Primera linea= 4        Última linea= 12100
Tamaño de la matriz y vector:552         / Tiempo(seg.) 0.092871754
Tiempo = 0.092871754     Primera linea= 4        Última linea= 305809
Tamaño de la matriz y vector:1104        / Tiempo(seg.) 1.271545863
Tiempo = 1.271545863     Primera linea= 4        Última linea= 1221025
Tamaño de la matriz y vector:1500        / Tiempo(seg.) 4.312967895
Tiempo = 4.312967895     Primera linea= 4        Última linea= 2253001

-- Ejecucion paralela con 3 hebras:
Tamaño de la matriz y vector:109         / Tiempo(seg.) 0.000384655
Tiempo = 0.000384655     Primera linea= 4        Última linea= 12100
Tamaño de la matriz y vector:552         / Tiempo(seg.) 0.062723760
Tiempo = 0.062723760     Primera linea= 4        Última linea= 305809
Tamaño de la matriz y vector:1104        / Tiempo(seg.) 0.831486583
Tiempo = 0.831486583     Primera linea= 4        Última linea= 1221025
Tamaño de la matriz y vector:1500        / Tiempo(seg.) 3.022712990
Tiempo = 3.022712990     Primera linea= 4        Última linea= 2253001

-- Ejecucion paralela con 4 hebras:
Tamaño de la matriz y vector:109         / Tiempo(seg.) 0.000293227
Tiempo = 0.000293227     Primera linea= 4        Última linea= 12100
Tamaño de la matriz y vector:552         / Tiempo(seg.) 0.051864293
Tiempo = 0.051864293     Primera linea= 4        Última linea= 305809
Tamaño de la matriz y vector:1104        / Tiempo(seg.) 0.614871099
Tiempo = 0.614871099     Primera linea= 4        Última linea= 1221025
Tamaño de la matriz y vector:1500        / Tiempo(seg.) 2.254855054
Tiempo = 2.254855054     Primera linea= 4        Última linea= 2253001
alex@alex-pc:~/UNI/AC/PRÁCTICAS/OPEN_MPI/bp3/ej10$
```

TABLA RESULTADOS, Y GRÁFICA (PC)

Tamaño	Secuencial	2 hebras	3 hebras	4 hebras
109	0,003591861	0,000569429	0,00384655	0,000293227
552	0,176656448	0,092871754	0,062723760	0,051864293
1104	2,302397709	1,271545863	0,831486583	0,614871099
1500	8,655371133	4,312967895	3,022712990	2,254855054


10
9
8
7
6
5
4
3
2
1
0
0
200
400
600
800
1000
1200
1400
1600
Secuencial
2 hebras
3 hebras
4 hebras