

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): Ernesto Serrano Collado

Grupo de prácticas: A2

Fecha de entrega: 10/05/2016

Fecha evaluación en clase: 13/05/2016

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>

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

int main(int argc, char **argv)
{
    int i, n=20, tid, x;
    int a[n], suma=0, sumalocal;

    if(argc < 2)
    {
        fprintf(stderr, "[ERROR]-Falta iteraciones\n");
        exit(-1);
    }
    if(argc < 3)
    {
        fprintf(stderr, "[ERROR]-Falta numero threads \n");
        exit(-1);
    }

    n = atoi(argv[1]);
    x = atoi(argv[2]);

    if (n>20)
        n=20;

    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);
    }
}

```

CAPTURAS DE PANTALLA:

```

bender:practica3 ernesto$ bin/if-clauseModificado 5 4
thread 1 suma de a[2]=2 sumalocal=2
thread 2 suma de a[3]=3 sumalocal=3
thread 0 suma de a[0]=0 sumalocal=0
thread 3 suma de a[4]=4 sumalocal=4
thread 0 suma de a[1]=1 sumalocal=1
thread master=0 imprime suma=10
bender:practica3 ernesto$ bin/if-clauseModificado 4 4
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
bender:practica3 ernesto$ bin/if-clauseModificado 6 4
thread 0 suma de a[0]=0 sumalocal=0
thread 2 suma de a[4]=4 sumalocal=4
thread 1 suma de a[2]=2 sumalocal=2
thread 3 suma de a[5]=5 sumalocal=5
thread 0 suma de a[1]=1 sumalocal=1
thread 1 suma de a[3]=3 sumalocal=5
thread master=0 imprime suma=15
bender:practica3 ernesto$

```

RESPUESTA: Como podemos ver en el código, solo se paraleliza cuando el número de iteraciones es mayor de 4, con `num_threads()` le podemos indicar el número de hebras que se crearán sin tener que recompilar, aunque eso sí, como tenemos el `if>4` sólo se crearán cuando tengamos al menos 5 iteraciones.

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

- iteraciones: 16 (0,...15)
- chunk= 1, 2 y 4

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

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

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

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

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

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

RESPUESTA: usando `static` todas las tareas se reparten equitativamente usando `round-robin`, con `dynamic` y `guided` el reparto no se sabe como se distribuirá, pero se sabe que el

tamaño de las tareas vendrá definido por el chunk, es decir, cada hebra hará chunk iteraciones.

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
#define omp_get_num_threads() 1
#define omp_set_num_threads(int)
#endif

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

    if(argc < 3)
    {
        fprintf(stderr, "\nFalta iteraciones o chunk \n");
        exit(-1);
    }

    // omp_set_num_threads(4);

    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 'parallel for':\n");
            printf(" static = 1, dynamic = 2, guided = 3, auto = 4\n");
            omp_get_schedule(&schedule_type, &chunk_value);
            printf(" dyn-var: %d, nthreads-var:%d, thread-limit-var:%d,run-sched-var:
%d, chunk: %d\n", \
omp_get_dynamic(), omp_get_max_threads(),
omp_get_thread_limit(), schedule_type, chunk_value);
        }

    }

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

```

printf("    static = 1, dynamic = 2, guided = 3, auto = 4\n");

omp_get_schedule(&schedule_type, &chunk_value);
printf("    dyn-var: %d, nthreads-var:%d, thread-limit-var:%d,run-sched-var: %d,
chunk: %d\n", \
        omp_get_dynamic(),omp_get_max_threads(), omp_get_thread_limit(),
schedule_type, chunk_value);
}

```

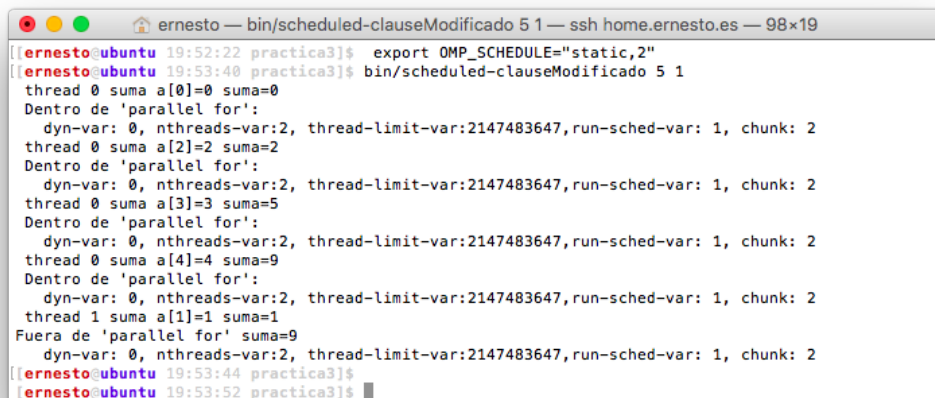
CAPTURAS DE PANTALLA:



```

ernesto — bin/scheduled-clauseModificado 5 1 — ssh home.ernesto.es — 98x19
[ernesto@ubuntu 19:52:21 practica3]$ bin/scheduled-clauseModificado 5 1
thread 0 suma a[0]=0 suma=0
Dentro de 'parallel for':
  dyn-var: 0, nthreads-var:2, thread-limit-var:2147483647,run-sched-var: 2, chunk: 1
thread 0 suma a[1]=1 suma=1
Dentro de 'parallel for':
  dyn-var: 0, nthreads-var:2, thread-limit-var:2147483647,run-sched-var: 2, chunk: 1
thread 0 suma a[2]=2 suma=3
Dentro de 'parallel for':
  dyn-var: 0, nthreads-var:2, thread-limit-var:2147483647,run-sched-var: 2, chunk: 1
thread 0 suma a[3]=3 suma=6
Dentro de 'parallel for':
  dyn-var: 0, nthreads-var:2, thread-limit-var:2147483647,run-sched-var: 2, chunk: 1
thread 0 suma a[4]=4 suma=10
Dentro de 'parallel for':
  dyn-var: 0, nthreads-var:2, thread-limit-var:2147483647,run-sched-var: 2, chunk: 1
Fuera de 'parallel for' suma=10
  dyn-var: 0, nthreads-var:2, thread-limit-var:2147483647,run-sched-var: 2, chunk: 1
[ernesto@ubuntu 19:52:22 practica3]$

```



```

ernesto — bin/scheduled-clauseModificado 5 1 — ssh home.ernesto.es — 98x19
[[ernesto@ubuntu 19:52:22 practica3]$ export OMP_SCHEDULE="static,2"
[ernesto@ubuntu 19:53:40 practica3]$ bin/scheduled-clauseModificado 5 1
thread 0 suma a[0]=0 suma=0
Dentro de 'parallel for':
  dyn-var: 0, nthreads-var:2, thread-limit-var:2147483647,run-sched-var: 1, chunk: 2
thread 0 suma a[2]=2 suma=2
Dentro de 'parallel for':
  dyn-var: 0, nthreads-var:2, thread-limit-var:2147483647,run-sched-var: 1, chunk: 2
thread 0 suma a[3]=3 suma=5
Dentro de 'parallel for':
  dyn-var: 0, nthreads-var:2, thread-limit-var:2147483647,run-sched-var: 1, chunk: 2
thread 0 suma a[4]=4 suma=9
Dentro de 'parallel for':
  dyn-var: 0, nthreads-var:2, thread-limit-var:2147483647,run-sched-var: 1, chunk: 2
thread 1 suma a[1]=1 suma=1
Fuera de 'parallel for' suma=9
  dyn-var: 0, nthreads-var:2, thread-limit-var:2147483647,run-sched-var: 1, chunk: 2
[ernesto@ubuntu 19:53:44 practica3]$
[ernesto@ubuntu 19:53:52 practica3]$

```

```

ernesto — bin/scheduled-clauseModificado 5 1 — ssh home.ernesto.es — 98x10
[[ernesto@ubuntu 19:53:52 practica3]$ export OMP_NUM_THREADS=10
[[ernesto@ubuntu 19:54:29 practica3]$ bin/scheduled-clauseModificado 5 1
thread 2 suma a[0]=0 suma=0
thread 3 suma a[1]=1 suma=1
thread 4 suma a[2]=2 suma=2
thread 5 suma a[3]=3 suma=3
thread 6 suma a[4]=4 suma=4
Fuera de 'parallel for' suma=4
dyn-var: 0, nthreads-var:10, thread-limit-var:2147483647,run-sched-var: 1, chunk: 2
[[ernesto@ubuntu 19:54:32 practica3]$

```

```

ernesto — bin/scheduled-clauseModificado 5 1 — ssh home.ernesto.es — 98x18
[[ernesto@ubuntu 19:55:18 practica3]$ export OMP_THREAD_LIMIT=4
[[ernesto@ubuntu 19:55:56 practica3]$ bin/scheduled-clauseModificado 5 1
thread 0 suma a[0]=0 suma=0
Dentro de 'parallel for':
dyn-var: 1, nthreads-var:10, thread-limit-var:4,run-sched-var: 1, chunk: 2
thread 0 suma a[2]=2 suma=2
Dentro de 'parallel for':
dyn-var: 1, nthreads-var:10, thread-limit-var:4,run-sched-var: 1, chunk: 2
thread 0 suma a[3]=3 suma=5
Dentro de 'parallel for':
dyn-var: 1, nthreads-var:10, thread-limit-var:4,run-sched-var: 1, chunk: 2
thread 0 suma a[4]=4 suma=9
Dentro de 'parallel for':
dyn-var: 1, nthreads-var:10, thread-limit-var:4,run-sched-var: 1, chunk: 2
thread 1 suma a[1]=1 suma=1
Fuera de 'parallel for' suma=9
dyn-var: 1, nthreads-var:10, thread-limit-var:4,run-sched-var: 1, chunk: 2
[[ernesto@ubuntu 19:55:59 practica3]$

```

```

ernesto — bin/scheduled-clauseModificado 5 1 — ssh home.ernesto.es — 98x18
[[ernesto@ubuntu 19:54:32 practica3]$ export OMP_DYNAMIC=TRUE
[[ernesto@ubuntu 19:55:15 practica3]$ bin/scheduled-clauseModificado 5 1
thread 0 suma a[0]=0 suma=0
Dentro de 'parallel for':
dyn-var: 1, nthreads-var:10, thread-limit-var:2147483647,run-sched-var: 1, chunk: 2
thread 0 suma a[2]=2 suma=2
Dentro de 'parallel for':
dyn-var: 1, nthreads-var:10, thread-limit-var:2147483647,run-sched-var: 1, chunk: 2
thread 0 suma a[3]=3 suma=5
Dentro de 'parallel for':
dyn-var: 1, nthreads-var:10, thread-limit-var:2147483647,run-sched-var: 1, chunk: 2
thread 0 suma a[4]=4 suma=9
Dentro de 'parallel for':
dyn-var: 1, nthreads-var:10, thread-limit-var:2147483647,run-sched-var: 1, chunk: 2
thread 1 suma a[1]=1 suma=1
Fuera de 'parallel for' suma=9
dyn-var: 1, nthreads-var:10, thread-limit-var:2147483647,run-sched-var: 1, chunk: 2
[[ernesto@ubuntu 19:55:18 practica3]$

```

RESPUESTA: Podemos ver que nos devuelve los mismos valores 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 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.

CÓDIGO FUENTE: scheduled-clauseModificado4.c

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

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

    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 'parallel for':\n");
            printf(" static = 1, dynamic = 2, guided = 3, auto = 4\n");
            omp_get_schedule(&schedule_type, &chunk_value);
            printf(" dyn-var: %d, nthreads-var:%d, thread-limit-var:%d, run-sched-var:
%d, chunk: %d\n", \
                omp_get_dynamic(), \
                omp_get_max_threads(), omp_get_thread_limit(), \
                schedule_type, chunk_value);

            printf(" get_num_threads: %d, get_num_procs: %d, in_parallel():%d \n", \
                omp_get_num_threads(), omp_get_num_procs(), omp_in_parallel());
        }
    }

    printf("Fuera de 'parallel for' suma=%d\n", suma);
    printf(" static = 1, dynamic = 2, guided = 3, auto = 4\n");
    omp_get_schedule(&schedule_type, &chunk_value);
    printf(" dyn-var: %d, nthreads-var:%d, thread-limit-var:%d, run-sched-var: %d,
chunk: %d\n", \
        omp_get_dynamic(), \
        omp_get_max_threads(), omp_get_thread_limit(), \
        schedule_type, chunk_value);
}
```

```
printf(" get_num_threads: %d,get_num_procs: %d,in_parallel():%d \n", \
      omp_get_num_threads(),omp_get_num_procs(),omp_in_parallel());
}
```

CAPTURAS DE PANTALLA:

```
ernesto@ubuntu 20:14:55 practica3]$ bin/scheduled-clauseModificado4 5 1
thread 0 suma a[0]=0 suma=0
Dentro de 'parallel for':
  dyn-var: 1, nthreads-var:10, thread-limit-var:4,run-sched-var: 1, chunk: 2
  get_num_threads: 2,get_num_procs: 2,in_parallel():1
  thread 0 suma a[1]=1 suma=1
  Dentro de 'parallel for':
    dyn-var: 1, nthreads-var:10, thread-limit-var:4,run-sched-var: 1, chunk: 2
    get_num_threads: 2,get_num_procs: 2,in_parallel():1
    thread 0 suma a[2]=2 suma=3
    Dentro de 'parallel for':
      dyn-var: 1, nthreads-var:10, thread-limit-var:4,run-sched-var: 1, chunk: 2
      get_num_threads: 2,get_num_procs: 2,in_parallel():1
      thread 0 suma a[3]=3 suma=6
      Dentro de 'parallel for':
        dyn-var: 1, nthreads-var:10, thread-limit-var:4,run-sched-var: 1, chunk: 2
        get_num_threads: 2,get_num_procs: 2,in_parallel():1
        thread 0 suma a[4]=4 suma=10
        Dentro de 'parallel for':
          dyn-var: 1, nthreads-var:10, thread-limit-var:4,run-sched-var: 1, chunk: 2
          get_num_threads: 2,get_num_procs: 2,in_parallel():1
          thread 0 suma a[5]=5 suma=15
          Fuera de 'parallel for' suma=10
          dyn-var: 1, nthreads-var:10, thread-limit-var:4,run-sched-var: 1, chunk: 2
          get_num_threads: 1,get_num_procs: 2,in_parallel():0
ernesto@ubuntu 20:14:55 practica3]$
```

RESPUESTA: La única que se mantiene dentro y fuera de la región paralela es `omp_get_num_procs()`, las otras dos si que varían dependiendo de si están dentro o fuera.

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
    #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, n=200, chunk, a[n], suma=0;
    omp_sched_t schedule_type;
    int chunk_value;

    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;

    // Imprimimos antes del cambio
    printf("Antes del cambio\n");
    printf("    static = 1, dynamic = 2, guided = 3, auto = 4\n");
    omp_get_schedule(&schedule_type, &chunk_value);
    printf("    dyn-var: %d, nthreads-var:%d, thread-limit-var:%d,run-sched-var: %d,
chunk: %d\n" \
        , omp_get_dynamic(), \
        omp_get_max_threads(), omp_get_thread_limit(), \
        schedule_type, chunk_value);

    printf(" get_num_threads: %d,get_num_procs: %d,in_parallel():%d \n", \
        omp_get_num_threads(),omp_get_num_procs(),omp_in_parallel());

    // Cambiamos los valores
    omp_set_dynamic(2);
    omp_set_num_threads(2);
    omp_set_schedule(2, 1);

    #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 'parallel for':\n");
        }
    }

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

    // Imprimimos despues del cambio
    printf("    static = 1, dynamic = 2, guided = 3, auto = 4\n");
    omp_get_schedule(&schedule_type, &chunk_value);
    printf("    dyn-var: %d, nthreads-var:%d, thread-limit-var:%d,run-sched-var: %d,
chunk: %d\n" \
        , omp_get_dynamic(), \
        omp_get_max_threads(), omp_get_thread_limit(), \
        schedule_type, chunk_value);

    printf(" get_num_threads: %d,get_num_procs: %d,in_parallel():%d \n", \
        omp_get_num_threads(),omp_get_num_procs(),omp_in_parallel());
}

```

CAPTURAS DE PANTALLA:

```

ernesto@ubuntu 20:30:22 practica3$ bin/scheduled-clauseModificado5 5 1
Antes del cambio
  dyn-var: 1, nthreads-var:10, thread-limit-var:4,run-sched-var: 1, chunk: 2
get_num_threads: 1,get_num_procs: 2,in_parallel():0
thread 0 suma a[0]=0 suma=0
Dentro de 'parallel for':
thread 0 suma a[1]=1 suma=1
Dentro de 'parallel for':
thread 0 suma a[2]=2 suma=3
Dentro de 'parallel for':
thread 0 suma a[3]=3 suma=6
Dentro de 'parallel for':
thread 0 suma a[4]=4 suma=10
Dentro de 'parallel for':
Fuera de 'parallel for' suma=10
  dyn-var: 1, nthreads-var:2, thread-limit-var:4,run-sched-var: 2, chunk: 1
get_num_threads: 1,get_num_procs: 2,in_parallel():0
ernesto@ubuntu 20:30:24 practica3$

```

RESPUESTA: Como podemos apreciar en la captura, al cambiar los valores vemos que las funciones devuelven los valores establecidos.

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 **argv)
{
    int i, j;

    //Leer argumento de entrada
    if(argc < 2)
    {
        fprintf(stderr, "Falta size\n");
        exit(-1);
    }

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

    // Inicializamos la matriz triangular (superior)
    int *vector, *result, **matrix;
    vector = (int *) malloc(N*sizeof(int)); // malloc necesita el tamaño en bytes
    result = (int *) malloc(N*sizeof(int)); //si no hay espacio suficiente malloc
devuelve NULL
    matrix = (int **) malloc(N*sizeof(int*));

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

    for (i=0; i<N; i++)
    {

```

```

        for (j=i; j<N; j++)
            matrix[i][j] = 2;
        vector[i] = 4;
        result[i]=0;
    }

    // Pintamos la matriz
    printf("Matriz:\n");
    for (i=0; i<N; i++)
    {
        for (j=0; j<N; j++)
        {
            if (j >= i)
                printf("%d ", matrix[i][j]);
            else
                printf("0 ");
        }
        printf("\n");
    }

    // Pintamos el vector
    printf("Vector:\n");
    for (i=0; i<N; i++)
        printf("%d ", vector[i]);
    printf("\n");

    // Obtenemos los resultados
    for (i=0; i<N; i++)
        for (j=i; j<N; j++)
            result[i] += matrix[i][j] * vector[j];

    // Pintamos los resultados
    printf("Resultado:\n");
    for (i=0; i<N; i++)
        printf("%d ", result[i]);
    printf("\n");

    // Liberamos la memoria
    for (i=0; i<N; i++)
        free(matrix[i]);
    free(matrix);
    free(vector);
    free(result);

    return 0;
}

```

CAPTURAS DE PANTALLA:

```

ernesto@ubuntu: 21:01:31 practica3$ bin/pmtv-secuencial 10
Matriz:
2 2 2 2 2 2 2 2 2 2
0 2 2 2 2 2 2 2 2 2
0 0 2 2 2 2 2 2 2 2
0 0 0 2 2 2 2 2 2 2
0 0 0 0 2 2 2 2 2 2
0 0 0 0 0 2 2 2 2 2
0 0 0 0 0 0 2 2 2 2
0 0 0 0 0 0 0 2 2 2
0 0 0 0 0 0 0 0 2 2
0 0 0 0 0 0 0 0 0 2
Vector:
4 4 4 4 4 4 4 4 4 4
Resultado:
80 72 64 56 48 40 32 24 16 8
ernesto@ubuntu: 21:01:34 practica3$ bin/pmtv-secuencial 15
Matriz:
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
0 2 2 2 2 2 2 2 2 2 2 2 2 2 2
0 0 2 2 2 2 2 2 2 2 2 2 2 2 2
0 0 0 2 2 2 2 2 2 2 2 2 2 2 2
0 0 0 0 2 2 2 2 2 2 2 2 2 2 2
0 0 0 0 0 2 2 2 2 2 2 2 2 2 2
0 0 0 0 0 0 2 2 2 2 2 2 2 2 2
0 0 0 0 0 0 0 2 2 2 2 2 2 2 2
0 0 0 0 0 0 0 0 2 2 2 2 2 2 2
0 0 0 0 0 0 0 0 0 2 2 2 2 2 2
0 0 0 0 0 0 0 0 0 0 2 2 2 2 2
0 0 0 0 0 0 0 0 0 0 0 2 2 2 2
0 0 0 0 0 0 0 0 0 0 0 0 2 2 2
0 0 0 0 0 0 0 0 0 0 0 0 0 2 2
Vector:
4 4 4 4 4 4 4 4 4 4 4 4 4 4
Resultado:
120 112 104 96 88 80 72 64 56 48 40 32 24 16 8

```

(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: Todas ofrecen un rendimiento parecido, quizá `static` vaya un poco mejor quizá porque es menos costoso hacer el reparto round-robin, pero no difieren demasiado.

a) `static` no tiene, `dynamic` y `guided` es uno, lo he visto en la documentación en <https://computing.llnl.gov/tutorials/openMP/>

b) hará el numero de `chunk` * numero de fila

c) pues en `guided` que el ultimo no será igual, por lo demas que se ejecutarán o 64 operaciones de golpe o 1

CÓDIGO FUENTE: pmtv-OpenMP.c

```
#include <stdio.h>
#include <stdlib.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)
{
    int i, j, debug=0;

    //Leer argumento de entrada
    if(argc < 2)
    {
        fprintf(stderr, "Falta size [optional debug]\n");
        exit(-1);
    }

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

```

if(argc == 3)
    debug = atoi(argv[2]);

// Inicializamos la matriz triangular (superior)
int *vector, *result, **matrix;
vector = (int *) malloc(N*sizeof(int)); // malloc necesita el tamaño en bytes
result = (int *) malloc(N*sizeof(int)); //si no hay espacio suficiente malloc
devuelve NULL
matrix = (int **) malloc(N*sizeof(int*));

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

for (i=0; i<N; i++)
{
    for (j=i; j<N; j++)
        matrix[i][j] = 2;
    vector[i] = 4;
    result[i]=0;
}

if (debug==1)
{
    // Pintamos la matriz
    printf("Matriz:\n");
    for (i=0; i<N; i++)
    {
        for (j=0; j<N; j++)
        {
            if (j >= i)
                printf("%d ", matrix[i][j]);
            else
                printf("0 ");
        }
        printf("\n");
    }

    // Pintamos el vector
    printf("Vector:\n");
    for (i=0; i<N; i++)
        printf("%d ", vector[i]);
    printf("\n");
}

double start, end, total;
start = omp_get_wtime();

// Obtenemos los resultados

// Usamos runtime para poder variarlo luego con la variable OMP_SCHEDULE
#pragma omp parallel for private(j) schedule(runtime)
for (i=0; i<N; i++)
    for (j=i; j<N; j++)
        result[i] += matrix[i][j] * vector[j];

end = omp_get_wtime();
total = end - start;

if (debug==1)
{
    // Pintamos los resultados
    printf("Resultado:\n");
    for (i=0; i<N; i++)
        printf("%d ", result[i]);
    printf("\n");
}

printf("Tiempo = %11.9f\t Primera = %d\t Ultima=%d\n", total, result[0], result[N-1]);

// Liberamos la memoria
for (i=0; i<N; i++)
    free(matrix[i]);

```

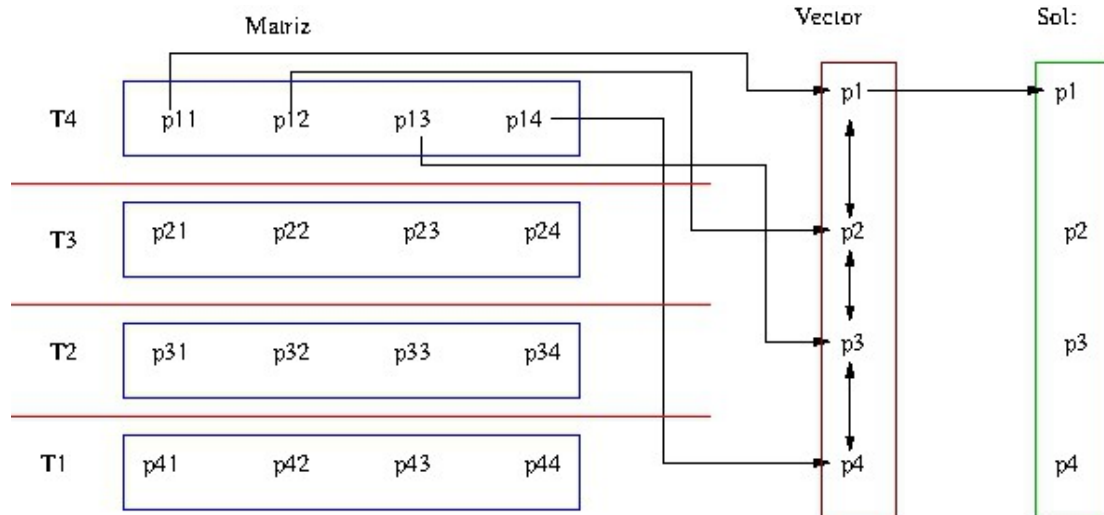
```

    free(matrix);
    free(vector);
    free(result);

    return 0;
}

```

DESCOMPOSICIÓN DE DOMINIO: Se reparten las filas de la matriz y cada thread va calculando un valor del vector final, recorriendo una fila diferente de la matriz.



CAPTURAS DE PANTALLA:
(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```

ernesto — bin/pmtv-OpenMP 10 1 — ssh home.ernesto.es — 102x18
[ernesto@ubuntu 22:12:33 practica3]$ bin/pmtv-OpenMP 10 1
Matriz:
2 2 2 2 2 2 2 2 2 2
0 2 2 2 2 2 2 2 2 2
0 0 2 2 2 2 2 2 2 2
0 0 0 2 2 2 2 2 2 2
0 0 0 0 2 2 2 2 2 2
0 0 0 0 0 2 2 2 2 2
0 0 0 0 0 0 2 2 2 2
0 0 0 0 0 0 0 2 2 2
0 0 0 0 0 0 0 0 2 2
0 0 0 0 0 0 0 0 0 2
Vector:
4 4 4 4 4 4 4 4 4 4
Resultado:
80 72 64 56 48 40 32 24 16 8
Tiempo = 0.000205367 Primera = 80 Ultima=8
[ernesto@ubuntu 22:22:59 practica3]$

```

TABLA RESULTADOS, SCRIPT Y GRÁFICA ATCGRID

SCRIPT: pmtv-OpenMP_atcgrid.sh

```

#!/bin/bash

#PBS -N ej7_atcgrid
#PBS -q ac
echo "Id$PBS_O_WORKDIR usuario del trabajo: $PBS_O_LOGNAME"
echo "Id$PBS_O_WORKDIR 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

export OMP_SCHEDULE="static"
echo "static y chunk por defecto"
$PBS_O_WORKDIR/pmtv-OpenMP 15360

export OMP_SCHEDULE="static,1"
echo "static y chunk 1"
$PBS_O_WORKDIR/pmtv-OpenMP 15360

export OMP_SCHEDULE="static,64"
echo "static y chunk 64"
$PBS_O_WORKDIR/pmtv-OpenMP 15360

export OMP_SCHEDULE="dynamic"
echo "dynamic y chunk por defecto"
$PBS_O_WORKDIR/pmtv-OpenMP 15360

export OMP_SCHEDULE="dynamic,1"
echo "dynamic y chunk 1"
$PBS_O_WORKDIR/pmtv-OpenMP 15360

export OMP_SCHEDULE="dynamic,64"
echo "dynamic y chunk 64"
$PBS_O_WORKDIR/pmtv-OpenMP 15360

export OMP_SCHEDULE="guided"
echo "guided y chunk por defecto"
$PBS_O_WORKDIR/pmtv-OpenMP 15360

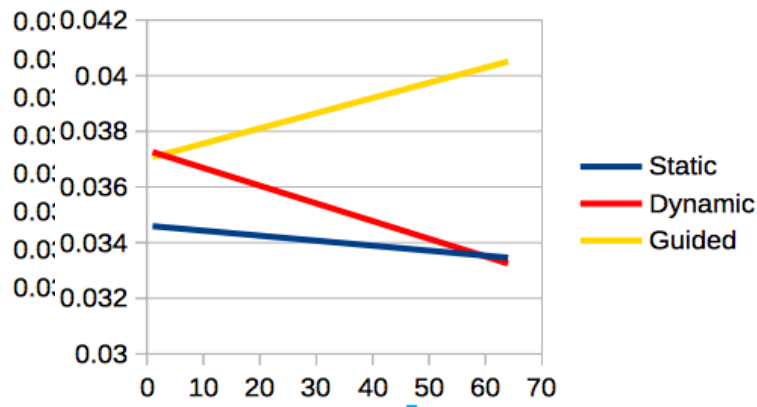
export OMP_SCHEDULE="guided,1"
echo "guided y chunk 1"
$PBS_O_WORKDIR/pmtv-OpenMP 15360

export OMP_SCHEDULE="guided,64"
echo "guided y chunk 64"
$PBS_O_WORKDIR/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=15360** , 12 threads

Chunk	Static	Dynamic	Guided
por defecto	0.046711916	0.036238721	0.038668644
1	0.035047011	0.037187167	0.034859763
64	0.033244486	0.034451319	0.034451054
Chunk	Static	Dynamic	Guided
por defecto	0.058979410	0.036976165	0.041331375
1	0.034590594	0.037256078	0.037069255
64	0.033459864	0.033246293	0.040510096



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

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

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 **argv)
{
    unsigned i, j, k;

    if(argc < 2)
    {
        fprintf(stderr, "falta size\n");
        exit(-1);
    }

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

    int **a, **b, **c;
    a = (int **) malloc(N*sizeof(int));
    b = (int **) malloc(N*sizeof(int));
    c = (int **) malloc(N*sizeof(int));

    for (i=0; i<N; i++)
    {
        a[i] = (int *) malloc(N*sizeof(int));
        b[i] = (int *) malloc(N*sizeof(int));
        c[i] = (int *) malloc(N*sizeof(int));
    }

    // Inicializamos las matrices
    for (i=0; i<N; i++)
    {
        for (j=0; j<N; j++)
        {
            a[i][j] = 0;
            b[i][j] = 2;
            c[i][j] = 2;
        }
    }
}
```



```

struct timespec cgt1,cgt2; double ncgt;

clock_gettime(CLOCK_REALTIME,&cgt1);
// Multiplicacion
for (i=0; i<N; i++)
    for (j=0; j<N; j++)
        for (k=0; k<N; k++)
            a[i][j] += b[i][k] * c[k][j];
clock_gettime(CLOCK_REALTIME,&cgt2);

ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+( double) ((cgt2.tv_nsec-cgt1.tv_nsec)/
(1.e+9));

// Pitamos la primera y la ultima linea de la matriz resultante
printf("Tiempo = %11.9f\t Primera = %d\t Ultima=%d\n",ncgt,a[0][0],a[N-1][N-1]);

// Liberamos la memoria
for (i=0; i<N; i++)
{
    free(a[i]);
    free(b[i]);
    free(c[i]);
}
free(a);
free(b);
free(c);

return 0;
}

```

CAPTURAS DE PANTALLA:
(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

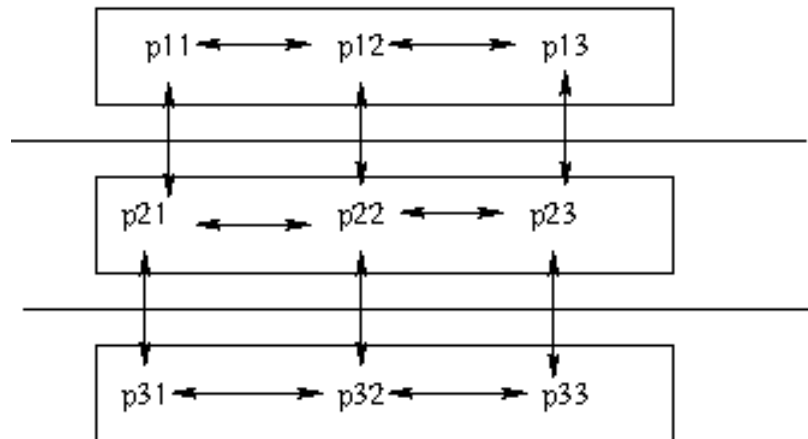
```

ernesto — bin/pmm-secuencial 1000 — ssh home.ernesto.es — 102x11
[ernesto@ubuntu 22:56:05 practica3]$ bin/pmm-secuencial 20
Tiempo = 0.000020095 Primera = 80 Ultima=80
[ernesto@ubuntu 22:56:06 practica3]$ bin/pmm-secuencial 20
Tiempo = 0.000020096 Primera = 80 Ultima=80
[ernesto@ubuntu 22:56:08 practica3]$ bin/pmm-secuencial 100
Tiempo = 0.002162968 Primera = 400 Ultima=400
[ernesto@ubuntu 22:59:10 practica3]$ bin/pmm-secuencial 10
Tiempo = 0.000004411 Primera = 40 Ultima=40
[ernesto@ubuntu 22:59:13 practica3]$ bin/pmm-secuencial 1000
Tiempo = 17.11645035 Primera = 4000 Ultima=4000
[ernesto@ubuntu 22:59:38 practica3]$

```

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: Se reparten las filas de la matriz resultado, cada thread va recorriendo diferentes filas de la primera matriz, pero todas recorren todas sus columnas, y todas las filas y columnas de la segunda matriz

**CÓDIGO FUENTE: pmm-OpenMP.c**

```
#include <stdio.h>
#include <stdlib.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)
{
    unsigned i, j, k;

    if(argc < 2)
    {
        fprintf(stderr, "falta size\n");
        exit(-1);
    }

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

    int **a, **b, **c;
    a = (int **) malloc(N*sizeof(int*));
    b = (int **) malloc(N*sizeof(int*));
    c = (int **) malloc(N*sizeof(int*));

    for (i=0; i<N; i++)
    {
        a[i] = (int *) malloc(N*sizeof(int));
        b[i] = (int *) malloc(N*sizeof(int));
        c[i] = (int *) malloc(N*sizeof(int));
    }

    // Inicializamos las matrices
    #pragma omp parallel for private(j)
    for (i=0; i<N; i++)
    {
        for (j=0; j<N; j++)
        {
            a[i][j] = 0;
            b[i][j] = 2;
            c[i][j] = 2;
        }
    }

    double start, end, total;
```

```

start = omp_get_wtime();

// Multiplicacion
#pragma omp parallel for private(k,j)
for (i=0; i<N; i++)
    for (j=0; j<N; j++)
        for (k=0; k<N; k++)
            a[i][j] += b[i][k] * c[k][j];

end = omp_get_wtime();

total = end - start;

// Pitamos la primera y la ultima linea de la matriz resultante
printf("Tiempo = %11.9f\t Primera = %d\t Ultima=%d\n",total,a[0][0],a[N-1][N-1]);

// Liberamos la memoria
for (i=0; i<N; i++)
{
    free(a[i]);
    free(b[i]);
    free(c[i]);
}
free(a);
free(b);
free(c);

return 0;
}

```

CAPTURAS DE PANTALLA: (ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```

ernesto — bin/pmm-OpenMP 1000 — ssh home.ernesto.es — 102x9
[[ernesto@ubuntu 23:14:31 practica3]$ bin/pmm-OpenMP
falta size
[[ernesto@ubuntu 23:14:43 practica3]$ bin/pmm-OpenMP 10
Tiempo = 0.000032839    Primera = 40    Ultima=40
[[ernesto@ubuntu 23:14:45 practica3]$ bin/pmm-OpenMP 200
Tiempo = 0.013298357    Primera = 800    Ultima=800
[[ernesto@ubuntu 23:14:48 practica3]$ bin/pmm-OpenMP 1000
Tiempo = 9.202124053    Primera = 4000    Ultima=4000
[[ernesto@ubuntu 23:15:01 practica3]$

```

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 PCLOCAL:

SCRIPT: `pmm-OpenMP_pclocal.sh`

```

#!/bin/bash

echo "secuencial"
bin/pmm-secuencial 100
bin/pmm-secuencial 500
bin/pmm-secuencial 1000

```

```

bin/pmm-secuencial 1500

echo "paralelo 2 threads"
export OMP_NUM_THREADS=2

bin/pmm-OpenMP 100
bin/pmm-OpenMP 500
bin/pmm-OpenMP 1000
bin/pmm-OpenMP 1500

echo "paralelo 4 threads"
export OMP_NUM_THREADS=4

bin/pmm-OpenMP 100
bin/pmm-OpenMP 500
bin/pmm-OpenMP 1000
bin/pmm-OpenMP 1500

```

	secuencial	2 thread	4 thread
100	0.002167379	0.001265039	0.002555077
500	0.646629926	0.724434041	1.732943774
1000	17.142421625	17.342668457	19.168253238
1500	84.503619663	104.521513044	80.797145658

El PC donde realizo las practicas es un AMD antiguo con dos cores pero sin multithread, quizá por eso al ejecutar me da estos tiempos tan raros

