

Arquitectura de Computadores (AC)

Cuaderno de prácticas.

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

Estudiante (nombre y apellidos): Fco Javier Merchan Martin

Grupo de prácticas:B2

Fecha de entrega: 15-05-18

Fecha evaluación en clase:

2º curso / 2º cuatr.

Grado Ing. Inform.

Doble Grado Ing.
Inform. y Mat.

Ejercicios basados en los ejemplos del seminario práctico

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

CAPTURA CÓDIGO FUENTE: if-clauseModificado.c

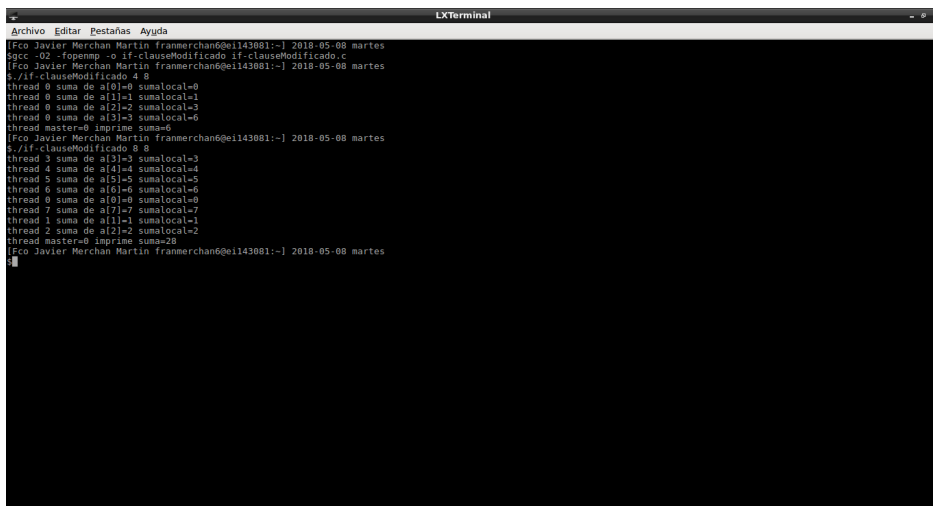
```
#include <stdio.h>
#include <stdlib.h>
#ifdef _OPENMP
#include <omp.h>
#else
#define omp_get_thread_num() 0
#define omp_get_num_threads() 1
#define omp_set_num_threads(int)
#endif

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



```
Archivo Editar Pestañas Ayuda
fco Javier Merchan Martin ffranmerchan@e1143881:~$ 2018-05-08 martes
gcc -O2 -fopenmp -o if-clauseModificado if-clauseModificado.c
fco Javier Merchan Martin ffranmerchan@e1143881:~$ 2018-05-08 martes
$ ./if-clauseModificado 8 8
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
fco Javier Merchan Martin ffranmerchan@e1143881:~$ 2018-05-08 martes
$ ./if-clauseModificado 8 8
thread 3 suma de a[3]=3 sumalocal=3
thread 4 suma de a[4]=4 sumalocal=4
thread 5 suma de a[5]=5 sumalocal=5
thread 6 suma de a[6]=6 sumalocal=6
thread 0 suma de a[0]=0 sumalocal=0
thread 7 suma de a[7]=7 sumalocal=7
thread 1 suma de a[1]=1 sumalocal=1
thread 2 suma de a[2]=2 sumalocal=2
thread master=0 imprime suma=28
fco Javier Merchan Martin ffranmerchan@e1143881:~$ 2018-05-08 martes
```

RESPUESTA: En las cláusulas tenemos puesto para que se repartan las hebras a partir de 4 por lo que para que repartan las iteraciones tienen que ser 5 o más.

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

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

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

RESPUESTA:

Con `static` las iteraciones se reparten entre las hebras en tiempo de compilación. Con `dynamic` en tiempo de ejecución y no se sabe cuántas iteraciones van a ejecutar cada hebra. Con `guided` es igual que `dynamic` pero el reparto es más uniforme.

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

CAPTURA CÓDIGO FUENTE: scheduled-clauseModificado.c

```

#include <stdio.h>
#include <stdlib.h>
#ifdef _OPENMP
#include <omp.h>
#else
#define omp_get_thread_num() 0
#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, chunk_value;
    omp_sched_t schedule_type;

    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("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:

```

[fcj@javier Merchan Martin franmerchan@eii143081:~] 2018-05-08 martes
$gcc -O2 -fopenmp -o scheduled-clauseModificado scheduled-clauseModificado.c
[fcj@javier Merchan Martin franmerchan@eii143081:~] 2018-05-08 martes
$./scheduled-clauseModificado 5 5
thread 0 suma a[0]=0 suma=0
Dentro de 'parallel for':
static = 1, dynamic = 2, guided = 3, auto = 4
dyn-var: 0, nthreads-var: 4, thread-limit-var: 2147483647, run-sched-var: 2, chunk: 1
thread 0 suma a[1]=1 suma=1
Dentro de 'parallel for':
static = 1, dynamic = 2, guided = 3, auto = 4
dyn-var: 0, nthreads-var: 4, thread-limit-var: 2147483647, run-sched-var: 2, chunk: 1
thread 0 suma a[2]=2 suma=3
Dentro de 'parallel for':
static = 1, dynamic = 2, guided = 3, auto = 4
dyn-var: 0, nthreads-var: 4, thread-limit-var: 2147483647, run-sched-var: 2, chunk: 1
thread 0 suma a[3]=3 suma=6
Dentro de 'parallel for':
static = 1, dynamic = 2, guided = 3, auto = 4
dyn-var: 0, nthreads-var: 4, thread-limit-var: 2147483647, run-sched-var: 2, chunk: 1
thread 0 suma a[4]=4 suma=10
Dentro de 'parallel for':
static = 1, dynamic = 2, guided = 3, auto = 4
dyn-var: 0, nthreads-var: 4, thread-limit-var: 2147483647, run-sched-var: 2, chunk: 1
Fuera de 'parallel for' suma=10
[fcj@javier Merchan Martin franmerchan@eii143081:~] 2018-05-08 martes
$./scheduled-clauseModificado

```

```

LXTerminal
Archivo Editar Pestañas Ayuda
[IA[FCo Javier Merchan Martin franmerchan6@eil143081:~] 2018-05-08 martes
$ export OMP_NUM_THREADS=10
[FCo Javier Merchan Martin franmerchan6@eil143081:~] 2018-05-08 martes
$ ./scheduled-claseModificado 5 5
thread 2 suma a[0]=0 suma=0
thread 2 suma a[1]=1 suma=1
thread 2 suma a[2]=2 suma=3
thread 2 suma a[3]=3 suma=6
thread 2 suma a[4]=4 suma=10
Fuera de 'parallel for' suma=10
static = 1, dynamic = 2, guided = 3, auto = 4
dyn-var: 0, nthreads-var: 10, thread-limit-var:2147483647, run-sched-var:2, chunk:1
[FCo Javier Merchan Martin franmerchan6@eil143081:~] 2018-05-08 martes
$

```

```

LXTerminal
Archivo Editar Pestañas Ayuda
[FCo Javier Merchan Martin franmerchan6@eil143081:~] 2018-05-08 martes
$ export OMP_THREAD_LIMIT=2
[FCo Javier Merchan Martin franmerchan6@eil143081:~] 2018-05-08 martes
$ ./scheduled-claseModificado 5 5
thread 0 suma a[0]=0 suma=0
Dentro de 'parallel for':
static = 1, dynamic = 2, guided = 3, auto = 4
dyn-var: 0, nthreads-var: 10, thread-limit-var:2, run-sched-var:2, chunk:1
thread 0 suma a[1]=1 suma=1
Dentro de 'parallel for':
static = 1, dynamic = 2, guided = 3, auto = 4
dyn-var: 0, nthreads-var: 10, thread-limit-var:2, run-sched-var:2, chunk:1
thread 0 suma a[2]=2 suma=3
Dentro de 'parallel for':
static = 1, dynamic = 2, guided = 3, auto = 4
dyn-var: 0, nthreads-var: 10, thread-limit-var:2, run-sched-var:2, chunk:1
thread 0 suma a[3]=3 suma=6
Dentro de 'parallel for':
static = 1, dynamic = 2, guided = 3, auto = 4
dyn-var: 0, nthreads-var: 10, thread-limit-var:2, run-sched-var:2, chunk:1
thread 0 suma a[4]=4 suma=10
Dentro de 'parallel for':
static = 1, dynamic = 2, guided = 3, auto = 4
dyn-var: 0, nthreads-var: 10, thread-limit-var:2, run-sched-var:2, chunk:1
Fuera de 'parallel for' suma=10
static = 1, dynamic = 2, guided = 3, auto = 4
dyn-var: 0, nthreads-var: 10, thread-limit-var:2, run-sched-var:2, chunk:1
[FCo Javier Merchan Martin franmerchan6@eil143081:~] 2018-05-08 martes
$

```

```

LXTerminal
Archivo Editar Pestañas Ayuda
[FCo Javier Merchan Martin franmerchan6@eil143081:~] 2018-05-08 martes
$ export OMP_DYNAMIC=TRUE
[FCo Javier Merchan Martin franmerchan6@eil143081:~] 2018-05-08 martes
$ ./scheduled-claseModificado 5 5
thread 0 suma a[0]=0 suma=0
Dentro de 'parallel for':
static = 1, dynamic = 2, guided = 3, auto = 4
dyn-var: 1, nthreads-var: 10, thread-limit-var:2, run-sched-var:2, chunk:1
thread 0 suma a[1]=1 suma=1
Dentro de 'parallel for':
static = 1, dynamic = 2, guided = 3, auto = 4
dyn-var: 1, nthreads-var: 10, thread-limit-var:2, run-sched-var:2, chunk:1
thread 0 suma a[2]=2 suma=3
Dentro de 'parallel for':
static = 1, dynamic = 2, guided = 3, auto = 4
dyn-var: 1, nthreads-var: 10, thread-limit-var:2, run-sched-var:2, chunk:1
thread 0 suma a[3]=3 suma=6
Dentro de 'parallel for':
static = 1, dynamic = 2, guided = 3, auto = 4
dyn-var: 1, nthreads-var: 10, thread-limit-var:2, run-sched-var:2, chunk:1
thread 0 suma a[4]=4 suma=10
Dentro de 'parallel for':
static = 1, dynamic = 2, guided = 3, auto = 4
dyn-var: 1, nthreads-var: 10, thread-limit-var:2, run-sched-var:2, chunk:1
Fuera de 'parallel for' suma=10
static = 1, dynamic = 2, guided = 3, auto = 4
dyn-var: 1, nthreads-var: 10, thread-limit-var:2, run-sched-var:2, chunk:1
[FCo Javier Merchan Martin franmerchan6@eil143081:~] 2018-05-08 martes
$

```

RESPUESTA: Se obtienen los mismos resultados fuera y dentro 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.

CAPTURA 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, chunk_value;
    omp_sched_t schedule_type;

    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:

```

Archivo  Editar  Pestañas  Ayuda
[FCo Javier Merchan Martin franmerchan@ei143081:~] 2018-05-08 martes
$gcc -O2 -fopenmp -o scheduled-claseModificado scheduled-claseModificado4.c
[FCo Javier Merchan Martin franmerchan@ei143081:~] 2018-05-08 martes
$./scheduled-claseModificado 5 6
thread 0 suma a[0]=0 suma=0
Dentro de 'parallel for':
  static = 1, dynamic = 2, guided = 3, auto = 4
  dyn-var: 1, nthreads-var: 10, thread-limit-var:2, run-sched-var:2, chunk:1
  get_num_threads: 2, get_num_procs: 4, in_parallel(): 1
  thread 0 suma a[1]=1 suma=1
Dentro de 'parallel for':
  static = 1, dynamic = 2, guided = 3, auto = 4
  dyn-var: 1, nthreads-var: 10, thread-limit-var:2, run-sched-var:2, chunk:1
  get_num_threads: 2, get_num_procs: 4, in_parallel(): 1
  thread 0 suma a[2]=2 suma=3
Dentro de 'parallel for':
  static = 1, dynamic = 2, guided = 3, auto = 4
  dyn-var: 1, nthreads-var: 10, thread-limit-var:2, run-sched-var:2, chunk:1
  get_num_threads: 2, get_num_procs: 4, in_parallel(): 1
  thread 0 suma a[3]=3 suma=6
Dentro de 'parallel for':
  static = 1, dynamic = 2, guided = 3, auto = 4
  dyn-var: 1, nthreads-var: 10, thread-limit-var:2, run-sched-var:2, chunk:1
  get_num_threads: 2, get_num_procs: 4, in_parallel(): 1
  thread 0 suma a[4]=4 suma=10
Dentro de 'parallel for':
  static = 1, dynamic = 2, guided = 3, auto = 4
  dyn-var: 1, nthreads-var: 10, thread-limit-var:2, run-sched-var:2, chunk:1
  get_num_threads: 2, get_num_procs: 4, in_parallel(): 1
  thread 0 suma a[5]=5 suma=15
Fuera de 'parallel for' suma=15
static = 1, dynamic = 2, guided = 3, auto = 4
dyn-var: 1, nthreads-var: 10, thread-limit-var:2, run-sched-var:2, chunk:1
get_num_threads: 1, get_num_procs: 4, in_parallel(): 0
[FCo Javier Merchan Martin franmerchan@ei143081:~] 2018-05-08 martes
$

```

RESPUESTA: La única que se mantiene dentro y fuera es `omp_get_num_proc()`, las otras dos varían.

5. Añadir al programa `scheduled-claseModificado.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-claseModificado5.c`

```

scheduled-claseModificado5.c (PEN AZUL /media/fran/PEN AZUL/AC/Practica 3) - gedit
Abrir  Guardar
#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, chunk_value;
    omp_sched_t schedule_type;

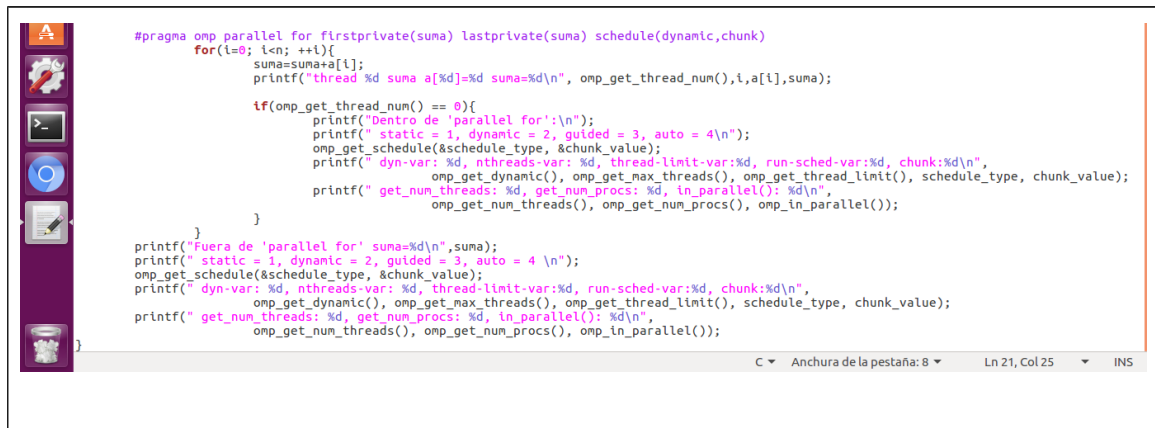
    if(argc < 3){
        fprintf(stderr, "\nFalta iteraciones o chunk\n");
        exit(-1);
    }
    n=atoi(argv[1]);
    if(n>200)
        n=200;
    chunk=atoi(argv[2]);

    for(i=0; i<n; ++i)
        a[i]=i;
    printf("Antes de la modificación\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());

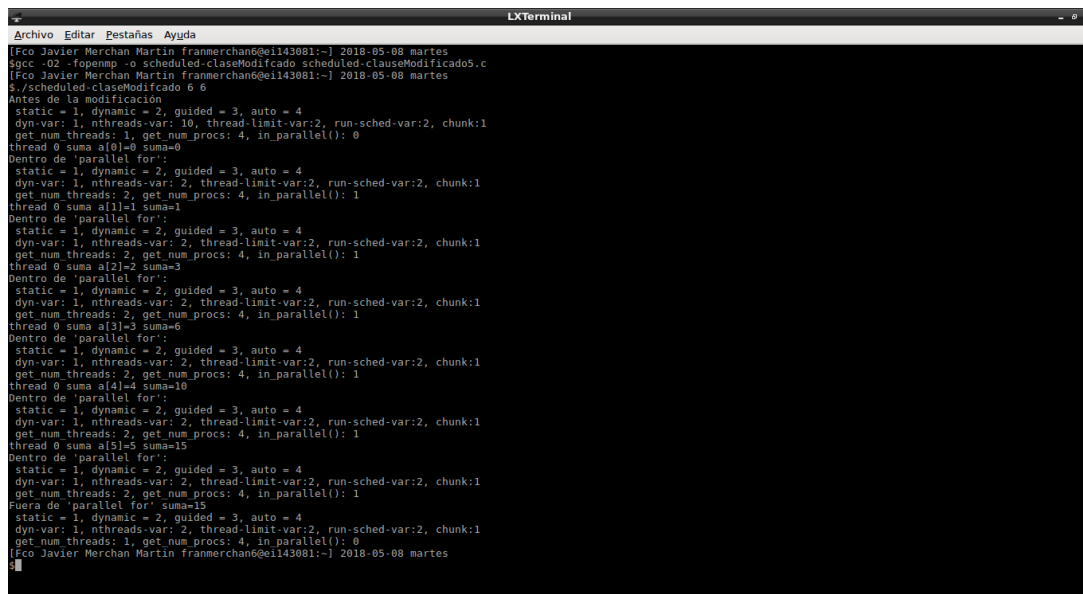
    omp_set_dynamic(2);
    omp_set_num_threads(2);
    omp_set_schedule(2,1);

#pragma omp parallel for firstprivate(suma) lastprivate(suma) schedule(dynamic,chunk)

```



CAPTURAS DE PANTALLA:



RESPUESTA:

Resto de ejercicios

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

NOTAS: (1) el número de filas/columnas debe ser un argumento de entrada; (2) se debe inicializar las matrices antes del cálculo; (3) se debe imprimir siempre la primera y última componente del resultado antes de que termine el programa.

CAPTURA CÓDIGO FUENTE: pmtv-secuencial.c

The screenshot displays a Linux desktop environment. At the top, a panel shows the system clock as 00:45 and various status icons. Below this, a dark-themed taskbar contains several application icons. The main workspace features a window titled "pmtv-secuencia1.c (PEN AZUL /media/fran/PEN AZUL/AC/Practica 3) - gedit". Inside the editor, a C program is written, which initializes a triangular matrix and a vector, then computes their dot product by summing the products of corresponding elements. Comments in Spanish explain the steps, such as initializing the matrix size and freeing memory. A vertical orange bar is visible on the right side of the editor window. At the bottom, a terminal window is open, showing a shell prompt. The system tray at the very bottom indicates the current page is 38 of 30, with a width of 8 columns.

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

int main(int argc, char **argv){
    int i, j;

    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);
}
```

CAPTURAS DE PANTALLA:

```

* LXTerminal
Archivo Editar Pestañas Ayuda
[fc@ Javier Merchan Martin franmerchan@el143081:~] 2018-05-08 martes
gcc -O2 -fopenmp -o pmtv-secuencial pmtv-secuencial.c
[fc@ Javier Merchan Martin franmerchan@el143081:~] 2018-05-08 martes
$ ./pmtv-secuencial
Matrix size
[fc@ Javier Merchan Martin franmerchan@el143081:~] 2018-05-08 martes
$ ./pmtv-secuencial 30
Matrix:
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 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 2 2 2 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 2 2 2 2 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 2 2 2 2 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 2 2 2 2 2 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 2 2 2 2 2 2 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 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
0 0 0 0 0 0 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
0 0 0 0 0 0 0 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
0 0 0 0 0 0 0 0 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
0 0 0 0 0 0 0 0 0 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
0 0 0 0 0 0 0 0 0 0 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
0 0 0 0 0 0 0 0 0 0 0 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
0 0 0 0 0 0 0 0 0 0 0 0 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 2 2 2 2 2 2 2 2 2 2 2 2
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 2 2 2 2 2 2 2 2 2 2 2
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 2 2 2 2 2 2 2 2 2 2
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 2 2 2 2 2 2 2 2 2
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 2 2 2 2 2 2 2 2
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 2 2 2 2 2 2 2
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 2 2 2 2 2 2
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 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 0 0 0 0 0 0 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 0 0 0 0 0 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 0 0 0 0 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 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 2
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2
Vector:
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
Resultado:
240 232 224 216 208 200 192 184 176 168 160 152 144 136 128 120 112 104 96 88 80 72 64 56 48 40 32 24 16 8
[fc@ Javier Merchan Martin franmerchan@el143081:~] 2018-05-08 martes
$

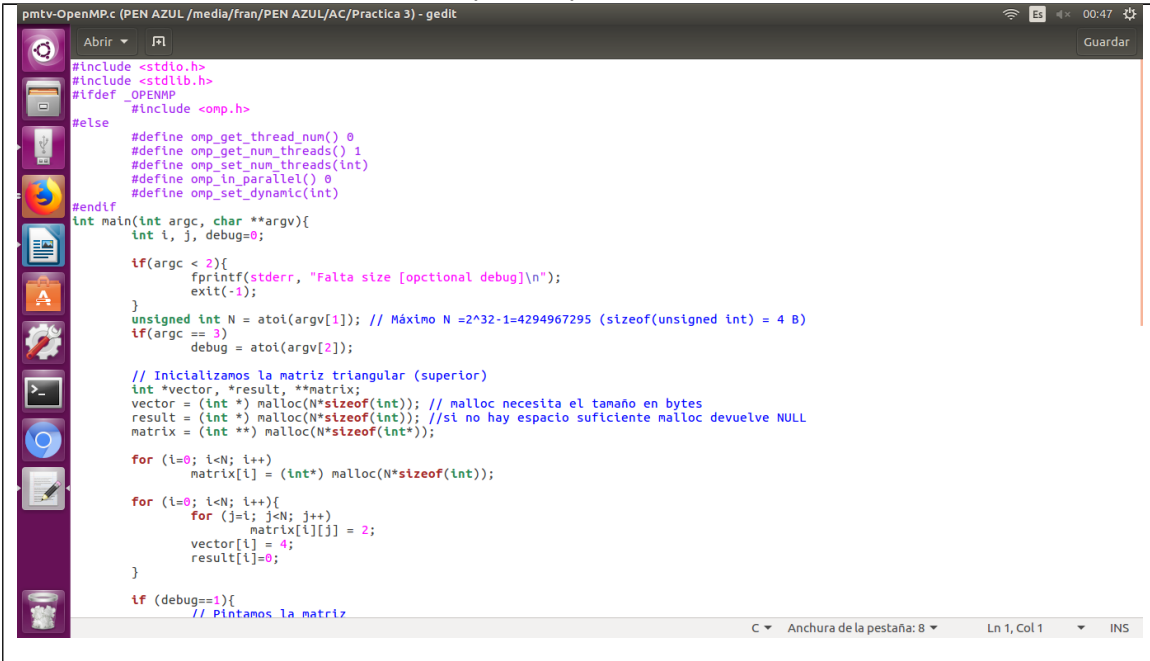
```

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:

CAPTURA 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;

    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
    }
}
```

```

pmtv-OpenMP.c (PEN AZUL /media/fran/PEN AZUL/AC/Practica 3) - gedit
Abrir  Guardar

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]);
}

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);

```

DESCOMPOSICIÓN DE DOMINIO:

CAPTURAS DE PANTALLA:

TABLA RESULTADOS, SCRIPT Y GRÁFICA atcgrid

SCRIPT: pmtv-OpenMP_PCaula.sh

```

#!/bin/bash

#PBS -N pmtv-OpenMP
#PBS -q ac
echo "Id SPBS_O_WORKDIR usuario del trabajo: $PBS_O_LOGNAME"
echo "Id SPBS_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= , 12 threads			
Chunk	Static	Dynamic	Guided
por defecto			
1			
64			
Chunk	Static	Dynamic	Guided
por defecto			
1			
64			

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.

CAPTURA CÓDIGO FUENTE: pmm-secuencial.c

```

pmm-secuencial.c (PEN AZUL /media/fran/PEN AZUL/AC/Practica 3) - gedit
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

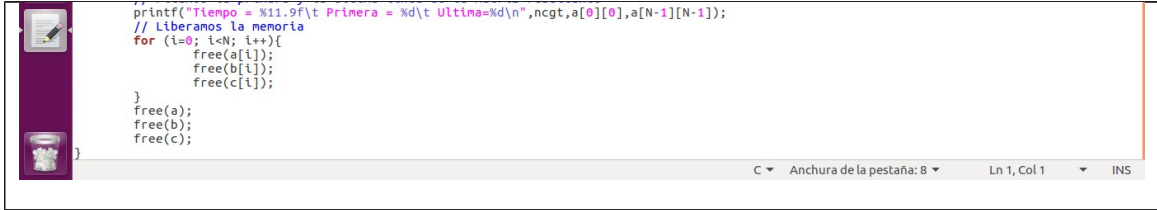
int main(int argc, char **argv){
    int i, j, k;

    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;
        }
    }

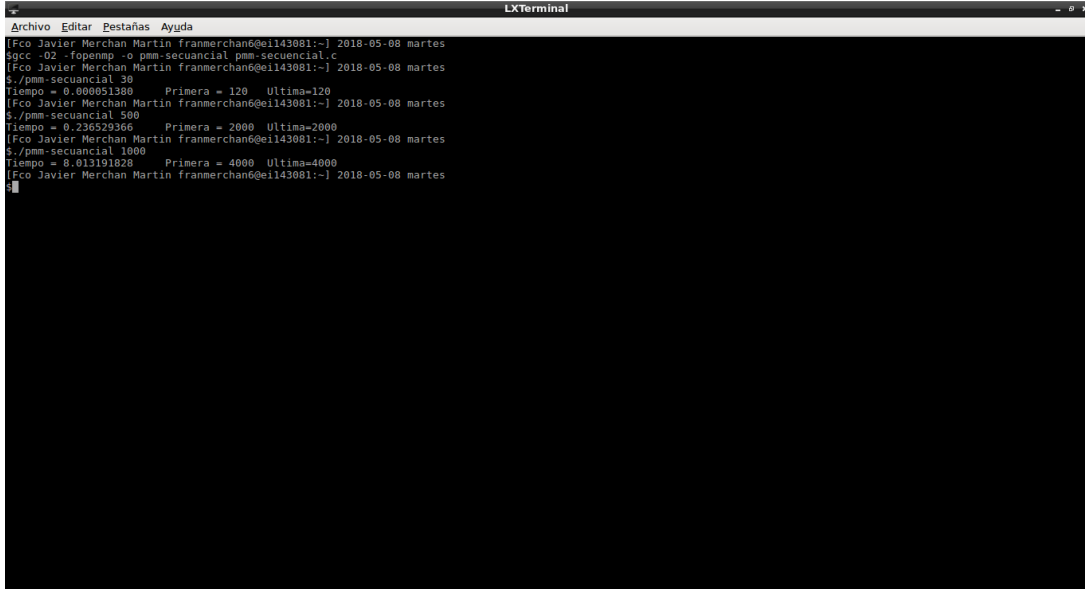
    struct timespec cgt1, cgt2; double ncgt;
    clock_gettime(CLOCK_REALTIME, &cgt1);
    // Multiplicacion
    for (i=0; i<N; i++){
        for (j=0; j<N; j++){
            for (k=0; k<N; k++){
                a[i][j] += b[i][k] * c[k][j];
            }
        }
    }
    clock_gettime(CLOCK_REALTIME, &cgt2);
    ncgt = (double) (cgt2.tv_sec - cgt1.tv_sec) + ((double) (cgt2.tv_nsec - cgt1.tv_nsec) / (1.e+9));
    // Printamos 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);
```

CAPTURAS DE PANTALLA:



```

[FCo Javier Merchan Martin franmerchan6@eil43081:~] 2018-05-08 martes
$gcc -O2 -fopenmp -o pmm-secuencial pmm-secuencial.c
[FCo Javier Merchan Martin franmerchan6@eil43081:~] 2018-05-08 martes
$./pmm-secuencial 30
Tiempo = 0.000051380    Primera = 120    Ultima=120
[FCo Javier Merchan Martin franmerchan6@eil43081:~] 2018-05-08 martes
$./pmm-secuencial 500
Tiempo = 0.236529266    Primera = 2000    Ultima=2000
[FCo Javier Merchan Martin franmerchan6@eil43081:~] 2018-05-08 martes
$./pmm-secuencial 1000
Tiempo = 8.013191828    Primera = 4000    Ultima=4000
[FCo Javier Merchan Martin franmerchan6@eil43081:~] 2018-05-08 martes

```

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).

10. **DESCOMPOSICIÓN DE DOMINIO:**

CAPTURA CÓDIGO FUENTE: pmm-OpenMP.c

```

pmm-OpenMP.c (PEN AZUL /media/fran/PEN AZUL/AC/Practica 3) - gedit
Abrir Guardar

#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;

    // Printamos 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);
}

```

C Anchura de la pestaña: 8 Ln 1, Col 1 INS

CAPTURAS DE PANTALLA:

```

LXTerminal
Archivo Editar Pestañas Ayuda

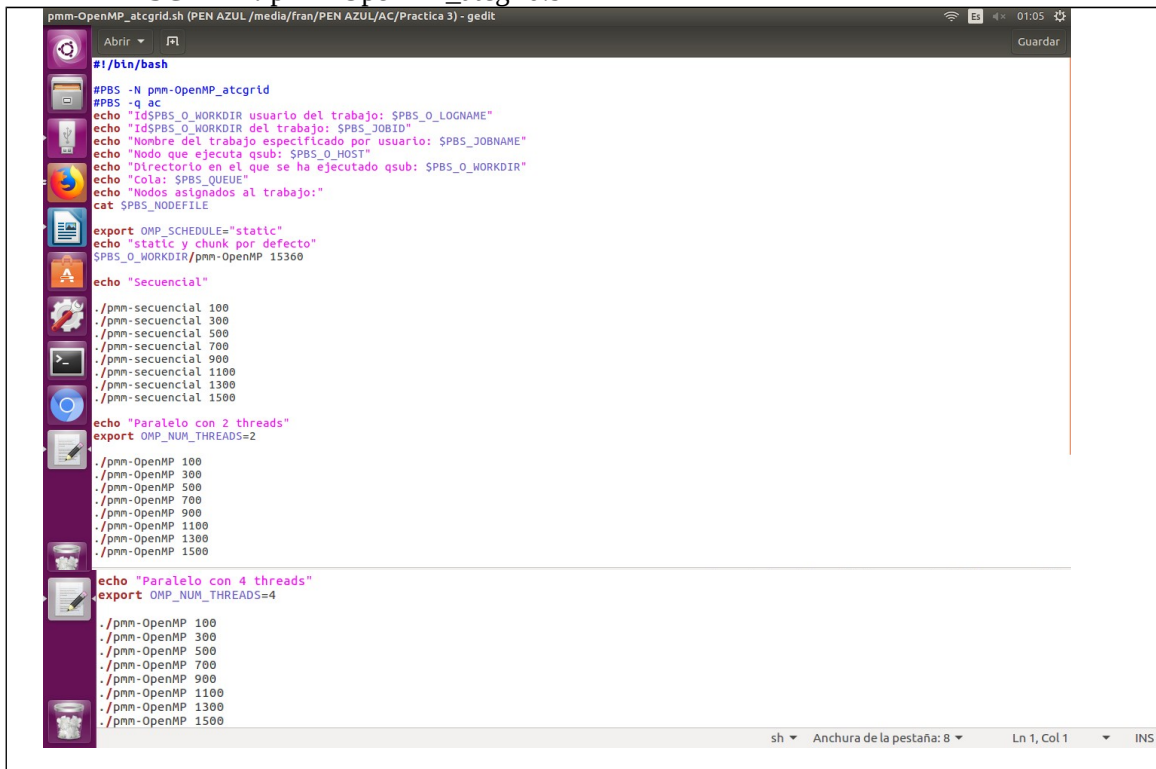
[FCo Javier Merchan Martin franmerchan6@eil43081:~] 2018-05-08 martes
$gcc -O2 -fopenmp -o pmm-OpenMP pmm-OpenMP.c
[FCo Javier Merchan Martin franmerchan6@eil43081:~] 2018-05-08 martes
$./pmm-OpenMP 30
Tiempo = 0.000056188      Primera = 120      Ultima=120
[FCo Javier Merchan Martin franmerchan6@eil43081:~] 2018-05-08 martes
$./pmm-OpenMP 3
Tiempo = 0.000026242      Primera = 12      Ultima=12
[FCo Javier Merchan Martin franmerchan6@eil43081:~] 2018-05-08 martes
$./pmm-OpenMP 50
Tiempo = 0.000167975      Primera = 200      Ultima=200
[FCo Javier Merchan Martin franmerchan6@eil43081:~] 2018-05-08 martes
$./pmm-OpenMP 500
Tiempo = 0.157503099      Primera = 2000      Ultima=2000
[FCo Javier Merchan Martin franmerchan6@eil43081:~] 2018-05-08 martes
$./pmm-OpenMP 1000
Tiempo = 3.839797274      Primera = 4000      Ultima=4000
[FCo Javier Merchan Martin franmerchan6@eil43081:~] 2018-05-08 martes
$

```

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



```
pmm-OpenMP_atcgrid.sh (PEN AZUL /media/fran/PEN AZUL/AC/Practica 3) - gedit
Abrir Guardar
#!/bin/bash

#PBS -N pmm-OpenMP_atcgrid
#PBS -q ac
echo "IdSPBS_O_WORKDIR usuario del trabajo: SPBS_O_LOGNAME"
echo "IdSPBS_O_WORKDIR del trabajo: SPBS_JOBID"
echo "Nombre del trabajo especificado por usuario: SPBS_JOBNAME"
echo "Nodo que ejecuta qsub: SPBS_O_HOST"
echo "Directorio en el que se ha ejecutado qsub: SPBS_O_WORKDIR"
echo "Cola: SPBS_QUEUE"
echo "Nodos asignados al trabajo:"
cat SPBS_NODEFILE

export OMP_SCHEDULE="static"
echo "static y chunk por defecto"
SPBS_O_WORKDIR/pmm-OpenMP 15360

echo "Secuencial"
./pmm-secuencial 100
./pmm-secuencial 300
./pmm-secuencial 500
./pmm-secuencial 700
./pmm-secuencial 900
./pmm-secuencial 1100
./pmm-secuencial 1300
./pmm-secuencial 1500

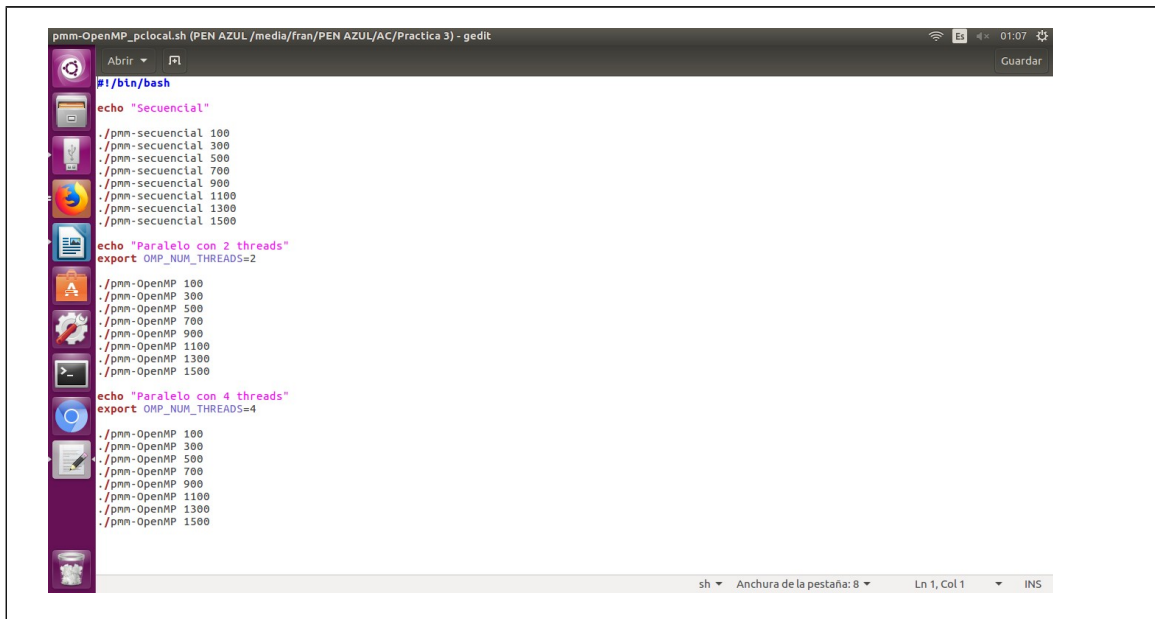
echo "Paralelo con 2 threads"
export OMP_NUM_THREADS=2
./pmm-OpenMP 100
./pmm-OpenMP 300
./pmm-OpenMP 500
./pmm-OpenMP 700
./pmm-OpenMP 900
./pmm-OpenMP 1100
./pmm-OpenMP 1300
./pmm-OpenMP 1500

echo "Paralelo con 4 threads"
export OMP_NUM_THREADS=4
./pmm-OpenMP 100
./pmm-OpenMP 300
./pmm-OpenMP 500
./pmm-OpenMP 700
./pmm-OpenMP 900
./pmm-OpenMP 1100
./pmm-OpenMP 1300
./pmm-OpenMP 1500

sh Anchura de la pestaña: 8 Ln 1, Col 1 INS
```

ESTUDIO DE ESCALABILIDAD EN PCLOCAL:

SCRIPT: pmm-OpenMP_pcllocal.sh



```
pmm-OpenMP_pclocal.sh (PEN AZUL /media/fran/PEN AZUL/AC/Practica 3) - gedit
Abrir Guardar
#1/bin/bash
echo "Secuencial"
./pmm-secuencial 100
./pmm-secuencial 300
./pmm-secuencial 500
./pmm-secuencial 700
./pmm-secuencial 900
./pmm-secuencial 1100
./pmm-secuencial 1300
./pmm-secuencial 1500
echo "Paralelo con 2 threads"
export OMP_NUM_THREADS=2
./pmm-OpenMP 100
./pmm-OpenMP 300
./pmm-OpenMP 500
./pmm-OpenMP 700
./pmm-OpenMP 900
./pmm-OpenMP 1100
./pmm-OpenMP 1300
./pmm-OpenMP 1500
echo "Paralelo con 4 threads"
export OMP_NUM_THREADS=4
./pmm-OpenMP 100
./pmm-OpenMP 300
./pmm-OpenMP 500
./pmm-OpenMP 700
./pmm-OpenMP 900
./pmm-OpenMP 1100
./pmm-OpenMP 1300
./pmm-OpenMP 1500
sh Anchura de la pestaña: 8 Ln 1, Col 1 INS
```