

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): Antonio Jesús Heredia Castillo

Grupo de prácticas: A2

Fecha de entrega: 15-05-2018

Fecha evaluación en clase: 15-05-2018

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>
#include <omp.h>
int main(int argc, char **argv)
{
    int i, n = 20, tid;
    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:

```
[AntonioJesusHerediaCastillo antonio@Linux-Antonio:~/git/AC/Practica3/codigos] 2018-05-13 domingo
$ ./if-clauseModificado.out 3 2
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
[AntonioJesusHerediaCastillo antonio@Linux-Antonio:~/git/AC/Practica3/codigos] 2018-05-13 domingo
$ ./if-clauseModificado.out 5 2
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 1 suma de a[3]=3 sumalocal=3
thread 1 suma de a[4]=4 sumalocal=7
thread master=0 imprime suma=10
[AntonioJesusHerediaCastillo antonio@Linux-Antonio:~/git/AC/Practica3/codigos] 2018-05-13 domingo
$ ./if-clauseModificado.out 5 5
thread 2 suma de a[2]=2 sumalocal=2
thread 0 suma de a[0]=0 sumalocal=0
thread 1 suma de a[1]=1 sumalocal=1
thread 4 suma de a[4]=4 sumalocal=4
thread 3 suma de a[3]=3 sumalocal=3
thread master=0 imprime suma=10
```

RESPUESTA:

Lo que esta pasando es que cuando tiene n es menor que 4, no se ejecuta en paralelo.

Cuando n es mayor que 4 se paraleliza y esta paralelizacion depende de la variable x , ya que el numero de thread lo cambiamos en tiempo de ejecución con la clausula `num_threads(X)`

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 `scheduleleg-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	1	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0
2	0	1	0	0	1	0	0	0	0
3	1	1	0	0	1	0	0	0	0
4	0	0	1	0	0	1	0	0	0
5	1	0	1	0	0	1	0	0	0
6	0	1	1	0	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	1	0	0	0	0
13	1	0	1	0	1	0	0	0	0
14	0	1	1	0	1	0	0	0	0
15	1	1	1	0	1	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-clause.d.c			schedule-clause.g.c		
	1	2	4	1	2	4	1	2	4
0	0	0	0	0	3	2	0	1	2
1	1	0	0	2	3	2	0	1	2
2	2	1	0	1	0	2	0	1	2
3	3	1	0	3	0	2	0	1	2
4	0	2	1	0	1	0	1	3	0
5	1	2	1	0	1	0	1	3	0
6	2	3	1	0	2	0	1	3	0
7	3	3	1	0	2	0	3	0	0
8	0	0	2	0	0	1	3	0	1
9	1	0	2	0	0	1	3	0	1
10	2	1	2	0	0	1	2	2	1
11	3	1	2	0	0	1	2	2	1
12	0	2	3	0	0	3	0	3	3
13	1	2	3	0	0	3	0	3	3
14	2	3	3	0	0	3	0	3	3
15	3	3	3	0	0	3	0	3	3

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

RESPUESTA:

Con `static` el reparto se hará de forma que a cada hebra al principio le toque, n iteraciones (siendo n el chunk) y cuando ya se haya repartido a todas las hebras, se volverá a repartir n iteraciones a la primera y n a la segunda, etc.

Con `dynamic`, las iteraciones se dividirán en grupo de n elementos. Y la hebra que este disponible (es decir la más rápida) ejecutará ese paquete de n iteraciones. Así hasta que se acaben. Esto se realiza en tiempo de ejecución siempre.

`Guided` es una especialización del reparto dinámico. En este reparto se empieza con tamaños de chunk grandes y se va decrementando para balancear la carga.

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
#endif
```

```

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

            omp_get_schedule(&tipo_schedule, &valor_chunk);
            printf(" thread-limit-var:%d, nthreads-var:%d, dyn-var: %d, run-sched-var: %d, chunk: %d\n",
                omp_get_thread_limit(), omp_get_max_threads(), omp_get_dynamic(), tipo_schedule,
                valor_chunk);
        }
    }
    printf("Fuera de 'parallel for' suma=%d\n", suma);
    omp_get_schedule(&tipo_schedule, &valor_chunk);
    printf(" thread-limit-var:%d, nthreads-var:%d, dyn-var: %d, run-sched-var: %d, chunk: %d\n",
        omp_get_thread_limit(), omp_get_max_threads(), omp_get_dynamic(), tipo_schedule,
        valor_chunk);
    return 0;
}

```

CAPTURAS DE PANTALLA:

```

[AntonioJesúsHerediaCastillo antonio@Linux-Antonio:~/git/AC/Practica3/codigos] 2018-05-14 lunes
$ ./scheduled-clauseModificado.out 3 2
thread 3 suma a[0]=0 suma=0
thread 3 suma a[1]=1 suma=1
thread 0 suma a[2]=2 suma=2
Dentro de 'parallel for':
thread-limit-var:2147483647, nthreads-var:4, dyn-var: 0, run-sched-var: 2, chunk: 1
Fuera de 'parallel for' suma=2
thread-limit-var:2147483647, nthreads-var:4, dyn-var: 0, run-sched-var: 2, chunk: 1
[AntonioJesúsHerediaCastillo antonio@Linux-Antonio:~/git/AC/Practica3/codigos] 2018-05-14 lunes

```

```
[AntonioJesusHerediaCastillo antonio@Linux-Antonio:~/git/AC/Practica3/codigos] 2018-05-14 lunes
$export OMP_NUM_THREADS=7
[AntonioJesusHerediaCastillo antonio@Linux-Antonio:~/git/AC/Practica3/codigos] 2018-05-14 lunes
$./scheduled-clauseModificado.out 3 2
thread 3 suma a[0]=0 suma=0
thread 3 suma a[1]=1 suma=1
thread 6 suma a[2]=2 suma=2
Fuera de 'parallel for' suma=2
thread-limit-var:2147483647, nthreads-var:7, dyn-var: 0, run-sched-var: 2, chunk: 1
[AntonioJesusHerediaCastillo antonio@Linux-Antonio:~/git/AC/Practica3/codigos] 2018-05-14 lunes
$export OMP_SCHEDULE="guided,3"
[AntonioJesusHerediaCastillo antonio@Linux-Antonio:~/git/AC/Practica3/codigos] 2018-05-14 lunes
$./scheduled-clauseModificado.out 3 2
thread 5 suma a[0]=0 suma=0
thread 4 suma a[2]=2 suma=2
thread 5 suma a[1]=1 suma=1
Fuera de 'parallel for' suma=2
thread-limit-var:2147483647, nthreads-var:7, dyn-var: 0, run-sched-var: 3, chunk: 3
```

RESPUESTA:

Los valores dentro y fuera del parallel son iguales.

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

        omp_get_schedule(&tipo_schedule, &valor_chunk);
        printf(" thread-limit-var:%d, nthreads-var:%d, dyn-var: %d, run-sched-var: %d, chunk: %d\n",
            omp_get_thread_limit(), omp_get_max_threads(), omp_get_dynamic(), tipo_schedule, valor_chunk);
        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);
omp_get_schedule(&tipo_schedule, &valor_chunk);
printf(" thread-limit-var:%d, nthreads-var:%d, dyn-var: %d, run-sched-var: %d, chunk: %d\n",
    omp_get_thread_limit(), omp_get_max_threads(), omp_get_dynamic(), tipo_schedule, valor_chunk);
printf(" get_num_threads: %d, get_num_procs: %d, in_parallel():%d\n", \
    omp_get_num_threads(), omp_get_num_procs(), omp_in_parallel());
return 0;
}

```

CAPTURAS DE PANTALLA:

```

[AntonioJesusHerediaCastillo antonio@Linux-Antonio:~/git/AC/Practica3/codigos] 2018-05-14 lunes
$ ./scheduled-clauseModificado4.out 5 1
thread 2 suma a[0]=0 suma=0
thread 3 suma a[2]=2 suma=2
thread 4 suma a[1]=1 suma=1
thread 5 suma a[3]=3 suma=3
thread 0 suma a[4]=4 suma=4
Dentro de 'parallel for':
thread-limit-var:2147483647, nthreads-var:8, dyn-var: 0, run-sched-var: 1, chunk: 3
get_num_threads: 8, get_num_procs: 4, in_parallel():1
Fuera de 'parallel for' suma=4
thread-limit-var:2147483647, nthreads-var:8, dyn-var: 0, run-sched-var: 1, chunk: 3
get_num_threads: 1, get_num_procs: 4, in_parallel():0

```

RESPUESTA:

Se obtiene valores distintos en de hebras y en cuantas hay en paralelo.

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

```

#include <stdio.h>

#include <stdlib.h>

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

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

```

```

int valor_chunk;
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 antes modificarlo'\n");
        omp_get_schedule(&tipo_schedule, &valor_chunk);
        printf(" thread-limit-var:%d nthreads-var:%d dyn-var:%d run-sched-var: %d chunk: %d\n",
            omp_get_thread_limit(), omp_get_max_threads(), omp_get_dynamic(), tipo_schedule, valor_chunk);
    }
}
printf("Fuera de 'parallel for' suma=%d\n", suma);
// Cambiamos los valores
omp_set_dynamic(3);
omp_set_num_threads(3);
omp_set_schedule(3, 3);
printf("Despues de modificarlo\n");
omp_get_schedule(&tipo_schedule, &valor_chunk);
printf(" thread-limit-var:%d nthreads-var %d dyn-var: %d run-sched-var: %d chunk: %d\n",
    omp_get_thread_limit(), omp_get_max_threads(), omp_get_dynamic(), tipo_schedule, valor_chunk);
return 0;
}

```


CAPTURAS DE PANTALLA:

```
[AntonioJesusHerediaCastillo antonio@Linux-Antonio:~/git/AC/Practica3/codigos] 2018-05-14 lunes
$./scheduled-clauseModificado5.out 5 1
thread 3 suma a[0]=0 suma=0
thread 2 suma a[1]=1 suma=1
thread 5 suma a[2]=2 suma=2
thread 0 suma a[4]=4 suma=4
Dentro de 'parallel for antes modificarlo':
thread-limit-var:2147483647, nthreads-var:8, dyn-var: 0, run-sched-var: 1, chunk: 3
thread 4 suma a[3]=3 suma=3
Fuera de 'parallel for' suma=4
Despues de modificarlo
thread-limit-var:2147483647, nthreads-var:3, dyn-var: 1, run-sched-var: 3, chunk: 3
```

RESPUESTA:

El valor de las variables cambia de forma correcta.

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

```
#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, "Numero de parametros incorrectos\n");
        exit(-1);
    }
    int N = atoi(argv[1]);
    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] = 1;
        vector[i] = 3;
        result[i] = 0;
    }
}
```

```

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

    // 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");
    }

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

    printf("Primera posicion: %d", result[0]);
    printf("\n");
    printf("Ultima posicion: %d", result[N-1]);
    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:

```

[AntonioJesusHerediaCastillo antonio@Linux-Antonio:~/git/AC/Practica3/codigos] 2018-05-14 lunes
$ ./pmtv-secuencial.out 12
Vector:
1 1 1 1 1 1 1 1 1 1 1 1
Matriz:
1 1 1 1 1 1 1 1 1 1 1 1
0 1 1 1 1 1 1 1 1 1 1 1
0 0 1 1 1 1 1 1 1 1 1 1
0 0 0 1 1 1 1 1 1 1 1 1
0 0 0 0 1 1 1 1 1 1 1 1
0 0 0 0 0 1 1 1 1 1 1 1
0 0 0 0 0 0 1 1 1 1 1 1
0 0 0 0 0 0 0 1 1 1 1 1
0 0 0 0 0 0 0 0 1 1 1 1
0 0 0 0 0 0 0 0 0 1 1 1
0 0 0 0 0 0 0 0 0 0 1 1
Primera posicion: 36
Ultima posicion: 3

```

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;
    //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:

CAPTURAS DE PANTALLA:

```

AntonioJesusHerediaCastillo antonio@Linux-Antonio:~/git/AC/Practica3/codigos] 2018-05-14 lunes
$ ./pmtv-OpenMP.out 121
Tiempo = 0.000472147    Primera = 968    Ultima=8
AntonioJesusHerediaCastillo antonio@Linux-Antonio:~/git/AC/Practica3/codigos] 2018-05-15 martes
$

```

TABLA RESULTADOS, SCRIPT Y GRÁFICA atcgrid

SCRIPT: pmtv-OpenMP_PCaula.sh

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:

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

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

resultado antes de que termine el programa.

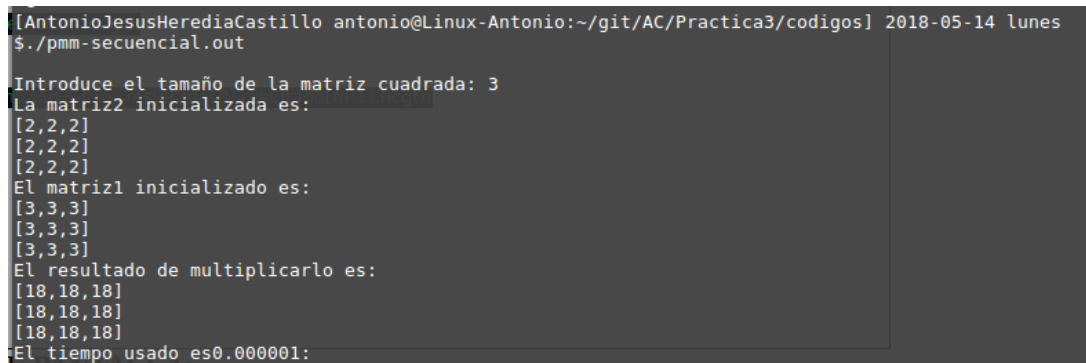
CAPTURA CÓDIGO FUENTE: pmm-secuencial.c

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h> // biblioteca donde se encuentra la función clock_gettime()
#define PRUEBAS
int main(int argc, char **argv) {
    int fil;
    printf("\nIntroduce el tamaño de la matriz cuadrada: ");
    scanf("%d", &fil);
    //Reservamos memoria
    int **matriz1 = (int **)malloc(fil*sizeof(int*));
    int **resultado = (int **)malloc(fil*sizeof(int*));
    int **matriz2 = (int **)malloc(fil*sizeof(int*));
    for(int i = 0; i < fil; i++){
        matriz2[i] = (int*)malloc(fil*sizeof(int));
        matriz1[i] = (int*)malloc(fil*sizeof(int));
        resultado[i] = (int*)malloc(fil*sizeof(int));
        if(matriz2[i] == NULL) perror("Error: ");
    }
    //Inicializamos la matriz2
    for(int f = 0; f < fil; f++){
        for(int c = 0; c < fil; c++){
            matriz2[f][c] = 2.0;
            matriz1[f][c] = 3.0;
            resultado[f][c] = 0;
        }
    }
    struct timespec cgt1,cgt2;
    double ncgt; //para tiempo de ejecución
    clock_gettime(CLOCK_REALTIME,&cgt1);
    for(int f = 0; f < fil; f++){
        for(int c = 0; c < fil; c++){
            for(int j = 0; j < fil; j++){
                resultado[f][c] += matriz1[f][j]*matriz2[j][c];
            }
        }
        clock_gettime(CLOCK_REALTIME,&cgt2);
        ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+ (double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));
    }
    //imprimimos los datos
    #ifdef PRUEBAS
    printf("La matriz2 inicializada es:\n");
    for(int f = 0; f < fil; f++){
        printf("[");
        for(int c = 0; c < fil-1; c++){
            printf(" %d,",matriz2[f][c]);
        }
        printf(" %d\n",matriz2[f][fil-1]);
    }
    printf("El matriz1 inicializado es:\n");
```

```

for(int f = 0; f < fil; f++){
printf("[");
for(int c = 0; c < fil-1; c++){
printf("%d,",matriz1[f][c]);
printf("%d\\n",matriz1[fil-1][fil-1]);
}
printf("El resultado de multiplicarlo es:\\n");
for(int f = 0; f < fil; f++){
printf("[");
for(int c = 0; c < fil-1; c++){
printf("%d,",resultado[f][c]);
printf("%d\\n",resultado[fil-1][fil-1]);
}
printf("El tiempo usado es%f\\n",ncgt);
#else
printf("Primero:%d Ultimo:%d %f\\n",resultado[0],resultado[fil-1],ncgt);
#endif
free(matriz2);
free(matriz1);
free(resultado);
return 0;
}

```

CAPTURAS DE PANTALLA:


```

[AntonioJesusHerediaCastillo antonio@Linux-Antonio:~/git/AC/Practica3/codigos] 2018-05-14 lunes
$ ./pmm-secuencial.out
Introduce el tamaño de la matriz cuadrada: 3
La matriz2 inicializada es:
[2,2,2]
[2,2,2]
[2,2,2]
El matriz1 inicializado es:
[3,3,3]
[3,3,3]
[3,3,3]
El resultado de multiplicarlo es:
[18,18,18]
[18,18,18]
[18,18,18]
El tiempo usado es0.000001:

```

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

CAPTURAS DE PANTALLA:

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

ESTUDIO DE ESCALABILIDAD EN atcgrid:

SCRIPT: pmm-OpenMP_atcgrid.sh

--

ESTUDIO DE ESCALABILIDAD EN PCLOCAL:

SCRIPT: pmm-OpenMP_pclocal.sh

--