

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): Juan Manuel Salcedo Serrano

Grupo de prácticas: B1

Fecha de entrega: 12/05/2016

Fecha evaluación en clase: 12/05/2016

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>
#include <omp.h>
int main(int argc, char **argv)
{
    int i, n=20, tid; int a[n], suma=0, sumalocal;
    if(argc < 3) {
        fprintf(stderr, "[ERROR]-Falta iteraciones o n hebras\n");
        exit(-1);
    }
    n = atoi(argv[1]); if (n>20) n=20;
    for (i=0; i<n; i++) {a[i] = i;}
    int hebras=atoi(argv[2]);

    #pragma omp parallel if(n>4) num_threads(hebras) 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);
    }
    return(0);
}
```

CAPTURAS DE PANTALLA:

```
* Modificados gcc -O2 -fopenmp -o if if_clause.c
* Modificados ./if 2 3
thread 0 suma de a[0]=0 sumalocal=0
thread 0 suma de a[1]=1 sumalocal=1
thread master=0 imprime suma=1
* Modificados ./if 6 3
thread 0 suma de a[0]=0 sumalocal=0
thread 0 suma de a[1]=1 sumalocal=1
thread 1 suma de a[2]=2 sumalocal=2
thread 1 suma de a[3]=3 sumalocal=5
thread 2 suma de a[4]=4 sumalocal=4
thread 2 suma de a[5]=5 sumalocal=9
thread master=0 imprime suma=15
* Modificados ./if 10 5
thread 1 suma de a[2]=2 sumalocal=2
thread 1 suma de a[3]=3 sumalocal=5
thread 3 suma de a[6]=6 sumalocal=6
thread 3 suma de a[7]=7 sumalocal=13
thread 0 suma de a[0]=0 sumalocal=0
thread 0 suma de a[1]=1 sumalocal=1
thread 4 suma de a[8]=8 sumalocal=8
thread 4 suma de a[9]=9 sumalocal=17
thread 2 suma de a[4]=4 sumalocal=4
thread 2 suma de a[5]=5 sumalocal=9
thread master=0 imprime suma=45
```

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

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

RESPUESTA:

Static: Se reparte de forma equitativa las las iteraciones entre hebras, en función del tamaño del chunk se repartirán un número contiguo de iteraciones.

Dynamic: Se reparten un número de iteraciones en función del chunk a cada hebra. Cuando una hebra acaba con las que tenía asignadas se vuelven a asignar.

Guided: Se reparten en función del tamaño del chunk grupos de iteraciones a cada hebra, conforme acaban se vuelven a asignar grupos de iteraciones, y así sucesivamente, reduciendo el número de iteraciones asignadas.

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
#endif

main(int argc, char **argv) {
    int i, n=200, chunk, a[n], suma=0;
    if(argc < 3) {
        fprintf(stderr, "\nFalta iteraciones o chunk \n");
        exit(-1);
    }
    n = atoi(argv[1]);
    if (n>200){ n=200;
    }
    chunk = atoi(argv[2]);
    for (i=0; i<n; i++){a[i] = i;}

    #pragma omp parallel
    {
        #pragma omp 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);
        }

        #pragma omp sections
        {
            #pragma omp section
            printf("Dentro de parallel dyn-var:%d \n", omp_get_dynamic());

            #pragma omp section
            printf("Dentro de parallel nthreads-var:%d\n", omp_get_max_threads());
        }

        printf("Fuera de parallel for suma = %d \n", suma);
        printf("Fuera de parallel dyn-var:%d \n", omp_get_dynamic());
        printf("Fuera de parallel nthreads-var:%d \n", omp_get_max_threads());
    }
}

```

CAPTURAS DE PANTALLA:

```

Modificados gcc -O2 -fopenmp -o scheduled scheduled-clause.c
Modificados ./scheduled 10 4
thread 2 suma a[4]=4 suma=4
thread 2 suma a[5]=5 suma=9
thread 2 suma a[6]=6 suma=15
thread 2 suma a[7]=7 suma=22
thread 1 suma a[8]=8 suma=8
thread 1 suma a[9]=9 suma=17
thread 0 suma a[0]=0 suma=0
thread 0 suma a[1]=1 suma=1
thread 0 suma a[2]=2 suma=3
thread 0 suma a[3]=3 suma=6
dentro de parallel dyn-var:0
dentro de parallel nthreads-var:4
fuera de parallel for suma = 17
fuera de parallel dyn-var:0
fuera de parallel nthreads-var:4

```

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
#endif

main(int argc, char **argv) {
    int i, n=200, chunk, a[n], suma=0;
    if(argc < 3) {
        fprintf(stderr, "\nFalta iteraciones o chunk \n");
        exit(-1);
    }
    n = atoi(argv[1]);
    if (n>200){
        n=200;
    }
    chunk = atoi(argv[2]);

    for (i=0; i<n; i++){
        a[i] = i;
    }

    #pragma omp parallel
    {
        #pragma omp single
        {

```

```

        printf("Fuera de parallel omp_get_num_threads: %d\n", omp_get_num_threads());
        printf("Dentro de parallel omp_get_num_procs: %d\n", omp_get_num_procs());
        printf("Dentro de parallel omp_in_parallel: %d\n", omp_in_parallel());
    }

    #pragma omp 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);
    }

    #pragma omp sections
    {
        #pragma omp section
        printf("Dentro de parallel dyn-var:%d\n", omp_get_dynamic());

        #pragma omp section
        printf("Dentro de parallel nthreads-var:%d\n", omp_get_max_threads());
    }

    printf("Fuera de parallel for suma = %d\n", suma);
    printf("Fuera de parallel omp_get_num_threads: %d\n", omp_get_num_threads());
    printf("Fuera de parallel omp_get_num_procs: %d\n", omp_get_num_procs());
    printf("Fuera de parallel omp_in_parallel: %d\n", omp_in_parallel());
}

```

CAPTURAS DE PANTALLA:

```

+ Modificados gcc -O2 -fopenmp -o scheduled4 scheduled-claused4.c
+ Modificados ./scheduled4 8 2
Fuera de parallel omp_get_num_threads: 4
Dentro de parallel omp_get_num_procs: 4
Dentro de parallel omp_in_parallel: 1
 thread 0 suma a[0]=0 suma=0
 thread 0 suma a[1]=1 suma=1
 thread 3 suma a[6]=6 suma=6
 thread 3 suma a[7]=7 suma=13
 thread 1 suma a[4]=4 suma=4
 thread 1 suma a[5]=5 suma=9
 thread 2 suma a[2]=2 suma=2
 thread 2 suma a[3]=3 suma=5
Dentro de parallel dyn-var:0
Dentro de parallel nthreads-var:4
Fuera de parallel for suma = 13
Fuera de parallel omp_get_num_threads: 1
Fuera de parallel omp_get_num_procs: 4
Fuera de parallel omp_in_parallel: 0

```

RESPUESTA:

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*

```
/* Tipo de letra Courier new o Liberation Mono. Tamaño 8 o 9 .*/  
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/  
/* INTERLINEADO SENCILLO */  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <omp.h>  
  
int main(int argc, char ** argv)  
{  
...  
}
```

CAPTURAS DE PANTALLA:

RESPUESTA:

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

void main(int argc, char **argv) {
    if(argc<2){
        fprintf(stderr, "Necesario tamaño. Ej: ./pmtvsec N \n");
        exit(-1);
    }

    int N=atoi(argv[1]), i, j;
    int *V;
    V = (int *)malloc (10*N);
    int *Vres;
    Vres = (int *)malloc (10*N);
    int **M;
    M = (int **)malloc (10*N);
    for(i=0; i<N; i++){ M[i] = (int *)malloc (10*N); }

    for(i=0; i<N; i++){
        for(j=0; j<N; j++){
            if(j >= i){M[i][j] = i+j+1; }
            else{ M[i][j] = 0; }
        }
    }

    for(i=0; i<N; i++){
        V[i] = i+1;
        Vres[i] = 0;
    }

    printf("M: \n");
    for(i=0; i<N; i++){
        for(j=0; j<N; j++){ printf("%d ", M[i][j]); }
        printf("\n");
    }
    printf("\nV: \n");
    for(i=0; i<N; i++){printf("%d \n", V[i]); }
    printf("\n");

    for(i=0; i<N; i++){
        Vres[i] = 0;
        for(j=0; j<N; j++){
            if(j >= i){ Vres[i] += M[i][j] * V[j]; }
        }
    }
}
```



```

        else{Vres[i] = 0; }
    }

    if(N<10) {
        printf("\nResultado:\n");
        for(i=0; i<N; i++){ printf("%d ",Vres[i]); }
    }else{
        printf("Primer componente(Vresultados[0]):%d\n",Vres[0]);
        printf("Último componente(Vresultados[N-1]):%d\n",Vres[N-1]);
    }

    for(i=0; i<N; i++) {free(M[i]); }
    free(M);
    free(V);
    free(Vres);
}

```

CAPTURAS DE PANTALLA:

```

+ Ejercicios gcc -O2 -fopenmp -o pmtvsec pmtv-secuencial.c
+ Ejercicios ./pmtvsec 2
M:
1 2
0 3
V:
1
2
Resultado:
5 6 3

```

```

+ Ejercicios ./pmtvsec 11
M:
1 2 3 4 5 6 7 8 9 10 11
0 3 4 5 6 7 8 9 10 11 12
0 0 5 6 7 8 9 10 11 12 13
0 0 0 7 8 9 10 11 12 13 14
0 0 0 0 9 10 11 12 13 14 15
0 0 0 0 0 11 12 13 14 15 16
0 0 0 0 0 0 13 14 15 16 17
0 0 0 0 0 0 0 15 16 17 18
0 0 0 0 0 0 0 0 17 18 19
0 0 0 0 0 0 0 0 0 19 20
0 0 0 0 0 0 0 0 0 0 21
V:
1
2
3
4
5
6
7
8
9
10
11
Primer componente(Vresultados[0]):506
Último componente(Vresultados[N-1]):231

```

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 ategrid 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:

CÓDIGO FUENTE: *pmtv-OpenMP.c*

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

void main(int argc, char **argv) {
    if(argc<2){
        fprintf(stderr, "Necesario tamaño. Ej: ./pmtvpar N \n");
        exit(-1);
    }

    int N=atoi(argv[1]), i, j;
    int *V;
    V = (int *)malloc (10*N);
    int *Vres;
    Vres = (int *)malloc (10*N);
    int **M;
    M = (int **)malloc (10*N);

    for(i=0; i<N; i++){ M[i] = (int *)malloc (10*N); }

    #pragma omp parallel shared(M,V,Vres) private(i,j)
        #pragma omp for schedule (runtime)
    for(i=0; i<N; i++){
        for(j=0; j<N; j++){
            if(j >= i){M[i][j] = i+j+1; }
            else{ M[i][j] = 0; }
        }
    }

    #pragma omp for schedule (runtime)
    for(i=0; i<N; i++){ V[i] = i+1; }

    #pragma omp for schedule (runtime)
    for(i=0; i<N; i++){ Vres[i] = 0; }

    printf("M:\n");
    for(i=0; i<N; i++){
```

```

        for(j=0; j<N; j++){ printf("%d ",M[i][j]); }
        printf("\n");
    }
    printf("\nV:\n");
    for(i=0; i<N; i++){printf("%d \n",V[i]);}
    printf("\n");

#pragma omp parallel shared(M,V,Vres) private(i,j)
    #pragma omp for schedule (runtime)
    for(i=0; i<N; i++){
        Vres[i] = 0;
        for(j=0; j<N; j++){
            if(j >= i){ Vres[i] += M[i][j] * V[j]; }
            else{ Vres[i] = 0; }
        }
    }

    if(N<10){
        printf("\nResultado:\n");
        for(i=0; i<N; i++){printf("%d ",Vres[i]); }
    }else{
        printf("Primer componente (Vresultados[0]):%d\n", Vres[0]);
        printf("Último componente (Vresultados[N-1]):%d\n", Vres[N-1]);
    }

    //Memoria
    for(i=0; i<N; i++)
        free(M[i]);

    free(M);
    free(V);
    free(Vres);
}

```

CAPTURAS DE PANTALLA:

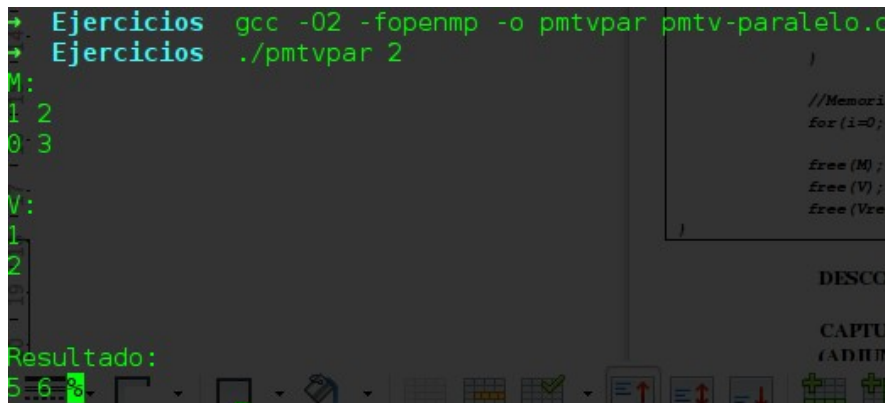


TABLA RESULTADOS, SCRIPT Y GRÁFICA ATCGRID

SCRIPT: pmvt-OpenMP_atcgrid.sh

```
#!/bin/bash
size=10000
echo "Chunk" > ./logs/log.0
echo 1 >> ./logs/log.0
echo 64 >> ./logs/log.0

echo "Static" > ./logs/s
./pmtv-OpenMP-static 1 $size ok >> ./logs/s
./pmtv-OpenMP-static 64 $size ok >> ./logs/s

echo "Dynamic" > ./logs/d
./pmtv-OpenMP-dynamic 1 $size ok >> ./logs/d
./pmtv-OpenMP-dynamic 64 $size ok >> ./logs/d

echo "Guided" > ./logs/g
./pmtv-OpenMP-guided 1 $size ok >> ./logs/g
./pmtv-OpenMP-guided 64 $size ok >> ./logs/g

paste ./logs/log.0 ./logs/s ./logs/d ./logs/g > ./logs/log_final

cat ./logs/log_final
```

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.

CÓDIGO FUENTE: `pmm-secuencial.c`

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define MAX 4096
int main(int argc, char **argv){
    int i, j, k; int dimension_matrices; int suma; suma = 0;
    int **matrizB; int **matrizC; int **matrizA;
    if (argc < 2){
        printf("Falta el número de componentes\n");
        return(1);
    }
    dimension_matrices = atoi(argv[1]);
    if (dimension_matrices > MAX){
        printf("Tamaño demasiado grande. No superar %d\n\n", MAX);
        return(1);
    }
    matrizB = (int **)malloc(dimension_matrices * sizeof(int*));
    matrizC = (int **)malloc(dimension_matrices * sizeof(int*));
    matrizA = (int **)malloc(dimension_matrices * sizeof(int*));
    for (i=0; i<dimension_matrices; i++){
        matrizB[i] = (int *)malloc(dimension_matrices * sizeof(int));
        matrizC[i] = (int *)malloc(dimension_matrices * sizeof(int));
        matrizA[i] = (int *)malloc(dimension_matrices * sizeof(int));
    }
    for (j=0; j<dimension_matrices; j++){
        for (i=0; i<dimension_matrices; i++){
            matrizB[j][i] = j+i;
            matrizC[j][i] = j*i;
        }
    }
    for (i=0; i<dimension_matrices; i++){
        for(j=0; j<dimension_matrices; j++){
            suma = 0;
            for (k=0; k<dimension_matrices; k++){
                suma += (matrizB[i][k]*matrizC[k][j]);
            }
            matrizA[i][j]=suma;
        }
    }
    printf ("Resultado[0][0] = %d\n",matrizA[0][0]);
    printf ("Componente(N-1,N-1) del resultado de la multiplicación de ambas matrices=
%d\n",matrizA[dimension_matrices-1][dimension_matrices-1]);
    free(matrizA);
    free(matrizB);
    free(matrizC);
    return 0;
}
```

CAPTURAS DE PANTALLA:

```

+ Ejercicios gcc -O2 -fopenmp -o pmmsec pmm-secuencial.c
+ Ejercicios ./pmmsec 12
Resultado[0][0] = 0
Componente(N-1,N-1) del resultado de la multiplicación de ambas matrices=1
3552
+ Ejercicios ./pmmsec 5
Resultado[0][0] = 0
Componente(N-1,N-1) del resultado de la multiplicación de ambas matrices=2
80

```

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).

CÓDIGO FUENTE: *pmm-OpenMP.c*

```

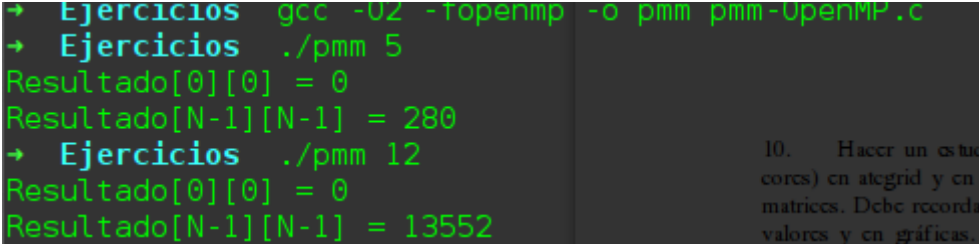
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define MAX 4096

int main(int argc, char **argv){
    int i, j, k; int dimension_matrices; int suma; suma = 0;
    int **matrizB; int **matrizC; int **matrizA;
    if (argc < 2){
        printf("Falta el número de componentes\n");
        return(1);
    }
    dimension_matrices = atoi(argv[1]);
    if (dimension_matrices > MAX){
        printf("Tamaño demasiado grande. No superar %d\n\n",MAX);
        return(1);
    }
    matrizB = (int **)malloc(dimension_matrices * sizeof(int*));
    matrizC = (int **)malloc(dimension_matrices * sizeof(int*));
    matrizA = (int **)malloc(dimension_matrices * sizeof(int*));
    for (i=0; i<dimension_matrices; i++){
        matrizB[i] = (int *)malloc(dimension_matrices * sizeof(int));
        matrizC[i] = (int *)malloc(dimension_matrices * sizeof(int));
        matrizA[i] = (int *)malloc(dimension_matrices * sizeof(int));
    }
    for (j=0; j<dimension_matrices; j++){
        for (i=0; i<dimension_matrices; i++){
            matrizB[j][i] = j+i; matrizC[j][i] = j*i;
        }
    }
    #pragma omp for private(i,j,k) schedule (runtime)
    for (i=0; i<dimension_matrices; i++){
        for(j=0; j<dimension_matrices; j++){
            suma = 0;
            for (k=0; k<dimension_matrices; k++){
                suma += (matrizB[i][k]*matrizC[k][j]);
            }
            matrizA[i][j]=suma;
        }
    }
    printf ("Resultado[0][0] = %d\n",matrizA[0][0]);

```

```
        printf ("Resultado[N-1][N-1] = %d\n",matrizA[dimension_matrices-1][dimension_matrices-1]);  
    }  
    free(matrizA);  
    free(matrizB);  
    free(matrizC);  
    return 0;  
}
```

CAPTURAS DE PANTALLA:



→ Ejercicios gcc -O2 -fopenmp -o pmm pmm-OpenMP.c
→ Ejercicios ./pmm 5
Resultado[0][0] = 0
Resultado[N-1][N-1] = 280
→ Ejercicios ./pmm 12
Resultado[0][0] = 0
Resultado[N-1][N-1] = 13552

10. Hacer un estudio de los núcleos (cores) en la arquitectura de la tarjeta gráfica y en las matrices. Debe recordarse los valores y en gráficas.