

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): jiaqi wang

Grupo de prácticas: b1

Fecha de entrega:19/05

Fecha evaluación en clase:

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

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

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

    #pragma omp parallel if(n>4) default(none) private(sumalocal,tid)
    shared(a,suma,n) num_threads(x)
    {
        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:

```
jiaqi@jiaqi:~/Dropbox/2_Cuatrimestre/acPractica/BP3$ ./if-clauseModificado 5 5
thread 1 suma de a[1]=1 sumalocal=1
thread 0 suma de a[0]=0 sumalocal=0
thread 2 suma de a[2]=2 sumalocal=2
thread 3 suma de a[3]=3 sumalocal=3
thread 4 suma de a[4]=4 sumalocal=4
thread master=0 imprime suma=10
```

RESPUESTA:

El argumento que he pasado es 5 (iteraciones) y 5 (hebras) por lo tanto se ejecuta una iteración por hebra.

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

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

RESPUESTA:

El `static` se asigna estáticamente los trabajos a cada hebra disponible mientras que en el `dynamic`, una hebra puede escoger trabajos conforme acaba su ejecución. En el `guided`, se asigna número de trabajos decrecientes, comenzando por el tamaño máximo especificado.

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

void main(int argc, char **argv){
    int i, n=200, chunk, a[n], suma=0;
    if(argc<3){
        fprintf(stderr, "\nFalta numero de interacciones 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());
            #pragma omp section
            printf("Dentro de parallel thread-limit-var:%d \n", omp_get_thread_limit());
            /*#pragma omp section
            printf("Dentro de parallel run-sched-var:%d \n", omp_get_schedule(dynamic, chunk));
            */
        }
    }
    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());
    printf("Fuera de parallel thread-limit-var:%d \n", omp_get_thread_limit());
    // printf("Fuera de parallel run-sched-var:%d\n", omp_get_schedule(dynamic, chunk));
}
```

CAPTURAS DE PANTALLA:

```
jiaqi@jiaqi:~/Dropbox/2_Cuatrimestre/acPractica/BP3$ ./scheduled-clauseModificad
o 5 5
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
thread 0 suma a[4]=4 suma=10
Dentro de parallel dyn-var:0
Dentro de parallel thread-limit-var:2147483647
Dentro de parallel nthreads-var:2
Fuera de parallel for suma = 10
Fuera de parallel dyn-var:0
Fuera de parallel nthreads-var:2
Fuera de parallel thread-limit-var:2147483647
```

```
jiaqi@jiaqi:~/Dropbox/2_Cuatrimestre/acPractica/BP3$ export OMP_NUM_THREAD=5
jiaqi@jiaqi:~/Dropbox/2_Cuatrimestre/acPractica/BP3$ ./scheduled-clauseModificad
o 5 2
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
thread 0 suma a[4]=4 suma=10
Dentro de parallel dyn-var:0
Dentro de parallel nthreads-var:2
Dentro de parallel thread-limit-var:2147483647
Fuera de parallel for suma = 10
Fuera de parallel dyn-var:0
Fuera de parallel nthreads-var:2
Fuera de parallel thread-limit-var:2147483647
```

```
jiaqi@jiaqi:~/Dropbox/2_Cuatrimestre/acPractica/BP3$ export OMP_DYNAMIC=FALSE
jiaqi@jiaqi:~/Dropbox/2_Cuatrimestre/acPractica/BP3$ ./scheduled-clauseModificad
o 5 4
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
thread 0 suma a[4]=4 suma=10
Dentro de parallel dyn-var:0
Dentro de parallel nthreads-var:2
Dentro de parallel thread-limit-var:2147483647
Fuera de parallel for suma = 10
Fuera de parallel dyn-var:0
Fuera de parallel nthreads-var:2
Fuera de parallel thread-limit-var:2147483647
```

RESPUESTA:

Si, muestra mismo numero

ps: el run-sched-var no compila

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

void main(int argc, char **argv){
    int i, n=200, chunk, a[n], suma=0;
    if(argc<3){
        fprintf(stderr, "\nFalta numero de interacciones 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 master
        printf("Dentro de parallel omp_get_num_threads():%d\n", omp_get_num_threads());
        #pragma omp master
        printf("Dentro de parallel omp_get_num_procs():%d \n", omp_get_num_procs());
        #pragma omp master
        printf("Dentro de parallel omp_in_parallel():%d \n", omp_in_parallel());
    }
    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());
}
```

```
jiaqi@jiaqi:~/Dropbox/2_Cuatrimestre/acPractica/BP3$ ./scheduled-clauseModificado5 5 5
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
thread 0 suma a[4]=4 suma=10
Dentro de parallel omp_get_num_threads():2
Dentro de parallel omp_get_num_procs():2
Dentro de parallel omp_in_parallel():1
Fuera de parallel omp_get_num_threads():1
Fuera de parallel omp_get_num_procs():2
Fuera de parallel omp_in_parallel():0
```

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

void main(int argc, char **argv){
    int i, n=200, chunk, a[n], suma=0;
    if(argc<3){
        fprintf(stderr, "\nFalta numero de interacciones 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 master
        printf("Dentro de parallel omp_get_dynamic():%d \n", omp_get_dynamic());
        #pragma omp master
        printf("Dentro de parallel omp_get_max_threads():%d\n", omp_get_max_threads());
        /* #pragma omp master
        printf("Dentro de parallel omp_get_schedule(&kind, &modifier):%d\n", omp_get_schedule(&kind, &modifier));*/
    }
    printf("Fuera de parallel omp_get_dynamic():%d \n", omp_get_dynamic());
    printf("Fuera de parallel omp_get_max_threads():%d\n", omp_get_max_threads());
    /* printf("Fuera de parallel omp_get_schedule(&kind, &modifier):%d\n", omp_get_schedule(&kind, &modifier));
    */

    printf("Total suma=%d \n", suma);
}
```

CAPTURAS DE PANTALLA:

```

jiaqi@jiaqi:~/Dropbox/2_Cuatrimestre/acPractica/BP3$ ./scheduled-clauseModificad
5 5 5
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
thread 0 suma a[4]=4 suma=10
Dentro de parallel omp_get_dynamic():0
Dentro de parallel omp_get_max_threads():2
Dentro de parallel omp_get_dynamic():0
Dentro de parallel omp_get_max_threads():2
Total suma=10

```

6. Implementar un programa secuencial en C que multiplique una matriz triangular por un 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

```

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

main(int argc, char **argv) {
    if(argc < 2) {
        fprintf(stderr, "falta el número de
filas/columnas \n");
        exit(-1);
    }

    int N=atoi(argv[1]), i, j; //filas/columnas de la matriz
    double t1,t2;
    int *Vector;
    Vector = (int *)malloc (sizeof(int)*N);
    int *Vector2; //es el vector resultado
    Vector2 = (int *)malloc (sizeof(int)*N);
    int **matriz;
    matriz = (int **)malloc(N * sizeof(int *));
    int col=N;
    for (i = 0; i < N; ++i){
        matriz[i] = (int *)malloc(col * sizeof(int));
        col=col-1;
    }

    //inicializando la matriz
    col=N;
    //Inicializa la matriz

```



```

        for (i=0; i<N; i++){
            for (j=0; j<col; j++){
                matriz[i][j]=i+1;
                //printf("Valor
%d\n",matriz[i][j]);

            }
            col--;
        }

//inicializando el vector
for(i=0; i<N; i++){
    Vector[i] = i+1;
    Vector2[i] = 0;
}

//tiempo
t1 = omp_get_wtime();
//multiplicacion
for(i=0; i<N; i++){
    Vector2[i] = 0;
    for(j=0; j<N; j++){
        Vector2[i] += matriz[i][j] * Vector[j];
    }
}
t2 = omp_get_wtime();
t2 = t2-t1;

//mostrar resultados
if(N<10){
    printf("matriz:\n");
    for(i=0; i<N; i++){
        for(j=0; j<N; j++)
        {
            printf("%d
",matriz[i][j]);

        }
        printf("\n");
    }
    printf("\nVector:\n");
    for(i=0; i<N; i++) printf("%d ",Vector[i]);
    printf("\nResultado matriz x Vector:\n");
    for(i=0; i<N; i++) printf("%d ",Vector2[i]);
}else{//si la matriz es demasiado grande
    printf("Vector2[0]:%d\n",Vector2[0]);
    printf("Vector2[N-1]:%d\n",Vector2[N-1]);
}
printf("\nTiempo en ejecutar matrizxVector:%8.6f\n",t2);

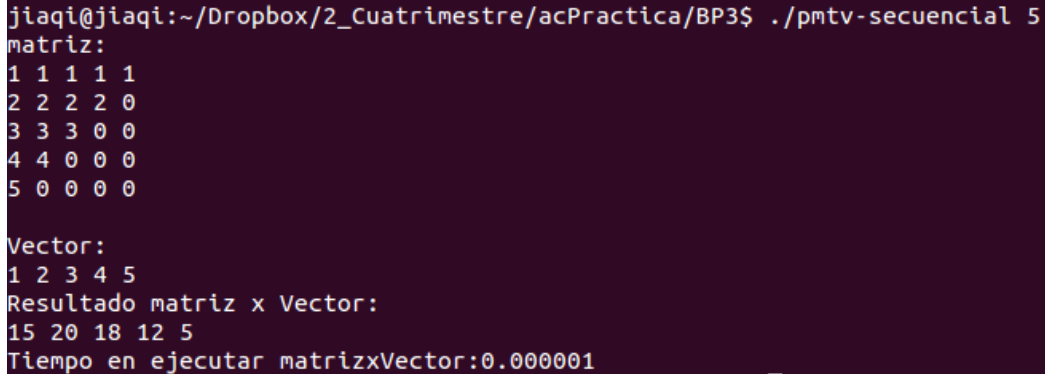
//liberar memoria
for(i=0; i<N; i++)
    free(matriz[i]);
free(matriz);

```

```

        free(Vector);
        free(Vector2);
    }

```

CAPTURAS DE PANTALLA:


```

jiaqi@jiaqi:~/Dropbox/2_Cuatrimestre/acPractica/BP3$ ./pmtv-secuencial 5
matriz:
1 1 1 1 1
2 2 2 2 0
3 3 3 0 0
4 4 0 0 0
5 0 0 0 0

Vector:
1 2 3 4 5
Resultado matriz x Vector:
15 20 18 12 5
Tiempo en ejecutar matrizxVector:0.000001

```

(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. 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 que multiplica una matriz triangular por un vector con las alternativas de planificación `static`, `dynamic` y `guided` para `chunk` de 2, 64, 128, 1024 y el `chunk` por defecto para la alternativa. No use vectores mayores de 32768 componentes ni menores de 4096 componentes. El número de threads en las ejecuciones debe coincidir con el número de cores. Rellenar la Tabla 3 con los tiempos obtenidos, ponga en la tabla el número de threads que utilizan las ejecuciones. Representar el tiempo para `static`, `dynamic` y `guided` en función del tamaño del `chunk` en una gráfica. Rellenar la tabla y realizar la gráfica también para el PC local. ¿Qué alternativa ofrece mejores prestaciones? Razone por qué.

CÓDIGO FUENTE: pmtv-OpenMP.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>
#ifdef _OPENMP
    #include <omp.h>
#else
    #define omp_get_thread_num() 0
#endif

main(int argc, char **argv) {
    if(argc < 2) {
        fprintf(stderr, "falta el número de
filas/columnas  \n");
        exit(-1);
    }
}

```

```

int N=atoi(argv[1]), i, j;//filas/columnas de la matriz
double t1,t2;
int *Vector;
Vector = (int *)malloc (sizeof(int)*N);
int *Vector2;//es el vector resultado
Vector2 = (int *)malloc (sizeof(int)*N);
int **matriz;
matriz = (int **)malloc(N * sizeof(int *));
int col=N;
for (i = 0; i < N; ++i){
    matriz[i] = (int *)malloc(col * sizeof(int));
    col=col-1;
}

//Inicializa la matriz
col=N;
#pragma omp parallel shared(matriz,Vector,Vector2,col)
private(i,j)
{
    #pragma omp for schedule (runtime)
    for (i=0; i<N; i++){
        for (j=0; j<col; j++){
            matriz[i]
[j]=i+1;
            //printf("V
alor%d\n",matriz[i][j]);

        }
        col--;
    }

    //inicializando el vector
    #pragma omp for schedule (runtime)
    for(i=0; i<N; i++){
        Vector[i] = i+1;
        Vector2[i] = 0;
    }
}

//tiempo
t1 = omp_get_wtime();
//multiplicacion
#pragma omp parallel shared(matriz,Vector,Vector2) private(i,j)
{
    #pragma omp for schedule (runtime)
    for(i=0; i<N; i++){
        Vector2[i] = 0;
        for(j=0; j<N; j++){
            Vector2[i] =
Vector2[i] +matriz[i][j] * Vector[j];
        }
    }
}
t2 = omp_get_wtime();
t2 = t2-t1;

```

```

        //mostrar resultados
        if(N<10){
            printf("matriz:\n");
            for(i=0; i<N; i++){
                for(j=0; j<N; j++)
                {
                    printf("%d
",matriz[i][j]);

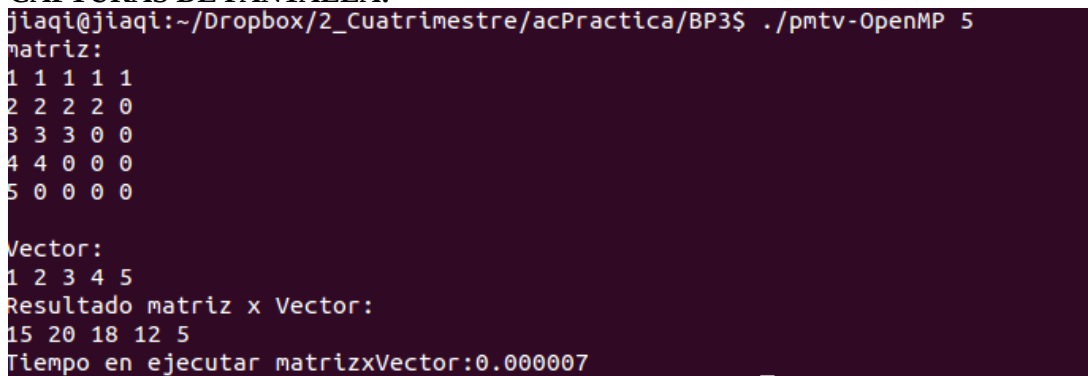
                }
                printf("\n");
            }
            printf("\nVector:\n");
            for(i=0; i<N; i++) printf("%d ",Vector[i]);
            printf("\nResultado matriz x Vector:\n");
            for(i=0; i<N; i++) printf("%d ",Vector2[i]);
        }else{//si la matriz es demasiado grande
            printf("Vector2[0]:%d\n",Vector2[0]);
            printf("Vector2[N-1]:%d\n",Vector2[N-1]);
        }
        printf("\nTiempo en ejecutar matrizxVector:%8.6f\n",t2);

        //liberar memoria
        for(i=0; i<N; i++)
            free(matriz[i]);
        free(matriz);
        free(Vector);
        free(Vector2);
    }

#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

int main(int argc, char ** argv)
{
    ...
}

```

CAPTURAS DE PANTALLA:


```

jlaqi@jlaqi:~/Dropbox/2_Cuatrimestre/acPractica/BP3$ ./pmtv-OpenMP 5
matriz:
1 1 1 1 1
2 2 2 2 0
3 3 3 0 0
4 4 0 0 0
5 0 0 0 0

Vector:
1 2 3 4 5
Resultado matriz x Vector:
15 20 18 12 5
Tiempo en ejecutar matrizxVector:0.000007

```

(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

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

void main(int argc, char **argv) {

    if(argc < 2) {
        fprintf(stderr, "falta el número de
filas/columnas\n");
        exit(-1);
    }

    int N=atoi(argv[1]), i, j, k;
    double t1,t2;

    int **matriz;//matriz resultante
    int **matriz1;
    int **matriz2;
    //reservar memoria
    matriz = (int **)malloc (N*sizeof(int *));
    for(i=0; i<N; i++)
        matriz[i] = (int *)malloc (N*sizeof(int));

    matriz1 = (int **)malloc (N*sizeof(int *));
    for(i=0; i<N; i++)
        matriz1[i] = (int *)malloc (N*sizeof(int));

    matriz2 = (int **)malloc (N*sizeof(int *));
    for(i=0; i<N; i++)
        matriz2[i] = (int *)malloc (N*sizeof(int));

    //inicializamos la matriz

    for(i=0; i<N; i++)
        for(j=0; j<N; j++)
            matriz[i][j] = 0;
```

```

        for(i=0; i<N; i++)
            for(j=0; j<N; j++)
                matriz1[i][j] = i+1;

        for(i=0; i<N; i++)
            for(j=0; j<N; j++)
                matriz2[i][j] = 1;//para facilitar la
comprobacion

        //multiplicacion
        t1 = omp_get_wtime();
        for (i = 0; i < N; ++i) {
        for (j = 0; j < N; ++j) {
        for (k = 0; k < N; ++k) {
            matriz[i][j] = matriz[i][j] + matriz1[i][k] * matriz2[k][j];
        }
        }
    }
    t2 = omp_get_wtime();
    t2 = t2-t1;

    //mostrar resultado
    if(N<10){
        printf("matriz1:\n");
        for(i=0; i<N; i++){
            for(j=0; j<N; j++)
                printf("%d ",matriz1[i]
[j]);

            printf("\n");
        }
        printf("matriz2:\n");
        for(i=0; i<N; i++){
            for(j=0; j<N; j++)
                printf("%d ",matriz2[i]
[j]);

            printf("\n");
        }
        printf("\nResultado de multiplicación:\n");
        for(i=0; i<N; i++){
            for(j=0; j<N; j++)
                printf("%d ",matriz[i]
[j]);

            printf("\n");
        }
    }else{
        printf("matriz[0][0]:%d\n",matriz[0][0]);
        printf("matriz[N-1][N-1]:%d\n",matriz[N-1][N-1]);
    }
    printf("\nTiempo en ejecutar matriz1xmatriz2:%8.6f\n",t2);

    //libera memoria
    for(i=0; i<N; i++)
        free(matriz1[i]);
    free(matriz1);
    for(i=0; i<N; i++)
        free(matriz2[i]);
    free(matriz2);
    for(i=0; i<N; i++)
        free(matriz[i]);

```

```

        free(matriz);
    }

```

CAPTURAS DE PANTALLA:

```

jiaqi@jiaqi:~/Dropbox/2_Cuatrimestre/acPractica/BP3$ ./pmm-OpenMP 5
matriz1:
1 1 1 1 1
2 2 2 2 2
3 3 3 3 3
4 4 4 4 4
5 5 5 5 5
matriz2:
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
Resultado de multiplicación:
5 5 5 5 5
10 10 10 10 10
15 15 15 15 15
20 20 20 20 20
25 25 25 25 25
Tiempo en ejecutar matriz1xmatriz2:0.000015

```

(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

```

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

void main(int argc, char **argv) {

    if(argc < 2) {
        fprintf(stderr, "falta el número de
filas/columnas\n");
        exit(-1);
    }

```

```

    }

    int N=atoi(argv[1]), i, j, k;
    double t1,t2;

    int **matriz;//matriz resultante
    int **matriz1;
    int **matriz2;
    //reservar memoria
    matriz = (int **)malloc (N*sizeof(int *));
    for(i=0; i<N; i++)
        matriz[i] = (int *)malloc (N*sizeof(int));

    matriz1 = (int **)malloc (N*sizeof(int *));
    for(i=0; i<N; i++)
        matriz1[i] = (int *)malloc (N*sizeof(int));

    matriz2 = (int **)malloc (N*sizeof(int *));
    for(i=0; i<N; i++)
        matriz2[i] = (int *)malloc (N*sizeof(int));

    //inicializamos la matriz
    #pragma omp parallel shared(matriz1,matriz2,matriz) private(i,j)
    {
        #pragma omp for schedule (runtime)
        for(i=0; i<N; i++)
            for(j=0; j<N; j++)
                matriz[i][j] = 0;
        #pragma omp for schedule (runtime)
        for(i=0; i<N; i++)
            for(j=0; j<N; j++)
                matriz1[i][j] = i+1;
        #pragma omp for schedule (runtime)
        for(i=0; i<N; i++)
            for(j=0; j<N; j++)
                matriz2[i][j] = 1;//para
    }
    facilitar la comprobacion

    }
    //multiplicacion
    t1 = omp_get_wtime();
    #pragma omp parallel shared(matriz1,matriz2,matriz)
private(i,j,k)
    {
        #pragma omp for schedule (runtime)
        for (i = 0; i < N; ++i) {
            for (j = 0; j < N; ++j) {
                for (k = 0; k < N; ++k) {
                    matriz[i][j] = matriz[i][j] + matriz1[i][k]
* matriz2[k][j];
                }
            }
        }
    }
    t2 = omp_get_wtime();

```



```

        t2 = t2-t1;

        //mostrar resultado
        if(N<10){
            printf("matriz1:\n");
            for(i=0; i<N; i++){
                for(j=0; j<N; j++)
                    printf("%d ",matriz1[i]
[j]);
                printf("\n");
            }
            printf("matriz2:\n");
            for(i=0; i<N; i++){
                for(j=0; j<N; j++)
                    printf("%d ",matriz2[i]
[j]);
                printf("\n");
            }
            printf("\nResultado de multiplicación:\n");
            for(i=0; i<N; i++){
                for(j=0; j<N; j++)
                    printf("%d ",matriz[i]
[j]);
                printf("\n");
            }
        }
        else{
            printf("matriz[0][0]:%d\n",matriz[0][0]);
            printf("matriz[N-1][N-1]:%d\n",matriz[N-1][N-1]);
        }
        printf("\nTiempo en ejecutar matriz1xmatriz2:%8.6f\n",t2);

        //libera memoria
        for(i=0; i<N; i++)
            free(matriz1[i]);
        free(matriz1);
        for(i=0; i<N; i++)
            free(matriz2[i]);
        free(matriz2);
        for(i=0; i<N; i++)
            free(matriz[i]);
        free(matriz);
    }

```

CAPTURAS DE PANTALLA:

```
jtaqi@jtaqi:~/Dropbox/2_Cuatrimestre/acPractica/8P3$ ./pmm-secuencial 5
matriz1:
1 1 1 1 1
2 2 2 2 2
3 3 3 3 3
4 4 4 4 4
5 5 5 5 5
matriz2:
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
Resultado de multiplicación:
5 5 5 5 5
10 10 10 10 10
15 15 15 15 15
20 20 20 20 20
25 25 25 25 25
```

(ADJUNTAR CÓDIGO FUENTE AL .ZIP)