

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): Francisco Javier Bolívar Lupiáñez

Grupo de prácticas: B1

Fecha de entrega: 12/05/2014

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`

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

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

    if(argc < 3) {
        fprintf(stderr, "[ERROR] params: <iteraciones> <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:

```

fb lupi@fb lupi-ElementaryOS:~/Dropbox/FACULTAD/Grado Informatica/2o Curso/2o Cuatrimestre/AC/Practicas/P3$ ./if-clause 10 2
thread 1 suma de a[5]=5 sumalocal=5
thread 1 suma de a[6]=6 sumalocal=11
thread 1 suma de a[7]=7 sumalocal=18
thread 1 suma de a[8]=8 sumalocal=26
thread 1 suma de a[9]=9 sumalocal=35
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 0 suma de a[4]=4 sumalocal=10
thread master=0 imprime suma=45
fb lupi@fb lupi-ElementaryOS:~/Dropbox/FACULTAD/Grado Informatica/2o Curso/2o Cuatrimestre/AC/Practicas/P3$ ./if-clause 10 4
thread 3 suma de a[8]=8 sumalocal=8
thread 3 suma de a[9]=9 sumalocal=17
thread 2 suma de a[6]=6 sumalocal=6
thread 2 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 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 1 suma de a[5]=5 sumalocal=12
thread master=0 imprime suma=45
fb lupi@fb lupi-ElementaryOS:~/Dropbox/FACULTAD/Grado Informatica/2o Curso/2o Cuatrimestre/AC/Practicas/P3$ ./if-clause 10 8
thread 2 suma de a[4]=4 sumalocal=4
thread 0 suma de a[0]=0 sumalocal=0
thread 7 suma de a[9]=9 sumalocal=9
thread 3 suma de a[5]=5 sumalocal=5
thread 4 suma de a[6]=6 sumalocal=6
thread 5 suma de a[7]=7 sumalocal=7
thread 6 suma de a[8]=8 sumalocal=8
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 master=0 imprime suma=45

```

RESPUESTA: Con la cláusula `num_threads(n)` podemos fijar el número de threads que se ejecutarán en un código paralelo. En los ejemplos pasé en el primero 2, en el segundo 4 y en el tercero 8. Se puede observar que se han utilizado los threads que se fijaron con la cláusula `num_threads(n)`.

2. (a) Rellenar la Tabla 1 (se debe poner en la tabla el *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			scheduled-clause.c			scheduleg-clause.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	0	0	0
13	1	0	1	0	0	0	0	0	0
14	0	1	1	0	1	0	0	0	0
15	1	1	1	1	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			scheduled-clause.c			scheduleg-clause.c		
	1	2	4	1	2	4	1	2	4
0	0	0	0	1	0	0	1	0	0
1	1	0	0	0	0	0	1	0	0
2	2	1	0	3	1	0	1	0	0
3	3	1	0	2	1	0	1	0	0
4	0	2	1	1	3	1	0	2	2
5	1	2	1	1	3	1	0	2	2
6	2	3	1	1	2	1	0	2	2
7	3	3	1	1	2	1	3	3	2
8	0	0	2	1	0	3	3	3	1
9	1	0	2	1	0	3	3	3	1
10	2	1	2	1	0	3	2	1	1
11	3	1	2	1	0	3	2	1	1
12	0	2	3	1	0	2	1	0	3
13	1	2	3	1	0	2	1	0	3
14	2	3	3	1	0	2	1	0	3
15	3	3	3	1	0	2	1	0	3

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=16, chunk, a[n], suma=0, kind_, modifier_;

    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 master
    {
        omp_get_schedule(&kind_,&modifier_);
        printf("Dentro de 'parallel':\n");
        printf("dyn-var: %d\n", omp_get_dynamic());
        printf("nthreads-var: %d\n", omp_get_num_threads());
        printf("thread-limit-var: %d\n", omp_get_thread_limit());
        printf("run-sched-var: %d, %d\n", kind_, modifier_);
    }
}
//printf("Fuera de 'parallel for' suma=%d\n",suma);
omp_get_schedule(&kind_,&modifier_);
printf("Fuera de 'parallel':\n");
printf("dyn-var: %d\n", omp_get_dynamic());
printf("nthreads-var: %d\n", omp_get_num_threads());
printf("thread-limit-var: %d\n", omp_get_thread_limit());
printf("run-sched-var: %d, %d\n", kind_, modifier_);
}

```

CAPTURAS DE PANTALLA:

```

fblupi@fblupi-ElementaryOS:~/Dropbox/FACULTAD/Grado Informatica/2o Curso/2o Cuatrimestre/AC/Practicas/P3$ ./scheduled-clause 10 2
Dentro de 'parallel':
dyn-var: 0
nthreads-var: 4
thread-limit-var: 2147483647
run-sched-var: 2, 2
Fuera de 'parallel':
dyn-var: 0
nthreads-var: 1
thread-limit-var: 2147483647
run-sched-var: 2, 2
fblupi@fblupi-ElementaryOS:~/Dropbox/FACULTAD/Grado Informatica/2o Curso/2o Cuatrimestre/AC/Practicas/P3$ export OMP_SCHEDULE="STATIC,2"
fblupi@fblupi-ElementaryOS:~/Dropbox/FACULTAD/Grado Informatica/2o Curso/2o Cuatrimestre/AC/Practicas/P3$ ./scheduled-clause 10 2
Dentro de 'parallel':
dyn-var: 0
nthreads-var: 4
thread-limit-var: 2147483647
run-sched-var: 1, 2
Fuera de 'parallel':
dyn-var: 0
nthreads-var: 1
thread-limit-var: 2147483647
run-sched-var: 1, 2
fblupi@fblupi-ElementaryOS:~/Dropbox/FACULTAD/Grado Informatica/2o Curso/2o Cuatrimestre/AC/Practicas/P3$ export OMP_SCHEDULE="STATIC,5"
fblupi@fblupi-ElementaryOS:~/Dropbox/FACULTAD/Grado Informatica/2o Curso/2o Cuatrimestre/AC/Practicas/P3$ ./scheduled-clause 10 2
Dentro de 'parallel':
dyn-var: 0
nthreads-var: 4
thread-limit-var: 2147483647
run-sched-var: 1, 5
Fuera de 'parallel':
dyn-var: 0
nthreads-var: 1
thread-limit-var: 2147483647
run-sched-var: 1, 5

```

RESPUESTA: Tiene valor distinto la variable `nthreads-var`. En el bucle hay 4 y fuera solo 1. También hay que notar que como las variables de entorno tienen más prioridad que las cláusulas, estas cambiarán los valores de `run-sched-var`.

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) {

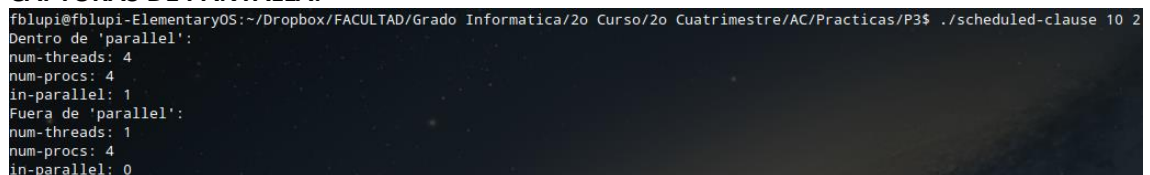
    int i, n=16, chunk, a[n], suma=0;

    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 master
        {
            printf("Dentro de 'parallel':\n");
            printf("num-threads: %d\n", omp_get_num_threads());
            printf("num-procs: %d\n", omp_get_num_procs());
            printf("in-parallel: %d\n", omp_in_parallel());
        }
    }
    //printf("Fuera de 'parallel for' suma=%d\n", suma);
    printf("Fuera de 'parallel':\n");
    printf("num-threads: %d\n", omp_get_num_threads());
    printf("num-procs: %d\n", omp_get_num_procs());
    printf("in-parallel: %d\n", omp_in_parallel());
}

```

CAPTURAS DE PANTALLA:


```

fblupi@fblupi-ElementaryOS:~/Dropbox/FACULTAD/Grado Informatica/2o Curso/2o Cuatrimestre/AC/Practicas/P3$ ./scheduled-clause 10 2
Dentro de 'parallel':
num-threads: 4
num-procs: 4
in-parallel: 1
Fuera de 'parallel':
num-threads: 1
num-procs: 4
in-parallel: 0

```

RESPUESTA: Cuando está dentro del parallel, el valor de in-parallel es 1, fuera es 0. El valor de num-procs siempre es constante, pero el de num-threads si varía dentro y fuera del parallel.

5. Añadir al programa scheduled-clause.c lo necesario para modificar las variables de control dyn-var, nthreads-var y run-sched-var y para poder imprimir el valor de estas variables antes y después de dicha modificación. Incorporar en su cuaderno de prácticas volcados de pantalla con los resultados de ejecución obtenidos.

CÓDIGO FUENTE: scheduled-clauseModificado5.c

```

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

main(int argc, char **argv) {

    int i, n=16, chunk, a[n], suma=0, dyn_, nthreads_, kind_, modifier_;

    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;

    omp_get_schedule(&kind_, &modifier_);
    printf("Antes de modificar:\n");
    printf("dyn-var: %d\n", omp_get_dynamic());
    printf("nthreads-var: %d\n", omp_get_num_threads());
    printf("run-sched-var: %d, %d\n", kind_, modifier_);

    printf("Introduce los valores de dyn-var, nthreads-var y run-sched-var (kind y modifier):\n");
    scanf("%d %d %d %d", &dyn_, &nthreads_, &kind_, &modifier_);

    omp_set_dynamic(dyn_);
    omp_set_num_threads(nthreads_);
    omp_set_schedule(kind_, modifier_);

    omp_get_schedule(&kind_, &modifier_);
    printf("Despues de modificar:\n");
    printf("dyn-var: %d\n", omp_get_dynamic());
    printf("nthreads-var: %d\n", omp_get_num_threads());
    printf("run-sched-var: %d, %d\n", kind_, modifier_);

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

```

CAPTURAS DE PANTALLA:

```

fblupi@fblupi-ElementaryOS:~/Dropbox/FACULTAD/Grado Informatica/2o Curso/2o Cuatrimestre/AC/Practicas/P3$ ./scheduled-clone 10 2
Antes de modificar:
dyn-var: 0
nthreads-var: 1
run-sched-var: 2, 1
Introduce los valores de dyn-var, nthreads-var y run-sched-var (kind y modifier):
1 2 3 4
Despues de modificar:
dyn-var: 1
nthreads-var: 1
run-sched-var: 3, 4

```

RESPUESTA: Cambian todos los valores de forma adecuada a excepción del `nthreads-var`, ya que estando fuera de la región parallel solo hay un thread.

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 las componentes (0,0) y (N-1, N-1) del resultado antes de que termine el programa.

CÓDIGO FUENTE: `pmtv-secuencial.c`

```

// gcc -O2 -fopenmp -o pmtv-secuencial pmtv-secuencial.c -lrt

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

// #define TIMES
// #define PRINTF_ALL

main(int argc, char **argv) {
    // 1. Lectura valores de entrada
    if(argc < 2) {
        fprintf(stderr, "Falta num\n");
        exit(-1);
    }
    int n = atoi(argv[1]);

    int i, j;
    struct timespec ini, fin; double transcurrido;

    // 2. Creación e inicialización de vector y matriz
    // Creación
    int *v1, *v2;
    v1 = (int*) malloc(n*sizeof(double));
    v2 = (int*) malloc(n*sizeof(double));

    int **M;
    M = (int**) malloc(n*sizeof(int*));
    for(i=0; i<n; i++)
        M[i] = (int*) malloc(n*sizeof(int));

    // Inicialización
    for(i=0; i<n; i++)
        v1[i] = i+1;

    int num=1;

```

```

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

// 3. Impresión de vector y matriz
#ifndef TIMES
#ifdef PRINTF_ALL
    printf("Vector inicial:\n");
    for (i=0; i<n; i++) printf("%d ",v1[i]);
    printf("\n");

    printf("Matriz inicial:\n");
    for (i=0; i<n; i++) {
        for (j=0; j<n; j++) {
            if(M[i][j]<10) printf(" %d ",M[i][j]);
            else printf("%d ",M[i][j]);
        }
        printf("\n");
    }
#endif
#endif

// 4. Cálculo resultado
clock_gettime(CLOCK_REALTIME,&ini);
for (i=0; i<n; i++) {
    v2[i]=0;
    for (j=0; j<=i; j++) {
        v2[i]+=M[i][j]*v1[j];
    }
}
clock_gettime(CLOCK_REALTIME,&fin);
transcurrido=(double) (fin.tv_sec-ini.tv_sec)+(double) ((fin.tv_nsec-
ini.tv_nsec)/(1.e+9));

// 5. Impresión de vector resultado
#ifndef TIMES
    printf("%d %11.9f\n",n,transcurrido);
#else
#ifdef PRINTF_ALL
    printf("Tiempo: %11.9f\n",transcurrido);
    printf("Vector resultado (M x v1):\n");
    for (i=0; i<n; i++) printf("%d ",v2[i]);
    printf("\n");
#else
    printf("Tiempo: %11.9f\n",transcurrido);
    printf("v2[0]: %d, v2[n-1]: %d\n",v2[0],v2[n-1]);
#endif
#endif

// 6. Eliminar de memoria
free(M);
free(v1);
free(v2);
}

```


CAPTURAS DE PANTALLA:

```
fblupi@fblupi-ElementaryOS:~/Dropbox/FACULTAD/Grado Informatica/2o Curso/2o Cuatrimestre/AC/Practicas/P3$ ./pmtv-secuencial 10
Tiempo: 0.000000958
v2[0]: 1, v2[n-1]: 2860
```

```
fblupi@fblupi-ElementaryOS:~/Dropbox/FACULTAD/Grado Informatica/2o Curso/2o Cuatrimestre/AC/Practicas/P3$ ./pmtv-secuencial 10000
Tiempo: 0.085762658
v2[0]: 1, v2[n-1]: 967099416
```

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`

```
// gcc -O2 -fopenmp -o pmtv-OpenMP pmtv-OpenMP.c -lrt

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

#define TIMES
// #define PRINTF_ALL

main(int argc, char **argv) {
    // 1. Lectura valores de entrada
    if(argc < 2) {
        fprintf(stderr, "Falta num\n");
        exit(-1);
    }
    int n = atoi(argv[1]);

    int i, j;
    struct timespec ini, fin; double transcurrido;

    // 2. Creación e inicialización de vector y matriz
    // Creación
    int *v1, *v2;
    v1 = (int*) malloc(n*sizeof(double));
    v2 = (int*) malloc(n*sizeof(double));

    int **M;
    M = (int**) malloc(n*sizeof(int*));
    for(i=0; i<n; i++)
```

```

    M[i] = (int*)malloc(n*sizeof(int));

    // Inicialización
    for(i=0;i<n;i++)
        v1[i]=i+1;

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

    // 3. Impresión de vector y matriz
    #ifndef TIMES
    #ifdef PRINTF_ALL
        printf("Vector inicial:\n");
        for (i=0; i<n; i++) printf("%d ",v1[i]);
        printf("\n");

        printf("Matriz inicial:\n");
        for (i=0; i<n; i++) {
            for (j=0; j<n; j++) {
                if(M[i][j]<10) printf(" %d ",M[i][j]);
                else printf("%d ",M[i][j]);
            }
            printf("\n");
        }
    #endif
    #endif

    // 4. Cálculo resultado
    clock_gettime(CLOCK_REALTIME,&ini);

    #pragma omp parallel for default(none) private(i,j) shared(n,v1,v2,M)
    schedule(runtime)
    for (i=0; i<n; i++) {
        v2[i]=0;
        for (j=0; j<=i; j++) {
            v2[i]+=M[i][j]*v1[j];
        }
    }

    clock_gettime(CLOCK_REALTIME,&fin);
    transcurrido=(double) (fin.tv_sec-ini.tv_sec)+(double) ((fin.tv_nsec-
    ini.tv_nsec)/(1.e+9));

    // 5. Impresión de vector resultado
    #ifndef TIMES
    printf("%d %11.9f\n",n,transcurrido);
    #else
    #ifdef PRINTF_ALL
        printf("Tiempo: %11.9f\n",transcurrido);
        printf("Vector resultado (M x v1):\n");
        for (i=0; i<n; i++) printf("%d ",v2[i]);
        printf("\n");
    #else
        printf("Tiempo: %11.9f\n",transcurrido);
        printf("v2[0]: %d, v2[n-1]: %d\n",v2[0],v2[n-1]);
    #endif
    #endif

```

```
#endif

// 6. Eliminar de memoria
free(M);
free(v1);
free(v2);
}
```

CAPTURAS DE PANTALLA:

```
fblupi@fblupi-ElementaryOS:~/Dropbox/FACULTAD/Grado Informatica/2o Curso/2o Cuatrimestre/AC/Practicas/P3$ ./pmtv-OpenMP 10
Tiempo: 0.000175800
v2[0]: 1, v2[n-1]: 2860
```

```
fblupi@fblupi-ElementaryOS:~/Dropbox/FACULTAD/Grado Informatica/2o Curso/2o Cuatrimestre/AC/Practicas/P3$ ./pmtv-OpenMP 10000
Tiempo: 0.048535908
v2[0]: 1, v2[n-1]: 967099416
```

Tabla 3 . Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones paralelas

PC Local			
Chunk	Static 4 threads	Dynamic 4 threads	Guided 4 threads
por defecto	0.039525456	0.038314050	0.036015239
2	0.039832814	0.038226166	0.036066118
32	0.037082367	0.036423377	0.036354660
64	0.036781692	0.036301228	0.036669390
2048	0.039225454	0.039179677	0.039310897

ATCGRID			
Chunk	Static 12 threads	Dynamic 12 threads	Guided 12 threads
por defecto	0.011242940	0.011261140	0.009462215
2	0.009570831	0.010860271	0.009358393
32	0.009491332	0.009659568	0.009486420
64	0.009477587	0.009593140	0.009571934
2048	0.021810070	0.019843144	0.019229252

RESPUESTA: En todos, el valor por defecto suele ser más o menos parecido al chunk=2. En atcgrid el usar 3 veces los threads que uso en el pc, los tiempos son aproximadamente tres veces más rápidos. Tanto en static como en dynamic, a mayor chunk, mayor eficiencia, ya que se reduce la carga de asignación de trabajo, pero si el chunk es muy grande, no se paraleliza del todo y suele ser menos eficiente como se ve con 2048. Por lo general, dynamic es más eficiente que static. El caso de guided, es especial, es el único en el que con menor chunk es más eficiente y tanto en el pc como en atcgrid, con guided, chunk=2 se obtiene el mejor resultado.

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

```
// gcc -O2 -fopenmp -o pmm-secuencial pmm-secuencial.c -lrt

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

// #define TIMES
// #define PRINTF_ALL

main(int argc, char **argv) {
    // 1. Lectura valores de entrada
    if(argc < 2) {
        fprintf(stderr, "Falta num\n");
        exit(-1);
    }
    int n = atoi(argv[1]);
    int i, j, k;
    struct timespec ini, fin; double transcurrido;

    // 2. Creación e inicialización de vector y matriz
    // 2.1. Creación
    int **A, **B, **C;
    A = (int**) malloc(n*sizeof(int*));
    for(i=0; i<n; i++)
        A[i] = (int*) malloc(n*sizeof(int));

    B = (int**) malloc(n*sizeof(int*));
    for(i=0; i<n; i++)
        B[i] = (int*) malloc(n*sizeof(int));

    C = (int**) malloc(n*sizeof(int*));
    for(i=0; i<n; i++)
        C[i] = (int*) malloc(n*sizeof(int));

    // 2.2. Inicialización
    for(i=0; i<n; i++){
        for(j=0; j<n; j++){
            B[i][j] = n*i+j;
            C[i][j] = n*i+j;
        }
    }

    // 3. Impresión de vector y matriz
    #ifndef TIMES
    #ifdef PRINTF_ALL
```

```

printf("Matriz inicial B:\n");
for (i=0; i<n; i++) {
    for (j=0; j<n; j++) {
        if(B[i][j]<10) printf(" %d ",B[i][j]);
        else printf("%d ",B[i][j]);
    }
    printf("\n");
}
printf("Matriz inicial C:\n");
for (i=0; i<n; i++) {
    for (j=0; j<n; j++) {
        if(C[i][j]<10) printf(" %d ",C[i][j]);
        else printf("%d ",C[i][j]);
    }
    printf("\n");
}
#endif
#endif

// 4. Cálculo resultado
clock_gettime(CLOCK_REALTIME,&ini);

for (i=0; i<n; i++) {
    for (j=0; j<n; j++) {
        A[i][j]=0;
        for (k=0; k<n; k++) {
            A[i][j]+=B[i][k]*C[k][j];
        }
    }
}

clock_gettime(CLOCK_REALTIME,&fin);
transcurrido=(double) (fin.tv_sec-ini.tv_sec)+(double) ((fin.tv_nsec-
ini.tv_nsec)/(1.e+9));

// 5. Impresión de vector resultado
#ifdef TIMES
    printf("%d %11.9f\n",n,transcurrido);
#else
    #ifdef PRINTF_ALL
        printf("Tiempo: %11.9f\n",transcurrido);
        printf("Matriz resultado A=B*C:\n");
        for (i=0; i<n; i++) {
            for (j=0; j<n; j++) {
                if(A[i][j]<10) printf(" %d ",A[i][j]);
                else printf("%d ",A[i][j]);
            }
            printf("\n");
        }
    #else
        printf("Tiempo: %11.9f\n",transcurrido);
        printf("A[0][0]: %d, A[n-1][n-1]: %d\n",A[0][0],A[n-1][n-1]);
    #endif
#endif

// 6. Eliminar de memoria
free(A);
free(B);
free(C);
}

```

CAPTURAS DE PANTALLA:

```
fblupi@fblupi-ElementaryOS:~/Dropbox/FACULTAD/Grado Informatica/2o Curso/2o Cuatrimestre/AC/Practicas/P3$ ./pmm-secuencial 3
Tiempo: 0.000000757
A[0][0]: 15, A[n-1][n-1]: 111
```

```
fblupi@fblupi-ElementaryOS:~/Dropbox/FACULTAD/Grado Informatica/2o Curso/2o Cuatrimestre/AC/Practicas/P3$ ./pmm-secuencial 100
Tiempo: 0.003205229
A[0][0]: 32835000, A[n-1][n-1]: 736867754
```

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

```
// gcc -O2 -fopenmp -o pmm-OpenMP pmm-OpenMP.c -lrt

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

// #define TIMES
// #define PRINTF_ALL

main(int argc, char **argv) {
    // 1. Lectura valores de entrada
    if(argc < 2) {
        fprintf(stderr, "Falta num\n");
        exit(-1);
    }
    int n = atoi(argv[1]);
    int i, j, k;
    struct timespec ini, fin; double transcurrido;

    // 2. Creación e inicialización de vector y matriz
    // 2.1. Creación
    int **A, **B, **C;
    A = (int**) malloc(n*sizeof(int*));
    for(i=0; i<n; i++)
        A[i] = (int*) malloc(n*sizeof(int));

    B = (int**) malloc(n*sizeof(int*));
    for(i=0; i<n; i++)
        B[i] = (int*) malloc(n*sizeof(int));

    C = (int**) malloc(n*sizeof(int*));
    for(i=0; i<n; i++)
        C[i] = (int*) malloc(n*sizeof(int));

    // 2.2. Inicialización
    #pragma omp parallel for default(none) private(i, j) shared(n, B, C)
    for(i=0; i<n; i++){
        for(j=0; j<n; j++){
            B[i][j] = n*i+j;
        }
    }
```

```

        C[i][j]=n*i+j;
    }
}

// 3. Impresión de vector y matriz
#ifndef TIMES
#ifdef PRINTF_ALL
    printf("Matriz inicial B:\n");
    for (i=0; i<n; i++) {
        for (j=0; j<n; j++) {
            if(B[i][j]<10) printf(" %d ",B[i][j]);
            else printf("%d ",B[i][j]);
        }
        printf("\n");
    }
    printf("Matriz inicial C:\n");
    for (i=0; i<n; i++) {
        for (j=0; j<n; j++) {
            if(C[i][j]<10) printf(" %d ",C[i][j]);
            else printf("%d ",C[i][j]);
        }
        printf("\n");
    }
#endif
#endif

// 4. Cálculo resultado
clock_gettime(CLOCK_REALTIME,&ini);

#pragma omp parallel for default(none) private(i,j,k) shared(n,A,B,C)
for (i=0; i<n; i++) {
    for (j=0; j<n; j++) {
        A[i][j]=0;
        for (k=0; k<n; k++) {
            A[i][j]+=B[i][k]*C[k][j];
        }
    }
}

clock_gettime(CLOCK_REALTIME,&fin);
transcurrido=(double) (fin.tv_sec-ini.tv_sec)+(double) ((fin.tv_nsec-
ini.tv_nsec)/(1.e+9));

// 5. Impresión de vector resultado
#ifdef TIMES
    printf("%d %11.9f\n",n,transcurrido);
#else
#ifdef PRINTF_ALL
    printf("Tiempo: %11.9f\n",transcurrido);
    printf("Matriz resultado A=B*C:\n");
    for (i=0; i<n; i++) {
        for (j=0; j<n; j++) {
            if(A[i][j]<10) printf(" %d ",A[i][j]);
            else printf("%d ",A[i][j]);
        }
        printf("\n");
    }
#else
    printf("Tiempo: %11.9f\n",transcurrido);
    printf("A[0][0]: %d, A[n-1][n-1]: %d\n",A[0][0],A[n-1][n-1]);
#endif
#endif

```

```
#endif

// 6. Eliminar de memoria
free(A);
free(B);
free(C);
}
```

RESPUESTA:

$$\begin{pmatrix} A_{00} & A_{01} & A_{02} \\ A_{10} & A_{11} & A_{12} \\ A_{20} & A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{00} & B_{01} & B_{02} \\ B_{10} & B_{11} & B_{12} \\ B_{20} & B_{21} & B_{22} \end{pmatrix}$$

$$\begin{array}{l} \text{thread0} \rightarrow A_{00} \cdot B_{00} + A_{01} \cdot B_{10} + A_{02} \cdot B_{20}, A_{00} \cdot B_{01} + A_{01} \cdot B_{11} + A_{02} \cdot B_{21}, A_{00} \cdot B_{02} + A_{01} \cdot B_{12} + A_{02} \cdot B_{22} \\ \text{thread1} \rightarrow A_{10} \cdot B_{00} + A_{11} \cdot B_{10} + A_{12} \cdot B_{20}, A_{10} \cdot B_{01} + A_{11} \cdot B_{11} + A_{12} \cdot B_{21}, A_{10} \cdot B_{02} + A_{11} \cdot B_{12} + A_{12} \cdot B_{22} \\ \text{thread2} \rightarrow A_{20} \cdot B_{00} + A_{21} \cdot B_{10} + A_{22} \cdot B_{20}, A_{20} \cdot B_{01} + A_{21} \cdot B_{11} + A_{22} \cdot B_{21}, A_{20} \cdot B_{02} + A_{21} \cdot B_{12} + A_{22} \cdot B_{22} \end{array}$$

CAPTURAS DE PANTALLA:

```
fblupi@fblupi-ElementaryOS:~/Dropbox/FACULTAD/Grado Informatica/2o Curso/2o Cuatrimestre/AC/Practicas/P3$ ./pmm-OpenMP 3
Tiempo: 0.003991877
A[0][0]: 15, A[n-1][n-1]: 111
```

```
fblupi@fblupi-ElementaryOS:~/Dropbox/FACULTAD/Grado Informatica/2o Curso/2o Cuatrimestre/AC/Practicas/P3$ ./pmm-OpenMP 100
Tiempo: 0.001338680
A[0][0]: 32835000, A[n-1][n-1]: 736867754
```

10. Hacer un estudio de escalabilidad (ganancia en velocidad en función del número de cores) en atcgrid y en el PC local del código paralelo implementado para tres tamaños de las matrices (N = 10, 100 y 500). 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. Consulte la Lección 6/Tema 2.

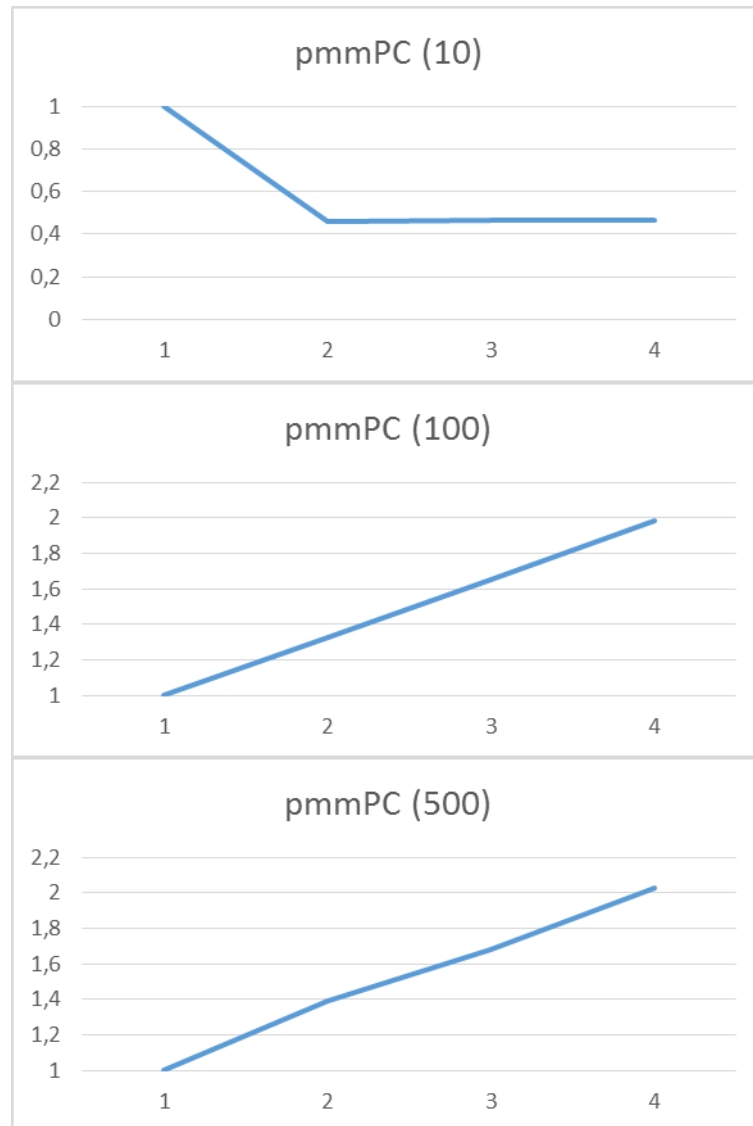
PC LOCAL**1. TIEMPOS**

	cores			
tama	1 core	2 cores	3 cores	4 cores
10	7,427E-06	1,6099E-05	1,5909E-05	1,5963E-05
100	0,00427313	0,00321766	0,00258787	0,00215079
500	0,44661713	0,32147728	0,26523091	0,22039197

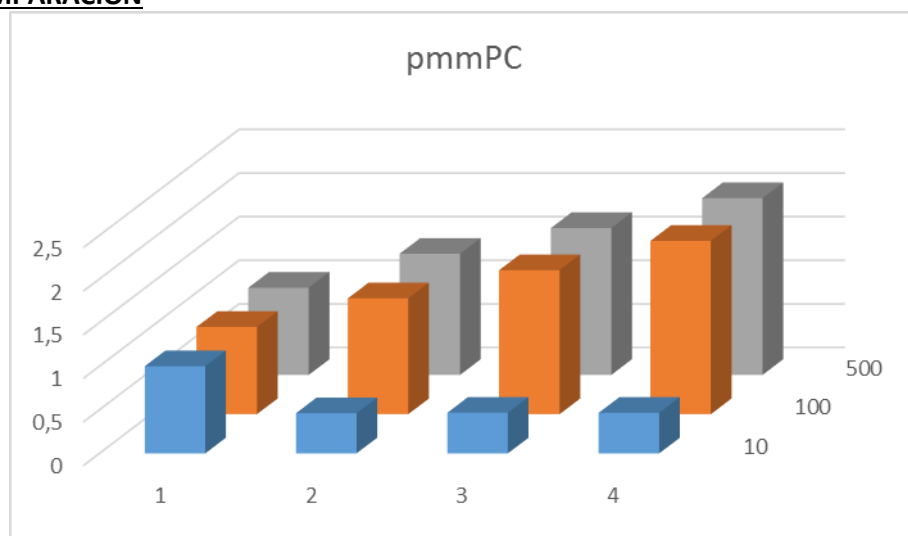
2. GANANCIAS

	cores			
tama	1 core	2 cores	3 cores	4 cores
10	1	0,461333	0,46684267	0,46526342
100	1	1,32802512	1,65121623	1,98677045
500	1	1,38926501	1,6838804	2,02646736

3. GRÁFICAS



4. COMPARACIÓN



ATCGRID

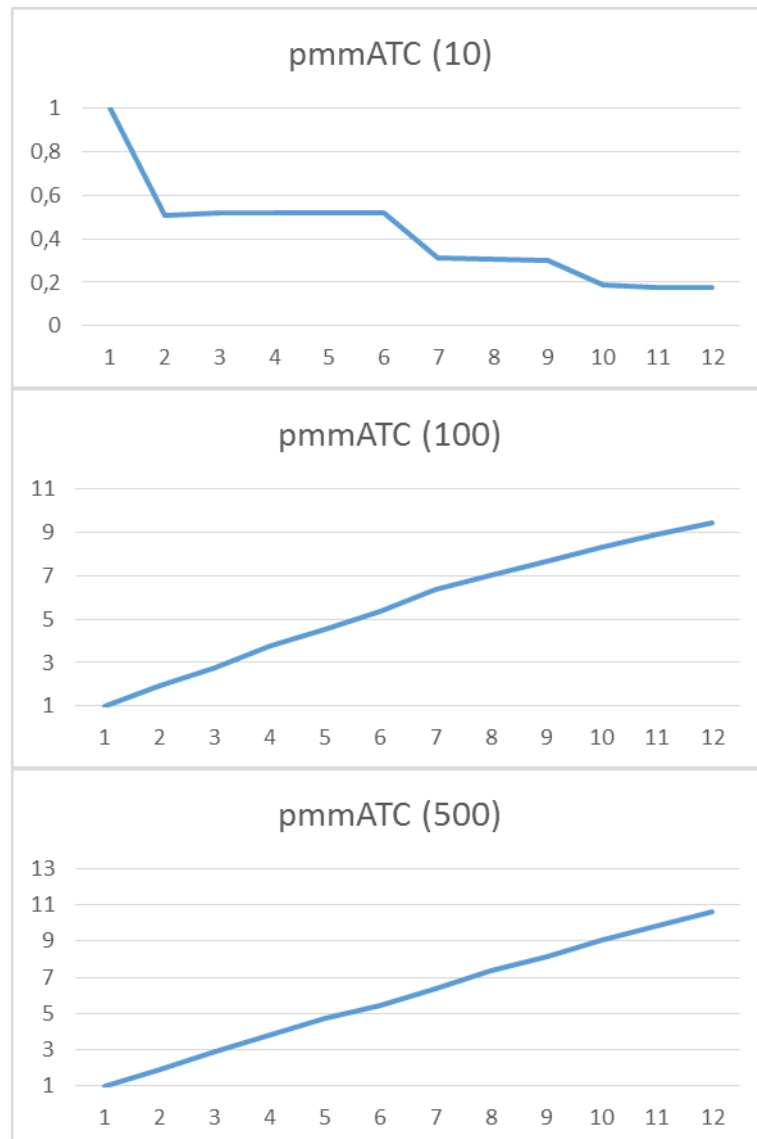
1. TIEMPOS

	cores											
tama	1 core	2 cores	3 cores	4 cores	5 cores	6 cores	7 cores	8 cores	9 cores	10 cores	11 cores	12 cores
10	4E-06	7,8E-06	7,7E-06	7,6E-06	7,6E-06	7,7E-06	1,3E-05	1,3E-05	1,3E-05	2,1E-05	2,2E-05	2,3E-05
100	0,00278	0,00143	0,001	0,00073	0,00061	0,00052	0,00044	0,0004	0,00036	0,00033	0,00031	0,0003
500	0,36938	0,19386	0,1271	0,09657	0,07797	0,06737	0,05788	0,05043	0,04523	0,04074	0,03751	0,03487

2. GANANCIAS

	cores											
tama	1 core	2 cores	3 cores	4 cores	5 cores	6 cores	7 cores	8 cores	9 cores	10 cores	11 cores	12 cores
10	1	0,50838	0,51558	0,52044	0,5203	0,51732	0,30978	0,30738	0,29977	0,18609	0,17707	0,17638
100	1	1,94815	2,78592	3,79088	4,53059	5,34804	6,33284	6,99721	7,66424	8,31212	8,91884	9,40517
500	1	1,90542	2,90619	3,82512	4,73719	5,48326	6,38143	7,32463	8,16698	9,06604	9,84739	10,5939

3. GRÁFICAS



4. COMPARACIÓN

