

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): Cristóbal López Peñalver

Grupo de prácticas: B1

Fecha de entrega:

Fecha evaluación en clase:

**[-RECORDATORIO, quitar todo este texto en rojo del cuaderno definitivo-]**

### **1. COMENTARIO**

Este cuaderno de prácticas se utilizará para asignarle una puntuación durante la evaluación continua de prácticas y también lo utilizará como material de estudio y repaso para preparar el examen de prácticas escrito. Luego redáctelo con cuidado, y sea ordenado y claro.

### **2. NORMAS SOBRE EL USO DE LA PLANTILLA**

1) Usar **interlineado SENCILLO**.

2) Respetar los tipos de letra y tamaños indicados:

- Calibri-11 o Liberation Serif-11 para el texto

- Courier New-10 o Liberation Mono-10 para nombres de fichero, comandos, variables de entorno, etc., cuando se usan en el texto.

- Courier New o Liberation Mono de tamaño 8 o 9 para el código fuente en los listados de código fuente.

- Formatee el código fuente de los listados para que sea legible, limpio y claro. Consulte, como ejemplo, los Listados 1 y 2 del guion (tabule, comente, ...)

---

3) Insertar las capturas de pantalla donde se pidan y donde se considere oportuno

Recuerde que debe **adjuntar al zip de entrega, el pdf de este fichero, todos los ficheros con código fuente implementados/utilizados y el resto de ficheros que haya implementado/utilizado (scripts, hojas de cálculo, etc.), lea la Sección 1.4 del guion]**

---

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]-Introduce número de
iteraciones y de hebras\n");
        exit(-1);
    }
    n = atoi(argv[1]);

    if (n>20) n=20;
    for (i=0; i<n; i++) {
        a[i] = i;
    }
    #pragma omp parallel num_threads(atoi(argv[2])) if(n>4)
    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:

```

crisobal@PG6:~/Escritorio/COLE xd/2.2/AC/Prácticas/p3$ ./if-clause 5 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 master=0 imprime suma=10
crisobal@PG6:~/Escritorio/COLE xd/2.2/AC/Prácticas/p3$ ./if-clause 6 6
thread 3 suma de a[3]=3 sumalocal=3
thread 4 suma de a[4]=4 sumalocal=4
thread 1 suma de a[1]=1 sumalocal=1
thread 0 suma de a[0]=0 sumalocal=0
thread 5 suma de a[5]=5 sumalocal=5
thread 2 suma de a[2]=2 sumalocal=2
thread master=0 imprime suma=15

```

**RESPUESTA:** Esta cláusula fija el número de hebras a usar en tiempo de ejecución.

2. (a) Rellenar la Tabla 1 (se debe poner en la tabla el id del *thread* que ejecuta cada iteración) ejecutando los ejemplos del seminario `schedule-clause.c`, `scheduled-clause.c` y `scheduleg-clause.c` con dos *threads* (0,1) y unas entradas de:

- iteraciones: 16 (0,...15)
- chunk= 1, 2 y 4

**Tabla 1.** Tabla `schedule`. En la segunda fila, 1, 2 4 representan el tamaño del chunk (consulte seminario)

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

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

**RESPUESTA:**

Static es el único que asigna el reparto de hebras de forma estática.

En dynamic repartimos las tareas entre las hebras en “chunk” iteraciones. Conforme van acabando sus tareas, las hebras eligen el siguiente paquete de iteraciones.

Guided es parecido a dynamic con la diferencia de que los paquetes de iteraciones a repartir entre hebras van decreciendo con la ejecución.

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`

Adjunto al fichero.
---------------------

**CAPTURAS DE PANTALLA:**

```
cristobal@PG6: ~/Escritorio/COLE xd/2.2/AC/Prácticas/p3
cristobal@PG6:~/Escritorio/COLE xd/2.2/AC/Prácticas/p3$ export OMP_DYNAMIC=
FALSE; export OMP_NUM_THREADS=8; ./schedule-clause-ejercicio2 8 1
thread 1 suma a[2]=2 suma=2
thread 1 suma a[7]=7 suma=9
thread 0 suma a[0]=0 suma=0
thread 5 suma a[3]=3 suma=3
thread 3 suma a[4]=4 suma=4
thread 2 suma a[5]=5 suma=5
thread 4 suma a[6]=6 suma=6
thread 7 suma a[1]=1 suma=1
Dentro de la region paralela:
dyn-var: 0
nthreads-var:8
thread-limit-var:50
Fuera de la region paralela:
dyn-var: 0
nthreads-var:8
thread-limit-var:50
Fuera de 'parallel for' suma=9
cristobal@PG6:~/Escritorio/COLE xd/2.2/AC/Prácticas/p3$ export OMP_DYNAMIC=
TRUE; export OMP_THREAD_LIMIT=2; ./schedule-clause-ejercicio2 8 1
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
thread 0 suma a[5]=5 suma=15
thread 0 suma a[6]=6 suma=21
thread 0 suma a[7]=7 suma=28
thread 1 suma a[0]=0 suma=0
Dentro de la region paralela:
dyn-var: 1
nthreads-var:8
thread-limit-var:2
Fuera de la region paralela:
dyn-var: 1
nthreads-var:8
thread-limit-var:2
Fuera de 'parallel for' suma=28
cristobal@PG6:~/Escritorio/COLE xd/2.2/AC/Prácticas/p3$ _
```

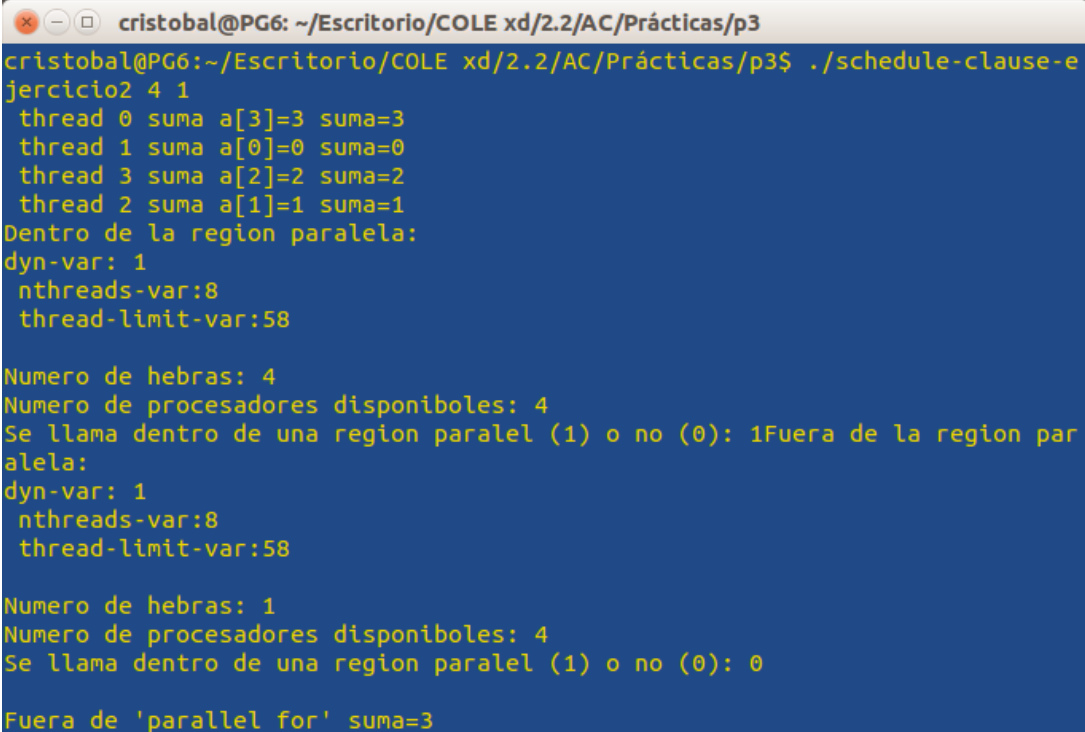
**RESPUESTA:** No se imprimen valores distintos.

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`

Adjunto código.

**CAPTURAS DE PANTALLA:**



```
cristobal@PG6: ~/Escritorio/COLE xd/2.2/AC/Prácticas/p3$ ./schedule-clause-ejercicio2 4 1
thread 0 suma a[3]=3 suma=3
thread 1 suma a[0]=0 suma=0
thread 3 suma a[2]=2 suma=2
thread 2 suma a[1]=1 suma=1
Dentro de la region paralela:
dyn-var: 1
nthreads-var:8
thread-limit-var:58

Numero de hebras: 4
Numero de procesadores disponibles: 4
Se llama dentro de una region paralel (1) o no (0): 1Fuera de la region paralela:
dyn-var: 1
nthreads-var:8
thread-limit-var:58

Numero de hebras: 1
Numero de procesadores disponibles: 4
Se llama dentro de una region paralel (1) o no (0): 0

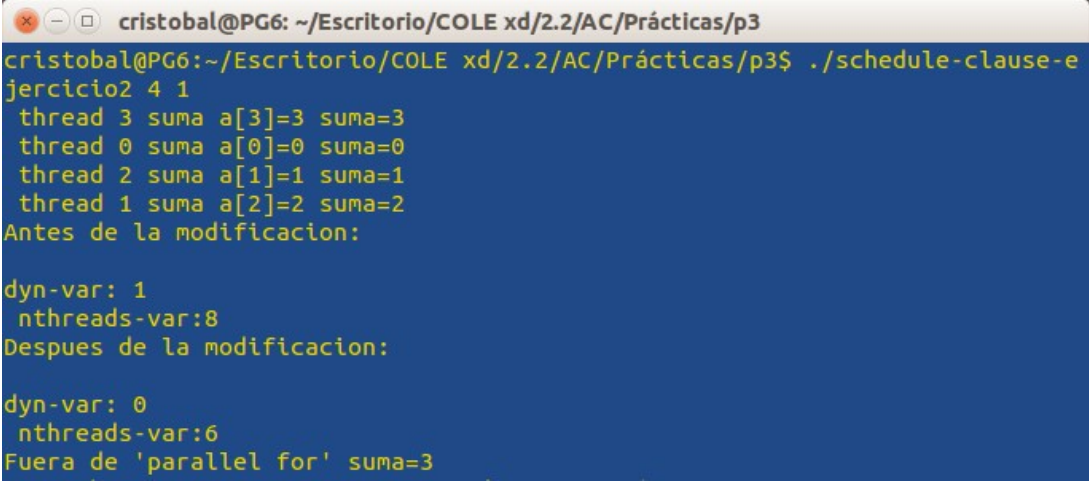
Fuera de 'parallel for' suma=3
```

**RESPUESTA:** En la función que nos dice el número de hebras, ya que la región paralela no es secuencial, y en la que nos dice si estamos en una región paralela activa por razones obvias.

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`

**CAPTURAS DE PANTALLA:**



```

crisobal@PG6: ~/Escritorio/COLE xd/2.2/AC/Prácticas/p3
crisobal@PG6:~/Escritorio/COLE xd/2.2/AC/Prácticas/p3$ ./schedule-clause-ejercicio2 4 1
thread 3 suma a[3]=3 suma=3
thread 0 suma a[0]=0 suma=0
thread 2 suma a[1]=1 suma=1
thread 1 suma a[2]=2 suma=2
Antes de la modificacion:
dyn-var: 1
nthreads-var:8
Despues de la modificacion:
dyn-var: 0
nthreads-var:6
Fuera de 'parallel for' suma=3

```

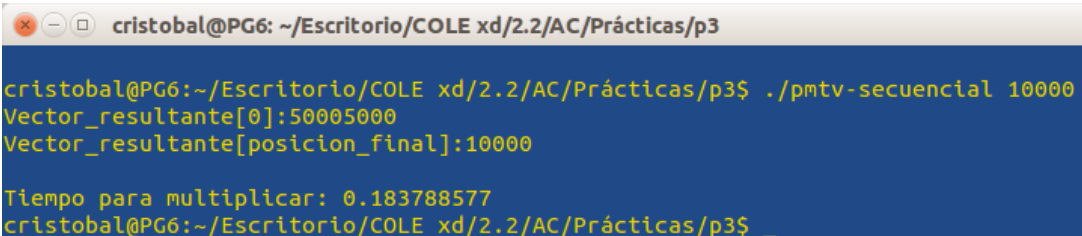
6. Implementar un programa secuencial en C que multiplique una matriz triangular por un vector.

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`

Código adjunto.

**CAPTURAS DE PANTALLA:**  
(ADJUNTAR CÓDIGO FUENTE AL .ZIP)



```

crisobal@PG6: ~/Escritorio/COLE xd/2.2/AC/Prácticas/p3
crisobal@PG6:~/Escritorio/COLE xd/2.2/AC/Prácticas/p3$ ./pmtv-secuencial 10000
Vector_resultante[0]:50005000
Vector_resultante[posicion_final]:10000

Tiempo para multiplicar: 0.183788577
crisobal@PG6:~/Escritorio/COLE xd/2.2/AC/Prácticas/p3$ _

```



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.

**RESPUESTA:**

**CÓDIGO FUENTE:** `pmtv-OpenMP.c`

Adjunto.

**DESCOMPOSICIÓN DE DOMINIO:**

**CAPTURAS DE PANTALLA:**

**(ADJUNTAR CÓDIGO FUENTE AL .ZIP)**

### TABLA RESULTADOS Y GRÁFICA ATCGRID

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

Chunk	Static 12 threads	Dynamic 12 threads	Guided 12 threads
por defecto			
2	0.020081042	0.038508211	0.041540336
32	0.020247244	0.014197913	0.014185730
64	0.014465768	0.026250261	0.023376236
1024	0.046697455	0.065819935	0.046682854

### TABLA RESULTADOS Y GRÁFICA PC LOCAL

**Tabla 4** .Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones paralelas

Chunk	Static 12 threads	Dynamic 12 threads	Guided 12 threads
<b>por defecto</b>			
2	0.042070649	0.046579747	0.038152699
32	0.036410621	0.014185730	0.042726719
64	0.041584157	0.023376236	0.051395071
1024	0.051453363	0.046682854	0.043400844

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>
#include <omp.h>

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

}

**CAPTURAS DE PANTALLA:**  
**(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).

10.

**DESCOMPOSICIÓN DE DOMINIO:**

**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>
#include <omp.h>

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

**CAPTURAS DE PANTALLA:**  
**(ADJUNTAR CÓDIGO FUENTE AL .ZIP)**

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. Debe recordar usar -O2 al compilar. 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.

**ESTUDIO DE ESCALABILIDAD EN ATCGRID:**

**ESTUDIO DE ESCALABILIDAD EN PCLOCAL:**