

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): Elena Cantero Molina

Grupo de prácticas: E2

Fecha de entrega:

Fecha evaluación en clase:

#### 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`

```
/* 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, x, n=20, tid;
    int a[n], suma=0, sumalocal;
    if(argc < 2) {
        fprintf(stderr, "[ERROR]-Falta iteraciones\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:**

Captura de pantalla 1:

```

usuario@asus:~/Escritorio/2.Practica 3$ gcc -fopenmp -O2 -o ifclause if-clauseModificado.c
usuario@asus:~/Escritorio/2.Practica 3$ ./ifclause 4 8
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 master=0 imprime suma=6

```

Captura de pantalla 2:

```

usuario@asus:~/Escritorio/2.Practica 3$ ./ifclause 5 8
thread 1 suma de a[1]=1 sumalocal=1
thread 0 suma de a[0]=0 sumalocal=0
thread 3 suma de a[3]=3 sumalocal=3
thread 2 suma de a[2]=2 sumalocal=2
thread 4 suma de a[4]=4 sumalocal=4
thread master=0 imprime suma=10
usuario@asus:~/Escritorio/2.Practica 3$ ./ifclause 5 5
thread 0 suma de a[0]=0 sumalocal=0
thread 2 suma de a[2]=2 sumalocal=2
thread 1 suma de a[1]=1 sumalocal=1
thread 3 suma de a[3]=3 sumalocal=3
thread 4 suma de a[4]=4 sumalocal=4
thread master=0 imprime suma=10
usuario@asus:~/Escritorio/2.Practica 3$ ./ifclause 5 4
thread 0 suma de a[0]=0 sumalocal=0
thread 0 suma de a[1]=1 sumalocal=1
thread 3 suma de a[4]=4 sumalocal=4
thread 1 suma de a[2]=2 sumalocal=2
thread 2 suma de a[3]=3 sumalocal=3
thread master=0 imprime suma=10

```

**RESPUESTA:**

En la captura de pantalla 1 podemos ver que como el número de iteraciones no supera a 4, la región paralela ejecuta un único thread. Sin embargo, en la captura de pantalla 2 sí se realiza la ejecución en paralelo al haber los threads suficientes y podemos fijar de distintas maneras el número de hebras que realizan la tarea.

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	1	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0
2	0	1	0	0	1	0	0	0	0
3	1	1	0	0	1	0	0	0	0

4	0	0	1	0	0	1	0	0	0
5	1	0	1	0	0	1	0	0	0
6	0	1	1	0	0	1	0	0	0
7	1	1	1	0	0	1	0	0	0
8	0	0	0	0	0	1	1	1	1
9	1	0	0	0	0	1	1	1	1
10	0	1	0	0	0	1	1	1	1
11	1	1	0	0	0	1	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	0	0	0	0	0
15	1	1	1	0	0	0	0	0	0

(b) Rellenar otra tabla como la de la figura pero esta vez usando cuatro *threads* (0,1,2,3).

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

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

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

#### RESPUESTA:

Con `static`, las iteraciones del bucle se dividen en tantas unidades como chunks haya antes de que se comience a realizar la región paralela, y cada una de esas unidades se asignan en round-robin. Además cuando usamos `schedule(static)` se asigna un único chunk a cada thread.

Con `dynamic`, la distribución de las iteraciones del bucle se realizan en tiempo de ejecución, pero siempre se asigna el mismo tamaño de bloque.

Con `guided`, se empieza asignando grupos de iteraciones grandes y a medida que disminuyen el

número de threads, también lo hace el número de iteraciones, pero nunca disminuye el número de threads por debajo del chunk. Todo esto supone menos sobrecarga para las hebras con respecto a dynamic.

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`

```
/* 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

int main(int argc, char ** argv)
{
    int i, n=200, chunk, a[n], suma=0;
    omp_sched_t kind;
    int 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 for firstprivate(suma) lastprivate(suma)
    schedule(dynamic, chunk)
    for (i=0; i<n; i++)
    { suma = suma + a[i];
      printf(" thread %d suma a[%d]=%d suma=%d \n",
omp_get_thread_num(), i, a[i], suma);
      if(i == n-1){
          printf("Thread %d \n", i);
          printf("Dentro del parallel -> dyn-var = %d \n", omp_get_dynamic());
          printf("Dentro del parallel -> nthreads-var = %d \n",
omp_get_max_threads());
          printf("Dentro del parallel -> thread-limit-var = %d \n",
omp_get_thread_limit());
          omp_get_schedule(&kind, &modifier);
          printf("Dentro del parallel -> run-sched-var(kind, modifier)= %d, %d
\n", kind, modifier);
      }
    }

    printf("Fuera de 'parallel for' suma=%d\n", suma);
    printf("Fuera del parallel -> dyn-var = %d \n", omp_get_dynamic());
    printf("Fuera del parallel -> nthreads-var = %d \n", omp_get_max_threads());
```

```

printf("Fuera del parallel -> thread-limit-var = %d \n",
omp_get_thread_limit());
omp_get_schedule(&kind,&modifier);
printf("Fuera del parallel -> run-sched-var(kind, modifier)= %d, %d \n",
kind, modifier);
}

```

**CAPTURAS DE PANTALLA:**

```

usuario@asus:~/Escritorio/2.Practica 3$ export OMP_SCHEDULE="static,1"
usuario@asus:~/Escritorio/2.Practica 3$ export OMP_DYNAMIC=false
usuario@asus:~/Escritorio/2.Practica 3$ export OMP_THREAD_LIMIT=8
usuario@asus:~/Escritorio/2.Practica 3$ export OMP_NUM_THREADS=8
usuario@asus:~/Escritorio/2.Practica 3$ ./scheduledM 7 10
thread 1 suma a[0]=0 suma=0
thread 1 suma a[1]=1 suma=1
thread 1 suma a[2]=2 suma=3
thread 1 suma a[3]=3 suma=6
thread 1 suma a[4]=4 suma=10
thread 1 suma a[5]=5 suma=15
thread 1 suma a[6]=6 suma=21
Thread 6
Dentro del parallel -> dyn-var = 0
Dentro del parallel -> nthreads-var = 8
Dentro del parallel -> thread-limit-var = 8
Dentro del parallel -> run-sched-var(kind, modifier)= 1, 1
Fuera de 'parallel for' suma=21
Fuera del parallel -> dyn-var = 0
Fuera del parallel -> nthreads-var = 8
Fuera del parallel -> thread-limit-var = 8
Fuera del parallel -> run-sched-var(kind, modifier)= 1, 1

```

```

usuario@asus:~/Escritorio/2.Practica 3$ export OMP_SCHEDULE="guided,2"
usuario@asus:~/Escritorio/2.Practica 3$ export OMP_DYNAMIC=true
usuario@asus:~/Escritorio/2.Practica 3$ export OMP_NUM_THREADS=8
usuario@asus:~/Escritorio/2.Practica 3$ export OMP_THREAD_LIMIT=4
usuario@asus:~/Escritorio/2.Practica 3$ ./scheduledM 7 10
thread 1 suma a[0]=0 suma=0
thread 1 suma a[1]=1 suma=1
thread 1 suma a[2]=2 suma=3
thread 1 suma a[3]=3 suma=6
thread 1 suma a[4]=4 suma=10
thread 1 suma a[5]=5 suma=15
thread 1 suma a[6]=6 suma=21
Thread 6
Dentro del parallel -> dyn-var = 1
Dentro del parallel -> nthreads-var = 8
Dentro del parallel -> thread-limit-var = 4
Dentro del parallel -> run-sched-var(kind, modifier)= 3, 2
Fuera de 'parallel for' suma=21
Fuera del parallel -> dyn-var = 1
Fuera del parallel -> nthreads-var = 8
Fuera del parallel -> thread-limit-var = 4
Fuera del parallel -> run-sched-var(kind, modifier)= 3, 2

```

**RESPUESTA:** Como vemos en las dos ejecuciones de las capturas de pantalla, se imprime el mismo resultado dentro y fuera de la región paralela.

- 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

```
/* 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

int main(int argc, char **argv) {
    int i, n=8, chunk, a[n], suma=0;
    omp_sched_t kind;
    int modifier;

    if(argc < 2) {
        fprintf(stderr, "\nFalta chunk \n");
        exit(-1);
    }
    chunk = atoi(argv[1]);

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

    #pragma omp parallel for firstprivate(suma) lastprivate(suma)
    schedule(dynamic, chunk)
    for (i=0; i<n; i++)
    {
        suma = suma + a[i];
        printf(" thread %d suma a[%d]=%d suma=%d \n",
omp_get_thread_num(), i, a[i], suma);
        if(i == n-1){
            printf("Thread %d \n", i);
            printf("Dentro del parallel -> dyn-var = %d \n", omp_get_dynamic());
            printf("Dentro del parallel -> nthreads-var = %d \n",
omp_get_max_threads());
            printf("Dentro del parallel -> thread-limit-var = %d \n",
omp_get_thread_limit());
            omp_get_schedule(&kind, &modifier);
            printf("Dentro del parallel -> run-sched-var(kind, modifier)= %d, %d
\n", kind, modifier);
            printf("Dentro del parallel -> omp_get_num_threads = %d \n",
omp_get_num_threads());
            printf("Dentro del parallel -> omp_get_num_procs = %d \n",
omp_get_num_procs());
            printf("Dentro del parallel -> omp_in_parallel = %d \n",
omp_in_parallel());
        }
    }

    printf("Fuera de 'parallel for' suma=%d\n", suma);
    printf("Fuera del parallel -> dyn-var = %d \n", omp_get_dynamic());
    printf("Fuera del parallel -> nthreads-var = %d \n", omp_get_max_threads());
    printf("Fuera del parallel -> thread-limit-var = %d \n",
omp_get_thread_limit());
```

```

omp_get_schedule(&kind,&modifier);
printf("Fuera del parallel -> run-sched-var(kind, modifier)= %d, %d \n",
kind, modifier);
printf("Fuera del parallel -> omp_get_num_threads = %d \n",
omp_get_num_threads());
printf("Fuera del parallel -> omp_get_num_procs = %d \n",
omp_get_num_procs());
printf("Fuera del parallel -> omp_in_parallel = %d \n", omp_in_parallel());
}

```

**CAPTURAS DE PANTALLA:**

```

usuario@asus:~/Escritorio/2.Practica 3$ ./scheduled 10
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
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 7
Dentro del parallel -> dyn-var = 0
Dentro del parallel -> nthreads-var = 4
Dentro del parallel -> thread-limit-var = 2147483647
Dentro del parallel -> run-sched-var(kind, modifier)= 2, 1
Dentro del parallel -> omp_get_num_threads = 4
Dentro del parallel -> omp_get_num_procs = 4
Dentro del parallel -> omp_in_parallel = 1
Fuera de 'parallel for' suma=28
Fuera del parallel -> dyn-var = 0
Fuera del parallel -> nthreads-var = 4
Fuera del parallel -> thread-limit-var = 2147483647
Fuera del parallel -> run-sched-var(kind, modifier)= 2, 1
Fuera del parallel -> omp_get_num_threads = 1
Fuera del parallel -> omp_get_num_procs = 4
Fuera del parallel -> omp_in_parallel = 0
usuario@asus:~/Escritorio/2.Practica 3$ |

```

**RESPUESTA:**

Se observa que el número de procesadores disponibles para el programa en el momento de la ejecución no varía. Por otro lado, varía el número de threads que se ejecutan fuera y dentro de la región paralela. Y por último `omp_in_parallel()` dentro de la región paralela devuelve 1, mientras que fuera de ella devuelve 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

int main(int argc, char **argv) {
    int i, n=8, chunk, a[n], suma=0;
    omp_sched_t kind;
    int modifier;

    if(argc < 2) {
        fprintf(stderr, "\nFalta chunk \n");
        exit(-1);
    }
    chunk = atoi(argv[1]);

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

    #pragma omp parallel for firstprivate(suma) lastprivate(suma)
    schedule(dynamic, chunk)
    for (i=0; i<n; i++)
    {
        suma = suma + a[i];
        printf(" thread %d suma a[%d]=%d suma=%d \n",
omp_get_thread_num(), i, a[i], suma);
        /*printf("Thread %d \n", i);
        if(i == n-1){
            printf("Dentro del parallel -> dyn-var = %d \n", omp_get_dynamic());
            printf("Dentro del parallel -> nthreads-var = %d \n",
omp_get_max_threads());
            printf("Dentro del parallel -> thread-limit-var = %d \n",
omp_get_thread_limit());
            omp_get_schedule(&kind, &modifier);
            printf("Dentro del parallel -> run-sched-var(kind, modifier)= %d, %d
\n", kind, modifier);
            printf("Dentro del parallel -> get_num_threads = %d \n",
omp_get_num_threads());
            printf("Dentro del parallel -> num_procs = %d \n", omp_get_num_procs());
            printf("Dentro del parallel -> omp_in_parallel = %d \n",
omp_in_parallel());
        }*/
    }

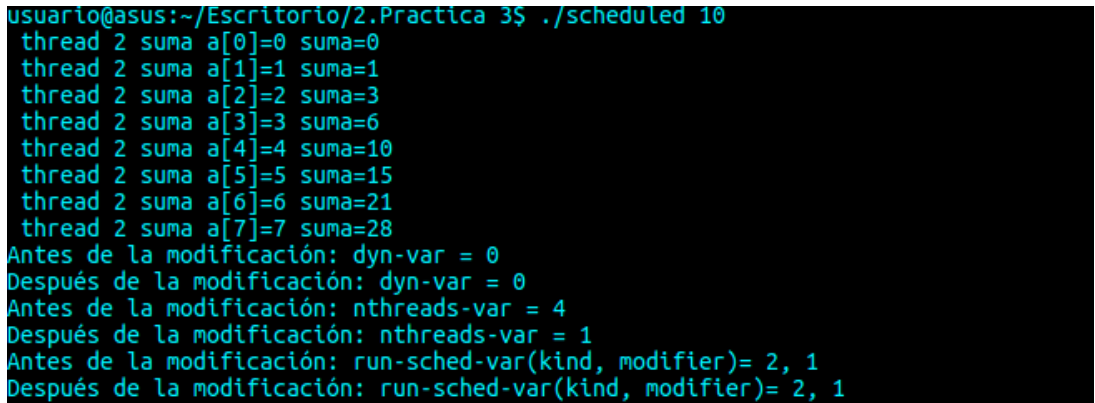
    /*printf("Fuera de 'parallel for' suma=%d\n", suma);
    printf("Fuera del parallel -> dyn-var = %d \n", omp_get_dynamic());
    printf("Fuera del parallel -> nthreads-var = %d \n", omp_get_max_threads());
    printf("Fuera del parallel -> thread-limit-var = %d \n",
omp_get_thread_limit());*/
    omp_get_schedule(&kind, &modifier);
    /*printf("Fuera del parallel -> run-sched-var(kind, modifier)= %d, %d \n",
kind, modifier);
    printf("Fuera del parallel -> num_threads = %d \n", omp_get_num_threads());
    printf("Fuera del parallel -> num_procs = %d \n", omp_get_num_procs());
    printf("Fuera del parallel -> omp_in_parallel = %d \n",
omp_in_parallel());*/

    printf("Antes de la modificación: dyn-var = %d \n", omp_get_dynamic());
    omp_set_dynamic(0);
    printf("Después de la modificación: dyn-var = %d \n", omp_get_dynamic());
    printf("Antes de la modificación: nthreads-var = %d \n",
omp_get_max_threads());
    omp_set_num_threads(0);
    printf("Después de la modificación: nthreads-var = %d \n",

```



```
omp_get_max_threads());
printf("Antes de la modificación: run-sched-var(kind, modifier)= %d, %d \n",
kind, modifier);
omp_set_schedule(omp_sched_guided,1);
printf("Después de la modificación: run-sched-var(kind, modifier)= %d, %d
\n", kind, modifier);
}
```

**CAPTURAS DE PANTALLA:**


```
usuario@asus:~/Escritorio/2.Practica 3$ ./scheduled 10
thread 2 suma a[0]=0 suma=0
thread 2 suma a[1]=1 suma=1
thread 2 suma a[2]=2 suma=3
thread 2 suma a[3]=3 suma=6
thread 2 suma a[4]=4 suma=10
thread 2 suma a[5]=5 suma=15
thread 2 suma a[6]=6 suma=21
thread 2 suma a[7]=7 suma=28
Antes de la modificación: dyn-var = 0
Después de la modificación: dyn-var = 0
Antes de la modificación: nthreads-var = 4
Después de la modificación: nthreads-var = 1
Antes de la modificación: run-sched-var(kind, modifier)= 2, 1
Después de la modificación: run-sched-var(kind, modifier)= 2, 1
```

**RESPUESTA:** Podemos observar cómo cambia la variable nthreads, pero run-sched-var aparece igual.

**Resto de ejercicios**

**6.** Implementar un programa secuencial en C que multiplique una matriz triangular por un vector (use variables dinámicas). Compare el orden de complejidad del código que ha implementado con el código que implementó para el producto matriz por vector.

NOTAS: (1) el número de filas/columnas debe ser un argumento de entrada; (2) se debe inicializar las matrices antes del cálculo; (3) se debe imprimir siempre la primera y última componente del resultado antes de que termine el programa.

**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>
#include <omp.h>
#include <time.h>

#define VECTOR_DYNAMIC

int main(int argc, char const *argv[])
{
    // comprobamos que los argumentos son correctos
    if (argc < 2) {
        printf("Falta el tamaño de la matriz\n");
        exit(-1);
    }

    int N = atoi(argv[1]);
    int i, j;

    int *v, **matriz;
```

```

v = (int *) malloc(N*sizeof(double)); // malloc necesita el tamaño en bytes
matriz = (int **) malloc(N*sizeof(int *));
for(i=0; i<N; i++)
    matriz[i] = (int *) malloc(N*sizeof(int));

if ( (v==NULL) || (matriz==NULL) ){
    printf("Error en la reserva de espacio para los vectores\n");
    exit(-2);
}

// inicializamos la matriz y el vector
for (i = 0; i<N; i++) {
    for (j = 0; j<i+1; j++)
        matriz[i][j] = i;
    v[i] = i+1;
}

struct timespec cgt1,cgt2; double ncgt; //para tiempo de ejecución
int vector_resultado[N], k = 0, suma=0;

clock_gettime(CLOCK_REALTIME, &cgt1);

// Calculamos el producto
for (i=0; i<N;i++) {
    // calculamos la suma de la columna de la matriz:
    for (k=0; k<N; k++)
        suma+=matriz[k][i];

    // y ahora multiplicamos la suma por el vector:
    vector_resultado[i] = suma * v[i];
    suma = 0;
}

...
}

```

**CAPTURAS DE PANTALLA:**

```

usuario@asus:~/Escritorio/2.Practica 3$ gcc -fopenmp -O2 -o pmtv pmtv-secuencial.c
usuario@asus:~/Escritorio/2.Practica 3$ ./pmtv 8
Tiempo: 0.000000283
v[0] = 28
v[1] = 56
v[2] = 81
v[3] = 100
v[4] = 110
v[5] = 108
v[6] = 91
v[7] = 56
usuario@asus:~/Escritorio/2.Practica 3$ |

```

**(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. El código debe repartir entre los threads las iteraciones del bucle que recorre las filas. Dibujar en el cuaderno de prácticas la descomposición de dominio utilizada (Lección 4/Tema 2) en el código paralelo implementado para asignar tareas a los threads (Lección 5/Tema 2). Añadir lo necesario para que el usuario pueda fijar la planificación de tareas usando la variable de entorno `OMP_SCHEDULE`. Obtener en `atcgrid` los tiempos de ejecución del código paralelo (usando, como siempre, `-O2` al compilar) que multiplica una matriz triangular por un vector con las alternativas de planificación `static`, `dynamic` y `guided` para `chunk` de 1, 64 y el `chunk` por defecto

para la alternativa. Use un tamaño de vector  $N$  múltiplo del número de cores y de 64 que no sea inferior a 15360. El número de threads en las ejecuciones debe coincidir con el número de cores. Rellenar la Tabla 3 dos veces con los tiempos obtenidos. Representar el tiempo para static, dynamic y guided en función del tamaño del chunk en una gráfica. ¿Qué alternativa ofrece mejores prestaciones? Razone por qué. Incluya los scripts utilizado en el cuaderno de prácticas. NOTA: Nunca ejecute en atcgrid código que imprima todos los componentes del resultado.

Conteste a las siguientes preguntas: (a) ¿Qué valor por defecto usa OpenMP para chunk con static, dynamic y guided? Indique qué ha hecho para obtener este valor por defecto para cada alternativa. (b) ¿Qué número de operaciones de multiplicación y suma realizan cada uno de los threads en la asignación static para cada uno de los chunks? (c) Con la asignación dynamic y guided, ¿qué cree que debe ocurrir con el número de operaciones de multiplicación y suma que realizan cada uno de los threads?

**RESPUESTA:** 1. Por defecto, para static se asigna un chunk nulo, para dynamic se asigna una unidad de chunk y para guided se asigna también una unidad de chunk.

Para obtener este valor, sólo he usado “omp\_get\_schedule(&kind,&modifier)” al principio del programa y antes de ejecutar el código he usado OMP\_SCHEDULE para definir cada distribución.

2. Realiza el número de chunk \* número de fila.

3. Con guided que el tamaño de bloque será distinto, y por lo demás que se ejecutarán 64 operaciones de golpe o de una en una.

#### 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

int main(int argc, char const *argv[]){

    omp_sched_t kind;
    int modifier;
    omp_get_schedule(&kind,&modifier);
    // comprobamos que los argumentos son correctos
    if (argc < 2) {
        printf("Falta el tamaño de la matriz\n");
        exit(-1);
    }

    int N = atoi(argv[1]);
    int i, j;

    int *v, **matriz;
    v = (int *) malloc(N*sizeof(double)); // malloc necesita el tamaño en bytes
    matriz = (int **) malloc(N*sizeof(int *));
    for(i=0; i<N; i++)
        matriz[i] = (int *) malloc(N*sizeof(int));

    if ( (v==NULL) || (matriz==NULL) ){
        printf("Error en la reserva de espacio para los vectores\n");
        exit(-2);
    }
}
```

```

}

// inicializamos la matriz y el vector
#pragma omp parallel for schedule(runtime) private(j)
for (i = 0; i<N; i++) {
    for (j = 0; j<i+1; j++)
        matriz[i][j] = 1;
    for(int l=N; l>i; l--)
        matriz[i][l]=0;
}

printf("Matriz: \n");
for(int l=0; l<N; l++){
    for(int m=0; m<N; m++)
        printf("%d ", matriz[l][m]);
    printf("\n");
}

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

printf("vector: \n");
for(int l=0; l<N; l++){
    printf("%d \n", v[l]);
}

double t1=omp_get_wtime();
int vector_resultado[N], k = 0, suma=0;

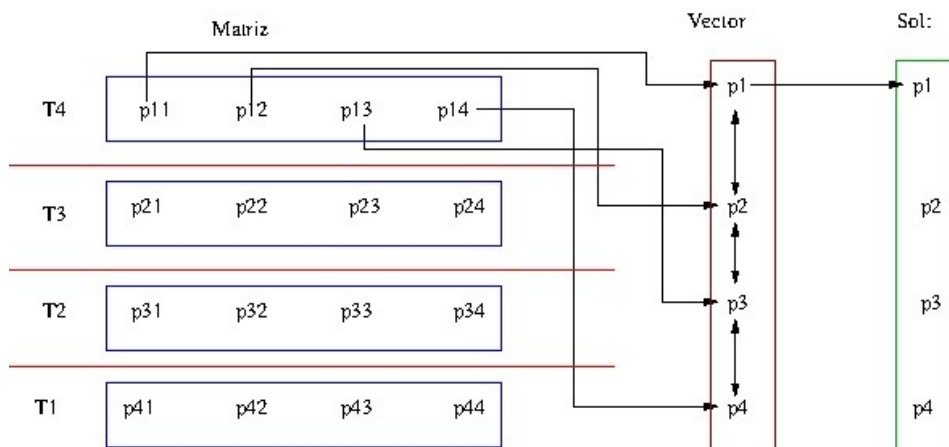
#pragma omp parallel for schedule(runtime) private(k)
// Calculamos el producto
for (i=0; i<N;i++) {
    vector_resultado[i]=0;
    for (k=0; k<N; k++)
        vector_resultado[i]+=matriz[i][k]*v[k];
}

double t2=omp_get_wtime();
t2 = t2-t1;

...
}

```

**DESCOMPOSICIÓN DE DOMINIO:** Se reparten las filas de la matriz y cada thread va



calculando un valor del vector final, recorriendo una fila diferente de la matriz.

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

```

usuario@asus:~/Escritorio/2.Practica 3$ ./pmtv 8
Matriz:
1 0 0 0 0 0 0 0
1 1 0 0 0 0 0 0
1 1 1 0 0 0 0 0
1 1 1 1 0 0 0 0
1 1 1 1 1 0 0 0
1 1 1 1 1 1 0 0
1 1 1 1 1 1 1 0
1 1 1 1 1 1 1 1
vector:
1
2
3
4
5
6
7
8
Tiempo: 0.0000374
v[0] = 1
v[1] = 3
v[2] = 6
v[3] = 10
v[4] = 15
v[5] = 21
v[6] = 28
v[7] = 36

```

### TABLA RESULTADOS, SCRIPT Y GRÁFICA ATCGRID

#### SCRIPT: pmvt-OpenMP\_atcgrid.sh

```

#!/bin/bash

#PBS -N ej7_atcgrid
#PBS -q ac
echo "Id$PBS_O_WORKDIR usuario del trabajo: $PBS_O_LOGNAME"
echo "Id$PBS_O_WORKDIR del trabajo: $PBS_JOBID"

echo "Nombre del trabajo especificado por usuario: $PBS_JOBNAME"
echo "Nodo que ejecuta qsub: $PBS_O_HOST"
echo "Directorio en el que se ha ejecutado qsub: $PBS_O_WORKDIR"
echo "Cola: $PBS_QUEUE"
echo "Nodos asignados al trabajo:"
cat $PBS_NODEFILE

export OMP_SCHEDULE="static"
echo "static y chunk por defecto"
$PBS_O_WORKDIR/pmtv-OpenMP 15360

export OMP_SCHEDULE="static,1"
echo "static y chunk 1"
$PBS_O_WORKDIR/pmtv-OpenMP 15360

export OMP_SCHEDULE="static,64"
echo "static y chunk 64"
$PBS_O_WORKDIR/pmtv-OpenMP 15360

export OMP_SCHEDULE="dynamic"
echo "dynamic y chunk por defecto"
$PBS_O_WORKDIR/pmtv-OpenMP 15360

```

```
export OMP_SCHEDULE="dynamic,1"
echo "dynamic y chunk 1"
$PBS_O_WORKDIR/pmtv-OpenMP 15360

export OMP_SCHEDULE="dynamic,64"
echo "dynamic y chunk 64"
$PBS_O_WORKDIR/pmtv-OpenMP 15360

export OMP_SCHEDULE="guided"
echo "guided y chunk por defecto"
$PBS_O_WORKDIR/pmtv-OpenMP 15360

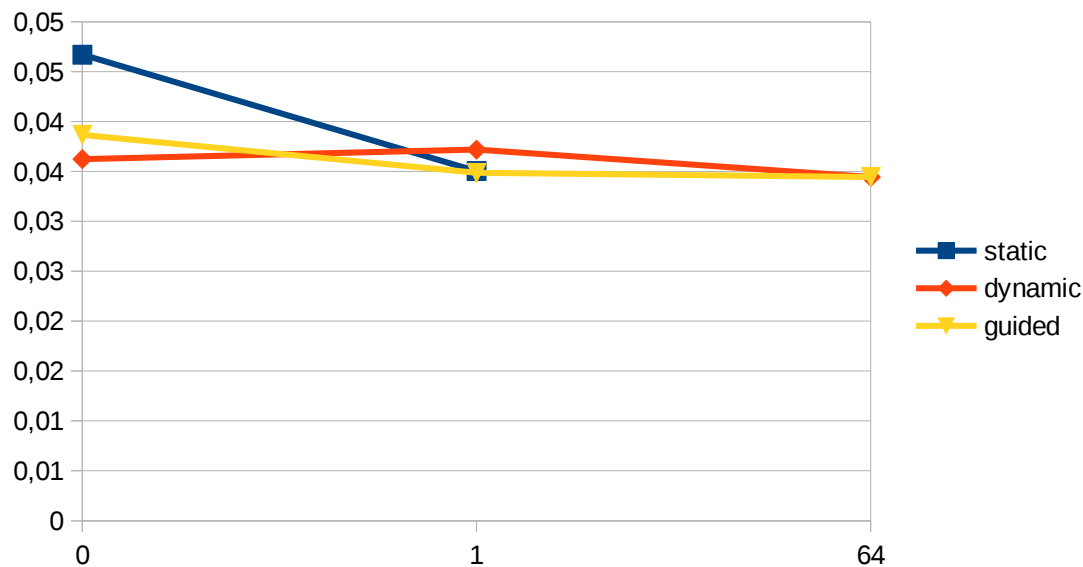
export OMP_SCHEDULE="guided,1"
echo "guided y chunk 1"
$PBS_O_WORKDIR/pmtv-OpenMP 15360

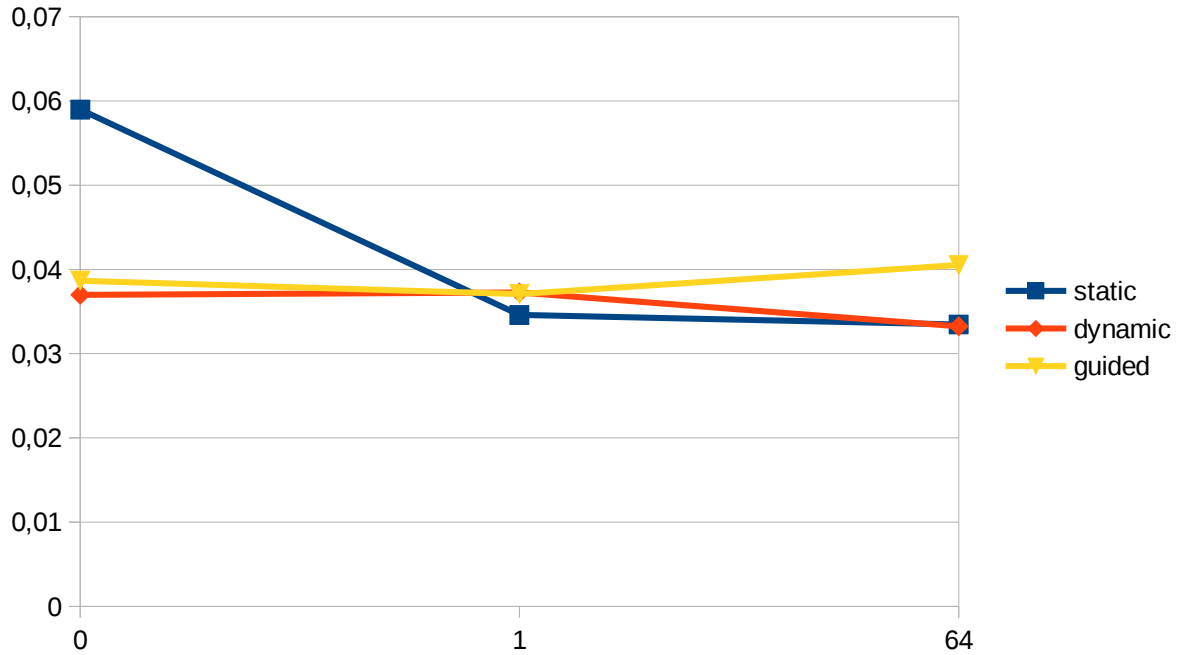
export OMP_SCHEDULE="guided,64"
echo "guided y chunk 64"
$PBS_O_WORKDIR/pmtv-OpenMP 15360
```

**Tabla 3 .** Tiempos de ejecución de la versión paralela del producto de una matriz triangular por un vector r **para vectores de tamaño N=15360** , 12 threads

Chunk	Static	Dynamic	Guided
por defecto	0.046711916	0.036238721	0.038668644
1	0.035047011	0.037187167	0.034859763
64	0.033244486	0.034451319	0.034451054

Chunk	Static	Dynamic	Guided
por defecto	0.058979410	0.036976165	0.041331375
1	0.034590594	0.037256078	0.037069255
64	0.033459864	0.033246293	0.040510096





8. Implementar un programa secuencial en C que calcule la multiplicación de matrices cuadradas, B y C:

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

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 <time.h>

int main(int argc, char **argv)
{
    unsigned i, j, k;

    if(argc < 2)
    {
        fprintf(stderr, "falta size\n");
        exit(-1);
    }

    unsigned int N = atoi(argv[1]); // Máximo N =2^32-1=4294967295
    (sizeof(unsigned int) = 4 B)

    int **a, **b, **c;
    a = (int **) malloc(N*sizeof(int*));
    b = (int **) malloc(N*sizeof(int*));
    c = (int **) malloc(N*sizeof(int*));

    for (i=0; i<N; i++)
```

```

{
    a[i] = (int *) malloc(N*sizeof(int));
    b[i] = (int *) malloc(N*sizeof(int));
    c[i] = (int *) malloc(N*sizeof(int));
}

// Inicializamos las matrices
for (i=0; i<N; i++)
{
    for (j=0; j<N; j++)
    {
        a[i][j] = 0;
        b[i][j] = 2;
        c[i][j] = 2;
    }
}

struct timespec cgt1,cgt2; double ncgt;

clock_gettime(CLOCK_REALTIME,&cgt1);
// Multiplicacion
for (i=0; i<N; i++)
    for (j=0; j<N; j++)
        for (k=0; k<N; k++)
            a[i][j] += b[i][k] * c[k][j];
clock_gettime(CLOCK_REALTIME,&cgt2);

ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+( double) ((cgt2.tv_nsec-
cgt1.tv_nsec)/(1.e+9));

// Pitamos la primera y la ultima linea de la matriz resultante
printf("Tiempo = %11.9f\t Primera = %d\t Ultima=%d\n",ncgt,a[0][0],a[N-1]
[N-1]);

// Liberamos la memoria
for (i=0; i<N; i++)
{
    free(a[i]);
    free(b[i]);
    free(c[i]);
}
free(a);
free(b);
free(c);

return 0;
}

```

**CAPTURAS DE PANTALLA:**

```

usuario@asus:~/Escritorio/2.Practica 3$ gcc -fopenmp -O2 -o pmm pmm-secuencial.c
usuario@asus:~/Escritorio/2.Practica 3$ ./pmm
falta size
usuario@asus:~/Escritorio/2.Practica 3$ ./pmm 8
Tiempo = 0.000000886    Primera = 32    Ultima=32
usuario@asus:~/Escritorio/2.Practica 3$ ./pmm 15
Tiempo = 0.000004343    Primera = 60    Ultima=60
usuario@asus:~/Escritorio/2.Practica 3$ ./pmm 30
Tiempo = 0.000051138    Primera = 120   Ultima=120

```

**(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 dos 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 (pruebe con valores de N entre 100 y 1500). Consulte la Lección 6/Tema 2. Incluya los scripts utilizado en el cuaderno de prácticas. NOTA: Nunca ejecute en atcgrid código que imprima todos los componentes del resultado.

**ESTUDIO DE ESCALABILIDAD EN ATCGRID:****SCRIPT:** pmm-OpenMP\_atcgrid.sh**ESTUDIO DE ESCALABILIDAD EN PCLOCAL:****SCRIPT:** pmm-OpenMP\_pclocal.sh