2º curso / 2º cuatr.

Grado Ing. Inform.

Arquitectura de Computadores (AC)

Cuaderno de prácticas.
Bloque Práctico 3. Programación paralela III: Interacción con el entorno en OpenMP

Estudiante (nombre y apellidos): Álvaro Vega Romero
Grupo de prácticas: B2
Fecha de entrega: 13/05/2021
Fecha evaluación en clase: 14/05/2021

Antes de comenzar a realizar el trabajo de este cuaderno consultar el fichero con los normas de prácticas que se encuentra en SWAD

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.

CAPTURA CÓDIGO FUENTE: if-clauseModificado.c

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

//gcc -O2 -fopenmp -o if-clauseModificado if-clauseModificado.c

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

    if(argc < 3)
    {
        fprintf(stderr,"[ERROR]-Falta iteraciones\n");
        exit(-1);
    }

    n = atoi(argv[1]);
    int x = atoi(argv[2]);

    if (n>20)
        n=20;

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

    #pragma omp parallel num_threads(x) 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:

```
[AlvaroVega zuki@LAPTOP-9IDMGVGU:~/bp3/ejer1] 2021-04-23 Friday
$./if-clauseModificado 7 2
 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 1 suma de a[4]=4 sumalocal=4
 thread 1 suma de a[5]=5 sumalocal=9
 thread 1 suma de a[6]=6 sumalocal=15
thread master=0 imprime suma=21
[AlvaroVega zuki@LAPTOP-9IDMGVGU:~/bp3/ejer1] 2021-04-23 Friday
$./if-clauseModificado 7 5
 thread 1 suma de a[2]=2 sumalocal=2
 thread 1 suma de a[3]=3 sumalocal=5
 thread 0 suma de a[0]=0 sumalocal=0
 thread 0 suma de a[1]=1 sumalocal=1
 thread 2 suma de a[4]=4 sumalocal=4
 thread 3 suma de a[5]=5 sumalocal=5
 thread 4 suma de a[6]=6 sumalocal=6
thread master=0 imprime suma=21
[AlvaroVega zuki@LAPTOP-9IDMGVGU:~/bp3/ejer1] 2021-04-23 Friday
$./if-clauseModificado 7 3
 thread 2 suma de a[5]=5 sumalocal=5
 thread 2 suma de a[6]=6 sumalocal=11
 thread 1 suma de a[3]=3 sumalocal=3
 thread 1 suma de a[4]=4 sumalocal=7
 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 master=0 imprime suma=21
[AlvaroVega zuki@LAPTOP-9IDMGVGU:~/bp3/ejer1] 2021-04-23 Friday
```

RESPUESTA: ejecutamos el programa con el número de hebras que se especifican como segundo argumento, pudiendo hacer el ejercicio con el número de hebras que queramos. (con un máximo que es el número de hebras del ordenador y n, que hace que si por ejemplo tengamos 7 iteraciones y 10 hebras, se asigne como mucho 1 hebra a cada iteración).

2. Rellenar la Tabla 1 (se debe poner en la tabla el id del *thread* que ejecuta cada iteración) usando scheduler-clause.c con tres *threads* (0,1,2) y un número de iteraciones de 16 (0 a 15 en la tabla). Con este ejercicio se pretende comparar distintas alternativas de planificación de bucles. Se van a usar distintos tipos (static, dynamic, guided), modificadores (monotonic y nonmonotonic) y tamaños de chunk (x = 1, 2 y 4).

Tabla 1 . Tabla schedule. Rellenar esta tabla ejecutando scheduler-clause.c asignando previamente a la variable de entorno OMP_SCHEDULE los valores que se indican en la tabla (por ej.: export OMP_SCHEDULE="nonmonotonic:static,2). En la segunda fila, 1, 2 4 representan el tamaño del chunk

(no funciona el monotonic/nonmonotonic (libgomp: Unknown value for environment variable OMP_SCHEDULE), así que lo hago sin ponerlo y ocupo la parte de monotonic)

Iteración	"monotonic:static,x"		"nonmonotonic:static,x"		"monotonic:dynamic,x"		"monotonic:guided,x"	
	x=1	x=2	x=1	x=2	x=1	x=2	x=1	x=2
0	0	0	?	?	2	2	1	2
1	1	0			0	2	1	2
2	2	1			1	1	1	2
3	0	1			2	1	1	2
4	1	2			2	0	1	2
5	2	2			2	0	1	2
6	0	0			2	0	0	0
7	1	0			2	0	0	0
8	2	1			2	0	0	0
9	0	1			2	0	0	0
10	1	2			2	0	2	1
11	2	2			2	0	2	1
12	0	0			2	0	1	0
13	1	0			2	0	1	0
14	2	1			2	0	1	0
15	0	1			2	0	1	0

Iteración	"monotonic:static,x"		"nonmonotonic:static,x"		"monotonic:dynamic,x"		"monotonic:guided,x"	
	x=4	-	x=4	-	x=4	-	x=4	-
0	0		?	-	2		2	
1	0				2		2	
2	0				2		2	
3	0				2		2	
4	1				0		2	
5	1				0		2	
6	1				0		0	
7	1				0		0	
8	2				1		0	
9	2				1		0	
10	2				1		1	
11	2				1		1	
12	0				2		1	
13	0				2		1	
14	0				2		2	
15	0				2		2	

Destacar las diferencias entre las 4 alternativas de planificación de la tabla, en particular, las que hay entre static, dynamic y guided y las diferencias entre usar monotonic y nonmonotonic.

RESPUESTA:

La diferencia entre las 3 alternativas son:

-Static, distribución de iteraciones en tiempo de compilación, diviendo las iteraciones en chunks de manera que las hebras se van repartiendo las iteraciones de chunk en chunk (ej de 2 en 2…).

-Dynamic, la distribución se hace en tiempo de ejecución, por lo que se desconoce a priori, que hebra ejecutará cada iteración. Así, las hebras más rápidas ejecutarán más trabajo, aunque tendrán que ejecutar el chunk asignado.

-Guided, también se hace la distribución en tiempo de ejecución, pero aquí, se empieza con un bloque formado por todas las iteraciones, el cual va decreciendo dependiendo del número de iteraciones que se resten, pero nunca menor que el chunk asignado. (Las hebras rápidas también ejecutarán más trabajo).

La diferencia entre monotonic y nonmonotonic son:

En monotonic indica que el reparto se va a realizar en round-robin de forma correlativa, comenzando en la 0. *"Si una hebra ejecuta una iteración i, entonces la hebra debe ejecutar iteracciones más grande que i."*

En nonmonotic indica que el reparto se va a realizar en round-robin pero no tiene porqué comenzar en la hebra 0, es decir, no va a haber una correlación lineal.

3. ¿Qué valor por defecto usa OpenMP para chunk y modifier con static, dynamic y guided? Explicar qué ha hecho para contestar a esta pregunta.

Consultamos la variable de entorno o función de entorno.

```
int modifier;
omp_sched_t kind;

omp_get_schedule(&kind, &modifier);
```

Usaremos el programa Schedule-clause y haremos el siguiente printf:

```
printf("Valores por defecto: %d y el chunk: %d \n", kind, modifier);
```

Así, podemos observar que cuando es static, kind vale 1 y modifier vale 0, con dynamic, kind vale 2 y modifier 1, luego con guided kind devuelve 3 y modifier 1 y si lo ponemos en auto, kind vale 4 y modifier vale 1.

4. 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?

CAPTURA CÓDIGO FUENTE: scheduled-clauseModificado.c

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

    n = atoi(argv[1]);
    if (n>200)
        n=200;
    chunk = atoi(argv[2]);

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

    omp_sched_t tipo;
    int modif;

    #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 single
        {

        omp_get_schedule(&tipo, &modif);
        printf("DENTRO DE LA REGIÓN PARALELIRAZADA: \n");
        printf("dyn_var: %d\n", omp_get_dynamic());
        printf("nthreads-var: %d\n", omp_get_max_threads());
        printf("thread-limit-var: %d\n", omp_get_thread_limit());
        printf("run-sched-var: %d y modif: %d \n", tipo, modif);

        }
    }

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

    omp_get_schedule(&tipo, &modif);
    printf("DENTRO DE LA REGIÓN PARALELIRAZADA: \n");
    printf("dyn_var: %d\n", omp_get_dynamic());
    printf("nthreads-var: %d\n", omp_get_max_threads());
    printf("thread-limit-var: %d\n", omp_get_thread_limit());
    printf("run-sched-var: %d y modif: %d \n", tipo, modif);
}
```

CAPTURAS DE PANTALLA:

```
 thread 2 suma a[4]=4 suma=4
 thread 2 suma a[5]=5 suma=9
 thread 2 suma a[6]=6 suma=15
 thread 2 suma a[7]=7 suma=22
 thread 2 suma a[12]=12 suma=34
 thread 2 suma a[13]=13 suma=47
 thread 2 suma a[14]=14 suma=61
 thread 2 suma a[15]=15 suma=76
 thread 1 suma a[8]=8 suma=8
 thread 1 suma a[9]=9 suma=17
 thread 1 suma a[10]=10 suma=27
 thread 1 suma a[11]=11 suma=38
 thread 0 suma a[0]=0 suma=0
 thread 0 suma a[1]=1 suma=1
 thread 0 suma a[2]=2 suma=3
 thread 0 suma a[3]=3 suma=6
DENTRO DE LA REGIÓN PARALELIRAZADA:
dyn_var: 0
nthreads-var: 3
thread-limit-var: 2147483647
run-sched-var: 2 y modif: 1
Fuera de 'parallel for' suma=76
DENTRO DE LA REGIÓN PARALELIRAZADA:
dyn_var: 0
nthreads-var: 3
thread-limit-var: 2147483647
run-sched-var: 2 y modif: 1
[AlvaroVega zuki@LAPTOP-9IDMGVGU:~/bp3/ejer4] 2021-04-23 Friday
$./scheduled-clauseModificado 8 3
 thread 2 suma a[3]=3 suma=3
 thread 2 suma a[4]=4 suma=7
 thread 2 suma a[5]=5 suma=12
 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 0 suma a[6]=6 suma=6
 thread 0 suma a[7]=7 suma=13
DENTRO DE LA REGIÓN PARALELIRAZADA:
dyn_var: 0
nthreads-var: 3
thread-limit-var: 2147483647
run-sched-var: 2 y modif: 1
Fuera de 'parallel for' suma=13
DENTRO DE LA REGIÓN PARALELIRAZADA:
dyn_var: 0
nthreads-var: 3
thread-limit-var: 2147483647
run-sched-var: 2 y modif: 1
[AlvaroVega zuki@LAPTOP-9IDMGVGU:~/bp3/ejer4] 2021-04-23 Friday
```

RESPUESTA:

No se imprimen valores diferentes ya que son variables de entorno y no locales a la región en la que se imprimen.

5. 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.

CAPTURA CÓDIGO FUENTE: scheduled-clauseModificado4.c

```c
#include <stdio.h>
#include <stdlib.h>
#ifdef _OPENMP
    #include <omp.h>
#else
    #define omp_get_thread_num() 0
#endif
main(int argc, char **argv)
{
    int i, n=200,chunk,a[n],suma=0;
    if(argc < 3)
    {
        fprintf(stderr,"\nFalta iteraciones o chunk \n");
        exit(-1);
    }

    n = atoi(argv[1]);
    if (n>200)
        n=200;
    chunk = atoi(argv[2]);

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

    omp_sched_t tipo;
    int modif;

    #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 single
        {

        printf("DENTRO DE LA REGIÓN PARALELIRAZADA: \n");
        printf("omp_get_num_threads(): %d\n", omp_get_num_threads());
        printf("omp_get_num_procs() : %d\n", omp_get_num_procs());
        printf("omp_in_parallel() : %d\n", omp_in_parallel() );

        }
    }

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

    printf("FUERA DE LA REGIÓN PARALELIRAZADA: \n");
    printf("omp_get_num_threads(): %d\n", omp_get_num_threads());
    printf("omp_get_num_procs() : %d\n", omp_get_num_procs());
    printf("omp_in_parallel() : %d\n", omp_in_parallel() );
}
```

CAPTURAS DE PANTALLA:

```
[AlvaroVega zuki@LAPTOP-9IDMGVGU:~/bp3/ejer5] 2021-04-23 Friday
$./scheduled-clauseModificado4 16 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[12]=12 suma=18
 thread 0 suma a[13]=13 suma=31
 thread 0 suma a[14]=14 suma=45
 thread 0 suma a[15]=15 suma=60
 thread 1 suma a[8]=8 suma=8
 thread 1 suma a[9]=9 suma=17
 thread 1 suma a[10]=10 suma=27
 thread 1 suma a[11]=11 suma=38
 thread 2 suma a[4]=4 suma=4
 thread 2 suma a[5]=5 suma=9
 thread 2 suma a[6]=6 suma=15
 thread 2 suma a[7]=7 suma=22
DENTRO DE LA REGIÓN PARALELIRAZADA:
omp_get_num_threads(): 3
omp_get_num_procs() : 8
omp_in_parallel() : 1
Fuera de 'parallel for' suma=60
FUERA DE LA REGIÓN PARALELIRAZADA:
omp_get_num_threads(): 1
omp_get_num_procs() : 8
omp_in_parallel() : 0
[AlvaroVega zuki@LAPTOP-9IDMGVGU:~/bp3/ejer5] 2021-04-23 Friday
$./scheduled-clauseModificado4 5 2
 thread 1 suma a[2]=2 suma=2
 thread 1 suma a[3]=3 suma=5
 thread 0 suma a[4]=4 suma=4
 thread 2 suma a[0]=0 suma=0
 thread 2 suma a[1]=1 suma=1
DENTRO DE LA REGIÓN PARALELIRAZADA:
omp_get_num_threads(): 3
omp_get_num_procs() : 8
omp_in_parallel() : 1
Fuera de 'parallel for' suma=4
FUERA DE LA REGIÓN PARALELIRAZADA:
omp_get_num_threads(): 1
omp_get_num_procs() : 8
omp_in_parallel() : 0
[AlvaroVega zuki@LAPTOP-9IDMGVGU:~/bp3/ejer5] 2021-04-23 Friday
```

RESPUESTA: Se obtienen diferentes valores dentro y fuera de la región paralelizada ya que fuera solo hay una hebra y dentro puede haber más. También, la función omp_in_parallel devuelve diferentes valores depende de si estamos dentro (1) o fuera de la región (0).

6. Añadir al programa scheduled-clause.c lo necesario para, usando funciones, modificar las variables de control dyn-var, nthreads-var y run-sched-var dentro de la región paralela y fuera de la región paralela. En la modificación de run-sched-var se debe usar un valor de kind distinto al utilizado en la cláusula schedule(). Añadir lo necesario para imprimir el contenido de estas variables antes y después de cada una de las dos modificaciones. Comentar los resultados.

CAPTURA CÓDIGO FUENTE: scheduled-clauseModificado5.c

```
        #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 single
        {

        omp_get_schedule(&tipo, &modif);
        printf("DENTRO DE LA REGIÓN PARALELIRAZADA, ANTES DE MODIFICAR: \n");

        printf("dyn_var: %d\n", omp_get_dynamic());
        printf("nthreads-var: %d\n", omp_get_max_threads());
        printf("run-sched-var: %d y modif: %d \n\n", tipo, modif);

        if(tipo == omp_sched_static) printf("Tipo es static\n");
        else if(tipo == omp_sched_dynamic) printf("Tipo es dinámico\n");
        else if(tipo == omp_sched_guided) printf("Tipo es guided\n");
        else printf("Tipo es auto\n");

        //Modificamos

        omp_set_dynamic(10);
        omp_set_num_threads(4); //asignarle 4 hebras
        omp_set_schedule(1, 2);

        omp_get_schedule(&tipo, &modif);
        printf("DENTRO DE LA REGIÓN PARALELIRAZADA, DESPUÉS DE MODIFICAR: \n");

        printf("dyn_var: %d\n", omp_get_dynamic());
        printf("nthreads-var: %d\n", omp_get_max_threads());
        printf("run-sched-var: %d y modif: %d \n\n", tipo, modif);

        }
    }

    omp_get_schedule(&tipo, &modif);
    printf("FUERA DE LA REGIÓN PARALELIRAZADA, ANTES DE MODIFICAR: \n");

    printf("dyn_var: %d\n", omp_get_dynamic());
    printf("nthreads-var: %d\n", omp_get_max_threads());
    printf("run-sched-var: %d y modif: %d \n\n", tipo, modif);

    omp_set_dynamic(5);
    omp_set_num_threads(2); //asignarle 4 hebras
    omp_set_schedule(2, 1);

    printf("Fuera de 'parallel for' suma=%d\n",suma);
    printf("FUERA DE LA REGIÓN PARALELIRAZADA, DESPUÉS DE MODIFICAR: \n");
    printf("dyn_var: %d\n", omp_get_dynamic());
    printf("nthreads-var: %d\n", omp_get_max_threads());
    printf("run-sched-var: %d y modif: %d \n", tipo, modif);
}
```

CAPTURAS DE PANTALLA:

```
[AlvaroVega zuki@LAPTOP-9IDMGVGU:~/bp3/ejer6] 2021-05-12 Wednesday
$./scheduled-clauseModificado5 16 2
 thread 0 suma a[4]=4 suma=4
 thread 0 suma a[5]=5 suma=9
 thread 5 suma a[8]=8 suma=8
 thread 5 suma a[9]=9 suma=17
 thread 3 suma a[12]=12 suma=12
 thread 3 suma a[13]=13 suma=25
 thread 4 suma a[2]=2 suma=2
 thread 4 suma a[3]=3 suma=5
 thread 7 suma a[6]=6 suma=6
 thread 7 suma a[7]=7 suma=13
 thread 2 suma a[0]=0 suma=0
 thread 2 suma a[1]=1 suma=1
 thread 6 suma a[10]=10 suma=10
 thread 6 suma a[11]=11 suma=21
 thread 1 suma a[14]=14 suma=14
 thread 1 suma a[15]=15 suma=29
DENTRO DE LA REGIÓN PARALELIRAZADA, ANTES DE MODIFICAR:
dyn_var: 0
nthreads-var: 8
run-sched-var: 2 y modif: 1

Tipo es dinámico
DENTRO DE LA REGIÓN PARALELIRAZADA, DESPUÉS DE MODIFICAR:
dyn_var: 1
nthreads-var: 4
run-sched-var: 1 y modif: 2

FUERA DE LA REGIÓN PARALELIRAZADA, ANTES DE MODIFICAR:
dyn_var: 0
nthreads-var: 8
run-sched-var: 2 y modif: 1

Fuera de 'parallel for' suma=29
FUERA DE LA REGIÓN PARALELIRAZADA, DESPUÉS DE MODIFICAR:
dyn_var: 1
nthreads-var: 2
run-sched-var: 2 y modif: 1
[AlvaroVega zuki@LAPTOP-9IDMGVGU:~/bp3/ejer6] 2021-05-12 Wednesday
```

RESPUESTA:

Al modificarse las variables de control toman los valores dados por las funciones de entorno en tiempos de ejecución.
Dentro de la zona paralela podemos ver como se modifican los valores de estar variables, pero al salir de la zona, se restauran los valores previos al cambio.

Fuera de la región paralela, podemos ver, como también se modifican, pero se modifican para el resto del programa (cabe decir que he modificado algunos valores a los valores previos y no se percibe el cambio).

Resto de ejercicios (usar en atcgrid la cola ac a no ser que se tenga que usar atcgrid4)

7. Implementar un programa secuencial en C que multiplique una matriz triangular inferior por un vector (use variables dinámicas y tipo de datos double). Comparar el orden de complejidad y el número total de operaciones (sumas y productos) de este código respecto al 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.

CAPTURA CÓDIGO FUENTE: pmtv-secuencial.c

```
    for(i = 0 ; i < n ; i++)
    {
        A[i] = (double*) malloc (n*sizeof(double));

        if(A[i] == NULL)
        {
            printf("Error en la reserva de memoria\n");
            exit(-3);
        }
    }

    for(i = 0 ; i < n ; i++) //Rellenar matriz de datos "aleatorios" y Vectores
    {
        B[i]= 0.5*i;
        C[i]=0;
        for(j = 0 ; j < n ; j++)
            A[i][j] = 0.5*i - 0.5*j;
    }

    cgt1 = omp_get_wtime();

    for(i = 0 ; i < n ; i++) //Calculo del producto
    {
        for(j = i ; j >= 0 ; j--)
        {
            C[i] += A[i][j] * B[j];
/*
                printf("A[%d][%d] = %f con i=%d y j=%d ", i, j, A[i][j], i, j);
                printf("B[%d] = %f con j=%d ", j, B[j], j);
                printf("\n\n");
                */
        }
    }

    cgt2 = omp_get_wtime();
    tiempo = cgt2 - cgt1;

    //Salida
```

CAPTURAS DE PANTALLA:

```
[ÁlvaroVega b2estudiante24@atcgrid:~/bp3/ejer7] 2021-05-12 Wednesday
$srun -pac pmtv-secuencial 3
Tiempo de ejecución: 0.000002
Valor de n: 3
C[0] = 0.000000
C[ULTIMO] = 0.250000
[ÁlvaroVega b2estudiante24@atcgrid:~/bp3/ejer7] 2021-05-12 Wednesday
$srun -pac pmtv-secuencial 100
Tiempo de ejecución: 0.000013
Valor de n: 100
C[0] = 0.000000
C[ULTIMO] = 40425.000000
[ÁlvaroVega b2estudiante24@atcgrid:~/bp3/ejer7] 2021-05-12 Wednesday
$srun -pac pmtv-secuencial 10000
Tiempo de ejecución: 0.074306
Valor de n: 10000
C[0] = 0.000000
C[ULTIMO] = 41654167500.000000
[ÁlvaroVega b2estudiante24@atcgrid:~/bp3/ejer7] 2021-05-12 Wednesday
```

Ahora el for de la multiplicación, la j empieza desde i, en vez de desde 0, haciéndose así muchas menos

iteraciones (los elementos que se encuentran encima de la diagonal principal no se usan para nada)

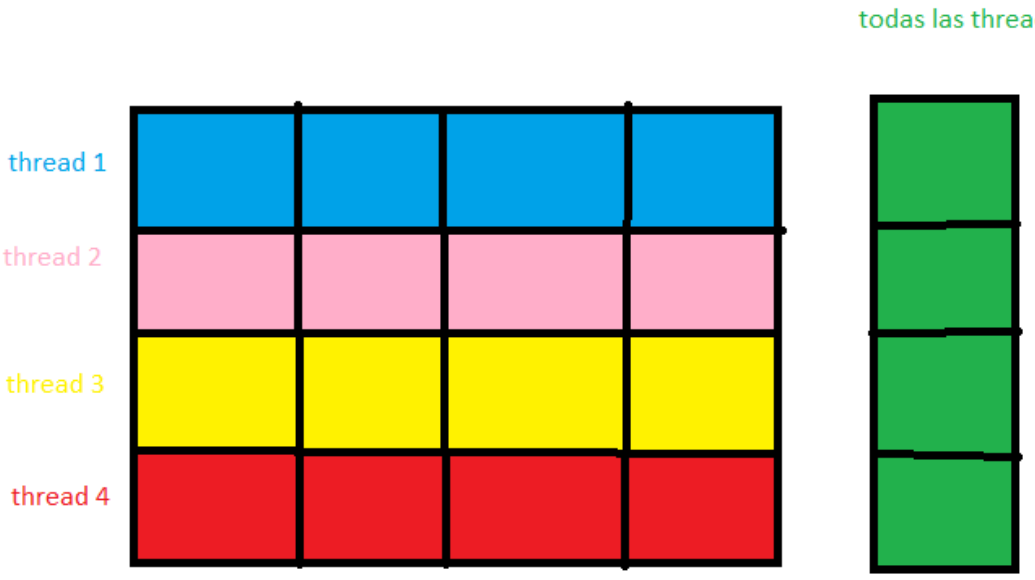
8. Implementar en paralelo la multiplicación de una matriz triangular inferior por un vector a partir del código secuencial realizado para el ejercicio anterior utilizando la directiva for de OpenMP. El código debe repartir entre los threads las iteraciones del bucle que recorre las filas. La inicialización de los datos la debe hacer el thread 0. 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. Mostrar en una captura de pantalla que el código resultante funciona correctamente. NOTA: usar para generar los valores aleatorios, por ejemplo, drand48_r().

CAPTURA CÓDIGO FUENTE: pmtv-OpenMP.c

```
for(i = 0 ; i < n ; i++) //Rellenar matriz de datos "aleatorios" y Vectores - Se puede hacer con directiva master, o de forma secuencial
{
    B[i]= 0.5*i;
    C[i]=0;
    for(j = 0 ; j < n ; j++)
        A[i][j] = 0.5*i - 0.5*j;
}

cgt1 = omp_get_wtime();

#pragma omp parallel for shared(A,B,C,n) schedule(runtime) private(j) //i ya es privada
    for(i = 0 ; i < n ; i++) //Calculo del producto
    {

        printf(" thread %d, fila: %d \n", omp_get_thread_num(),i);

        for(j = i ; j >= 0 ; j--)
        {
            C[i] += A[i][j] * B[j];
                /*
                printf("A[%d][%d] = %f con i=%d y j=%d ", i, j, A[i][j], i, j);
                printf("B[%d] = %f con j=%d ", j, B[j], j);
                printf("\n\n");
                */
        }
    }

cgt2 = omp_get_wtime();
tiempo = cgt2 - cgt1;
```

DESCOMPOSICIÓN DE DOMINIO:

Todas las hebras accederán a todo el vector, pero cada hebra se encargará de procesar una fila (aunque en cada fila no se procese entera al ser una matriz triangular inferior).
El número de fila que este procesando el thread, será el resultado que se almacenará en las posición fila del vector resultado.



CAPTURAS DE PANTALLA:

```
[ÁlvaroVega b2estudiante24@atcgrid:~/bp3/ejer8] 2021-05-12 Wednesday
$export OMP_NUM_THREADS=16
[ÁlvaroVega b2estudiante24@atcgrid:~/bp3/ejer8] 2021-05-12 Wednesday
$export OMP_SCHEDULE="static,2"
[ÁlvaroVega b2estudiante24@atcgrid:~/bp3/ejer8] 2021-05-12 Wednesday
$srun -pac pmtv-OpenMP 10
 thread 1, fila: 2
 thread 1, fila: 3
 thread 2, fila: 4
 thread 2, fila: 5
 thread 3, fila: 6
 thread 3, fila: 7
 thread 4, fila: 8
 thread 4, fila: 9
 thread 0, fila: 0
 thread 0, fila: 1
Tiempo de ejecución: 0.000630
Valor de n: 10
C[0] = 0.000000
C[ULTIMO] = 30.000000
[ÁlvaroVega b2estudiante24@atcgrid:~/bp3/ejer8] 2021-05-12 Wednesday
$export OMP_SCHEDULE="dynamic,2"
[ÁlvaroVega b2estudiante24@atcgrid:~/bp3/ejer8] 2021-05-12 Wednesday
$srun -pac pmtv-OpenMP 10
 thread 1, fila: 0
 thread 1, fila: 1
 thread 2, fila: 2
 thread 2, fila: 3
 thread 3, fila: 4
 thread 3, fila: 5
 thread 4, fila: 6
 thread 4, fila: 7
 thread 5, fila: 8
 thread 5, fila: 9
Tiempo de ejecución: 0.000667
Valor de n: 10
C[0] = 0.000000
C[ULTIMO] = 30.000000
[ÁlvaroVega b2estudiante24@atcgrid:~/bp3/ejer8] 2021-05-12 Wednesday
```

(Destacar que Dynamic con srun no se comporta igual que con ./)

9. Contestar a las siguientes preguntas sobre el código del ejercicio anterior:

(a) ¿Qué número de operaciones de multiplicación y qué número de operaciones de suma realizan cada uno de los threads en la asignación static con monotonic y un chunk de 1?

RESPUESTA:

La última hebra (supongo que no es la hebra 0 la misma que cierra) realizará más operaciones que la hebra que se inicia multiplicando, al tener que procesarse la fila entera para la última fila y en cambio, para la primera, solo la primera componente, estando esta hebra más osciosa. Por tanto, la repartición estática no es la más conveniente.

Si tuviésemos 4 hebras y la matriz fuese de 4x4, la hebra 1 estaría ociosa, mientras que la hebra 4 siguiese trabajando.

El número de multiplicaciones para la primera hebra sería N-3 y N-4 sumas y para la última sería N multiplicaciones y N-1 sumas.

OJO: es matriz triangular inferior y no superior.

(b) 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: Si el reparto fuera dinámico, cuando una hebra termine, se le asigna el siguiente chunk, de forma que no hay hebras ociosas.

Si usamos guided, partiremos de un número de iteraciones y de hebras, y se irán reduciendo el bloque de iteraciones mientras van terminando cada hebra.

Por tanto, con el reparto dinámico, el número total de multiplicaciones y sumas que hará cada hebra, será más parejo.

(c) ¿Qué alternativa ofrece mejores prestaciones? Razonar la respuesta.

RESPUESTA:

Mejor un reparto dinámico, ya que evitamos que haya hebras ociosas y la carga se reparta de la mejor forma para que se termine la ejecución lo antes posible.

10. 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 (con monotonic en todos los casos). Usar un tamaño de vector N múltiplo del número de cores y de 64 que esté entre 11520 y 23040. El número de threads en las ejecuciones debe coincidir con el número de núcleos del computador. 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 (representar los valores de las dos tablas). Incluir los scripts utilizado en el cuaderno de prácticas. NOTA: Nunca ejecute en atcgrid código que imprima todos los componentes del resultado.

CAPTURA CÓDIGO FUENTE: pmtv-OpenMP.c

DESCOMPOSICIÓN DE DOMINIO:

CAPTURAS DE PANTALLA:

TABLA RESULTADOS, SCRIPT Y GRÁFICA atcgrid

SCRIPT: pmvt-OpenMP_atcgrid.sh

Tabla 2 . Tiempos de ejecución de la versión paralela del producto de una matriz triangular por un vector para vectores de tamaño N= (solo se ha paralelizado el producto, no la inicialización de los datos).

Chunk	Static	Dynamic	Guided
por defecto			
1			
64			

Chunk	Static	Dynamic	Guided
por defecto			
1			
64			