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):
Grupo de prácticas:

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>

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

    /*
     * No se produce la paralelizacion a menos que el numero de iteraciones sea mayor que
     * 4
    */
    if (argc < 2)
    {
        fprintf(stderr,"[ERROR] Falta el número de iteraciones\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(5) if(n>4) default(none) \
              private(sumalocal,tid) shared(a,suma,n)
    */

    #pragma omp parallel num_threads(n) 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:

```
┌──(icaro㉿kali)-[~/…/AC/Practica/Seminario3/Codigo]
└─$ export OMP_NUM_THREADS=8

┌──(icaro㉿kali)-[~/…/AC/Practica/Seminario3/Codigo]
└─$ ./if-clauseModificado 2
 thread 0 suma de a[0]=0 sumalocal=0
 thread 0 suma de a[1]=1 sumalocal=1
thread master=0 imprime suma=1

┌──(icaro㉿kali)-[~/…/AC/Practica/Seminario3/Codigo]
└─$ ./if-clauseModificado 4
 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

┌──(icaro㉿kali)-[~/…/AC/Practica/Seminario3/Codigo]
└─$ ./if-clauseModificado 5
 thread 2 suma de a[2]=2 sumalocal=2
 thread 0 suma de a[0]=0 sumalocal=0
 thread 4 suma de a[4]=4 sumalocal=4
 thread 3 suma de a[3]=3 sumalocal=3
 thread 1 suma de a[1]=1 sumalocal=1
thread master=0 imprime suma=10

┌──(icaro㉿kali)-[~/…/AC/Practica/Seminario3/Codigo]
└─$ ./if-clauseModificado 9
 thread 0 suma de a[0]=0 sumalocal=0
 thread 1 suma de a[1]=1 sumalocal=1
 thread 6 suma de a[6]=6 sumalocal=6
 thread 2 suma de a[2]=2 sumalocal=2
 thread 7 suma de a[7]=7 sumalocal=7
 thread 5 suma de a[5]=5 sumalocal=5
 thread 8 suma de a[8]=8 sumalocal=8
 thread 4 suma de a[4]=4 sumalocal=4
 thread 3 suma de a[3]=3 sumalocal=3
thread master=0 imprime suma=36

┌──(icaro㉿kali)-[~/…/AC/Practica/Seminario3/Codigo]
└─$ gcc -O2 -fopenmp if-clauseModificado.c -o if-clauseModificado

┌──(icaro㉿kali)-[~/…/AC/Practica/Seminario3/Codigo]
└─$ ./if-clauseModificado 9
 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 4 suma de a[8]=8 sumalocal=8
 thread 2 suma de a[4]=4 sumalocal=4
 thread 2 suma de a[5]=5 sumalocal=9
 thread 3 suma de a[6]=6 sumalocal=6
 thread 3 suma de a[7]=7 sumalocal=13
thread master=0 imprime suma=36
```

RESPUESTA:
if-clause lee un parametro de entrada que sera el numero de iteraciones del bucle, como maximo sera 20
si el parametro introducido no es mayor a 4 no se produce una ejecucion paralela, si no una secuencial.

La condicion la impone el if y para indicar el numero de hebras que se ejecutaran la parte paralela se usa la clausula num_threads.

Si se cumple la condicion, la clausula num thread establece a n el numero de threads que ejecutara el parallel (n tmb es el n.º de iteracion por lo que 1 hebra por iteracion)

RECUERDA EL ORDEN DE PREFERENCIA A LA HORA DE ASIGNAR N.º THREADS

1º - n.º resulte de evaluar el if
2º - luego el num_thread
3º - omp_set_num_threads()
4º - export OMP_NUM_THREADS

en el ejemplo antes de compilar y ejecutamos ./if-clauseModificado 9 estamos diciendo que habra 9 hebras ya que num_thread(9) pero previamente hemos dicho que export OMP_NUM_THREADS = 8, prevalece el primero y se crean las hebras del 0 al 8 (9 hebras totales)

Luego de compilar estamos diciendo que num_threads (5) en vez de num_threads(n) por eso al volver a ejecutar ./if-clauseMOdificado 9 luego de la compilacion se ejecutan 5 hebras, no n hebras

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 y 2).

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 y 2 representan el tamaño del chunk

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

*monotonic:static y nonmonotonic:static me salen igual, no se por que

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`.

Diferencias static, dynamic guided:

Static: Distribucion de iteraciones en tiempo de compilacion, dividiendo las iteraciones en chuncks de manera que las hebras se van repartiendo las iteraciones de chunk en chunk

Ej: chunk = 2 y iter = 16 y num_threads = 3 cada hebra ejecutara chunk iteraciones (2) siguiendo una politica Round Robin → 0 0 1 1 2 2 0 0 1 1 2 2 ….

Dynamic: La distribucion se hace en tiempo de ejecucion, por lo que se desconoce a priori, que hebra ejecuta cada iteracion. Asi las hebras más rapidas ejecutaran más trabajpp, aunque tendrán que ejecutar el chunk asignado.
No reliza Round Robin, si no que asigna más iteracionas a la hebra más rapida
n.º unidades O(iter/chunk) en este ej → 16/3=5

Ej en la tabla de arriba vemos que la hebra más rapida es la 0 ya que es la que ejecuta más

Guided: distribucion en tiempo de ejecucion, aquí empieza con un bloque formado por todas las iteraciones, el cual va decreciendo dependiendo del numero de iteraciones que resten, pero nunca menor que el chunck asignado (las hebras más rapidas tmb ejecutaran más trabajo)

Diferencia entre monotonic y nonmonotonic

monotonic: indica que el reparto se realiza en round robin de forma correlativa comenzando en la 0. " si una hebra ejecuta una iteracion i , entonces la hebra debe ejecutar iteraciones más grande que i"

nonmonotonic: indica que el reparto se va a realizar en round robin pero no tiene por que comenzar en la hebra 0 , no hay una correlacion 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.

Para ver el valor por defecto vamos a coger scheduler-clause.c y copiarlo a valorDEfecto y vamos a añadirle una clausula omp_get_schedule para ver los valores de kind y modifier

```
int modifier;
omp_sched_t kind;

omp_get_schedule(&kind,&modifier);

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

Cuando es static → kind = basura ; modifier = 0 * revisar
dynamic → kind = 2; modifier = 1
guided → kind = 3 ; modifier = 1
auto → kind = 4; modifier = 1

```
┌──(icaro㉿kali)-[~/…/AC/Practica/Seminario3/Codigo]
└─$ export OMP_SCHEDULE="nonmonotonic:static"

┌──(icaro㉿kali)-[~/…/AC/Practica/Seminario3/Codigo]
└─$ ./valorDefecto 16
thread 0 suma a[0]=0 suma=0
thread 0 suma a[1]=1 suma=1
thread 4 suma a[8]=8 suma=8
thread 4 suma a[9]=9 suma=17
thread 6 suma a[12]=12 suma=12
thread 6 suma a[13]=13 suma=25
thread 5 suma a[10]=10 suma=10
thread 5 suma a[11]=11 suma=21
thread 7 suma a[14]=14 suma=14
thread 7 suma a[15]=15 suma=29
thread 1 suma a[2]=2 suma=2
thread 1 suma a[3]=3 suma=5
thread 3 suma a[6]=6 suma=6
thread 3 suma a[7]=7 suma=13
thread 2 suma a[4]=4 suma=4
thread 2 suma a[5]=5 suma=9
fuera de 'parallel for' suma=29
Valores por defecto: 1 y el chunk: 0

┌──(icaro㉿kali)-[~/…/AC/Practica/Seminario3/Codigo]
└─$ export OMP_SCHEDULE="monotonic:static"

┌──(icaro㉿kali)-[~/…/AC/Practica/Seminario3/Codigo]
└─$ ./valorDefecto 16
thread 0 suma a[0]=0 suma=0
thread 0 suma a[1]=1 suma=1
thread 7 suma a[14]=14 suma=14
thread 7 suma a[15]=15 suma=29
thread 3 suma a[6]=6 suma=6
thread 3 suma a[7]=7 suma=13
thread 4 suma a[8]=8 suma=8
thread 6 suma a[12]=12 suma=12
thread 6 suma a[13]=13 suma=25
thread 5 suma a[10]=10 suma=10
thread 5 suma a[11]=11 suma=21
thread 2 suma a[4]=4 suma=4
thread 2 suma a[5]=5 suma=9
thread 4 suma a[9]=9 suma=17
thread 1 suma a[2]=2 suma=2
thread 1 suma a[3]=3 suma=5
fuera de 'parallel for' suma=29
Valores por defecto: -2147483647 y el chunk: 0
```

El modo “monotonic” es el modo por defecto en la planificacion “static” mientras que para el resto se considera “nonmonotonic” . Monotonic pone a 1 el bit más significativo, el bit de sifno.

Por eso, si imprimimos por pantalla el tipo de planificación en modo hexadecimal en lugar de entero sale lo siguiente:

“static” (equivalente a “monotonic:static”) à -2147483647 (int) , 0x80000001 (hex)

“nonmonotonic:static” à 1 (int), 0x00000001 (hex)

“dynamic” (equivalente a “nonmonotonic:dynamic”) à 2 (int), 0x00000002 (hex)

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
int main(int argc, char **argv)
{
    int i, n=200, chunk, a[n], suma=0;

    int modifier;
    omp_sched_t kind;

    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);
    }

     #pragma omp single
        {
                omp_get_schedule(&kind,&modifier);
                printf("DENTRO DE LA REGION PARALELA\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("kind: : %d  | modifier: %d\n", kind,modifier);
        }

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

    omp_get_schedule(&kind,&modifier);
    printf("\nFUERA DE LA REGION PARALELA\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("kind: : %d  | modifier: %d\n", kind,modifier);
}
```

CAPTURAS DE PANTALLA:

```
┌──(icaro㉿kali)-[~/…/AC/Practica/Seminario3/Codigo]
└─$ export OMP_DYNAMIC=FALSE && export OMP_NUM_THREADS=3 && export OMP_THREAD_LIMIT=8

┌──(icaro㉿kali)-[~/…/AC/Practica/Seminario3/Codigo]
└─$ ./scheduled-clause 6 3
 thread 0 suma a[3]=3 suma=3
 thread 0 suma a[4]=4 suma=7
 thread 0 suma a[5]=5 suma=12
 thread 2 suma a[0]=0 suma=0
 thread 2 suma a[1]=1 suma=1
 thread 2 suma a[2]=2 suma=3
DENTRO DE LA REGION PARALELA
dyn-var: 0
nthreads-var: 3
thread-limit-var: 8
kind: : -2147483647  | modifier: 2
Fuera de 'parallel for' suma=12

FUERA DE LA REGION PARALELA
dyn-var: 0
nthreads-var: 3
thread-limit-var: 8
kind: : -2147483647  | modifier: 2

┌──(icaro㉿kali)-[~/…/AC/Practica/Seminario3/Codigo]
└─$ export OMP_SCHEDULE="guided,6"

┌──(icaro㉿kali)-[~/…/AC/Practica/Seminario3/Codigo]
└─$ ./scheduled-clause 6 3
 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 2 suma a[3]=3 suma=3
 thread 2 suma a[4]=4 suma=7
 thread 2 suma a[5]=5 suma=12
DENTRO DE LA REGION PARALELA
dyn-var: 0
nthreads-var: 3
thread-limit-var: 8
kind: : 3  | modifier: 6
Fuera de 'parallel for' suma=12

FUERA DE LA REGION PARALELA
dyn-var: 0
nthreads-var: 3
thread-limit-var: 8
kind: : 3  | modifier: 6
```

RESPUESTA:

Dentro y fuera de la region paralela los valores no cambian

Un aspecto importante a destacar es el la var run-shed var (kind y modifier). Esta variable solo funciona con runtime, no el resto. En sheduled usamos shedule(dynamic, chunk), es decir las variables que imprimamos en run-shed-var no van a estar relacionadas y por eso expulsa basura. SI actualizamos dicha var con expor omp_schedule actualizamos esas variables como en el ejemplo de abajo, sin embrago no tienen nada que ver en como ell shedule ha ejecutado la region paralela ya que no es runtime (kind 3 es guided no dynamic)

EL EJEMPLO NO PARALELIZO, AUNQUE DA IGUAL

TENGO QUE SEPARAR EL PARALEL Y ENGLOBARLO LUEGO AL FOR Y AL SINGLE

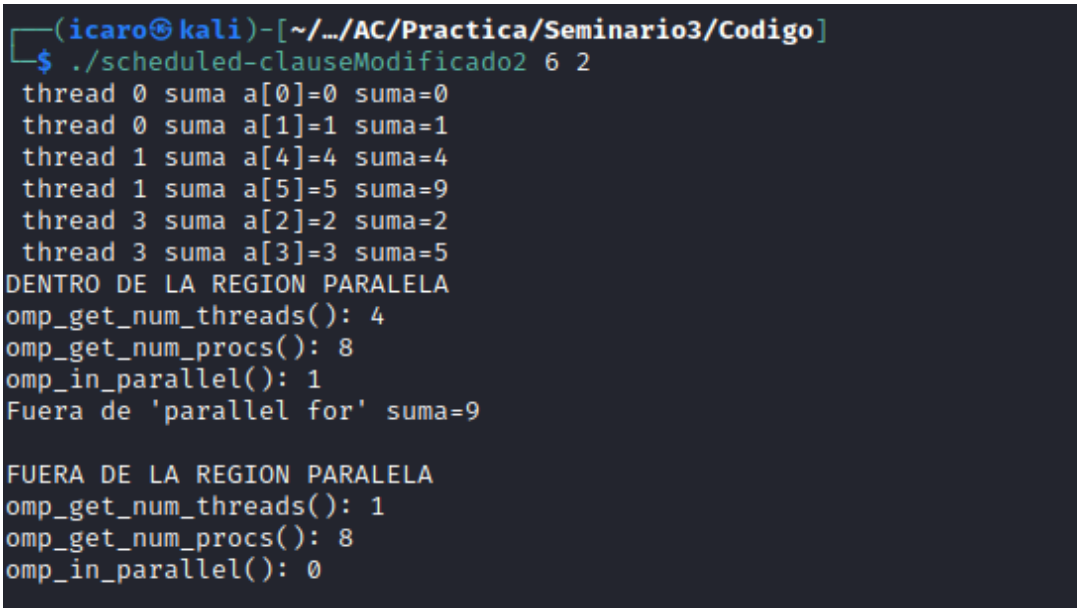
COMO EN EL EJERCICIO DE ABAJO

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-clauseModificado2.c`

```
int main(int argc, char **argv)
{
    int i, n=200, chunk, a[n], suma=0;

    int modifier;
    omp_sched_t kind;

    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 single
        {
                omp_get_schedule(&kind,&modifier);
                printf("DENTRO DE LA REGION PARALELA\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);

    omp_get_schedule(&kind,&modifier);
    printf("\nFUERA DE LA REGION PARALELA\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:



RESPUESTA:

En este ejercicio si cambian las variables

-omp_in_parallel: 1 si esta paralelizando 0 si no

-omp_get_num_threads: paralela obtiene el n.º de threads que se esta usando en esta region

secueencial: siempre es 1

-omp_get_num_procs: devuelve el n.º de procesadores disponibles para el programa en tiempo de ejecucion

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-clauseModificado3.c`

```
        #pragma omp single
           {
                   omp_get_schedule(&kind,&modifier); // tipo y moodificador
                   printf("\nDENTRO DE LA REGION PARALELA, ANTES DE MODIFICAR\n");

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

                   if (kind == omp_sched_static) printf("Kind es static\n");
                   else if (kind == omp_sched_dynamic) printf("Kind es dinamico\n");
                   else if (kind == omp_sched_guided) printf("Kind es guided\n");
                   else printf("Tipo es auto");


                   // modificamos

                   omp_set_dynamic(10);
                   omp_set_num_threads(4);
                   omp_set_schedule(1,2);


                   omp_get_schedule(&kind,&modifier); // tipo y moodificador
                   printf("DENTRO DE LA REGION PARALELA, LUEGO DE MODIFICAR\n");

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



           }
}

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

omp_get_schedule(&kind,&modifier); // tipo y moodificador
printf("DENTRO DE LA REGION PARALELA, ANTES DE MODIFICAR\n");

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

// modificamos

omp_set_dynamic(5);
omp_set_num_threads(2);
omp_set_schedule(2,1);


omp_get_schedule(&kind,&modifier); // tipo y moodificador
printf("DENTRO DE LA REGION PARALELA, LUEGO DE MODIFICAR\n");

printf("dyn_var: %d\n" , omp_get_dynamic());
printf("nthreads-var: %d\n" , omp_get_max_threads());
printf("run-shed-var: %d y modif: %d \n\n", kind, modifier);
```

CAPTURAS DE PANTALLA:

```
┌──(icaro㉿kali)-[~/…/AC/Practica/Seminario3/Codigo]
└─$ gcc -O2 -fopenmp scheduled-clauseModificado3.c -o scheduled-clauseModificado3

┌──(icaro㉿kali)-[~/…/AC/Practica/Seminario3/Codigo]
└─$ export OMP_SCHEDULE="dynamic"

┌──(icaro㉿kali)-[~/…/AC/Practica/Seminario3/Codigo]
└─$ ./scheduled-clauseModificado3 16 2
 thread 0 suma a[12]=12 suma=12
 thread 0 suma a[13]=13 suma=25
 thread 7 suma a[4]=4 suma=4
 thread 7 suma a[5]=5 suma=9
 thread 3 suma a[6]=6 suma=6
 thread 3 suma a[7]=7 suma=13
 thread 2 suma a[14]=14 suma=14
 thread 2 suma a[15]=15 suma=29
 thread 4 suma a[8]=8 suma=8
 thread 4 suma a[9]=9 suma=17
 thread 1 suma a[2]=2 suma=2
 thread 1 suma a[3]=3 suma=5
 thread 5 suma a[0]=0 suma=0
 thread 5 suma a[1]=1 suma=1
 thread 6 suma a[10]=10 suma=10
 thread 6 suma a[11]=11 suma=21

DENTRO DE LA REGION PARALELA, ANTES DE MODIFICAR
dyn_var: 0
nthreads-var: 8
run-shed-var: 2 y modif: 1

Kind es dinamico
DENTRO DE LA REGION PARALELA, LUEGO DE MODIFICAR
dyn_var: 1
nthreads-var: 4
run-shed-var: 1 y modif: 2

Fuera de 'parallel for' suma=29
DENTRO DE LA REGION PARALELA, ANTES DE MODIFICAR
dyn_var: 0
nthreads-var: 8
run-shed-var: 2 y modif: 1

Kind es dinamico
DENTRO DE LA REGION PARALELA, LUEGO DE MODIFICAR
dyn_var: 1
nthreads-var: 2
run-shed-var: 2 y modif: 1
```

RESPUESTA:

Al modificarse las variables de control toman los valores dados por las funciones de entorno en tiempos de ejecuciones. Dentro de la zona paralela podemos ver como se modifican los valores de estas variables, pero al salir se restauran los valores previos al cambio. Fuera de la region paralela , podemos ver, como tambien se modifican, pero se modifican para el resto del programa

por defecto = dinamic false, n.º threads = 8 y kind = 2 = dynamic, 1 chunk

dentro, modificamos dinamic a 10 → n.º par = true(por lo general pon 0 o 1) → ponemos 4 threads y cambiamos el schedule con 1 “static” y a 2 chunk, pero luego fuera del paralel vuelve a la normalidad y podemos cambiarle a otros valores.

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

7. 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. 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`

DESCOMPOSICIÓN DE DOMINIO:

CAPTURAS DE PANTALLA:

8. 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:

(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:

(c) ¿Qué alternativa cree que debería ofrecer mejores prestaciones? Razonar la respuesta.

RESPUESTA:

9. 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			