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 Merelo Molina
Grupo de prácticas: 2
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.

CAPTURA CÓDIGO FUENTE: `if_clause_modif.c`

```c
//Elena Merelo Molina

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

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

  if(argc != 3) {
    fprintf(stderr,"Número de argumentos incorrecto, ha de introducir iteraciones y threads\n");
    exit(-1);
  }

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

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

  /*Al imprimir omp_get_max_threads en mi pc sale 4, es el máximo número de threads
  con los que puedo trabajar.*/
  if( x > 4)
    x= 4;

  if (n>20)
    n=20;
```

```
  /*No se ejecuta en paralelo si n<= 4. Con default(none) hacemos que no se compartan
  o por defecto sean privadas las variables, de manera que nosotros mismos podamos
  establecer su visibilidad, poniendo sumalocal y tid privadas(con private(sumalocal, tid))
  y a, suma, n compartidas.*/
  #pragma omp parallel if(n > 4) num_threads(x) default(none) \
  private(sumalocal,tid) shared(a,suma,n)
  {
    sumalocal=0;
    tid=omp_get_thread_num();
    /*Con private(i) hacemos que cada hebra tenga un valor de i propio, schedule(static)
    hace que openMP divida el número de iteraciones de forma equitativa entre las hebras,
    aunque al poner nowait no han de esperarse unas a otras.*/
    #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 //Se espera a que todas las hebras terminen para imprimir el resultado
    #pragma omp master
    printf("thread master=%d imprime suma=%d\n",tid,suma);
  }
}
```

CAPTURAS DE PANTALLA:

Compilamos y generamos el ejecutable:

```
 2018-04-20 12:37:16  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestr
e/AC/Prácticas/BP3/src
→ gcc -fopenmp -O2 if_clause_modif.c -o ../bin/if_clause_modif
```

Y ejecutamos con un número de threads igual a 14 (en mi portátil hay 4 threads/core, por ello le he puesto que si se introduce un número de threads mayor que 4 ponga x a 4):

```
 2018-04-20 13:13:25  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestr
e/AC/Prácticas/BP3/src
→ ../bin/if_clause_modif 14 14
 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 1 suma de a[7]=7 sumalocal=22
 thread 2 suma de a[8]=8 sumalocal=8
 thread 2 suma de a[9]=9 sumalocal=17
 thread 2 suma de a[10]=10 sumalocal=27
 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 3 suma de a[11]=11 sumalocal=11
 thread 3 suma de a[12]=12 sumalocal=23
 thread 3 suma de a[13]=13 sumalocal=36
thread master=0 imprime suma=91
```

Se ve reflejado en que solo usa de las threads 0 a la 3, mientras que si por ejemplo establezco el número de threads a 2, entonces solo usa las threads 0 y 1:

```
 2018-04-20 13:14:04  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestr
e/AC/Prácticas/BP3/src
→ ../bin/if_clause_modif 14 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 0 suma de a[4]=4 sumalocal=10
 thread 0 suma de a[5]=5 sumalocal=15
 thread 0 suma de a[6]=6 sumalocal=21
 thread 1 suma de a[7]=7 sumalocal=7
 thread 1 suma de a[8]=8 sumalocal=15
 thread 1 suma de a[9]=9 sumalocal=24
 thread 1 suma de a[10]=10 sumalocal=34
 thread 1 suma de a[11]=11 sumalocal=45
 thread 1 suma de a[12]=12 sumalocal=57
 thread 1 suma de a[13]=13 sumalocal=70
thread master=0 imprime suma=91
```

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

Compilamos y generamos los ejecutables de todos los archivos:

```
 2018-04-24 17:25:03  elena in ~
→ cd Escritorio/University\ stuff/2º/2º\ Cuatrimestre/AC/Prácticas/BP3/src/

 2018-04-24 17:25:13  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestr
e/AC/Prácticas/BP3/src
→ gcc -fopenmp -O2 -lrt schedule-clause.c -o ../bin/schedule-clause

 2018-04-24 17:25:34  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestr
e/AC/Prácticas/BP3/src
→ gcc -fopenmp -O2 -lrt sched_dyn.c -o ../bin/sched_dyn

 2018-04-24 17:25:49  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestr
e/AC/Prácticas/BP3/src
→ gcc -fopenmp -O2 -lrt sched_guided.c -o ../bin/sched_guided
```

Resultados obtenidos:

```
 2018-04-24 17:35:44  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre
/AC/Prácticas/BP3/src
○ → export OMP_NUM_THREADS=2

 2018-04-24 17:35:51  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre
/AC/Prácticas/BP3/src
○ → ../bin/schedule-clause 1
 thread 0 suma a[0] suma=0
 thread 0 suma a[2] suma=2
 thread 0 suma a[4] suma=6
 thread 0 suma a[6] suma=12
 thread 0 suma a[8] suma=20
 thread 0 suma a[10] suma=30
 thread 0 suma a[12] suma=42
 thread 0 suma a[14] suma=56
 thread 1 suma a[1] suma=1
 thread 1 suma a[3] suma=4
 thread 1 suma a[5] suma=9
 thread 1 suma a[7] suma=16
 thread 1 suma a[9] suma=25
 thread 1 suma a[11] suma=36
 thread 1 suma a[13] suma=49
 thread 1 suma a[15] suma=64
Fuera de 'parallel for' suma=64
```

Visto que funciona correctamente, para recoger los datos ejecutamos los scripts:

```bash
#!/usr/bin/bash

export OMP_NUM_THREADS=2

for ((N= 1;N<= 4;N *= 2))
do
  echo "Chunk $N"
  ../bin/schedule-clause $N
done
```

```bash
#!/usr/bin/bash

export OMP_NUM_THREADS=2

for ((N= 1;N<= 4;N *= 2))
do
  echo "Chunk $N"
  ../bin/sched_dyn 16 $N
done
```

```bash
#!/usr/bin/bash

export OMP_NUM_THREADS=2

for ((N= 1;N<= 4;N *= 2))
do
  echo "Chunk $N"
  ../bin/sched_guided 16 $N
done
```

Tabla:

Iteración	Schedule_clause.c			sched_dyn.c			sched_guided.c		
	1	2	4	1	2	4	1	2	4
0	0	0	0	0	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0
2	0	1	0	0	1	0	0	0	0
3	1	1	0	0	1	0	0	0	0
4	0	0	1	0	0	1	0	0	0
5	1	0	1	0	0	1	0	0	0
6	0	1	1	0	0	1	0	0	0
7	1	1	1	0	0	1	0	0	0
8	0	0	0	0	0	0	1	1	1
9	1	0	0	0	0	0	1	1	1
10	0	1	0	0	0	0	1	1	1
11	1	1	0	0	0	0	1	1	1
12	0	0	1	0	0	0	0	0	0
13	1	0	1	0	0	0	0	0	0
14	0	1	1	0	1	0	1	1	0
15	1	1	1	0	1	0	1	1	0

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

Iteración	schedule_clause.c			sched_dyn.c			sched_guided.c		
	1	2	4	1	2	4	1	2	4
0	0	0	0	3	1	0	1	0	2
1	1	0	0	2	1	0	1	0	2
2	2	1	0	1	0	0	1	0	2
3	3	1	0	0	0	0	1	0	2
4	0	2	1	1	3	3	3	2	3
5	1	2	1	1	3	3	3	2	3
6	2	3	1	1	2	3	3	2	3
7	3	3	1	1	2	3	2	3	3
8	0	0	2	1	0	2	2	3	0
9	1	0	2	1	0	2	2	3	0
10	2	1	2	1	0	2	0	1	0
11	3	1	2	1	0	2	0	1	0
12	0	2	3	1	0	1	1	2	1
13	1	2	3	1	0	1	1	2	1
14	2	3	3	1	3	1	1	2	1
15	3	3	3	1	3	1	2	2	1

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

RESPUESTA:

Al ejecutar el bucle con schedule static se ve claramente que las threads siguen un orden a la hora de hacer las iteraciones, y cuando se pone el chunk a 1 las threads se organizan de una a una, mientras que con 2 la thread 0 por ejemplo hace dos iteraciones, luego la thread 1 hace otras dos iteraciones, y así sucesivamente. Cuando el chunk es 4 cada thread se hace cargo de cuatro iteraciones. Así pues, vemos que schedule(static) divide el número de iteraciones a realizar, en este caso 16, entre el tamaño del chunk (1, 2 y 4) y lo distribuye entre el número de threads que tengamos. Por eso se ve tan bien cuando tenemos 2 o 4 threads que si el chunk es 1 se alternan la thread 1 y la 0 al ejecutar las iteraciones, si el chunk es 2 las ejecutan de dos en dos, y si hubiéramos puesto un chunk de 16 aparecerían 16 ceros.

Por otro lado, no hay un orden particular en el que los chunks se distribuyen entre las threads, de hecho cambian cada vez que ejecutamos el programa, es arbitrario. Cuando el chunk es 1 el thread 1 hace más iteraciones, no se ve distribuyen de manera uniforme, pero cuando el chunk es 2 o 4, y el número de threads es 4, se pueden observar varias ocasiones en las que una misma thread ejecuta 2 o 4 iteraciones seguidas, aunque no es generalizado, depende de lo que el procesador decida.

El tipo schedule guided es parecido al schedule dynamic. OpenMP divide como en las otras las iteraciones entre chunks. La diferencia con el tipo dinámico es en el tamaño de los chunks. Éste es proporcional al número de iteraciones que no han sido asignadas a threads todavía dividido entre el número de threads. De esta manera el tamaño de los chunks va decreciendo.
Al poner schedule(guided, 2) decimos que el chunk mínimo es 2, aunque el chunk que contiene las últimas iteraciones puede tener un tamaño menor. En las tablas se parece más a lo que ha salido con schedule static, pero porque el número de cpus de mi portátil es 4, luego organiza los chunks de cuatro en cuatro como máximo si no se indica nada.

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?

CAPTURA CÓDIGO FUENTE: `sched_dyn_modif.c`

```c
//By: Elena Merelo Molina

#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 chunk y/o iteraciones\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;
```

```c
#pragma omp parallel
{
  #pragma omp for firstprivate(suma) \
                  lastprivate(suma) schedule(dynamic, chunk)
  for(i= 0; i< n; i++){
    suma += a[i];
    printf("Thread %d suma a 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 región paralela. dyn-var: %d, nthreads-var: %d, thread-limit-var: %d , run-sched-var:
    kind- %d, modifier- %d \n", omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit(), kind, modifier);
  }
}

printf("Fuera de parallel for suma= %d\n", suma);
omp_get_schedule(&kind, &modifier);
printf("dyn-var: %d, nthreads-var: %d, thread-limit-var: %d , run-sched-var:
kind- %d, modifier- %d \n", omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit(), kind, modifier);
}
```

Si lo compilamos y ejecutamos sin modificar ninguna variable de control:

```
 2018-05-02 16:28:43  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestr
e/AC/Prácticas/BP3/src
○ → gcc -lrt -O2 -fopenmp sched_dyn_modif.c -o ../bin/sched_dyn_modif

 2018-05-02 16:29:01  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestr
e/AC/Prácticas/BP3/src
○ → ../bin/sched_dyn_modif 7 3
Thread 3 suma a a[0]= 0 suma= 0
Thread 3 suma a a[1]= 1 suma= 1
Thread 3 suma a a[2]= 2 suma= 3
Thread 1 suma a a[6]= 6 suma= 6
Thread 0 suma a a[3]= 3 suma= 3
Thread 0 suma a a[4]= 4 suma= 7
Thread 0 suma a a[5]= 5 suma= 12
Dentro de la región paralela. dyn-var: 0, nthreads-var: 4, thread-limit-var: 214
7483647 , run-sched-var: kind- 2, modifier- 1
Fuera de parallel for suma= 6
dyn-var: 0, nthreads-var: 4, thread-limit-var: 2147483647 , run-sched-var: kind-
 2, modifier- 1
```

Si modificamos dyn-var y lo ponemos a false (aunque antes también lo estaba):

```
 2018-05-02 16:29:03  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestr
e/AC/Prácticas/BP3/src
○ → export OMP_DYNAMIC=FALSE

 2018-05-02 16:30:48  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestr
e/AC/Prácticas/BP3/src
○ → ../bin/sched_dyn_modif 7 3
Thread 0 suma a a[0]= 0 suma= 0
Thread 0 suma a a[1]= 1 suma= 1
Thread 0 suma a a[2]= 2 suma= 3
Thread 2 suma a a[3]= 3 suma= 3
Thread 2 suma a a[4]= 4 suma= 7
Thread 2 suma a a[5]= 5 suma= 12
Thread 3 suma a a[6]= 6 suma= 6
Dentro de la región paralela. dyn-var: 0, nthreads-var: 4, thread-limit-var: 214
7483647 , run-sched-var: kind- 2, modifier- 1
Fuera de parallel for suma= 6
dyn-var: 0, nthreads-var: 4, thread-limit-var: 2147483647 , run-sched-var: kind-
 2, modifier- 1
```

Si lo ponemos a true observamos que pasa de valer 0 a 1 en lo que muestra nuestro programa por pantalla:

```
 2018-05-02 16:30:52  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestr
e/AC/Prácticas/BP3/src
○ → export OMP_DYNAMIC=TRUE

 2018-05-02 16:31:17  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestr
e/AC/Prácticas/BP3/src
○ → ../bin/sched_dyn_modif 7 3
Thread 0 suma a a[3]= 3 suma= 3
Thread 0 suma a a[4]= 4 suma= 7
Thread 0 suma a a[5]= 5 suma= 12
Thread 2 suma a a[0]= 0 suma= 0
Thread 1 suma a a[6]= 6 suma= 6
Thread 2 suma a a[1]= 1 suma= 1
Thread 2 suma a a[2]= 2 suma= 3
Dentro de la región paralela. dyn-var: 1, nthreads-var: 4, thread-limit-var: 214
7483647 , run-sched-var: kind- 2, modifier- 1
Fuera de parallel for suma= 6
dyn-var: 1, nthreads-var: 4, thread-limit-var: 2147483647 , run-sched-var: kind-
 2, modifier- 1
```

Si cambiamos el número de threads:

```
 2018-05-02 16:31:18  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestr
e/AC/Prácticas/BP3/src
○ → export OMP_NUM_THREADS=8

 2018-05-02 16:31:45  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestr
e/AC/Prácticas/BP3/src
○ → ../bin/sched_dyn_modif 7 3
Thread 2 suma a a[0]= 0 suma= 0
Thread 1 suma a a[6]= 6 suma= 6
Thread 3 suma a a[3]= 3 suma= 3
Thread 3 suma a a[4]= 4 suma= 7
Thread 3 suma a a[5]= 5 suma= 12
Thread 2 suma a a[1]= 1 suma= 1
Thread 2 suma a a[2]= 2 suma= 3
Dentro de la región paralela. dyn-var: 1, nthreads-var: 8, thread-limit-var: 214
7483647 , run-sched-var: kind- 2, modifier- 1
Fuera de parallel for suma= 6
dyn-var: 1, nthreads-var: 8, thread-limit-var: 2147483647 , run-sched-var: kind-
 2, modifier- 1
```

Y el límite de threads a 2 con OMP_THREAD_LIMIT=2:

```
 2018-05-02 16:32:33  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestr
e/AC/Prácticas/BP3/src
○ → ../bin/sched_dyn_modif 7 3
Thread 0 suma a a[0]= 0 suma= 0
Thread 0 suma a a[1]= 1 suma= 1
Thread 0 suma a a[2]= 2 suma= 3
Thread 0 suma a a[6]= 6 suma= 9
Thread 1 suma a a[3]= 3 suma= 3
Thread 1 suma a a[4]= 4 suma= 7
Thread 1 suma a a[5]= 5 suma= 12
Dentro de la región paralela. dyn-var: 1, nthreads-var: 8, thread-limit-var: 2 ,
 run-sched-var: kind- 2, modifier- 1
Fuera de parallel for suma= 9
dyn-var: 1, nthreads-var: 8, thread-limit-var: 2 , run-sched-var: kind- 2, modif
ier- 1
```

Por último si cambiamos el schedule:

```
 2018-05-02 16:51:57  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestr
e/AC/Prácticas/BP3/src
○ → export OMP_SCHEDULE="static,2"

 2018-05-02 16:52:08  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestr
e/AC/Prácticas/BP3/src
○ → ../bin/sched_dyn_modif 7 3
Thread 0 suma a a[0]= 0 suma= 0
Thread 0 suma a a[1]= 1 suma= 1
Thread 0 suma a a[2]= 2 suma= 3
Thread 1 suma a a[3]= 3 suma= 3
Thread 1 suma a a[4]= 4 suma= 7
Thread 1 suma a a[5]= 5 suma= 12
Thread 0 suma a a[6]= 6 suma= 9
Dentro de la región paralela. dyn-var: 1, nthreads-var: 8, thread-limit-var: 2 ,
 run-sched-var: kind- 1, modifier- 2
Fuera de parallel for suma= 9
dyn-var: 1, nthreads-var: 8, thread-limit-var: 2 , run-sched-var: kind- 1, modif
ier- 2
```

RESPUESTA:

Se imprimen siempre los mismos valores dentro y fuera de la región paralela.

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.

CAPTURA CÓDIGO FUENTE: `sched_dyn_modif_2.c`

```
  #pragma omp parallel
  {
    #pragma omp for firstprivate(suma) \
                    lastprivate(suma) schedule(dynamic, chunk)
    for(i= 0; i< n; i++){
      suma += a[i];
      printf("Thread %d suma a 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 región paralela. dyn-var: %d, nthreads-var: %d, thread-limit-var: %d,
      run-sched-var: kind- %d, modifier- %d \n", omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit(), kind, modifier);
      printf("numprocs: %d, omp_in_parallel: %d", omp_get_num_procs(), omp_in_parallel());
    }
  }

  printf("\nFuera de parallel for suma= %d\n", suma);
  omp_get_schedule(&kind, &modifier);
  printf("dyn-var: %d, nthreads-var: %d, thread-limit-var: %d , run-sched-var: kind- %d,
  modifier- %d \n", omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit(), kind, modifier);
  printf("numprocs: %d, omp_in_parallel: %d", omp_get_num_procs(), omp_in_parallel());
}
```

Mostramos unicamente lo que hay a partir de #pragma omp parallel, lo anterior es igual que en el ejercicio 3.

CAPTURAS DE PANTALLA:
Al compilar y ejecutar:

```
 2018-05-02 16:53:21  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestr
e/AC/Prácticas/BP3/src
○ → gcc -lrt -O2 -fopenmp sched_dyn_modif_2.c -o ../bin/sched_dyn_modif_2

 2018-05-02 16:53:51  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestr
e/AC/Prácticas/BP3/src
○ → export OMP_DYNAMIC=true

 2018-05-02 16:53:54  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestr
e/AC/Prácticas/BP3/src
○ → ../bin/sched_dyn_modif_2 7 3
Thread 0 suma a a[0]= 0 suma= 0
Thread 0 suma a a[1]= 1 suma= 1
Thread 0 suma a a[2]= 2 suma= 3
Thread 0 suma a a[6]= 6 suma= 9
Thread 1 suma a a[3]= 3 suma= 3
Thread 1 suma a a[4]= 4 suma= 7
Thread 1 suma a a[5]= 5 suma= 12
Dentro de la región paralela. dyn-var: 1, nthreads-var: 8, thread-limit-var: 2 ,
 run-sched-var: kind- 1, modifier- 2
numprocs: 4, omp_in_parallel: 1
Fuera de parallel for suma= 9
dyn-var: 1, nthreads-var: 8, thread-limit-var: 2 , run-sched-var: kind- 1, modif
ier- 2
numprocs: 4, omp_in_parallel: 0
```

RESPUESTA:

Las variables de control del ejercicio anterior siguen sin cambiar al salir de la región paralela, y el número de procesadores también es el mismo. No obstante, dentro de la región paralela omp_in_parallel() es 1, está a true como tiene sentido, y fuera es 0, es false.

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.

CAPTURA CÓDIGO FUENTE: `sched_dyn_modif_3.c`

```c
//Elena Merelo Molina
#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, modifier;
  omp_sched_t kind;

  if(argc < 3){
    fprintf(stderr,"\nFalta chunk y/o iteraciones\n");
    exit(-1);
  }

  n= atoi(argv[1]);

  if(n > 200)
    n= 200;

  chunk= atoi(argv[2]);

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

```
  omp_get_schedule(&kind, &modifier);
  printf("Antes de modificar: dyn-var= %d, nthreads-var= %d, run-sched-var= kind %d,
  modifier %d\n", omp_get_dynamic(), omp_get_max_threads(), kind, modifier);

  omp_set_dynamic(0);
  omp_set_num_threads(12);
  omp_set_schedule(omp_sched_guided, 0);
  printf("Después de modificar: dyn-var= %d, nthreads-var= %d, run-sched-var= kind %d,
  modifier %d\n", omp_get_dynamic(), omp_get_max_threads(), kind, modifier);


  #pragma omp parallel for firstprivate(suma) \
lastprivate(suma) schedule(dynamic, chunk)
  for(i= 0; i< n; i++){
    suma += a[i];
    printf("Thread %d suma a a[%d]= %d suma= %d\n", omp_get_thread_num(), i, a[i], suma);
  }

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

CAPTURAS DE PANTALLA:

```
 2018-05-02 17:13:33  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestr
e/AC/Prácticas/BP3/src
○ → gcc -lrt -O2 -fopenmp sched_dyn_modif_3.c -o ../bin/sched_dyn_modif_3

 2018-05-02 17:15:12  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestr
e/AC/Prácticas/BP3/src
○ → ../bin/sched_dyn_modif_3 7 3
Antes de modificar: dyn-var= 1, nthreads-var= 8, run-sched-var= kind 1, modifier
 2
Después de modificar: dyn-var= 0, nthreads-var= 12, run-sched-var= kind 1, modif
ier 2
Thread 4 suma a a[0]= 0 suma= 0
Thread 4 suma a a[1]= 1 suma= 1
Thread 4 suma a a[2]= 2 suma= 3
Thread 2 suma a a[6]= 6 suma= 6
Thread 3 suma a a[3]= 3 suma= 3
Thread 3 suma a a[4]= 4 suma= 7
Thread 3 suma a a[5]= 5 suma= 12
Fuera de parallel for suma= 6
```

RESPUESTA:

Comprobamos así como dynamic se ha puesto a false y el número de threads a cambiado a 12.

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.

CAPTURA CÓDIGO FUENTE: `pmtv_seq.c`

```c
//Autora: Elena Merelo Molina
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#define PRINTF_RESULT // descomentar para que imprima el resultado

int main(int argc, char **argv){
  int n, i, j;
  struct timespec cgt1, cgt2;
  double ncgt;  //para tiempo de ejecución

  if(argc != 2){
    printf("Faltan número de filas y columnas de la matriz\n");
    exit(-1);
  }

  n= atoi(argv[1]);

  int **m, *v, *r;

  v= (int*) malloc(n*sizeof(int));
  r= (int*) malloc(n*sizeof(int));

  //Creamos una matriz usando un array de punteros a arrays
  m= (int**) malloc(n*sizeof(int*));
  for(i= 0; i< n; i++)
    m[i]= (int*) malloc(n*sizeof(int));

  if ((m == NULL) || (v == NULL) || (r == NULL)) {
    printf("Error en la reserva de espacio\n");
    exit(-2);
  }
//Inicializamos la matriz y los vectores
for(i= 0; i< n; i++){
  v[i]= i;
  r[i]= 0;
  for(j= 0; j< n; j++)
    m[i][j]= (i <= j) ? i+j : 0;
}

clock_gettime(CLOCK_REALTIME, &cgt1);

//Realizamos el producto de la matriz triangular m por el vector v, guardando el resultado en r
for(i= 0; i< n; i++)
  for(j= i; j< n; j++)
    r[i] += m[i][j] * v[j];

clock_gettime(CLOCK_REALTIME, &cgt2);

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

//Para comprobar que hace bien el producto
#ifdef PRINTF_RESULT
  printf("Matriz m: \n");
  for(i= 0; i< n; i++){
    for(j= 0; j< n; j++)
      printf("%d\t", m[i][j]);
    printf("\n");
  }

  printf("Vector v: \n");
  for(i= 0; i< n; i++)
    printf("%d ", v[i]);
```

```
    printf("Vector v: \n");
    for(i= 0; i< n; i++)
      printf("%d ", v[i]);

    printf("Vector resultante del producto\n");
    for(i= 0; i< n; i++)
      printf("%d ", r[i]);
  #endif

  printf("Tiempo de ejecución: %11.9f\t / r[0]= %d\t / r[%d]= %d\n", ncgt, r[0], n-1, r[n-1]);
  printf("Primera componente del resultado: %d, segunda: %d\n", r[0], r[n-1]);
  //Liberación de memoria
  for(int i= 0; i< n; i++)
    free(m[i]);

    free(m);
    free(r);
    free(v);

}
```

CAPTURAS DE PANTALLA:

```
 2018-05-02 18:08:59  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestr
e/AC/Prácticas/BP3/src
○ → gcc -lrt -O2 pmtv_seq.c -o ../bin/pmtv_seq.c

 2018-05-02 18:09:49  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestr
e/AC/Prácticas/BP3/src
○ → ../bin/pmtv_seq.c 10
Matriz m:
0       1       2       3       4       5       6       7       8       9
0       2       3       4       5       6       7       8       9       10
0       0       4       5       6       7       8       9       10      11
0       0       0       6       7       8       9       10      11      12
0       0       0       0       8       9       10      11      12      13
0       0       0       0       0       10      11      12      13      14
0       0       0       0       0       0       12      13      14      15
0       0       0       0       0       0       0       14      15      16
0       0       0       0       0       0       0       0       16      17
0       0       0       0       0       0       0       0       0       18
Vector v:
0 1 2 3 4 5 6 7 8 9 Vector resultante del producto
285 330 372 406 427 430 410 362 281 162 Tiempo de ejecución: 0.000001346
/ r[0]= 285      / r[9]= 162
Primera componente del resultado: 285, segunda: 162
```

Comprobado que funciona bien, comparamos con pmv_seq.c:

```
 2018-05-02 18:31:17  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestr
e/AC/Prácticas/BP2/src
○ → ../bin/pmv_seq 10000
Tiempo de ejecución: 0.056785881         / r[0]= -1724114088     / r[9999]= -403
15424
```

```
 2018-05-02 18:31:37  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestr
e/AC/Prácticas/BP3/src
○ → gcc -O2 -lrt pmtv_seq.c -o ../bin/pmtv_seq

 2018-05-02 18:31:57  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestr
e/AC/Prácticas/BP3/src
○ → ../bin/pmtv_seq 10000
Tiempo de ejecución: 0.040320201         / r[0]= -1724114088     / r[9999]= 1999
60002
```

Vemos que para una matriz de un tamaño considerable el producto de la matriz triangular superior por el vector es más eficiente, lo que nos permite reafirmarnos en el pensamiento de que éste tiene un orden de complejidad menor, pero ambos siguen siendo $O(n^2)$.

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.

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:

a) Como se ve al final, cuando ejecutamos schedule static sin especificar chunk toma 0, para dynamic coge 1 y para guided toma 1 también. Lo que he hecho se ve en el código, al llamar a la función omp_get_schedule(&kind, &chunk_size) guardamos en una variable el tipo de scheduling y el tamaño de chunk y los imprimimos. De esta forma, cuando luego en el script de atc_.._static hago export OMP_SCHEDULE=”static”, el tamaño que se imprime es 0, e igual con dynamic y guided, y los chunks obtenidos son los anteriores.

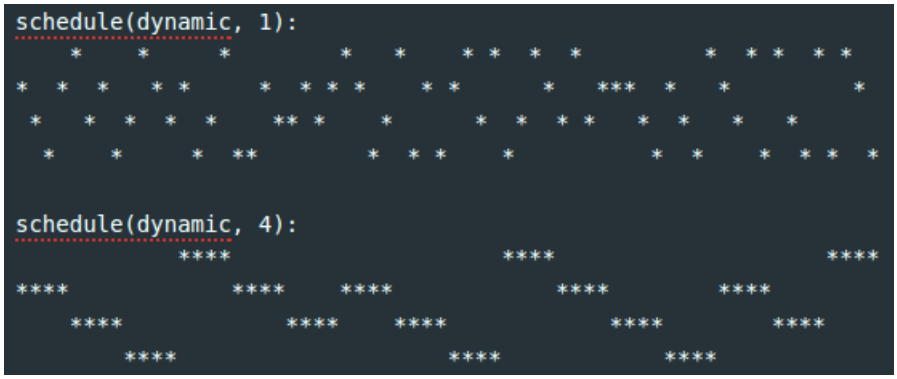
b) Al ser static OpenMP divide las iteraciones entre chunks del tamaño especificado (0, 1 y 64 respectivamente) y distribuye los chunks entre los threads en orden circular. Por lo tanto, para chunk= 0 se dividirán el número de iteraciones entre el de threads de manera equitativa, para chunk= 1 cada thread hará una iteración hasta que se acabe y para chunk= 64 cada thread ejecutará 64 iteraciones con sus operaciones.

c) No será uniforme.

Se ve más fácil con un dibujillo para ver qué ejecuta cada thread según el schedule y chunk:

Donde el número de estrellitas es el de iteraciones partido por el de threads. schedule(static, 64) será igual pero con 64 estrellitas.

En schedule(dynamic) cada thread ejecuta según se vea conveniente en el momento.



schedule(guided) reparte los chunks entre los threads como dynamic, con la diferencia de que el tamaño de chunk asignado es proporcional al número de iteraciones que no han sido asignadas entre el número de threads:

Son parecidos porque como hemos dicho el chunk_size por defecto para dynamic es 1.
No se mantiene orden en el que los chunks se distribuyen a las threads, se ejecuta diferente de una vez para otra, y cada thread ejecuta un chunk de iteraciones y luego demanda otro chunk hasta que no hay más.

CAPTURA CÓDIGO FUENTE: `pmtv_omp.c`
`Edit:` Luego hice un pequeño cambio de manera que el número de threads fuera el máximo posible donde se ejecutara, esto es, declaré una variable int x= omp_get_max_threads() y en #pragma omp parallel puse num_threads(x), y quité #pragma_omp_atomic.

```
Autora: Elena Merelo Molina.*/

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

#define PRINTF_MATRIX // descomentar para imprimir la matriz triangular
#define PRINTF_VECTOR // descomentar para imprimir el vector que va a ser multiplicado
#define PRINTF_RESULT // descomentar para imprimir el vector resultante del producto
#define PRINTF_SCHED // descomentar para imprimir el tipo de scheduling y el chunk usado

int main(int argc, char **argv){
  int n, i, j, suma_local;
  struct timespec cgt1, cgt2;
  double ncgt;  //para tiempo de ejecución

  if(argc != 2){
    printf("Faltan número de filas y columnas de la matriz\n");
    exit(-1);
  }

  n= atoi(argv[1]);

  int **m, *v, *r;

  v= (int*) malloc(n*sizeof(int));
  r= (int*) malloc(n*sizeof(int));
```

```
//Creamos una matriz usando un array de punteros a arrays
m= (int**) malloc(n*sizeof(int*));
for(i= 0; i< n; i++)
  m[i]= (int*) malloc(n*sizeof(int));

  if ((m == NULL) || (v == NULL) || (r == NULL)) {
    printf("Error en la reserva de espacio\n");
    exit(-2);
  }


//Inicializamos la matriz y los vectores paralelamente
for(i= 0; i< n; i++){
  v[i]= i;
  r[i]= 0;
  #pragma omp parallel for
  for(j= 0; j< n; j++)
    m[i][j]= (i <= j) ? i+j : 0;
}

clock_gettime(CLOCK_REALTIME, &cgt1);

/*Realizamos el producto de la matriz triangular m por el vector v, guardando
el resultado en r. Ponemos schedule a runtime para que desde la terminal podamos
decidir cómo queremos que se reparta el trabajo entre las threads.*/
```

```
#pragma omp parallel num_threads(4) private(i, j, suma_local)
{
   #pragma omp for schedule(runtime)
    for(i= 0; i< n; i++){
      suma_local= 0;
      for(j= i; j< n; j++)
        suma_local += m[i][j] * v[j];

        #pragma omp atomic
        r[i] += suma_local;
    }
    #ifdef PRINTF_SCHED
      //Para que lo ejecute un solo thread
      #pragma omp single
      {
        omp_sched_t kind;
        int chunk_size;

        omp_get_schedule(&kind, &chunk_size);

        printf("\nTipo de scheduling usado: %d (1= static, 2= dynamic, 3= guided, 4= auto), chunk_size: %d", kind, chunk_size);
      }
    #endif
}

clock_gettime(CLOCK_REALTIME, &cgt2);

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

Éste es el cuerpo principal del programa. Empezamos estableciendo el número de threads a 4 como indica el enunciado y ponemos privadas i, j y suma_local de manera que cada thread tenga la suya propia y se realice correctamente el producto fila por fila, no se líen unas con otras y sumen donde no deben. Establecemos schedule como runtime para que así se decida el scheduling en tiempo de ejecución, y desde la terminal podamos cambiarlo con OMP_SCHEDULE. Posteriormente, #pragma omp atomic es para que se almacene correctamente en el vector solución el producto de cada fila por el vector y #pragma omp single para que el tipo de scheduling elegido y el tamaño de chunk lo imprima un único thread. En un momento vemos como funciona.

```
  //Para comprobar que hace bien el producto
  #ifdef PRINTF_MATRIX
    printf("\nMatriz m:\n");
    for(i= 0; i< n; i++){
      for(j= 0; j< n; j++)
        printf("%d\t", m[i][j]);
      printf("\n");
    }
  #endif

  #ifdef PRINTF_VECTOR
    printf("\nVector v: ");
    for(i= 0; i< n; i++)
      printf("%d ", v[i]);
  #endif

  #ifdef PRINTF_RESULT
    printf("\nVector resultante del producto: ");
    for(i= 0; i< n; i++)
      printf("%d ", r[i]);
  #endif

  printf("\nTiempo de ejecución: %11.9f\t / r[0]= %d\t / r[%d]= %d\n", ncgt, r[0], n-1, r[n-1]);

  //Liberación de memoria
  for(int i= 0; i< n; i++)
    free(m[i]);

    free(m);
    free(r);
    free(v);
}
```

Compilamos y ejecutamos:

```
 2018-05-09 16:35:36  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre/AC/Prácticas/BP3/src
○ → gcc -O2 -fopenmp -lrt pmtv_omp.c -o ../bin/pmtv_omp

 2018-05-09 16:56:17  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre/AC/Prácticas/BP3/src
○ → ../bin/pmtv_omp 4

Tipo de scheduling usado: 2 (1= static, 2= dynamic, 3= guided, 4= auto), chunk_size: 1
Matriz m:
0       1       2       3
0       2       3       4
0       0       4       5
0       0       0       6

Vector v: 0 1 2 3
Vector resultante del producto: 14 20 23 18
Tiempo de ejecución: 0.000072243         / r[0]= 14      / r[3]= 18
```

Comprobado que funciona bien, comentamos PRINTF_RESULT, PRINTF_MATRIX, PRINTF_VECTOR de manera que solo imprima el tiempo de ejecución, el scheduling y el tamaño de chunk.

Observamos como, por defecto OMP_SCHEDULE es dynamic y el tamaño de chunk 1. y el resultado está bien. Como calculamos en el programa multiplo_64.cpp, el primer múltiplo de 64 y 12 mayor de 15360 es 15552 , éste es el tamaño del vector que escogemos:

```
 2018-05-09 17:46:50  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre/AC/Prácticas/BP3/src
○ → g++ multiplo_64.cpp -o ../bin/multiplo_64

 2018-05-09 17:51:49  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre/AC/Prácticas/BP3/src
○ → ../bin/multiplo_64
15552
```

DESCOMPOSICIÓN DE DOMINIO:

TABLA RESULTADOS, SCRIPT Y GRÁFICA atcgrid

He hecho tres scripts, uno para cada tipo de schedule, que son iguales cambiando únicamente static por dynamic y así:

```
#!/usr/bin/bash

#Se asigna al trabajo el nombre pmv
#PBS -N pmv_parallel_b
#Se asigna al trabajo la cola ac
#PBS -q ac
#Se imprime información del trabajo usando variables de entorno de PBS.


echo "Id. usuario del trabajo: $PBS_O_LOGNAME"
echo "Id. del trabajo: $PBS_JOBID"
echo "Nombre del trabajo especificando 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

clear

export OMP_SCHEDULE="static"
$PBS_O_WORKDIR/pmtv_omp 15552

export OMP_SCHEDULE="static,1"
$PBS_O_WORKDIR/pmtv_omp 15552

export OMP_SCHEDULE="static, 64"
$PBS_O_WORKDIR/pmtv_omp 15552
```

Los mandamos al frontend junto con el archivo binario creado anteriormente:

```
sftp> put ./bin/pmtv_omp
Uploading ./bin/pmtv_omp to /home/E2estudiante10/pmtv_omp
./bin/pmtv_omp                                100%   13KB  13.4KB/s   00:00
sftp> put ./sc
schedule.png    screenshots/    scripts/
sftp> put ./scr
screenshots/    scripts/
sftp> put ./scripts/atc*
Uploading ./scripts/atcgrid_pmtv_guided.sh to /home/E2estudiante10/atcgrid_pmtv_
guided.sh
./scripts/atcgrid_pmtv_guided.sh              100%  717     0.7KB/s   00:00
Uploading ./scripts/atcgrid_pmtv_omp_dynamic to /home/E2estudiante10/atcgrid_pmt
v_omp_dynamic
./scripts/atcgrid_pmtv_omp_dynamic            100%    0     0.0KB/s   00:00
Uploading ./scripts/atcgrid_pmtv_omp_dynamic.sh to /home/E2estudiante10/atcgrid_
pmtv_omp_dynamic.sh
./scripts/atcgrid_pmtv_omp_dynamic.sh         100%  720     0.7KB/s   00:00
Uploading ./scripts/atcgrid_pmtv_omp_static.sh to /home/E2estudiante10/atcgrid_p
mtv_omp_static.sh
./scripts/atcgrid_pmtv_omp_static.sh          100%  717     0.7KB/s   00:00
```

```
[ElenaMereloMolina E2estudiante10@atcgrid:~/BP3_ejer7] 2018-05-09 miércoles
$ls
atcgrid_pmtv_guided.sh    atcgrid_pmtv_omp_dynamic.sh  pmtv_omp
atcgrid_pmtv_omp_dynamic  atcgrid_pmtv_omp_static.sh
```

Y los ejecutamos (el nombre de los archivos obtenidos se me olvidó cambiarlo en el script pero bueno, como lo que importa es lo obtenido da igual):

```
[ElenaMereloMolina E2estudiante10@atcgrid:~/BP3_ejer7] 2018-05-09 miércoles
$qsub atcgrid_pmtv_omp_static.sh -q ac
77566.atcgrid
[ElenaMereloMolina E2estudiante10@atcgrid:~/BP3_ejer7] 2018-05-09 miércoles
$qsub atcgrid_pmtv_omp_dynamic.sh -q ac
77567.atcgrid
[ElenaMereloMolina E2estudiante10@atcgrid:~/BP3_ejer7] 2018-05-09 miércoles
$qsub atcgrid_pmtv_omp_guided.sh -q ac
qsub: script file 'atcgrid_pmtv_omp_guided.sh' cannot be loaded - No such file o
r directory
[ElenaMereloMolina E2estudiante10@atcgrid:~/BP3_ejer7] 2018-05-09 miércoles
$qsub atcgrid_pmtv_guided.sh -q ac
77568.atcgrid
```

Para static:

```
Tipo de scheduling usado: 1 (1= static, 2= dynamic, 3= guided, 4= auto), chunk_s
ize: 0
Tiempo de ejecución: 0.028281652         / r[0]= -424754656      / r[15551]= 483
667202

Tipo de scheduling usado: 1 (1= static, 2= dynamic, 3= guided, 4= auto), chunk_s
ize: 1
Tiempo de ejecución: 0.022551383         / r[0]= -424754656      / r[15551]= 483
667202

Tipo de scheduling usado: 1 (1= static, 2= dynamic, 3= guided, 4= auto), chunk_s
ize: 64
Tiempo de ejecución: 0.023298709         / r[0]= -424754656      / r[15551]= 483
667202
[ElenaMereloMolina E2estudiante10@atcgrid:~/BP3_ejer7] 2018-05-09 miércoles
```

Para dynamic:

```
Tipo de scheduling usado: 2 (1= static, 2= dynamic, 3= guided, 4= auto), chunk_s
ize: 1
Tiempo de ejecución: 0.021637203         / r[0]= -424754656      / r[15551]= 483
667202

Tipo de scheduling usado: 2 (1= static, 2= dynamic, 3= guided, 4= auto), chunk_s
ize: 1
Tiempo de ejecución: 0.022310745         / r[0]= -424754656      / r[15551]= 483
667202

Tipo de scheduling usado: 2 (1= static, 2= dynamic, 3= guided, 4= auto), chunk_s
ize: 64
Tiempo de ejecución: 0.021018442         / r[0]= -424754656      / r[15551]= 483
667202
[ElenaMereloMolina E2estudiante10@atcgrid:~/BP3_ejer7] 2018-05-09 miércoles
```

Y finalmente para guided:

```
Tipo de scheduling usado: 3 (1= static, 2= dynamic, 3= guided, 4= auto), chunk_s
ize: 1
Tiempo de ejecución: 0.103859035         / r[0]= -424754656      / r[15551]= 483
667202
```

No imprime más, da el siguiente error:

```
[ElenaMereloMolina E2estudiante10@atcgrid:~/BP3_ejer7] 2018-05-09 miércoles
$cat pmv_parallel_b.e77568
clear: terminal attributes: No such device or address

=>> PBS: job killed: walltime 65 exceeded limit 60
```

Cambié el tamaño de vector a 15360 y el tamaño de chunk a 32 pero seguía dando el mismo error, así que lo restablecí como estaba al principio y ya está. Por ello lo separé en tres scripts, para que no se mandara tanto trabajo al frontend y no lo detuviera.

Para la opción schedule(guided) obtenemos:

```
Tipo de scheduling usado: 3 (1= static, 2= dynamic, 3= guided, 4= auto), chunk_s
ize: 1
Tiempo de ejecución: 0.083031679         / r[0]= -424754656      / r[15551]= 483
667202
[ElenaMereloMolina E2estudiante10@atcgrid:~/BP3_ejer7] 2018-05-10 jueves
```

Para schedule(guided,1):

```
Tipo de scheduling usado: 3 (1= static, 2= dynamic, 3= guided, 4= auto), chunk_s
ize: 1
Tiempo de ejecución: 0.096270130         / r[0]= 120924576       / r[15551]= 311
02
[ElenaMereloMolina E2estudiante10@atcgrid:~/BP3_ejer7] 2018-05-10 jueves
```

Y finalmente para schedule(guided, 64):

```
Tipo de scheduling usado: 3 (1= static, 2= dynamic, 3= guided, 4= auto), chunk_s
ize: 64
Tiempo de ejecución: 0.094458821         / r[0]= 120924576       / r[15551]= 311
02
[ElenaMereloMolina E2estudiante10@atcgrid:~/BP3_ejer7] 2018-05-10 jueves
```

Luego los mandamos otra vez para ver la diferencia de tiempos, y rellenamos la tabla. Ésta segunda vez que lo hice se ve que más gente estaba usando atcgrid y me mataba los scripts antes de que terminaran, por lo que tuve que divirlos en tres también. Ésta segunda vez que los mandé sí cambié el nombre del archivo de salida:

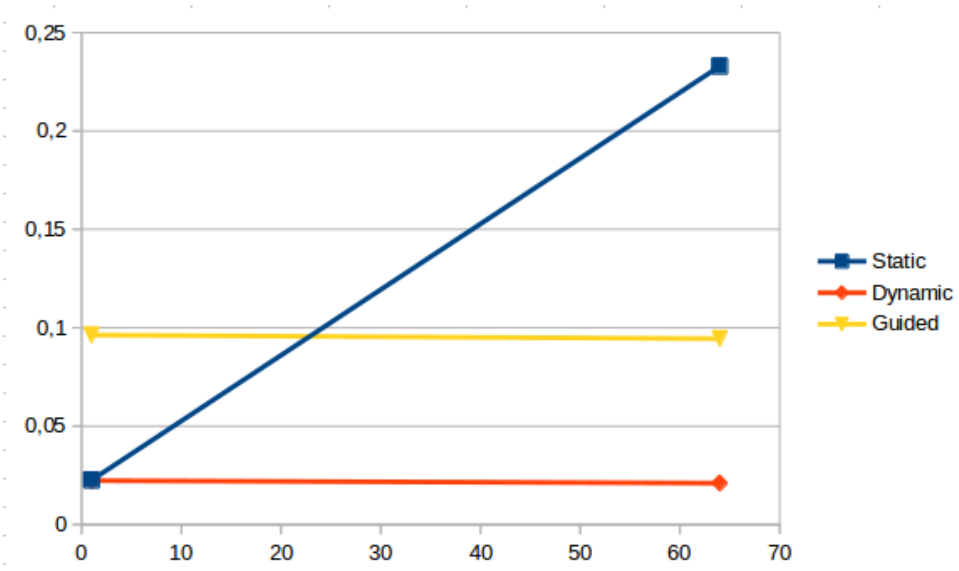
Tabla 1 . Tiempos de ejecución de la versión paralela del producto de una matriz triangular por un vector r para vectores de tamaño N= , 12 threads

Chunk	Static	Dynamic	Guided
por defecto	0.02828165	0.021637203	0.083031679
1	0.022551383	0.022310745	0.096270130
64	0.23298709	0.021018442	0.094458821
Chunk	**Static**	**Dynamic**	**Guided**
por defecto	0.073060231	0.254551499	0.100064563
1	0.074859923	0.296253048	0.032501339
64	0.101524572	0.076336133	0.115689217

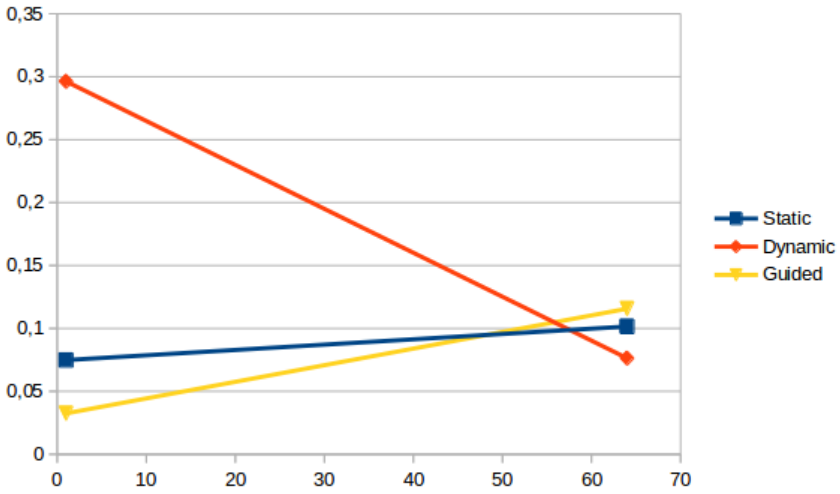
En la siguiente imagen vemos todos los archivos generados en el frontend para este ejercicio:

```
[ElenaMereloMolina E2estudiante10@atcgrid:~/BP3_ejer7] 2018-05-10 jueves
$ls
atcgrid_pmtv_dynamic_1.sh     pmtv_guided_1.e77905    pmv_parallel_b.e77567
atcgrid_pmtv_dynamic_2.sh     pmtv_guided_1.o77847    pmv_parallel_b.e77568
atcgrid_pmtv_dynamic_3.sh     pmtv_guided_1.o77905    pmv_parallel_b.e77574
atcgrid_pmtv_guided_1.sh      pmtv_guided_2.e77860    pmv_parallel_b.e77581
atcgrid_pmtv_guided_2.sh      pmtv_guided_2.e77913    pmv_parallel_b.e77582
atcgrid_pmtv_guided_3.sh      pmtv_guided_2.o77860    pmv_parallel_b.e77585
atcgrid_pmtv_guided.sh        pmtv_guided_2.o77913    pmv_parallel_b.e77862
atcgrid_pmtv_omp_dynamic.sh   pmtv_guided_3.e77861    pmv_parallel_b.e77863
atcgrid_pmtv_omp_static.sh    pmtv_guided_3.e77916    pmv_parallel_b.e77876
atcgrid_pmtv_static_1.sh      pmtv_guided_3.o77861    pmv_parallel_b.o77566
atcgrid_pmtv_static_2.sh      pmtv_guided_3.o77916    pmv_parallel_b.o77567
atcgrid_pmtv_static_3.sh      pmtv_omp                pmv_parallel_b.o77568
pmtv_dynamic_1.e77897         pmtv_static_1.e77880    pmv_parallel_b.o77574
pmtv_dynamic_1.o77897         pmtv_static_1.o77880    pmv_parallel_b.o77581
pmtv_dynamic_2.e77901         pmtv_static_2.e77892    pmv_parallel_b.o77582
pmtv_dynamic_2.o77901         pmtv_static_2.o77892    pmv_parallel_b.o77585
pmtv_dynamic_3.e77903         pmtv_static_3.e77894    pmv_parallel_b.o77862
pmtv_dynamic_3.o77903         pmtv_static_3.o77894    pmv_parallel_b.o77863
pmtv_guided_1.e77847          pmv_parallel_b.e77566   pmv_parallel_b.o77876
```

Gráfica de la tabla primera:



Y de la segunda tabla:



Vemos como, en general, cuando se aumenta mucho el chunk también lo hace el tiempo de ejecución, y en la primera tabla tarda menos el dynamic, mientras que en la segunda lo hace el static. Esto está relacionado con el hecho de que la segunda vez que he ejecutado los schedule static y dynamic estaban en scripts separados, he tenido que hacer esto al estar atcgrid muy concurrido, por eso salen tan diferentes, pero se coge la idea.

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

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

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.

CAPTURA CÓDIGO FUENTE: `pmm_seq.c`

```c
//Autora: Elena Merelo Molina
#include <stdlib.h>
#include <stdio.h>
#include <time.h>

#define PRINTF_MATRIX_1 // descomentar para imprimir la matriz 1
#define PRINTF_MATRIX_2 // descomentar para imprimir la matriz 2
#define PRINTF_RESULT // descomentar para imprimir la matriz resultado del producto

int main(int argc, char **argv){
  int tam, i, j, k;
  struct timespec cgt1, cgt2;
  double ncgt;  //para tiempo de ejecución

  if(argc != 2){
    printf("Faltan número de filas y columnas de la matriz\n");
    exit(-1);
  }

  tam= atoi(argv[1]);

  int **m, **n, **r;

  //Reservamos espacio para las tres matrices
  m= (int**) malloc(tam*sizeof(int*));
  n= (int**) malloc(tam*sizeof(int*));
  r= (int**) malloc(tam*sizeof(int*));
```

```c
for(i= 0; i< tam; i++){
  m[i]= (int*) malloc(tam*sizeof(int));
  n[i]= (int*) malloc(tam*sizeof(int));
  r[i]= (int*) malloc(tam*sizeof(int));
}

  if ((m == NULL) || (n == NULL) || (r == NULL)) {
    printf("Error en la reserva de espacio\n");
    exit(-2);
  }


//Inicializamos las matrices
for(i= 0; i< tam; i++)
  for(j= 0; j< tam; j++){
    m[i][j]= i+j;
    n[i][j]= j;
    r[i][j]= 0;
}

clock_gettime(CLOCK_REALTIME, &cgt1);
```

```
//Realizamos el producto de la matriz triangular m por el vector v, guardando el resultado en r
int suma_local= 0;

for(i= 0; i< tam; i++)
  for(j= 0; j< tam; j++)
    for(k= 0; k< tam; k++)
      r[i][j] += m[i][k] * n[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));

//Para comprobar que hace bien el producto
#ifdef PRINTF_MATRIX_1
  printf("Matriz m: \n");
  for(i= 0; i< tam; i++){
    for(j= 0; j< tam; j++)
      printf("%d\t", m[i][j]);
    printf("\n");
  }
#endif

#ifdef PRINTF_MATRIX_2
  printf("Matriz n: \n");
  for(i= 0; i< tam; i++){
    for(j= 0; j< tam; j++)
      printf("%d\t", n[i][j]);
    printf("\n");
  }
```

```
  #ifdef PRINTF_RESULT
    printf("Matriz resultante del producto\n");
    for(i= 0; i< tam; i++){
      for(j= 0; j< tam; j++)
        printf("%d\t", r[i][j]);
      printf("\n");
    }
  #endif

  printf("Tiempo de ejecución: %11.9f\t / r[0][0]= %d\t / r[%d][%d]= %d\n", ncgt, r[0][0], tam-1, tam-1, r[tam-1][tam-1]);

  //Liberación de memoria
  for(int i= 0; i< tam; i++){
    free(m[i]);
    free(n[i]);
    free(r[i]);
  }

  free(m);
  free(n);
  free(r);
}
```

CAPTURAS DE PANTALLA:

Compilamos y ejecutamos:

```
 2018-05-10 11:07:38  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre
/AC/Prácticas/BP3/src
○ → gcc -O2 -lrt pmm_seq.c -o ../bin/pmm_seq

 2018-05-10 11:12:52  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre
/AC/Prácticas/BP3/src
○ → ../bin/pmm_seq 3
Matriz m:
0       1       2
1       2       3
2       3       4
Matriz n:
0       1       2
0       1       2
0       1       2
Matriz resultante del producto
0       3       6
0       6       12
0       9       18
Tiempo de ejecución: 0.000001347         / r[0][0]= 0    / r[2][2]= 18
```

Incluso para n= 10 tarda poquito:

```
 2018-05-10 11:25:06  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre
/AC/Prácticas/BP3/src
○ → ../bin/pmm_seq 10
Matriz m:
0       1       2       3       4       5       6       7       8       9
1       2       3       4       5       6       7       8       9       10
2       3       4       5       6       7       8       9       10      11
3       4       5       6       7       8       9       10      11      12
4       5       6       7       8       9       10      11      12      13
5       6       7       8       9       10      11      12      13      14
6       7       8       9       10      11      12      13      14      15
7       8       9       10      11      12      13      14      15      16
8       9       10      11      12      13      14      15      16      17
9       10      11      12      13      14      15      16      17      18
Matriz n:
0       1       2       3       4       5       6       7       8       9
0       1       2       3       4       5       6       7       8       9
0       1       2       3       4       5       6       7       8       9
0       1       2       3       4       5       6       7       8       9
0       1       2       3       4       5       6       7       8       9
0       1       2       3       4       5       6       7       8       9
0       1       2       3       4       5       6       7       8       9
0       1       2       3       4       5       6       7       8       9
0       1       2       3       4       5       6       7       8       9
0       1       2       3       4       5       6       7       8       9
Matriz resultante del producto
0       45      90      135     180     225     270     315     360     405
0       55      110     165     220     275     330     385     440     495
0       65      130     195     260     325     390     455     520     585
0       75      150     225     300     375     450     525     600     675
0       85      170     255     340     425     510     595     680     765
0       95      190     285     380     475     570     665     760     855
0       105     210     315     420     525     630     735     840     945
0       115     230     345     460     575     690     805     920     1035
0       125     250     375     500     625     750     875     1000    1125
0       135     270     405     540     675     810     945     1080    1215
Tiempo de ejecución: 0.000000933          / r[0][0]= 0    / r[9][9]= 1215
```

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).

DESCOMPOSICIÓN DE DOMINIO:

CAPTURA CÓDIGO FUENTE: `pmm_omp.c`

```c
//Autora: Elena Merelo Molina
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <omp.h>

#define PRINTF_MATRIX_1 // descomentar para imprimir la matriz 1
#define PRINTF_MATRIX_2 // descomentar para imprimir la matriz 2
#define PRINTF_RESULT // descomentar para imprimir la matriz resultado del producto

int main(int argc, char **argv){
  int tam, i, j, k;
  struct timespec cgt1, cgt2;
  double ncgt;  //para tiempo de ejecución

  if(argc != 2){
    printf("Faltan número de filas y columnas de la matriz\n");
    exit(-1);
  }

  tam= atoi(argv[1]);

  int **m, **n, **r;

  //Reservamos espacio para las tres matrices
  m= (int**) malloc(tam*sizeof(int*));
  n= (int**) malloc(tam*sizeof(int*));
  r= (int**) malloc(tam*sizeof(int*));
```

```
for(i= 0; i< tam; i++){
  m[i]= (int*) malloc(tam*sizeof(int));
  n[i]= (int*) malloc(tam*sizeof(int));
  r[i]= (int*) malloc(tam*sizeof(int));
}

  if ((m == NULL) || (n == NULL) || (r == NULL)) {
    printf("Error en la reserva de espacio\n");
    exit(-2);
  }


//Inicializamos las matrices
#pragma omp parallel for private(i, j)
for(i= 0; i< tam; i++)
  for(j= 0; j< tam; j++){
    m[i][j]= i+j;
    n[i][j]= j;
    r[i][j]= 0;
}

clock_gettime(CLOCK_REALTIME, &cgt1);
```

```
//Realizamos el producto de la matriz triangular m por el vector v, guardando el resultado en r
int suma_local= 0;
#pragma omp parallel private(i,j,k)
{
  for(i= 0; i< tam; i++){
    for(j= 0; j< tam; j++){
    #pragma omp for reduction(+:suma_local)
      for(k= 0; k< tam; k++)
        suma_local += m[i][k] * n[k][j];

    #pragma omp single
    {
      r[i][j]= suma_local;
      suma_local= 0;
    }
  }
  }
}
clock_gettime(CLOCK_REALTIME, &cgt2);

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

El resto del programa, correspondiente a la impresión por pantalla de las matrices y liberación de memoria, es igual al de pmm_seq.c

Compilamos y ejecutamos:

```
 2018-05-10 11:25:14  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre
/AC/Prácticas/BP3/src
○ → gcc -O2 -lrt -fopenmp pmm_omp.c -o ../bin/pmm_omp

 2018-05-10 11:30:28  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre
/AC/Prácticas/BP3/src
○ → ../bin/pmm_omp 3
Matriz m:
0       1       2
1       2       3
2       3       4
Matriz n:
0       1       2
0       1       2
0       1       2
Matriz resultante del producto
0       3       6
0       6       12
0       9       18
Tiempo de ejecución: 0.000038342         / r[0][0]= 0    / r[2][2]= 18
```

```
 2018-05-10 11:30:46  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre
/AC/Prácticas/BP3/src
○ → ../bin/pmm_omp 10
Matriz m:
0       1       2       3       4       5       6       7       8       9
1       2       3       4       5       6       7       8       9       10
2       3       4       5       6       7       8       9       10      11
3       4       5       6       7       8       9       10      11      12
4       5       6       7       8       9       10      11      12      13
5       6       7       8       9       10      11      12      13      14
6       7       8       9       10      11      12      13      14      15
7       8       9       10      11      12      13      14      15      16
8       9       10      11      12      13      14      15      16      17
9       10      11      12      13      14      15      16      17      18
Matriz n:
0       1       2       3       4       5       6       7       8       9
0       1       2       3       4       5       6       7       8       9
0       1       2       3       4       5       6       7       8       9
0       1       2       3       4       5       6       7       8       9
0       1       2       3       4       5       6       7       8       9
0       1       2       3       4       5       6       7       8       9
0       1       2       3       4       5       6       7       8       9
0       1       2       3       4       5       6       7       8       9
0       1       2       3       4       5       6       7       8       9
0       1       2       3       4       5       6       7       8       9
Matriz resultante del producto
0       45      90      135     180     225     270     315     360     405
0       55      110     165     220     275     330     385     440     495
0       65      130     195     260     325     390     455     520     585
0       75      150     225     300     375     450     525     600     675
0       85      170     255     340     425     510     595     680     765
0       95      190     285     380     475     570     665     760     855
0       105     210     315     420     525     630     735     840     945
0       115     230     345     460     575     690     805     920     1035
0       125     250     375     500     625     750     875     1000    1125
0       135     270     405     540     675     810     945     1080    1215
Tiempo de ejecución: 0.000271676         / r[0][0]= 0    / r[9][9]= 1215
```

Y comprobamos así que sale lo mismo que en la versión secuencial.

10. Hacer un estudio de escalabilidad (ganancia en velocidad en función del número de cores) en atcgrid y en su PC del código paralelo implementado para dos tamaños de las matrices. Debe recordar usar `-O2` al compilar. El número de núcleos máximo en este estudio debe ser el igual al de núcleos físicos del computador. 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