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-secuencial.c`

CAPTURAS DE PANTALLA:

7. Implementar en paralelo la multiplicación de una matriz triangular por un vector a partir del código secuencial realizado para el ejercicio anterior utilizando la directiva `for` de OpenMP. El código debe repartir entre los threads las iteraciones del bucle que recorre las filas. Dibujar en el cuaderno de prácticas la descomposición de dominio utilizada (Lección 4/Tema 2) en el código paralelo implementado para asignar tareas a los threads (Lección 5/Tema 2). Añadir lo necesario para que el usuario pueda fijar la planificación de tareas usando la variable de entorno `OMP_SCHEDULE`. Obtener en atcgrid los tiempos de ejecución del código paralelo (usando, como siempre, -O2 al compilar) que multiplica una matriz triangular por un vector con las alternativas de planificación `static`, `dynamic` y `guided` para `chunk` de 1, 64 y el `chunk` por defecto para la alternativa. Use un tamaño de vector N múltiplo del número de cores y de 64 que no sea inferior a 15360. El número de threads en las ejecuciones debe coincidir con el número de cores. Rellenar la Tabla 3 dos veces con los tiempos obtenidos. Representar el tiempo para `static`, `dynamic` y `guided` en función del tamaño del `chunk` en una gráfica. ¿Qué alternativa ofrece mejores prestaciones? Razone por qué. Incluya los scripts utilizado en el cuaderno de prácticas. NOTA: Nunca ejecute en atcgrid código que imprima todos los componentes del resultado.

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

RESPUESTA:

CAPTURA CÓDIGO FUENTE: `pmtv-OpenMP.c`

DESCOMPOSICIÓN DE DOMINIO:

CAPTURAS DE PANTALLA:

TABLA RESULTADOS, SCRIPT Y GRÁFICA atcgrid

SCRIPT: pmvt-OpenMP_PCaula.sh

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			
1			
64			
Chunk	**Static**	**Dynamic**	**Guided**
por defecto			

1

64

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-secuencial.c`

CAPTURAS DE PANTALLA:

9. Implementar en paralelo la multiplicación de matrices cuadradas con OpenMP a partir del código escrito en el ejercicio anterior. Use las directivas, las cláusulas y las funciones de entorno que considere oportunas. Se debe paralelizar también la inicialización de las matrices. Dibuje en su cuaderno de prácticas la descomposición de dominio que ha utilizado en el código paralelo implementado para asignar tareas a los threads (Lección 4/Tema 2,Lección 5/Tema 2).

10.

DESCOMPOSICIÓN DE DOMINIO:

CAPTURA CÓDIGO FUENTE: `pmm-OpenMP.c`

CAPTURAS DE PANTALLA:

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