2º curso / 2º cuatr.

Grado Ing. Inform.

Doble Grado Ing. Inform. y Mat.

Arquitectura de Computadores (AC)

Cuaderno de prácticas.

Bloque Práctico 1. Programación paralela I: Directivas OpenMP

Estudiante (nombre y apellidos):

Grupo de prácticas:

Fecha de entrega:

Fecha evaluación en clase:

Ejercicios basados en los ejemplos del seminario práctico

1. Usar la directiva `parallel` combinada con directivas de trabajo compartido en los ejemplos `bucle-for.c` y `sections.c` del seminario. Incorporar el código fuente resultante al cuaderno de prácticas.

RESPUESTA: Captura que muestre el código fuente `bucle-forModificado.c`

```c
/* --Elena Merelo Molina--
Compilamos con gcc -fopenmp bucle-for.c -o bucle-for, ejecutamos con ./bucle 5
y obtenemos:
thread 0 ejecuta la iteración 0 del bucle
thread 2 ejecuta la iteración 2 del bucle
thread 4 ejecuta la iteración 4 del bucle
thread 1 ejecuta la iteración 1 del bucle
thread 3 ejecuta la iteración 3 del bucle
*/

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

int main(int argc, char **argv) {
  int i, n = 9;
  if(argc < 2) {
    fprintf(stderr,"\n[ERROR] - Falta no iteraciones \n");
    exit(-1);
  }
  n = atoi(argv[1]);
  #pragma omp parallel for
    for (i=0; i<n; i++)
    printf("thread %d ejecuta la iteración %d del bucle\n", omp_get_thread_num(),i);
}
```

RESPUESTA: Captura que muestre el código fuente `sectionsModificado.c`

```c
// Elena Merelo Molina

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

void funcA() {
  printf("En funcA: esta sección la ejecuta el thread %d\n", omp_get_thread_num());
}

void funcB() {
  printf("En funcB: esta sección la ejecuta el thread %d\n", omp_get_thread_num());
}

int main() {
  #pragma omp parallel sections
  {
    #pragma omp section
    (void) funcA();
    #pragma omp section
    (void) funcB();
  }
}
```

Resultado de la ejecución antes y después de modificar el código:

```
2018-03-14 23:07:01 elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre/AC/Prácticas/BP1
→ gcc -fopenmp sections.c -o sections

2018-03-14 23:07:20 elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre/AC/Prácticas/BP1
→ ./sections
En funcB: esta sección la ejecuta el thread 0
En funcA: esta sección la ejecuta el thread 1

2018-03-14 23:07:24 elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre/AC/Prácticas/BP1
→ gcc -fopenmp sections.c -o sections

2018-03-14 23:08:56 elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre/AC/Prácticas/BP1
→ ./sections
En funcB: esta sección la ejecuta el thread 2
En funcA: esta sección la ejecuta el thread 4
```

. Imprimir los resultados del programa `single.c` usando una directiva `single` dentro de la construcción `parallel` en lugar de imprimirlos fuera de la región `parallel`. Añadir lo necesario, dentro de la nueva directiva `single` incorporada, para que se imprima el identificador del thread que ejecuta el bloque estructurado de la directiva `single`. Incorpore en su cuaderno de trabajo el código fuente y volcados de pantalla con los resultados de ejecución obtenidos.

RESPUESTA: Captura que muestre el código fuente `singleModificado.c`

```c
//Elena Merelo Molina

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

int main() {
  int n = 9, i, a, b[n];

  for (i=0; i<n; i++)
    b[i] = -1;

  #pragma omp parallel
  {
    #pragma omp single
    {
      printf("Introduce valor de inicialización a: ");
      scanf("%d", &a );
      printf("Single ejecutada por el thread %d\n", omp_get_thread_num());
    }

    #pragma omp for
      for (i=0; i<n; i++)
        b[i] = a;

    #pragma omp single
    {
      for (i=0; i<n; i++){
        printf("b[%d] = %d\t",i,b[i]);
        printf("Single ejecutada por el thread %d\n", omp_get_thread_num());
      }
    }
  }
}
```

Al compilarlo y ejecutarlo:

```
2018-03-15 17:53:33 elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre/AC/Prácticas/BP1
→ gcc -fopenmp single.c -o single

2018-03-15 17:53:54 elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre/AC/Prácticas/BP1
→ ./single
Introduce valor de inicialización a: 32
Single ejecutada por el thread 1
b[0] = 32       Single ejecutada por el thread 0
b[1] = 32       Single ejecutada por el thread 0
b[2] = 32       Single ejecutada por el thread 0
b[3] = 32       Single ejecutada por el thread 0
b[4] = 32       Single ejecutada por el thread 0
b[5] = 32       Single ejecutada por el thread 0
b[6] = 32       Single ejecutada por el thread 0
b[7] = 32       Single ejecutada por el thread 0
b[8] = 32       Single ejecutada por el thread 0
```

Probamos y ejecutamos más veces:

```
 2018-03-15 17:53:56  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre/AC/Prácticas/BP1
○ → export OMP_DYNAMIC=FALSE

 2018-03-15 20:22:38  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre/AC/Prácticas/BP1
○ → export OMP_NUM_THREADS=8

 2018-03-15 20:22:49  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre/AC/Prácticas/BP1
○ → ./single
Introduce valor de inicialización a: 32
Single ejecutada por el thread 0
b[0] = 32       Single ejecutada por el thread 4
b[1] = 32       Single ejecutada por el thread 4
b[2] = 32       Single ejecutada por el thread 4
b[3] = 32       Single ejecutada por el thread 4
b[4] = 32       Single ejecutada por el thread 4
b[5] = 32       Single ejecutada por el thread 4
b[6] = 32       Single ejecutada por el thread 4
b[7] = 32       Single ejecutada por el thread 4
b[8] = 32       Single ejecutada por el thread 4
```

```
 2018-03-15 20:23:01  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre/AC/Prácticas/BP1
○ → export OMP_NUM_THREADS=3

 2018-03-15 20:24:06  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre/AC/Prácticas/BP1
○ → ./single
Introduce valor de inicialización a: 32
Single ejecutada por el thread 2
b[0] = 32       Single ejecutada por el thread 0
b[1] = 32       Single ejecutada por el thread 0
b[2] = 32       Single ejecutada por el thread 0
b[3] = 32       Single ejecutada por el thread 0
b[4] = 32       Single ejecutada por el thread 0
b[5] = 32       Single ejecutada por el thread 0
b[6] = 32       Single ejecutada por el thread 0
b[7] = 32       Single ejecutada por el thread 0
b[8] = 32       Single ejecutada por el thread 0
```

. Imprimir los resultados del programa `single.c` usando una directiva `master` dentro de la construcción `parallel` en lugar de imprimirlos fuera de la región `parallel`. Añadir lo necesario, dentro de la nueva directiva `master` incorporada, para que se imprima el identificador del thread que ejecuta el bloque estructurado de la directiva `master`. Incorpore en su cuaderno el código fuente y volcados de pantalla con los resultados de ejecución obtenidos. ¿Qué diferencia observa con respecto a los resultados de ejecución del ejercicio anterior?

RESPUESTA: Captura que muestre el código fuente `singleModificado2.c`

```c
//Elena Merelo Molina

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

int main() {
  int n = 9, i, a, b[n];

  for (i=0; i<n; i++)
    b[i] = -1;

  #pragma omp parallel
  {
    #pragma omp single
    {
      printf("Introduce valor de inicialización a: ");
      scanf("%d", &a );
      printf("Single ejecutada por el thread %d\n", omp_get_thread_num());
    }

    #pragma omp for
      for (i=0; i<n; i++)
        b[i] = a;

    #pragma omp master
    {
      for (i=0; i<n; i++){
        printf("b[%d] = %d\t",i,b[i]);
        printf("Single ejecutada por el thread %d\n", omp_get_thread_num());
      }
    }
  }
}
```

Resultado de la ejecución:

```
2018-03-15 20:24:21 elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre/AC/Prácticas/BP1
→ gcc -fopenmp single_modificado_2.c -o single_modificado_2

2018-03-15 20:37:38 elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre/AC/Prácticas/BP1
→ ./single_modificado_2
Introduce valor de inicialización a: 32
Single ejecutada por el thread 0
b[0] = 32       Single ejecutada por el thread 0
b[1] = 32       Single ejecutada por el thread 0
b[2] = 32       Single ejecutada por el thread 0
b[3] = 32       Single ejecutada por el thread 0
b[4] = 32       Single ejecutada por el thread 0
b[5] = 32       Single ejecutada por el thread 0
b[6] = 32       Single ejecutada por el thread 0
b[7] = 32       Single ejecutada por el thread 0
b[8] = 32       Single ejecutada por el thread 0
```

Haciendo más pruebas:

```
 2018-03-15 20:37:48  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre/AC/Prácticas/BP1
○ → export OMP_DYNAMIC=FALSE

 2018-03-15 20:39:40  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre/AC/Prácticas/BP1
○ → export OMP_NUM_THREADS=8

 2018-03-15 20:39:43  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre/AC/Prácticas/BP1
○ → ./single_modificado_2
Introduce valor de inicialización a: 32
Single ejecutada por el thread 2
b[0] = 32       Single ejecutada por el thread 0
b[1] = 32       Single ejecutada por el thread 0
b[2] = 32       Single ejecutada por el thread 0
b[3] = 32       Single ejecutada por el thread 0
b[4] = 32       Single ejecutada por el thread 0
b[5] = 32       Single ejecutada por el thread 0
b[6] = 32       Single ejecutada por el thread 0
b[7] = 32       Single ejecutada por el thread 0
b[8] = 32       Single ejecutada por el thread 0
```

```
 2018-03-15 20:39:47  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre/AC/Prácticas/BP1
○ → export OMP_NUM_THREADS=3

 2018-03-15 20:40:08  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre/AC/Prácticas/BP1
○ → ./single_modificado_2
Introduce valor de inicialización a: 32
Single ejecutada por el thread 0
b[0] = 32       Single ejecutada por el thread 0
b[1] = 32       Single ejecutada por el thread 0
b[2] = 32       Single ejecutada por el thread 0
b[3] = 32       Single ejecutada por el thread 0
b[4] = 32       Single ejecutada por el thread 0
b[5] = 32       Single ejecutada por el thread 0
b[6] = 32       Single ejecutada por el thread 0
b[7] = 32       Single ejecutada por el thread 0
b[8] = 32       Single ejecutada por el thread 0
```

RESPUESTA A LA PREGUNTA:

Lo que hay dentro de #pragma omp master es siempre ejecutado por el thread 0, a diferencia del programa anterior, ya que con #pragma omp single le indicas que lo que haya dentro de los corchetes ha de ser ejecutado por una única thread cada vez, mientras que con la directiva master además esta thread ha de ser la principal, la 0, y no pone una barrera implícita.

. ¿Por qué si se elimina directiva `barrier` en el ejemplo `master.c` la suma que se calcula e imprime no siempre es correcta? Responda razonadamente.

RESPUESTA: Al quitar la directiva barrier siempre devuelve que el tiempo es 0, exportemos el número de threads que exportemos, ya que no esperan a los tres segundos cuando se cumple que tid < omp_get_num_threads()/2, con lo que time(NULL) es 0. En definitiva la respuesta no siempre es correcta porque la directiva barrier identifica un punto de sincronización en el cual las threads de una región parallel no ejecutarán más allá de dicha #pragma omp barrier hasta que el resto hayan completado la tarea especificada. En este programa pues no esperan los tres segundos y por eso devuelven 0.

Resto de ejercicios

. El programa secuencial C del Listado 1 calcula la suma de dos vectores (v3 = v1 + v2; v3(i) = v1(i) + v2(i), i=0,…N-1). Generar el ejecutable del programa del Listado 1 para **vectores globales**. Usar `time` (Lección 3/ Tema 1) en la línea de comandos para obtener, en atcgrid, el tiempo de ejecución (*elapsed time*) y el tiempo de CPU del usuario y del sistema generado. Obtenga los tiempos para vectores con 10000000 componentes. ¿La suma de los tiempos de CPU del usuario y del sistema es menor, mayor o igual que el tiempo real (*elapsed*)? Justifique la respuesta.

CAPTURAS DE PANTALLA:

. Generar el código ensamblador a partir del programa secuencial C del Listado 1 para **vectores globales** (para generar el código ensamblador tiene que compilar usando `-S` en lugar de `-o`). Utilice el fichero con el código fuente ensamblador generado y el fichero ejecutable generado en el ejercicio 5 para obtener para atcgrid los MIPS (*Millions of Instructions Per Second*) y los MFLOPS (*Millions of FLOating-point Per Second*) del código que obtiene la suma de vectores (código entre las funciones `clock_gettime()`); el cálculo se debe hacer para 10 y 10000000 componentes en los vectores (consulte la Lección 3/Tema1 AC). Incorpore **el código ensamblador de la parte de la suma de vectores** en el cuaderno.

CAPTURAS DE PANTALLA:

RESPUESTA: cálculo de los MIPS y los MFLOPS

RESPUESTA: Captura que muesre el código ensamblador generado de la parte de la suma de vectores

. Implementar un programa en C con OpenMP, a partir del código del Listado 1, que calcule en paralelo la suma de dos vectores (v3 = v1 + v2; v3(i)=v1(i)+v2(i), i=0,…N-1) usando las directivas `parallel` y `for`. Se debe paralelizar también las tareas asociadas a la inicialización de los vectores. Como en el código del Listado 1 se debe obtener el tiempo (*elapsed time*) que supone el cálculo de la suma. Para obtener este tiempo usar la función `omp_get_wtime()`, que proporciona el estándar OpenMP, en lugar de `clock_gettime()`. NOTAS: (1) el número de componentes N de los vectores debe ser un argumento de entrada al programa; (2) se deben inicializar los vectores antes del cálculo; (3) se debe asegurar que el programa calcula la suma correctamente imprimiendo todos los componentes del vector resultante, v3, para varios tamaños pequeños de los vectores (por ejemplo, N = 8 y N=11); (5) se debe imprimir sea cual sea el tamaño de los vectores el tiempo de ejecución del código paralelo que suma los vectores y, al menos, el primer y último componente de v1, v2 y v3 (esto último evita que las optimizaciones del compilador eliminen el código de la suma).

RESPUESTA: Captura que muestre el código fuente implementado

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

CAPTURAS DE PANTALLA (compilación y ejecución para N=8 y N=11):

. Implementar un programa en C con OpenMP, a partir del código del Listado 1, que calcule en paralelo la suma de dos vectores usando las `parallel` y `sections/section` (se debe aprovechar el paralelismo de datos usando estas directivas en lugar de la directiva `for`); es decir, hay que repartir el trabajo (tareas) entre varios threads usando `sections/section`. Se debe paralelizar también las tareas asociadas a la inicialización de los vectores. Para obtener este tiempo usar la función `omp_get_wtime()` en lugar de `clock_gettime()`. NOTAS: (1) el número de componentes N de los vectores debe ser un argumento de entrada al programa; (2) se deben inicializar los vectores antes del cálculo; (3) se debe asegurar que el programa calcula la suma correctamente imprimiendo todos los componentes del vector resultante, v3, para tamaños pequeños de los vectores (por ejemplo, N = 8 y N=11); (5) se debe imprimir sea cual sea el tamaño de los vectores el tiempo de ejecución del código paralelo que suma los vectores y, al menos, el primer y

último componente de v1, v2 y v3 (esto último evita que las optimizaciones del compilador eliminen el código de la suma).

RESPUESTA: Captura que muestre el código fuente implementado

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

CAPTURAS DE PANTALLA (compilación y ejecución para N=8 y N=11):

. ¿Cuántos threads y cuántos cores como máximo podría utilizar la versión que ha implementado en el ejercicio 7? Razone su respuesta. ¿Cuántos threads y cuantos cores como máximo podría utilizar la versión que ha implementado en el ejercicio 8? Razone su respuesta.

RESPUESTA:

. Rellenar una tabla como la Tabla 2 para atcgrid y otra para su PC con los tiempos de ejecución de los programas paralelos implementados en los ejercicios 7 y 8 y el programa secuencial del Listado 1. Generar los ejecutables usando -O2. En la tabla debe aparecer el tiempo de ejecución del trozo de código que realiza la suma en paralelo (este es el tiempo que deben imprimir los programas). Ponga en la tabla el número de threads/cores que usan los códigos. Represente en una gráfica los tres tiempos. NOTA: Nunca ejecute código que imprima todos los componentes del resultado cuando este número sea elevado.

RESPUESTA:

Tabla 2. Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones paralelas. Sustituir en el encabezado de la tabla "¿?" por el número de threads utilizados, que debe coincidir con el número de cores físicos utilizados.

Nº de Componentes	**T. secuencial vect. Globales 1 thread/core**	**T. paralelo (versión `for`) ¿?threads/cores**	**T. paralelo (versión `sections`) ¿?threads/cores**
16384			
32768			
65536			
131072			
262144			
524288			
1048576			
2097152			
4194304			
8388608			
16777216			
33554432			
67108864			

. Rellenar una tabla como la Tabla 3 para atcgrid con el tiempo de ejecución, tiempo de CPU del usuario y tiempo CPU del sistema obtenidos con `time` para el ejecutable del ejercicio 7 y para el programa secuencial del Listado 1. Ponga en la tabla el número de threads/cores que usan los códigos. ¿El tiempo de CPU que se obtiene es mayor o igual que el tiempo real (*elapsed*)? Justifique la respuesta.
RESPUESTA:

Tabla 3. Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones paralelas. Sustituir en el encabezado de la tabla "¿?" por el número de threads utilizados.

Nº de Componentes	**Tiempo secuencial vect. Globales 1 thread/core**			**Tiempo paralelo/versión `for` ¿? Threads/cores**		
	Elapsed	***CPU-user***	***CPU- sys***	***Elapsed***	***CPU-user***	***CPU- sys***
65536						
131072						
262144						
524288						
1048576						
2097152						
4194304						
8388608						
16777216						
33554432						
67108864						