Universidad del Valle de Guatemala
Facultad de Ingeniería
Departamento de Ciencias de la Computación
CC3086 Programación de microprocesadores

Ciclo 2 de 2,024
Temario A

Laboratorio 05

Competencias para desarrollar
Distribuir la carga de trabajo entre hilos utilizando programación en C y OpenMP.

Instrucciones
Esta actividad se realizará individualmente. Al finalizar los períodos de laboratorio o clase, deberá entregar este archivo en formato PDF y los archivos .c en la actividad correspondiente en Canvas.

1. **(18 pts.)** Explica con tus propias palabras los siguientes términos:
   a) Private

      Una variable en la que cada hilo mantiene su propia copia, dónde los demás no pueden acceder o modificarla. Cuando una variable es privada, se inicializa al comienzo de una región paralela y su valor no se conserva al salir de esta.

   b) Shared

      Una variable que se comparte por todos los hilos en una región paralela, aquí todos pueden acceder y modificar un mismo espacio de memoria al hacer referencia a esta variable. Es importante mantener en cuenta los riesgos de declarar una variable compartida, ya que pueden ocurrir race conditions entre diferentes hilos queriendo acceder al mismo tiempo

   c) Firstprivate

      Este tipo de variable funciona de manera similar a private, sin embargo, se inicializa con el valor que tenía antes de entrar a la región paralela. Cada hilo sigue teniendo su propia copia, sin embargo, no se debe declarar dentro de la región paralela.

   d) Barrier

      Esta directiva se encarga de sincronizar todos los hilos en un punto específico del código. Cuando un hilo se topa con una barrera, espera hasta que todos los demás lleguen al mismo punto antes de continuar.

   e) Critical

      Esta directiva se utiliza para crear una sección que puede ser ejecutada únicamente por un hilo a la vez.

   f) Atomic

      Esta directiva funciona de manera similar a Critical, sin embargo, se utiliza únicamente para operaciones simples cómo incrementos o sumas.

2. **(12 pts.)** Escribe un programa en C que calcule la suma de los primeros N números naturales utilizando un ciclo ***for*** **paralelo**. Utiliza la cláusula **reduction con +** para acumular la suma en una variable compartida.

   Ejercicio2.c en el repositorio de Github

a) Define N como una constante grande, por ejemplo, N = 1000000.

Utilizado N = 10,000,000

b) Usa omp_get_wtime() para medir los tiempos de ejecución.

Pantalla de salida con tiempos de ejecución en paralelo vs secuencial.

```
PS C:\Users\Jose\Desktop\PMP> ./bin/ejercicio2
Paralelo: 50000005000000, Tiempo de Corrida: 0.002000
Secuencial: 50000005000000, Tiempo de Corrida: 0.015000
```

3. (15 pts.) Escribe un programa en C que ejecute tres funciones diferentes en paralelo usando la **directiva #pragma omp sections**. Cada sección debe ejecutar una función distinta, por ejemplo, una que calcule el factorial de un número, otra que genere la serie de Fibonacci, y otra que encuentre el máximo en un arreglo, operaciones matemáticas no simples. Asegúrate de que cada función sea independiente y no tenga dependencias con las otras.

Ejercicio3.c en el repositorio de Github, utilizados cálculos para:

- Fibonacci #40
- Conteo de números primos hasta 1,750,000
- Método de Montecarlo para estimar pi con 17,500,000 muestras

```
Numeros Primos Menores a 1750000 131608, Thread: 2. Tiempo: 0.180000
Pi Usando Montecarlo (17500000 muestras): 3.140932, Thread: 3. Tiempo: 0.268000
Fibonnacci #50: 102334155, Thread: 1. Tiempo: 0.331000
Tiempo total: 0.331000
```

4. (15 pts.) Escribe un programa en C que tenga un ciclo for donde se modifiquen dos variables de manera paralela usando #pragma omp parallel for.

a. Usa la cláusula shared para gestionar el acceso a la variable1 dentro del ciclo.
b. Usa la cláusula private para gestionar el acceso a la variable2 dentro del ciclo.
c. Prueba con ambas cláusulas y explica las diferencias observadas en los resultados.

En la cláusula shared, todos los hilos hacen referencia a una misma posición de memoria y pueden modificar la misma variable en conjunto. En cambio, con la cláusula private, cada hilo almacena una copia diferente a la que los demás no pueden acceder ni modificar.

```
Thread 3, iter 9: variable1 = 12, variable2 = 9
Thread 3, iter 10: variable1 = 535, variable2 = 19
Thread 3, iter 11: variable1 = 546, variable2 = 30
Thread 4, iter 12: variable1 = 39, variable2 = 12
Thread 4, iter 13: variable1 = 559, variable2 = 25
Thread 4, iter 14: variable1 = 573, variable2 = 39
```

Podemos ver que cuándo el thread 3 modifica la variable1, ya se vió afectada por los demás y tiene un valor de 535 a la hora de modificarse. Mientras que cada thread lleva su propio conteo de variable2, sin ser afectados por los demás. También al final variable1 se ve sobre escrita y variable 2 se queda sin modificación alguna.

```
Final: variable1 = 1225, variable2 = 0
```

5. **(30 pts.)** Analiza el código en el programa Ejercicio_5A.c, que contiene un programa secuencial. Indica cuántas veces aparece un valor key en el vector a. Escribe una versión paralela en OpenMP utilizando una descomposición de tareas **recursiva**, en la cual se generen tantas tareas como hilos.

   Ejercicio5.c en el repositorio de Github

6. **REFLEXIÓN DE LABORATORIO: se habilitará en una actividad independiente.**