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
      a. Hace que cada subproceso tenga su propia instancia de un variable.
   b) shared
      a. Recalca que solo una o muchas variables sean compartidas entre los diferentes subprocesos.
   c) firstprivate
      a. Fija que cada subproceso tenga su propia instancia de una variable y así cada variable debe de ser inicializada con el valor de la variable, ya que esta fue creada antes de la construccion paralela.
   d) barrier
      a. Es una sincronización que asegura que todos los hilos de un equipo alcancen un punto específico en el código antes de que cualquiera de ellos pueda continuar.
   e) critical
      a. Restringe la ejecucion del bloque estructurado asociado a un unico hilo a la vez.
   f) atomic
      a. Es para especificar la ubicacion de memoria que será actualizada atómicacmente.

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.
   a) Define N como una constante grande, por ejemplo, N = 1000000.
   b) Usa omp_get_wtime() para medir los tiempos de ejecución.

C Parte2Lab5.c > ...

```c
/*
 * Archivo: Ejercicio_5A.c
 * Descripción: Este programa calcula la suma de los primeros N números naturales
 * utilizando un ciclo for paralelo con OpenMP.
 *
 * Funcionalidad:
 * - Define N como una constante grande.
 * - Utiliza la cláusula reduction con + para acumular la suma en una variable compartida.
 * - Mide los tiempos de ejecución usando omp_get_wtime().
 *
 * Referencia:
 * OPENMP API Specification: Version 5.0 November 2018
 *
 * Fecha modificación: 08-16-2024
 */

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

#define N 3
int main ()
{
    double iniciotiempo,
           fintiempo;
    int contador = 0;
    iniciotiempo = omp_get_wtime();
    #pragma omp parallel for reduction(+ : contador)
    for (int i = 1 ; i<=N ; ++i)
    {
        contador =+i;
    }
    fintiempo = omp_get_wtime();
    printf("Total: %d\nTiempo de ejecucion: %f",contador,fintiempo-iniciotiempo);
    return 0;
}
```

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.

```c
/*
 * Archivo: Parte3Lab5.c
 * Descripción: Este programa ejecuta tres secciones en paralelo utilizando OpenMP.
 *
 * Funcionalidad:
 * - Calcula el factorial de un número.
 * - Genera la serie de Fibonacci.
 * - Encuentra el valor máximo en un arreglo.
 *
 * Estructura:
 * - Utiliza la directiva #pragma omp parallel sections para ejecutar las secciones en paralelo.
 * - La función Fibonacci está diseñada para utilizar tareas anidadas con #pragma omp task.
 *
 * Referencia:
 * OPENMP API Specification: Version 5.0 November 2018
 *
 * Fecha modificación: 08-24-2024
 */

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

int fib(int n) {
    if (n < 2)
        return n;
    else {
        int i, j;
        #pragma omp task shared(i)
```

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.

Parte4Lab5.c > ...

```c
/*
 * Archivo: ModificacionVariables.c
 * Descripción: Este programa modifica dos variables en paralelo utilizando OpenMP.
 *
 * Funcionalidad:
 * - Modifica variable1 de manera compartida entre los hilos.
 * - Modifica variable2 de manera privada para cada hilo.
 *
 * Estructura:
 * - Utiliza la directiva #pragma omp parallel for para ejecutar el ciclo en paralelo.
 * - La variable1 se gestiona con la cláusula shared.
 * - La variable2 se gestiona con la cláusula private.
 *
 * Referencia:
 * OPENMP API Specification: Version 5.0 November 2018
 *
 * Fecha modificación: 08-24-2024
 */

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

int main() {
    int variable1 = 0;
    int variable2 = 0;
    int i;

    // Usando shared para variable1 y private para variable2
    #pragma omp parallel for shared(variable1) private(variable2)
    for (i = 0; i < 10; i++) {
        variable2 = i; // Cada hilo tiene su propia copia de variable2
        #pragma omp critical
        {
            variable1 += variable2; // Acceso compartido a variable1
        }
        printf("Hilo %d: variable1 = %d, variable2 = %d\n", omp_get_thread_num(), variable1, variable2);
    }

    printf("Resultado final: variable1 = %d\n", variable1);
    return 0;
}
```

Con shared todos los hilos comparten la misma variable1, por lo que las modificaciones realizadas por un hilo son visibles para los demás. Esto puede llevar a condiciones de carrera si no se controla adecuadamente con una sección crítica (#pragma omp critical), mientras que on private cada hilo tiene su propia copia de variable2, por lo que no hay interferencia entre los hilos al modificar esta variable.

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.

Parte5Lab4.c > ...

```c
/*
 * Archivo: Ejercicio_5A_Paralelo.c
 * Descripción: Este programa cuenta cuántas veces aparece un valor específico
 * ('key') en un arreglo de números aleatorios utilizando OpenMP con descomposición de tareas recursiva.
 *
 * Funcionalidad:
 * - Genera un arreglo de tamaño N con valores aleatorios.
 * - Inserta manualmente el valor 'key' en tres posiciones específicas del arreglo.
 * - Cuenta cuántas veces aparece 'key' en el arreglo usando una función llamada count_key_parallel().
 * - Imprime el número de apariciones de 'key' en el arreglo.
 *
 * Estructura:
 * - Utiliza la directiva #pragma omp parallel y #pragma omp single para iniciar la ejecución paralela.
 * - La función count_key_parallel() utiliza descomposición de tareas recursiva con #pragma omp task.
 *
 * Referencia:
 * Chandra, . R. et al. Parallel Programming in OpenMP
 *
 * Fecha modificación: 08-24-2024
 */

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

#define N 131072
#define THRESHOLD 1024  // Umbral para cambiar a ejecución secuencial

long count_key_parallel(long *a, long key, long start, long end) {
    long count = 0;

    if (end - start <= THRESHOLD) {
        // Ejecución secuencial si el tamaño del subarreglo es menor o igual al umbral
        for (long i = start; i < end; i++)
            if (a[i] == key) count++;
    } else {
        // Ejecución paralela
        long mid = (start + end) / 2;
        long count1 = 0, count2 = 0;

        #pragma omp task shared(count1)
        count1 = count_key_parallel(a, key, start, mid);

        #pragma omp task shared(count2)
        count2 = count_key_parallel(a, key, mid, end);

        #pragma omp taskwait
        count = count1 + count2;
```

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