	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor:	Jesús Cruz Navarro
Asignatura:	Estructuras de Datos y Algoritmos 2
Grupo:	1
No. de Práctica(s):	13
Integrante(s):	Ugalde Velasco Armando
No. de Equipo de cómputo empleado:	
No. de Lista o Brigada:	32
Semestre:	2021-1
Fecha de entrega:	23 de enero de 2021
Observaciones:	

CALIFICACIÓN: _____

PRÁCTICA 13: ALGORITMOS PARALELOS II

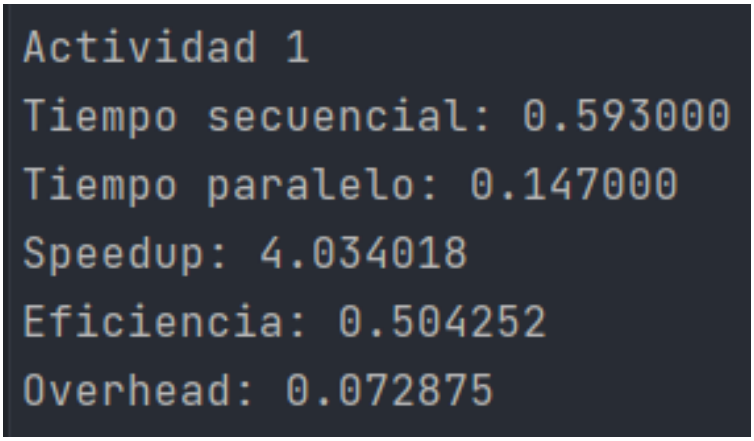
Objetivo: El estudiante utilizará algunos patrones paralelos para resolver algunos problemas de forma paralela y con ello adquiera experiencia en el desarrollo de programas multihilo en sistemas multiprocesador de memoria compartida.

Actividad 1

Se realizó el programa solicitado en la práctica, el cual consiste en calcular la multiplicación entre dos matrices de orden 500x500. Se implementó el programa en su versión serial y paralela, obteniendo sus respectivas métricas. Como se logra observar, la versión paralela superó considerablemente a la versión serial respecto a sus rendimientos.

```
void actividad1()
{
    printf( _Format: "Actividad 1\n");
    int N = 500;
    int MAX_VAL = 10;

    int **A = generateMatrix(N, N, MAX_VAL),
        **B = generateMatrix(N, N, MAX_VAL),
        **C = allocateMatrix(N, N),
        **C2 = allocateMatrix(N, N);

    double t1_serial = omp_get_wtime();
    multiply_serial(A, B, C, N, N, N);
    double t2_serial = omp_get_wtime();

    double t1_parallel = omp_get_wtime();
    multiply_parallel(A, B, C2, N, N, N);
    double t2_parallel = omp_get_wtime();

    free(A);
    free(B);
    free(C);
    free(C2);
    printData( tiempo_secuencial: t2_serial - t1_serial, tiempo_paralelo: t2_parallel - t1_parallel);
}
```

Función principal



Salida del programa

Actividad 2

Se implementó el programa solicitado en la práctica, el cual consiste en calcular el histograma de una imagen en tono de grises. De igual forma, se implementó la versión serial y paralela y se obtuvo el promedio de tiempo de ejecución de tres lecturas. En este caso, se puede observar que la mejora de rendimiento no fue considerablemente alta, probablemente debido al overhead de crear los histogramas temporales para cada hilo y adquirir el lock para modificar el histograma principal (sección critical).

¿Cuánto tiempo tardaron ambas versiones?

Secuencial: 0.00233s

Paralelo: 0.002s

¿Por qué en la versión paralela el cálculo de histo[] está delimitado con el constructor critical?

Para evitar condiciones de carrera al actualizar los valores de frecuencia.

```
void actividad2()
{
    printf( _Format: "Actividad 2\n");
    int TONOS_DE_GRIS = 256;
    int N = 1000;
    int **IMAGE = generateMatrix(N, N, TONOS_DE_GRIS);
    int *histogram_serial = (int*) malloc( _Size: sizeof(int) * TONOS_DE_GRIS);
    int *histogram_parallel = (int*) malloc( _Size: sizeof(int) * TONOS_DE_GRIS);

    int K = 3;
    double t_serial_avg = 0, t_parallel_avg = 0;

    for (int i = 0; i < K; ++i)
    {
        double t1_serial = omp_get_wtime();
        get_histogram_serial(IMAGE, N, TONOS_DE_GRIS, histogram_serial);
        double t2_serial = omp_get_wtime();

        double t1_parallel = omp_get_wtime();
        get_histogram_parallel(IMAGE, N, TONOS_DE_GRIS, histogram_parallel);
        double t2_parallel = omp_get_wtime();

        t_serial_avg += t2_serial - t1_serial;
        t_parallel_avg += t2_parallel - t1_parallel;
    }
    t_serial_avg /= K;
    t_parallel_avg /= K;

    free(histogram_parallel);
    free(histogram_serial);
    free(IMAGE);
    printData(t_serial_avg, t_parallel_avg);
```

Función principal

```
Actividad 2
Tiempo secuencial: 0.002333
Tiempo paralelo: 0.002000
Speedup: 1.166567
Eficiencia: 0.145821
Overhead: 0.001708
```

Salida del programa

Actividad 3

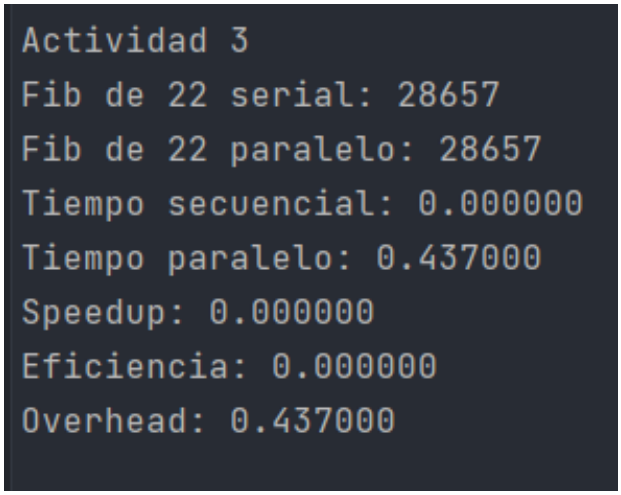
Se implementó el programa solicitado en la práctica, el cual obtiene el número solicitado de la sucesión de fibonacci. Se realizó la versión serial y paralela y se obtuvieron las métricas correspondientes. En este caso, el tiempo tomado por la versión paralela fue mayor, lo cual posiblemente se deba al overhead por el levantamiento y manejo de hilos, además de los constructos utilizados.

¿Qué pasa si f1 y f2 no se colocan como compartidas? Se obtiene un resultado incorrecto del número.

¿Por qué sucede lo observado? Debido a que f1 y f2 se encuentran almacenadas en el stack, al momento de finalizar cierta tarea es posible que se pierda o se corrompa el lugar donde f1 o f2 están siendo almacenadas. Por ello, es necesario expresar de forma explícita que se deben de compartir mediante el constructo shared, el cual realiza una copia de la variable en cuestión en el stack a utilizar.

```
void actividad3()
{
    printf( _Format: "Actividad 3\n");
    int to_calculate = rand() % 30;

    double t1_serial = omp_get_wtime();
    long serial_fib = fib_serial(to_calculate);
    double t2_serial = omp_get_wtime();

    long parallel_fib;
    double t1_parallel, t2_parallel;
    #pragma omp parallel
    {
        #pragma omp single
        {
            t1_parallel = omp_get_wtime();
            parallel_fib = fib_parallel(to_calculate);
            t2_parallel = omp_get_wtime();
        }
    }

    printf( _Format: "Fib de %d serial: %ld\n", to_calculate, serial_fib);
    printf( _Format: "Fib de %d paralelo: %ld\n", to_calculate, parallel_fib);

    printData( tiempo_secuencial: t2_serial - t1_serial, tiempo_paralelo: t2_parallel - t1_parallel);
}
```

Función principal



Salida del programa

CONCLUSIONES

El paralelismo nos permite distribuir tareas entre distintas unidades de ejecución, para lograr aprovechar los recursos presentes en computadoras con arquitectura de memoria compartida. En la práctica, se implementaron algunos algoritmos paralelos, partiendo de sus versiones secuenciales. Cabe mencionar que en algunos casos los tiempos de ejecución empeoraron en sus versiones paralelas, debido a ciertos factores como el levantamiento y manejo de hilos.

Es importante comprender las técnicas de programación en paralelo y los constructos provistos por la librería OpenMP para así tomar ventaja de las arquitecturas multiprocesador, que forman parte de la mayoría de las computadoras actuales.