UNIVERSIDAD DEL VALLE DE GUATEMALA

Computación Paralela

Sección 10

Ing. Juan Luis García Zarceño

Miniproyecto OpenMP

Maria Ramirez #21342

Gustavo González #21438

30 de agosto de 2024

Índice

Introducción

El propósito de este proyecto es explorar las diferencias de rendimiento entre la programación secuencial y la programación paralela utilizando OpenMP en el lenguaje C. Para ello, se desarrolló un protector de pantalla (screensaver) que dibuja curvas de Lissajous, tanto en su versión secuencial como en su versión paralela.

Las curvas de Lissajous, conocidas por sus patrones oscilatorios complejos y visualmente atractivos, fueron elegidas como el objeto gráfico a renderizar debido a su capacidad para ilustrar de manera clara el impacto del procesamiento paralelo en operaciones gráficas intensivas. Al ejecutar ambas versiones del programa, se requiere la entrada de dos parámetros: el número de fotogramas (frames) a renderizar y el número de curvas a dibujar. Cada curva se genera de forma aleatoria, proporcionando un escenario ideal para observar las ventajas de paralelizar el cálculo y la renderización gráfica.

Este proyecto se enfoca en medir el aumento de velocidad al utilizar OpenMP para paralelizar la tarea de dibujar múltiples curvas en comparación con la ejecución secuencial del mismo proceso. El análisis de los resultados permitirá evaluar el desempeño y la eficiencia del procesamiento paralelo en este contexto específico.

Antecedentes

El desarrollo de este proyecto se llevó a cabo en un entorno controlado utilizando contenedores Docker, lo que permitió estandarizar el ambiente de desarrollo y garantizar la consistencia en las pruebas de rendimiento. Para ello, se configuró un contenedor basado en la versión más reciente de Ubuntu, dentro del cual se instaló todo el conjunto de herramientas y librerías necesarias mediante un Dockerfile personalizado.

Entre las principales librerías utilizadas se encuentran **Cairo Graphics**, una poderosa librería para la creación de gráficos vectoriales en 2D, y **OpenMP**, que facilita la paralelización de procesos en aplicaciones C. Además, se incorporaron las librerías **libgif-dev**, **libpng-dev** y **ImageMagick** para

permitir la exportación del screensaver en formato GIF, lo cual fue crucial para la visualización y análisis de los resultados.

En cuanto a la elección de las curvas de **Lissajous** como el objeto gráfico a renderizar, estas curvas se generan a partir de ecuaciones paramétricas que describen movimientos armónicos en dos direcciones perpendiculares. Las curvas de Lissajous son conocidas por sus patrones oscilatorios complejos y estéticamente interesantes. Adicionalmente, cada curva puede ser computada de forma independiente lo que las hace ideales para visualizar diferencias en rendimiento entre ejecuciones secuenciales y paralelas.

Al desarrollar y ejecutar el programa en este entorno configurado, fue posible realizar un análisis riguroso del impacto de la paralelización con OpenMP en el rendimiento gráfico, comparado con la implementación secuencial del mismo proceso.

Anexo 1 - Diagrama de flujo

https://lucid.app/lucidchart/3a8bb457-ad59-4093-a5ad-9383fb6b086b/edit?viewport_loc=-6416%2C-2614%2C13186%2C6180%2C0_0&invitationId=inv_4e9ac73e-ecb4-4548-ba24-05e7dd0f24b9

Anexo 2 - Catálogo de funciones

1. **clear_frames_directory**
    a. Descripción: Elimina todos los archivos en la carpeta frames.
    b. Parámetros: Ninguno.
    c. Retorno: Ninguno

2. **draw_lissajous_curve**
    a. Descripción: Dibuja una curva de Lissajous con opacidad en un punto de inicio específico.
    b. Parámetros:
        i. cairo_t *cr: Contexto de dibujo.
        ii. int N: Número de puntos a dibujar.
        iii. float time: Tiempo actual del fotograma.
        iv. CurveParams params: Parámetros de la curva (a, b, delta).
        v. CurveStart start: Punto de inicio de la curva.
        vi. float opacity: Opacidad de la curva.
    c. Retorno: Ninguno.

3. **save_frame_as_image**
    a. Descripción: Guarda un fotograma como imagen PNG en la carpeta frames.
    b. Parámetros:

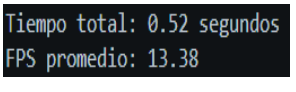
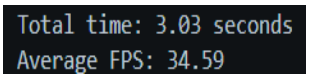
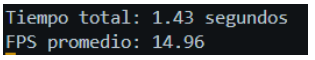
i. int frame_num: Número del fotograma.
ii. int num_curves: Número de curvas a dibujar.
iii. CurveStart *starts: Array de puntos de inicio de las curvas.
iv. CurveParams *params: Array de parámetros de las curvas.
v. float *opacities: Array de opacidades de las curvas.

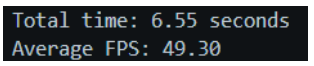
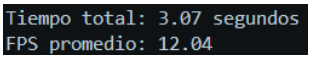
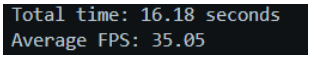
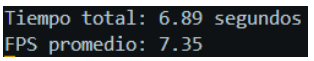
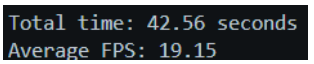
c. Retorno: Ninguno.

4. **create_gif**
   a. Descripción: Crea un GIF a partir de los fotogramas generados en la carpeta frames.
   b. Parámetros:
      i. int num_frames: Número de fotogramas generados.
   c. Retorno: Ninguno.

5. **main**
   a. Descripción: Función principal que controla el flujo del programa.
   b. Parámetros:
      i. int argc: Contador de argumentos de la línea de comandos.
      ii. char *argv[]: Argumentos de la línea de comandos.
   c. Retorno: Código de salida (1 en caso de error, 0 en caso de éxito).

Anexo 3 - Bitácora de Pruebas

Prueba Nº	Implementación	Frames	Curvas	Tiempo Total (s)	FPS Promedio	Captura de Pantalla
1	Paralela	100	5	0.52	192.30	
2	Secuencial	100	5	3.03	34.59	
3	Paralela	300	10	1.43	209.79	

4	Secuencial	300	10	6.55	49.30	
5	Paralela	500	25	3.07	162.86	
6	Secuencial	500	25	16.18	35.05	
7	Paralela	700	50	6.89	101.59	
8	Secuencial	700	50	42.56	19.15	

Nota: Los FPS fueron calculados individualmente para cada hilo por lo que parece que se tuvo un framerate más bajo, cuando realmente no es el caso. Si miramos la cantidad total de cuadros renderizados y el tiempo total que pasó, podemos calcular los FPS promedio reales.

Anexo 4 - Repositorio de Github

https://github.com/mariaRam2003/Proyecto1_Paralela/tree/main

Conclusiones

Impacto Significativo de la Paralelización en el Rendimiento Gráfico: El uso de OpenMP para paralelizar el cálculo y la renderización de las curvas de Lissajous resultó en una reducción notable del tiempo total de ejecución en comparación con la versión secuencial. Esto demuestra que la programación paralela puede mejorar considerablemente la eficiencia en tareas gráficas intensivas, especialmente cuando se manejan múltiples elementos independientes como las curvas de Lissajous.

Escalabilidad y Desempeño Dependientes de la Complejidad del Escenario: A medida que el número de fotogramas y curvas aumentó, la ventaja del enfoque paralelo se hizo más evidente. Sin embargo, la relación entre el número de curvas y la mejora del rendimiento no fue lineal, lo que sugiere que la eficiencia de la paralelización depende de la complejidad y la cantidad de trabajo a distribuir entre los hilos de ejecución.

Entorno de Desarrollo Controlado Facilita Análisis Rigurosos: El uso de contenedores Docker para estandarizar el ambiente de desarrollo permitió realizar pruebas de rendimiento consistentes y reproducibles. Esto fue crucial para identificar las diferencias de rendimiento entre las implementaciones paralela y secuencial, asegurando que las mejoras observadas fueran resultado directo de la paralelización y no de variaciones en el entorno de ejecución.

Referencias

Al-Khazali, H. A., & Askari, M. R. (2012). Geometrical and graphical representations analysis of lissajous figures in rotor dynamic system. *IOSR Journal of Engineering*, *2*(5), 971-978.

Chandra, R. (2001). *Parallel Programming in OpenMP*. Academic Press.

Cairo Graphics (2024). Cairo. https://www.cairographics.org