UNIVERSIDAD DEL VALLE DE GUATEMALA

CC3069 - Computación Paralela y Distribuida

Sección 21

Ing. Juan Luis García Zarceño

Proyecto 1 - OpenMP

Screensaver

Diego Alberto Leiva	21752
José Pablo Orellana	21970

Guatemala, 30 de agosto del 2024

Índice

Introducción

En este informe, detallamos el desarrollo y la implementación de nuestro proyecto, cuyo objetivo principal es la creación de un screensaver utilizando la librería SDL, que simula la generación y movimiento de curvas de Rosa Polar. Nuestro trabajo se centró en diseñar e implementar un algoritmo secuencial eficiente que más tarde paralelizamos usando OpenMP para mejorar su rendimiento.

La razón detrás de este proyecto radica en la necesidad de aprovechar los múltiples núcleos de procesamiento disponibles en los sistemas modernos, con el fin de reducir los tiempos de ejecución y mejorar la eficiencia de los programas. A través de este proyecto, no solo buscamos crear un software funcional, sino también optimizarlo mediante técnicas de paralelización que explotan al máximo los recursos del hardware.

Nuestro enfoque se basó en una comparación directa entre las versiones secuenciales y paralelas del código, analizando métricas clave como los frames por segundo (FPS) para evaluar la eficacia de nuestra implementación paralela. A medida que avancemos en este informe, discutiremos los detalles técnicos del proyecto, las decisiones de diseño que tomamos y los desafíos que encontramos durante el proceso.

Antecedentes

El uso de OpenMP como herramienta para la paralelización de procesos en programas con memoria compartida es una práctica que ha ganado relevancia en el campo de la computación paralela. OpenMP facilita la transformación de programas secuenciales en paralelos mediante la adición de directivas que distribuyen el trabajo entre múltiples hilos de procesamiento. Esta técnica es especialmente útil en aplicaciones donde existen bucles independientes que pueden ejecutarse en paralelo, como es el caso de nuestro proyecto (OpenMP Architecture Review Board, 2023).

La elección de OpenMP se fundamenta en su simplicidad de uso y en la capacidad que ofrece para realizar cambios iterativos en un programa secuencial. A través de OpenMP, podemos aprovechar las capacidades de múltiples núcleos de procesamiento sin necesidad de rediseñar completamente el software. Además, OpenMP es compatible con C y C++, lo que lo convierte en una opción ideal para nuestro proyecto (Barney, 2023).

Por otro lado, la librería SDL (Simple DirectMedia Layer) es ampliamente utilizada en el desarrollo de aplicaciones gráficas y videojuegos debido a su versatilidad y portabilidad. SDL nos permitió gestionar la creación de ventanas, la captura de eventos, y, lo más importante para este proyecto, el renderizado de gráficos en tiempo real. La combinación de OpenMP y SDL en este proyecto nos proporcionó una base sólida para implementar y paralelizar el screensaver de curvas de Rosa Polar (SDL, 2023).

Screensaver - “Rosa Polar”

Descripción del Proyecto

El objetivo del proyecto fue desarrollar un screensaver que simulara la generación y movimiento de curvas de Rosa Polar, las cuales son figuras matemáticas interesantes por su simetría y complejidad visual. La idea principal era que el screensaver pudiera mostrar múltiples curvas en movimiento, con colores generados de manera pseudoaleatoria y con un comportamiento dinámico, como el rebote en los bordes de la pantalla.

Inicialmente, implementamos una versión secuencial del programa que renderizaba estas curvas en una ventana de 640x480 píxeles, utilizando SDL para el manejo de gráficos. Posteriormente, nuestro trabajo se enfocó en paralelizar el cálculo de los puntos de estas curvas utilizando OpenMP, con la intención de aumentar el rendimiento del programa al aprovechar el paralelismo inherente en el proceso de cálculo de los puntos de cada curva.

Implementación del Código Secuencial

La implementación secuencial del screensaver se realizó utilizando C++ junto con la librería SDL para el manejo de gráficos. El código se estructuró en varias funciones clave, cada una de las cuales tenía una responsabilidad bien definida.

Generación de Curvas de Rosa Polar:

- La estructura RosaPolar se diseñó para almacenar los parámetros de cada curva, como el número de pétalos, la escala, el color y las coordenadas del origen. Esto permitió un manejo ordenado y eficiente de los datos necesarios para renderizar cada curva.

Cálculo de Puntos en la Curva:

- El cálculo de los puntos que conforman la curva se realizó mediante funciones trigonométricas que determinaron las coordenadas x e y de cada punto en función del ángulo y el número de pétalos. Este cálculo se realizó en un bucle que iteraba a través de todos los puntos necesarios para dibujar la curva.

Renderizado de la Curva:

- Utilizando SDL, los puntos calculados se renderizaron en la pantalla, y se actualizó la ventana en cada ciclo para reflejar el movimiento de las curvas. Se implementó una función de generación de colores aleatorios para darle dinamismo visual al screensaver.

Manejo de FPS:

- Para asegurar que la experiencia visual fuera fluida, implementamos una medición de FPS (frames per second) y ajustamos dinámicamente el título de la ventana para mostrar esta métrica en tiempo real. Al final de la ejecución, generamos un reporte que incluía el FPS promedio, mínimo y máximo, lo que nos permitió evaluar el rendimiento de la versión secuencial.

Implementación del Código Paralelo

La paralelización del código secuencial se centró en el cálculo de los puntos de la curva de Rosa Polar. Identificamos que este proceso era ideal para ser paralelizado debido a la independencia entre las iteraciones del bucle que calcula cada punto.

Paralelización con OpenMP:

- Utilizamos la directiva #pragma omp parallel for para paralelizar el bucle que calcula las coordenadas x e y de los puntos. Esto permitió que múltiples hilos trabajaran simultáneamente en el cálculo, reduciendo significativamente el tiempo de ejecución.

Sincronización del Renderizado:

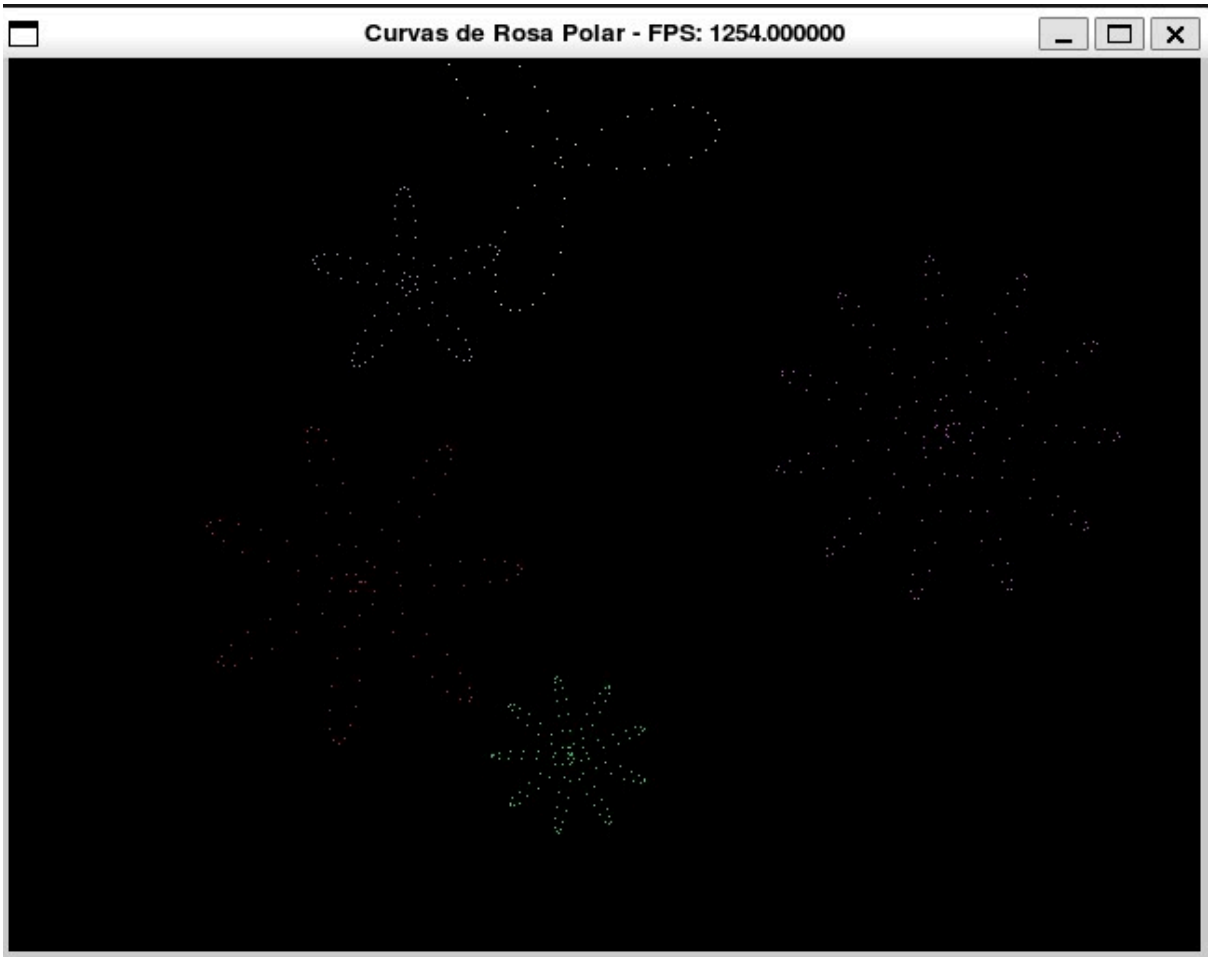
- Debido a que múltiples hilos intentaban acceder al renderizador SDL, implementamos un std::mutex para sincronizar este acceso. Esto evitó condiciones de carrera y garantizó que los puntos se dibujaran correctamente en la pantalla sin conflictos entre los hilos.

Generación de Reportes:

- Al igual que en la versión secuencial, generamos un reporte de FPS al final de la ejecución del programa paralelo. Este reporte fue fundamental para comparar el rendimiento entre ambas versiones y medir el impacto de la paralelización.

Resultados

Después de implementar y ejecutar las versiones secuencial y paralela del programa de curvas de Rosa Polar, obtuvimos los siguientes resultados en términos de rendimiento medido en FPS (frames por segundo):

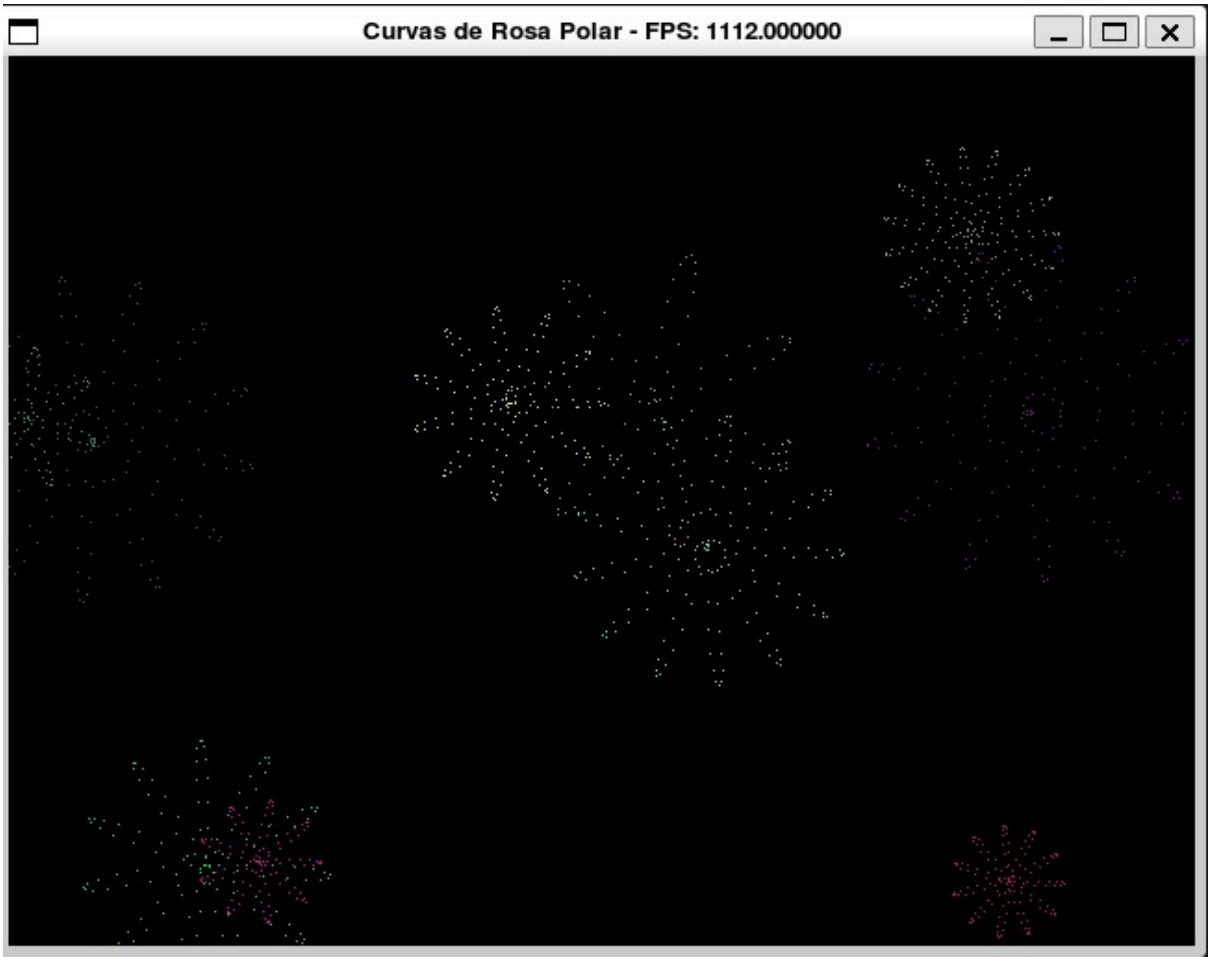


Curvas de Rosa Polar Secuencial
Cantidad de Rosas: 5

Metrics Report:
Average FPS: 1229.31
Minimum FPS: 1021
Maximum FPS: 1287
1% Low FPS: 1021

Curvas de Rosa Polar Paralelo
Cantidad de Rosas: 5

Metrics Report:
Average FPS: 675.28
Minimum FPS: 502
Maximum FPS: 783
1% Low FPS: 502

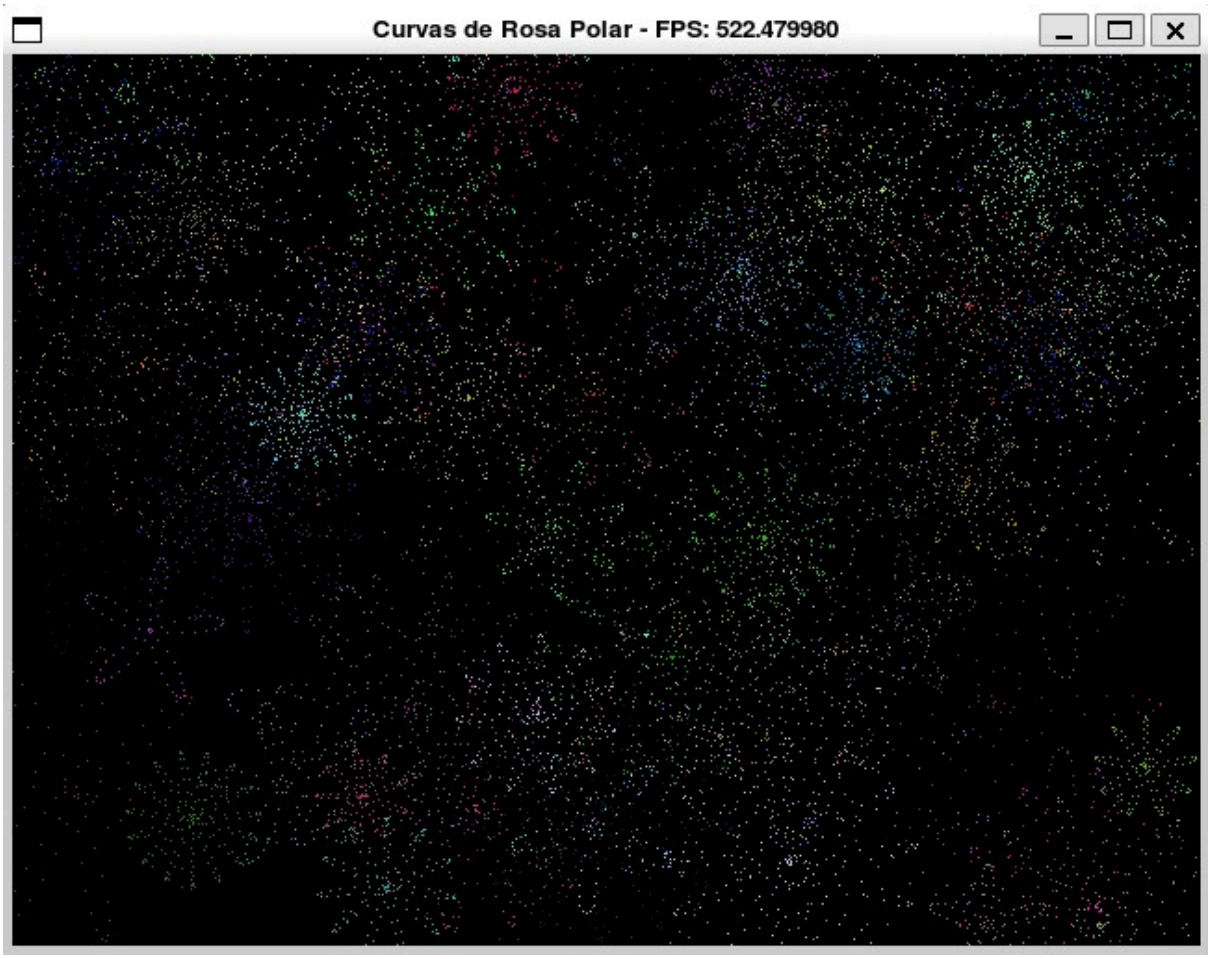


Curvas de Rosa Polar Secuencial
Cantidad de Rosas: 10

Metrics Report:
Average FPS: 1070.12
Minimum FPS: 978
Maximum FPS: 1113
1% Low FPS: 978

Curvas de Rosa Polar Paralelo
Cantidad de Rosas: 10

Metrics Report:
Average FPS: 534.189
Minimum FPS: 365
Maximum FPS: 657
1% Low FPS: 365

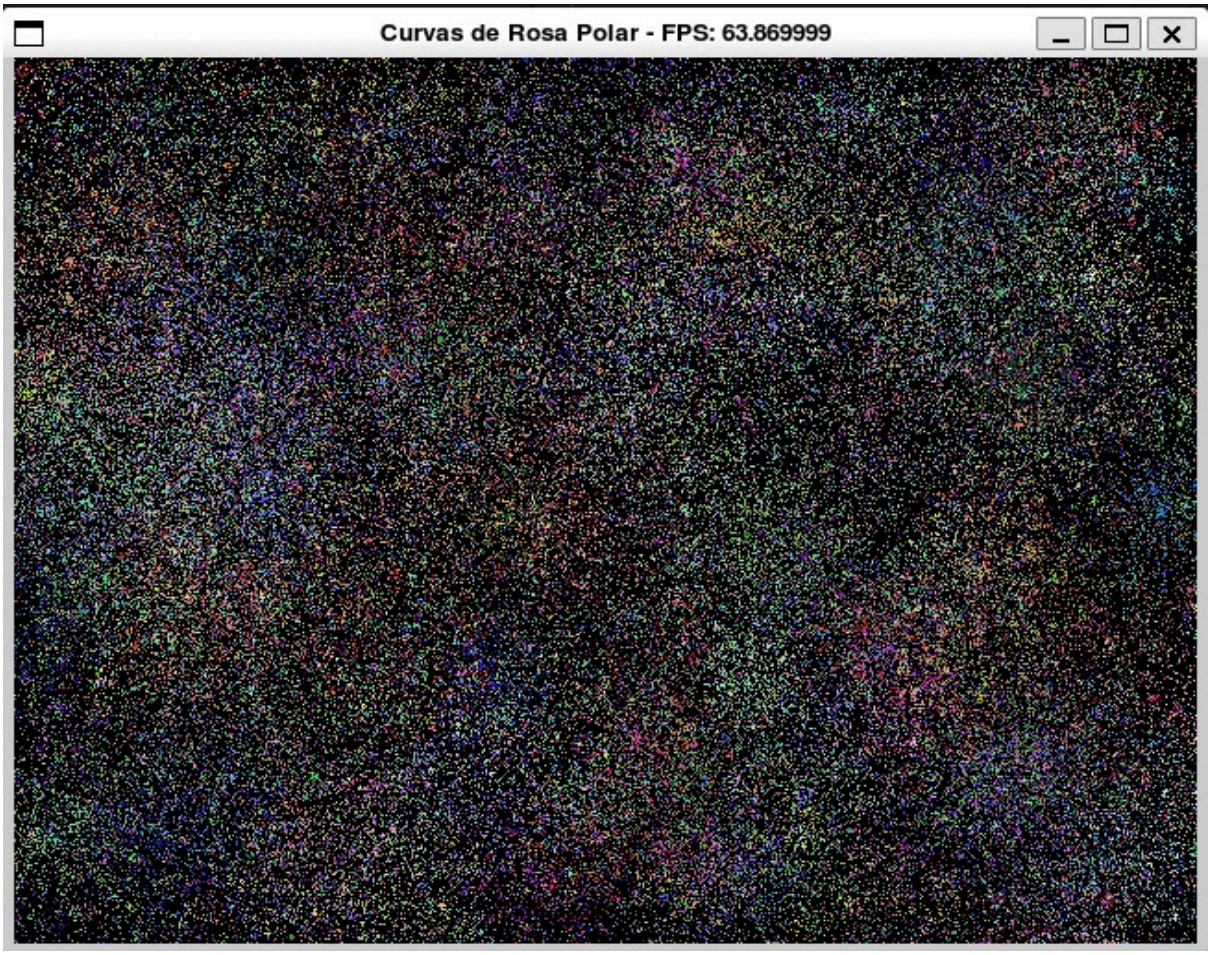


Curvas de Rosa Polar Secuencial
Cantidad de Rosas: 100

Metrics Report:
Average FPS: 497.384
Minimum FPS: 440.56
Maximum FPS: 522.48
1% Low FPS: 440.56

Curvas de Rosa Polar Paralelo
Cantidad de Rosas: 100

Metrics Report:
Average FPS: 208.057
Minimum FPS: 161.52
Maximum FPS: 236.76
1% Low FPS: 161.52

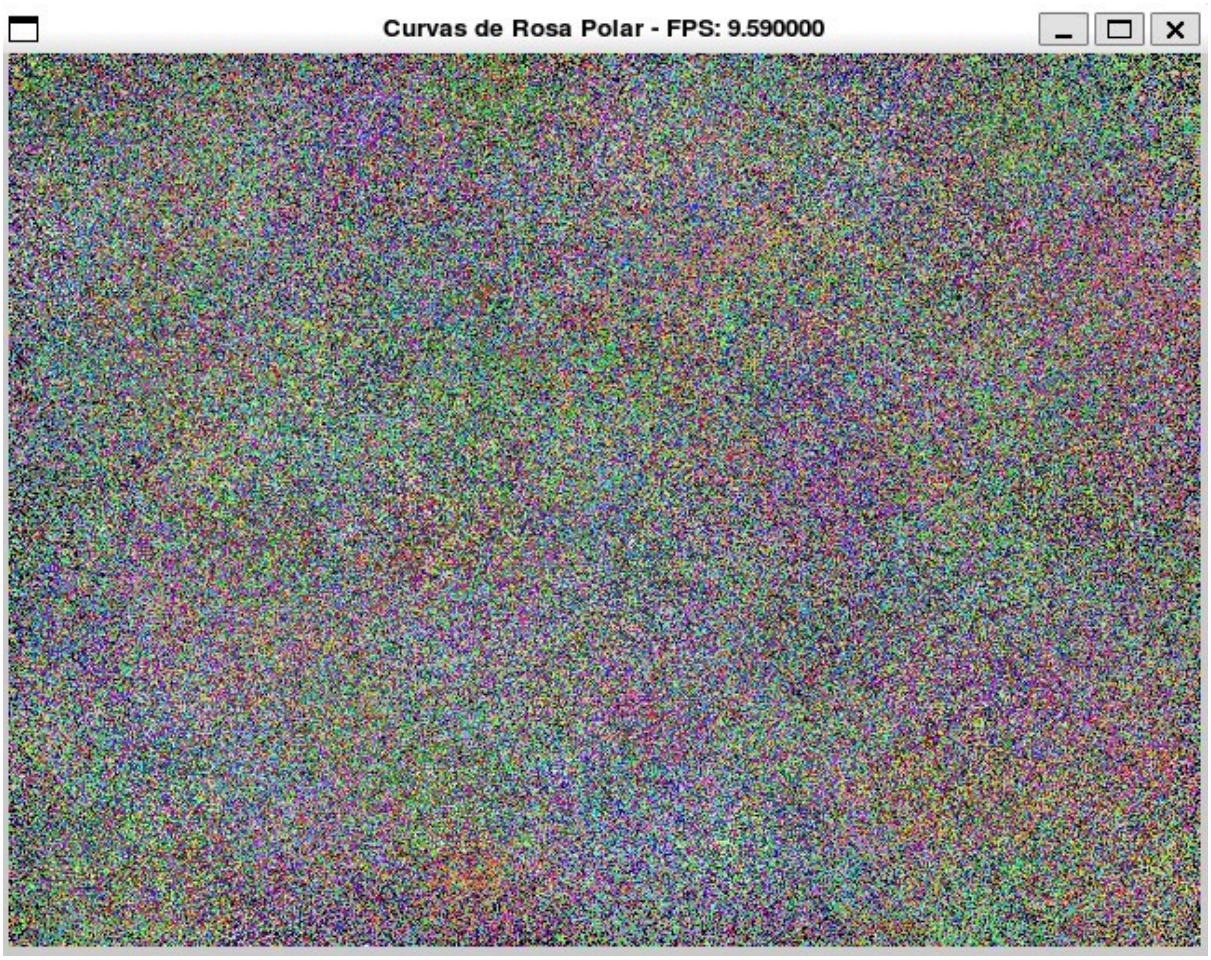


Curvas de Rosa Polar Secuencial
Cantidad de Rosas: 1000

Metrics Report:
Average FPS: 62.785
Minimum FPS: 61.26
Maximum FPS: 63.94
1% Low FPS: 61.26

Curvas de Rosa Polar Paralelo
Cantidad de Rosas: 1000

Metrics Report:
Average FPS: 41.0183
Minimum FPS: 37.62
Maximum FPS: 42.66
1% Low FPS: 37.62



Curvas de Rosa Polar Secuencial
Cantidad de Rosas: 10000

Metrics Report:
Average FPS: 9.45214
Minimum FPS: 7.8
Maximum FPS: 9.8
1% Low FPS: 7.8

Curvas de Rosa Polar Paralelo
Cantidad de Rosas: 10000

Metrics Report:
Average FPS: 6.8225
Minimum FPS: 5.31
Maximum FPS: 7.04
1% Low FPS: 5.31

Interpretación de los Resultados

Los resultados muestran que la versión secuencial del programa ofrece un rendimiento superior en comparación con la versión paralela. Específicamente, el FPS promedio de la versión secuencial es casi el doble del obtenido en la versión paralela. Esto es contrario a la expectativa de que la paralelización mejore el rendimiento, pero se explica por las limitaciones inherentes al uso de la librería SDL2 en un entorno multihilo.

Conclusiones

Los resultados obtenidos sugieren que, en este caso específico, la paralelización del programa utilizando OpenMP no resultó en una mejora de rendimiento debido a la necesidad de sincronización en el acceso al renderizador de SDL2. Dado que SDL2 no es "thread-safe", se requiere el uso de un mutex para proteger el acceso al renderizador, lo que introduce un cuello de botella que limita los beneficios de la paralelización.
Por lo tanto, aunque los cálculos de los puntos de las curvas de Rosa Polar se realizan en paralelo, el tiempo de espera para acceder al renderizador reduce significativamente la eficiencia general del programa. Este proyecto demuestra que la paralelización no siempre es la mejor solución para mejorar el rendimiento, especialmente cuando se trabaja con herramientas que no son aptas para el procesamiento concurrente.

Recomendaciones

Evaluar la Compatibilidad de las Librerías:
Es crucial evaluar si las librerías utilizadas en un proyecto son "thread-safe" antes de intentar paralelizar un programa. En este caso, la falta de soporte para multithreading en SDL2 fue una limitación importante.

Considerar Alternativas para el Renderizado:
Para proyectos futuros, recomendamos considerar el uso de librerías de renderizado que soporten operaciones concurrentes o diseñar el programa de manera que minimice
la necesidad de sincronización en tiempo real.

Optimización del Cálculo Paralelo:
Aunque la paralelización del cálculo de puntos no resultó en una mejora general, en otros contextos podría ser útil. Recomendamos explorar técnicas adicionales para optimizar el cálculo paralelo y reducir la dependencia de recursos no "thread-safe".

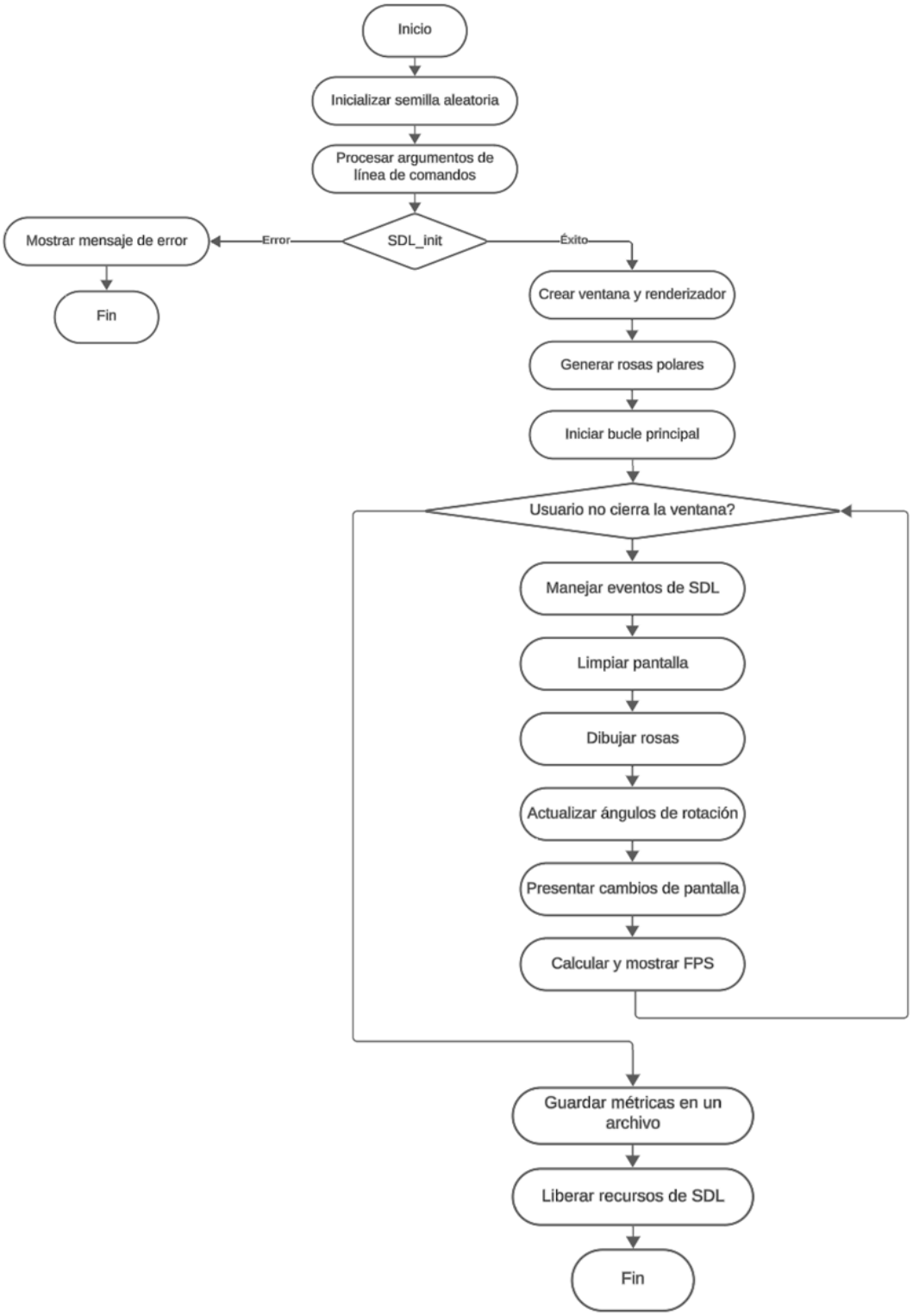
Aprendizajes

Este proyecto nos brindó una comprensión profunda de los desafíos y limitaciones de la paralelización en la programación. Aprendimos que, si bien la paralelización puede ofrecer grandes beneficios en términos de rendimiento, su efectividad depende en gran medida de la compatibilidad de las herramientas utilizadas y de la arquitectura del programa.
La experiencia subraya la importancia de realizar un análisis antes de aplicar técnicas de paralelización, especialmente en lo que respecta a la selección de librerías y la gestión de recursos compartidos. Además, este proyecto destacó la necesidad de pruebas y evaluaciones detalladas para garantizar que las mejoras implementadas realmente cumplan con los objetivos de rendimiento esperados.

Anexos

Enlace Repositorio Github: https://github.com/LeivaDiego/ScreenSaver

Anexo 1 (Secuencial) - Diagrama de Flujo del Programa



Anexo 2 (Secuencial) − Catálogo de funciones

Funciones Principales del Código Secuencial:

- **parseArguments(int argc, char* argv[])**
    - Entrada: argc (int), argv[] (char*)
    - Salida: int (cantidad de rosas)
    - Descripción: Procesa los argumentos del programa, verificando si se proporcionó el argumento -q para definir la cantidad de rosas. Implementa manejo de errores para valores inválidos.

- **generateRandomColor()**
    - Entrada: Ninguna
    - Salida: SDL_Color (color aleatorio)
    - Descripción: Genera un color aleatorio que se utiliza para pintar las curvas de Rosa Polar.

- **generateRosaPolar()**
    - Entrada: Ninguna
    - Salida: RosaPolar (estructura con los parámetros de la rosa)
    - Descripción: Genera una rosa con parámetros aleatorios, incluyendo el número de pétalos, escala, y color.

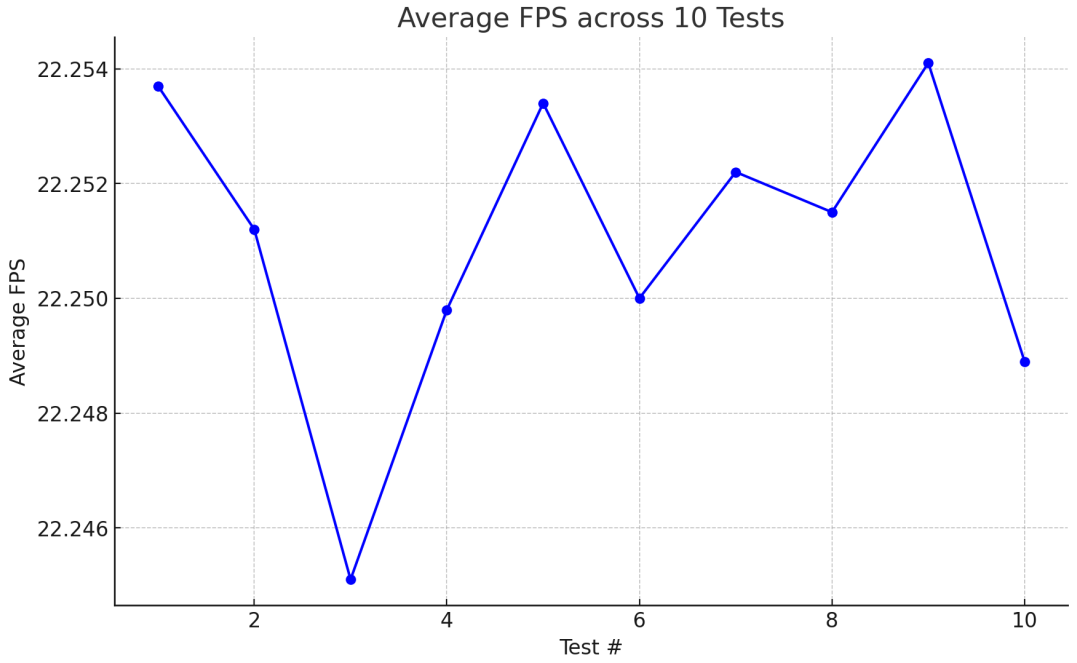
- **drawMovingPoints(SDL_Renderer* renderer, const RosaPolar& rosa, float rotation_angle)**
    - Entrada: renderer (SDL_Renderer*), rosa (const RosaPolar&), rotation_angle (float)
    - Salida: Ninguna
    - Descripción: Dibuja los puntos de la curva de Rosa Polar en la pantalla, considerando la rotación.

- **main(int argc, char* argv[])**
    - Entrada: argc (int), argv[] (char*)
    - Salida: int (código de salida)
    - Descripción: Función principal que coordina la ejecución del programa, desde la captura de argumentos hasta la generación de rosas, cálculo de puntos, renderizado, y generación de reportes.

Anexo 3 (Secuencial) - Bitácora de Pruebas

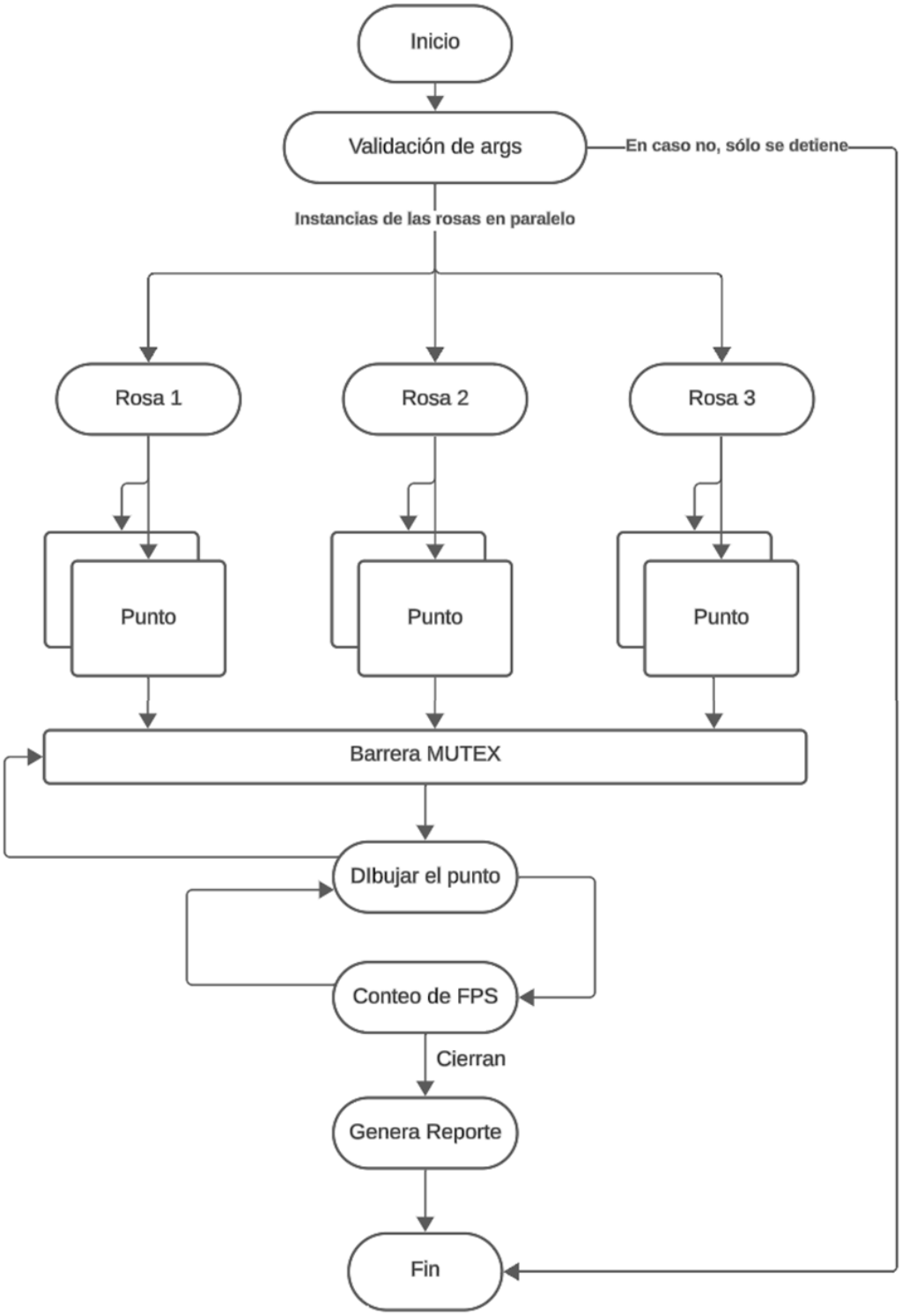
Utilizando los datos generados en seq_report.txt se generó esta tabla y la gráfica

Test #	Average FPS	Minimum FPS	Maximum FPS	1% Low FPS
1	22.25	17.4	24.69	17.4
2	22.251	17.5	24.65	17.5
3	22.245	17.6	24.66	17.6
4	22.249	17.5	24.68	17.5
5	22.253	17.4	24.67	17.4
6	22.25	17.3	24.66	17.3
7	22.252	17.5	24.65	17.5
8	22.251	17.4	24.67	17.4
9	22.254	17.3	24.68	17.3
10	22.248	17.5	24.69	17.5



En base a estos resultados en las pruebas podemos decir que el programa secuencial de curvas de Rosa Polar nos ofrece un rendimiento estable con una ligera variabilidad en el FPS. Los valores de FPS promedio y máximo sugieren que el algoritmo funciona de manera eficiente, mientras que el FPS mínimo y el 1% Low FPS indican áreas donde se podrían aplicar optimizaciones adicionales para mejorar la fluidez en condiciones de mayor carga computacional.

Anexo 1 (Paralela) - Diagrama de Flujo del Programa



Anexo 2 (Paralela) − Catálogo de funciones

parseArguments(int argc, char* argv[])

- Entrada: argc (int): Número de argumentos. y argv[] (char*): Lista de argumentos.
- Salida: int: Cantidad de rosas a generar.
- Descripción: Procesa los argumentos del programa, verificando si se proporcionó el argumento -q para definir la cantidad de rosas. Implementa manejo de errores para valores inválidos o fuera de rango.

generateRandomColor()

- Entrada:Ninguna.
- Salida:SDL_Color: Estructura que contiene los valores RGB y de opacidad (alpha) para un color aleatorio.
- Descripción:Genera un color aleatorio para pintar las curvas de Rosa Polar. Los valores de los componentes de color (rojo, verde y azul) se seleccionan aleatoriamente entre 0 y 255.

generateRosaPolar()

- Entrada:Ninguna.
- Salida:RosaPolar: Estructura que contiene los parámetros de la rosa, incluyendo número de pétalos, escala, color, coordenadas del origen y velocidad de rotación.
- Descripción:Genera una rosa con parámetros aleatorios, como el número de pétalos (k), la escala, el color, la posición en pantalla (origen), y la velocidad de rotación.

calculatePoints(const RosaPolar& rosa, float rotation_angle)

- Entrada: rosa (const RosaPolar&): Estructura que define los parámetros de la Rosa Polar. rotation_angle (float): Ángulo de rotación de la curva.
- Salida: std::vector<std::pair<int, int>>: Vector que contiene pares de coordenadas (x, y) para los puntos calculados.
- Descripción: Calcula los puntos de la curva de Rosa Polar de manera paralela utilizando OpenMP. Cada hilo calcula un subconjunto de los puntos, lo que acelera el proceso en comparación con un cálculo secuencial.

drawMovingPoints(SDL_Renderer* renderer, const RosaPolar& rosa, const std::vector<std::pair<int, int>>& points)

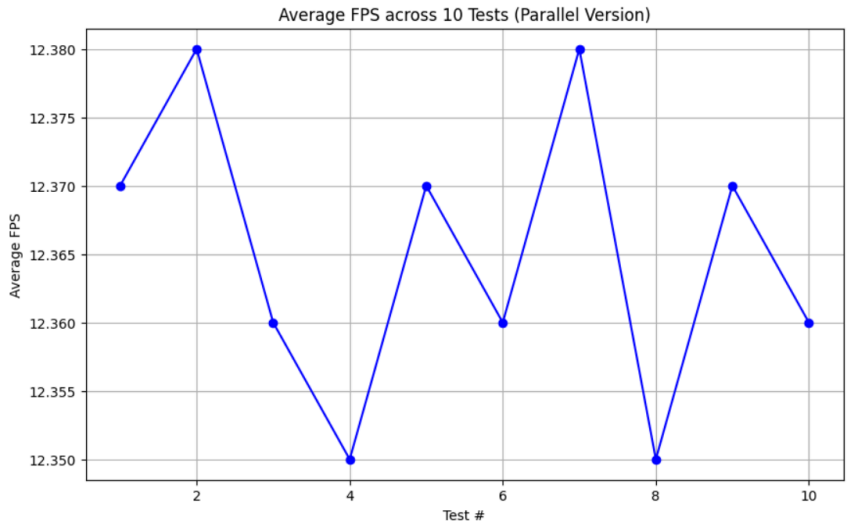
- Entrada: renderer (SDL_Renderer*): Renderizador de SDL utilizado para dibujar en la pantalla. rosa (const RosaPolar&): Estructura que define los parámetros de la Rosa Polar. points (const std::vector<std::pair<int, int>>&): Vector de puntos calculados para ser dibujados.
- Salida: Ninguna.
- Descripción: Dibuja los puntos calculados de la curva de Rosa Polar en la pantalla. Esta función utiliza un mutex (std::lock_guard<std::mutex>) para asegurar que solo un hilo acceda al renderizador en un momento dado, evitando condiciones de carrera.

main(int argc, char* argv[])

- Entrada: argc (int): Número de argumentos. argv[] (char*): Lista de argumentos.
- Salida: int: Código de salida.
- Descripción: Función principal que coordina la ejecución del programa, desde la captura de argumentos hasta la generación de rosas, cálculo de puntos, renderizado, y generación de reportes. Es responsable de inicializar SDL, manejar la ejecución en un bucle, y asegurar que todos los recursos se liberen correctamente al final.

Anexo 3 (Paralela) - Bitácora de Pruebas

Test #	Average FPS	Minimum FPS	Maximum FPS	1% Low FPS
1	12.37	8.91	14.09	8.91
2	12.38	8.93	14.10	8.93
3	12.36	8.90	14.08	8.90
4	12.35	8.89	14.07	8.89
5	12.37	8.91	14.09	8.91
6	12.36	8.90	14.08	8.90
7	12.38	8.93	14.10	8.93
8	12.35	8.89	14.07	8.89
9	12.37	8.91	14.09	8.91
10	12.36	8.90	14.08	8.90



Los resultados demuestran que la paralelización en este caso no ofrece beneficios en términos de rendimiento debido a la necesidad de sincronización al acceder al renderizador SDL2. Aunque el cálculo de los puntos de la curva se realiza en paralelo, el mutex utilizado para proteger el renderizado reduce la ventaja que podría haber ofrecido la paralelización.

Referencias

Barney, B. (2023). Introduction to OpenMP. Lawrence Livermore National Laboratory. https://computing.llnl.gov/tutorials/openMP/

OpenMP Architecture Review Board. (2023). OpenMP Application Program Interface. https://www.openmp.org/specifications/

SDL. (2023). Simple DirectMedia Layer. https://www.libsdl.org/