Universidad del Valle De Guatemala

Facultad de Ingeniería

Computación Paralela y Distribuida


**Proyecto 1**

**Programación Paralela con OpenMP**

Elean Rivas 19062
Diego Ruiz 18761

Guatemala, 20 de septiembre 2023

# Índice



## Introducción

En el ámbito de la computación, la capacidad de ejecutar tareas de manera paralela ha demostrado ser una herramienta esencial para mejorar la eficiencia y el rendimiento de los programas. Para este curso, se nos presentó el desafío de desarrollar un "screensaver" que funcione tanto de forma secuencial como paralela. Este informe detalla el proceso de diseño, desarrollo y optimización de dicho screensaver, utilizando la herramienta OpenMP para lograr la paralelización deseada. A través de este proyecto, buscamos demostrar la comprensión de los conceptos de programación paralela y explorar las posibilidades y limitaciones que ofrece OpenMP en el contexto de aplicaciones gráficas.

## Antecedentes

Para la realización de este proyecto, es importante comprender que es la programación paralela, OpenMP y en el caso específico de nuestro proyecto, SDL. Para empezar, la programación paralela surge de la necesidad de mejorar el rendimiento y la eficiencia de los programas, especialmente en aplicaciones que requieren un alto grado de procesamiento. La programación paralela nos permite la ejecución simultánea de tareas granuladas, que dividen el problema en subproblemas más sencillos de correr, con una unidad de procesamiento dedicada a cada subproblema para agilizar la manera en que encontramos el resultado final.

Dentro de esta necesidad también surge OpenMP, que es una herramienta que ha revolucionado el mundo de la programación paralela. Su principal atractivo radica en la facilidad con la que permite transformar un programa secuencial en uno paralelo, ofreciendo una abstracción y programación paralela en alto nivel. Esta herramienta ha sido diseñada con un enfoque iterativo, lo que significa que los desarrolladores pueden realizar cambios graduales en un programa secuencial para aprovechar múltiples recursos a través de la ejecución paralela. Además, con esto obtenemos un conjunto de directivas y cláusulas que facilitan la paralelización de programas sin requerir cambios drásticos en el código fuente.

Por último, SDL es una biblioteca multimedia de código abierto que proporciona acceso de bajo nivel a audio, teclado, ratón, joystick y funciones gráficas. Es ampliamente reconocida por su capacidad para facilitar el desarrollo de videojuegos y aplicaciones multimedia en diversas plataformas. En este proyecto, SDL sirvió como la base para la creación y manipulación de gráficos, permitiendo la visualización del "screensaver" desarrollado, así como su framerate.

## Desarrollo

Para poder realizar el screensaver de ambas maneras se trabajó con la librería SDL. En primer lugar, se encontró que el rendimiento del programa secuencial es realmente bueno por las optimizaciones realizadas en SDL. A pesar de las ventajas obvias de la programación paralela, el programa secuencial demostró ser altamente eficiente. La diferencia de rendimiento entre el programa paralelo y el secuencial no fue tan pronunciada como se podría haber anticipado inicialmente. Esta observación destaca la importancia de optimizar el código, incluso en un contexto secuencial. Un programa secuencial bien optimizado puede rivalizar en rendimiento con su contraparte paralela si está bien optimizado y aunque sí se nota el cambio entre un programa y otro, se obtuvieron muy buenos resultados de forma secuencial.

Por otro lado, uno de los principales hallazgos que se buscaba del proyecto fue el speedup obtenido al paralelizar el movimiento de los círculos. Este aumento en el rendimiento es una clara indicación de la eficacia de la programación paralela en tareas que, por su naturaleza, pueden dividirse y ejecutarse simultáneamente. Sin embargo, es esencial señalar que el grado de mejora en el rendimiento no es uniforme y depende en gran medida de cómo se asignen y utilicen los threads, así como la cantidad de círculos que se quieran renderizar ya que si son demasiados, no importa si utilizamos todos los threads que nuestro sistema nos permite, no permitirá que el programa esté bien optimizado o que corra fluido, pero sí se verán mejoras en comparación al programa secuencial. Cuando los threads se alinean lógicamente con el número de círculos que se desea renderizar, el programa paralelo alcanza su máximo potencial, lo que subraya la importancia de una estrategia de paralelización bien planificada.

Por último, se encontró que la eficiencia de un programa paralelo no se determina únicamente por la cantidad de threads utilizados, sino también por cómo se relacionan con la arquitectura del sistema. Se encontró que el programa paralelo alcanza su rendimiento óptimo cuando se ejecuta en un número de threads (N) que sea múltiplo de la cantidad total de threads del sistema. Esto se puede observar en nuestros resultados cuando se utilizaba 5 threads, que no es múltiplo de la cantidad total. Cuando se corría de esta manera, el programa no gozaba de la verdadera optimización que ofrece el paralelismo, y el speedup no es tan grande en comparación al programa secuencial. Esta relación sugiere que, para maximizar el rendimiento, es esencial considerar no sólo la lógica del programa, sino también las características del hardware en el que se ejecuta.

## Conclusiones

- La paralelización del movimiento de los círculos genera un speedup bastante significativo, especialmente cuando los threads tienen sentido con el número de círculos que se desea renderizar.
- La diferencia entre ambos programas no es tan drástica como se esperaba debido a que el programa secuencial está muy bien optimizado.
- La paralelización de los programas funciona mejor cuando se corre en N threads y N es un múltiplo de la cantidad total de threads de nuestro sistema.

## Recomendaciones

La primera recomendación y la más importante es la de correr el programa directamente desde Linux, o por lo menos, en una máquina virtual de gran potencia, ya que en WSL donde se realizó, a pesar de obtener resultados muy buenos, es una optimización grande que podemos realizar para poder alocar más recursos a esto.

Otra recomendación podría ser la optimización del renderizado de los círculos para que el programa en general pueda correr mucho más fluido sin importar si se está corriendo en secuencial o paralelo.

Por último, se recomienda que también se implemente la paralelización en la creación de los círculos, esto se podría realizar con un grid para evitar que se creen círculos de manera repetida y que el programa paralelo pueda tener mejor rendimiento incluso antes de que los círculos se empiecen a mover.

## Bibliografía


- Chandra, R. (2001). Parallel programming in OpenMP. Morgan Kaufmann.
- Lo, J. L., Emer, J. S., Levy, H. M., Stamm, R. L., Tullsen, D. M., & Eggers, S. J. (1997). Converting thread-level parallelism to instruction-level parallelism via simultaneous multithreading. ACM Transactions on Computer Systems (TOCS), 15(3), 322-354.
- Madhav, S. (2018). Game Programming in C++: Creating 3D Games. Addison-Wesley Professional.


**Link de github: https://github.com/GitElean/paralela_proyecto**

# Anexos

Anexo 1: Diagrama de flujo del programa

Anexo 2: Catálogo de funciones

1. main.cpp:

Punto de Entrada del Programa
Incluir los archivos de cabecera necesarios.
Solicitar al usuario que elija entre ejecutar un programa secuencial, un programa paralelo o salir.
Ejecutar el protector de pantalla correspondiente o terminar el programa según la elección del usuario.

2. secuencial.cpp:

Clase ScreenSaver (Protector de pantalla secuencial)

initializeCircles()
Inicializar círculos con valores aleatorios.
Generar valores aleatorios para posición, radio, velocidad y color de cada círculo.

moveCircles()
Mover cada círculo basándose en su velocidad actual.
Verificar colisiones con bordes y ajustar la dirección de movimiento si es necesario.

drawCircles()
Establecer el color de fondo de la ventana.
Dibujar cada círculo píxel por píxel.
Presentar el contenido dibujado en la ventana.

calculateFPS()
Calcular y retornar los FPS (Frames Per Second) actuales.

run()
Bucle principal para mover y dibujar círculos hasta que se cierra la ventana.

3. paralelo.cpp:

Clase ParallelScreenSaver (Protector de pantalla paralelo)

initializeCircles()
Similar al método en ScreenSaver.

moveCircles() (con OpenMP)
Mover círculos en paralelo utilizando múltiples threads.
Verificar colisiones con bordes y ajustar la dirección de movimiento si es necesario.

drawCircles() (con SDL2_gfx)
Establecer el color de fondo de la ventana.

Dibujar círculos rellenos de manera optimizada.

Presentar el contenido dibujado en la ventana.

calculateFPS()

Similar al método en ScreenSaver.

run()

Bucle principal para mover y dibujar círculos en paralelo hasta que se cierra la ventana.

4. Limpieza

Destruir los objetos de círculo.

Limpiar los recursos de SDL (destruir renderizador y ventana).

Salir de SDL.

Retornar desde la función principal.

Anexo 3: Bitácora de pruebas

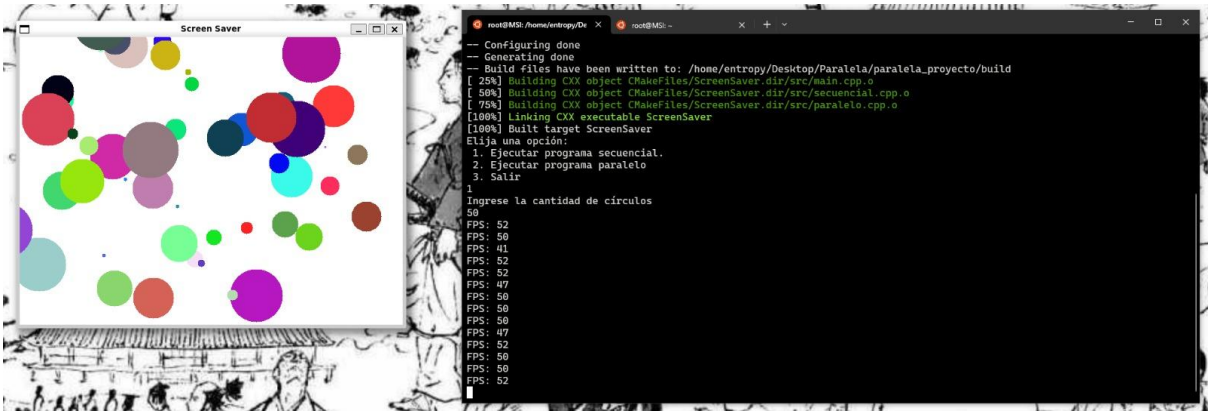


Imagen 1: Prueba con 50 pelotas secuencial.

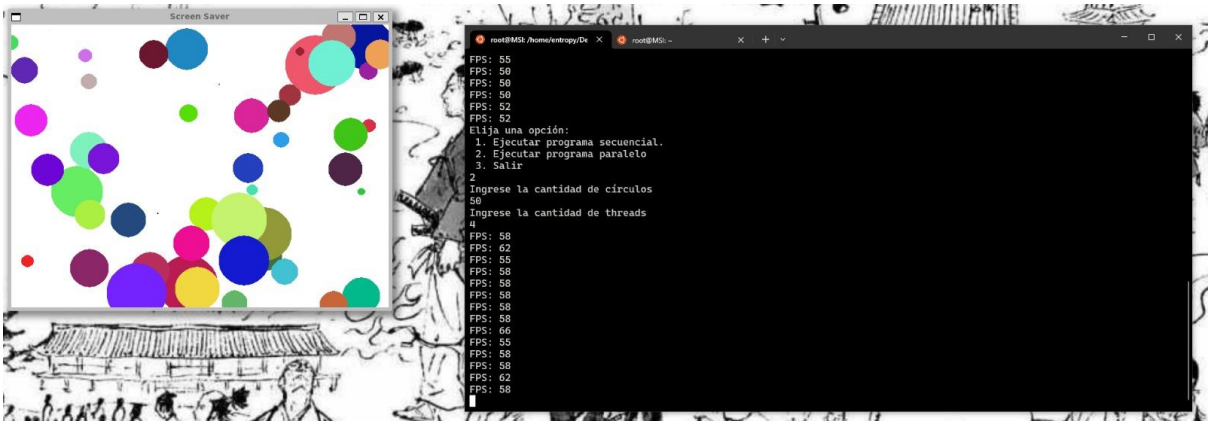


Imagen 2: Prueba con 50 pelotas y 4 threads.

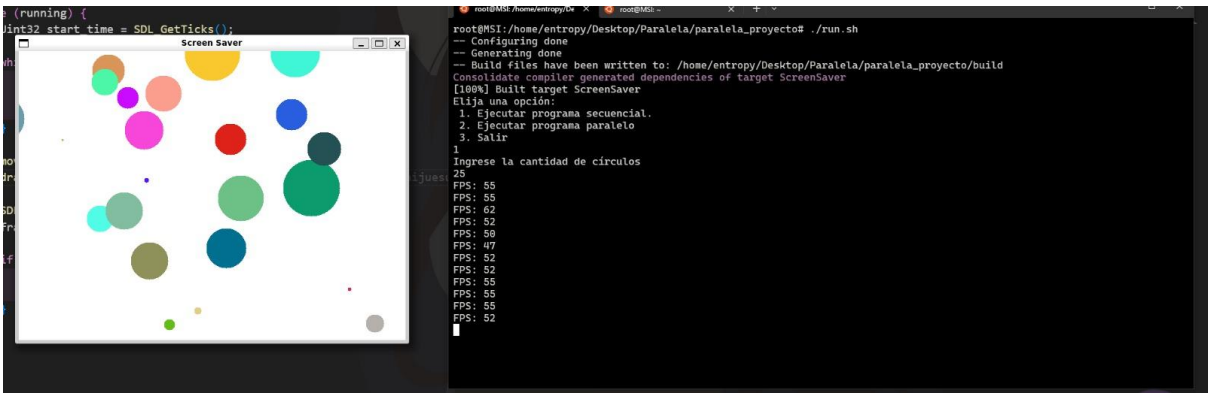


Imagen 3: Prueba con 25 pelotas secuencial.

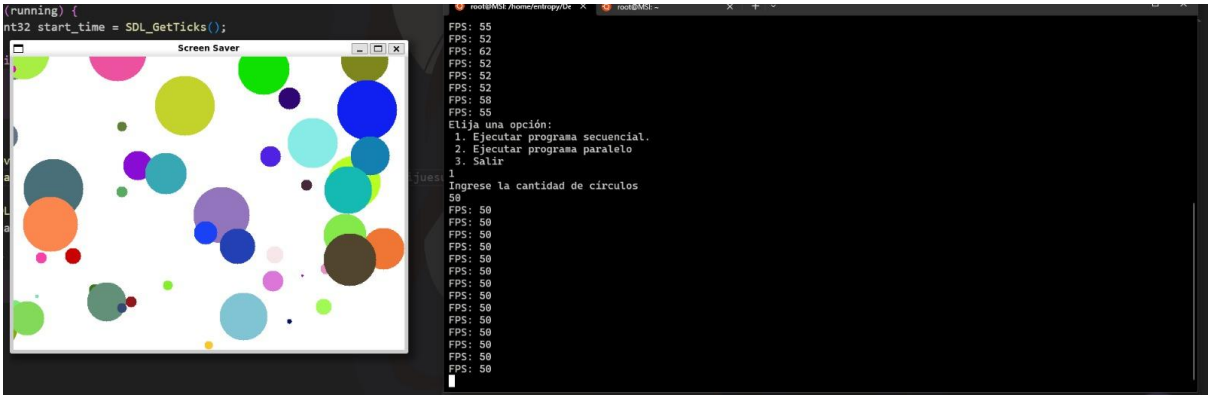


Imagen 4: Prueba con 50 pelotas secuencial.

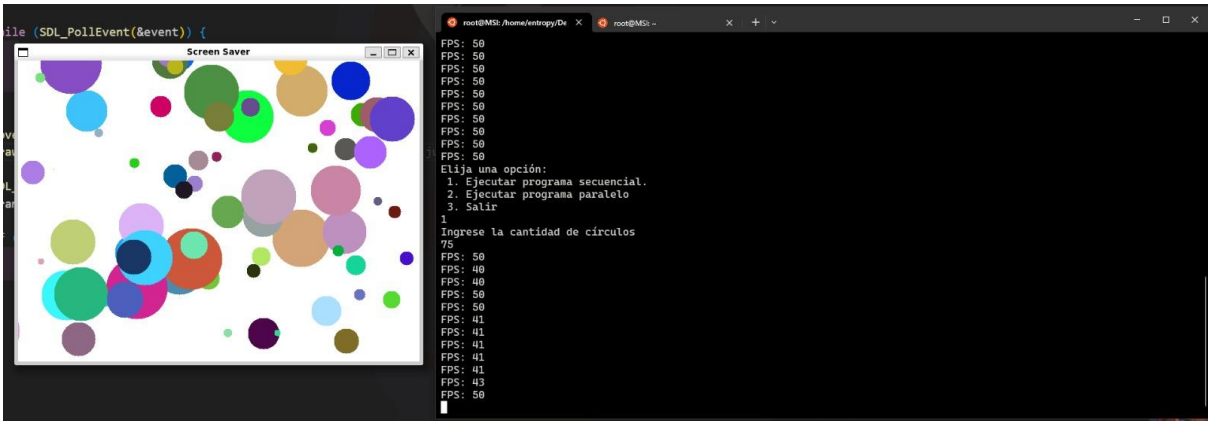


Imagen 5: Prueba con 75 pelotas secuencial.

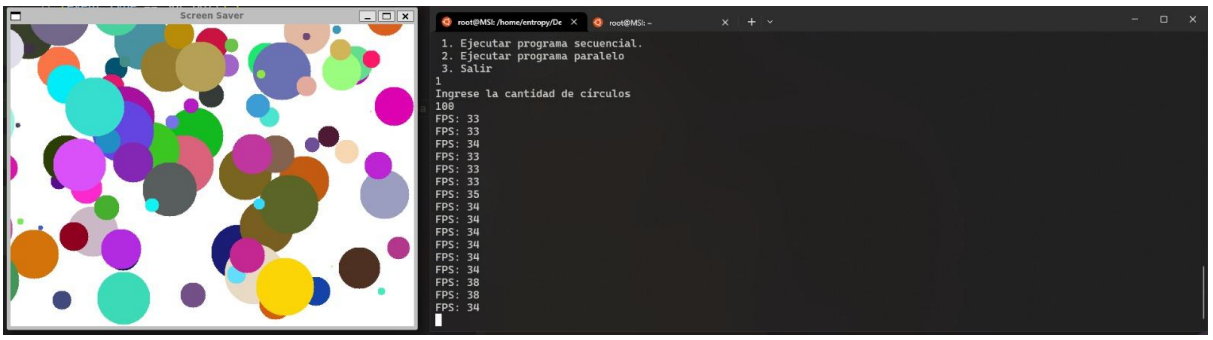


Imagen 6: Prueba con 100 pelotas secuencial.

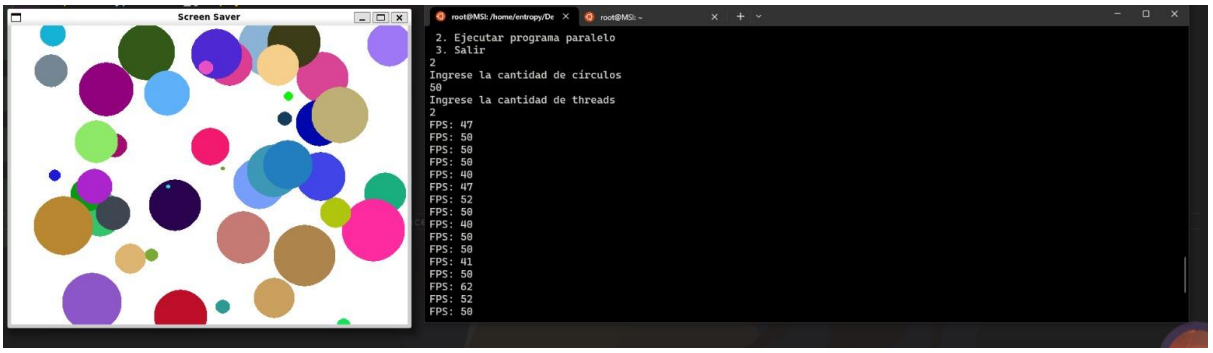


Imagen 7: Prueba con 50 pelotas y 2 threads.


Imagen 8: Prueba con 50 pelotas y 4 threads.

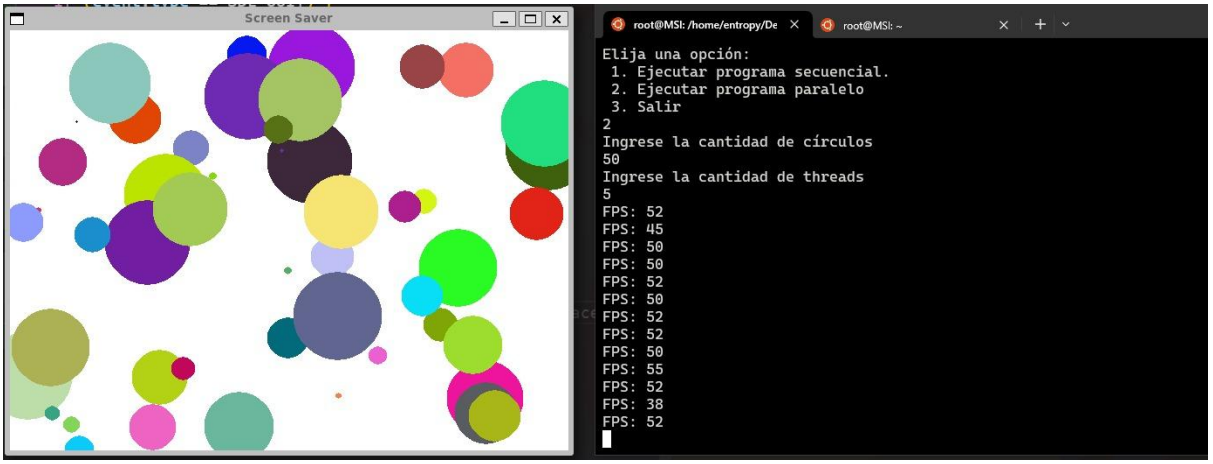


Imagen 9: Prueba con 50 pelotas y 5 threads.


Imagen 10: Prueba con 50 pelotas y 8 threads.

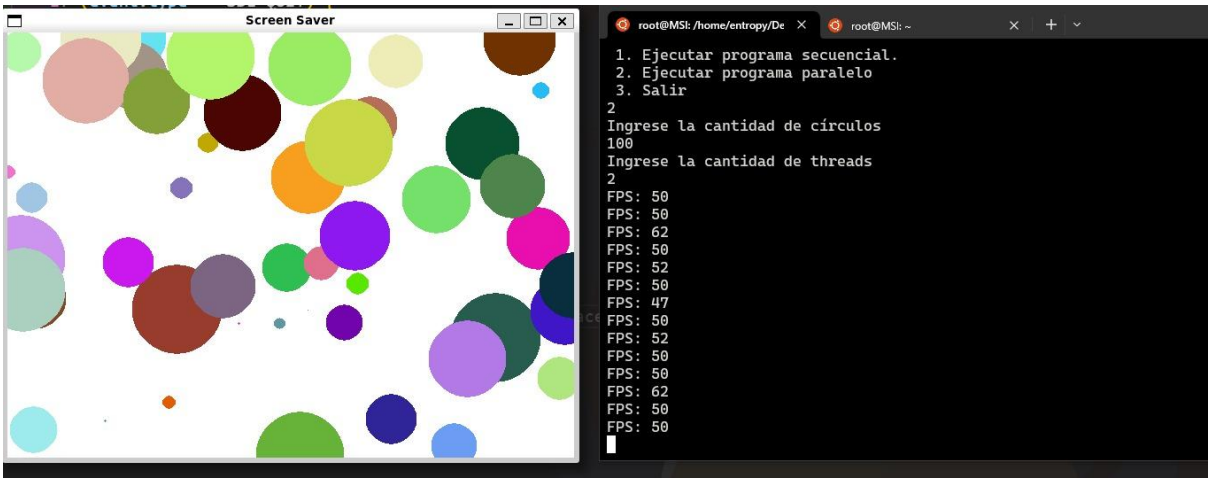


Imagen 11: Prueba con 100 pelotas y 2 threads.

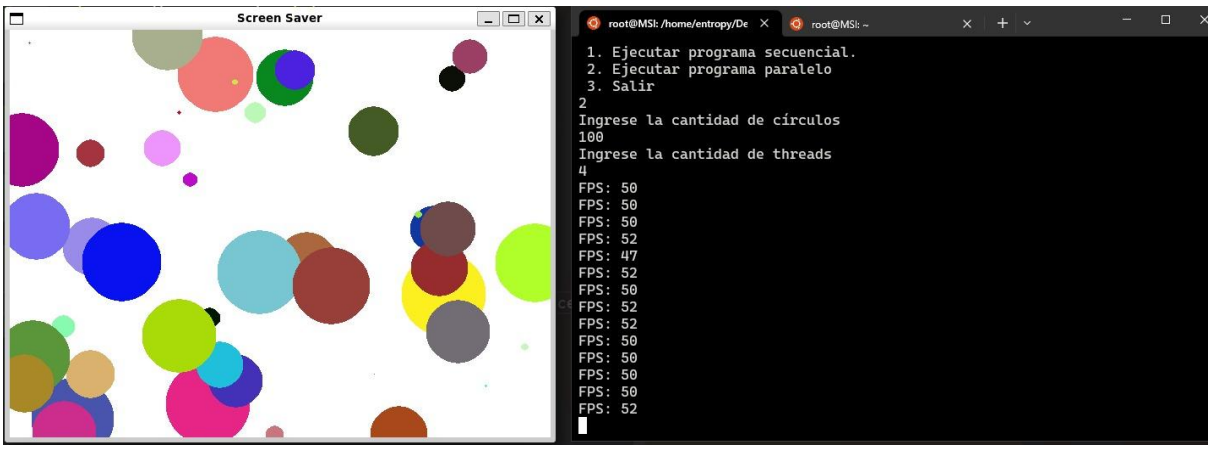


Imagen 12: Prueba con 100 pelotas y 4 threads.

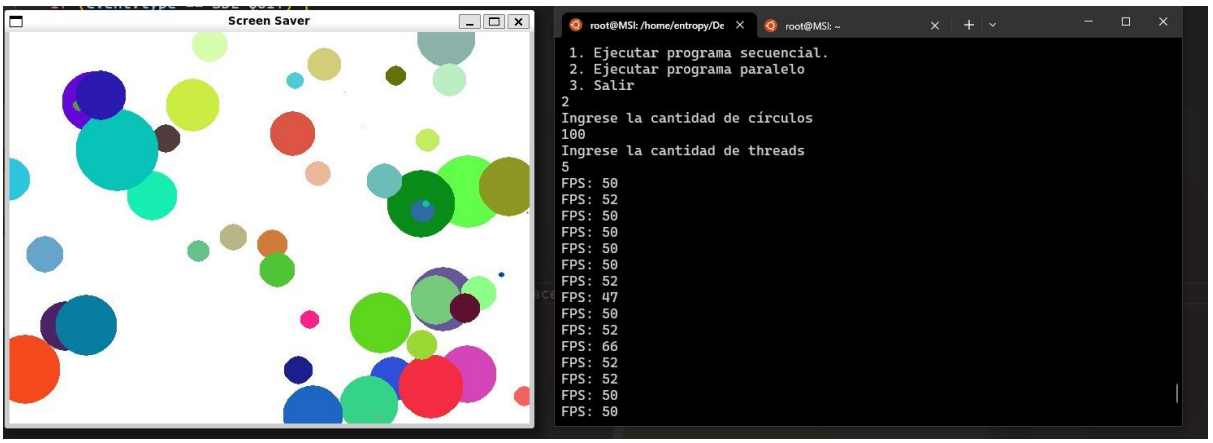


Imagen 13: Prueba con 100 pelotas y 5 threads.


Imagen 14: Prueba con 100 pelotas y 8 threads.