

UNIVERSIDAD DEL VALLE DE GUATEMALA

Computación Paralela

Sección 11



Excelencia que trasciende

DEL VALLE
GRUPO EDUCATIVO

“Proyecto 1”

ESTEBAN ALDANA GUERRA 20591

JESSICA PAMELA ORTIZ IXCOT 20192

GABRIEL ALEJANDRO VICENTE LORENZO 20498

GUATEMALA, Septiembre 2023

Índice

	Página
● Índice del informe	1
● Introducción	2
● Antecedentes	3
● Análisis de la primera iteración del proyecto	4
● Análisis de la segunda iteración del proyecto	5
● Análisis de la iteración final del proyecto	6
● Diagrama y catálogo	7
● Bitácora de pruebas	8 - 13
● Conclusiones	14
● Recomendaciones	15
● Bibliografía	16

Introducción

La programación paralela es una técnica fundamental en la informática moderna que permite aprovechar al máximo los recursos de hardware disponibles para resolver problemas de manera más eficiente. En este contexto, el presente informe se enfoca en el uso de OpenMP como herramienta para transformar programas secuenciales en aplicaciones paralelas, abriendo las puertas a un rendimiento significativamente mejorado.

En este informe, se describen los objetivos y competencias que se espera que los equipos de tres integrantes logren al implementar y diseñar programas para la paralelización de procesos con memoria compartida utilizando OpenMP. Se explorarán conceptos como el método PCAM y patrones de descomposición, con el fin de transformar programas secuenciales en aplicaciones paralelas más eficientes. Además, se promoverán mejoras y modificaciones iterativas para alcanzar versiones optimizadas, todo en busca de un aumento en el desempeño, basándonos en los conceptos de speedup y eficiencia estudiados en el proceso.

Antecedentes

En el vasto panorama de la programación paralela y el uso de OpenMP, es evidente que los antecedentes disponibles en internet brindan una visión enriquecedora de cómo esta tecnología ha revolucionado el campo de la computación. En línea, se pueden encontrar numerosos recursos que detallan el desarrollo histórico de OpenMP. Sin embargo al intentar buscar referencias junto a SDL (que corresponde a la librería utilizada para generar el screensaver en este proyecto) la búsqueda se vuelve un poco más específica, refiriéndose principalmente a problemas específicos presentados en hilos de programación que ciertas personas tuvieron al momento de intentar implementar estas dos tecnologías juntas (Simple Directmedia Layer, 2008).

Esto da una perspectiva respecto al reto que presenta este proyecto, o al menos la dificultad que presenta una implementación coherente del mismo que de resultados óptimos y destacables. A lo largo del informe se van explicando las dificultades y retos que se fueron presentando, además de las implementaciones que se utilizaron para obtener los mejores resultados posibles.

Es esencial tener en cuenta que la investigación llevada a cabo para poder proseguir con este proyecto se basó principalmente en encontrar implementaciones puntuales que resolvieran los problemas que presentaba pasar de la implementación secuencial a la implementación paralela.

Por lo que los antecedentes varían desde la aplicación de directivas de OpenMP en casos simples, las distintas funciones que presenta SDL y cómo al renderizar se tienen que tener ciertas precauciones al momento de intentar aplicar paralelismo (SDL2/FrontPage, 2023). Debido a los conflictos que puede presentar intentar renderizar varios objetos al mismo tiempo, además de que interactúen entre sí (qué es lo que se intentó implementar en este proyecto)

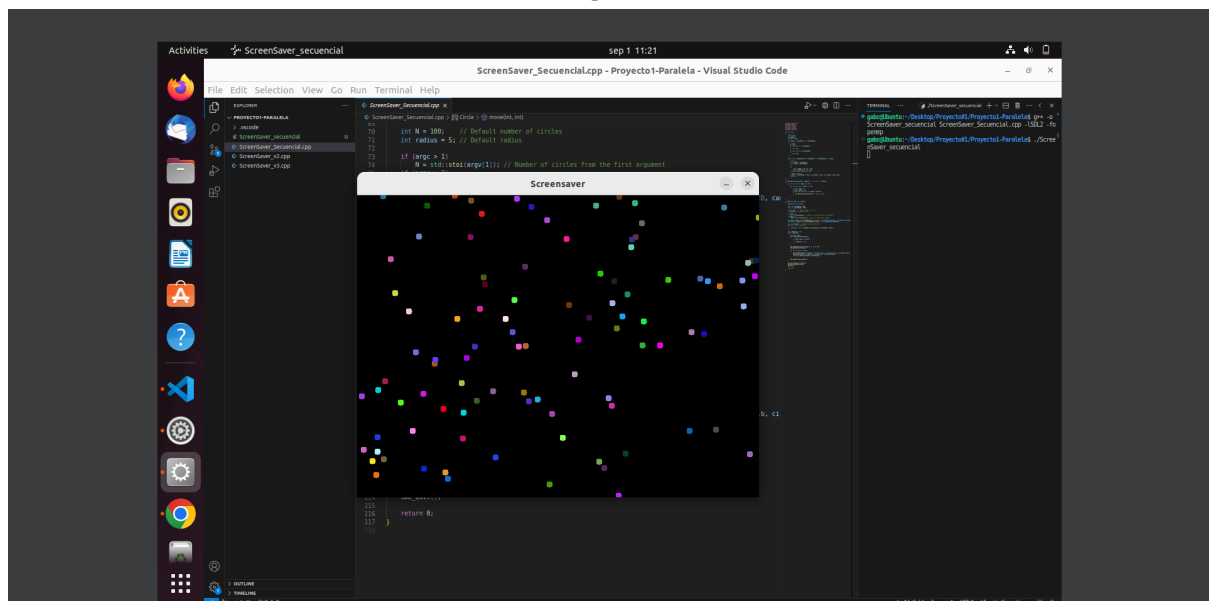
Análisis de la primera iteración del proyecto

Describiendo brevemente la secuencia que tuvo el desarrollo de este proyecto se realizaron dos versiones del programa secuencial y finalmente una tercera versión que cumplía con todos los ámbitos requeridos por este proyecto, entiéndase programación defensiva, medición de tiempos y paralelización utilizando openmp. A continuación se presenta la descripción de la primer versión secuencial de este proyecto que tiene como objetivo el diseño simple del screensaver que se desea, con ciertos detalles aún sin pulir pero que es funcional e introduce rápidamente al uso de SDL. El nombre de este programa consiste en “ScreenSaver_Secuencial.cpp” que se puede encontrar entre los archivos agregados en la entrega y en el propio repositorio de este proyecto. El código comienza por definir una estructura llamada Circle, que representa un círculo con propiedades como posición (x, y), velocidad (dx, dy), radio y color. Luego, se implementan funciones para mover estos círculos dentro de los límites del lienzo y para dibujar círculos en el renderer SDL.

En la función main, se configura la ventana SDL y se crea un conjunto de círculos basados en los argumentos de línea de comandos proporcionados por el usuario o utilizando valores predeterminados. Los círculos se inicializan con posiciones y velocidades aleatorias, así como colores aleatorios. El bucle principal de la aplicación se encarga de actualizar y renderizar los círculos en cada iteración. Además, maneja eventos de salida para que el programa se pueda cerrar al hacer clic en el botón de cierre de la ventana.

En resumen, este programa crea una animación visual atractiva que simula un efecto de salvapantallas con una variedad de círculos en movimiento constante que rebotan dentro de una ventana rectangular. Los detalles específicos de la animación, como el número de círculos y su radio, pueden personalizarse mediante argumentos de línea de comandos. A continuación se agrega una captura de pantalla de la visualización que presenta el programa

Imagen #1



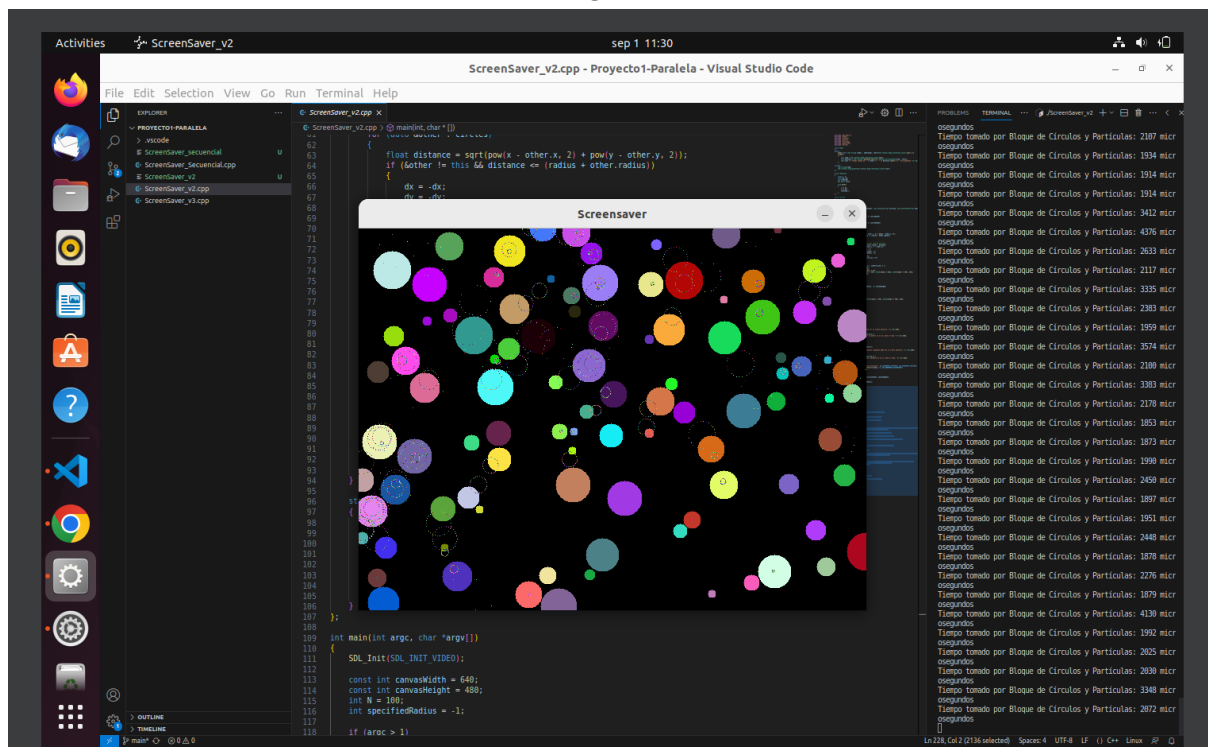
Captura de pantalla de resultado de programa secuencial original

Análisis de la segunda iteración del proyecto

En la segunda versión del programa secuencial se completaron los requisitos solicitados por las especificaciones del proyecto. Además, de que se agregaron ciertas estructuras (Timer, Particle) y se modificó la estructura original de Circle con el objetivo de lograr medir los tiempos de renderización, agregar animaciones de partículas a mover los círculos y que estos colisionan, además de que se realizaron modificaciones al main para que tuviera programación defensiva al momento de recibir parámetros de terminal.

Dentro del bucle principal, el programa maneja eventos de salida y controla la lógica de la animación. Los círculos se mueven y rebotan dentro de los límites del lienzo, detectando colisiones entre ellos. Cuando se produce una colisión, se generan partículas que se dispersan desde el punto de colisión, creando un efecto visual interesante. La aplicación también mide y muestra la velocidad de fotogramas (FPS) en la consola para controlar el rendimiento. En resumen, este programa crea un salvapantallas dinámico con círculos que se mueven, rebotan y generan partículas al colisionar, ofreciendo una animación visualmente atractiva y configurable. A continuación una captura de pantalla del resultado de esta iteración:

Imagen #2



Captura de pantalla de resultado de la segunda versión del programa

Análisis de la iteración final del proyecto

De manera concisa y breve a continuación se describe la iteración final del proyecto. Dentro de esta versión final del programa el objetivo fue realizar la parte más importante del proyecto que consiste en intentar paralelizar el programa de tal manera que se tenga una mejora en el rendimiento y los tiempos de renderización. Se estuvieron probando varios cambios en los cuales se intentaba lograr que el programa se ejecutará de una mejor manera, los cambios que resultaron consistentes correspondían a la paralelización con la directiva `#pragma omp parallel for` de la creación de círculos, la asignación de su radio y su almacenamiento de un vector, de tal manera que la renderización se realizará más rápida.

Además, se logró apreciar que las partículas podían tener un comportamiento paralelo sin presentar race conditions dentro del programa debido a que no dependen más que de la colisión entre los círculos en movimiento. A continuación se presentan las descripciones finales del programa, las cuales incluyen el diagrama de flujo, el catálogo de funciones y estructuras, además de la bitácora de pruebas con las capturas de pantalla correspondientes.

Anexo#1

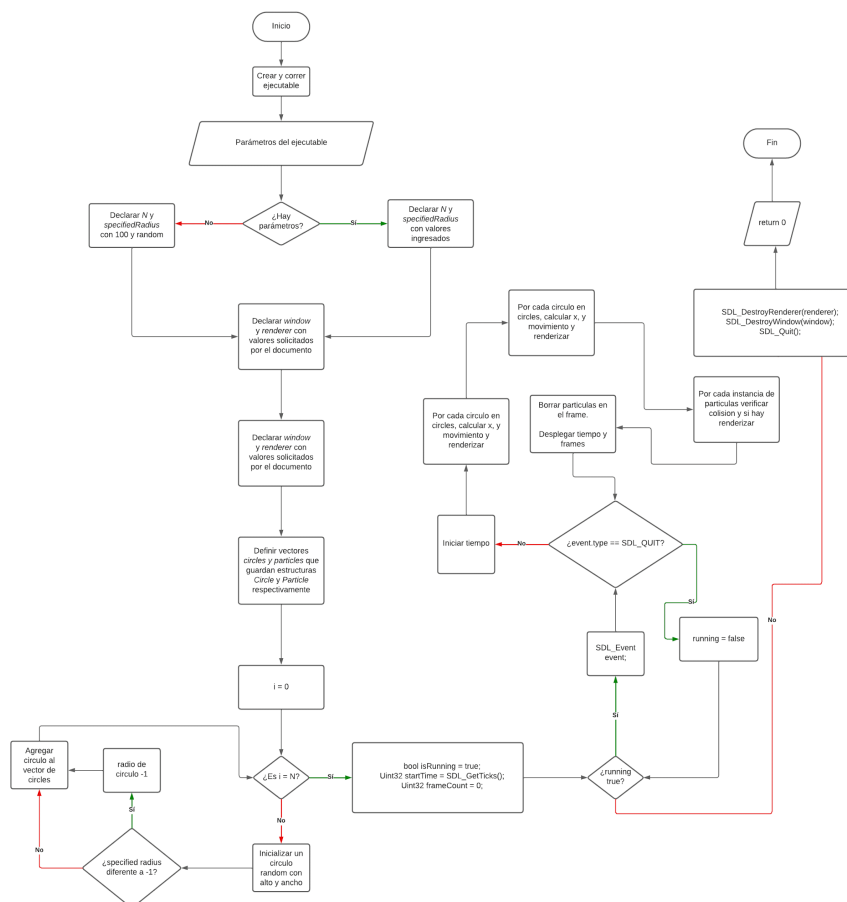


Diagrama de flujo del programa

Enlace al diagrama para mejor visualización

[Enlace de lucid chart](#)

Anexo#2: Catálogo de funciones y estructuras principales

Estructuras:

- Timer: Una clase que se utiliza para medir el tiempo transcurrido en un bloque de código.
- Particle: Una estructura que representa partículas en el salvapantallas, con propiedades como posición, velocidad, tiempo de vida y color.
- Circle: Una estructura que representa círculos en el salvapantallas, con propiedades como posición, velocidad, radio y color.

Funciones:

- main(): La función principal del programa que contiene la lógica principal.
- Circle::randomCircle(): Una función estática que genera un círculo aleatorio dentro de los límites de la ventana.

Variables:

- canvasWidth y canvasHeight: Representan el ancho y alto de la ventana del salvapantallas.
- N: El número de círculos que se generarán en el salvapantallas.
- specifiedRadius: Un valor opcional que especifica el radio de los círculos.
- window: Un puntero a la ventana de SDL.
- renderer: Un puntero al renderizador de SDL.
- circles: Un vector que almacena los círculos en el salvapantallas.
- particles: Un vector que almacena las partículas en el salvapantallas.
- isRunning: Una bandera que controla si el bucle principal sigue en ejecución.
- startTime: El tiempo en milisegundos en el que comenzó la ejecución del programa.
- frameCount: El número de cuadros dibujados por segundo.

Anexo#3: Bitácora de pruebas

Prueba #1

The screenshot shows a Visual Studio Code window with the following details:

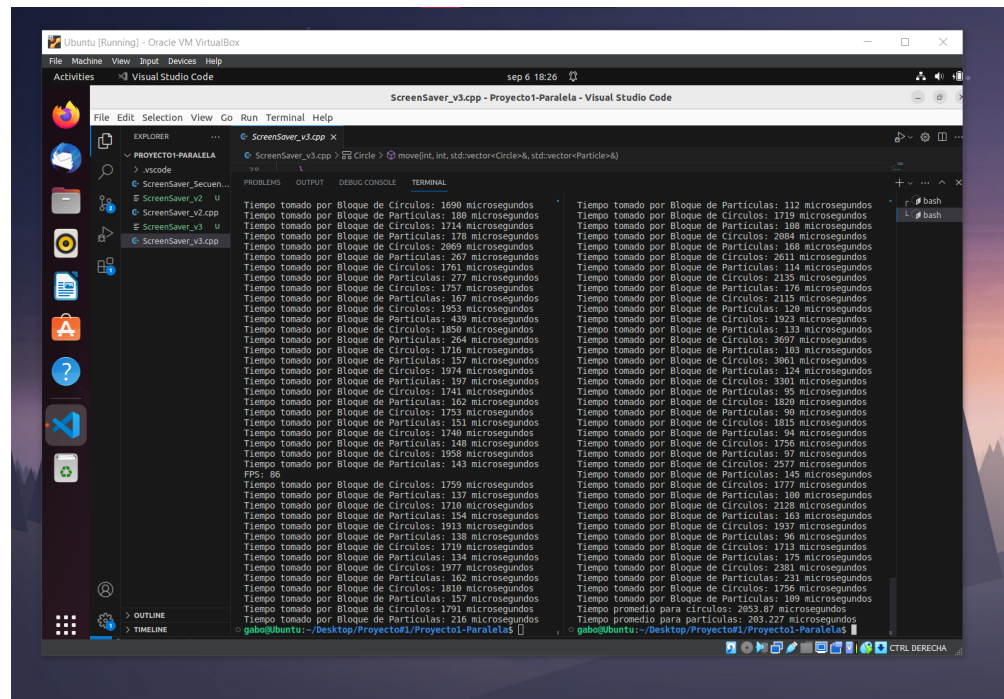
- File Explorer:** Shows the project structure for 'PROJECT01-PARALELA', including 'vscode', 'ScreenSaver_Secun...', 'ScreenSaver_v2.cpp', 'ScreenSaver_v3', and 'ScreenSaver_v3.cpp'.
- Terminal:** Displays the output of the program, showing 30 test cases. Each case includes a time taken (e.g., 'Tiempo tomado por Bloque de Partículas: 150 microsegundos') and a particle count (e.g., '122 microsegundos').
- Status Bar:** Shows the file path: 'gabo@Ubuntu: ~/Desktop/Project01/Project01-Paralela [C++]'. It also includes icons for Run and Debug, and a 'CTRL DIRECHA' button.

Prueba #2

Imagen ejecución	
Speedup	-279's 7's
Eficiencia	-12.3% 3.52%

Prueba #3

Imagen ejecución



Speedup

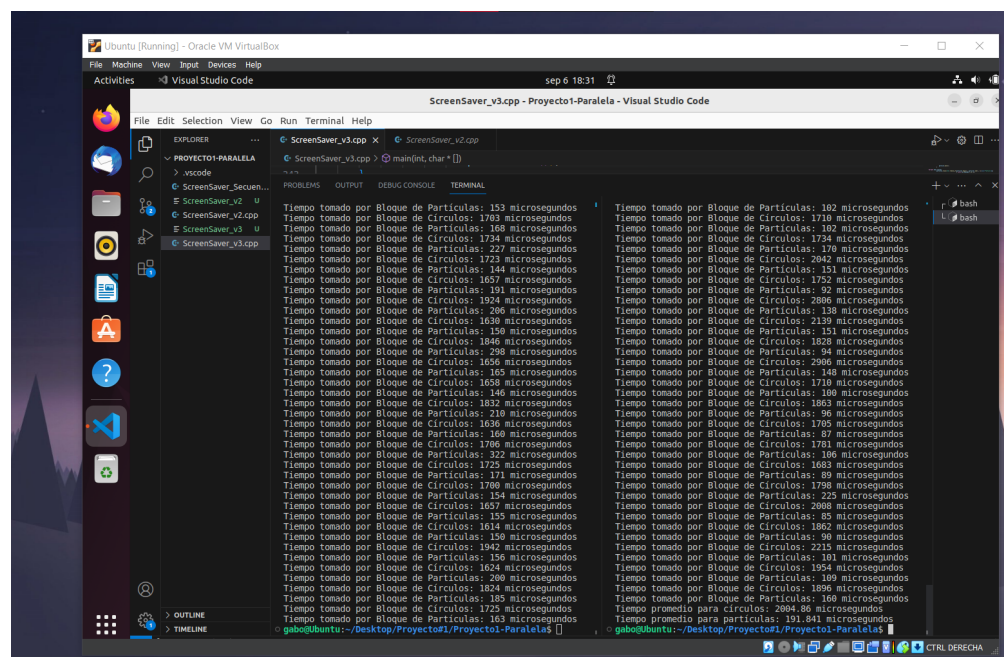
262's
-13's

Eficiencia

14.62869905%
-6.018518519%

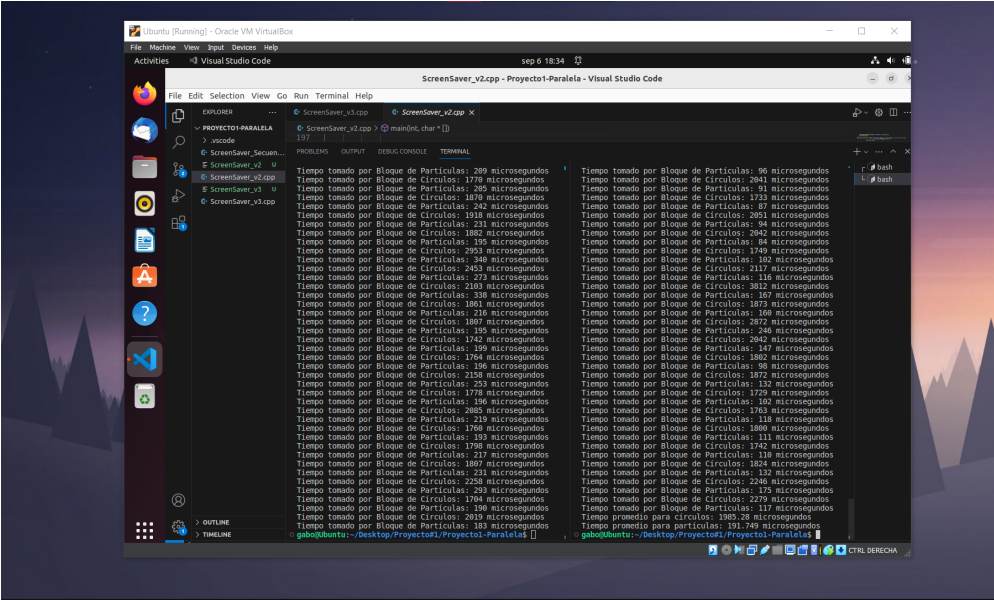
Prueba #4

Imagen
ejecución



Speedup	279's 28's
Eficiencia	16,17391304's 17,17791411's

Prueba #5

Imagen ejecución	
Speedup	-34's 8's
Eficiencia	-1.684001981% 4.371584699%

Prueba #6

Imagen
ejecución

The screenshot displays a Linux desktop environment. On the left is a vertical dock with various application icons. The top panel shows the system menu with 'Ubuntu [Running]' and 'Workspaces VirtualBox'. Below the dock, the 'Activities' overview is visible, showing a search bar and a list of open applications. The main workspace contains two windows:

- Terminal Window:** Displays the output of a script named 'ScreenSaver_V2.cpp'. The output shows a series of measurements for 'Tiempo promedio para Bloque de Partículas' (Average time for particle block) in microseconds. The values range from 154 to 2396 microseconds. The script concludes with 'Tiempo promedio para circuitos: 1989.31 microsegundos' and 'Tiempo promedio para partículas: 232.68 microsegundos'.
- Visual Studio Code Editor:** Opened to the file 'ScreenSaver_V2.cpp'. The editor shows the source code of the program, which includes a loop that generates random particle positions and calculates the time taken for a circuit to pass through them. The code is written in C++ and uses the `<random>` and `<chrono>` libraries.

The terminal window also shows a prompt for the user to press 'bash' to run the script.

Speedup	93's 36's
Eficiencia	4.60624071322437% 19.672131147541%

Prueba #7

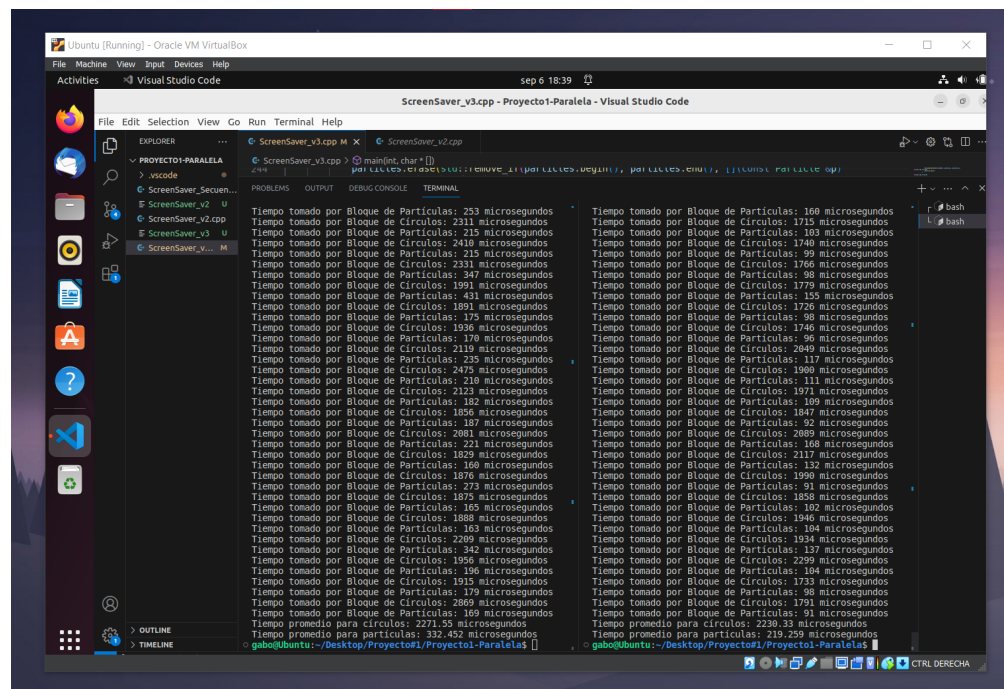
Imagen ejecución

[illegible]

Speedup	-189's -178's
Eficiencia	-8.65781035272561% -45.6410256410256%

Prueba #8

Imagen
ejecución



Speedup

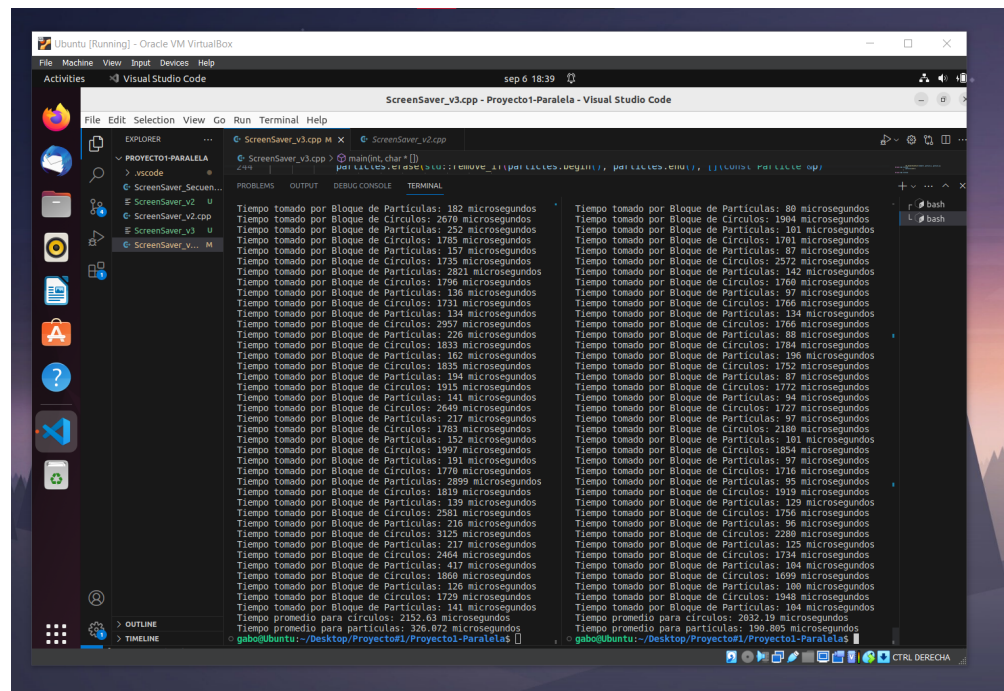
-41's
-113's

Eficiencia

-1.80537208278292%
-34.0361445783133%

Prueba #9

Imagen
ejecución



Speedup	-120's -136's
Eficiencia	-5,57620817843866% -41,7177914110429%

Prueba #10

Imagen ejecución

```

Tiempo tomado por Bloque de Partículas: 149 microsegundos
Tiempo tomado por Bloque de Partículas: 1702 microsegundos
Tiempo tomado por Bloque de Partículas: 196 microsegundos
Tiempo tomado por Bloque de Partículas: 1826 microsegundos
Tiempo tomado por Bloque de Partículas: 306 microsegundos
Tiempo tomado por Bloque de Partículas: 1738 microsegundos
Tiempo tomado por Bloque de Partículas: 1079 microsegundos
Tiempo tomado por Bloque de Partículas: 147 microsegundos
Tiempo tomado por Bloque de Partículas: 1745 microsegundos
Tiempo tomado por Bloque de Partículas: 1233 microsegundos
Tiempo tomado por Bloque de Partículas: 149 microsegundos
Tiempo tomado por Bloque de Partículas: 1825 microsegundos
Tiempo tomado por Bloque de Partículas: 142 microsegundos
Tiempo tomado por Bloque de Partículas: 1787 microsegundos
Tiempo tomado por Bloque de Partículas: 105 microsegundos
Tiempo tomado por Bloque de Partículas: 2056 microsegundos
Tiempo tomado por Bloque de Partículas: 180 microsegundos
Tiempo tomado por Bloque de Partículas: 1763 microsegundos
Tiempo tomado por Bloque de Partículas: 151 microsegundos
Tiempo tomado por Bloque de Partículas: 1843 microsegundos
Tiempo tomado por Bloque de Partículas: 170 microsegundos
Tiempo tomado por Bloque de Partículas: 2319 microsegundos
Tiempo tomado por Bloque de Partículas: 174 microsegundos
Tiempo tomado por Bloque de Partículas: 284 microsegundos
Tiempo tomado por Bloque de Partículas: 2585 microsegundos
Tiempo tomado por Bloque de Partículas: 153 microsegundos
Tiempo tomado por Bloque de Partículas: 2287 microsegundos
Tiempo tomado por Bloque de Partículas: 201 microsegundos
Tiempo tomado por Bloque de Partículas: 196 microsegundos
Tiempo tomado por Bloque de Partículas: 1697 microsegundos
Tiempo tomado por Bloque de Partículas: 150 microsegundos
Tiempo promedio para partículas: 1970.09 microsegundos

Tiempo tomado por Bloque de Partículas: 149 microsegundos
Tiempo tomado por Bloque de Partículas: 1894 microsegundos
Tiempo tomado por Bloque de Partículas: 145 microsegundos
Tiempo tomado por Bloque de Partículas: 2070 microsegundos
Tiempo tomado por Bloque de Partículas: 146 microsegundos
Tiempo tomado por Bloque de Partículas: 2140 microsegundos
Tiempo tomado por Bloque de Partículas: 183 microsegundos
Tiempo tomado por Bloque de Partículas: 1774 microsegundos
Tiempo tomado por Bloque de Partículas: 141 microsegundos
Tiempo tomado por Bloque de Partículas: 2414 microsegundos
Tiempo tomado por Bloque de Partículas: 136 microsegundos
Tiempo tomado por Bloque de Partículas: 1980 microsegundos
Tiempo tomado por Bloque de Partículas: 154 microsegundos
Tiempo tomado por Bloque de Partículas: 1980 microsegundos
Tiempo tomado por Bloque de Partículas: 136 microsegundos
Tiempo tomado por Bloque de Partículas: 2838 microsegundos
Tiempo tomado por Bloque de Partículas: 178 microsegundos
Tiempo tomado por Bloque de Partículas: 2035 microsegundos
Tiempo tomado por Bloque de Partículas: 164 microsegundos
Tiempo tomado por Bloque de Partículas: 1989 microsegundos
Tiempo tomado por Bloque de Partículas: 156 microsegundos
Tiempo tomado por Bloque de Partículas: 1889 microsegundos
Tiempo tomado por Bloque de Partículas: 154 microsegundos
Tiempo tomado por Bloque de Partículas: 1844 microsegundos
Tiempo tomado por Bloque de Partículas: 205 microsegundos
Tiempo tomado por Bloque de Partículas: 1844 microsegundos
Tiempo tomado por Bloque de Partículas: 140 microsegundos
Tiempo tomado por Bloque de Partículas: 1793 microsegundos
Tiempo tomado por Bloque de Partículas: 138 microsegundos
Tiempo tomado por Bloque de Partículas: 2033 microsegundos
Tiempo tomado por Bloque de Partículas: 133 microsegundos
Tiempo tomado por Bloque de Partículas: 1792 microsegundos
Tiempo tomado por Bloque de Partículas: 119 microsegundos
Tiempo promedio para partículas: 216.996 microsegundos

```

Speedup	76's -133's
Eficiencia	3,85786802030457% -38,10888252149%

Conclusiones

- A través del proyecto, se aprendió la importancia de dividir tareas de manera eficiente entre diferentes hilos para lograr una paralelización efectiva. Esto se manifestó especialmente cuando se notó que una distribución desequilibrada de la carga de trabajo entre los hilos podía llevar a un rendimiento subóptimo.
- Trabajar con OpenMP nos permitió entender cómo la memoria compartida funciona en un entorno de programación paralela.
- Ambos programas(secuencial y paralelizado) tienen sus usos y escenarios donde podrían ser más apropiados. El primero podría ser más adecuado para un entorno donde las características adicionales y la eficiencia son críticas, mientras que el segundo podría ser más fácil de entender y modificar para un principiante o para un proyecto más pequeño.

Recomendaciones

Para enriquecer el programa y el proyecto es fundamental considerar la incorporación de características adicionales para evaluar. Esto puede incluir la implementación de efectos visuales avanzados, interacciones de usuario más ricas, configuraciones personalizadas y opciones de ajuste de rendimiento. Además, sería beneficioso ampliar la variedad de elementos visuales, explorando diferentes formas geométricas además de círculos y partículas para aumentar la complejidad visual requerida.

Respecto al uso de OpenMP y SDL sería conveniente proporcionar ejemplos prácticos y desafíos que permitan a los estudiantes aplicar los conceptos aprendidos en proyectos reales. También es esencial brindar consejos específicos para optimizar el rendimiento del código, fomentando el uso de herramientas de perfilado y destacando buenas prácticas de codificación. Estas recomendaciones enriquecerán la experiencia de aprendizaje y da los elementos necesarios para enfrentar desafíos en el desarrollo de paralelización de programas de manera efectiva y creativa.

Bibliografia

- *OpenMP 4.0 API C/C++ Syntax Quick Reference Card C/C++*. (n.d.).
<https://www.openmp.org/wp-content/uploads/OpenMP-4.0-C.pdf>
- *SDL2/FrontPage*. (2023). Libsdl.org. <https://wiki.libsdl.org/SDL2/FrontPage>
- Simple Directmedia Layer. (2008, July 7). *SDL and OpenMP*. Simple Directmedia Layer. <https://discourse.libsdl.org/t/sdl-and-openmp/15441>