UNIVERSIDAD DEL VALLE DE GUATEMALA
Computación Paralela y Distribuida
Sección 20
Ing. Juan Luis Garcia

Proyecto # 1

Mark Albrand, 21004
Jimena Hernandez, 21199
Javier Heredia, 21600

Guatemala, 30 de agosto de 2024

Índice

Introducción 3
Antecedentes 3
Discusión 3
Versión secuencial 4
Versión Paralela 5
Comparación 6
Conclusiones 7
Recomendaciones 7
Bibliografía 7
Anexos 8

Introducción

Actualmente, la habilidad de adaptar y mejorar los sistemas operativos es de suma importancia en el área de computación. OpenMP, en particular, se ha convertido en una herramienta clave para quienes trabajan con programación paralela en sistemas que comparten memoria. Este documento tiene como objetivo investigar a fondo OpenMP, desarrollando un screensaver de manera secuencial y de forma paralela usando OpenMP. Esto nos permitirá no solo observar las ventajas del paralelismo, sino que también la mejora en la velocidad y eficiencia. De igual forma nos permitirá entender mejor cómo implementar y enfrentar los desafíos que implica cambiar de un código secuencial a uno paralelo con OpenMP.

Antecedentes

OpenMP es una herramienta diseñada para simplificar la transición de programas secuenciales a programas paralelos, permitiendo una programación paralela de alto nivel, es útil en sistemas de memoria compartida.

La transformación iterativa de programas secuenciales a paralelos con OpenMP no solo simplifica el uso eficiente de múltiples núcleos de procesamiento, sino también contribuye a mantener un menor nivel de sobrecarga en la ejecución secuencial del programa.

El modelo de programación paralela que aplica OpenMP es Fork - Join. En un determinado momento, el thread “master” genera p threads que se ejecutan en paralelo. Los threads creados por el thread master van a ejecutar todos la misma copia de código y cada thread lleva un identificador. Echaiz, J. (n.d.).

Discusión

En este proyecto se evaluaron dos programas, uno secuencial y uno paralelo a través de OpenMP implementando el mismo diseño de screensaver. Se optó por realizar un diseño vintage de *dvd* que incluye elementos de física, en este caso los elementos pueden rebotar en los bordes y al rebotar crean un nuevo elemento. De igual forma, al colisionar uno con otro hay una probabilidad de 20% que se elimine uno de los dos elementos, los dos o ninguno. Cabe mencionar que cuando se elimina una figura, hay una probabilidad de ⅔ de que se pueda salvar. El objetivo principal de este proyecto es evaluar el rendimiento de ambas versiones. Para esto, se genera un log en el que se registra el tiempo en milisegundos que tarda en crearse cada frame, el tiempo en milisegundos que tarda en desplegarse en pantalla y la cantidad de FPS (cuadros por segundo) en ese momento. Para realizar las pruebas de rendimiento, se variaron parámetros como el ancho y alto de la pantalla (*SCREEN_WIDTH* y *SCREEN_HEIGHT*), el número de objetos iniciales en el screensaver y la duración en frames de cada explosión. Es importante destacar que el límite de elementos que nuestra pantalla puede mostrar está determinado por el tamaño de la pantalla, el tamaño de la imagen, y nuestra función de overlaping, la cual verifica que no

se generen imágenes una encima de otra. Esto implica que, en pantallas de tamaño reducido, el número máximo de elementos que pueden visualizarse simultáneamente es menor. Estos parámetros nos permiten identificar si existe una carga computacional significativa y, posteriormente, compararlos para evaluar la mejora que ofrece la versión paralela en relación con la secuencial.

Versión secuencial

Al analizar la versión secuencial, se observó que al incrementar los parámetros, los FPS muestran un comportamiento interesante, este comportamiento puede ser visualizado en la bitácora de pruebas realizadas. Aunque los FPS promedio aumentan, también se observan picos altos, lo que sugiere una carga computacional elevada. Los resultados específicos de las pruebas muestran cómo los diferentes tamaños de pantalla y el número de objetos influyen en el rendimiento:

Configuración: Figuras: 100, Frames: 50, Resolución: 1920x1080

- Avg Display Time: 23.92 ms
- Avg FPS: 55.60
- Max FPS: 125.00
- Min FPS: 31.20
- Avg Figures: 358.48

Configuración: Figuras: 500, Frames: 50, Resolución: 1920x1080

- Avg Display Time: 21.70 ms
- Avg FPS: 52.61
- Max FPS: 111.11
- Min FPS: 47.76
- Avg Figures: 345.68

Configuración: Figuras: 2000, Frames: 50, Resolución: 2500x1500

- Avg Display Time: 37.50 ms
- Avg FPS: 35.31
- Max FPS: 66.67
- Min FPS: 3.70
- Avg Figures: 658.75
- Max Figures: 1984.00

Configuración: Figuras: 5000, Frames: 50, Resolución: 5000x3000

- Avg Display Time: 56.42 ms
- Avg FPS: 24.72
- Max FPS: 52.63
- Min FPS: 0.73
- Avg Figures: 1575.39
- Max Figures: 4975.00

Versión Paralela

Para la versión paralela se hicieron algunos ajustes para separar la misma lógica pero en distintas secciones para poder paralelizar de mejor forma y poder dividir en partes pequeñas un mismo problema, el cambió realizado se hizo en la función de mover figuras ya que es una función que se usa continuamente en el programa y que debía de tener la lógica separada para poder ser paralelizada.

La paralelización que se implementó se ve reflejada en la tabla de paralelización ubicada en los anexos, en donde se explica cómo se utiliza cada método. Se trató de paralelizar la mayoría de las funciones en donde se realizan cálculos para poder obtener un mayor rendimiento.

Los resultados pueden ser visualizados en la bitácora de pruebas realizadas. Los resultados específicos de las pruebas muestran cómo los diferentes tamaños de pantalla y el número de objetos influyen en el rendimiento:

Configuración: Figuras: 100, Frames: 50, Resolución: 1920x1080

- Avg Display Time: 34.70 ms
- Avg FPS: 41.94
- Max FPS: 100.00
- Min FPS: 5.88
- Avg Figures: 367.12
- Max Figures: 592.00

Configuración: Figuras: 500, Frames: 50, Resolución: 1920x1080

- Avg Display Time: 24.47 ms
- Avg FPS: 46.48
- Max FPS: 111.11
- Min FPS: 0.48
- Avg Figures: 369.17
- Max Figures: 508.00

Configuración: Figuras: 2000, Frames: 50, Resolución: 2500x1500

- Avg Display Time: 33.03 ms
- Avg FPS: 43.48
- Max FPS: 100.00
- Min FPS: 1.65
- Avg Figures: 569.48
- Max Figures: 1927.00

Configuración: Figuras: 5000, Frames: 50, Resolución: 5000x3000

- Avg Display Time: 22.78 ms
- Avg FPS: 60.33
- Max FPS: 99.91
- Min FPS: 0.32
- Avg Figures: 695.03
- Max Figures: 4952.00

Comparación

Comparar ambos programas fue un reto, debido a la naturaleza de alta entropía de nuestro programa. Al manejar tantos comportamientos de manera aleatoria y con probabilidades, es difícil mantener un mismo ecosistema para probar uno contra el otro. En las pruebas adjuntas en los anexos se ejecutaron una gran cantidad de pruebas y se mantuvieron las que desarrollaron ambientes más parecidos.

En base a lo anterior, se puede observar una mejora del programa paralelo en contra del secuencial, tanto en la cantidad de fotogramas por segundo como en la latencia en crear dichos fotogramas. En nuestra prueba más exigente, fue el programa paralelo el que logró alcanzar el máximo promedio de fotogramas por segundo de todas las pruebas realizadas, pero debido a que el secuencial no logró alcanzar condiciones tan cercanas a este consideramos que no eran comparables del todo a las pruebas secuenciales realizadas.

Es importante mencionar que se paralizaron grandes secciones del programa, y este logró ser completamente estable en todas las pruebas realizadas, en el sentido de que no presenta errores como condiciones de carrera u otros problemas relacionados a la memoria, problemas de los cuáles este programa era bastante susceptible debido a los comportamientos diseñados.

Uno de los retos de la paralelización para este proyecto fue poder mantener la mayor cantidad de las funciones y métodos utilizados para manejar el programa. Funciones como la generación de números aleatorios de C provocaron una gran cantidad de problemas en nuestro proceso de paralelizar, por lo que se tuvieron que adoptar medidas que es probable que perjudiquen el rendimiento del programa en paralelo.

Conclusiones

- Se determinó que el programa paralelo tiende a incrementar el rendimiento a lo largo del tiempo, mientras que el secuencial no.
- La paralelización puede ser empleada de manera eficiente cuando la lógica del programa se encuentra lo suficientemente dividida en secciones pequeñas.
- Es complicado mantener las mismas funciones y comportamientos entre programas secuenciales y paralelos en C, debido a la falta de medidas estandarizadas para estos manejos de paralelización.

Recomendaciones

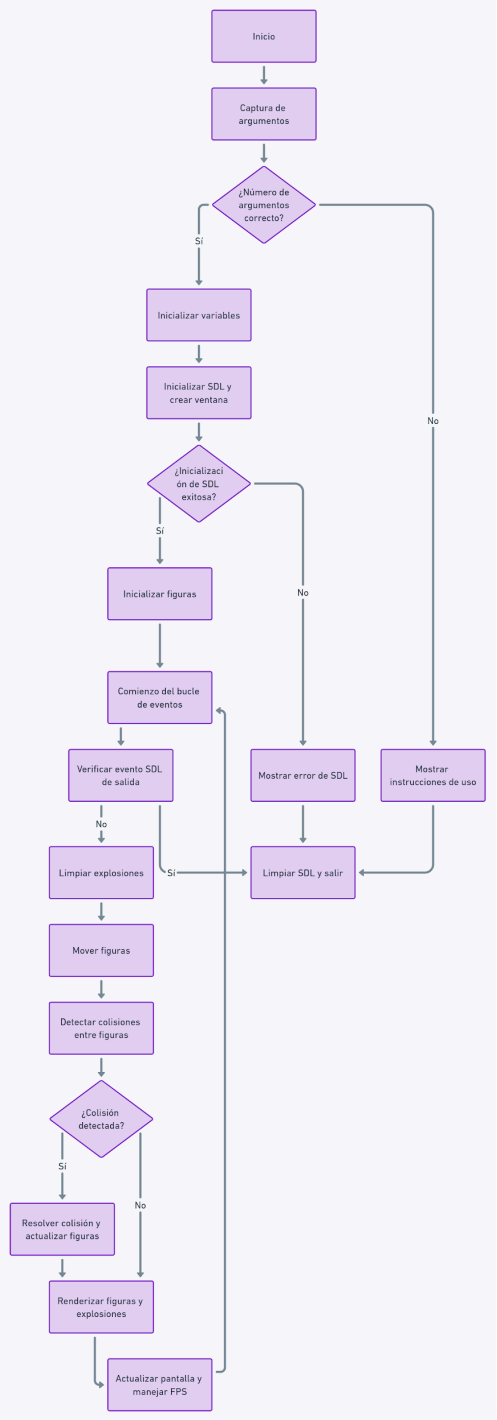
Para futuras pruebas, se recomienda separar la lógica de las funciones para que estas puedan ser paralelizadas sin mayor problema, esto usando la técnica de divide and conqueer en donde tomamos un problema grande y lo dividimos en subproblemas más pequeños para poder resolverlo.

De igual forma, se recomienda no utilizar tantas secciones críticas dentro de la paralelización a menos que sea necesario esto debido a que el uso de estas puede generar que los hilos queden esperando mucho tiempo.

Bibliografía

Echaiz, J. (n.d.). Sistemas Operativos y Distribuidos. Departamento de Ciencias e Ingeniería de la Computación - Universidad Nacional del Sur. Recuperado de http://cs.uns.edu.ar/~jechaiz

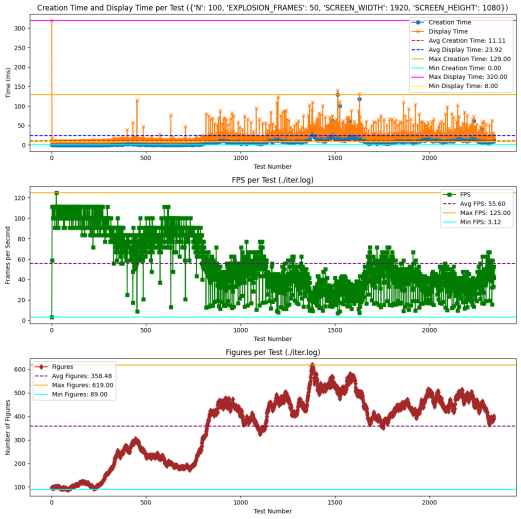
Anexos



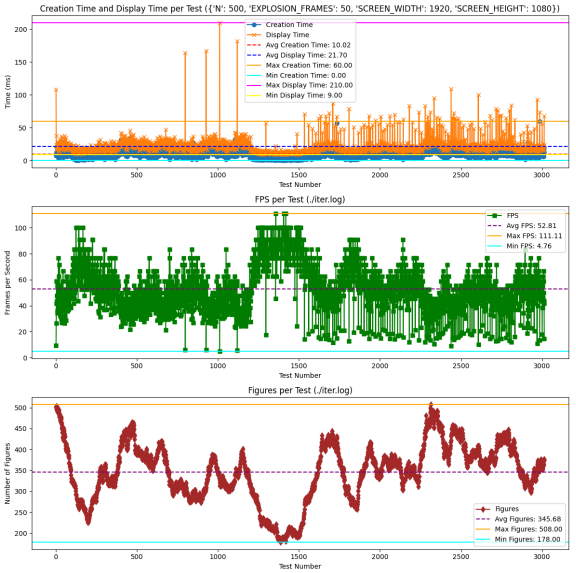
Anexo 1 - Diagrama de flujo del programa secuencial

Nombre	*Entradas*	*Salidas*	*Descripción*
isOverlapping	• Figura* a, Figura* b • Tipo: figuras • Se usan para verificar si se están superponiendo entre sí.	0 o 1 si se superpone o no.	Verificar si dos figuras se superponen
initFiguras	• SDL_Renderer *renderer • Tipo renderer • se le pasa de parámetro a la figura que se creará.	-	Inicialización de figuras
spawnFigura	• SDL_Renderer *renderer • Tipo renderer • se le pasa de parámetro a la figura que se creará.	-	Añadir una figura a la lista de figuras
spawnExplosion	• SDL_Renderer *renderer, int x, int y. • Tipo renderer e int. • Se le pasa de parámetro a la figura de la explosión que se creará.	-	*Crea una explosión en la posición (x, y), que previamente era la posición de una figura*
cleanExplosions	-	-	Limpia la explosión.
initializeWindow	• *char* title, int width, int height* • *Se usa para determinar el nombre, ancho y alto.*	*SDL_Windowwindow*	*Función para inicializar SDL y crear una ventana*
cleanup	• *SDL_Window* window* • *Cierra y destruye esa ventana*	-	*Función para cerrar SDL y destruir la ventana*
resolveCollision	• *Figura* a, Figura* b*	-	*Resuelve colisiones entre dos figuras*
moveFigura	• *Figura* figura, SDL_Renderer* renderer* • *La figura cambia de dirección.*	-	*Función para mover una figura*
loadTexture	• *char* filename, SDL_Renderer* renderer* • *El path para poner la imagen*	*texture* *La textura con la imagen colocada*	*Función para cargar la textura.*
createFigura	• *int x, int y, int width, int height, int speedX, int speedY, const char* image, SDL_Renderer* renderer, SDL_Color color* • *Parametros par ala creación de una nueva figura*	*Figura figura* *La nueva figura*	*Función que crea una figura.*
drawFigura	• *SDL_Renderer* renderer, Figura* figura* • *Se dibuja esa figura.*	-	*Función para dibujar una figura en la superficie dada*
checkCollision	• *Figura* a, Figura* b* • *Figuras a las que se chequean si colisionaron*	*1 o 0 si colisiona o no*	*Función para detectar colisión entre dos figuras y el eje de colisión*

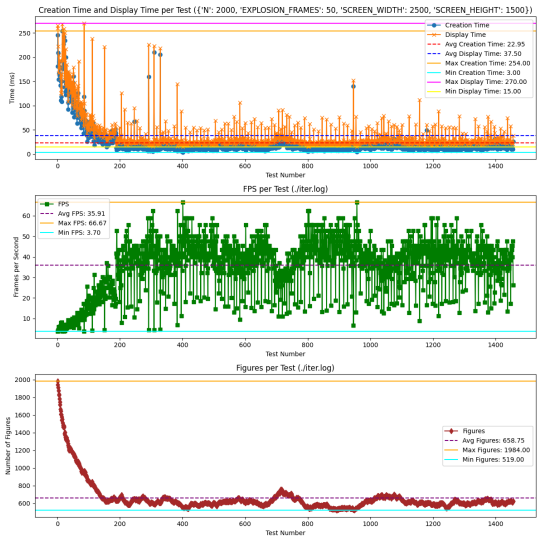
Anexo 2 – Catálogo de funciones



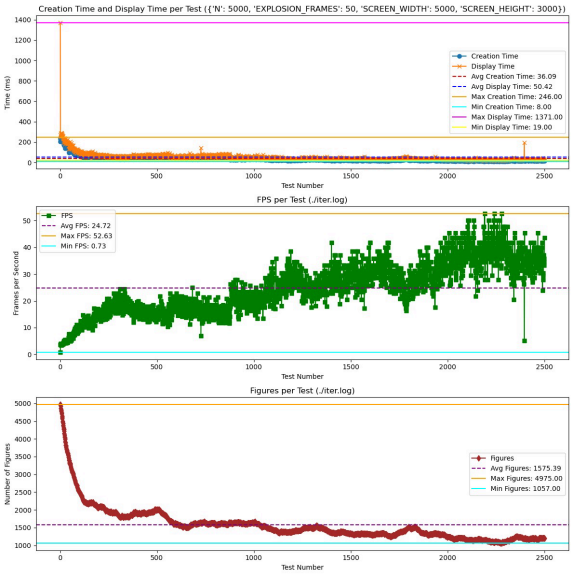
Cantidad inicial de figuras: 100
Frames de explosión: 50
Ancho: 1920
Alto:1080



Cantidad inicial de figuras: 500
Frames de explosión: 50
Ancho: 1920
Alto:1080

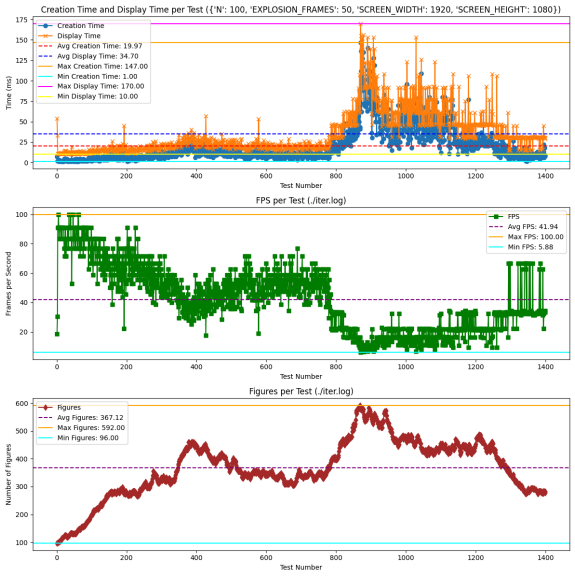


Cantidad inicial de figuras: 2000
Frames de explosión: 50
Ancho: 2500
Alto:1500

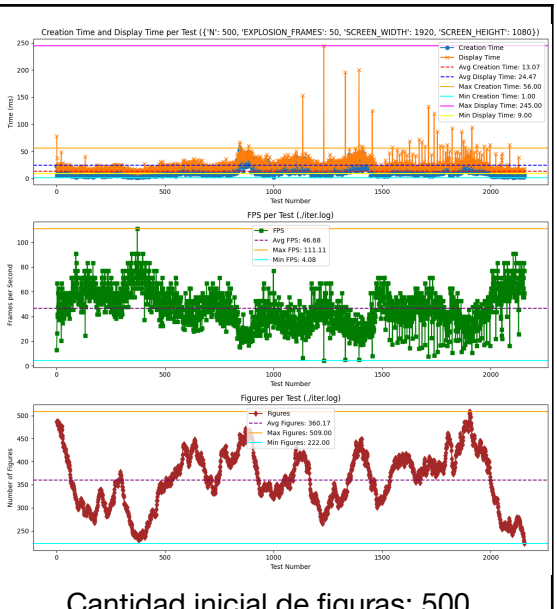


Cantidad inicial de figuras: 5000
Frames de explosión: 50
Ancho: 5000
Alto: 3000

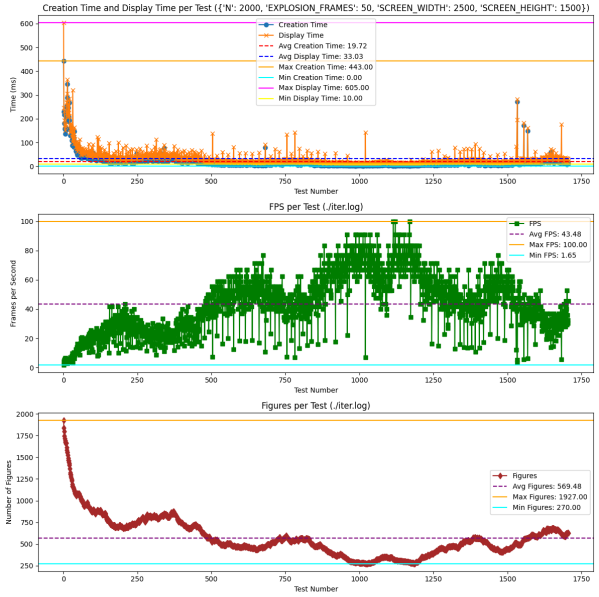
Anexo 3.1 – Bitácora de pruebas: Secuencial



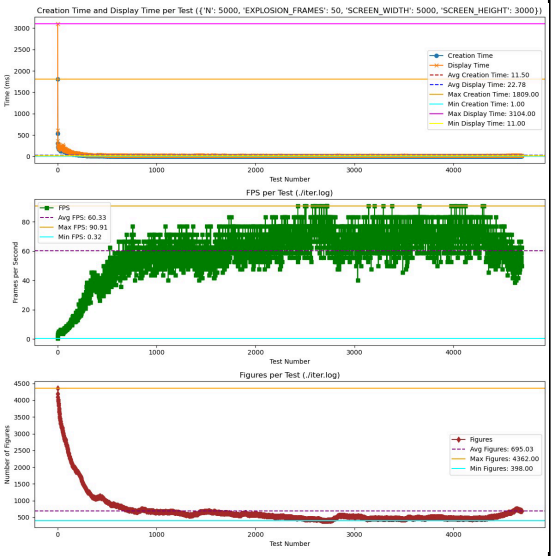
Cantidad inicial de figuras: 100
Frames de explosión: 50
Ancho: 1920
Alto:1080



Cantidad inicial de figuras: 500
Frames de explosión: 50
Ancho: 1920
Alto:1080



Cantidad inicial de figuras: 2000
Frames de explosión: 50
Ancho: 2500
Alto:1500



Cantidad inicial de figuras: 5000
Frames de explosión: 50
Ancho: 5000
Alto: 3000

Anexo 3.2 - Bitácora de pruebas: Paralelizado

Ubicación	Método	Propósito
initFiguras	*#pragma omp parallel y #pragma omp for*	Inicialización de figuras para asegurar que no estén superpuestas.
spawnFigura	*#pragma omp parallel for*	Generación de nuevas figuras iterando sobre combinaciones de colores para evitar condiciones de carrera durante la modificación de atributos de color.
spawnExplosion	*#pragma omp parallel for*	Generación de explosiones iterando sobre combinaciones de colores para asignar atributos de color de forma segura.
cleanExplosions	*#pragma omp parallel, #pragma omp for, #pragma omp atomic, #pragma omp critical, #pragma omp single*	Limpieza de explosiones. Se utiliza para decrementar los cuadros restantes de cada explosión y eliminar explosiones con cuadros restantes de cero. Utiliza omp critical para asegurar operaciones de memoria seguras cuando se elimina una explosión.
updateFiguras	*#pragma omp parallel for*	Actualización de la posición de las figuras en la pantalla. Utiliza omp critical para manejar la creación de nuevas figuras de forma segura cuando las figuras existentes tocan los bordes de la pantalla.
	omp_nested(1)	Permitir el uso de paralelización anidada.

Anexo 4. Tabla de paralelización