

**UNIVERSIDAD DEL VALLE DE GUATEMALA**

CC-3069 Computación Paralela y Distribuida

Sección 20

Ing. Luis Garcia



# Programación paralela con OpenMP

## Proyecto #1

Fernando José Garavito Ovando, 18071

Alexa Bravo, 18831

Daniela Villamar, 19086

Fernando Hengstenberg, 17699

**GUATEMALA, 10 de abril de 2023**

## **Índice**

Introducción.....	2
Antecedentes.....	3
Citas textuales.....	4
Conclusiones.....	4
Recomendaciones.....	5
Citas bibliográficas.....	5
Anexos.....	6
Anexo I: Diagrama de Flujo .....	6
Anexo II: Catálogo de Funciones .....	7
Anexo III: Bitácora de Pruebas .....	12

## **Introducción**

Open Multi-Processing mejor conocido como OpenMP es una interfaz de programación de aplicaciones que se utiliza para la programación de multiprocesos en memoria compartida en múltiples plataformas. Debido a las características con las que cuenta se pueden desarrollar aplicaciones paralelas, entre ellas un screensaver o protector de pantallas, los cuales se pueden realizar en distintos lenguajes de programación. Un protector de pantalla aparece cuando la computadora está inactiva y principalmente para seguridad, mostrando información del sistema, entretenimiento, entre otras.

En el proyecto detallado a continuación, se describe la implementación y funcionamiento de un screensaver utilizando la herramienta OpenMP en el lenguaje de programación C o C ++. El objetivo final de este proyecto es comparar el rendimiento de un screensaver desarrollado de manera secuencial y desarrollado de forma paralela. Cabe destacar que el número de cuadros por segundo (FPS) a los que corre el programa es la variable que demuestra una comparación entre los screensavers y saber cual es la mejor forma de implementar un screensaver con respecto a su rendimiento.

## **Antecedentes**

Los screensavers se crearon en 1988, por los programadores Bill Stewart e Ian MacDonald, qué crearon Magic Screensaver para Windows 2, el cual consiste en una línea qué rebota de esquina a esquina creando una variedad de intrincadas formas geométricas que tomaron por sorpresa al mundo de la computación.

En 1989, Apple implementó en sus computadoras after dark, qué es su versión de screensavers.

A principios de los 90, se lanzó Windows 3.0 acompañado de varios de sus propios protectores de pantalla, entre ellos, el clásico de los logos de Windows flotando en el espacio. A mediados de los 90, se lanzaron varios screensavers inspirados en productos de la cultura popular como Star Trek y Los Simpsons.

Windows 95 fue el más creativo con sus diseños, mientras qué con Windows 98 y XP, el arte de los screensavers decayó poco a poco, y en 2001, Windows XP agregó algunos screensavers con figuras en 3D.

La implementación puede realizarse en varios lenguajes de programación y entre estos incluir, OpenMP qué es una herramienta que facilita la transformación de un programa secuencial en paralelo, también permite la abstracción y programación paralela en alto nivel. Debido a que está pensado como una solución iterativa, se pueden realizar cambios graduales en un programa secuencial para aprovechar múltiples recursos mediante ejecución paralela

Hoy en día ya no se utilizan de la misma manera los screensavers ya que muchos usuarios prefieren apagar la computadora o cerrarla por completo para tener un ahorro en batería.

## Citas textuales

Microsoft describe que OpenMP usa el modelo de bifurcación o combinación de ejecución en paralelo. Está pensado para admitir programas que se ejecuten correctamente como programas paralelos y también será para los programas que se ejecuten correctamente como programas secuenciales.

En el libro “Parallel Programming in OpenMP”, se describe a OpenMP como un modelo de programación paralela para multiprocesadores de memoria compartida y memoria compartida distribuida. Iniciado por SGI y desarrollado en colaboración con otros fabricantes de ordenadores paralelos, OpenMP se está convirtiendo rápidamente en el estándar de facto para parallelizar aplicaciones.

## Conclusiones

1. El tiempo de ejecución disminuye considerablemente utilizando OpenMP.
2. La librería OpenMP se puede implementar en diferentes programas y es bien interesante todo lo qué se puede realizar con ella, ya qué es bastante completa para la programación paralela.
3. Los screensavers marcaron parte de la historia de los sistemas operativos, pero en especial de Windows.
4. La estructura de tipo arrays en C++ es la mejor para conservar los cálculos de los puntos, colores, dibujar los objetos, etc. Es bueno ya que se tiene un acceso fácil a los respectivos valores en tiempo real y constante.
5. Para hacer una paralización de los cálculos donde están las posiciones de los puntos para dibujar los objetos lo mejor es hacerlo en base a una función matemática, la forma en la que es graficado mejora.

## **Recomendaciones**

- Se recomienda tener una buena cantidad de memoria en su sistema al momento de ejecutar los programas de screensaver, ya que estos al momento de usarlos en la máquina pueden no funcionar si no se tiene el suficiente almacenamiento.
- Se recomienda descargar la librería que se vaya a utilizar y configurarlo tanto en el editor de texto que se use como en el propio sistema.
- Se recomienda implementar funciones matemáticas más complejas para poder evaluar la eficiencia de los métodos paralelos.
- Se recomienda investigar un poco más de la librería de OpenGL así como otras que permitan el renderizado en paralelo para evitar futuros inconvenientes con la creación de hilos y guardar valores fijos de renderizado.

## **Citas bibliográficas**

Chapman, B (2008). Using OpenMP. Extraído de:

<https://pdfs.semanticscholar.org/932d/5abe3d49f3c49d77c6e60ddbd0e3dfcae8dd.pdf>

Microsoft (2023). OpenMP. Extraído de:

<https://learn.microsoft.com/es-es/cpp/parallel/openmp/1-introduction?view=msvc-170>

Parallel Programming in OpenMP.

[https://books.google.com.gt/books?hl=en&lr=&id=vuAY5C5C1W0C&oi=fnd&pg=P2&dq=openmp&ots=6tfrV9Hs2w&sig=INWslxrUHJ4JFjaO\\_KP7JXkS7xM&redir\\_esc=y#v=onepage&q=openmp&f=false](https://books.google.com.gt/books?hl=en&lr=&id=vuAY5C5C1W0C&oi=fnd&pg=P2&dq=openmp&ots=6tfrV9Hs2w&sig=INWslxrUHJ4JFjaO_KP7JXkS7xM&redir_esc=y#v=onepage&q=openmp&f=false)

## **Anexos**

### **Anexo I: Diagrama de Flujo**

## Anexo II: Catálogo de Funciones

### SCREENSAVER SECUENCIAL

**<iostream>** es una biblioteca orientada a objetos que proporciona funcionalidad de entrada y salida mediante flujos.

**<cstdlib>** define una colección de funciones y macros para facilitar código C++ estandarizado, eficiente y de alto rendimiento en equipos y plataformas.

**<ctime>** convierte el tiempo dado en una hora local del calendario y luego en una representación de caracteres.

**<cmath>** declara un conjunto de funciones para realizar operaciones matemáticas.

**<GL/glut.h>** es el kit de herramientas de utilidades de OpenGL. Implementa una interfaz de programación de aplicaciones (API) de ventana simple para OpenGL.

**Ancho:** Ancho del screensaver.

**Altura:** Alto del screensaver.

**N:** Cantidad de figuras.

**PX[N]:** Posición de las figuras en el eje x.

**PY[N]:** Posición de las figuras en el eje y.

**VX[N]:** Velocidad de las figuras en el eje x.

**VY[N]:** Velocidad de las figuras en el eje y.

**Contador:** Recuento de los fps.

**fps:** Cantidad de los fps.

**Tiempo:** Tiempo de los fps.

**Antes:** Tiempo pasado de los fps.

**void drawScene():** Escena general del screensaver.

**Tiempo = glutGet(GLUT\_ELAPSED\_TIME):** Tiempo en milisegundos.

**Actual:** Tiempo actual se resta el tiempo presente del pasado de los fps.

**glClear(GL\_COLOR\_BUFFER\_BIT):** Indica los búferes habilitados actualmente para la escritura en color.

**Estructura del cuerpo del pez.**

**glPushMatrix()**

**glTranslated(PX[i], PY[i], 0.0):** Movimiento Inicial.

**glColor3f(0.0,1.0,0.0):** Color del objeto.

**glBegin(GL\_POLYGON):** Se crea un polígono.

**glVertex2f(0.7+a,0.05):** Coordenadas en 'x' y 'y'.

**glVertex2f(0.75+a,0.1):** Coordenadas en 'x' y 'y'.

**glVertex2f(0.85+a,0.05):** Coordenadas en 'x' y 'y'.

```
glVertex2f(0.75+a,0.0); Coordenadas en 'x' y 'y'.
glEnd()
```

#### Estructura de las aletas del pez.

```
glBegin(GL_TRIANGLES); Se crea un triángulo.
	glColor3f(0.0,0.0,1.0); Color del objeto.
	glVertex2f(0.83+a,0.05); Coordenadas en 'x' y 'y'.
	glColor3f(0.0,1,0); Color del objeto.
	glVertex2f(0.9+a,0.09); Coordenadas en 'x' y 'y'.
	glVertex2f(0.9+a,0.01); Coordenadas en 'x' y 'y'.
	glEnd()
```

#### Estructura de la cola del pez.

```
glBegin(GL_TRIANGLES); Se crea un triángulo
	glColor3f(1.0,1,0,0); Color del objeto.
	glVertex2f(0.75+a,0.007); Coordenadas en 'x' y 'y'.
	glColor3f(1.0,0,0,0); Color del objeto.
	glVertex2f(0.795+a,-0.035); Coordenadas en 'x' y 'y'.
	glVertex2f(0.77+a,0.02); Coordenadas en 'x' y 'y'.
	glEnd()
```

#### Estructura del ojo del pez.

```
	glColor3f(0.0,0.0,0.0); Color del objeto.
	glPointSize(4.0); Tamaño del punto.
	glBegin(GL_POINTS); Se crea un punto.
	glVertex2f(0.73+a,0.065); Coordenadas en 'x' y 'y'.
	glEnd()
```

**std::string fpsStr = "FPS: " + std::to\_string(fps);**: Se muestran los fps.

**PX[i] += VX[i];**: Modifica la posición de la figura en el eje 'x'.
**PY[i] += VY[i];**: Modifica la posición de la figura en el eje 'y'.

**void init();**: Se inicializa el objeto que se crea, en este caso, el screensaver.

**PX[i] = ((double)rand() / RAND\_MAX) \* 3.0 - 2.0;**: Valores random en la posición del eje 'x'.

**PY[i] = ((double)rand() / RAND\_MAX) \* 3.0 - 2.0;**: Valores random en la posición del eje 'y'.

**VX[i] = ((double)rand() / RAND\_MAX) \* 0.02 - 0.01:** Valores random en la velocidad del eje 'x'.

**VY[i] = ((double)rand() / RAND\_MAX) \* 0.02 - 0.01:** Valores random en la velocidad del eje 'y'.

**glClearColor(0, 0, 1, 0.0):** Color del fondo de la escena.

### Parámetros de visualización

**glMatrixMode(GL\_PROJECTION):** Modo proyección.

**glLoadIdentity():** Asegura que cada vez que ingresemos al modo de proyección, la matriz se restablecerá a matriz de identidad, para que los nuevos parámetros de visualización no se combinen con los anteriores .

**glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0):** proporciona un área de dibujo OpenGL con una proyección ortográfica.

**glMatrixMode(GL\_MODELVIEW):** Modo vista.

**main(int argc, char \*\*argv):** Main

**glutInitWindowSize(Ancho, Altura):** Tamaño del screensaver.

**glutInitWindowPosition(100, 100):** Posición del screensaver.

**glutCreateWindow("Screensaver"):** Nombre asignado al programa.

**glutDisplayFunc(drawScene):** Animación de la escena.

**double SGS = double(FF - GG) / CLOCKS\_PER\_SEC:** Tiempo total

**glutMainLoop():** Repetimos el proceso

## SCREENSAVER SECUENCIAL

<iostream> es una biblioteca orientada a objetos que proporciona funcionalidad de entrada y salida mediante flujos.

<cstdlib> define una colección de funciones y macros para facilitar código C + + estandarizado, eficiente y de alto rendimiento en equipos y plataformas.

<ctime> convierte el tiempo dado en una hora local del calendario y luego en una representación de caracteres.

<cmath> declara un conjunto de funciones para realizar operaciones matemáticas.

<GL/glut.h> es el kit de herramientas de utilidades de OpenGL. Implementa una interfaz de programación de aplicaciones (API) de ventana simple para OpenGL.

<open.h>: proporcionará la funcionalidad openmp.

**Ancho:** Ancho del screensaver.

**Altura:** Alto del screensaver.

**N:** Cantidad de figuras.

**PX[N]:** Posición de las figuras en el eje x.

**PY[N]:** Posición de las figuras en el eje y.

**VX[N]:** Velocidad de las figuras en el eje x.

**VY[N]:** Velocidad de las figuras en el eje y.

**Contador:** Recuento de los fps.

**fps:** Cantidad de los fps.

**Tiempo:** Tiempo de los fps.

**Antes:** Tiempo pasado de los fps.

**void drawScene():** Escena general del screensaver.

**Tiempo = glutGet(GLUT\_ELAPSED\_TIME):** Tiempo en milisegundos.

**Actual:** Tiempo actual se resta el tiempo presente del pasado de los fps.

**glClear(GL\_COLOR\_BUFFER\_BIT):** Indica los búferes habilitados actualmente para la escritura en color.

**Estructura del cuerpo del pez.**

**glPushMatrix()**

**glTranslated(PX[i], PY[i], 0.0):** Movimiento Inicial.

**glColor3f(0.0,1.0,0.0):** Color del objeto.

**glBegin(GL\_POLYGON):** Se crea un polígono.

**glVertex2f(0.7+a,0.05):** Coordenadas en 'x' y 'y'.

**glVertex2f(0.75+a,0.1):** Coordenadas en 'x' y 'y'.

**glVertex2f(0.85+a,0.05):** Coordenadas en 'x' y 'y'.

**glVertex2f(0.75+a,0.0):** Coordenadas en 'x' y 'y'.

**glEnd()**

**Estructura de las aletas del pez.**

**glBegin(GL\_TRIANGLES):** Se crea un triángulo.

**glColor3f(0.0,0.0,1.0):** Color del objeto.

**glVertex2f(0.83+a,0.05):** Coordenadas en 'x' y 'y'.

**glColor3f(0.0,1,0):** Color del objeto.

**glVertex2f(0.9+a,0.09):** Coordenadas en 'x' y 'y'.

**glVertex2f(0.9+a,0.01):** Coordenadas en 'x' y 'y'.

**glEnd()**

**Estructura de la cola del pez.**

**glBegin(GL\_TRIANGLES):** Se crea un triángulo

**glColor3f(1.0,1.0,0.0):** Color del objeto.

**glVertex2f(0.75+a,0.007):** Coordenadas en 'x' y 'y'.

**glColor3f(1.0,0.0,0.0):** Color del objeto.

**glVertex2f(0.795+a,-0.035):** Coordenadas en 'x' y 'y'.

**glVertex2f(0.77+a,0.02):** Coordenadas en 'x' y 'y'.

**glEnd()**

### Estructura del ojo del pez.

**glColor3f(0.0,0.0,0.0):** Color del objeto.

**glPointSize(4.0):** Tamaño del punto.

**glBegin(GL\_POINTS):** Se crea un punto.

**glVertex2f(0.73+a,0.065):** Coordenadas en 'x' y 'y'.

**glEnd()**

## pragma omp parallel: Paralelización con OpenMP.

### pragma omp atomic

**PX[i] += VX[i]:** Modificación de la posición de la figura en el eje 'x'.

### pragma omp atomic

**PY[i] += VY[i]:** Modificación de la posición de la figura en el eje 'y'.

**void init():** Se inicializa el objeto que se crea, en este caso, el screensaver.

**PX[i] = ((double)rand() / RAND\_MAX) \* 3.0 - 2.0:** Valores random en la posición del eje 'x'.

**PY[i] = ((double)rand() / RAND\_MAX) \* 3.0 - 2.0:** Valores random en la posición del eje 'y'.

**VX[i] = ((double)rand() / RAND\_MAX) \* 0.02 - 0.01:** Valores random en la velocidad del eje 'x'.

**VY[i] = ((double)rand() / RAND\_MAX) \* 0.02 - 0.01:** Valores random en la velocidad del eje 'y'.

**glClearColor(0, 0, 1, 0.0):** Color del fondo de la escena.

## Parámetros de visualización

**glMatrixMode(GL\_PROJECTION):** Modo proyección.

**glLoadIdentity():** Asegura que cada vez que ingresemos al modo de proyección, la matriz se restablecerá a matriz de identidad, para que los nuevos parámetros de visualización no se combinen con los anteriores .

**glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0):** proporciona un área de dibujo OpenGL con una proyección ortográfica.

**glMatrixMode(GL\_MODELVIEW):** Modo vista.

**main(int argc, char \*\*argv):** Main  
**glutInitWindowSize(Ancho, Altura):** Tamaño del screensaver.  
**glutInitWindowPosition(100, 100):** Posición del screensaver.  
**glutCreateWindow("Screensaver"):** Nombre asignado al programa.  
**glutDisplayFunc(drawScene):** Animación de la escena.

**std::cerr << "Usage: " << argv[0] << " <number\_of\_threads>" << std::endl;** Threads.  
**std::cerr << "Número de threads" << std::endl;** Número de threads.  
**int numThreads = std::stoi(argv[1]);** Cantidad de threads.  
**omp\_set\_num\_threads(numThreads);** Paralelismos los threads.

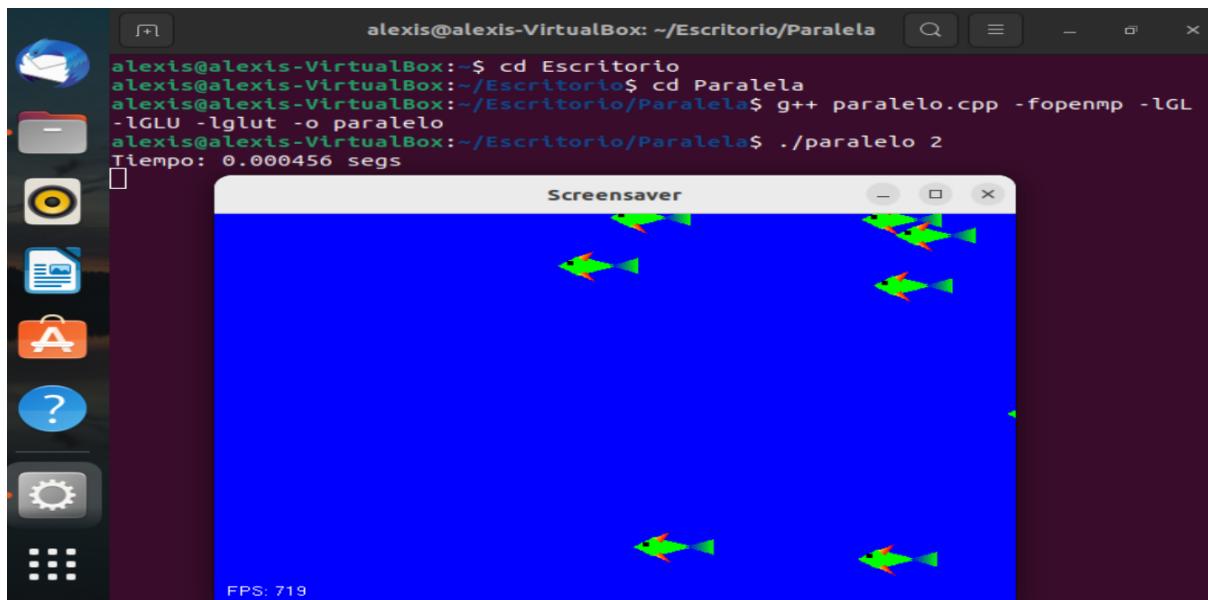
**double SGS = double(FF - GG) / CLOCKS\_PER\_SEC;** Tiempo total

**glutMainLoop();** Repetimos el proceso

### Anexo III: Bitácora de Pruebas

Número de Prueba	Tiempo de Ejecución	
	Paralelo	Secuencial
1	0.000456	$1.8e^{-0.5}$
2	0.59406	$1.6e^{-0.5}$
3	0.000847	$2.1e^{-0.5}$
4	0.001418	$1.7e^{-0.5}$
5	0.000875	$1.9e^{-0.5}$
6	0.00084	$1.2e^{-0.5}$
7	0.00104	$1.8e^{-0.5}$
8	0.001627	$2e^{-0.5}$
9	0.001047	$1.6e^{-0.5}$
10	0.001538	$2e^{-0.5}$

**Tabla I:** Mediciones de tiempo de ejecución del screensaver de manera paralela y secuencial.



**Imagen I:** Captura del programa en manera paralela.



**Imagen II:** Captura del programa en manera paralela.

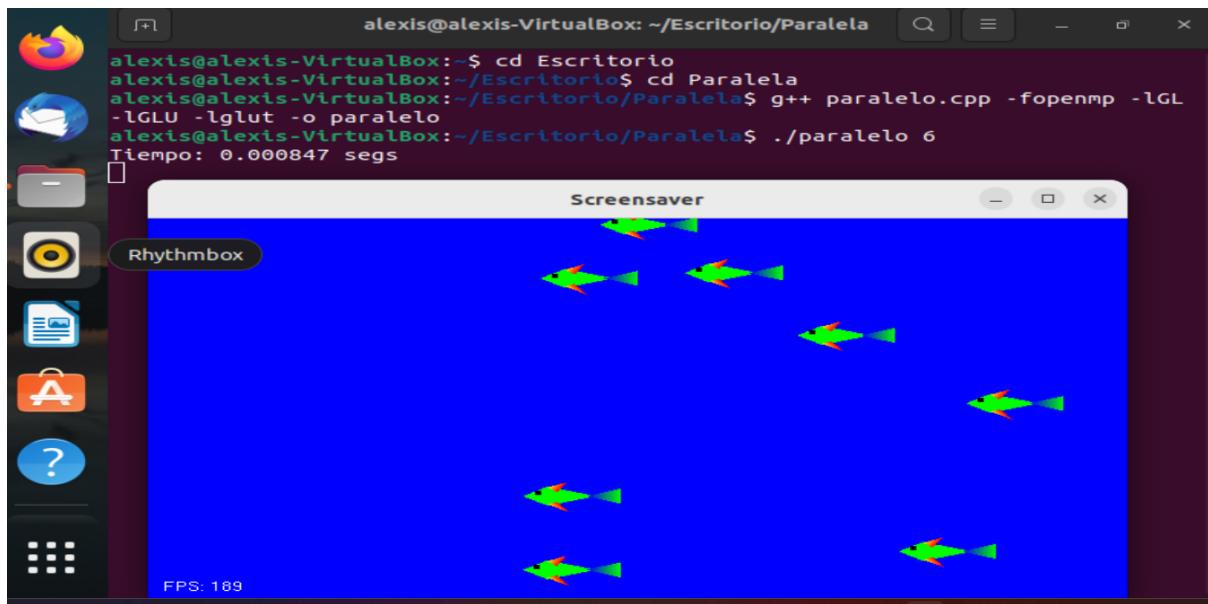


Imagen III: Captura del programa en manera paralela.

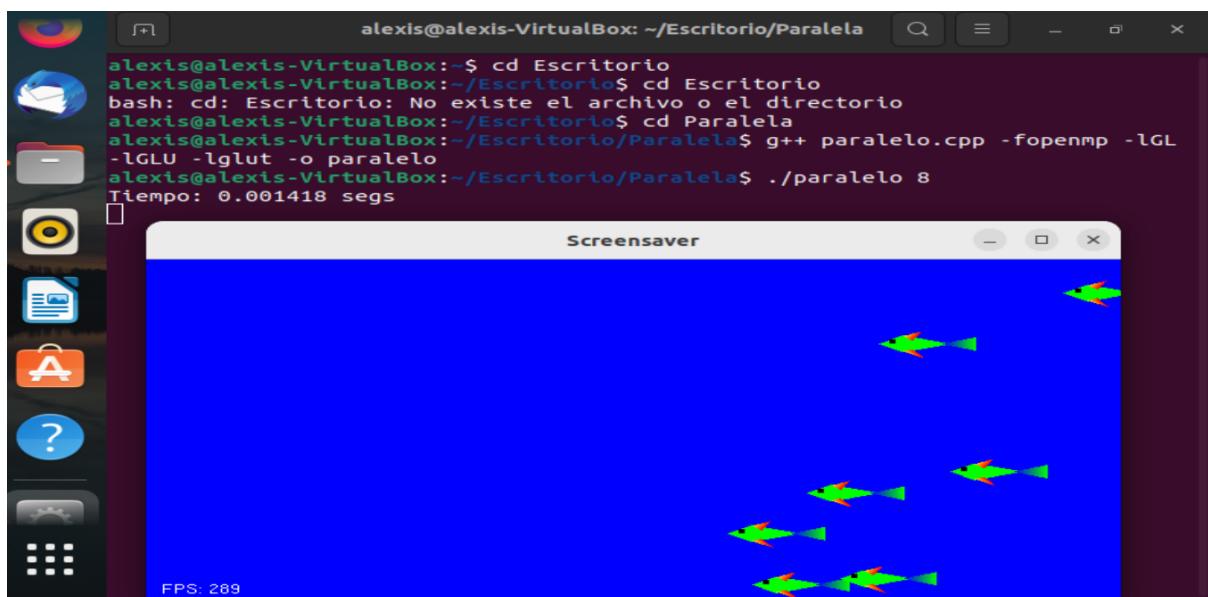


Imagen IV: Captura del programa en manera paralela.

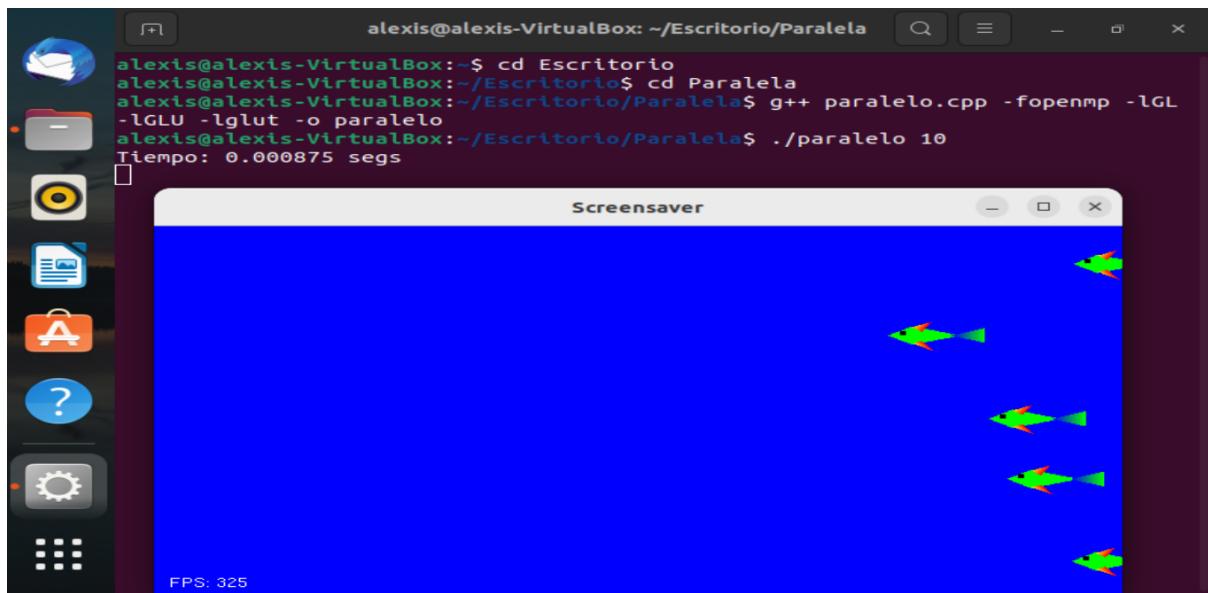


Imagen V: Captura del programa en manera paralela.

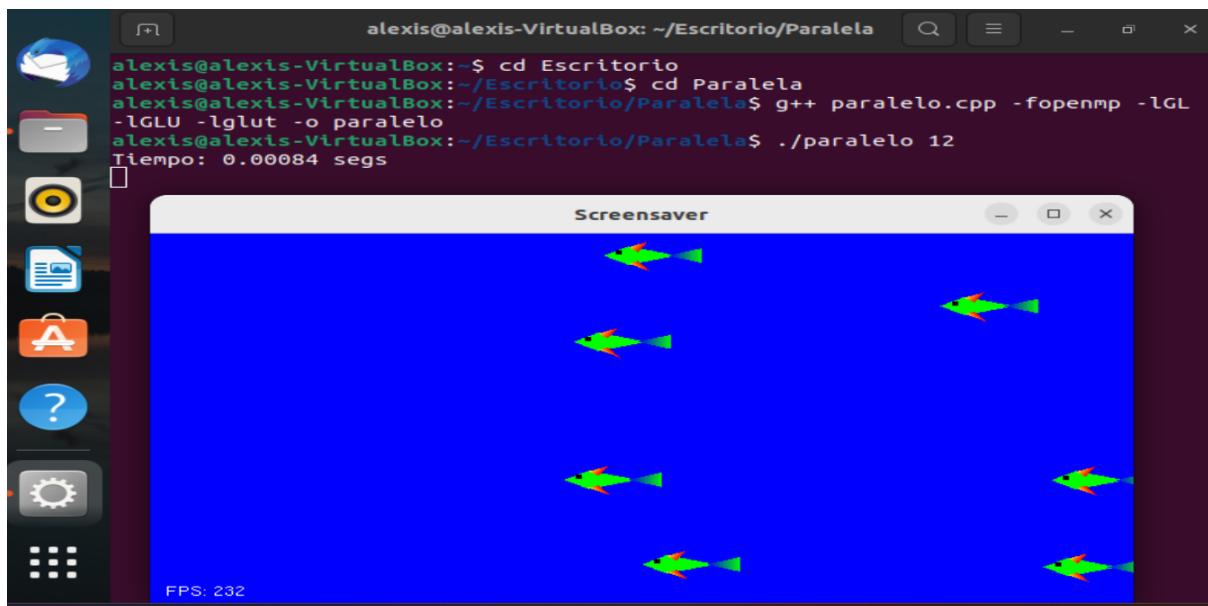


Imagen VI: Captura del programa en manera paralela.

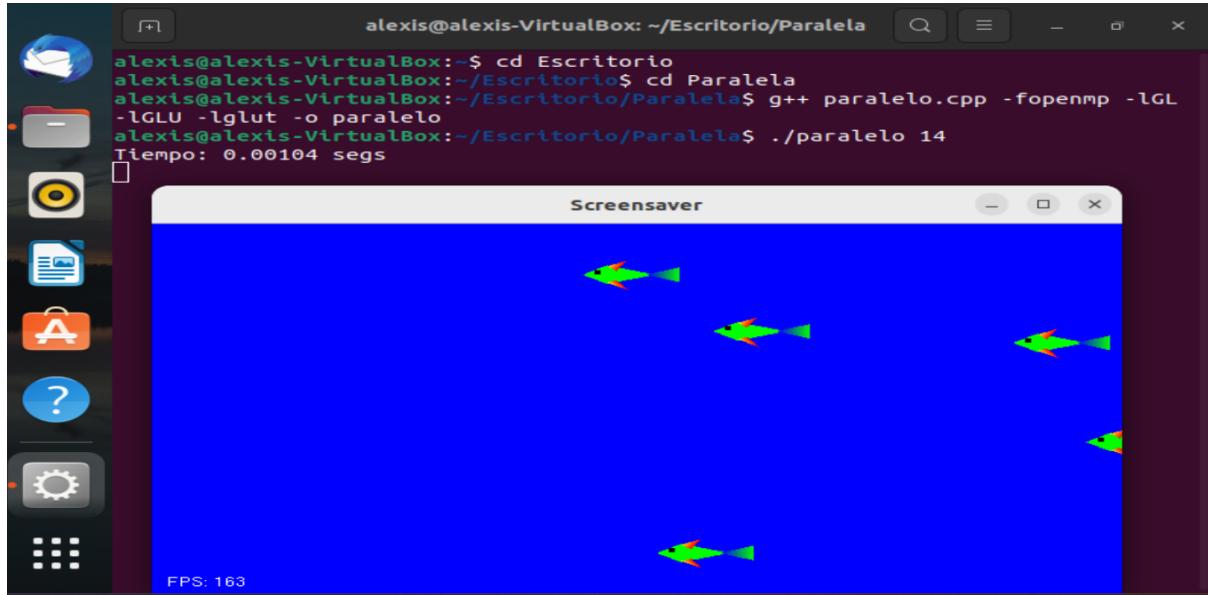


Imagen VII: Captura del programa en manera paralela.



Imagen VIII: Captura del programa en manera paralela.



**Imagen IX:** Captura del programa en manera paralela.



**Imagen X:** Captura del programa en manera paralela.



Imagen XI: Captura del programa en manera secuencial.



Imagen XII: Captura del programa en manera secuencial.

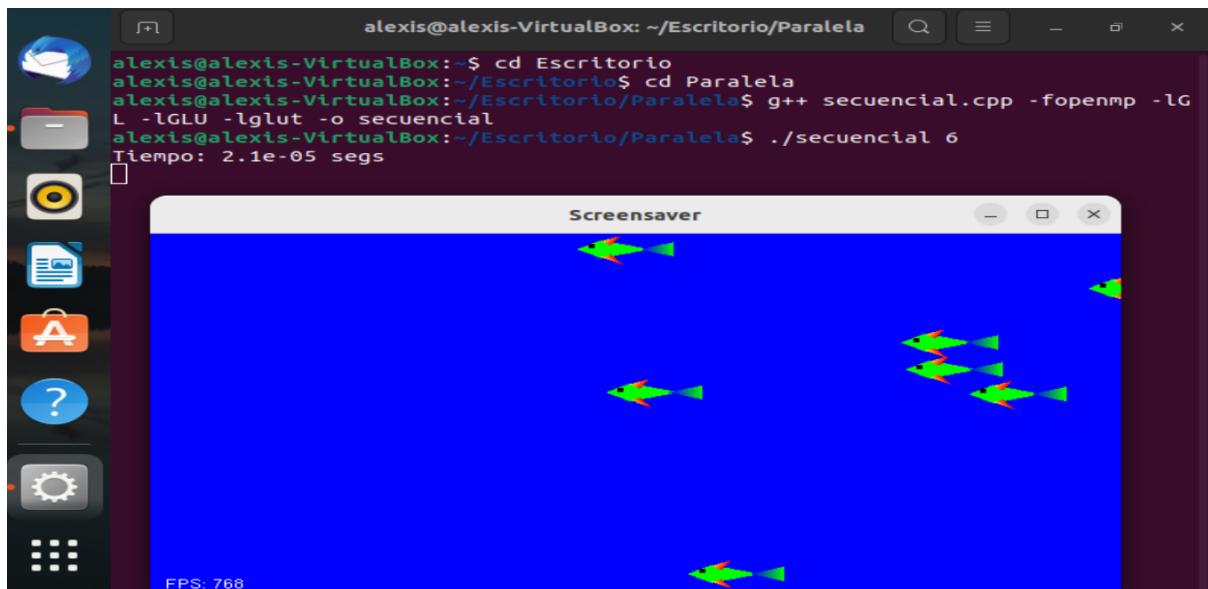


Imagen XII: Captura del programa en manera secuencial.



Imagen XIV: Captura del programa en manera secuencial.



Imagen XV: Captura del programa en manera secuencial.



Imagen XVI: Captura del programa en manera secuencial.



Imagen XVII: Captura del programa en manera secuencial.



Imagen XVIII: Captura del programa en manera secuencial.



Imagen XIX: Captura del programa en manera secuencial.



Imagen XX: Captura del programa en manera secuencial.

## Link del código:

<https://github.com/Jos260400/Proyecto-Paralela>