

# Impact of parallelization technologies on an optical flow-based motion interpolation algorithm

Juan Sebastián Rodríguez Castellanos  
Departamento de Sistemas e Industrial  
Universidad Nacional de Colombia  
Bogotá - Colombia  
[juarodriguezc@unal.edu.co](mailto:juarodriguezc@unal.edu.co)  
GitHub account

**Abstract**—One of the applications of video processing is the Motion interpolation or the motion-compensated frame interpolation (MCFI), in these algorithms a pair of frames are used to generate an intermediate frame. This paper presents the implementation of a frame interpolation algorithm based on optical flow using different parallelization technologies (CPU, GPU, MPI). The algorithm used is presented in the document. Since the calculation of the Optical Flow is a process that requires a high computational capacity but can be executed in parallel, the impact of these technologies on the execution time will be analyzed. Different tests were performed, resulting in a speedup greater than 3 using CPU and above 34 using GPU with the implementation in CUDA. However, in all cases the execution times were considerably high, allowing only the CUDA implementation to interpolate an average of 3 frames per second for high video resolutions.

**Keywords:** Motion interpolation, Optical flow, CPU, GPU, Speedup.

## I. INTRODUCTION

Motion interpolation is a form of video processing with numerous applications, this type of algorithm is widely used by different technology companies to provide the effect of “smoothing” on different audiovisual content. The goal is to create an intermediate frame or number of frames between existing frames [1]. There are two main types of motion interpolation algorithms, in the first type all frames to be interpolated are known, in the second type only the current and previous frames are known.

There are many uses for this type of algorithm. In most current TVs there is software that produces the effect of “smoothing” on video, to achieve this effect the frame rate is increased to 60 or more Frames, depending on the frequency of the screen. Another of the main uses of video interpolation is in Virtual Reality (VR), being used by companies such as Oculus, Microsoft and Steam for their VR headsets [1].

In this paper an optical flow-based motion interpolation algorithm is presented which can interpolate frames using only the current and previous frames. To generate the intermediate frame, the Optical Flow is calculated, then linear interpolation is applied using the information obtained from the Optical Flow and finally a Gaussian Blur is applied to the areas with the highest motion.

The calculation of the Optical Flow is the most important task for the algorithm developed, the objective of the optical flow is to predict the pixel-level motion between video frames [2]. Classical approaches have been superseded by learning-based methods [2], however, for the development of the algorithm, a brute force approach is used to calculate the motion vectors.

## II. RELATED WORK

### 1) Optical Flow:

The optical flow is the pattern of apparent motion of image objects between two consecutive frames caused by the movement of an object or camera [3]. It is a 2D vector where each vector is a displacement vector showing the movement of points from first frame to second [3]. To calculate this vector it is necessary to use the intensity or luminance of each pixel [4]. Consider a pixel  $I(x, y, t)$  in the first frame, it moves by distance  $(dx, dy)$  in the next frame after  $dt$  time, so:

$$I(x, y, t) = I(x + dx, y + dy, t + dt) \quad (1)$$

An example of the optical flow calculation is shown in Figure 1.

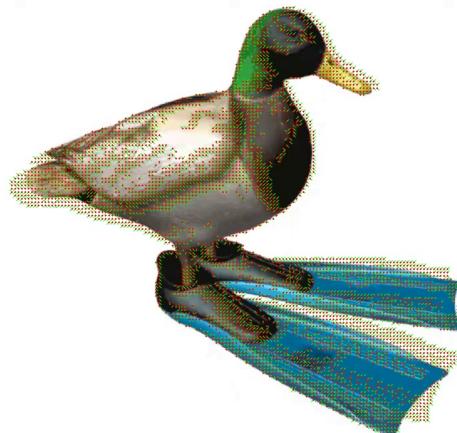


Fig. 1: Example of the Optical Flow

### 2) Luminance:

To calculate the Optical Flow it is necessary to use the intensity of each pixel. Intensity or luminance is a measure to describe the perceived brightness of a color [5], this value can be calculated using the following formula:

$$L = 0.2987R + 0.5870G + 0.1140B \quad (2)$$

Values for R, G and B must be between 0 and 1.

### 3) Gaussian Blur:

### 4) Linear Interpolation:

## III. ALGORITHM DESIGNED

### 1) Motion interpolation:

### 2) Parallel algorithm:

## IV. EXPERIMENTS

The main criteria to evaluate the algorithm designed

## V. RESULTS

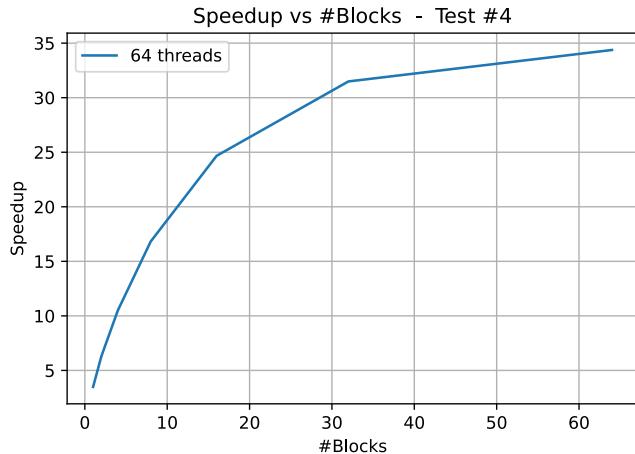


Fig. 2: Speedup CUDA

Los experimentos realizados se hicieron con imágenes de resolución 720p, 1080p y 4k, por otro lado se varió el número de hilos, donde se calculó el tiempo con cada cantidad de hilos y su speed up respectivo.

Las imágenes utilizadas para cada resolución son las siguientes:

### 1) Imagen 720p:



Fig. 3: Imagen de manzanas 1280x720 px

### 2) Imagen 1080p:



Fig. 4: Imagen de astronauta 1920x1080 px

### 3) Imagen 4K:



Fig. 5: Imagen de fondo 3840x2160 px

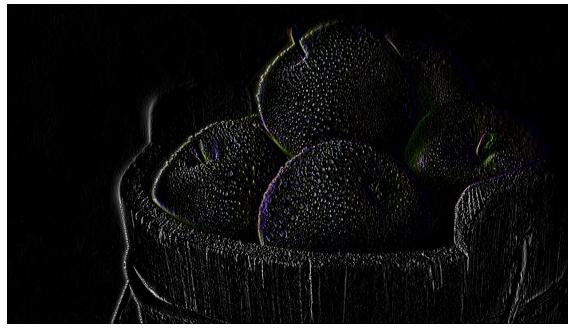
## VI. RESULTADOS

Al aplicar los **filtros** en las imágenes obtuvimos los siguientes resultados:

### A. Imágenes obtenidas

#### 1) Imagen 720p:

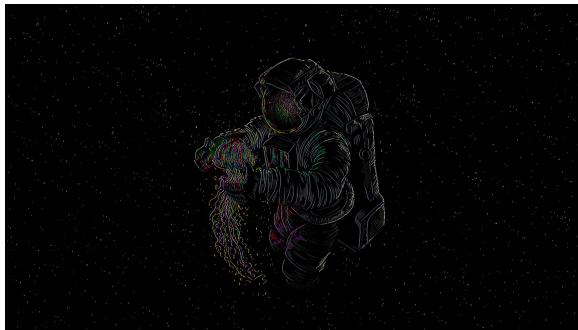
La imagen resultante al aplicar el filtro sobre la imagen de las manzanas (720p) es la siguiente:



Donde se puede apreciar que se detectaron los distintos bordes de la imagen.

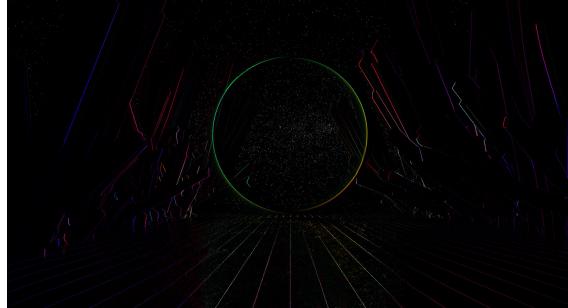
### 2) Imagen 1080p:

La imagen resultante al aplicar el filtro sobre la imagen del astronauta (1080p) es la siguiente:



### 3) Imagen 4K:

La imagen resultante al aplicar el filtro sobre la imagen (4K) es la siguiente:



En todos los casos de prueba se puede evidenciar la correcta aplicación del filtro sobre la imagen original. Sin embargo, es necesario tener en cuenta los tiempos de ejecución y speedup para ver si existe algún tipo de mejora al realizar paralelización sobre el algoritmo.

## B. Pthread (Entrega 1)

Esta primera forma de parallelizar el algoritmo es usando hilos, en este caso se varía el número de hilos en 1, 2, 4, 8, 16 y 32. Obteniendo los siguientes valores para cada resolución:

### 1) Imagen 720p:

Los tiempos de ejecución en segundos y el speedup se muestran a continuación:

[HTML]E0E0E0	[HTML]E0E0E0#hilos	[HTML]E0E0E0Tiempo (s)	[HTML]E0E0E0
	1	0.1220621667	
	2	0.0715495	
	4	0.059001	
	8	0.0540675	
	16	0.05175583333	
	32	0.05188533333	

Estos valores se muestran en la figura 14 y 15, cuyos gráficos representan el tiempo y el speedup respectivamente.

Tiempo vs. #hilos (720P)

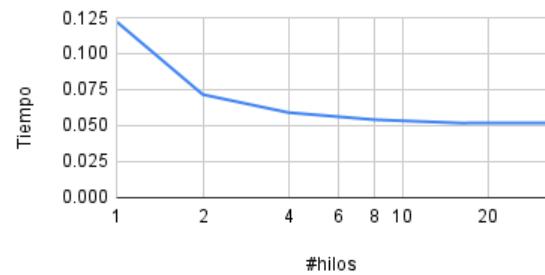


Fig. 9: Tiempo vs #Hilos 720p

SpeedUp vs. #hilos (720p)

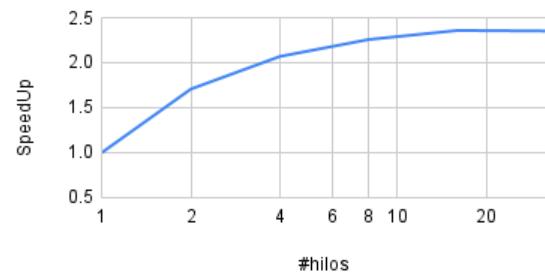


Fig. 10: SpeedUp vs #Hilos 720p

### 2) Imagen 1080p:

La tabla con los tiempos en segundos y el speedup conseguido se muestra a continuación:

[HTML]E0E0E0	[HTML]E0E0E0#hilos	[HTML]E0E0E0Tiempo (s)	[HTML]E0E0E0
	1	0.2559798333	
	2	0.1536336667	
	4	0.115372	
	8	0.1042533333	
	16	0.105398	
	32	0.10721	

Estos valores se muestran en la figura 16 y 17, cuyos gráficos representan el tiempo y el speedup respectivamente.

Tiempo vs #hilos (1080p)

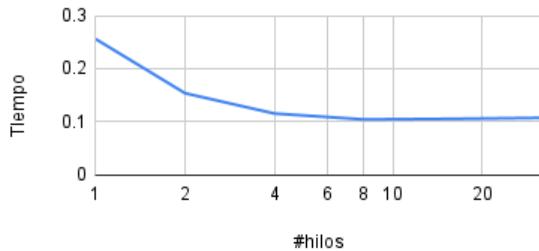


Fig. 11: Tiempo vs #Hilos 1080p

SpeedUp vs #hilos (1080p)

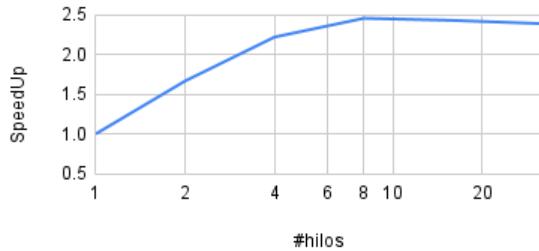


Fig. 12: SpeedUp vs #Hilos 1080p

### 3) Imagen 4K:

La tabla con los tiempos en segundos y el speedup conseguido se muestra a continuación:

[HTML]E0E0E0	[HTML]E0E0E0#hilos	[HTML]E0E0E0Tiempo (s)	[HTML]E0E0E0SpeedUp
	1	1.056775	1
	2	0.6465728333	1.634421304
	4	0.4914915	2.150138914
	8	0.4352833333	2.427786499
	16	0.4506848333	2.34482042
	32	0.4307628333	2.453264112

Estos valores se muestran en la figura 18 y 19, cuyos gráficos representan el tiempo y el speedup respectivamente.

Tiempo vs. #hilos (4K)

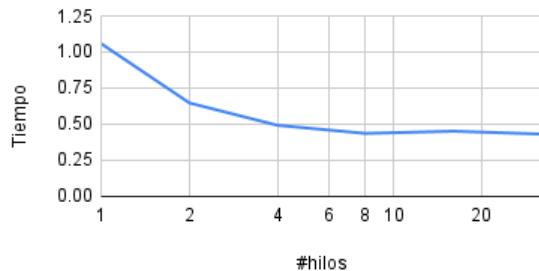


Fig. 13: Tiempo vs #Hilos 4k

SpeedUp vs. #hilos (4K)

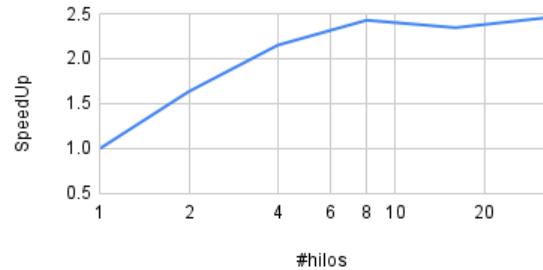
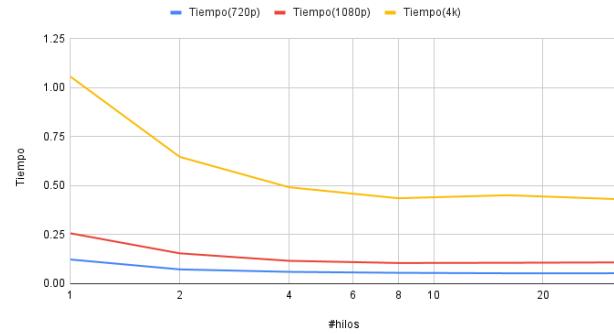


Fig. 14: SpeedUp vs #Hilos 4k

### 4) Comparación entre resoluciones:

A continuación se utilizan los datos obtenidos en cada uno de los experimentos anteriores y se comparan en un gráfico. El gráfico obtenido al realizar la comparación de los tiempos de ejecución es el siguiente:

Tiempo vs. #hilos



Mientras que el gráfico comparativo de los Speedup es el siguiente:

SpeedUp vs. #hilos

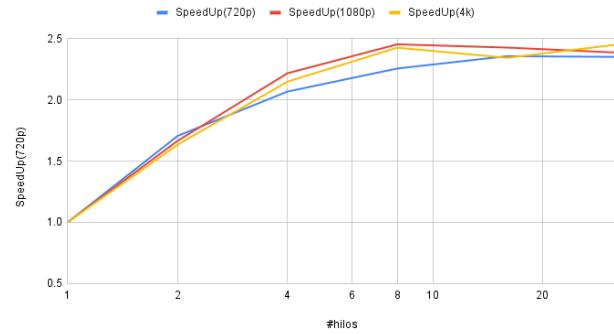


Fig. 16: Comparación SpeedUp tres resoluciones

Mostrando que en cada resolución se consigue un speedup entre 2,4 y 2,5.

### C. OpenMP (Entrega 2)

La segunda forma de parallelizar el algoritmo es usando hilos, de la misma firma que en la primera entrega. El número de hilos para las pruebas va a ser 1, 2, 4, 8, 16 y 32. Obteniendo los siguientes valores para cada resolución:

#### 1) Imagen 720p:

Los tiempos de ejecución en segundos y el speedup se muestran a continuación:

[HTML]E0E0E0	[HTML]E0E0E0#hilos	[HTML]E0E0E0Tiempo (s)	[HTM]
	1	0.1102825	
	2	0.06527666667	
	4	0.0517855	
	8	0.04993266667	
	16	0.043768	
	32	0.04322716667	

Estos valores se muestran en la figura 22 y 23, cuyos gráficos representan el tiempo y el speedup respectivamente.

Tiempo vs. #hilos (720P)

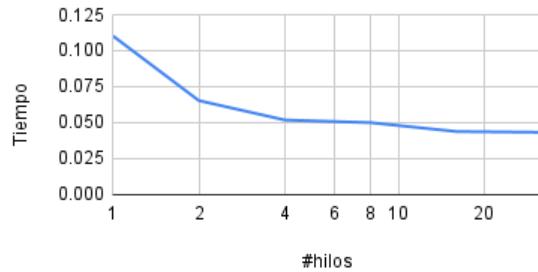


Fig. 17: Tiempo vs #Hilos 720p

SpeedUp vs. #hilos (720p)

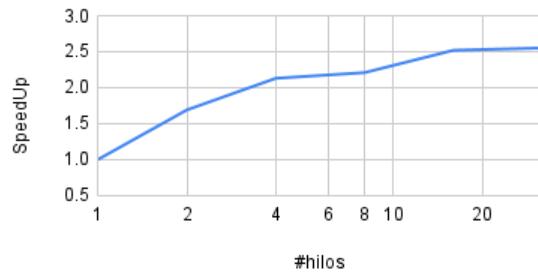


Fig. 18: SpeedUp vs #Hilos 720p

#### 2) Imagen 1080p:

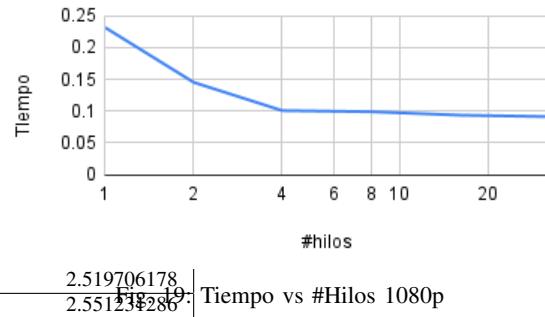
Los tiempos de ejecución en segundos y el speedup se muestran a continuación:

[HTML]E0E0E0	[HTML]E0E0E0#hilos	[HTML]E0E0E0Tiempo (s)	[HTM]
	1	0.2313803333	
	2	0.1451878333	
	4	0.1007515	
	8	0.0990045	
	16	0.0936125	
	32	0.0912388333	

Estos valores se muestran en la figura 24 y 25, cuyos gráficos representan el tiempo y el speedup respectivamente.

Estos valores se muestran en la figura 24 y 25, cuyos gráficos representan el tiempo y el speedup respectivamente.

Tiempo vs #hilos (1080p)



SpeedUp vs #hilos (1080p)

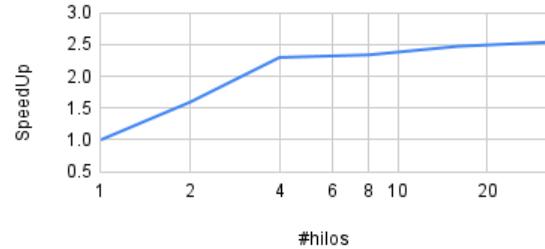


Fig. 20: SpeedUp vs #Hilos 1080p

#### 3) Imagen 4K:

Los tiempos de ejecución en segundos y el speedup se muestran a continuación:

[HTML]E0E0E0	[HTML]E0E0E0#hilos	[HTML]E0E0E0Tiempo (s)	[HTM]
	1	0.9825743333	
	2	0.5837858333	
	4	0.4080843333	
	8	0.3800511667	
	16	0.3786171667	
	32	0.3639261667	

Estos valores se muestran en la figura 26 y 27, cuyos gráficos representan el tiempo y el speedup respectivamente.

Tiempo vs. #hilos (4K)



### SpeedUp vs. #hilos (4K)

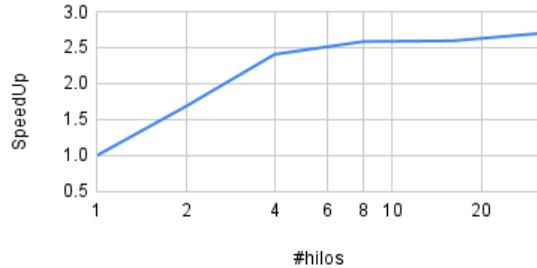


Fig. 22: SpeedUp vs #Hilos 4k

Mostrando que en cada resolución se consigue un speedup entre 2,5 y 2,7.

#### 4) Comparación entre resoluciones:

A continuación se utilizan los datos obtenidos en cada uno de los experimentos anteriores y se comparan en un gráfico. El gráfico obtenido al realizar la comparación de los tiempos de ejecución es el siguiente:

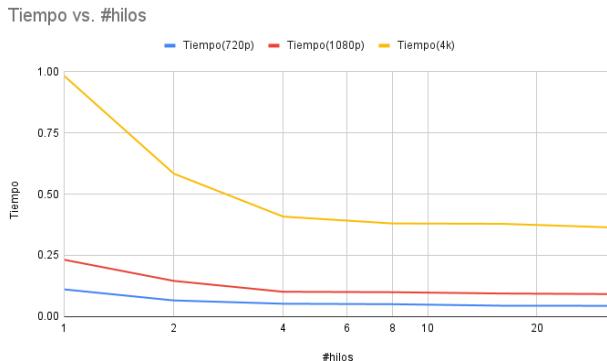


Fig. 23: Comparación tiempos de ejecución tres resoluciones

Mientras que el gráfico comparativo de los Speedup es el siguiente:



Fig. 24: Comparación SpeedUp tres resoluciones

### D. Comparativa Pthread vs OpenMP

Para obtener una comparativa entre las dos librerías utilizadas para la creación de hilos se realizaron los gráficos de tiempos de ejecución y speedup para cada una de las resoluciones.

#### 1) Imagen 720p:

Los gráficos con la ejecución en segundos y el speedup de las dos librerías se muestran en las figuras 30 y 31:

#### Tiempo vs. #hilos (Imágenes 720p)

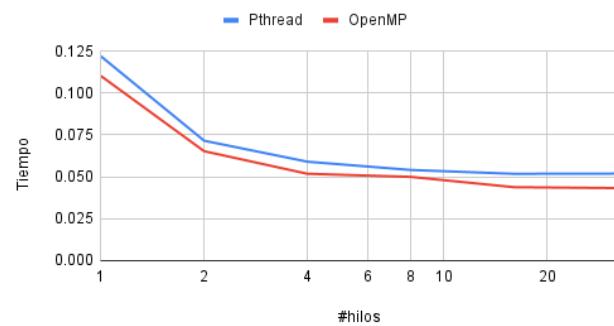


Fig. 25: Comparación tiempo de ejecución (720p) Pthread vs OpenMP

#### Speedup vs. #hilos (Imágenes 720p)

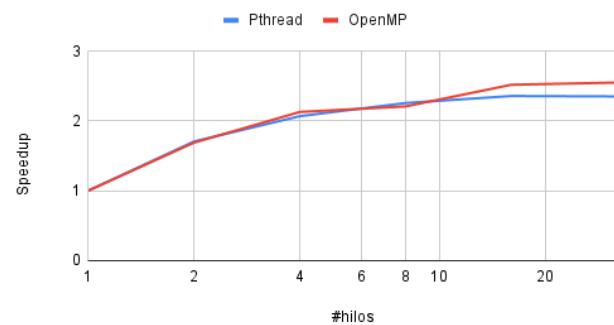


Fig. 26: Comparación SpeedUp (720p) Pthread vs OpenMP

#### 2) Imagen 1080p:

Los gráficos con la ejecución en segundos y el speedup de las dos librerías se muestran en las figuras 32 y 33:

Tiempo vs. #hilos (Imágenes 1080p)

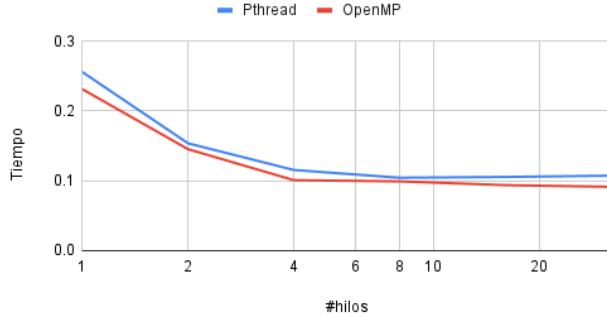


Fig. 27: Comparación tiempo de ejecución (1080p) Pthread vs OpenMP

Speedup vs. #hilos (Imágenes 4K)

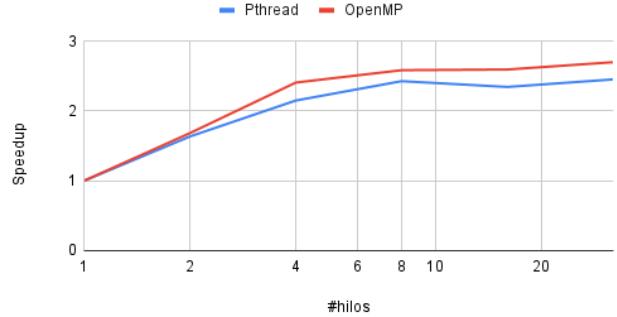


Fig. 30: Comparación SpeedUp (4K) Pthread vs OpenMP

Mostrando resultados ligeramente superiores al utilizar openMP.

#### E. CUDA

A continuación se muestran los gráficos de los resultados obtenidos al realizar la paralelización usando CUDA.

##### 1) Imagen 720p:

Debido a que se manejan dos variables para el cálculo del tiempo de ejecución y del speedup (Número de Bloques y Número de Hilos), es necesario realizar un gráfico con los resultados que se obtienen al variar el número de bloques y comparar los resultados.

Los gráficos con la ejecución en segundos y el speedup de las dos librerías se muestran en las figuras 36 y 37:

Speedup vs. #hilos (Imágenes 1080p)

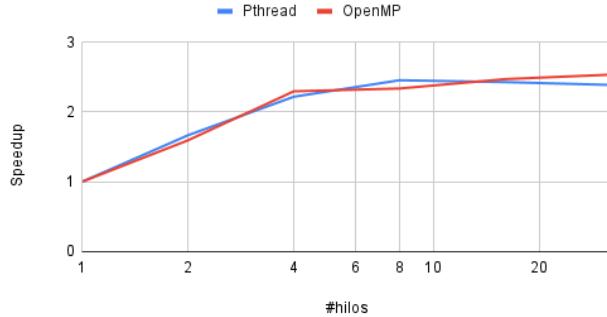


Fig. 28: Comparación SpeedUp (1080p) Pthread vs OpenMP

##### 3) Imagen 4K:

Los gráficos con la ejecución en segundos y el speedup de las dos librerías se muestran en las figuras 34 y 35:

Tiempo vs. #hilos (Imágenes 4K)

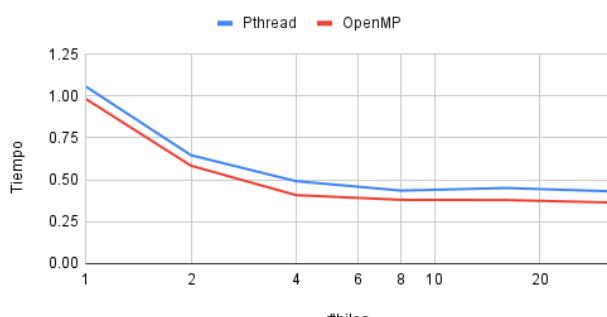


Fig. 29: Comparación tiempo de ejecución (4K) Pthread vs OpenMP

Fig. 31: Tiempos de ejecución (720p) CUDA

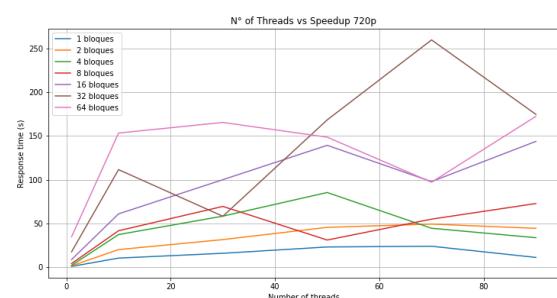


Fig. 32: Speedup (720p) CUDA

### 2) Imagen 1080p:

De forma similar se usan las variables (Número de Bloques y Número de Hilos), para realizar los gráficos.

Los gráficos con la ejecución en segundos y el speedup de las dos librerías se muestran en las figuras 38 y 39:

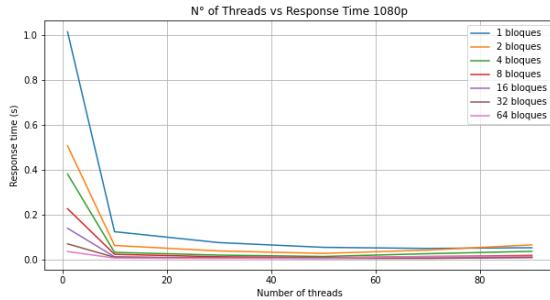


Fig. 33: Tiempos de ejecución (1080p) CUDA

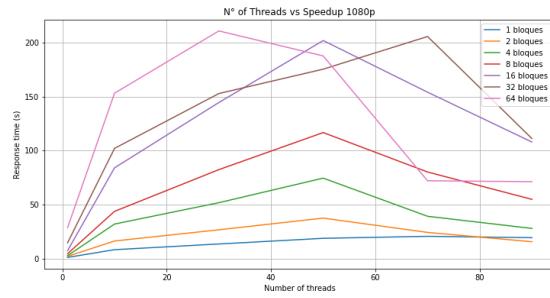


Fig. 34: Speedup (1080p) CUDA

### 3) Imagen 4K:

De forma similar se usan las variables (Número de Bloques y Número de Hilos), para realizar los gráficos.

Los gráficos con la ejecución en segundos y el speedup de las dos librerías se muestran en las figuras 40 y 41:

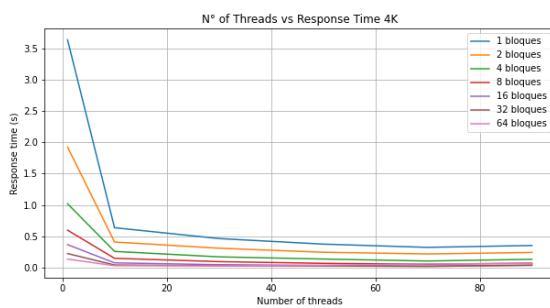


Fig. 35: Tiempos de ejecución (4K) CUDA

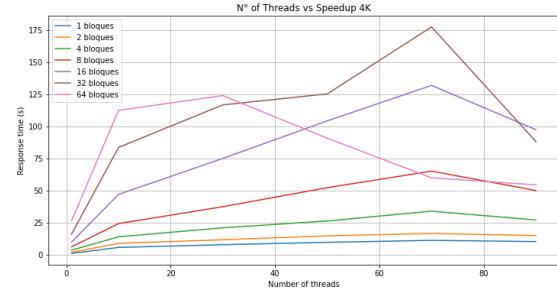


Fig. 36: Speedup (4K) CUDA

### F. OpenMPI (Entrega 4)

Para esta última entrega se cambió la forma en la que se realiza la parallelización, utilizando procesos. Para estas pruebas se creó un cluster usando Google Cloud Console, este cluster tenía 8 máquinas virtuales, cada una con dos núcleos. Por lo que se disponía de un máximo de 16 nodos para la ejecución del algoritmo. En este caso el número de procesos varió entre 1, 2, 4, 8 y 16.

#### 1) Imagen 720p:

Los tiempos de ejecución en segundos y el speedup se muestran a continuación:

[HTML]E0E0E0	[HTML]E0E0E0#hilos	[HTML]E0E0E0Tiempo (s)	[HTML]E0E0E0Speedup
	1	0.0672586667	1
	2	0.060159	2
	4	0.030431	4
	8	0.0157581667	8
	16	0.008554	16

Estos valores se muestran en la figura 42 y 43, cuyos gráficos representan el tiempo y el speedup 43.

### Tiempo vs. #procesos (720P)

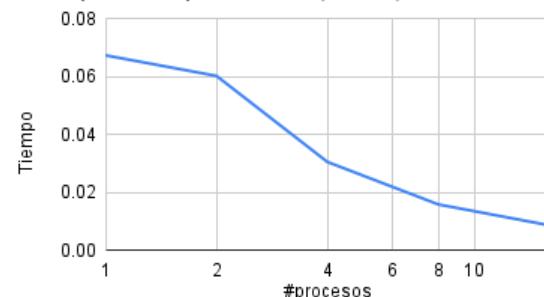


Fig. 37: Tiempo vs #Procesos 720p

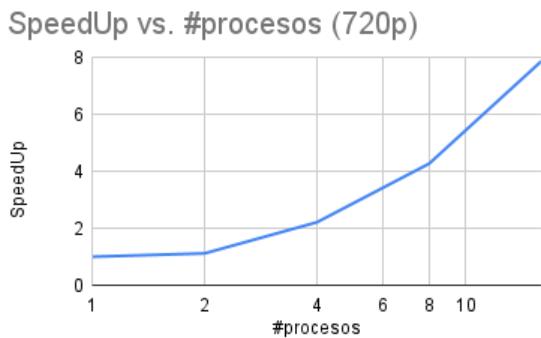


Fig. 38: SpeedUp vs #Procesos 720p

### 2) Imagen 1080p:

Los tiempos de ejecución en segundos y el speedup se muestran a continuación:

[HTML]E0E0E0	[HTML]E0E0E0#hilos	[HTML]E0E0E0Tiempo (s)	[HTN]
	1	0.1515245	
	2	0.1343546667	
	4	0.0685585	
	8	0.03542833333	
	16	0.01790383333	

Estos valores se muestran en la figura 44 y 45, cuyos gráficos representan el tiempo y el speedup respectivamente.

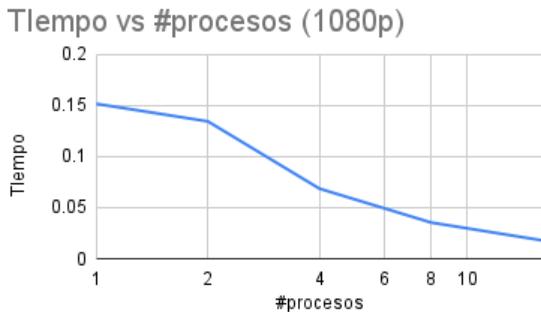


Fig. 39: Tiempo vs #Procesos 1080p

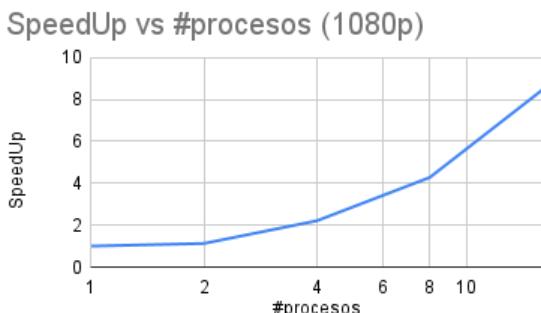


Fig. 40: SpeedUp vs #Procesos 1080p

### 3) Imagen 4K:

Los tiempos de ejecución en segundos y el speedup se muestran a continuación:

[HTML]E0E0E0	[HTML]E0E0E0#hilos	[HTML]E0E0E0Tiempo (s)	[HTML]E0E0E0SpeedUp
	1	0.61343	
	2	0.537634	
	4	0.274106	
	8	0.1398358	
	16	0.07188216667	

Estos valores se muestran en la figura 46 y 47, cuyos gráficos representan el tiempo y el speedup respectivamente.

### Tiempo vs. #procesos (4K)

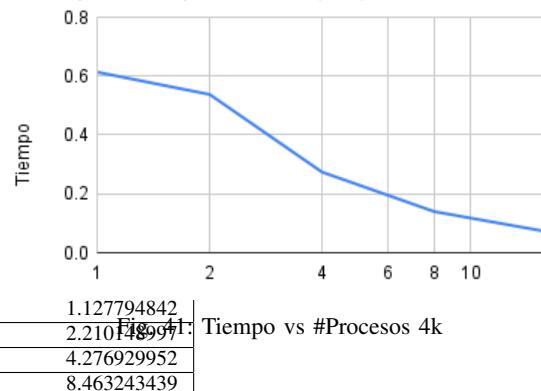


Fig. 41: Tiempo vs #Procesos 4k

### SpeedUp vs. #procesos (4K)

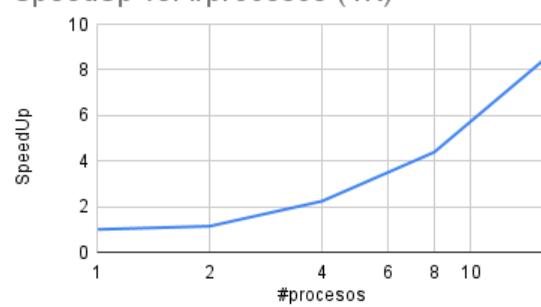


Fig. 42: SpeedUp vs #Procesos 4k

### 4) Comparación entre resoluciones:

A continuación se utilizan los datos obtenidos en cada uno de los experimentos anteriores y se comparan en un gráfico. El gráfico obtenido al realizar la comparación de los tiempos de ejecución es el siguiente:

#### Tiempo vs. #procesos comparativa

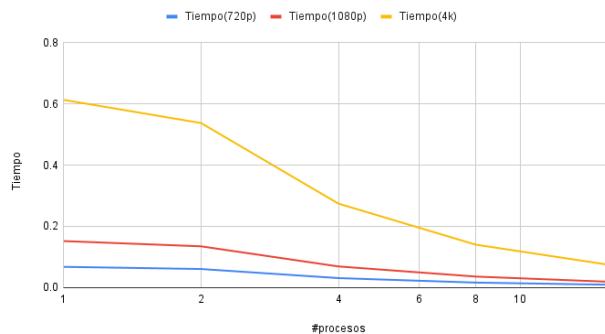


Fig. 43: Comparación tiempos de ejecución tres resoluciones

Mientras que el gráfico comparativo de los Speedup es el siguiente:

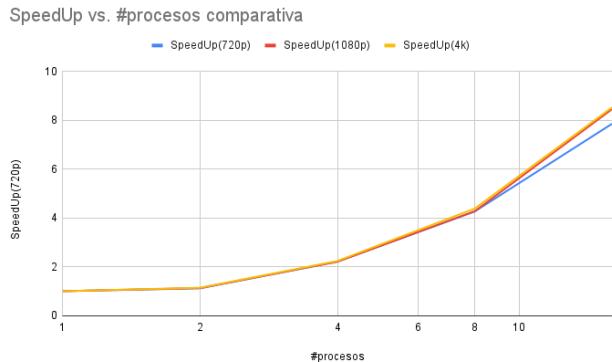


Fig. 44: Comparación SpeedUp tres resoluciones

## VII. CONCLUSIONES

De la práctica realizada se puede concluir:

- 1) El algoritmo para la aplicación del filtro sobre la imagen funciona correctamente detectando los bordes de la imagen original.
- 2) Los otros kernel disponibles para la aplicación del filtro producen resultados diferentes, con diferentes tipos de bordes detectados y colores resultantes.
- 3) Para las tres resoluciones de imágenes el filtro se aplica de forma correcta tanto de forma secuencial como paralela. Además, se puede evidenciar que al aumentar el número de hilos o procesos el tiempo de ejecución disminuye hasta un punto de estabilidad.
- 4) En todos los casos de prueba, excepto en CUDA, el comportamiento del SpeedUp es similar, aumentando cuando el número de hilos aumenta, hasta un punto máximo donde se estabiliza.
- 5) Los tiempos de procesamiento de imágenes 4K utilizando CPU considerable, teniendo que esperar entre 0.4 y 1.1 segundos. Esto puede ser problemático si se quisiera aplicar el filtro sobre un conjunto de imágenes con la misma resolución o sobre un video.
- 6) La paralelización del algoritmo mejora de gran medida el tiempo de procesamiento de la imagen, reduciendo el tiempo de ejecución a casi la mitad del secuencial.
- 7) La GPU funciona de mejor manera que la CPU para el procesamiento de imágenes, esto se puede apreciar comparando los tiempos obtenidos, llegando a procesar imágenes 4K usando la GPU de forma más rápida que la CPU procesando imágenes 720p.
- 8) CUDA y OpenMPI permitirán la obtención de mejores resultados que al utilizar paralelización por hilos usando pThread u OpenMP.
- 9) El repositorio con las distintas prácticas de la asignatura es el siguiente:  
<https://github.com/jucarmonam/Computacion-paralela>

## REFERENCES