

Computación Paralela y Distribuida

Juan Pablo Carmona

Departamento de Sistemas e Industrial
Universidad Nacional de Colombia
Bogotá - Colombia

jucarmonam@unal.edu.co

Juan Sebastián Rodríguez

Departamento de Sistemas e Industrial
Universidad Nacional de Colombia
Bogotá - Colombia

juarodriguezc@unal.edu.co

Abstract—En este documento se presenta el desarrollo de las prácticas de la asignatura *Computación Paralela y Distribuida*, en la cual se desarrolla un algoritmo que procesa imágenes en color (720p, 1080p y 4K.) y les aplica un filtro utilizando una operación de convolución. El programa funciona de forma secuencial (1 hilo) y de forma paralela con tres enfoques distintos: Usando Pthread y OpenMP para 2, 4, 8, 16 y 32 hilos; usando CUDA variando el número de bloques y de cores; y OpenMPI con 1, 2, 4, 8 y 16 procesos.

Palabras clave: Filtro, Paralelización, Matriz, Convolución, Speedup.

I. INTRODUCCIÓN

La computación paralela es una forma de computo que trae muchas ventajas con ciertas tareas, ya que permite realizar actividades muy largas o repetitivas de forma más rápida. Para esto se reparte, puede ser en partes iguales o no, una porción del calculo o del algoritmo que se realiza muchas veces, a cada uno de los núcleos de nuestro computador. Cada uno de estos núcleos trabaja al mismo tiempo y juntan sus resultados para encontrar la solución final, lo cual en muchos casos permitirá obtener un resultado de manera mas rápida, ya que no será un solo hilo haciendo todos los cálculos de manera secuencial sino que varios hilos trabajando al mismo tiempo pero con fracciones mas pequeñas del problema.

Con respecto a lo anterior, una de las aplicaciones más grandes de la computación paralela es el procesamiento gráfico y mas específicamente el procesamiento de imágenes. En esta aplicación se busca aplicar diferentes filtros a imágenes para obtener un tipo de información deseada, sin embargo, aplicar estos filtros significa realizar distintos cálculos sobre la imagen (la operación de convolución es muy utilizada en filtros), pero a nivel de píxeles. Por lo que a cada píxel se le debe hacer la operación de filtrado, es por esto que a medida que aumenta la resolución de la imagen, mayor será la cantidad de cálculos que se deberán hacer, en ese sentido la computación paralela es una de las mejores alternativas para solucionar este tipo de cálculos y procesamientos, ya que nos permitirá reducir los tiempos de ejecución en gran medida sin perder ningún tipo de calidad.

Con lo cual el presente trabajo busca mostrar cómo usar la computación paralela para aplicar un **filtro** de detección de bordes a diferentes tipos de imágenes y cómo el uso de mas núcleos en primera instancia mejora el rendimiento de gran manera.

II. DISEÑO DE LA APLICACIÓN

El programa se desarrolló sobre C, utilizando distintas librerías:

- 1) **pthread.h:** Para la creación de hilos
- 2) **omp.h:** Para la creación de hilos
- 3) **CUDA** Para la creación de hilos en GPU
- 4) **OpenMPI** Para la creación de procesos
- 5) **stb_image:** Para la carga y escritura de imágenes
- 6) **sys/time.h:** Para la medición de tiempos

El programa presenta ligeras variaciones respecto al número de parámetros requeridos para su funcionamiento. Al realizar la paralelización con hilos y con procesos, se necesitan cuatro parámetros para el correcto funcionamiento: El nombre de la imagen de entrada, el nombre con el que se va a guardar la imagen, el parámetro de kernel de **convolución** y el número de hilos o procesos para la ejecución. Mientras que al usar CUDA, son requeridos cinco parámetros para el correcto funcionamiento: El nombre de la imagen de entrada, el nombre con el que se va a guardar la imagen, el parámetro de kernel de **convolución**, el número de bloques y el número de hilos por bloque.

Al usar la librería **stb_image** para leer la imagen [2], se crea un arreglo de *unsigned char* de tamaño *width * height * channels*, en caso de ser un *JPG* se tienen 3 canales (RGB) y si es un *PNG* se tienen 4 canales (RGBA). Una vez leída la imagen se procede a separar cada canal de color, una primera aproximación es crear una **matriz** para cada canal, este proceso se puede evidenciar en la figura 1.

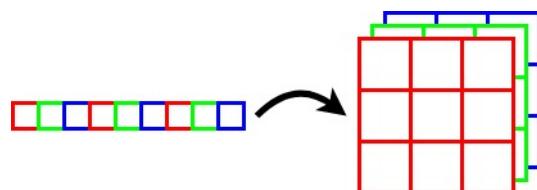


Fig. 1: Transformación imagen stb_image a matrices de color

Sin embargo, para la implementación del algoritmo no se utilizaron matrices, sino que se almacenaron los valores de cada canal en su respectivo arreglo, aplandando la **matriz** de píxeles. La figura 2 explica de forma breve el proceso utilizado para almacenar los píxeles.

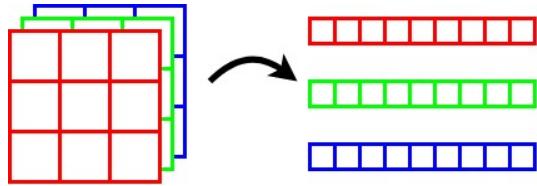


Fig. 2: Representación de matriz como arreglo

Con los canales de color creados se procede a realizar **convolución** [1] sobre cada uno de los píxeles de la imagen. El kernel utilizado para la detección básica de bordes es el siguiente:

$$\begin{array}{|c|c|c|} \hline -1 & 0 & 1 \\ \hline -2 & 0 & 2 \\ \hline -1 & 0 & 1 \\ \hline \end{array}$$

Fig. 3: Kernel detección de bordes (Sobel)

Sin embargo, al tratarse de un filtro por convolución, estos valores puedan cambiar, obteniendo otros filtros. La operación de **convolución** permite aplicar gran cantidad de **filtros** sobre imágenes y la lógica aplicada se puede apreciar en la figura 4.

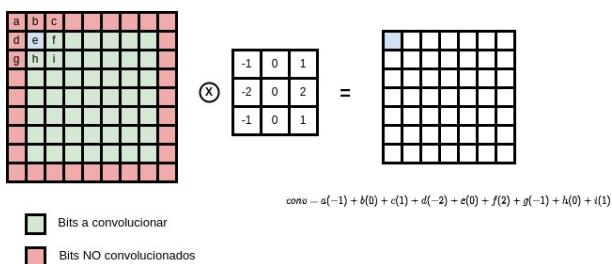


Fig. 4: Representación de convolución sobre una matriz

Una vez realizada esta operación sobre cada uno de los canales, se obtiene una nueva imagen en donde se resaltan únicamente los bordes de la imagen original, por lo que se procede a exportar la imagen. Para esto se realiza el proceso inverso al realizado en la figura 1. Obteniendo un arreglo *unsigned char* que será procesado por la librería **stb_image** para guardar la imagen.

No obstante, la lógica mencionada anteriormente funciona de manera secuencial, por lo que es necesario tener en cuenta otros factores para realizar una correcta **parallelización**. Al ejecutar el algoritmo de forma secuencial el número n de píxeles son procesados únicamente por 1 núcleo (figura 5), sin embargo es posible optimizar esta operación usando más núcleos.

hilos = 1 # píxeles = n

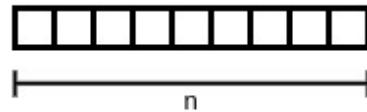


Fig. 5: Balanceo de carga en secuencial

Para esto se divide el número de píxeles a procesar en el número total de hilos o procesos solicitados por el usuario (En el caso de CUDA, el total de hilos es el número de bloques multiplicado por el número de cores por bloque). Por lo que se utiliza un balanceo de cargas **blockwise**. Sin embargo, esta división debe tener en cuenta los píxeles sobrantes, por lo que a cada hilo se le asigna $n/\text{Threads}$ o $n/\text{Threads} + 1$ píxeles. La figura 6 ejemplifica el balanceo de cargas utilizado.

hilos = 2 # píxeles = n

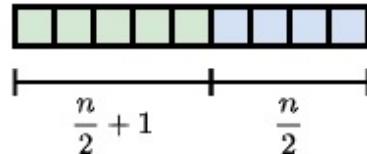


Fig. 6: Balanceo de carga en paralelo (2 hilos)

Una vez repartido los píxeles para cada hilo, se procede a realizar la operación de **convolución**. Otra forma de ver el balanceo de cargas se presenta a continuación en la figura 7.

hilos = 2 # píxeles = n

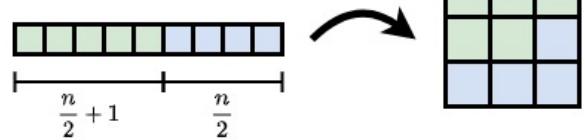


Fig. 7: Balanceo de carga en paralelo (Representación matricial)

Donde se puede evidenciar de mejor manera cómo se reparte cada uno de los píxeles de la **matriz**, en el número de hilos solicitado.

III. EXPERIMENTOS

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. 8: Imagen de manzanas 1280x720 px

2) *Imagen 1080p:*

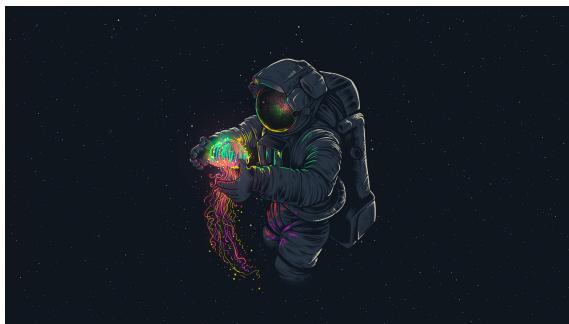


Fig. 9: Imagen de astronauta 1920x1080 px

3) *Imagen 4K:*



Fig. 10: Imagen de fondo 3840x2160 px

IV. 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:

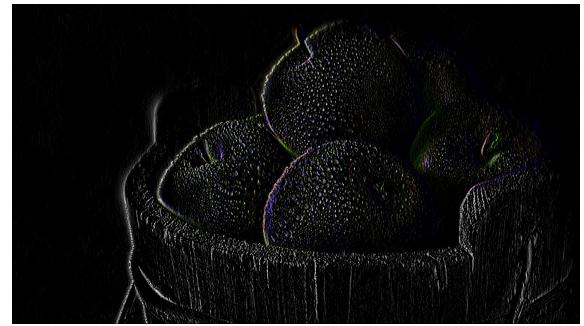


Fig. 11: Resultado del filtro imagen 720p

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:



Fig. 12: Resultado del filtro imagen 1080p

3) *Imagen 4K:*

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

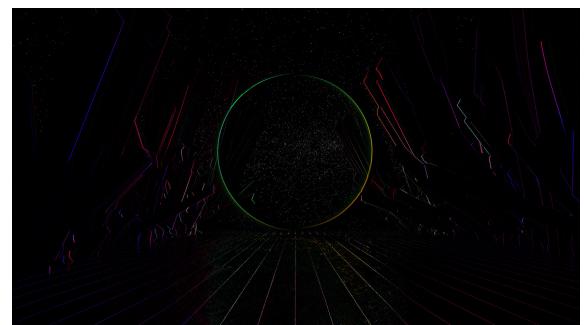


Fig. 13: Resultado del filtro imagen 4k

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 paralelizar 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:

#hilos	Tiempo (s)	SpeedUp
1	0.1220621667	1
2	0.0715495	1.705982106
4	0.059001	2.068815218
8	0.0540675	2.257588508
16	0.05175583333	2.358423366
32	0.05188533333	2.352537005

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

Tiempo vs. #hilos (720P)

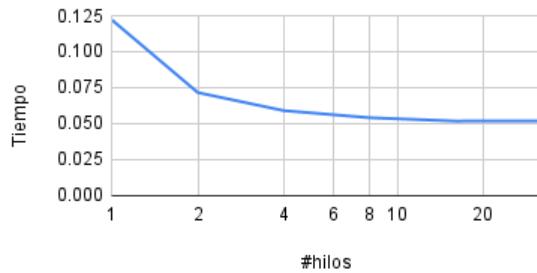


Fig. 14: Tiempo vs. #hilos 720p

SpeedUp vs. #hilos (720p)

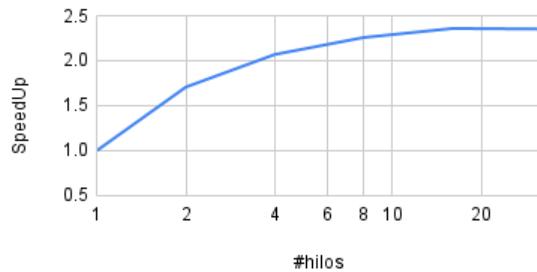


Fig. 15: SpeedUp vs. #hilos 720p

2) Imagen 1080p:

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

#hilos	Tiempo (s)	SpeedUp
1	0.2559798333	1
2	0.1536336667	1.666170175
4	0.115372	2.218734471
8	0.1042533333	2.455363538
16	0.105398	2.428697255
32	0.10721	2.387648851

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

Tiempo vs. #hilos (1080p)

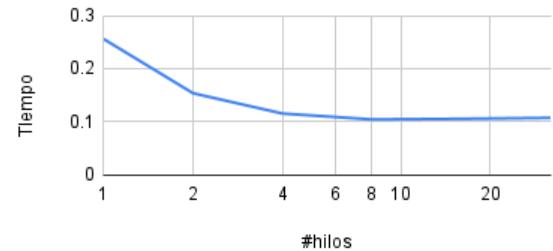


Fig. 16: Tiempo vs. #Hilos 1080p

SpeedUp vs. #hilos (1080p)

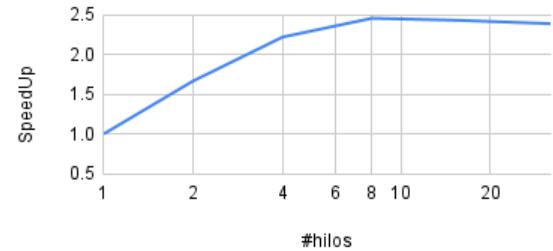


Fig. 17: SpeedUp vs. #Hilos 1080p

3) Imagen 4K:

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

#hilos	Tiempo (s)	SpeedUp
1	1.056775	1
2	0.6465728333	1.634425304
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)

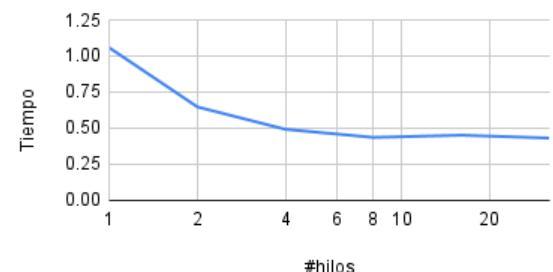


Fig. 18: Tiempo vs. #Hilos 4k



Fig. 19: 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:

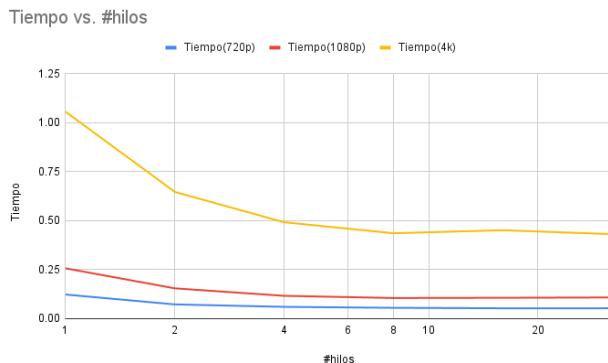


Fig. 20: Comparación tiempos de ejecución tres resoluciones

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

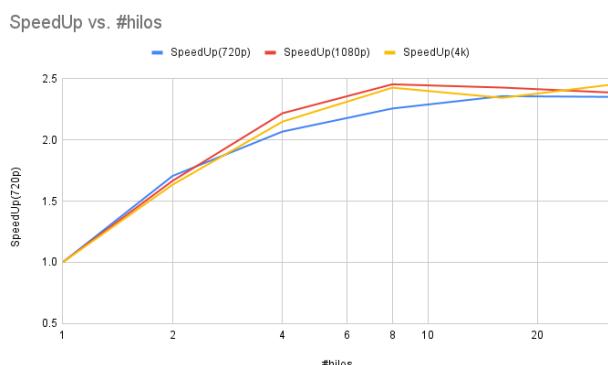


Fig. 21: Comparación SpeedUp tres resoluciones

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

C. OpenMP (Entrega 2)

La segunda forma de paralelizar 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:

#hilos	Tiempo (s)	SpeedUp
1	0.1102825	1
2	0.06527666667	1.689462799
4	0.0517855	2.129601916
8	0.04993266667	2.208624281
16	0.043768	2.519706178
32	0.04322716667	2.551231286

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

Tiempo vs. #hilos (720P)

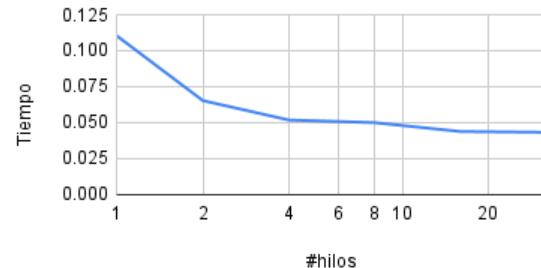


Fig. 22: Tiempo vs #Hilos 720p

SpeedUp vs. #hilos (720p)

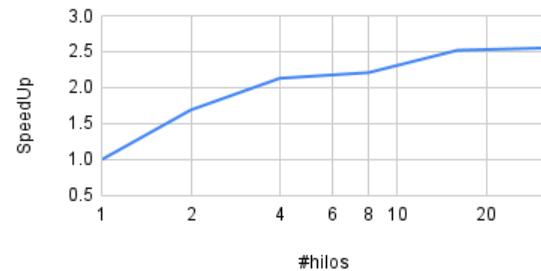


Fig. 23: SpeedUp vs #Hilos 720p

2) Imagen 1080p:

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

#hilos	Tiempo (s)	SpeedUp
1	0.2313803333	1
2	0.1451878333	1.593662003
4	0.1007515	2.296544799
8	0.0990045	2.337068854
16	0.0936125	2.471682023
32	0.09123883333	2.535985226

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

Tiempo vs #hilos (1080p)

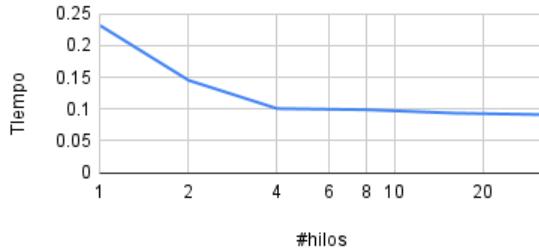


Fig. 24: Tiempo vs #Hilos 1080p

SpeedUp vs #hilos (1080p)

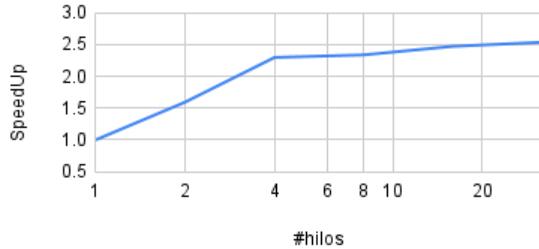


Fig. 25: SpeedUp vs #Hilos 1080p

3) Imagen 4K:

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

#hilos	Tiempo (s)	SpeedUp
1	0.9825743333	1
2	0.5837858333	1.683107532
4	0.4080843333	2.40777274
8	0.3800511667	2.585373812
16	0.3786171667	2.595165829
32	0.3639261667	2.699927687

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

Tiempo vs. #hilos (4K)

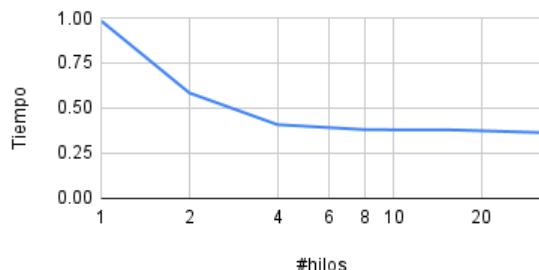


Fig. 26: Tiempo vs #Hilos 4k

SpeedUp vs. #hilos (4K)

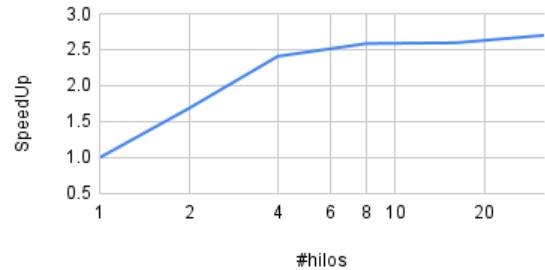


Fig. 27: SpeedUp vs #Hilos 4k

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:

Tiempo vs. #hilos

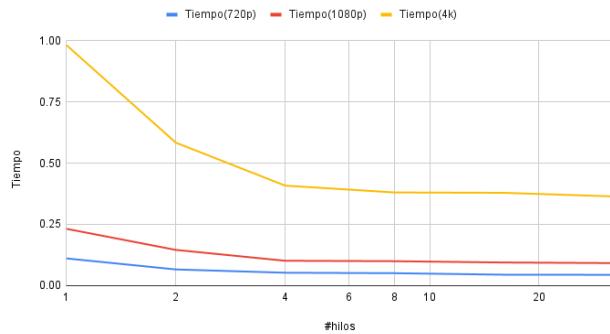


Fig. 28: Comparación tiempos de ejecución tres resoluciones

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

SpeedUp vs. #hilos

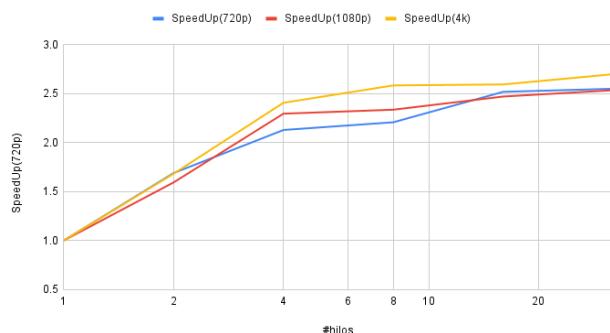


Fig. 29: Comparación SpeedUp tres resoluciones

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)

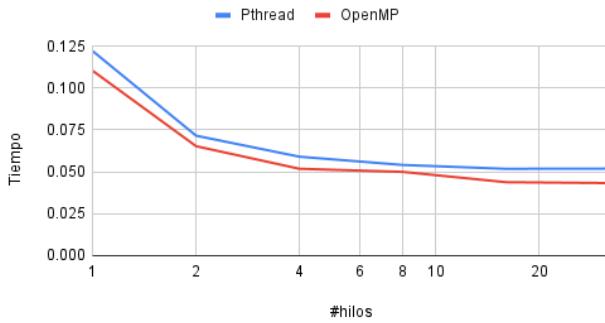


Fig. 30: Comparación tiempo de ejecución (720p) Pthread vs OpenMP

Tiempo vs. #hilos (Imágenes 1080p)

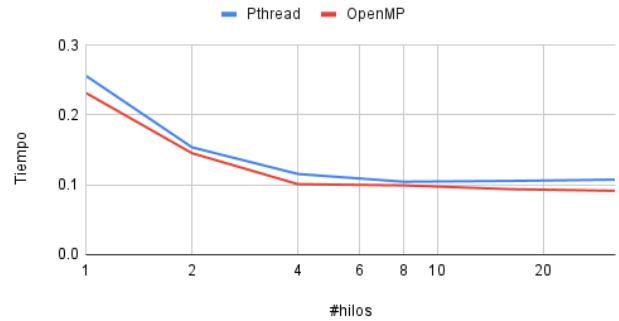


Fig. 32: Comparación tiempo de ejecución (1080p) Pthread vs OpenMP

Speedup vs. #hilos (Imágenes 1080p)

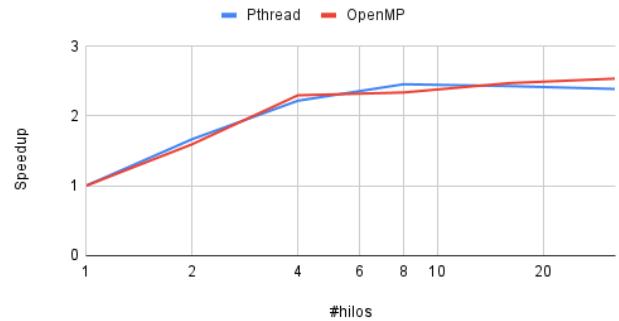


Fig. 33: Comparación SpeedUp (1080p) Pthread vs OpenMP

Speedup vs. #hilos (Imágenes 720p)

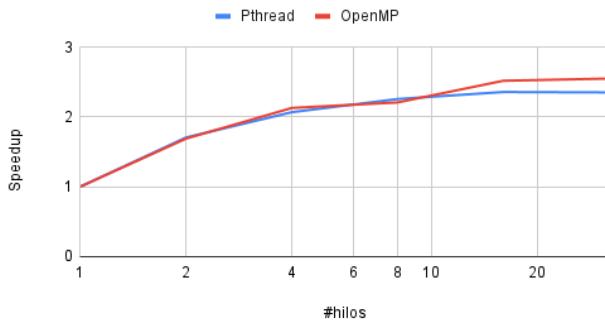


Fig. 31: Comparación SpeedUp (720p) Pthread vs OpenMP

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)

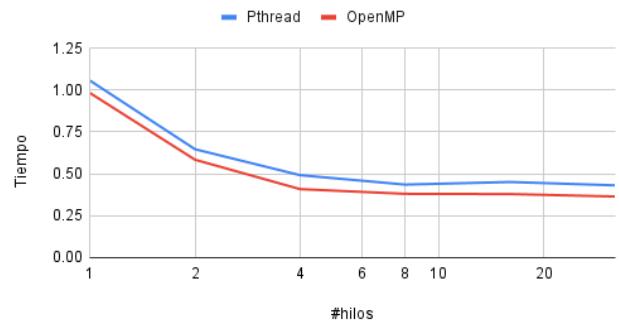


Fig. 34: Comparación tiempo de ejecución (4K) Pthread vs OpenMP

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:

Speedup vs. #hilos (Imágenes 4K)

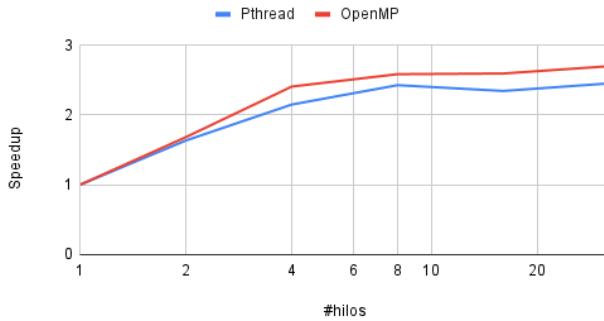


Fig. 35: Comparación SpeedUp (4K) Pthread vs OpenMP

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:

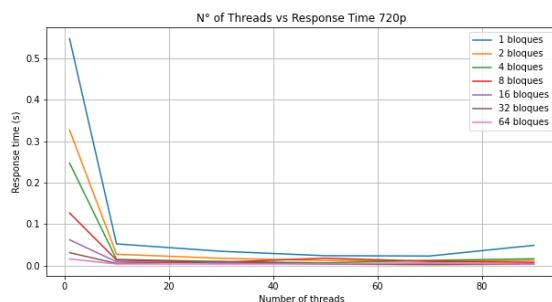


Fig. 36: Tiempos de ejecución (720p) CUDA

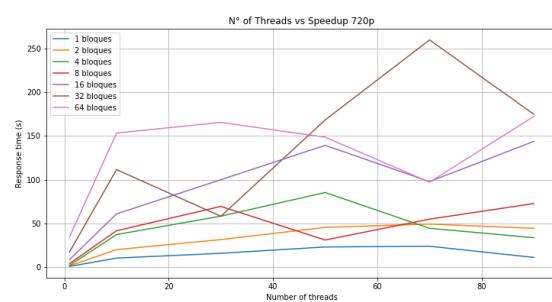


Fig. 37: Speedup (720p) CUDA

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:

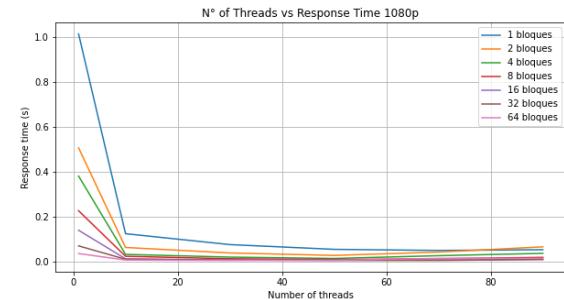


Fig. 38: Tiempos de ejecución (1080p) CUDA

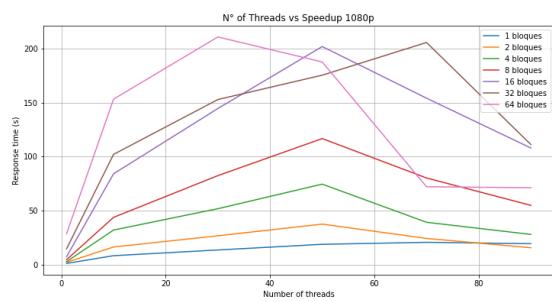


Fig. 39: Speedup (1080p) CUDA

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:

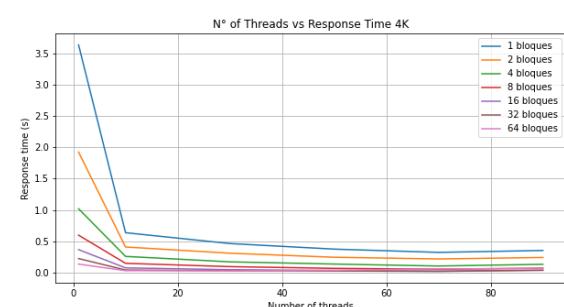


Fig. 40: Tiempos de ejecución (4K) CUDA

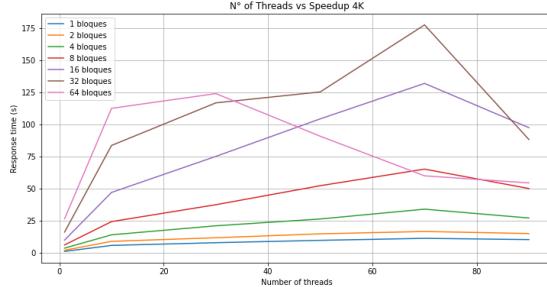


Fig. 41: Speedup (4K) CUDA

SpeedUp vs. #procesos (720p)

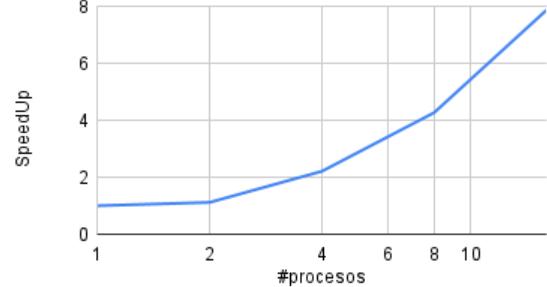


Fig. 43: SpeedUp vs #Procesos 720p

F. OpenMPI (Entrega 4)

Para esta última entrega se cambió la forma en la que se realiza la paralelización, 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:

#hilos	Tiempo (s)	SpeedUp
1	0.06725866667	1
2	0.060159	1.118015038
4	0.030431	2.210202316
8	0.01575816667	4.268178405
16	0.008554	7.862832203

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

Tiempo vs. #procesos (720P)

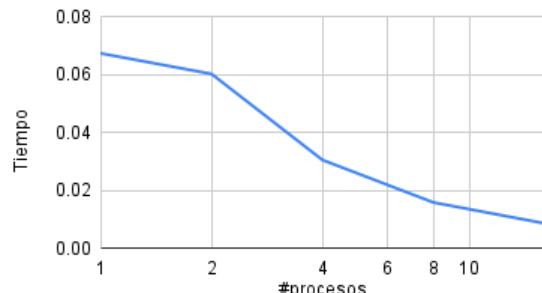


Fig. 42: Tiempo vs #Procesos 720p

Tiempo vs #procesos (1080p)

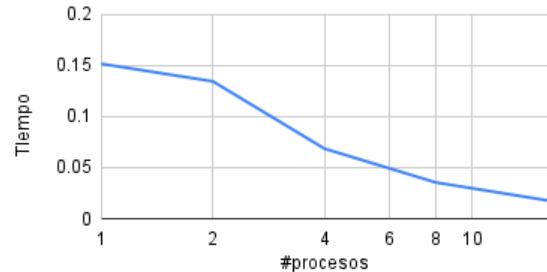


Fig. 44: Tiempo vs #Procesos 1080p

SpeedUp vs #procesos (1080p)

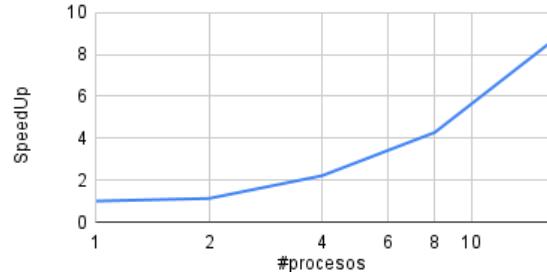


Fig. 45: SpeedUp vs #Procesos 1080p

3) Imagen 4K:

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

#hilos	Tiempo (s)	SpeedUp
1	0.61343	1
2	0.537634	1.140980667
4	0.274106	2.237929852
8	0.1398358	4.386787933
16	0.07188216667	8.533827352

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

Tiempo vs. #procesos (4K)

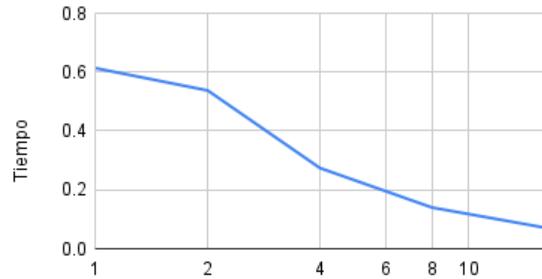


Fig. 46: Tiempo vs #Procesos 4k

SpeedUp vs. #procesos (4K)

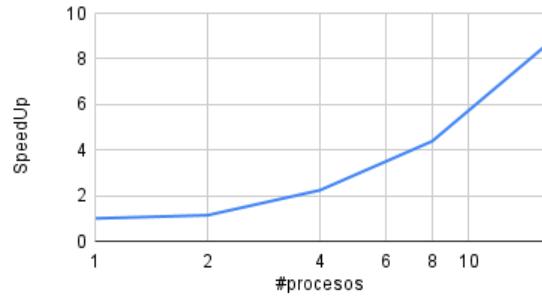


Fig. 47: SpeedUp vs #Procesos 4k

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

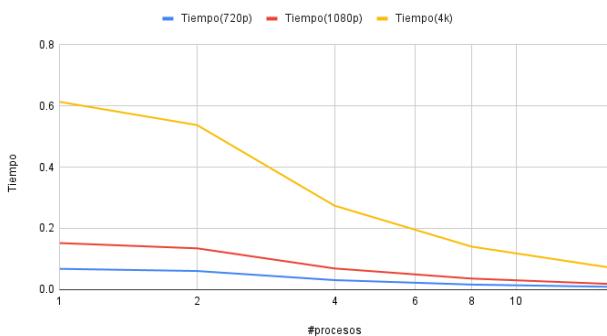
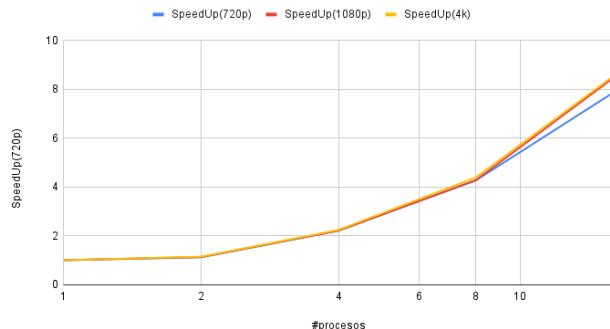


Fig. 48: Comparación tiempos de ejecución tres resoluciones

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

SpeedUp vs. #procesos comparativa



V. 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 permitieron 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

- [1] <https://docs.gimp.org/>. Matriz de convolucióng. URL:
<https://docs.gimp.org/2.6/es/plug-in-convmatrix.html>.
- [2] Solarian Programmer. C programming - reading and writing images with the *stbmagelibraries*. URL :
<https://github.com/nothings/stb>.