

# Práctica 1, 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 la primera práctica 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 en forma secuencial (1 hilo) y de forma paralela (2, 4, 8, 16 hilos).

**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) **stb\_image:** Para la carga y escritura de imágenes
- 3) **sys/time.h:** Para la medición de tiempos

El programa necesita 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 para la ejecución.

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.

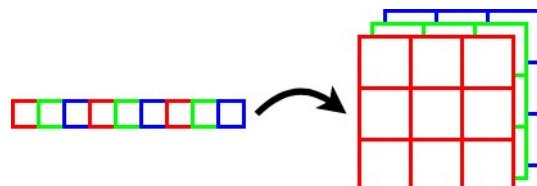


Fig. 1: Transformación imagen stb\_image a matrices de color

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.

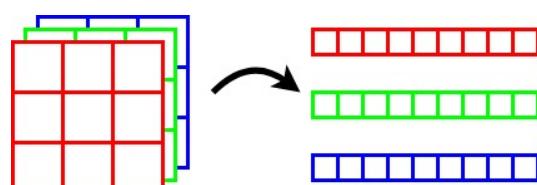


Fig. 2: Representación de matriz como arreglo

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:

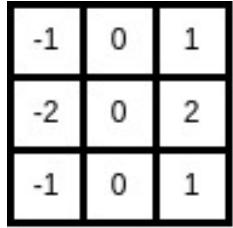


Fig. 3: Kernel de detección de bordes (Sobel)

La operación de **convolución** permite aplicar gran cantidad de **filtros** sobre imágenes y la lógica aplicada se puede apreciar en la figura 4.

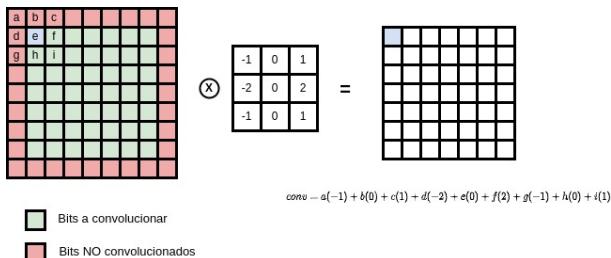


Fig. 4: Representación de convolución sobre una matriz

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 **paralelización**. 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.

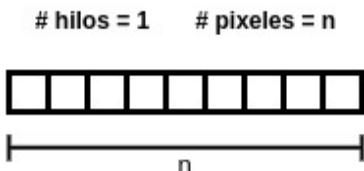


Fig. 5: Balanceo de carga en secuencial

Para esto se parte el número de píxeles a procesar en el número de hilos solicitados por el usuario. 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.

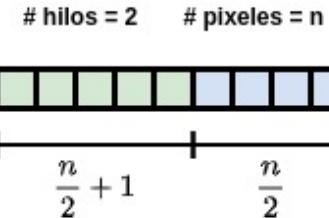


Fig. 6: Balanceo de carga en paralelo (2 hilos)

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.

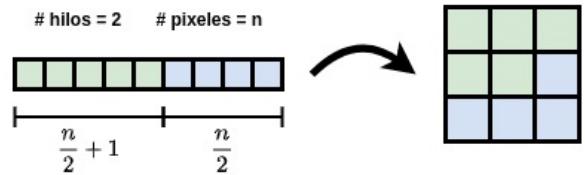


Fig. 7: Balanceo de carga en paralelo (Representación matricial)

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 utilizaron 1,2,4,8,16 y 32 hilos donde se calculó el tiempo con cada cantidad de hilos y su speed up respectivo.



Fig. 8: Imagen de manzanas 1280x720 px



Fig. 9: Imagen de astronauta 1920x1080 px



Fig. 10: Imagen de fondo 3840x2160 px

#### IV. RESULTADOS

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

##### A. Imagen 720p

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

#hilos	Tiempo (s)	SpeedUp
1	0.2408388333	1
2	0.2152116667	1.119078891
4	0.1970423333	1.222269495
8	0.1962956667	1.226918747
16	0.1975246667	1.219284849
32	0.1995108333	1.207146646

Estos valores se muestran en la figura 11 y 12, cuyos gráficos representan el tiempo y el speedup respectivamente.

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

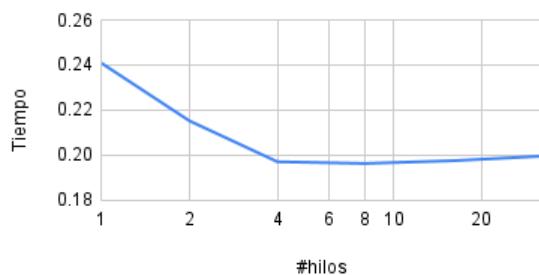


Fig. 11: Tiempo vs #Hilos 720p

##### SpeedUp vs. #hilos (720p)

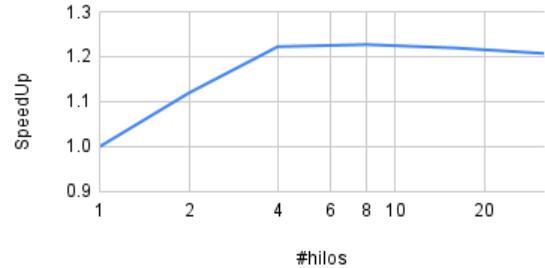


Fig. 12: SpeedUp vs #Hilos 720p

La imagen resultante al aplicar el filtro es la siguiente:

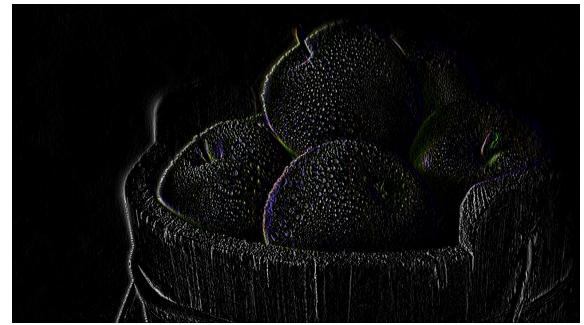


Fig. 13: Resultado del filtro imagen 720p

##### B. Imagen 1080p

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

#hilos	Tiempo (s)	SpeedUp
1	0.4564778333	1
2	0.397508	1.148348796
4	0.3710681667	1.230172444
8	0.3644926667	1.252364931
16	0.3658841667	1.247602042
32	0.3668415	1.244346219

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

##### Tiempo vs. #hilos (1080P)

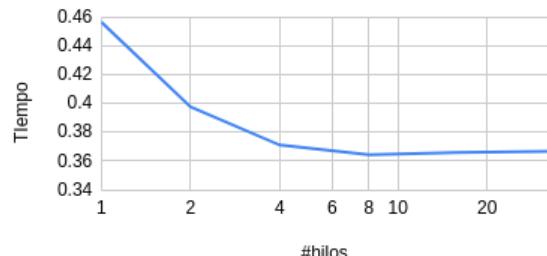


Fig. 14: Tiempo vs #Hilos 1080p



Fig. 15: SpeedUp vs #Hilos 1080p

La imagen resultante al aplicar el filtro es la siguiente:



Fig. 16: Resultado del filtro imagen 1080p

Donde se puede observar que la detección de filtros fue realizada correctamente.

### C. Imagen 4K

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

#hilos	Tiempo (s)	SpeedUp
1	2.127544	1
2	1.886911333	1.127527278
4	1.794608	1.18552018
8	1.793761	1.186079974
16	1.753907	1.21303125
32	1.778962	1.19594685

Estos valores se muestran en la figura 17 y 18, cuyos gráficos representan el tiempo y el speedup respectivamente.



Fig. 17: Tiempo vs #Hilos 4k



Fig. 18: SpeedUp vs #Hilos 4k

La imagen resultante al aplicar el filtro es la siguiente:

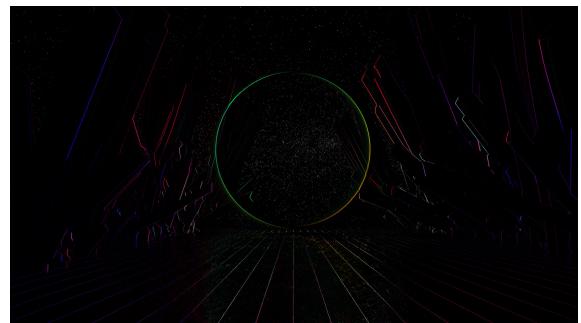


Fig. 19: Resultado del filtro imagen 4k

Mostrando que el algoritmo funciona correctamente en cada una de las resoluciones.

### D. 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:

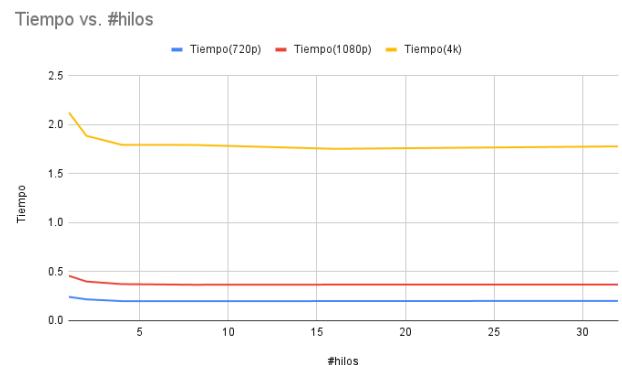


Fig. 20: Comparación tiempos de ejecución tres resoluciones

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

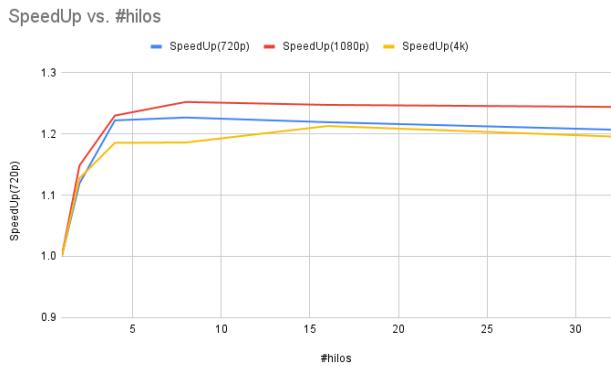


Fig. 21: Comparación SpeedUp tres resoluciones

## 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 el tiempo de ejecución disminuye hasta un punto de estabilidad.
- 4) En todas las imágenes 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 es considerable, teniendo que esperar entre 1.5 y 2.5 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, con una mejora de aproximadamente un 18%.

## REFERENCES

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