Efecto sobre una imagen

Gabriela García[1], **Ángel González** [2], **Felipe Rojas** [3]

Resumen

En este documento se muestra el desarrollo de la práctica 1, 2 y 3 referentes a la realización de un programa que aplica un filtro a un imagen (en distintos formatos de calidad) y que utiliza hilos para realizar los cálculos y operaciones sobre la imagen, veremos cómo afecta la aplicación del paralelismo en la ejecución del programa y los costos que pueda acarrear la implementación de este paradigma.

***Palabras Clave*— Computación paralela, hilos POSIX, openMP, CUDA, filtro sobre una imagen**

I. Introducción

A continuación se presentan experimentos, resultados y análisis de aplicar un filtro Sobel a una imagen en diferentes formatos de calidad: 720p, 1080p, 4K.
Se compara la ejecución secuencial (1 hilo) de este programa junto con la ejecución del programa en paralelo (2, 4, 8 y 16 hilos) para hilos POSIX - openMP y la ejecución secuencial (1 hilo - 1 bloque) junto con la ejecución en paralela (32, 64, 96, 128, 160 hilos - 10, 20, 40, 60 bloques) para hilos en CUDA, Este problema resulta ser de bastante interés debido a que en este incurren operaciones matriciales que a la larga, pueden resultar costosas computacionalmente.
Se busca comprender el comportamiento de aplicar un enfoque de paralelismo en el cálculo de la imagen final y si resulta tal ser mejor en términos de rendimiento y costos computacionales.

II. DISEÑO

Para la aplicación del filtro sobel se utilizan dos kernels (matrices) que están definidas para la detección de bordes, estas matrices en específico son:

$$\mathbf{G_x} = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * \mathbf{A} \quad \text{y} \quad \mathbf{G_y} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * \mathbf{A}$$

Fig. 1. Kernels utilizados para la convolución de bordes horizontales y verticales

Para la detección de bordes en horizontal se usa G_x y para la detección de bordes en vertical usamos G_y.
Estos kernels o máscaras, pasan por cada píxel de la imagen y desde allí se evidencia que el cálculo (multiplicaciones) se va a extender según el tamaño de la imagen original. Con una imagen muy grande, van a ser necesarios más cálculos con estos filtros.

Con este problema en mente, se puede introducir la paralelización, considerando que los hilos pueden calcular al tiempo distintas partes de la imagen y dividir así el trabajo.

Entonces se busca dividir el trabajo para que cada hilo realice solo una parte del cálculo de toda la imagen, logrando así que se utilicen los recursos disponibles en nuestra máquina y ver si es posible un rendimiento en cuanto al tiempo que toma procesar la imagen

Supongamos que tenemos dos hilos para realizar la aplicación del filtro y la imagen es representada en una matriz de tamaño n*n, entonces para asignar el trabajo lo que se hace es dividir la imagen verticalmente en este caso en dos (por los dos hilos) como lo muestra la imagen:

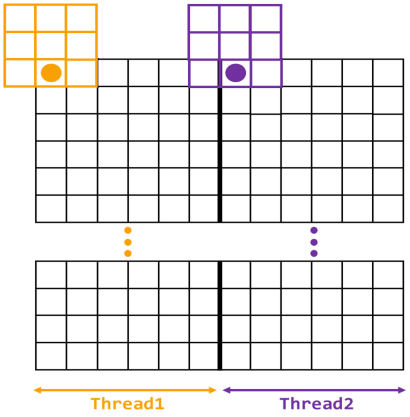

Fig. 2. División del trabajo en cada hilo para lograr la paralelización del programa. Ejemplo con dos hilos

Como se explicó anteriormente, los formatos de calidad que se manejan para la práctica son 720p 1080p y 4K. Estos se traducen en el tamaño de la matriz, en estos casos las imágenes van a ser más largas en vez de altas y es por ello que se hace la separación del cálculo verticalmente

Generalizando, se obtiene una imagen de tamaño $w \times h$ (en este caso $w > h$) y dividir el trabajo en m hilos para lograr una buena paralelización:

[1] Gabriela García: ggarciaro@unal.edu.co, Estudiante de pregrado - Ingeniería de Sistemas y Computación.
[2] Ángel González: angegonzalez@unal.edu.co, Estudiante de pregrado - Ingeniería de Sistemas y Computación.
[3] Felipe Rojas: jrojasce@unal..edu.co, Estudiante de pregrado - Ingeniería de Sistemas y Computación.

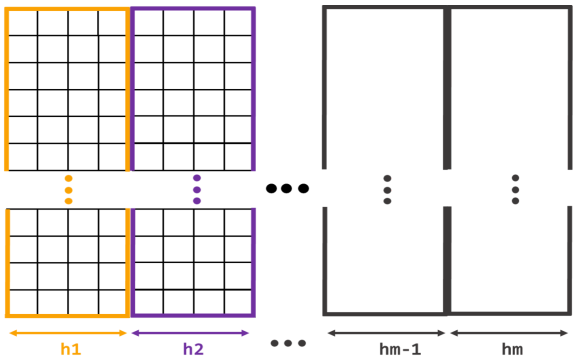

Fig. 3. División del trabajo en cada hilo para lograr la paralelización del programa. Ejemplo con m hilos

Finalmente, cada hilo aplica la máscara correspondiente para *x* y para *y* sobre su parte de la imagen y coloca el resultado en la imagen de salida, produciendo con éxito el filtro de detección de bordes en la imagen. Es necesario recordar que esta distribución del trabajo se da para los métodos POSIX y openMP ya que el número de hilos (2,4,8,16) que se lanzan es significativamente menor al número de columnas que obtiene la imagen.

La distribución de carga para el método CUDA debe ser diferente ya que el número de hilos lanzados puede llegar a superar el número de columnas en la imagen, produciendo así que cierta cantidad de hilos quede desperdiciada y no realice ninguna operación. Por lo anterior, es necesario realizar una división considerando las filas y columnas en la imagen, para esto, se utiliza un tipo de dato Dim3 que permitirá realizar la abstracción de memoria de las coordenadas (x,y,z) con mayor facilidad para el lanzamiento de bloques e hilos y distribuir el trabajo según el número de hilos totales, el ancho y alto de la imagen y el número de hilo actual tanto en la coordenada x como la y.

III. EXPERIMENTOS

Los experimentos que se realizaron consisten en la toma de 5 tiempos de ejecución de dos programas paralelizando con hilos POSIX, openMP y CUDA, para una combinación de imágenes en resoluciones de 720p, 1080p, 4K y un número de hilos entre 1, 2, 4, 8 y 16 en POSIX - openMP y un número de bloques entre 1, 20, 40, 60 con 1, 32, 64, 96, 128, 169 hilos por bloque. El medio utilizado para estos experimentos es un computador con procesador Intel Core I7 de 4 núcleos físicos y 8 virtuales en POSIX - openMP y una tarjeta gráfica Tesla T4 en CUDA.

También es importante resaltar que el uso de la librería STB Image para la lectura y escritura de imágenes no se encuentra considerado en la paralelización y por tanto es necesario realizar dos mediciones distintas. La primera medición considera todo el programa y es realizada con el comando *time* del sistema operativo Linux. La segunda medición se realiza con una variable dentro del programa que considera únicamente el intervalo de tiempo entre la creación de hilos y la terminación de los mismos (sección paralela). Para el método CUDA se realiza la medición previamente mencionada.

Como complemento a los experimentos, se realizó el cálculo del tiempo promedio que utiliza la librería STB Image y gestiones del sistema operativo durante la ejecución de los programas, se utilizaron todos los datos de tiempo recogido en ambos métodos para el promedio.

Por último, el objetivo de estos experimentos es determinar si una actividad como el aplicado de filtros a imágenes es meritoria de paralelización, los tiempos reales que utiliza la librería STB Image, las diferencias en rendimiento de los distintos métodos y realizar comprobantes sobre el procesador en el que se realizaron.

IV. RESULTADOS POSIX

Se presentan 9 gráficas correspondientes a los resultados de tiempo promedio y Speed up calculada contra número de hilos para cada resolución de imagen en la toma de tiempo utilizando la variable del programa. Para la toma de tiempo con el comando *time* solo se presentan los resultados de tiempo promedio contra número de hilos para cada resolución de imagen.

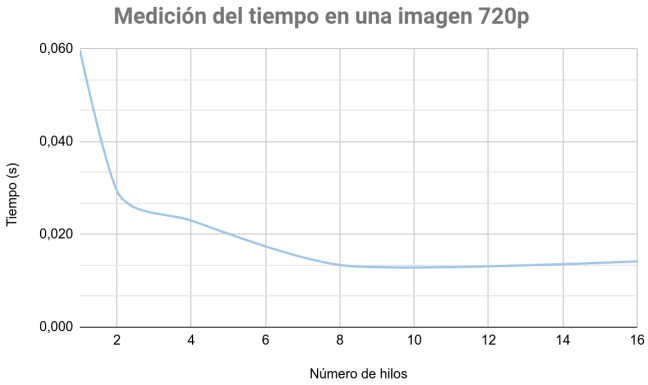

Fig. 4. Medición del tiempo contra número de hilos en una imagen 720p.

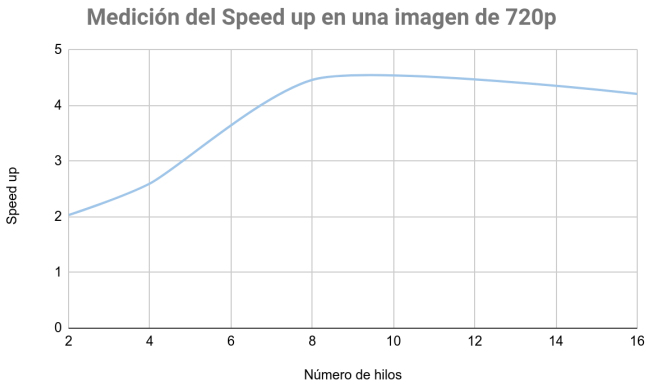

Fig. 5. Medición del Speed up contra número de hilos en una imagen 720p.

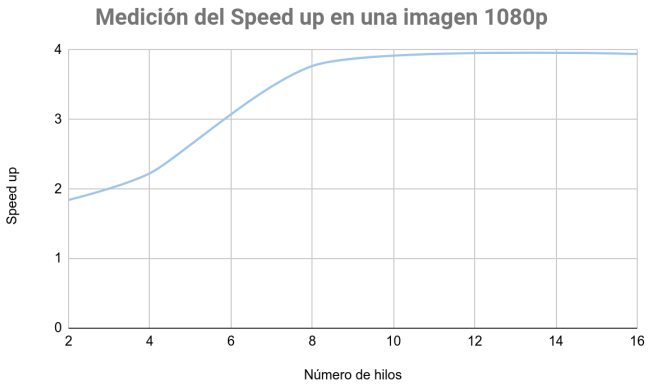

Fig. 8. Medición del Speed up contra número de hilos en una imagen 1080p.

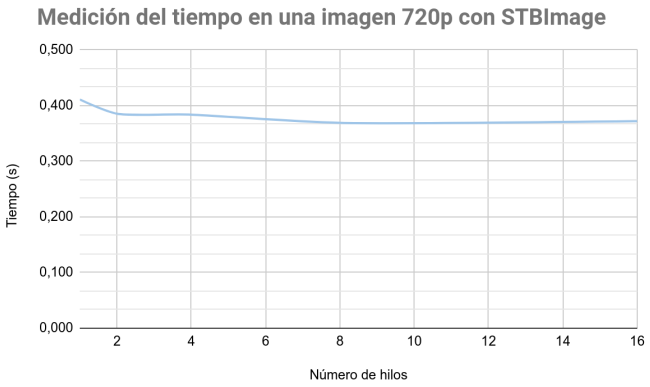

Fig. 6. Medición del tiempo contra número de hilos en una imagen 720p considerando la librería STB Image.

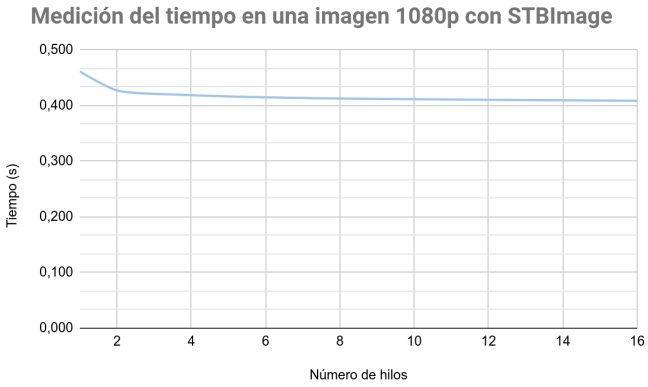

Fig. 9. Medición del tiempo contra número de hilos en una imagen 1080p considerando la librería STB Image.

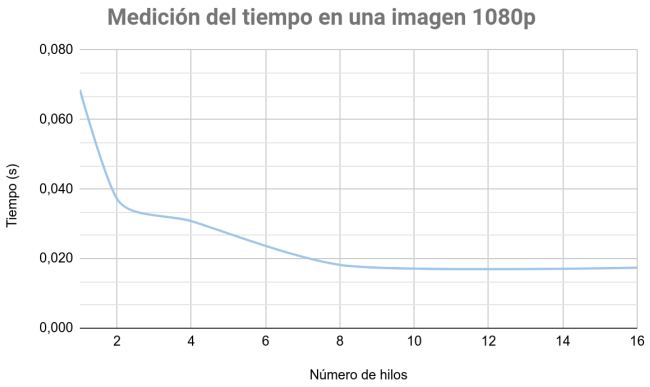

Fig. 7. Medición del tiempo contra número de hilos en una imagen 1080p.

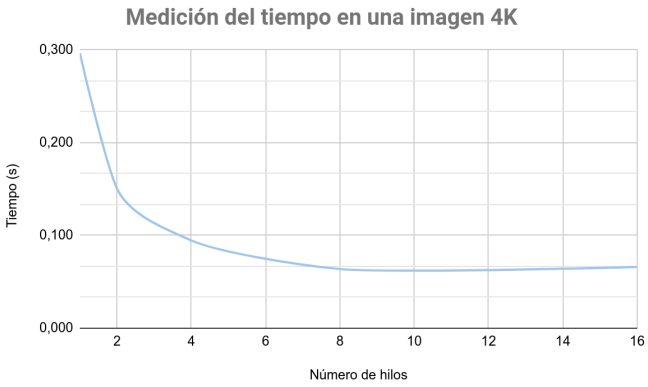

Fig. 10. Medición del tiempo contra número de hilos en una imagen 4K.

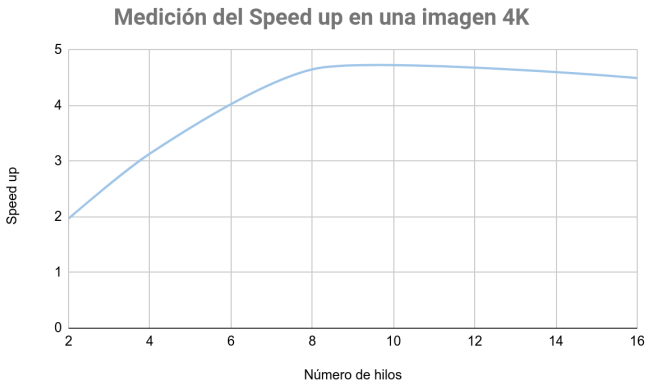

Fig. 11. Medición del Speed up contra número de hilos en una imagen 4K.

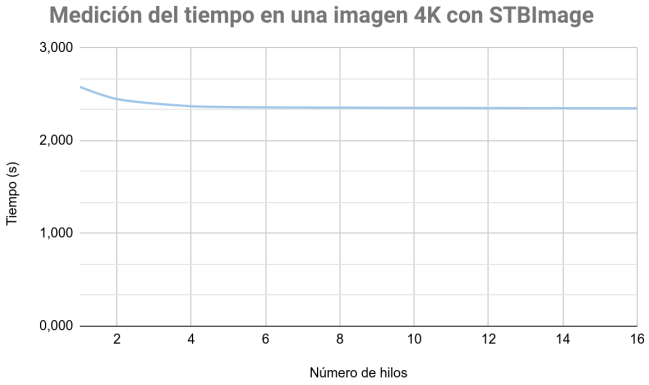

Fig. 12. Medición del tiempo contra número de hilos en una imagen 4K considerando la librería STB Image.

Adicional a esto, se presenta una tabla con los tiempos promedios de gestión del sistema operativo y lectura, escritura utilizando STB Image para cada resolución de imagen.

Resolución de la imagen	Tiempo promedio (s)
720p	0,356
1080p	0,391
4K	2,287

V. Resultados OpenMP

Se presentan 9 gráficas correspondientes a los resultados de tiempo promedio y Speed up calculada contra número de hilos para cada resolución de imagen en la toma de tiempo utilizando la variable del programa. Para la toma de tiempo con el comando *time* solo se presentan los resultados de tiempo promedio contra número de hilos para cada resolución de imagen.

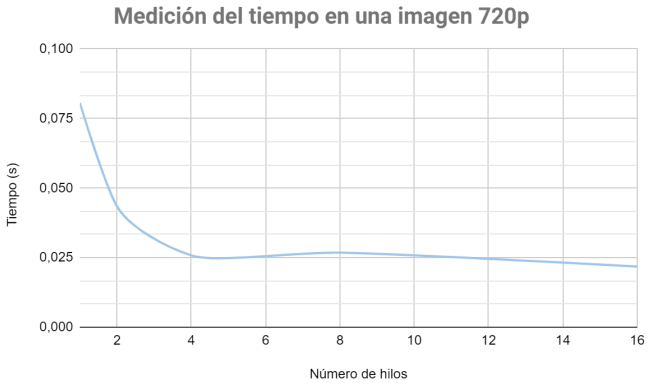

Fig. 13. Medición del tiempo contra número de hilos en una imagen 720p.

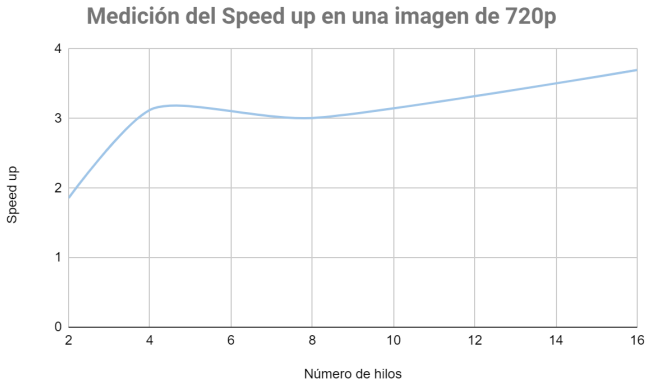

Fig. 14. Medición del Speed up contra número de hilos en una imagen 720p.

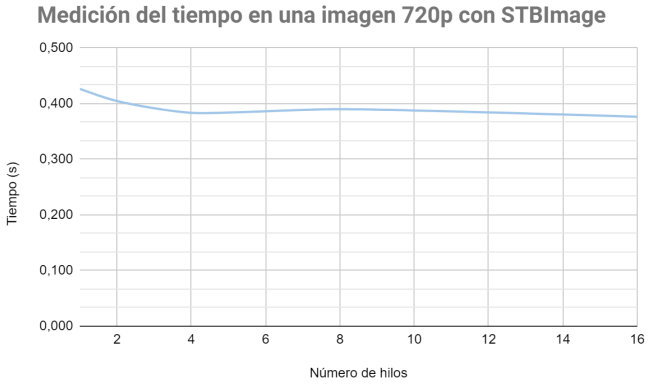

Fig. 15. Medición del tiempo contra número de hilos en una imagen 720p considerando la librería STB Image.

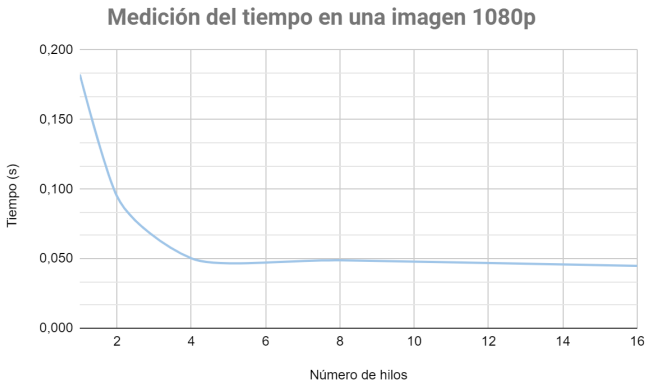

Fig. 16. Medición del tiempo contra número de hilos en una imagen 1080p.

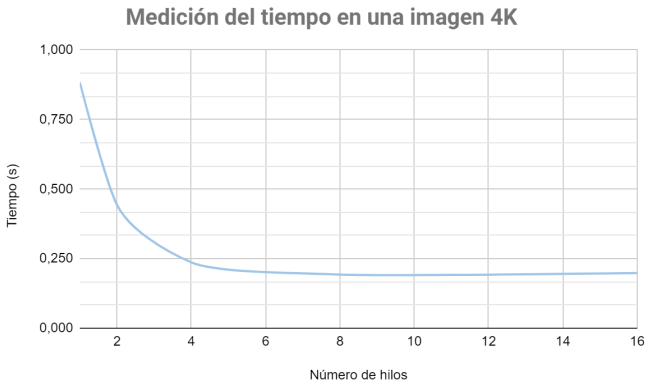

Fig. 19. Medición del tiempo contra número de hilos en una imagen 4K.

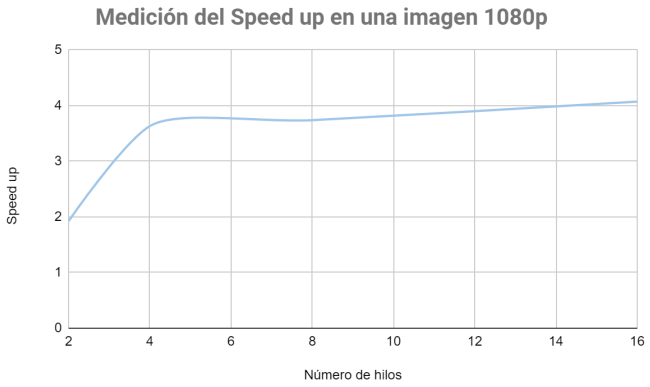

Fig. 17. Medición del Speed up contra número de hilos en una imagen 1080p.

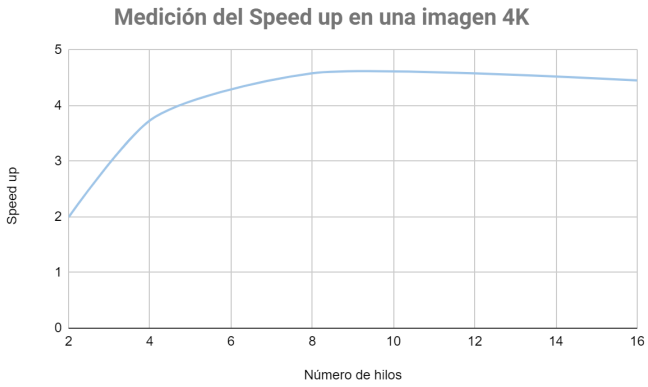

Fig. 20. Medición del Speed up contra número de hilos en una imagen 4K.

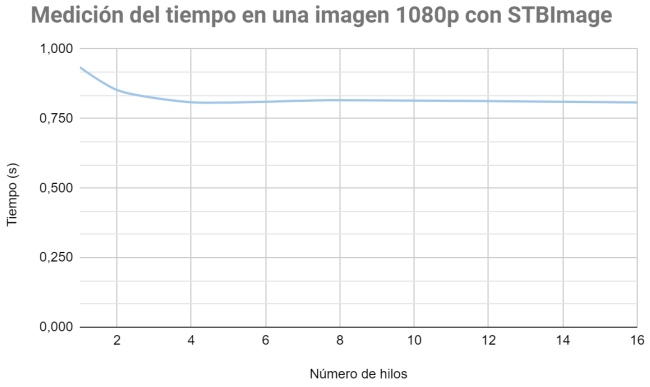

Fig. 18. Medición del tiempo contra número de hilos en una imagen 1080p considerando la librería STB Image.

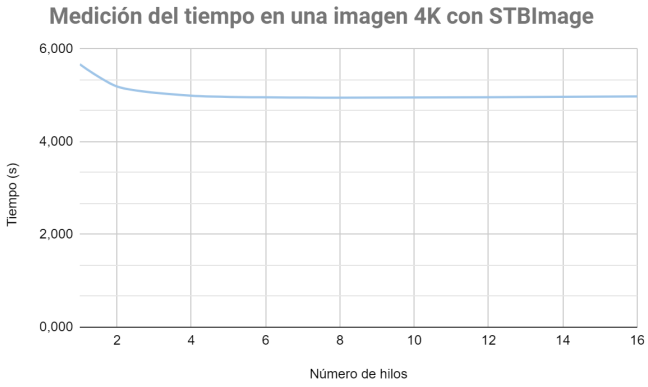

Fig. 21. Medición del tiempo contra número de hilos en una imagen 4K considerando la librería STB Image.

Adicional a esto, se presenta una tabla con los tiempos promedios de gestión del sistema operativo y lectura, escritura utilizando STB Image para cada resolución de imagen.

Resolución de la imagen	Tiempo promedio (s)

720p	0,357
1080p	0,760
4K	4,764

Por último, se presentan las comparaciones entre ambos programas.

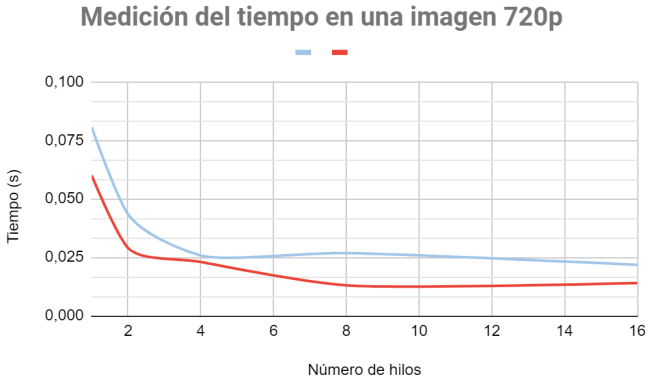

Fig. 22. Medición del tiempo contra número de hilos en una imagen 720p para Posix (azul) y OpenMP (rojo).

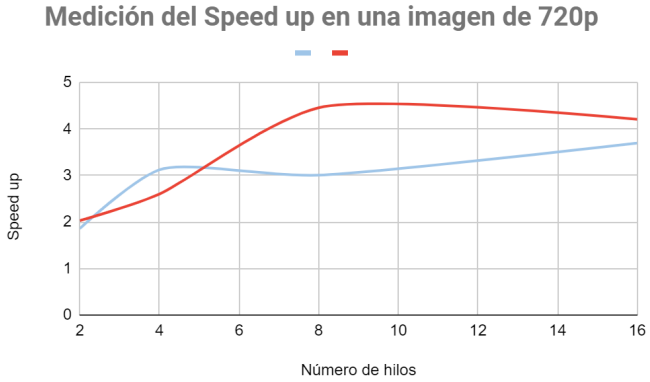

Fig. 23. Medición del Speed up contra número de hilos en una imagen 720p para Posix (azul) y OpenMP (rojo).

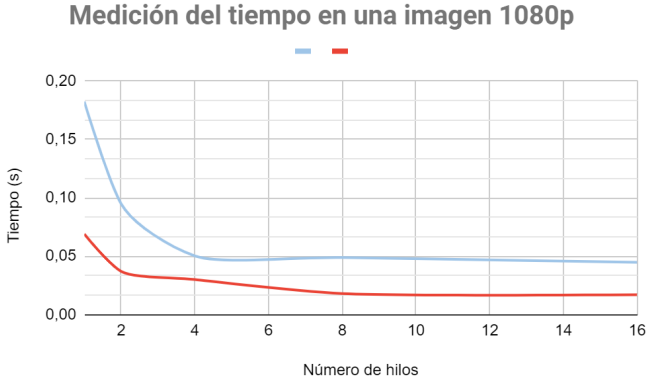

Fig. 24. Medición del tiempo contra número de hilos en una imagen 1080p para Posix (azul) y OpenMP (rojo).

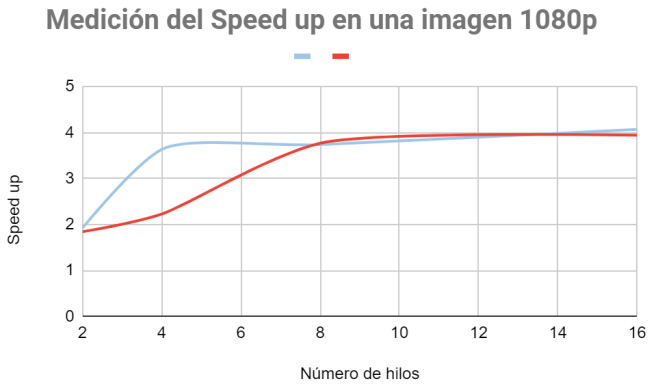

Fig. 25. Medición del Speed up contra número de hilos en una imagen 1080p para Posix (azul) y OpenMP (rojo).

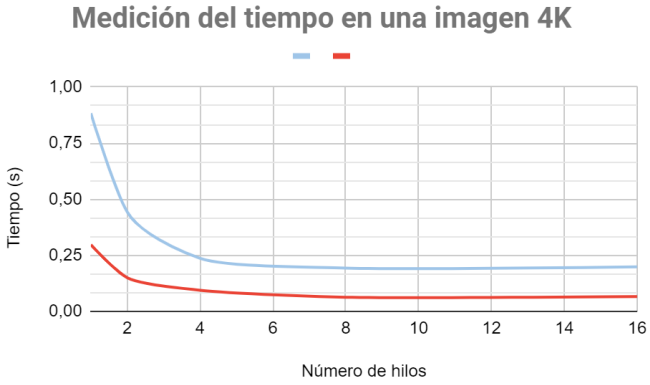

Fig. 26. Medición del tiempo contra número de hilos en una imagen 4K para Posix (azul) y OpenMP (rojo).

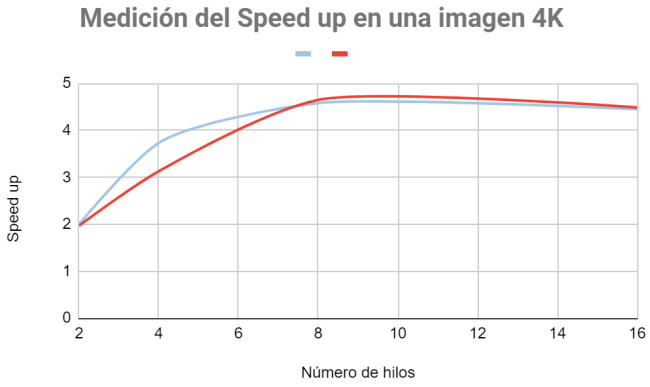

Fig. 27. Medición del Speed up contra número de hilos en una imagen 4K para Posix (azul) y OpenMP (rojo).

VI. RESULTADOS CUDA

Se presentan 24 gráficas correspondientes a los resultados de tiempo promedio y Speed up calculada contra número de hilos por bloque y un número de bloques fijo para cada resolución de imagen en la toma de tiempo utilizando la variable del programa.

Fig. 28. Medición del tiempo contra número de hilos en una imagen 720p y 1 bloque.

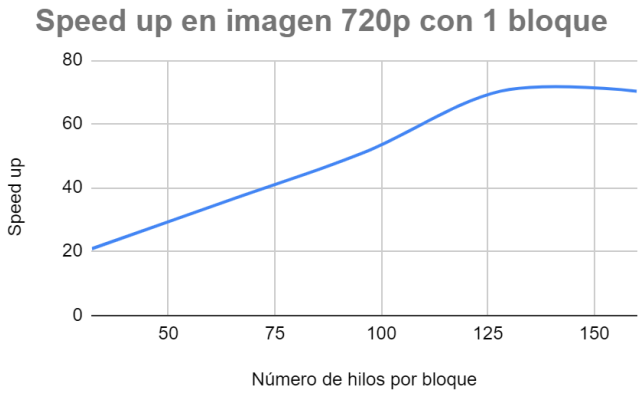

Fig. 29. Medición del Speed up contra número de hilos en una imagen 720p y 1 bloque.

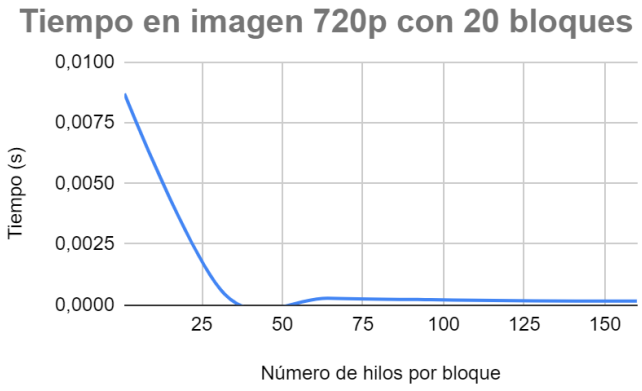

Fig. 30. Medición del tiempo contra número de hilos en una imagen 720p y 20 bloques.

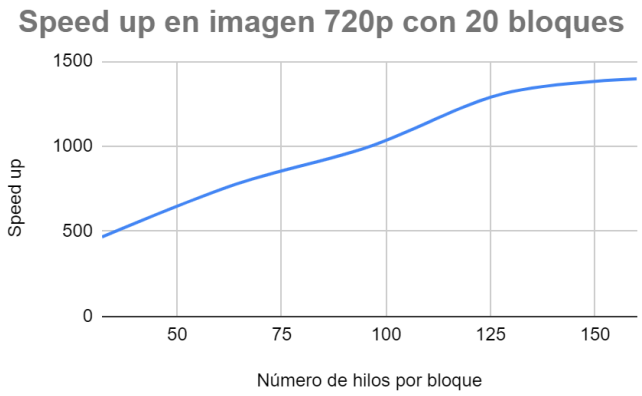

Fig. 31. Medición del Speed up contra número de hilos en una imagen 720p y 20 bloques.

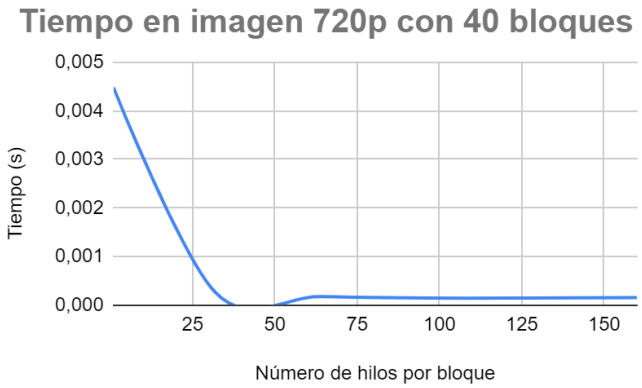

Fig. 32. Medición del tiempo contra número de hilos en una imagen 720p y 40 bloques.

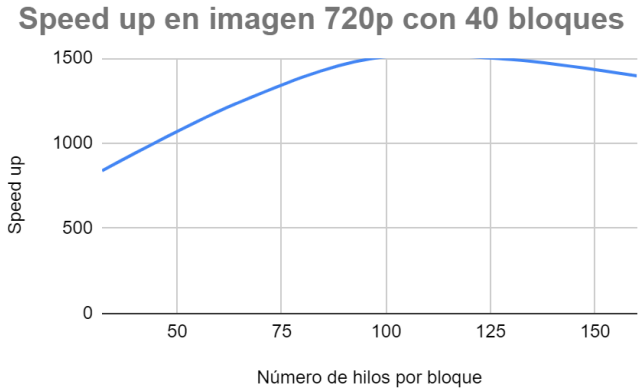

Fig. 33. Medición del Speed up contra número de hilos en una imagen 720p y 40 bloques.

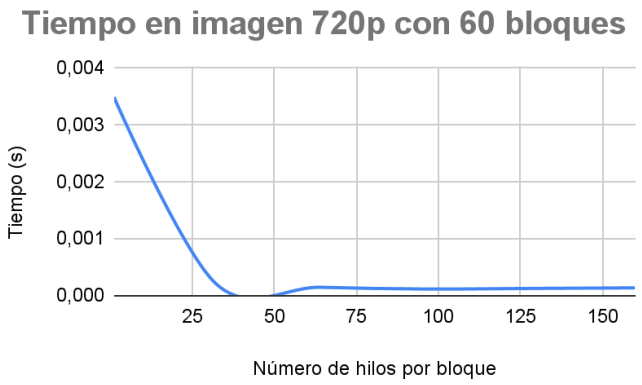

Fig. 34. Medición del tiempo contra número de hilos en una imagen 720p y 60 bloque.

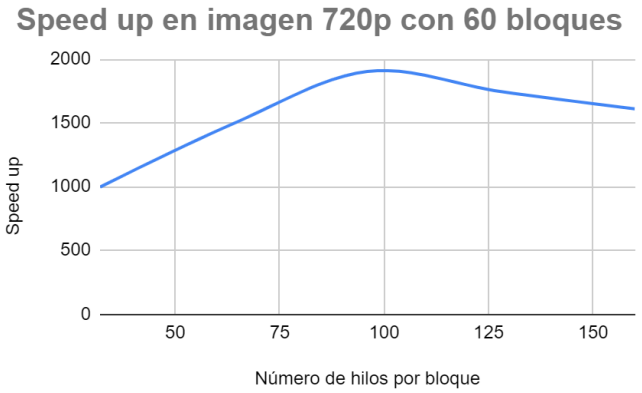

Fig. 35. Medición del Speed up contra número de hilos en una imagen 720p y 60 bloques.

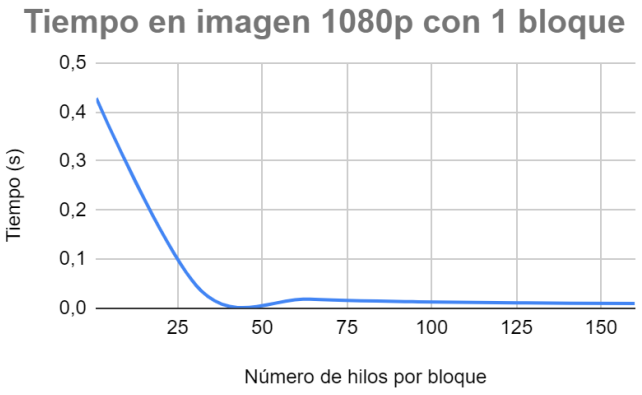

Fig. 36. Medición del tiempo contra número de hilos en una imagen 1080p y 1 bloque.

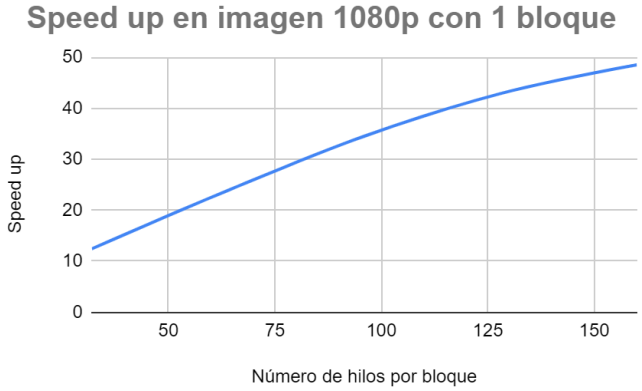

Fig. 37. Medición del Speed up contra número de hilos en una imagen 1080p y 1 bloque.

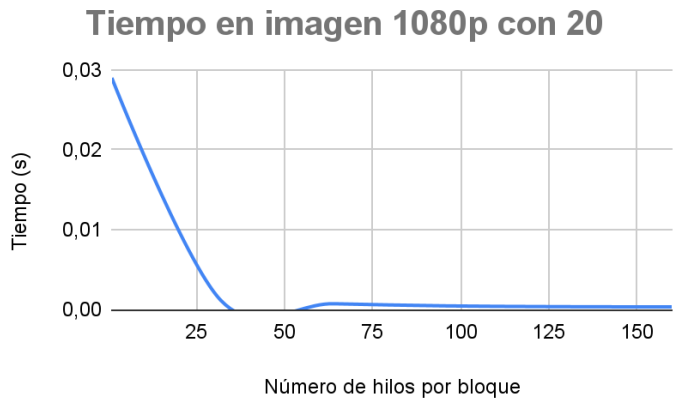

Fig. 38. Medición del tiempo contra número de hilos en una imagen 1080p y 20 bloques.

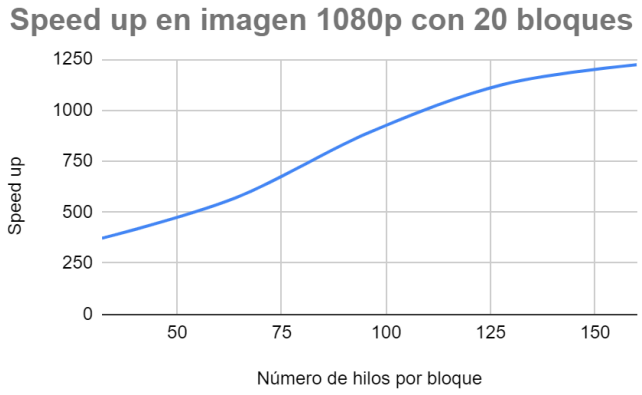

Fig. 39. Medición del Speed up contra número de hilos en una imagen 1080p y 20 bloques.

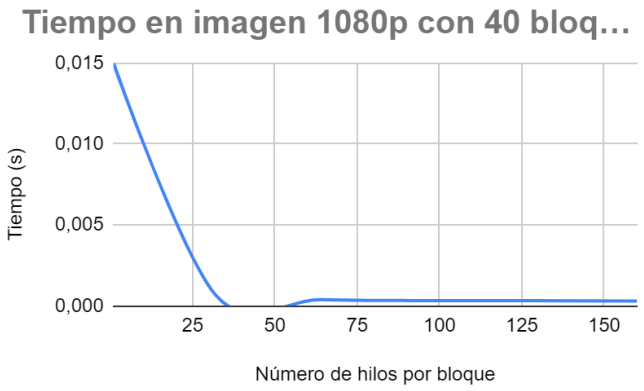

Fig. 40. Medición del tiempo contra número de hilos en una imagen 1080p y 40 bloques.

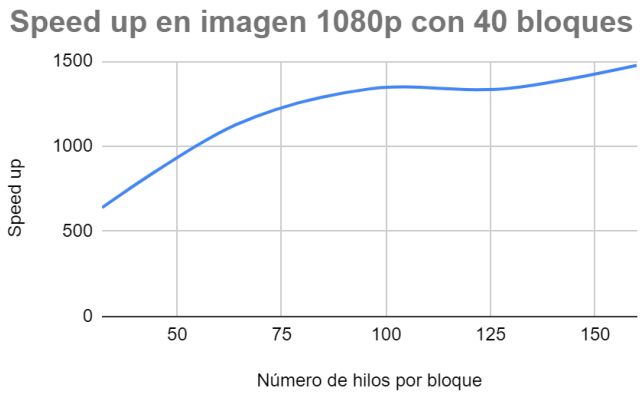

Fig. 41. Medición del Speed up contra número de hilos en una imagen 1080p y 40 bloques.

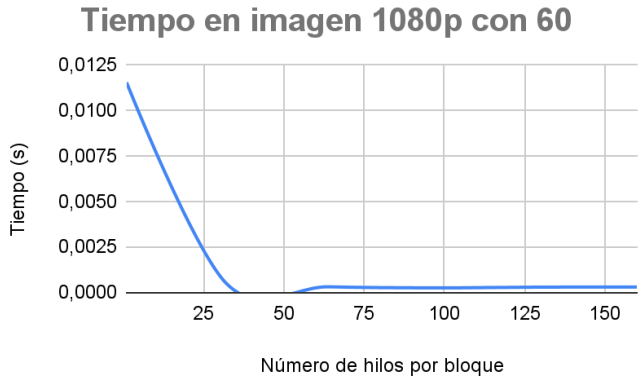

Fig. 42. Medición del tiempo contra número de hilos en una imagen 1080p y 60 bloques.

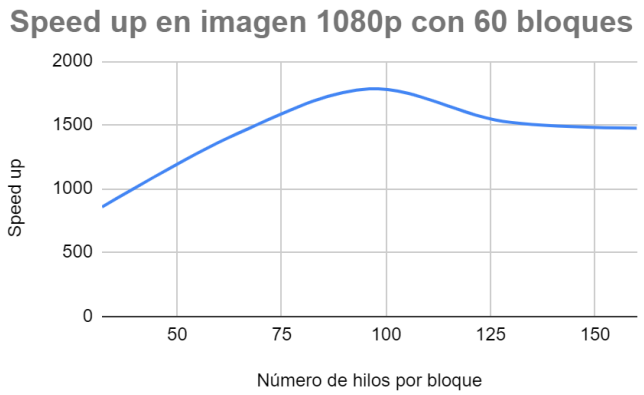

Fig. 43. Medición del Speed up contra número de hilos en una imagen 1080p y 60 bloques.

Fig. 44. Medición del tiempo contra número de hilos en una imagen 4K y 1 bloque.

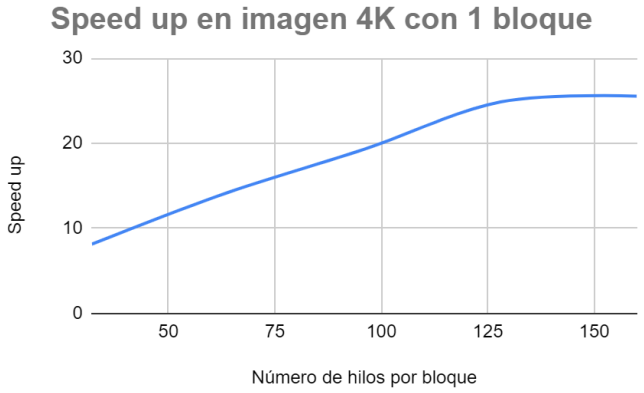

Fig. 45. Medición del Speed up contra número de hilos en una imagen 4K y 1 bloque.

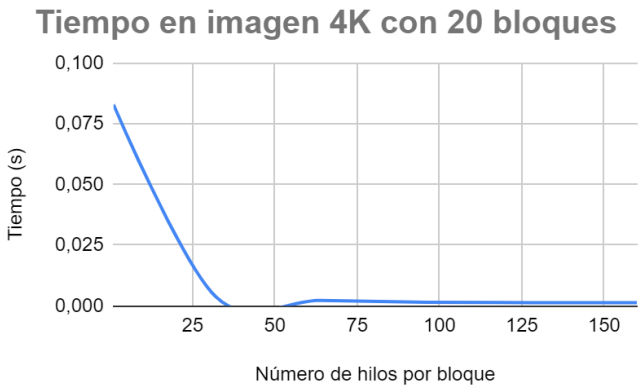

Fig. 46. Medición del tiempo contra número de hilos en una imagen 4K y 20 bloques.

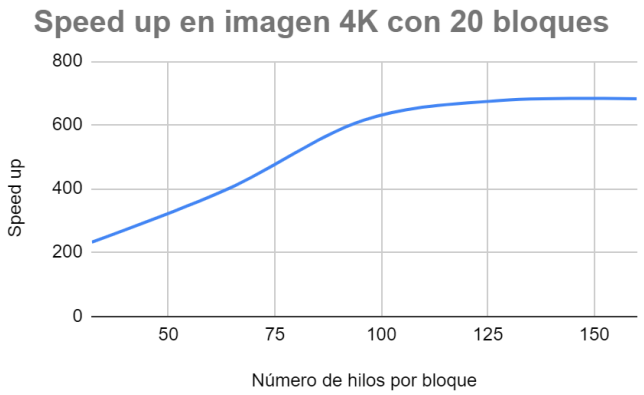

Fig. 47. Medición del Speed up contra número de hilos en una imagen 4K y 20 bloques.

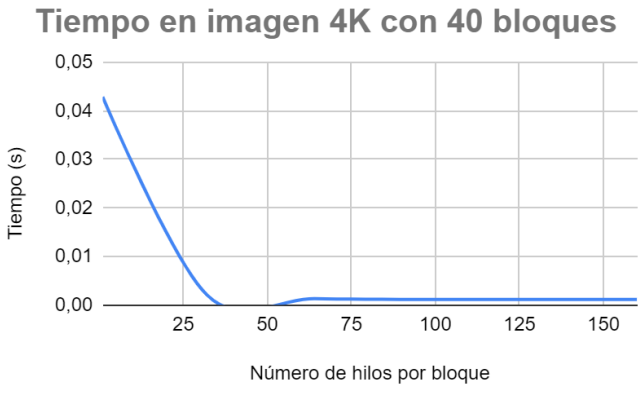

Fig. 48. Medición del tiempo contra número de hilos en una imagen 4K y 40 bloques.

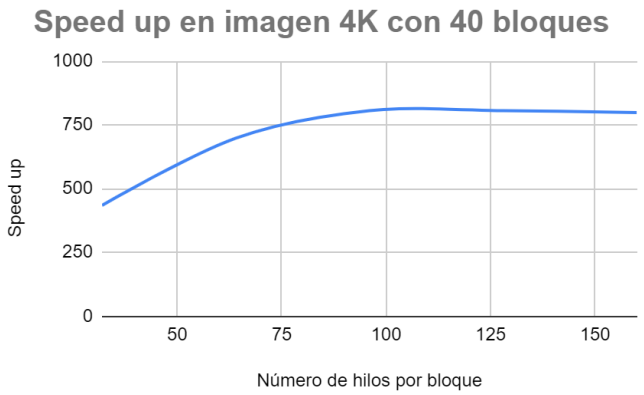

Fig. 49. Medición del Speed up contra número de hilos en una imagen 4K y 40 bloques.

Fig. 50. Medición del tiempo contra número de hilos en una imagen 4K y 60 bloques.

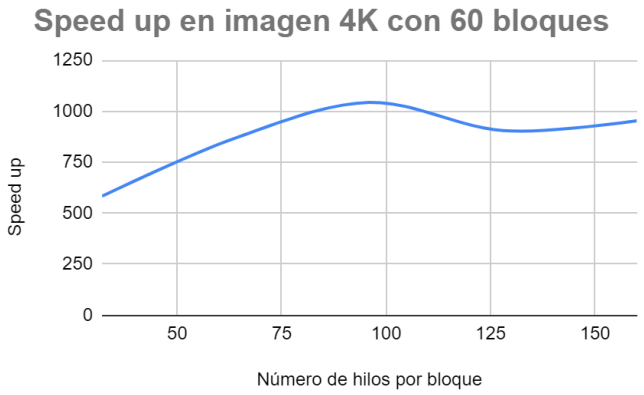

Fig. 51. Medición del Speed up contra número de hilos en una imagen 4K y 60 bloques.

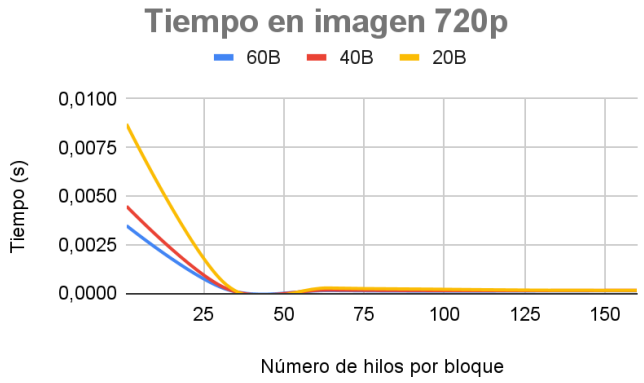

Fig. 52. Medición del tiempo contra número de hilos en una imagen 720p..

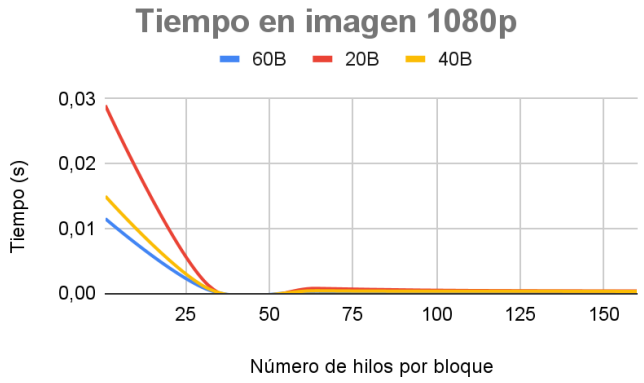

Fig. 53. Medición del tiempo contra número de hilos en una imagen 1080p.

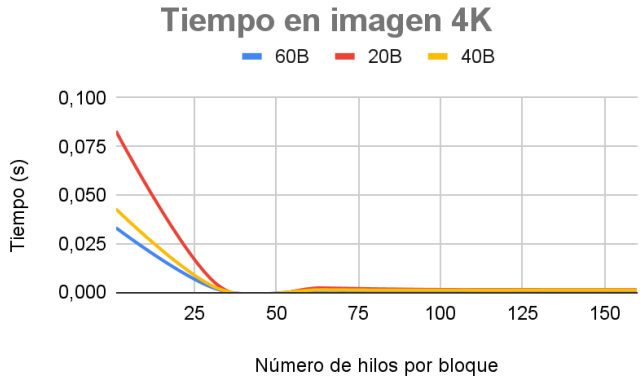

Fig. 54. Medición del tiempo contra número de hilos en una imagen 4K.

VII. Conclusiones

De los anteriores resultados, se pueden realizar estas conclusiones respectivas a la paralelización, librerías externas y procesador utilizado:

- Considerando la paralelización en la aplicación del filtro a las distintas resoluciones en imágenes, se concluye que esta es una actividad paralelizable y meritoria de lo mismo ya que se puede evidenciar un recorte amplio en los tiempos de ejecución y una aceleración al programa alta considerando el número de núcleos disponibles.

- Los tiempos que utiliza la librería STB Image para la lectura y escritura de imágenes son considerablemente superiores a los tiempos de ejecución en la aplicación del filtro, y por esto no es posible evidenciar un recorte y aceleración congruentes a los anteriores experimentos en las gráficas donde se tienen en cuenta los tiempos totales de ejecución.

- El procesador demostró ser congruente a sus núcleos ya que el Speed Up máximo obtenido es menor o igual al número de estos.

- En cuanto a la comparación de tiempos entre el programa Posix y OpenMP, se puede observar que el programa Posix es más rápido y conforme la imagen aumenta su resolución y el programa su complejidad es más notorio el aumento de tiempos en el programa de OpenMP. Este aumento de tiempo puede ser debido a los tiempos de gestión adicionales que utiliza la librería.

- En cuanto a la comparación de Speed ups entre el programa Posix y OpenMP, se pueden observar comportamientos similares en las imágenes de 1080p y 4k en donde se llega a una aceleración máxima similar. Las diferencias notorias se dan en el comportamiento de 4 núcleos donde OpenMP parece acelerar más los programas y aprovechar mejor los recursos.

- En cuanto a la imagen de 720p, se observa una aceleración considerablemente mayor en el caso de hilos Posix. Se puede relacionar este comportamiento a la facilidad con la que se realiza esta tarea y los tiempos de gestión adicionales que infieren en la librería OpenMP.

- En cuanto a CUDA, es posible evidenciar que las tarjetas gráficas son el mejor método de ejecución para el procesamiento de imágenes. La potencia y cantidad de unidades de procesamiento (núcleos) con la que cuentan estas tarjetas hacen que los Speed ups encontrados sean sustancialmente mayores con respecto a los de CPU.

- En cuanto a CUDA, para todas las imágenes se pudo evidenciar un comportamiento muy similar en donde fue necesario lanzar un número de hilos y bloques cercano al doble (40 multiprocesadores y 64 hilos en la tarjeta) para obtener los mejores tiempos de

procesamiento y por tanto las mayores aceleraciones del proceso con respecto al secuencial (1 hilo y 1 bloque).

Referencias

[1] Operador Sobel - Wikipedia, la enciclopedia libre", Es.wikipedia.org, 2022. [Online]. Available: https://es.wikipedia.org/wiki/Operador_Sobel [Accessed: 24- Apr- 2022].

[2] L., & L. (2022). CUDA-Sobel-Filter/imageLoader.cpp at master lukas783/CUDA-Sobel-Filter. GitHub. https://github.com/lukas783/CUDA-Sobel-Filter/blob/master/imageLoader.cpp