Practica 1,2,3
Redimensión de Videos con Resolución 1080p a 360p

Hernández C. Sebastián, Mendez M. Luis Fernando

Computación Paralela y Distribuida
Universidad Nacional de Colombia Sede Bogotá
Facultad de Ingeniería - Ingeniería de Sistemas y Computación
Bogotá 2023-II

El siguiente trabajo práctico se centrará en la comparación del rendimiento de un algoritmo o método para la reducción de resolución de un video para diferentes casos. Este trabajo es una exploración de las técnicas de paralelización dentro de la computación y su impacto en ciertos procesos paralelizables, en este caso en el procesamiento de videos.

I. INTRODUCCIÓN

El paralelismo en el ámbito de la computación es un concepto fundamental que ha revolucionado la forma en que se procesa la información y se ejecutan tareas. Se refiere a la capacidad de realizar múltiples operaciones al mismo tiempo (en el ciclo de reloj), ya sea en hardware o software, con el objetivo de aumentar la eficiencia y el rendimiento de los sistemas informáticos. En un mundo cada vez más impulsado por la demanda de mayor velocidad y capacidad de procesamiento, el paralelismo se ha convertido en un componente esencial para abordar tareas complejas, desde el procesamiento de grandes cantidades de datos hasta la ejecución de aplicaciones de alta demanda computacional, como la inteligencia artificial y el cómputo científico.

El objetivo de esta práctica se basa en la implementación y evaluación de una metodología para reducir la resolución de videos en color de alta definición (1080p) a una resolución más baja (360p). Este proceso es esencial en muchas aplicaciones prácticas, como la transmisión de videos en línea, donde la reducción de la resolución puede ayudar a ahorrar ancho de banda y mejorar la experiencia del usuario.

El desarrollo de la práctica contiene los siguientes elementos:

1. *Programa secuencial:* Esta es la forma más básica y directa de ejecución, donde el algoritmo se ejecuta secuencialmente.

2. *Programa paralelo:* Aquí, el algoritmo se paraleliza mediante diferentes tecnologias:
    a. OpenMP: Una biblioteca para programación multiproceso en C/C++. Se probarán diferentes números de hilos (2, 4, 8 y 16) para evaluar su impacto en el rendimiento.
    b. CUDA C/C++: Una plataforma de cómputo paralelo y un modelo de programación desarrollado por NVIDIA. Se probarán con diferente número de bloques e hilos.
    c. MPI: Una biblioteca estándar para la programación multiproceso en C/C++ que permite la comunicación entre procesadores en un clúster. Se probarán con distinta cantidad de procesos en máquinas virtuales en la nube.

II. CONCEPTOS

A continuación se describe de forma breve algunos conceptos o tecnologías empleadas para el diseño del programa.

- ***Computación Secuencial:***

Es el proceso de ejecución de instrucciones en una computadora de forma lineal, una tras otra, sin la utilización de múltiples hilos o procesos concurrentes. Se trata de un flujo de trabajo donde las tareas se realizan de manera secuencial.

- ***Computación Paralela:***

Se refiere a la capacidad de ejecutar múltiples tareas de manera simultánea en una computadora, lo que mejora significativamente el rendimiento y la velocidad de procesamiento.

- ***OpenMP:***

OpenMP es una API (Interfaz de Programación de Aplicaciones) que permite la programación paralela en lenguajes como C++.

- ***CUDA C/C++:***

Es una plataforma de cómputo paralelo y un modelo de programación desarrollado por NVIDIA para aprovechar el poder de las Unidades de Procesamiento Gráfico (GPU) en tareas de cómputo de propósito general. CUDA permite a los desarrolladores escribir código en lenguajes como C y C++ que puede ejecutarse en las GPU de NVIDIA para acelerar una amplia gama de aplicaciones, desde simulaciones científicas hasta aprendizaje profundo.

- ***OpenMPI:***

Es una biblioteca de paso de mensajes de código abierto diseñada para facilitar la creación de aplicaciones paralelas y

distribuidas. Permite la comunicación eficiente entre procesos en sistemas de memoria distribuida, como clústeres de computadoras. OpenMPI se utiliza comúnmente en la programación paralela y en la ejecución de aplicaciones en entornos de computación de alto rendimiento, ofreciendo un conjunto de herramientas y funciones que facilitan la coordinación y el intercambio de datos entre nodos de manera escalable y eficiente.

- ***Lenguaje de programación - C++:***

C++ es un lenguaje de programación compilado de alto nivel ampliamente utilizado en el desarrollo de software.

- ***Procesamiento de Video - OpenCV:***

OpenCV es una biblioteca de código abierto que se utiliza para el procesamiento de imágenes y video. Proporciona herramientas y funciones para realizar diversas operaciones en imágenes y secuencias de video.

III. Diseño

El programa se desarrolló usando como lenguaje de programación C++ para la parte lógica. El procesamiento de imagen: Lectura y escritura del video, se efectúan mediante la biblioteca OpenCV.
El procesamiento del programa en paralelo se realiza con OpenMP, CUDA o OpenMPI.

Captura y escritura de imagen:

En primer lugar, el programa toma como entrada el nombre del archivo de video con formato mp4. Utilizando la clase VideoCapture de OpenCV se abre el archivo de entrada y se obtienen caracteristicas como frames, fps, etc.
Luego, se configura un nuevo video de salida usando la clase VideoWriter especificando el codec de video, la velocidad de fotogramas y las dimensiones deseadas. Esto cumpliendo las condiciones donde el video de salida debe tener una resolución de 360p.
El metodo read de la clase VideoCapture nos va a permitir recorrer frame por frame del video de entrada para realizar su respectivo procesamiento.
Cuando capturamos un frame o fotograma mediante la libreria openCV, el frame representa una matriz donde cada casilla contiene una tupla de tres valores (una para el rojo, verde y azul) que simbolizan la tonalidad de color basada en el sistema RGB.

Algoritmo de procesamiento de imagen:

Durante el **procesamiento**, cada fotograma se reduce en tamaño, promediando los valores de píxeles en grupos de píxeles adyacentes según un factor de reducción predefinido, en este caso 3. La proporción entre 1080p y 360p es 1 a 3 en ancho y 1 a 3 en alto, por tanto vamos a tener que una submatrix de 3x3 en la matriz de 1080x1920 equivale a una casilla en la matriz de 360x640. Este promedio se realiza de forma independiente para cada color o valor de la tupla.
Es importante aclarar que al promedio se le aplica la función piso, lo cual en C++ esta por defecto cuando se opera con enteros.

A continuación se presenta un ejemplo donde se obtiene el valor para el color azul de un pixel para el video de salida.

100	150	200
80	75	68
68	75	65

Tabla 1. Submatriz del fotograma del video de entrada para el color azul.

100+150+200+80+75+68+68+75+65 = 881/(3*3) = ⌊97.8⌋ = **97**

97

Tabla 2. Casilla de respectiva para el azul del fotograma del video de salida.

El procesamiento del frame varía en su ejecución según si se realiza en secuencial o paralelo, es decir, el algoritmo es el mismo, sin embargo, el orden de las tareas se realiza de forma distinta.

Para el **programa secuencial** el procesamiento se realiza pixel por pixel de forma secuencial.

Procesamiento en paralelo (OpenMP):

Por otro lado, el **programa paralelo** aprovecha la potencia de cómputo de múltiples hilos. Durante el procesamiento de cada fotograma, se crea un equipo de hilos mediante OpenMP. Cada hilo se encarga de una porción de la imagen, dividiendo las filas del fotograma de manera equitativa. Esto permite que múltiples secciones de la imagen se procesen simultáneamente, acelerando significativamente la velocidad de procesamiento. Dentro de cada hilo, las operaciones de reducción de muestreo se realizan de forma independiente en su respectiva sección de la imagen, evitando conflictos de datos. La utilización de la cláusula "schedule(dynamic)" garantiza una asignación eficiente de tareas a los hilos, lo que significa que los hilos pueden recibir nuevas tareas tan pronto como hayan completado sus tareas anteriores, lo que minimiza la inactividad y maximiza la utilización de los recursos del procesador. La cláusula "nowait" permite que los hilos continúen sin bloquearse mutuamente, lo que mejora la eficiencia general del procesamiento paralelo

Finalmente, los fotogramas procesados se escriben en el nuevo video de salida.

Si desea conocer su implementación en código puede acceder al siguiente enlace: practica 1.

Procesamiento en paralelo (CUDA C/C++):

En el caso de CUDA, indicamos al programa la cantidad de hilos que se utilizarán por bloque. Dentro del programa, manejamos los bloques como elementos bidimensionales. Es decir, si especificamos 32 como parámetro, tendremos un total

de 32 * 32 = 1024 hilos por bloque. Además, para determinar la cantidad de bloques, se tiene en cuenta el número de hilos por bloque, así como el ancho y alto del video de entrada.

Se emplea la técnica 'block-wise', cada bloque de hilos se encarga de procesar una región de cada fotograma del video de entrada. Con el número de bloques e hilos por bloque definidos, aplicamos el algoritmo a cada fotograma utilizando el kernel y copiamos los datos de forma asincrónica cada fotograma en la memoria global de la GPU. Los hilos trabajan con la memoria global.

Posteriormente, después de procesar el fotograma, lo traemos desde la GPU a la memoria de la CPU y lo escribimos en el video de salida.

Si desea conocer su implementación en código puede acceder al siguiente enlace: practica 2.

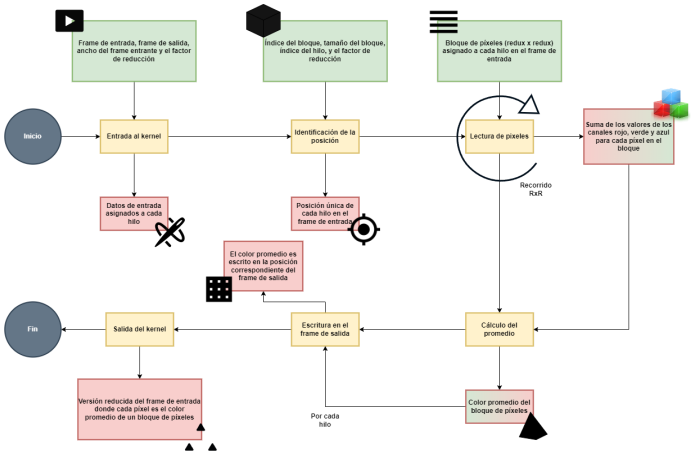

Figura 1. Secuencia de instrucciones realizadas en el GPU Kernel para realizar la redimensión. (Verde: entradas, Rojo: salidas, Amarillo: procesos)

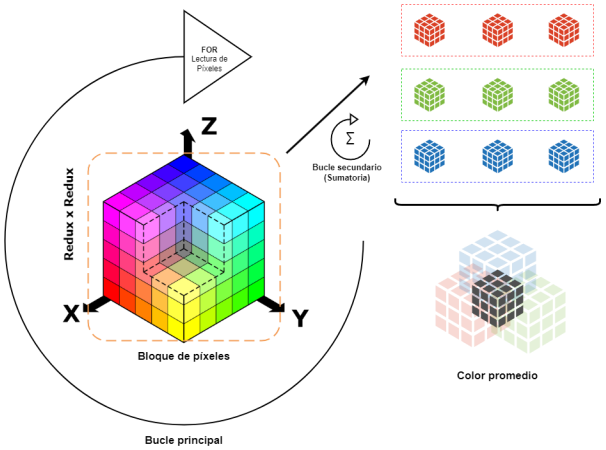

Figura 2. Proceso repetitivo con el que se promedia los píxeles del frame de entrada para redimensionar el video. Se realiza por cada hilo que es asignado a un bloque de píxeles de Redux x Redux de tamaño.

Procesamiento en paralelo (OpenMPI):

En esta ocasión, vamos mediante procesos paralelizar el trabajo. Adicionalmente, usaremos hilos para paralelizar las tareas de los procesos.
Primeramente se configura maquinas independientes para crear un cluster usando el servicio de Compute Engine (Virtual Machine) de Google Cloud. Una maquina actuara como servidor del sistema Network File System (NFS) con el fin de generar un espacio compartido para el cluster. Asi pues todos los nodos podran acceder a los archivos del programa.
Las configuraciones pertinentes se pueden encontrar en el archivo LEEME del código fuente.
Por lo anterior, cada proceso dispone de acceso al video de entrada. Entonces el video se parte en partes iguales segun el número de proceso a fin de que cada proceso se encargue de cierta cantidad de fotogramas. Cada fotograma se procesa usando hilos con openMP.
A medida que se procesa el video cada proceso genera un archivo video de salida que al finalizar la tarea guarda en la carpeta compartida.
Se coloco una barrera para que antes de generar el video de salida final todo el video de entrada haya sido procesado.
Luego de que todos los procesos hayan llegado a la barrera, el proceso cero se encargara de abrir cada archivo de salida generado por cada proceso y escribir el archivo de video final. Esta operación es bastante rapida ya que no se debe procesar ningun frame.
La cantidad de procesos e hilos son configurables como parametros al ejecutar el programa.
Si desea conocer su implementación en código puede acceder al siguiente enlace: practica 3.

IV. Experimentación

En esta sección, se presenta una descripción detallada de los experimentos llevados a cabo para evaluar el rendimiento y la efectividad del software de procesamiento de video desarrollado. El objetivo principal de estos experimentos fue analizar cómo el programa respondía a un video de 30 segundos.

- ***Datos de Entrada:***

  En general, para llevar a cabo los experimentos, se utilizó un video de 30, el cual puede encontrar en el código fuente.
  Para el caso de openMP se utilizaron cuatro videos de referencia, cada uno con una duración diferente: 10 segundos, 30 segundos, 60 segundos y 120 segundos. Estos videos se seleccionaron para representar una variedad de escenarios de uso comunes en aplicaciones de procesamiento de video.

- ***Configuración de Entorno (OpenMP):***

  Se realizaron pruebas en un sistema de máquina virtual equipado con el sistema operativo Linux Ubuntu 22.04 LTS, al que se le fueron asignados 4

procesadores lógicos, 5459 MB de memoria base y 75 GB de almacenamiento. Según información resultante del comando "lscpu", cada procesador tiene asignado 1 hilo, por lo que tendríamos 4 hilos en total con miras a realizar las pruebas.

- ***Configuración de Entorno (CUDA C/C++):***

Se realizaron pruebas en un entorno de desarrollo en la nube gratuito basado en Jupyter Notebook, que proporciona acceso a recursos de cómputo, como CPU y GPU, a través de los servidores de Google. Estas máquinas poseen una tarjeta gráfica T4 de NVIDA.

- ***Configuración de Entorno (OPENMPI):***

Se configura máquinas independientes para crear un cluster usando el servicio de Compute Engine (Virtual Machine) de Google Cloud. Las maquinas tendran una carpeta compartida mediante NFS, donde el nodo 0 actuara como servidor. Se configuran un total de 8 nodos o máquinas virtuales donde en cada máquina se puede llevar a cabo entre 1 y 2 procesos. Especificaciones de las máquinas virtuales:

- 2 vCPU
- 2 GB de memoria
- 10 GB de almacenamiento
- Debian GNU/Linux 11 (bullseye)

- ***Metodología de Prueba:***

Cada experimento consistió en ejecutar el software con un video de referencia específico. Se realizaron tres repeticiones de cada experimento para garantizar la consistencia de los resultados, de los cuales se realiza una medición del promedio entre estas 3 ejecuciones. Para cada repetición, se aplicó el script de ejecución completo al video de entrada, y se registraron los tiempos de procesamiento en la respectiva salida de resultados.

V. Resultados

- ***Programa secuencial:***

Duración de Video (s)	Tiempo de Ejecución (RT) (s)
10	11.84
30	54,82
60	66.46
120	95.18

Tabla 3. Resultados de la experimentación con el programa secuencial.

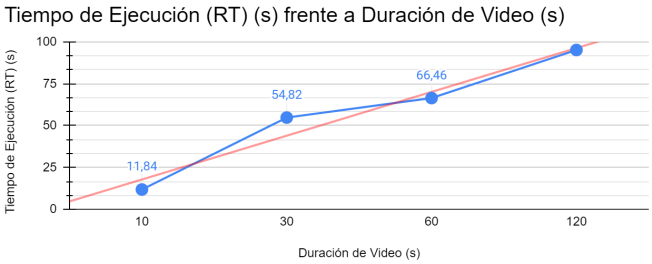

Figura 3. Gráfica de los resultados obtenidos para el método secuencial y de la tendencia que toman estos últimos.

- ***Programa paralelo (OpenMP):***

  - ***Video de 10 Segundos***

Cantidad de Hilos	Tiempo de Ejecución (RT) (s)	Factor de Aceleración (Sp) (s)
2	6.04	1.96
4	4.85	2.44
8	4.16	2.85
16	4.19	2.86

Tabla 4. Resultados de la experimentación con el método OpenMP para un video de 10 segundos.

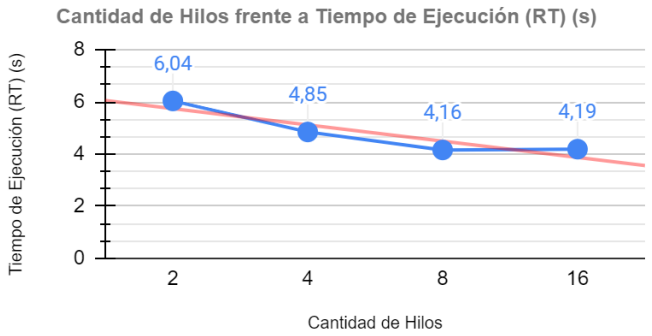

Figura 4. Gráfica de los resultados obtenidos para método OpenMP para un video de 10 segundos y de la tendencia que toman estos últimos.

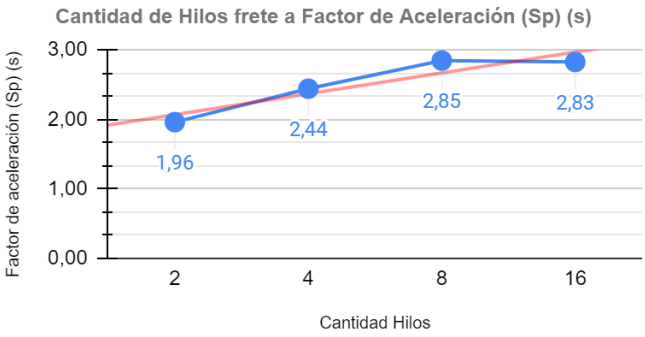

Figura 5. Gráfica de los factores de aceleración obtenidos para método OpenMP para un video de 10 segundos y de la tendencia que toman estos últimos.

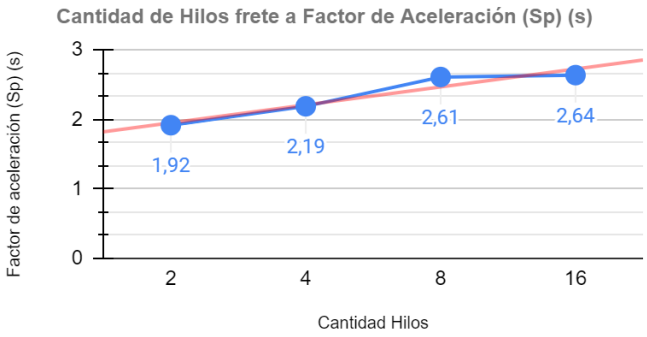

Figura 7. Gráfica de los factores de aceleración obtenidos para método OpenMP para un video de 30 segundos y de la tendencia que toman estos últimos.

- ***Video de 30 Segundos***

Cantidad de Hilos	Tiempo de Ejecución (RT) (s)	Factor de Aceleración (Sp) (s)
2	35.86	1.92
4	31.40	2.19
8	26.33	2.61
16	26.06	2.64

Tabla 5. Resultados de la experimentación con el método OpenMP para un video de 30 segundos.

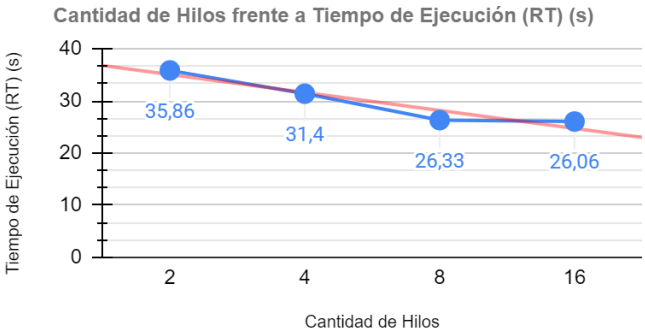

Figura 6. Gráfica de los resultados obtenidos para método OpenMP para un video de 30 segundos y de la tendencia que toman estos últimos.

- ***Video de 60 Segundos***

Cantidad de Hilos	Tiempo de Ejecución (RT) (s)	Factor de Aceleración (Sp) (s)
2	35.26	1.88
4	34.21	1.94
8	27.80	2.39
16	27.72	2.40

Tabla 6. Resultados de la experimentación con el método OpenMP para un video de 60 segundos.

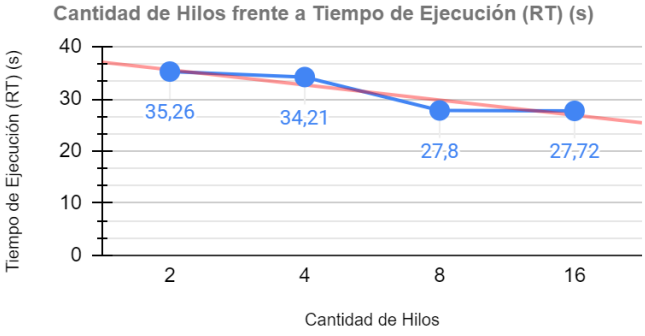

Figura 8. Gráfica de los resultados obtenidos para método OpenMP para un video de 60 segundos y de la tendencia que toman estos últimos.

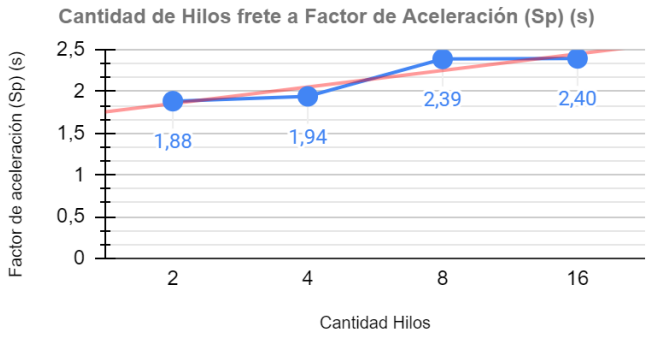

Figura 9. Gráfica de los factores de aceleración obtenidos para método OpenMP para un video de 60 segundos y de la tendencia que toman estos últimos.

- ***Video de 120 Segundos***

Cantidad de Hilos	Tiempo de Ejecución (RT) (s)	Factor de Aceleración (Sp) (s)
2	49.89	1.91
4	41.03	2.32
8	35.59	2.67
16	35.43	2.69

Tabla 7. Resultados de la experimentación con el método OpenMP para un video de 120 segundos.

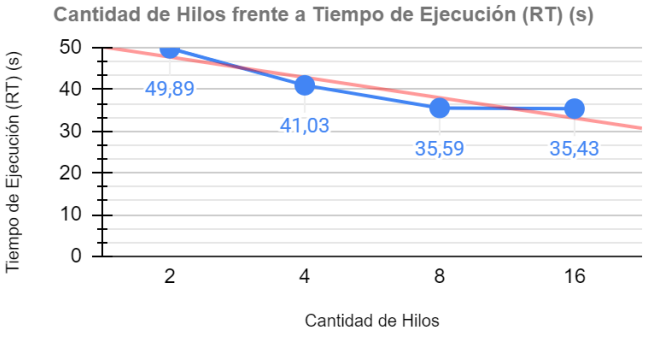

Figura 10. Gráfica de los resultados obtenidos para método OpenMP para un video de 120 segundos y de la tendencia que toman estos últimos.

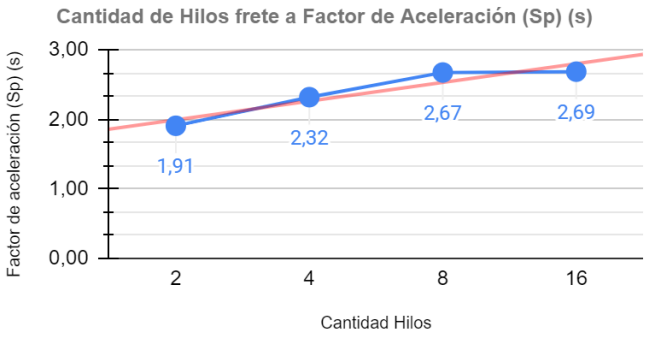

Figura 11. Gráfica de los factores de aceleración obtenidos para método OpenMP para un video de 120 segundos y de la tendencia que toman estos últimos.

El programa en paralelo con OpenMP depende del tamaño del fotograma del video de entrada, por ende la mejora que alcanza es limitada, por tal razon a medida que se aumente la duración del video este no alcanza los mejores resultados. Una mejor forma de paralelizar la reducción de video es mediante la parelelización de los frames del video, ya que son proporcionales a la duracion del video.

- ***Programa paralelo (CUDA):***

Para este caso, se hicieron las pruebas solo para un video de 30 segundos, de forma que se van aumentando los hilos por bloque de forma gradual.

- ***Video de 30 Segundos***

Cantidad de Hilos	Tiempo de Ejecución (RT) (s)	Factor de Aceleración (Sp) (s)
256	16,1	3,40
441	13,65	4,02
676	15,16	3,62
1024	16,15	3,39

Tabla 8. Resultados de la experimentación con el método CUDA para un video de 30 segundos.

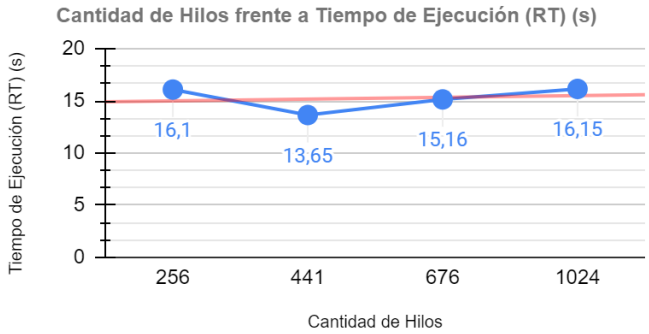

Figura 12. Gráfica de los resultados obtenidos para método CUDA para un video de 30 segundos y de la tendencia que toman estos últimos.

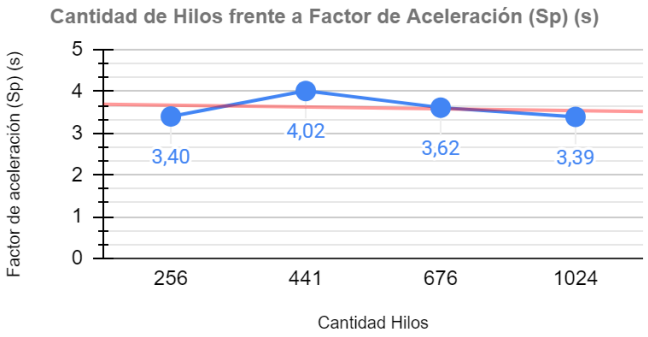

Figura 13. Gráfica de los resultados obtenidos para método CUDA para un video de 30 segundos y de la tendencia que toman estos últimos.

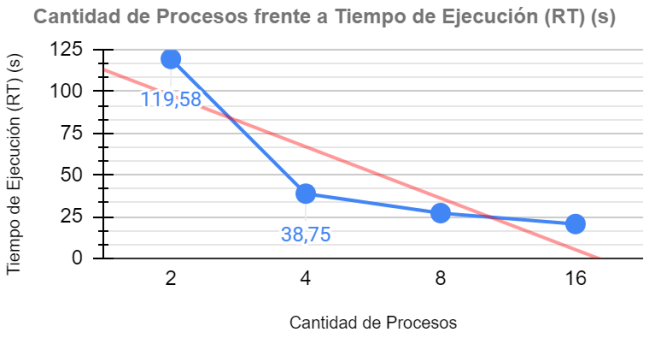

Figura 14. Gráfica de los resultados obtenidos para método OPENMPI para un video de 30 segundos y de la tendencia que toman estos últimos

En el caso de CUDA C/C++, para determinar el número de bloques, se parte de la cantidad de hilos por bloque y el tamaño del fotograma, lo que resulta en una proporción inversa entre la cantidad de hilos por bloque y la cantidad de bloques. Por lo tanto, aunque se reduzcan los hilos, se busca utilizar la mayor cantidad de bloques, y en caso contrario, se busca tener la mayor cantidad de hilos por bloque, de tal forma que se mantenga el mejor rendimiento posible.

En el programa en paralelo con CUDA se obtiene un punto de maxima mejora en 441 hilos para el video de 30 segundos, por tanto, aumentar la cantidad de hilos por bloque puede ser contraproducente debido a los accesos a memoria por hilo aumentan y aumenta el tiempo de procesamiento. Esto es aplicable a dentro de otras situaciones.

- ***Programa paralelo (OPENMPI):***

Para este caso, se hicieron las pruebas solo para un video de 30 segundos, de forma que se van aumentando los procesos de forma gradual.
Para obtener los datos cabe aclarar que se configuró que cada máquina ejecute dos procesos y un hilo por proceso.

Cantidad de Procesos	Tiempo de Ejecución (RT) (s)	Factor de Aceleración (Sp) (s)
2	119,58	0,46
4	38,75	1,41
8	27,17	2,02
16	20,56	2,67

Tabla 9. Resultados de la experimentación con el método OPENMPI para un video de 30 segundos.

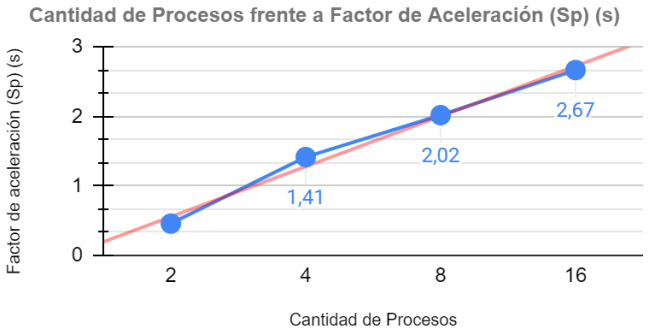

Figura 15. Gráfica de los resultados obtenidos para método OPENMPI para un video de 30 segundos y de la tendencia que toman estos últimos.

Podemos observar que el tiempo de ejecución entre 2 y 4 procesos aumenta drasticamente, esto puede deber se a que para el caso de dos procesos ambos se ejecuten en una misma máquina, por tanto, la creación de ambos procesos y asignaciones de memoria para cada uno de los procesos y sus hilos tarde y sume tiempo al tiempo de ejecución.

VI. Conclusiones

- El estudio realizado mostró que el rendimiento del algoritmo paralelo dependía en gran medida del hardware disponible. Si el sistema tiene un número limitado de núcleos de CPU o si ya está bajo una carga pesada de procesamiento, es posible que no veamos una mejora significativa con el algoritmo paralelo.

- Durante la práctica se emplearon diferentes técnicas para procesamiento de video como: paralelizar un frame y paralelizar por frames. Los resultados concluyen que el mejor método es paralelizar por frames teniendo en cuenta que la cantidad frames son proporcionales a la duración del video.

- OpenMP y OpenMPI generan resultados similares, pero el programa de OpenMPI exhibió una ventaja al paralelizar por la cantidad total de frames, mientras que con OpenMP se optó por paralelizar el procesamiento de un frame a la vez. A pesar de que el factor de aceleración es más notable para OpenMPI,

la diferencia entre ambos enfoques es relativamente pequeña. En última instancia, podemos concluir que la creación de procesos lleva más tiempo en comparación con la creación de hilos, lo que puede influir en el rendimiento general del programa.

- Importante destacar que la elección entre tecnologías como CUDA, OpenMP y OpenMPI depende en gran medida de la naturaleza específica de la tarea, los requisitos del sistema y las características del hardware disponible. En este caso particular, CUDA ha demostrado ser una opción sólida para el procesamiento de reducción de calidad de video debido a su capacidad para aprovechar las capacidades de las GPU.

- Basándonos en las pruebas realizadas para el procesamiento de reducción de calidad de video, los resultados indican que CUDA demostró ser superior en comparación con otras tecnologías, como OpenMP y OpenMPI. La superioridad de CUDA se evidenció debido a su capacidad para aprovechar la potencia de procesamiento masivamente paralela de las GPU.

  Es por este motivo que podemos deducir de los resultados obtenidos con CUDA que de cierta forma pueden atribuirse a varias características únicas de esta tecnología, lo que a su vez la hacen especialmente adecuada para el procesamiento de multimedia, como es el caso particular de esta investigación práctica. En primer lugar, CUDA permite un alto grado de paralelismo al permitir que miles de “hilos” se ejecuten simultáneamente. Cada hilo puede operar en un conjunto diferente de datos, lo que permite un procesamiento eficiente de tareas de gran volumen como la reducción de la calidad de un video. Esto es especialmente útil en el procesamiento de secuencias multimedia, donde las operaciones se pueden realizar en paralelo en cada píxel de un fotograma; y más aún cuando se tiene la disponibilidad de un entorno propicio para la ejecución de las pruebas, como fue en nuestro caso con una GPU T4 la cual no permitió evidenciar las características intrínsecas de CUDA.

  En segundo lugar, logramos determinar y demostrar que CUDA puede aprovechar la arquitectura de hardware de las GPU de una forma más eficiente y específicamente enfocada a un objetivo que otros medios vistos en la asignatura; es de destacar que estas herramientas basadas en una arquitectura de GPU están diseñadas para manejar operaciones de punto flotante de alta intensidad que son comunes en el procesamiento de elemento digitales como audios y videos. Es así como las GPU tienen cientos de núcleos que pueden procesar tareas en paralelo, en comparación con las CPU que tienen un número mucho menor de núcleos. Esto significa que las GPU pueden procesar una gran cantidad de datos simultáneamente, lo que resulta en un rendimiento significativamente mejorado en comparación con las CPU. Por lo tanto, no es inaudito para nosotros, como estudiantes de ingeniería de sistemas que han conocido el trasfondo de esta tecnología, que CUDA haya demostrado ser la tecnología más eficiente en nuestras pruebas para la reducción de la calidad del video en cuestión.

  En pocas palabras, con los resultados obtenidos se demuestra de una manera práctica que la tecnología de paralelismo CUDA, es una herramienta vanguardista a la hora de manejar información y realizar cálculos, operaciones y modificaciones sobre archivos digitales, los cuales son de vital importancia para lo que conocemos como sociedad moderna e interconectada del siglo XXI.

Referencias

[1] KeepCoding, "Tipos de paralelismo computacional," KeepCoding, 2022. [En línea]. Disponible en: https://keepcoding.io/blog/tipos-de-paralelismo-computacional. [Acceso: 13 octubre 2023].

[2] Pedraza, C., "Computación Paralela y Distribuida 2023-2" Universidad Nacional de Colombia, 2023. [En línea]. Disponible en: https://dis.unal.edu.co/~capedrazab/material_cursos/20232/pc/pc.html. [Acceso: 13 octubre 2023].