Universidad Sergio Arboleda

Ciencias de la Computación e Inteligencia Artificial

Taller Práctico 3:

Procesamiento de video en algoritmos CUDA con GPU, transformando un video a escala de grises

Informe de laboratorio

Estudiante:
Samuel Chaves Mora Cristian Bello Cuesta Daniel Ayala Guzman

Asignatura:
Metricas de rendimiento

Docente:
Guillermo Andrés De Mendoza Corrales

Bogotá D.C., Colombia
Noviembre 2025

Índice

1. Introducción **2**

2. Objetivos **2**

2.1. Objetivo general 2

2.2. Objetivos específicos 2

3. Marco teórico **3**

3.1. Procesamiento de video 3

3.2. Escala de grises 3

3.3. CUDA y procesamiento paralelo en GPU 3

3.4. Medición de rendimiento y speedup 4

4. Metodología **4**

4.1. Configuración de hardware 4

4.2. Configuración de software 4

4.3. Descripción general del procedimiento 4

4.4. Algoritmo secuencial (CPU) 5

4.5. Algoritmo paralelo (GPU con CUDA) 5

5. Resultados **6**

6. Análisis de rendimiento **6**

7. Conclusiones **7**

8. Referencias **8**

1. Introducción

El procesamiento de video es un área fundamental dentro de la ingeniería y la informática, ya que permite manipular, analizar y transformar secuencias de imágenes digitales para distintos propósitos, tales como compresión, transmisión, análisis de movimiento, visión por computadora y mejora de la calidad visual. En este contexto, el uso de arquitecturas paralelas como las Unidades de Procesamiento Gráfico (GPU) permite acelerar significativamente tareas que son altamente demandantes en términos computacionales.

En el presente informe se describe el desarrollo de un taller práctico cuyo objetivo principal es transformar un video en color a escala de grises utilizando dos enfoques: un algoritmo secuencial ejecutado en CPU y un algoritmo paralelo utilizando CUDA sobre una GPU. Posteriormente, se comparan los tiempos de ejecución de ambos enfoques para calcular el *speedup* y analizar las ventajas del paralelismo en la resolución de este tipo de problemas.

2. Objetivos

2.1. Objetivo general

Implementar y comparar la eficiencia de un algoritmo secuencial y uno paralelo basado en GPU para el procesamiento de video, transformando un video original en color a escala de grises, y analizar el *speedup* obtenido gracias al paralelismo.

2.2. Objetivos específicos

- Implementar un algoritmo secuencial en Python que convierta un video en color a escala de grises procesando frame por frame.
- Implementar un algoritmo paralelo utilizando CUDA (a través de Numba) para realizar la misma conversión aprovechando la GPU.
- Medir y registrar los tiempos de ejecución de ambos enfoques (CPU y GPU).
- Calcular el *speedup* logrado por el algoritmo paralelo en relación con el secuencial.
- Analizar los resultados obtenidos y discutir las ventajas del procesamiento paralelo en GPU.

3. Marco teórico

3.1. Procesamiento de video

Un video digital puede entenderse como una secuencia de imágenes fijas (frames) que se muestran de forma continua a una determinada tasa de fotogramas por segundo (FPS, *frames per second*). Cada frame está compuesto por una matriz de píxeles, y cada píxel almacena información de color e intensidad. Para representar el color de un píxel se utilizan con frecuencia modelos como RGB (Rojo, Verde, Azul), donde cada componente se codifica normalmente en 8 bits.

El procesamiento de video abarca todas las técnicas que permiten la manipulación de estas secuencias de imágenes: desde tareas simples como cambios de formato, redimensionamiento o filtrado, hasta operaciones avanzadas como segmentación, detección y seguimiento de objetos, compresión, super-resolución y análisis de movimiento.

3.2. Escala de grises

La conversión de una imagen en color a escala de grises consiste en obtener una representación de intensidad luminosa a partir de los componentes de color originales. Una forma común de calcular la intensidad es usar una combinación ponderada de los canales RGB, por ejemplo:

$$\text{Gray} = 0{,}299 \cdot R + 0{,}587 \cdot G + 0{,}114 \cdot B \tag{1}$$

Esta fórmula asigna mayor peso al canal verde, seguido del rojo y finalmente del azul, lo cual se aproxima mejor a la percepción humana de la luminancia.

3.3. CUDA y procesamiento paralelo en GPU

CUDA (*Compute Unified Device Architecture*) es una plataforma de computación paralela desarrollada por NVIDIA que permite a los programadores utilizar la GPU para realizar tareas de propósito general. En lugar de usar la GPU únicamente para renderizado gráfico, CUDA habilita la ejecución de *kernels*: funciones que se ejecutan de forma masivamente paralela en miles de hilos.

El modelo de programación CUDA se basa en la organización de los hilos en bloques (*blocks*) y cuadrículas (*grids*). Cada hilo ejecuta el mismo kernel pero opera sobre diferentes datos, lo que resulta especialmente útil en tareas de procesamiento de imágenes y video, donde cada píxel puede ser procesado de forma independiente o casi independiente.

3.4. Medición de rendimiento y speedup

Para evaluar el rendimiento de un algoritmo paralelo frente a su versión secuencial, se suele utilizar el concepto de *speedup*, definido como:

$$\text{Speedup} = \frac{T_{\text{secuencial}}}{T_{\text{paralelo}}} \tag{2}$$

donde $T_{\text{secuencial}}$ es el tiempo de ejecución del algoritmo en CPU (versión secuencial) y T_{paralelo} es el tiempo de ejecución del algoritmo paralelo en GPU. Un *speedup* mayor que 1 indica que el algoritmo paralelo es más rápido que el secuencial.

4. Metodología

4.1. Configuración de hardware

Para la realización de este laboratorio se utilizó el entorno de Google Colab, que proporciona acceso a una GPU NVIDIA a través de un entorno virtual en la nube. Aunque las características exactas de la GPU pueden variar, típicamente se dispone de tarjetas como Tesla T4 o similares, con memoria de video suficiente para procesar secuencias de imágenes de tamaño moderado.

4.2. Configuración de software

La implementación se desarrolló en Python utilizando las siguientes herramientas y librerías:

- **Python 3.x**: Lenguaje de programación principal del proyecto.
- **OpenCV** (`opencv-python`): Librería para el manejo de imágenes y videos (lectura, escritura y transformación de frames).
- **NumPy**: Manejo de arreglos y operaciones numéricas eficientes.
- **Numba CUDA**: Extensión de Numba que permite escribir kernels CUDA en Python para ejecutarlos en la GPU.
- **Google Colab**: Entorno de ejecución y cuadernos Jupyter en la nube.

4.3. Descripción general del procedimiento

El flujo de trabajo seguido en el laboratorio se puede resumir en las siguientes etapas:

1. Cargar el video original en color en el entorno de Python.

2. Extraer todos los frames del video y almacenarlos como imágenes individuales (`.jpg`).

3. Implementar y ejecutar un algoritmo secuencial en CPU que:

   - Lea cada frame.
   - Lo convierta a escala de grises.
   - Guarde el resultado en una carpeta de salida.

4. Implementar y ejecutar un algoritmo paralelo en GPU con CUDA que realice la misma conversión a escala de grises utilizando un kernel CUDA.

5. Reconstruir el video a partir de los frames en escala de grises, tanto para la versión CPU como para la versión GPU (si se desea).

6. Medir los tiempos de ejecución de cada enfoque y calcular el *speedup*.

4.4. Algoritmo secuencial (CPU)

El algoritmo secuencial se basa en el uso de OpenCV para leer el video y procesar cada frame de forma iterativa. El procedimiento básico es:

1. Abrir el archivo de video con `cv2.VideoCapture`.

2. Leer cada frame dentro de un ciclo hasta que no haya más imágenes.

3. Convertir el frame en color a escala de grises usando `cv2.cvtColor`.

4. Guardar el frame resultante en una carpeta de salida.

El tiempo de ejecución total se mide utilizando la diferencia entre las marcas de tiempo obtenidas antes y después del bucle principal.

4.5. Algoritmo paralelo (GPU con CUDA)

Para el algoritmo paralelo se utiliza Numba CUDA. La idea principal es copiar la imagen a memoria de la GPU y lanzar un kernel donde cada hilo se encargue de calcular el valor en escala de grises de un píxel. El procedimiento es el siguiente:

1. Convertir el frame leído con OpenCV a un arreglo de tipo `float32`.

2. Reservar un arreglo de salida para la imagen en escala de grises.

3. Definir un kernel CUDA que, para cada posición (x, y), lea los valores RGB del píxel, calcule la intensidad en escala de grises y escriba el resultado en el arreglo de salida.

4. Configurar la distribución de hilos en bloques y cuadrículas.

5. Ejecutar el kernel para cada frame.

6. Copiar el resultado de la GPU a la CPU (si es necesario) y guardar la imagen en disco.

Al igual que en el caso secuencial, se mide el tiempo total de procesamiento para todos los frames.

5. Resultados

En esta sección se presentan los resultados de los tiempos de ejecución obtenidos en el experimento, así como el *speedup* calculado para la versión GPU frente a la versión CPU.

Supongamos que, tras ejecutar el código, se obtuvieron los siguientes tiempos:

- Tiempo total secuencial (CPU): $T_{\text{secuencial}} = X_{\text{CPU}}$ segundos.
- Tiempo total paralelo (GPU): $T_{\text{paralelo}} = Y_{\text{GPU}}$ segundos.

A modo de ejemplo, se puede organizar la información en la Tabla 1.

Cuadro 1: Tiempos de ejecución de los algoritmos secuencial y paralelo.

Implementación	**Tiempo (s)**	**Observaciones**
CPU (secuencial)	X_CPU	Procesamiento frame a frame en CPU
GPU (CUDA)	Y_GPU	Kernel paralelo en GPU

El *speedup* se calcula como:

$$\text{Speedup} = \frac{T_{\text{secuencial}}}{T_{\text{paralelo}}} = \frac{X_{\text{CPU}}}{Y_{\text{GPU}}} \tag{3}$$

Reemplazando los valores medidos:

$$\text{Speedup} \approx S \tag{4}$$

donde S representa la ganancia de rendimiento obtenida al utilizar la GPU.

6. Análisis de rendimiento

A partir de los resultados obtenidos, se puede observar que el tiempo de ejecución del algoritmo paralelo en GPU es considerablemente menor que el de la versión secuencial en CPU (siempre que el tamaño del video y la cantidad de frames sean suficientemente

grandes). Esto se debe a que la GPU está compuesta por un gran número de núcleos diseñados para ejecutar de forma masiva operaciones similares sobre grandes conjuntos de datos, como ocurre en el procesamiento de imágenes.

Cuando el número de píxeles por frame y el número de frames del video son elevados, el trabajo puede dividirse eficientemente entre miles de hilos de ejecución, reduciendo así el tiempo total de procesamiento. Sin embargo, también es importante tener en cuenta ciertos factores:

- Existe un costo asociado a la transferencia de datos entre la memoria de la CPU y la memoria de la GPU.
- Para videos muy pequeños o con pocos frames, la sobrecarga de lanzar kernels y copiar datos podría hacer que el beneficio del paralelismo no sea tan notable.
- La eficiencia también depende de la configuración de los bloques y la cuadrícula en CUDA, así como del uso adecuado de la memoria de la GPU.

En términos generales, un *speedup* significativamente mayor que 1 confirma que el uso de CUDA y GPU es una estrategia adecuada para acelerar este tipo de tareas de procesamiento de video, especialmente cuando se manejan volúmenes de datos grandes.

7. Conclusiones

A partir del desarrollo de este taller práctico se pueden extraer las siguientes conclusiones:

- El procesamiento de video es una tarea inherentemente paralelizable, ya que cada píxel o cada frame puede ser tratado de forma relativamente independiente.
- La implementación secuencial en CPU resulta sencilla de comprender y programar, pero presenta limitaciones en cuanto a tiempo de ejecución cuando el volumen de datos crece.
- El uso de CUDA y GPU, a través de Numba en Python, permite distribuir el trabajo entre una gran cantidad de hilos, obteniendo tiempos de ejecución mucho menores para el mismo problema.
- El *speedup* obtenido en la práctica demuestra la ventaja del procesamiento paralelo para tareas intensivas, como la conversión de un video completo a escala de grises.
- Este tipo de ejercicios permite al estudiante comprender de manera práctica la importancia del paralelismo y la diferencia entre ejecutar algoritmos en CPU versus GPU.

8. Referencias

- OpenCV Documentation. Disponible en: https://docs.opencv.org/
- Numba CUDA Documentation. Disponible en: https://numba.readthedocs.io/en/stable/cuda/index.html
- NVIDIA CUDA Toolkit Documentation. Disponible en: https://docs.nvidia.com/cuda/