Taller Práctico 2

Detección de Bordes mediante GPUs y CUDA (Sobel Algorithm)

Santiago Mendivelso, Jhostin Aristizabal
Universidad Sergio Arboleda

2025

Resumen

Este informe aborda la ejecución y comparación del algoritmo de detección de bordes Sobel mediante procesamiento secuencial en CPU y procesamiento paralelo en GPU usando CuPy como interfaz de CUDA. Desde la implementación del algoritmo en Google Colab utilizando una GPU NVIDIA Tesla T4, se examinaron los tiempos de ejecución, la eficiencia computacional y el speedup logrado en comparación con la solución secuencial. Los resultados experimentales muestran una mejora considerable en el rendimiento al utilizar paralelismo masivo, logrando un speedup cercano a 6x en la fase de convolución del operador Sobel. Este estudio facilita una comprensión detallada del funcionamiento matemático del algoritmo y de los beneficios del procesamiento paralelo en la manipulación digital de imágenes.

1. Introducción

La detección de bordes es un procedimiento esencial en el estudio de imágenes y visión artificial, facilitando la acentuación de contornos, discontinuidades y variaciones bruscas de intensidad. Uno de los enfoques más comunes para esta labor es el operador Sobel, que realiza convoluciones en una imagen en escala de grises para estimar las aproximaciones del gradiente en las direcciones horizontal y vertical.

Aunque su ejecución de manera secuencial es conceptualmente fácil, es costosa en términos computacionales por la naturaleza repetitiva y altamente paralelizable del proceso de convolución. Así, la utilización de unidades de procesamiento gráfico (GPU) se presenta como una opción perfecta para agilizar este tipo de tareas.

En este informe se desarrollaron dos variantes del algoritmo Sobel: una variante secuencial en CPU y otra paralela utilizándose CuPy para llevar a cabo operaciones matriciales en GPU. Las dos versiones fueron analizadas y contrastadas utilizando Google Colab con una GPU NVIDIA Tesla T4.

2. Marco Teórico

2.1. Concepto de borde e intensidad

Una imagen digital puede interpretarse como una función bidimensional de intensidad $I(x, y)$. Un borde se define como una región donde ocurre un cambio abrupto de intensidad, lo cual puede detectarse mediante el cálculo del gradiente de la función.

2.2. Gradiente

El gradiente mide la tasa de cambio espacial de la intensidad en una imagen. Matemáticamente:

$$\nabla I = \left(\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y} \right) = (G_x, G_y)$$

Donde:

- G_x mide variaciones horizontales y detecta bordes verticales.
- G_y mide variaciones verticales y detecta bordes horizontales.

La magnitud del gradiente se calcula como:

$$G = \sqrt{G_x^2 + G_y^2}$$

2.3. Operador Sobel

El operador Sobel utiliza dos máscaras de convolución 3x3:

$$K_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \qquad K_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Estas máscaras ponderan de forma diferencial los píxeles vecinos, mejorando la sensibilidad al cambio inmediato de intensidad.

2.4. Convolución

La convolución es una operación clave en el procesamiento digital de imágenes. Dado un kernel K y una imagen I, la convolución en (i, j) está dada por:

$$(I * K)(i, j) = \sum_{m=-1}^{1} \sum_{n=-1}^{1} I(i + m, j + n)\, K(m + 1, n + 1)$$

Esta operación se repite para cada píxel de la imagen, lo cual la hace intensiva en cómputo.

2.5. Procesamiento en GPU y paralelismo masivo

Las GPU están concebidas para llevar a cabo miles de hilos simultáneamente, lo que es perfecto para procesos independientes por píxel como la convolución. Cada hilo puede encargarse de manejar un píxel o grupo de píxeles, alcanzando aceleraciones significativas en comparación con la CPU.

En este estudio se utilizó CuPy, una biblioteca que es compatible con NumPy y posibilita realizar las mismas operaciones utilizando CUDA de forma transparente

3. Metodología

3.1. Hardware

El experimento se ejecutó en Google Colab con la siguiente configuración:

- GPU: NVIDIA Tesla T4
- Núcleos CUDA: 2560
- Memoria: 16 GB GDDR6

3.2. Software

- Python 3.12
- CuPy CUDA 12.x
- NumPy 2.0
- OpenCV

3.3. Descripción de la implementación

3.3.1. Implementación CPU

La versión CPU utiliza dos bucles anidados para aplicar manualmente la convolución de Sobel. Esta versión sigue rigurosamente el enfoque secuencial tradicional.

- Conversión a escala de grises.
- Extracción de regiones 3x3.
- Multiplicaciones elemento a elemento con K_x y K_y.
- Cálculo de la magnitud del gradiente.

3.3.2. Implementación GPU

La versión GPU implementa una convolución manual usando operaciones vectorizadas con CuPy:

- Se agrega padding a la imagen en GPU.
- Se realiza la convolución deslizante con operaciones matriciales.
- Se calcula G_x y G_y.
- Se obtiene la magnitud final en GPU.

La versión GPU mantiene la lógica matemática del operador Sobel y utiliza CUDA a través de CuPy, adaptándose a las restricciones del entorno Colab.

4. Resultados

4.1. Tiempos obtenidos

Proceso	**Plataforma**	**Tiempo (s)**
Escala de grises	CPU	0.0181
Sobel	CPU	8.4639
Sobel	GPU	1.4266

Cuadro 1: Tiempos de ejecución medidos en Google Colab.

4.2. Resultados Imagen

En esta sección se presentan los resultados visuales obtenidos a partir del proceso de detección de bordes mediante el operador Sobel. La Figura 1 muestra la imagen original cargada por el usuario, mientras que la Figura 2 presenta la imagen resultante tras la aplicación del algoritmo Sobel desarrollado durante el laboratorio.

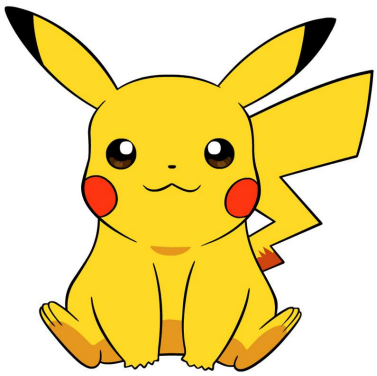

Figura 1: Imagen original utilizada como entrada para el procesamiento.

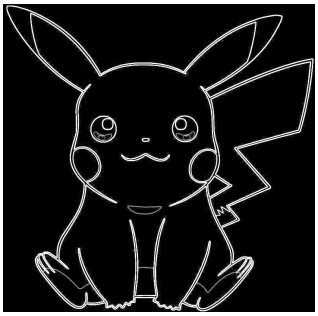

Figura 2: Imagen procesada mediante el operador Sobel, donde se resaltan los bordes detectados.

4.3. Cálculo de speedup

$$\text{Speedup} = \frac{T_{CPU}}{T_{GPU}}$$

$$\text{Speedup} = \frac{8{,}4639}{1{,}4266} \approx 5{,}93$$

El paralelismo en GPU permitió ejecutar el algoritmo aproximadamente **6 veces más rápido** que la versión secuencial.

5. Discusión

Los hallazgos corroboran que la convolución del operador Sobel se beneficia significativamente del procesamiento paralelo. La disminución del tiempo en GPU evidencia la naturaleza altamente paralelizable de este proceso, ya que cada píxel puede ser tratado de forma independiente.

La CPU, al llevar a cabo un bucle doble secuencial, experimenta un tiempo notablemente más largo, sobre todo cuando la imagen que se procesa es de un tamaño considerable. En cambio, la GPU reparte la carga entre miles de núcleos, posibilitando una aceleración significativa

6. Conclusiones

La implementación efectuada evidencia claramente de qué manera el paralelismo masivo de las GPU revoluciona el procesamiento digital de imágenes, transformando un algoritmo que normalmente es secuencial y costoso como Sobel en una operación sumamente eficiente. La GPU Tesla T4 logró casi un **6× de aceleración** en comparación con el CPU, incluso empleando una versión vectorizada y no un núcleo CUDA optimizado a mano. Estos hallazgos corroboran que las GPU no solo optimizan el rendimiento, sino que permiten situaciones donde el procesamiento en tiempo real se vuelve viable, como en visión por computadora, robótica y análisis de video.

7. Bibliografía

[1] Gonzalez, R. C., & Woods, R. E. (2008). *Digital Image Processing*. Pearson.

[2] NVIDIA. (2024). *CUDA C Programming Guide*.

[3] CuPy Developers. (2024). *CuPy Documentation*. Recuperado de: `https://cupy.dev`