Universidad Sergio Arboleda – Profesor Guillermo Andrés de Mendoza Corrales

Informe Técnico — Taller Práctico 1: Detección de Bordes con GPU y CUDA

Ayala Daniel – Bello Cristian – Chaves Samuel

24/10/2025

Marco Teórico

El procesamiento en CPU se realiza de forma secuencial, utilizando pocos núcleos que ejecutan múltiples tareas una tras otra. En contraste, una GPU está optimizada para el cálculo masivo de operaciones simples en paralelo, utilizando miles de núcleos que ejecutan el mismo tipo de instrucción sobre grandes volúmenes de datos.

En tareas como la conversión de una imagen o el cálculo de bordes (operaciones por píxel), la GPU puede lograr una aceleración significativa porque cada hilo puede encargarse de un píxel diferente.

Ventajas principales del procesamiento vectorial en GPU sobre CPU:

1.Paralelismo masivo: La GPU puede procesar miles de vectores o píxeles en paralelo, mientras que la CPU solo maneja decenas de hilos.

2.Rendimiento en cómputos repetitivos: En operaciones matemáticas intensivas (como transformaciones, filtrados o multiplicaciones matriciales), la GPU puede lograr aceleraciones de hasta cientos de veces.

3.Eficiencia energética: Realiza más operaciones por vatio consumido en comparación con una CPU.

4.Alta tasa de memoria (Bandwidth): La GPU tiene un ancho de banda de memoria muy superior, lo que permite mover grandes volúmenes de datos con menor latencia relativa.

5.Escalabilidad: Al aumentar el tamaño del problema (más píxeles o vectores), la GPU mantiene un rendimiento estable gracias a su arquitectura paralela.

En resumen, la GPU supera a la CPU en tareas donde el mismo cálculo debe aplicarse a grandes conjuntos de datos, como en gráficos, aprendizaje profundo o procesamiento de imágenes.

Metodología

Se desarrollaron dos algoritmos: uno en CPU y otro en GPU. Ambos procesan la misma imagen de entrada y realizan las etapas: conversión a gris, suavizado, detección de bordes, umbral adaptativo, limpieza morfológica y filtrado de componentes pequeños. El GPU utiliza CuPy para ejecutar un kernel CUDA que paraleliza la conversión RGB®gris.

Configuración del Hardware:

CPU: AMD EPYC 7B12, 2 núcleos virtuales (1 núcleo físico, 2 hilos).

Memoria RAM: 12 GB disponibles (8.8 GB en uso promedio durante ejecución).

GPU: NVIDIA Tesla T4 con 15 360 MiB (≈15 GB) de VRAM, 2560 núcleos CUDA, temperatura promedio 39 °C.

Versión CUDA: 12.4.

Driver NVIDIA: 550.54.15.

Configuración del Software:

Sistema operativo: Ubuntu 20.04 (entorno Google Colab).

Python: 3.12.12.

Librerías:

NumPy: 2.0.2 – operaciones numéricas vectorizadas.

OpenCV: 4.12.0 – procesamiento de imágenes y filtros.

Matplotlib: 3.10.0 – visualización y comparación gráfica de resultados.

CuPy: 13.3.0 – ejecución de código CUDA desde Python.

Explicación de los algoritmos desarrollados

CPU: convierte la imagen a escala de grises, aplica suavizado gaussiano, detecta bordes con Canny, combina umbrales adaptativos y realiza operaciones morfológicas para limpiar el resultado.

GPU: replica el mismo flujo, pero realiza la conversión RGB→grises en paralelo mediante un kernel CUDA, aprovechando la ejecución simultánea de miles de hilos.

Resultados

Se procesó la imagen vegetita.jpg (644×362 px) con ambos algoritmos.

El resultado visual fue el mismo: un contorno nítido y bien definido, sin pérdida de información.

Algoritmo	Gray (s)	Blur (s)	Canny (s)	Thresh+OR (s)	Morph (s)	CC+Invert (s)	Total (s)	Píxeles bordes	Archivo
CPU	0.0024	0.0090	0.0146	0.0048	0.0054	0.0147	0.1302	64 467	vegetita.jpg
GPU	0.0013	0.0102	0.0172	0.0056	0.0064	0.0229	0.0691	64 453	vegetita.jpg

Observaciones:

La diferencia en píxeles de borde es mínima (14 píxeles ≈ 0.02 %).

El delineado final conserva la calidad visual en ambas versiones.

El tiempo total en GPU fue significativamente menor.

Análisis de Rendimiento

El algoritmo paralelo ejecutado en GPU redujo el tiempo total de 0.1302 s a 0.0691 s, logrando un speedup de 1.88×.

Esto demuestra una mejora real de rendimiento gracias al uso de CUDA y la ejecución simultánea de múltiples hilos para la conversión RGB→gris.

La mejora se justifica porque:

1. La imagen tiene suficiente tamaño para amortiguar el costo de transferencia.

2. El kernel CUDA trabaja por píxel, aprovechando los 2560 núcleos de la Tesla T4.
3. Se redujo la latencia al paralelizar la etapa más intensiva (conversión y cálculo por canal).

Sin embargo, el resto del flujo (suavizado, Canny, morfología) aún corre en CPU, lo que limita el speedup máximo alcanzable.

Conclusiones

1. Se comprobó experimentalmente el speedup, con una aceleración de casi 2× al usar la GPU.

2. El paralelismo de datos en CUDA es eficaz cuando el tamaño de la imagen es moderado o grande.

3. La calidad del resultado no se vio afectada por el procesamiento paralelo.

4. Para alcanzar mayores aceleraciones, se podrían portar más etapas (filtros y Canny) a GPU.

El experimento valida el objetivo del laboratorio: comparar, medir y demostrar cómo el uso de GPU puede optimizar el procesamiento vectorial frente al enfoque secuencial de CPU.