Taller Práctico 1:
Detección de Bordeado Simple con CPU y GPU (CUDA)

Camilo Andrés Rodríguez — Universidad Sergio Arboleda

Noviembre 2025

1. Objetivo del Laboratorio

El objetivo principal de este laboratorio es implementar un proceso de detección de bordes en una imagen, utilizando dos enfoques distintos:

- Un algoritmo secuencial ejecutado en CPU.
- Un algoritmo paralelo ejecutado en GPU mediante CUDA.

Posteriormente, se evalúa el rendimiento de ambos métodos para determinar el *speedup* obtenido con el procesamiento paralelo.

(Lugar reservado para una imagen introductoria del proceso o la imagen original)

2. Marco Teórico

El procesamiento con GPU ofrece ventajas significativas frente a la CPU cuando se trabaja con vectores, matrices e imágenes. Las GPUs están diseñadas bajo un modelo masivamente paralelo, lo que permite ejecutar miles de hilos simultáneamente.

2.1. Ventajas de las GPU sobre CPU

- **Paralelismo masivo:** Las GPUs contienen cientos o miles de núcleos capaces de operar en paralelo.
- **Eficiencia en operaciones vectorizadas:** Ideal para operaciones repetitivas como recorrer cada píxel de una imagen.
- **Mayor ancho de banda de memoria:** Permite mover datos internos más rápido que en una CPU.

- **Reducción del tiempo de ejecución:** La división del trabajo entre múltiples hilos acelera el procesamiento.

Estas ventajas hacen que algoritmos como la detección de bordes puedan beneficiarse significativamente del paralelismo.

3. Metodología

3.1. Configuración del Hardware

- **CPU:** Máquina virtual de Google Colab (2 vCPUs, arquitectura x86_64).
- **GPU:** NVIDIA Tesla T4, 2560 núcleos CUDA, 16 GB GDDR6.

3.2. Configuración del Software

- Python 3.12
- OpenCV 4.x
- NumPy
- Matplotlib
- CuPy (para ejecución GPU con CUDA)

Captura del entorno de ejecución:

```
=== INFORMACIÓN DEL ENTORNO ===

Python version: 3.12.12 (main, Oct 10 2025, 08:52:57) [GCC 11.4.0]
Plataforma: Linux-6.6.105+-x86_64-with-glibc2.35

GPU detectada:
{'name': b'Tesla T4', 'totalGlobalMem': 15828320256, 'sharedMemPerBlock':

Versión CUDA (Numba): (12, 6)

=== Versiones de librerías ===
OpenCV: 4.12.0
NumPy: 2.0.2
CuPy:  13.6.0
```

3.3. Descripción de los Algoritmos

3.3.1. Algoritmo en CPU

El algoritmo secuencial recorre cada píxel de la imagen, calcula la suma de sus componentes RGB y aplica un umbral (*threshold*). Si la suma supera este valor, el píxel se convierte en blanco; de lo contrario, en negro.

- Complejidad: $O(N \times M)$

Código del algoritmo CPU:

```
def edge_cpu(img, threshold=150):
    height, width, _ = img.shape
    output = np.zeros((height, width), dtype=np.uint8)

    for i in range(height):
        for j in range(width):
            r, g, b = img[i, j]
            s = int(r) + int(g) + int(b)
            output[i, j] = 255 if s > threshold else 0

    return output
```

3.3.2. Algoritmo en GPU

El algoritmo en GPU utiliza un *kernel CUDA* que asigna un hilo por píxel. Esto permite paralelizar completamente el procesamiento de la imagen.

- Complejidad teórica por hilo: $O(1)$
- Complejidad global paralela: $O(\frac{N \times M}{\text{hilos}})$

Kernel CUDA utilizado para la ejecución en GPU:

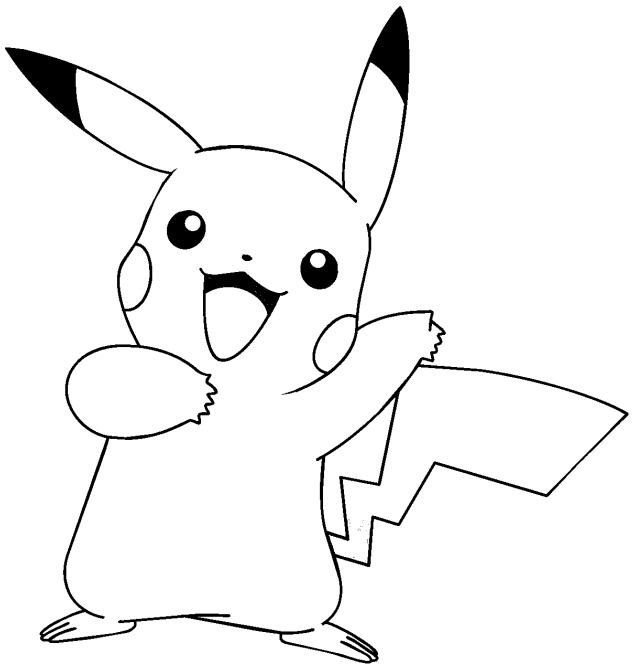
```
kernel = cp.ElementwiseKernel(
    in_params='uint8 r, uint8 g, uint8 b, int32 threshold',
    out_params='uint8 out',
    operation='''
        int s = (int)r + (int)g + (int)b;
        out = (s > threshold) ? 255 : 0;
    ''',
    name='edge_gpu'
)
```

4. Resultados

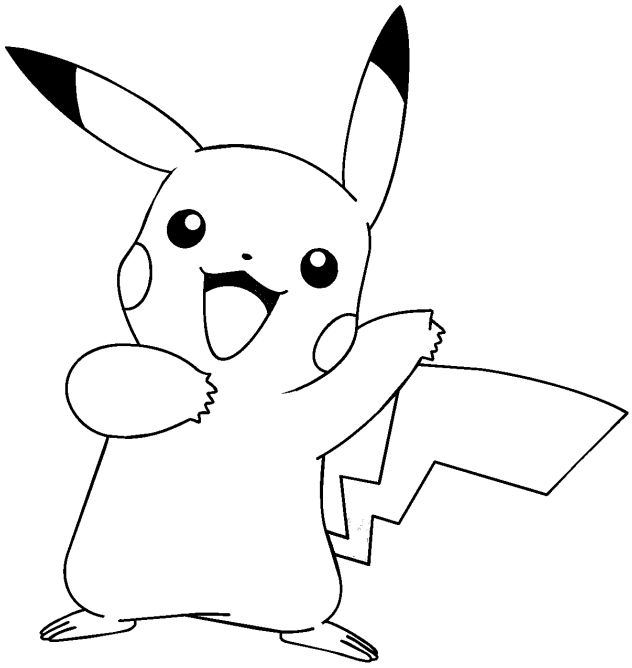
A continuación se presentan los resultados obtenidos al ejecutar ambos algoritmos sobre la imagen seleccionada.

4.1. Imagen Original

4.2. Imagen Procesada con CPU

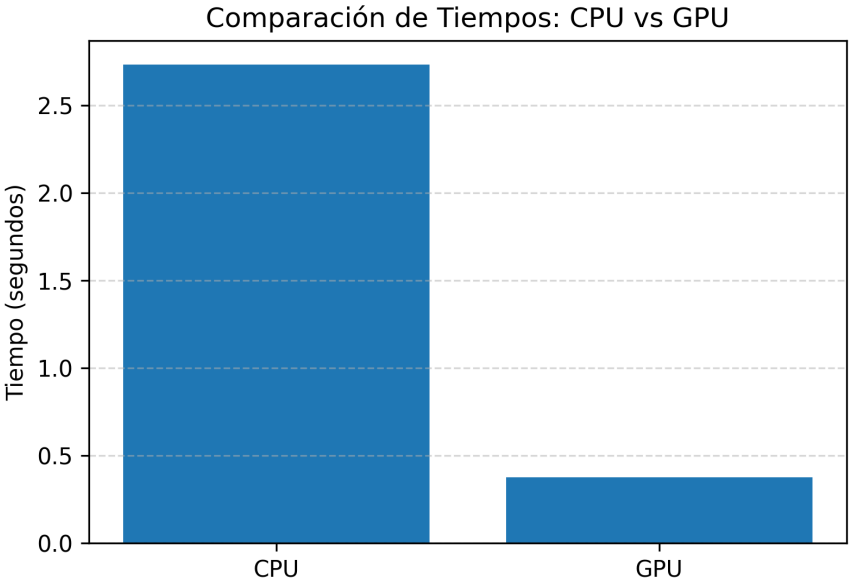


4.3. Imagen Procesada con GPU



4.4. Tiempos de Ejecución

- Tiempo CPU: **2.7330** segundos
- Tiempo GPU: **0.3760** segundos



5. Análisis de Rendimiento

El tiempo de ejecución en GPU fue significativamente menor en comparación con la CPU. Esto se debe principalmente a la arquitectura paralela de las GPUs, que permite ejecutar miles de hilos simultáneamente y dividir el procesamiento de los píxeles entre múltiples núcleos.

En este laboratorio, los tiempos obtenidos fueron los siguientes:

- Tiempo CPU (secuencial): **2.7330 s**
- Tiempo GPU (CUDA - CuPy): **0.3760 s**

Para evaluar el beneficio del paralelismo, se calcula el **speedup** como:

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

Sustituyendo los valores medidos:

$$\text{Speedup} = \frac{2{,}7330}{0{,}3760} \approx 7{,}27$$

Esto significa que el algoritmo ejecutado en GPU fue aproximadamente **7.27 veces más rápido** que su versión secuencial en CPU.

Este resultado demuestra la ventaja del cómputo paralelo para tareas altamente vectorizables como el procesamiento de imágenes, donde cada píxel puede ser procesado por un hilo independiente.

6. Conclusiones

- El procesamiento paralelo en GPU demuestra una mejora significativa en el tiempo de ejecución frente al enfoque secuencial en CPU.
- La arquitectura de las GPU es especialmente útil para tareas altamente paralelizables como el procesamiento de imágenes.
- Este taller permitió comprender la importancia del paralelismo y el uso eficiente del hardware para resolver problemas de cómputo intensivo.
- La comparación entre CPU y GPU evidencia el valor de emplear computación paralela en escenarios reales.