Detección de Bordeado mediante Algoritmos Paralelos

Autores

Kevin Nicolai García Rodríguez
Juan José Márquez Villarreal

Fecha

Noviembre 2025

1. Introducción

El procesamiento de imágenes es un área donde el uso de algoritmos paralelos marca la diferencia, porque permite acelerar muchísimo tareas que normalmente requieren bastante tiempo de cómputo. Una de esas tareas es la detección de bordes, una técnica básica pero muy importante para entender y analizar imágenes digitales.

En este laboratorio vamos a implementar dos versiones de un mismo algoritmo para detectar bordes usando el operador de Sobel: una versión secuencial que corre en la CPU y otra versión paralela que se ejecuta en la GPU. Luego compararemos los tiempos de ejecución para ver qué tanto mejora el rendimiento cuando usamos paralelismo y mediremos ese beneficio mediante el speedup.

Objetivo del Laboratorio

El objetivo de este laboratorio es implementar y comparar la eficiencia de un algoritmo secuencial en CPU y uno paralelo en GPU al procesar una imagen. El resultado buscado es demostrar el *speedup* (aceleración) logrado mediante el uso del paralelismo y el aprovechamiento de capacidades de cómputo masivo.

2. Marco Teórico

2.1. Operador de Sobel

El operador de Sobel es uno de los métodos más utilizados para la detección de bordes en imágenes digitales. Su propósito es resaltar las regiones donde existe un cambio brusco de intensidad, lo cual generalmente corresponde a un borde.

Este operador utiliza dos núcleos de convolución (kernels), una para detectar cambios en la dirección horizontal (G_x) y otra en la dirección vertical (G_y):

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

La magnitud del gradiente en cada píxel se obtiene como:

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

Este valor indica qué tan fuerte es el borde en esa región. El operador de Sobel combina suavizado y diferenciación, permitiendo reducir el ruido mientras se detectan gradientes, lo que lo hace adecuado para implementaciones secuenciales y paralelas.

3. Metodología

3.1. Configuración del Hardware

Las pruebas se realizaron en un equipo portátil con las siguientes características:

- **Equipo**: Dell Latitude 5510
- **CPU**: Intel Core i5-10210U (4 núcleos, 8 hilos, hasta 4.20 GHz)
- **GPU**: Intel UHD Graphics integrada (1.10 GHz)
- **Memoria RAM**: 16 GB
- **Sistema Operativo**: Ubuntu 24.04.3 LTS (64 bits)
- **Kernel**: Linux 6.14.0-36-generic

Aunque este procesador incluye una GPU integrada, el paralelismo utilizado en este trabajo no se realiza mediante GPU, sino mediante ejecución paralela en *CPU* usando la librería `multiprocessing`, la cual permite aprovechar todos los hilos del procesador.

3.2. Configuración del Software

El entorno de ejecución utilizado fue un entorno virtual de Python (`.venv`) configurado con:

- **Python**: 3.12.3
- **Entorno virtual**: `venv`

Las librerías principales empleadas fueron:

- `numpy 2.3.4`: operaciones numéricas y manejo matricial.
- `pillow 12.0.0`: carga y manipulación de imágenes.
- `matplotlib 3.10.7`: visualización de resultados.
- `ImageIO 2.37.2`: manejo adicional de imágenes.
- `multiprocessing` (librería estándar): ejecución paralela en CPU.

Estas herramientas permitieron implementar el procesamiento secuencial y paralelo del filtro de Sobel, así como visualizar y comparar los resultados obtenidos.

3.3. Descripción del Algoritmo Implementado

Se desarrollaron dos implementaciones del filtro de Sobel: una versión secuencial en CPU y otra paralela mediante el uso de múltiples procesos. Ambas utilizan el mismo núcleo:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Algoritmo Secuencial (CPU)

La versión secuencial recorre la imagen píxel por píxel y aplica el núcleo Sobel mediante convolución manual. El proceso consiste en:

1. Convertir la imagen a escala de grises.
2. Recorrer cada píxel de la imagen, excluyendo bordes.
3. Multiplicar la vecindad 3×3 por el núcleo G_x y G_y.
4. Calcular el gradiente:

$$G = \text{mín}(255, \sqrt{G_x^2 + G_y^2})$$

5. Generar la matriz final y convertirla nuevamente a imagen.

Esta implementación tiene una complejidad $\mathcal{O}(n \cdot m)$, donde n y m son las dimensiones de la imagen.

Algoritmo Paralelo (CPU con `multiprocessing`)

El paralelismo se logra dividiendo la imagen en bloques horizontales y asignando cada bloque a un proceso independiente mediante `multiprocessing.Pool`. Cada proceso ejecuta exactamente la misma lógica de Sobel pero sobre una sección distinta de la imagen.

El procedimiento es:

1. Dividir la imagen en tantos bloques como núcleos disponibles.
2. Añadir una fila adicional por arriba y por abajo para evitar pérdida de contexto en los bordes.
3. Enviar cada bloque a un proceso con la función `_process_chunk`.
4. Aplicar Sobel de forma independiente en cada bloque.
5. Eliminar las filas solapadas y unir los resultados en una única matriz final.

Este método permite que cada núcleo procese una porción distinta de la imagen, reduciendo significativamente el tiempo de ejecución. El speedup depende del número de hilos disponibles y del tamaño de la imagen procesada.

Resumen del Algoritmo

- **Secuencial**: un solo proceso recorre toda la imagen.
- **Paralelo**: varios procesos trabajan simultáneamente dividiendo la imagen.
- **Resultado esperado**: reducción notable del tiempo de ejecución en imágenes grandes.

4. Resultados

En esta sección se presentan los resultados obtenidos tras ejecutar los algoritmos de detección de bordeado en sus versiones secuencial y paralela. El procesamiento secuencial registró un tiempo total de **6.4598 segundos**, mientras que el algoritmo paralelo logró completar la misma tarea en **2.0388 segundos**. Esto evidencia una mejora sustancial en el rendimiento al emplear procesamiento paralelo.

Para facilitar la interpretación de los resultados, se generaron visualizaciones comparativas que incluyen:

- Una comparación visual entre las imágenes resultantes de ambos métodos.
- Una gráfica del tiempo empleado por cada algoritmo.
- Una gráfica del *speedup* obtenido.

Las figuras correspondientes se muestran a continuación.

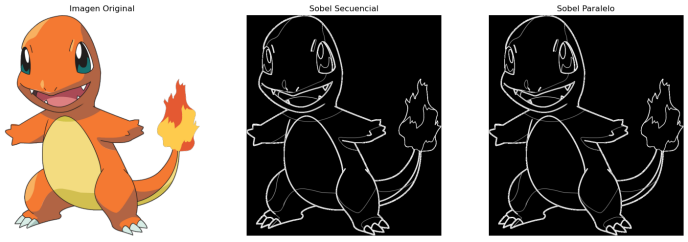

Figura 1: Comparación visual entre los resultados del algoritmo secuencial y paralelo.

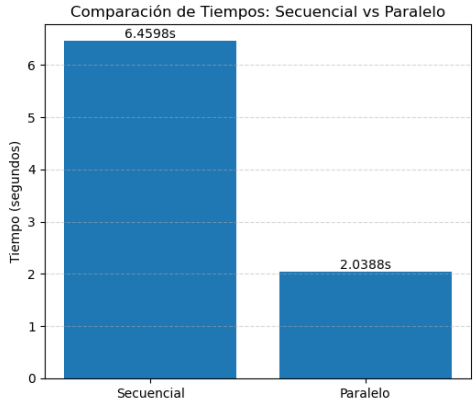

Figura 2: Tiempos de ejecución para los algoritmos secuencial y paralelo.

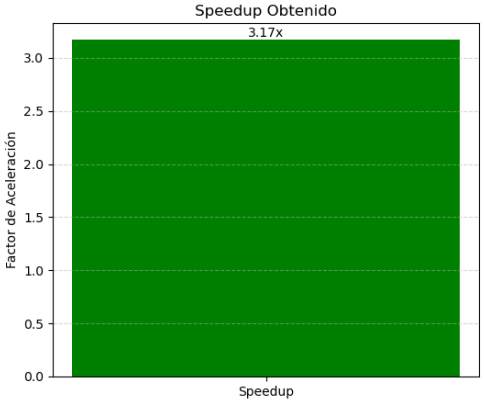

Figura 3: *Speedup* obtenido mediante procesamiento paralelo.

5. Análisis de Rendimiento

Los resultados muestran una mejora clara cuando se usa la versión paralela del algoritmo. La implementación secuencial tarda **6.4598 s** en procesar la imagen, mientras que la versión paralela baja este tiempo a **2.0388 s**. Esto representa un **speedup de 3.17x**, lo que demuestra que aprovechar el paralelismo realmente hace una diferencia en este tipo de tareas.

Este *speedup* indica que el algoritmo paralelo se comporta muy bien en problemas de procesamiento de imágenes, especialmente en operaciones que pueden hacerse por píxel sin depender unas de otras, como la conversión a escala de gri-

ses o la detección de bordes. Gracias a esta reducción en tiempo, es posible procesar imágenes o incluso video de manera mucho más rápida. Aun así, es importante tener en cuenta que el rendimiento puede variar según factores como:

- La cantidad de núcleos disponibles en la máquina.
- Qué tan bien se distribuyen las tareas entre los hilos o procesos.
- La sobrecarga generada por crear, coordinar y sincronizar las tareas.

En general, los resultados dejan claro que la versión paralela ofrece una mejora consistente y útil frente a la implementación secuencial.

6. Conclusiones

Los resultados obtenidos permiten concluir que la implementación paralela del algoritmo de detección de bordeado representa una mejora significativa frente a su versión secuencial. La reducción del tiempo de ejecución y el *speedup* alcanzado demuestran que este tipo de tareas, al estar compuestas por operaciones independientes por píxel, se benefician de manera notable del paralelismo.

El enfoque paralelo no solo acelera el procesamiento, sino que también permite escalar el sistema hacia aplicaciones más complejas, como análisis en tiempo real, procesamiento de video o sistemas de visión computacional de gran volumen.