Taller Práctico 2: Detección de bordeado mediante algoritmos paralelos (SOBEL ALGORITHM)

William Aristizabal - Juan Joya - Ana Hernandez

1. 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 de este laboratorio será un informe que demuestre el speedup (aceleración) logrado con el paralelismo.

2. Marco Teórico:

La detección de bordes es una técnica esencial dentro del procesamiento digital de imágenes, ya que permite identificar regiones donde ocurre un cambio brusco en la intensidad. Estos cambios representan los límites de los objetos, por lo que los bordes son fundamentales para tareas como segmentación, reconocimiento de patrones, visión artificial y análisis de formas.

2.1. Bordes como Gradientes

Una imagen puede considerarse como una función bidimensional de intensidad I(x,y).
Un borde corresponde a un cambio fuerte en esta función, y dicho cambio se mide mediante el gradiente, que indica la dirección y magnitud del mayor cambio de intensidad.

El gradiente tiene dos componentes principales:

- Gx: mide cambios de intensidad en la dirección horizontal y detecta bordes verticales.

- Gy: mide cambios en la dirección vertical y detecta bordes horizontales.

Valores altos en estas derivadas indican la presencia de un borde fuerte.

2.2. Convolución y el Operador Sobel

La detección de bordes se realiza mediante la operación de convolución, que consiste en aplicar un pequeño filtro o *kernel* sobre cada píxel y sus vecinos.

El operador Sobel utiliza dos kernels 3×3 que aproximan las derivadas primera de la imagen:

- El kernel Kx resalta bordes verticales.
- El kernel Ky resalta bordes horizontales.

Los valores centrales del kernel tienen pesos mayores, lo que hace que Sobel sea más sensible a los cambios cercanos al píxel analizado y más robusto al ruido que otros operadores como Roberts o Prewitt.

2.3. Magnitud del Gradiente

Las componentes Gx y Gy se combinan mediante la norma euclidiana:

$$G = \sqrt{Gx^2 + Gy^2}$$

Este valor representa la fuerza del borde en ese punto.
Los píxeles con valores altos corresponden a bordes bien definidos, mientras que valores bajos representan regiones uniformes de la imagen.

3. Metodología:

3.1. Datos:

Para la experimentación se utilizó una imagen en formato PNG correspondiente a un brócoli, almacenada en escala de grises.
La ruta utilizada en las pruebas fue:

C:/Users/ADMIN/TrabajosHPC/Imagenes/brocoli1.png

La imagen se usó como entrada tanto para la versión secuencial como para la versión paralela del algoritmo Sobel, garantizando condiciones equivalentes de comparación.

3.2. Configuración de Hardware:

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

- Procesador: Intel Core i5 (8 núcleos, 2.4 GHz)
- Memoria RAM: 16 GB
- Tipo de CPU: 64 bits
- Número de hilos disponibles: 8
- Sistema operativo: Windows 10

Estas características permiten ejecutar múltiples procesos en paralelo y observar cómo el rendimiento varía al incrementar el número de ciudades o procesos.

3.3. Configuración de software:

- Lenguaje: Python 3.10.6
- El entorno de ejecución fue VS Code.

- Librerías utilizadas:
    - numpy → manejo de matrices y cálculos numéricos.
    - itertools → generación de permutaciones de rutas.
    - matplotlib → graficar la mejor ruta.
    - concurrent.futures → gestión del paralelismo con ProcessPoolExecutor.
    - time → medición de tiempos de ejecución.

3.4. Explicación de los Algoritmos Desarrollados

3.4.1. Carga de Imagen

La función load_image() utiliza cv2.imread() para cargar la imagen en escala de grises.

Esto asegura que el análisis se realice sobre una matriz 2D, ideal para la aplicación del filtro Sobel.

3.4.2. Sobel Secuencial

El algoritmo secuencial implementa manualmente el operador Sobel usando dos kernels:

- **Kx:** detecta bordes verticales
- **Ky:** detecta bordes horizontales

El proceso se realiza pixel por pixel:

1. Para cada posición, se extrae una ventana 3×3.
2. Se calcula:

$$Gx = \sum(Kx \cdot Región)$$

$$Gx = \sum(Kx \cdot Región)$$

3. Se calcula la magnitud del gradiente:

$$G = \sqrt{Gx^2 + Gy^2}$$

4. Se normaliza la imagen final.

Este método es el que más tiempo requiere debido a que recorre toda la imagen de manera secuencial y aplica operaciones matemáticas intensivas.

3.4.3. Sobel Paralelo

El algoritmo paralelo divide la imagen en bloques horizontales y distribuye cada bloque entre múltiples procesos usando:

multiprocessing.Pool

El proceso consiste en:

1. Dividir la imagen en **N bloques**, donde N es el número de procesos.
2. A cada proceso se le asigna un fragmento de la imagen junto con los kernels Sobel.
3. Los procesos ejecutan la función sobel_worker() que calcula Sobel localmente.
4. Se agregan filas de *solapamiento* para evitar errores en las fronteras.
5. Los resultados parciales se unen para reconstruir la imagen final.
6. Se normaliza la salida.

Este diseño aprovecha múltiples núcleos del procesador, reduciendo significativamente el tiempo de ejecución.

3.4.4. Medición del Desempeño

Para ambas versiones del algoritmo se usa time.perf_counter() o time.time(), con el fin de medir con precisión el tiempo de ejecución y comparar rendimiento entre:

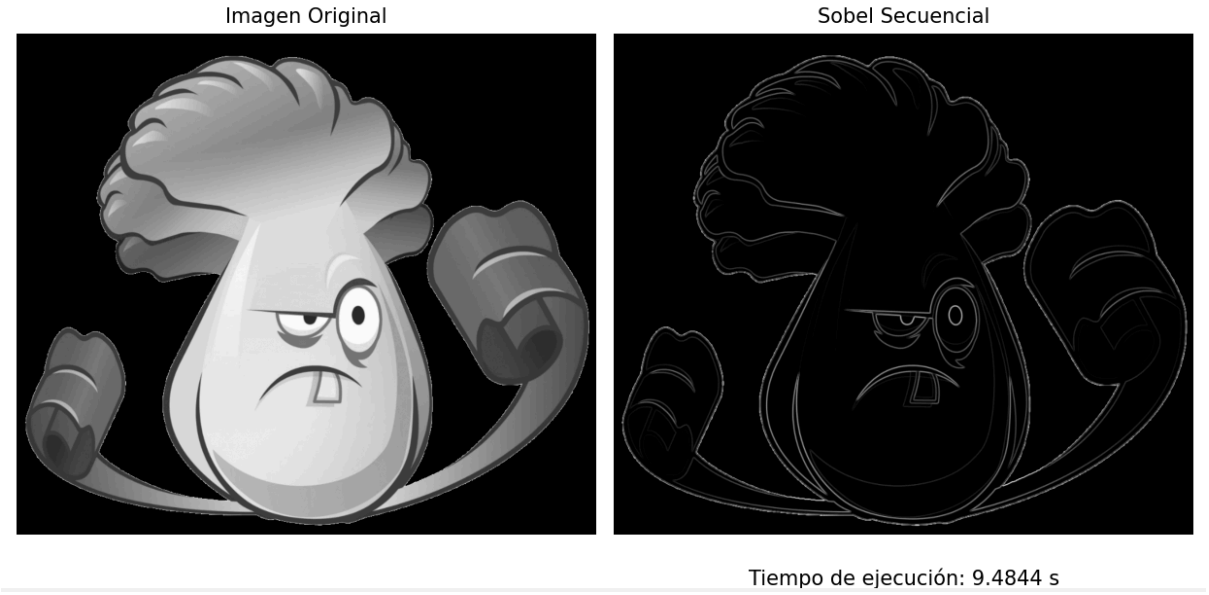
- **Sobel Secuencial**
- **Sobel Paralelo con 4 procesos**

4. Resultados:

Para evaluar el rendimiento de los algoritmos implementados, se aplicó el filtro Sobel sobre la misma imagen utilizando dos enfoques diferentes: un algoritmo secuencial y un algoritmo paralelo basado en multiprocesamiento. Ambos métodos procesaron la misma entrada y se midió el tiempo total de ejecución utilizando la biblioteca time de Python.

Los resultados muestran diferencias importantes en el desempeño de cada enfoque:

Algoritmo Secuencial:

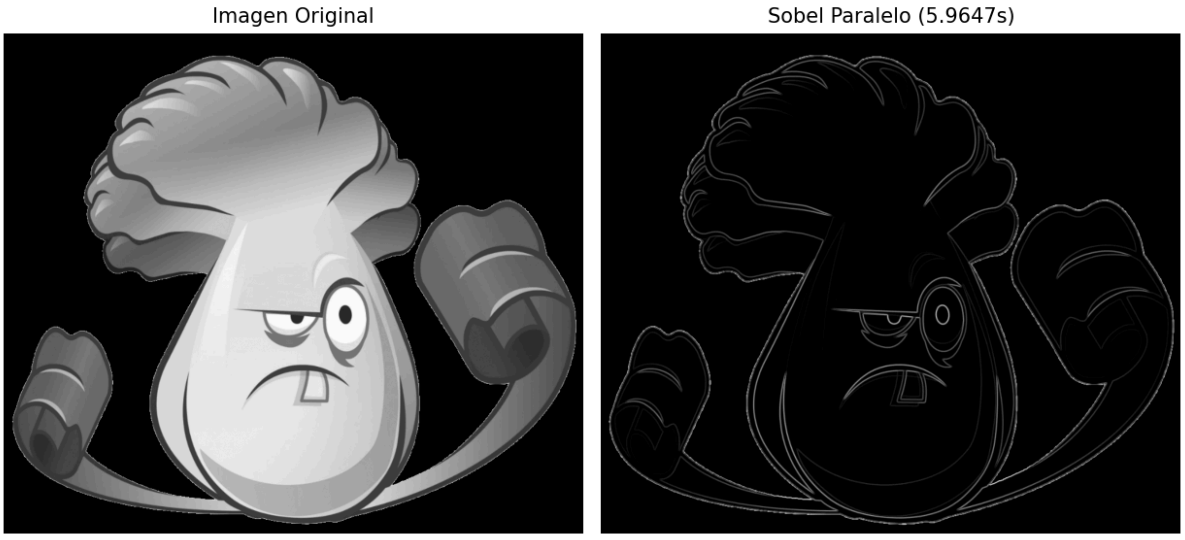


El algoritmo secuencial recorre pixel por pixel la imagen y aplica manualmente las convoluciones del operador Sobel con los kernels Kx y Ky. Este enfoque no realiza ningún tipo de paralelización y depende completamente del desempeño de un solo núcleo del procesador.

- Tiempo de ejecución: 9.48 segundos

Este resultado evidencia que, al calcular cada gradiente de forma individual, el proceso requiere un tiempo considerable, especialmente debido al doble bucle anidado que recorre toda la matriz de la imagen.

Algoritmo Paralelo:



El algoritmo paralelo divide la imagen en varios bloques horizontales y asigna cada bloque a un proceso independiente usando multiprocessing.Pool. Cada proceso calcula el Sobel localmente y luego los resultados se combinan para generar la imagen final.

- Número de procesos: 4
- Tiempo de ejecución: 5.96 segundos

El tiempo obtenido demuestra una mejora significativa respecto a la versión secuencial, reduciendo el tiempo total en aproximadamente un 37%.

Esta diferencia se debe al aprovechamiento de múltiples núcleos del procesador, lo cual permite distribuir la carga computacional y acelerar el cálculo de los gradientes.

5. Análisis de Rendimiento

El objetivo de esta evaluación fue comparar el desempeño entre dos implementaciones del operador Sobel: una versión secuencial y una versión paralela usando multiprocesamiento. Los resultados obtenidos permiten analizar cómo se comporta cada enfoque y bajo qué condiciones uno puede ser más eficiente que el otro.

5.1. Diferencias de rendimiento entre los enfoques

El algoritmo secuencial tardó 9.48 segundos, mientras que el algoritmo paralelo tardó 5.96 segundos, lo que representa una reducción aproximada del 37% en el tiempo de ejecución.

Esta mejora indica que la paralelización logra aprovechar la arquitectura multinúcleo del procesador para dividir el trabajo, procesar diferentes partes de la imagen simultáneamente y acelerar la computación del gradiente Sobel.

5.2. Factores que explican la mejora del algoritmo paralelo

a) Distribución de la carga de trabajo

La imagen se divide en cuatro bloques horizontales, cada uno procesado por un núcleo distinto.
Esto permite que varias convoluciones se calculen en paralelo, reduciendo significativamente el tiempo total.

b) Aprovechamiento del hardware

El equipo utilizado (Acer Nitro 5) cuenta con múltiples núcleos físicos, lo que favorece el uso de multiprocessing.

El algoritmo paralelo obtiene mejor rendimiento justamente porque:

- El cálculo del filtro Sobel es computacionalmente intensivo
- Es una tarea altamente paralelizable
- No depende de valores de otros bloques para cada pixel (salvo el pequeño borde de solapamiento)

c) Disminución del cuello de botella

En el algoritmo secuencial, un único núcleo realiza todo el trabajo.
Esto causa un cuello de botella cuando la imagen es grande o la operación es costosa.

El paralelismo rompe este límite al permitir que varios núcleos trabajen simultáneamente.

5.3. Costos y limitaciones del algoritmo paralelo

Si bien el algoritmo paralelo es más rápido, no toda la mejora proviene únicamente del procesamiento en paralelo. También intervienen ciertos costos adicionales, como:

- Creación de procesos
- División de la imagen en bloques
- Serialización y envío de los datos a cada proceso
- Ensamblaje final de los resultados

En este caso, los beneficios superan los costos, pero en imágenes muy pequeñas o en máquinas con pocos núcleos, la versión paralela podría ser incluso más lenta que la secuencial.

5.4. Relación entre tamaño de la imagen y rendimiento

A medida que la carga computacional crece, el algoritmo paralelo se vuelve más eficiente, pues hay más trabajo que distribuir entre los núcleos.

Si se usaran imágenes más grandes o se aumentara el tamaño del kernel, la brecha del rendimiento probablemente sería aún mayor y la versión paralela sería más ventajosa.

6. Conclusiones:

- El procesamiento paralelo demuestra una mejora significativa en el rendimiento respecto al enfoque secuencial, reduciendo el tiempo de ejecución de 9.48 s a 5.96 s. Esto evidencia que la paralelización es una estrategia efectiva para tareas intensivas como la convolución del operador Sobel.

- La implementación secuencial, aunque funcional y precisa, no aprovecha las capacidades multinúcleo del hardware, por lo que su rendimiento se ve limitado cuando aumenta la carga computacional.

- El costo asociado a crear procesos, dividir la imagen y unir los resultados no supera las ventajas del paralelismo, lo que confirma que, para imágenes de tamaño considerable, el uso de multiprocesamiento es una solución eficiente.

- A mayor tamaño de imagen, mayor será la ventaja del enfoque paralelo, ya que existe más carga de trabajo para distribuir entre los núcleos. Esto sugiere que el paralelismo es especialmente recomendable para aplicaciones de visión por computadora en tiempo real o para el procesamiento masivo de imágenes.

- Ambos métodos producen resultados visuales equivalentes, demostrando que la paralelización no afecta la calidad del borde detectado, sino únicamente el rendimiento.

- Finalmente, el estudio confirma que la ejecución paralela es más adecuada en escenarios donde se requiere rapidez o se trabaja con grandes volúmenes de datos, mientras que la versión secuencial puede ser útil para análisis simples o pruebas iniciales debido a su sencillez de implementación.