Informe Parcial 1

Laura Isabel Olivero

31/08/2025

Índice general

0.1. Introducción

En este trabajo se desarrollaron difentes versiones de un programa para aplicar filtros de procesamiento digital de imágenes en formato PGM y PPM.

Repositorio: https://github.com/isabel-olivero/Parcial1Paralela

video: https://youtu.be/I99cYXaTOKM
MPI: https://youtu.be/QdIHXN9YSFM

Los filtros implementados fueron:

- **Blur (difuminado)**
- **Laplace (detección de bordes)**
- **Sharpening (enfoque/nitidez)**

El objetivo es comparar el rendimiento de las versiones:

1. Implementación **secuencial en C++**.
2. Implementación **paralelizada con Pthreads**.
3. Implementación **paralelizada con OpenMP**.
4. Implementación **paralelizada con MPI**.

0.2. Tiempos de ejecución

Los tiempos medidos se resumen en la siguiente tabla:

2***Implementación**	**Blur**		**Laplace**		**Sharpen**	
	Total	**CPU**	**Total**	**CPU**	**Total**	**CPU**
Secuencial	4.882s	3.839s	4.432s	2.989s	4.683s	3.893s
Pthreads	3.878s	3.301s	4.553s	3.839	5.197s	4.016s

Cuadro 1: Tiempos totales y en CPU de cada filtro en las implementaciones secuencial y con Pthreads.

Implementación	**Tiempo total**	**Tiempo en CPU**
OpenMP	4.809s	8.484s

Cuadro 2: ya que la version de OpenMP genera los tres archivos de forma simultanea no se incluye en la tabla 1

0.3. conclusión

El análisis de los resultados obtenidos permite identificar diferencias claras entre los enfoques:

- La **implementación secuencial** fue la más sencilla de desarrollar y entender, sin embargo, debido a que el procesamiento se realiza de manera lineal, sin aprovechar la capacidad de paralelismo que ofrecen los procesadores modernos. Aunque cumple con la funcionalidad requerida, no es adecuada para imágenes de gran tamaño ni para aplicaciones en tiempo real.
- La versión con **OpenMP** mostró una ventaja significativa. Gracias a las directivas de paralelización, el procesamiento de píxeles se distribuyó entre múltiples hilos internos gestionados automáticamente por el compilador y el entorno de ejecución. Además, esta versión se diseñó para aplicar los tres filtros (*blur*, *laplace* y *sharpen*) en secciones paralelas independientes, lo que permite que se generen los tres archivos de salida en forma simultánea. Este enfoque no solo redujo los tiempos de ejecución, sino que también optimizó el flujo de trabajo al obtener todos los resultados en una sola corrida del programa.
- Para la implementación con **Pthreads** la estrategia se basó en dividir la imagen en cuatro regiones fijas (una por cada hilo), lo cual puede generar limitaciones al trabajar con arquitecturas que disponen de más núcleos. Aunque otorga un mayor control sobre los hilos y la memoria compartida, su complejidad de programación es mayor y los beneficios en rendimiento fueron menores en comparación con OpenMP.

Finalmente respecto a la pregunta de interes planteada se puede decir que la programación paralela reduce de forma significativa el tiempo de ejecución en comparación con la implementación secuencial. En particular, OpenMP mostró el mejor desempeño al aprovechar múltiples núcleos y generar los tres filtros en simultáneo, mientras que Pthreads logró mejoras más limitadas debido a la división fija de hilos. MPI, aunque útil en entornos distribuidos, no resulta tan eficiente en un entorno local simulado por la sobrecarga de comunicación. En conclusión, la paralelización con OpenMP ofrece el mayor beneficio práctico frente a la versión secuencial.