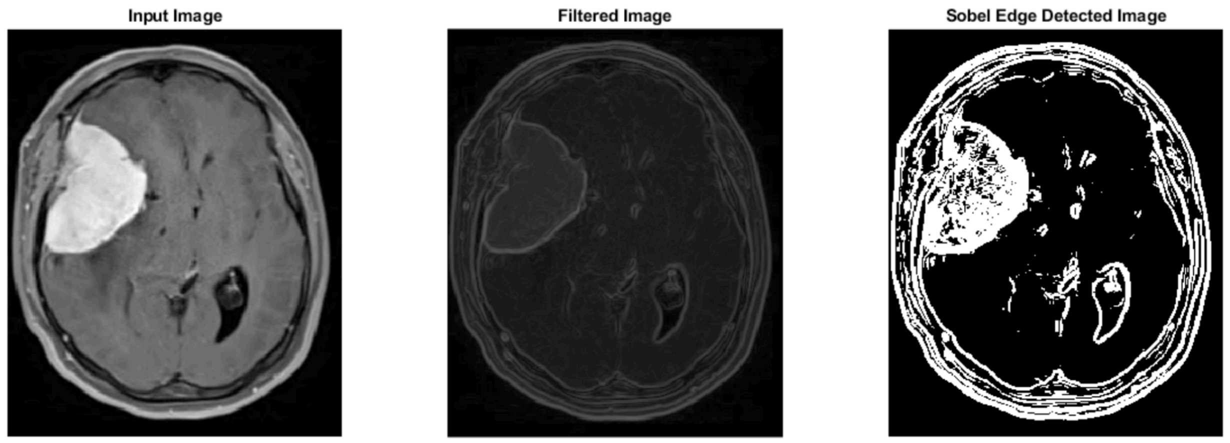
CMSC5702 (Advanced Topics in Parallel / Distributed Systems)

Assignment 1: Parallel Programming with Sobel Filter



Objective: Implement the Sobel operator for edge detection on grayscale images, and compare performance between sequential, shared-memory (OpenMP), and distributed-memory (MPI) implementations. Optionally include a blurring step to reduce noise before edge detection.

Problem Description: The Sobel filter computes image gradients to detect edges:

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * I, \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * I$$

where (I) is the input grayscale image. The gradient magnitude at each pixel is:

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

Requirements

1. Sequential Version (10%)

- Implement a standard sequential C program to apply the Sobel operator on a grayscale image.
- If including blurring, apply an average filter first (e.g., 3×3 kernel) before Sobel.
- Your code should receive an argument for the sample size.

2: OpenMP Programming (40%)

Implement a parallized version of Sobel filter program in C for edge detection on square grayscale images using OpenMP. You may:

- Process large images in tiles (e.g., 4000×4000 or 16000×16000) to reduce memory bandwidth issues.
- [Bonus Point] Optionally include a blurring step before the Sobel operator to reduce noise.
    - Definition of the 3×3 Average Filter: The 3×3 average (or **mean**) filter is a **linear spatial filter** that smooths an image by replacing each pixel with the **average intensity** of its 3×3 neighborhood.

$$K = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

- Your code should receive an argument for the sample size.

Measure performance for different thread counts (1, 2, 4, 8, 16, 32) and image sizes (256×256, 1024×1024, 4000×4000, 16000×16000). Repeat runs 3 times and fill out the csv template openmp_times.csv based on your outputs. You are also required to explain your method and logic of your code in the report template (Jupyter Notebook file).

Part 3: MPI Programming (50%)

You are required to implement the Sobel filter for edge detection on large images using MPI. You are required to:

- Choose a domain decomposition strategy, such as row-wise decomposition, column-wise decomposition, etc., decomposition. Your choice will affect performance, and better-designed decomposition and communication patterns will receive higher marks.
- For each process, you may need to include some approaches to allow seamless computation for the edges.
- [Bonus Point] You can include If a parallelized blurring step and you may need to try to optimize the communications.
- Verify your implementation using small test patterns (5×5 or 10×10) and visually inspect sample images for correctness.

You are required to measure performance with 1, 2, and 4 processes on one node and 8, 16, and 32 processes on 4 nodes using the samples provided. Your code should receive an argument for the sample size. Each run must be repeated three times, and execution times must be recorded in a CSV file named mpi_times.csv.

Reporting (Jupyter Notebook)

You are required to fill out the Jupyter Notebook template to summarize your results. The notebook will read data from three CSV files:

1. `openmp_times.csv` – OpenMP execution times.
2. `mpi_times.csv` – MPI execution times.
3. `verification.csv` – Verification results for correctness.

You also need to write a short description of your code and the methods you used to improve the algorithm. Optionally, you may include images showing Sobel-filtered outputs for verification.

Verification

- Sequential: check gradient results on a small test pattern (5×5 or 10×10).
- OpenMP: compare results with the sequential program.
- MPI: compare results with the sequential program.
- Optional: visually inspect edge detection on sample images.
- Recommended: include a few images (original + Sobel-filtered) in your report.

Performance Measurement

- Provided image sizes: 256×256, 1024×1024, 4000×4000, 16000×16000. You can use the `download_samples.py` script to download the 4K and 16K samples. You can also use `pgmio.h` code to read/write images. In the `test_pgmio.c` file, we have provided an example of how to read/write an image.
- Number of threads/processes:
    - OpenMP: 1, 2, 4, 8, 16, 32 threads
    - MPI: 1, 2, 4 processes 1 node, 8, 16, 32 processes on 4 nodes.
- Measure time: Please refer to Slide 41 of tutorial for more details
- Optional: include visual examples of filtered images for verification.

Submission

- **Source code** for sequential, OpenMP, and MPI implementations (well-commented).

    - You are required to follow this specific program naming scheme: sobel.c, sobel_omp.c, sobel_mpi.c or sobel.cpp, sobel_omp.cpp, sobel_mpi.cpp
- **Jupyter Notebook** report.
- **Charts** as described above and samples in the report.
- **Optional:** sample filtered images.

References

- YouTube: Sobel Edge Detection
- Medium: How Image Edge Detection Works