COMS 4240 Final Project Proposal

Justin Wenzel

Jwwenzel@iastate.edu

NetID: jwwenzel

Student ID#: 238716493

11/8/24

Problem:

Edge detection filters in image processing and computer vision is a technique to help identify the boundaries of objects in an image. Using edge detection to highlight areas of high-intensity change can assist in object recognition or other fields, such as autonomous vehicles and other machine learning operations performed on images. While multiple classic edge detection techniques consist of the Sobel and Prewitt filters, which compute the gradient of intensity of an image in different regions with high contrast. These filters perform well in identifying edges, but the problem with the filters involves high-resolution images. Processing high resolution images requires a lot of computations because every pixel must be compared with reference to its surrounding neighbors in the image.

This project will show how the high computation requirement that slows down the process of applying filters to an image can be sped up using parallelization. Using OpenMP and MPI to take advantage of multi-core and thread processing, the goal is to implement efficient edge detection and enhance performance speed through parallelization. For efficient edge detection, the Sobel and Prewitt filters will be applied to an image to help compare the potential differences in edges each filter detects. For enhanced performance through parallelization, OpenMP and MPI will be used to reduce processing time by applying both filters concurrently and parallelizing pixel-level computations.

In the end, this project aims to use parallelization to speed up processing large images more efficiently, helping improve the opportunity for real-time edge detection in systems as edge detection plays a crucial role in multiple implementations involving computer vision and image processing applications.

Implementation Plan:

The program's implementation will be done by following the steps outlined below. The development will start with a conventional single-threaded approach, then be expanded with OpenMP, and finally with MPI instead of OpenMP to show the difference in performance for all implementations. The program will consist of Python and C code to assist in the input image handling, processing, and output image handling.

Steps:

Step 1 Convert Image to Binary Data: Using Python a path to an image will be specified with the width and height of the image. The script will convert the image into an array format stored in a binary file for easier data handling in C. After converting the image, the Python script will save the binary to a specified folder.

Step 2 Execute C code for Edge Detection: The C program will read the binary image and allocate memory for the output. It will apply the Sobel, and Prewitt filters across the input image using convolution and saves the results to an output binary.

Step 3 Convert Binary Data to Image: Using Python and a path to the altered binary data from the C program output will be specified with the width and height of the original image. The script will convert the binary data into an image for visual purposes. After the binary data is converted, the resulting image will be saved in a specified folder.

Steps Combined:

All these steps will be combined to run in a single terminal command that begins with a Python script performing step 1. The script will then use the subprocess() function to execute step 2, the C program, and pass the required inputs to its execution. Once the C program executes, the Python script will continue to step 3 to turn the C program output back into an image.

Parallelization:

1.

OpenMP will be used to parallelize the C program while applying the Sobel and Prewitt filters by using OpenMP to run the two filters concurrently. Additionally, OpenMP will also be used to parallelize the pixel-level computation for each filter. This will create two levels of parallelization that threads can leverage, both distinct filters and pixel-level computations, as each pixel output is not dependent on another. They can be completed in parallel.

2.

MPI will be used in the C program by dividing the image into blocks, and each MPI process will receive a section of the image to apply the filter independently. Each process will still apply both filters to the given image section. After execution, all processes will return their section to be combined to construct the final resulting image.

Expected Results:

The implementation is expected to show a clear improvement in processing speed by utilizing the benefits from both OpenMP and MPI compared to the single-thread baseline implementation. I expect the OpenMP implementation to significantly improve runtime due to the filters and pixel computations being run concurrently. I also expect the MPI to provide a significant improvement when it has multiple nodes available to disperse the concurrent workload. Specific values cannot be inferred, but the result will compare the single thread baseline, OpenMP, and MPI final computational runtime.