Debugging Team

Image Processing Filters in Parallel

Introduction

The "Image Processing Filters in Parallel" project aims to leverage parallel processing techniques, specifically using MPI (Message Passing Interface), to enhance the efficiency of various image processing tasks. The project addresses the need for faster image processing by distributing computational tasks across multiple processes or nodes, thereby reducing the overall execution time.

By implementing image processing algorithms in a parallel computing environment, the project seeks to achieve significant speedup compared to traditional sequential processing methods. Key image processing tasks include Gaussian blur, edge detection, histogram equalization, image rotation, image scaling, color space conversion, global thresholding, local thresholding, image compression, and median filtering.

The project's overarching goal is to demonstrate the effectiveness of parallel processing in accelerating image processing tasks.

Objective

The main objectives of the "Image Processing Filters in Parallel" project are as follows:

1. Implement Parallel Processing.
2. Reduce Execution Time.
3. Evaluate Scalability.
4. Implement a Range of Filters.
5. Provide Comparative Analysis.

Technologies Used

- Programming Languages: c++
- Libraries and Frameworks: OpenCV, MPI

Implementation Details

1. Image Processing Algorithms: Various image processing algorithms are implemented using the OpenCV library. These algorithms include:

- Gaussian Blur: Applies a Gaussian blur filter to the image to reduce noise and detail.
- Edge Detection: Detects edges in the image using techniques like Sobel or Canny edge detection.
- Histogram Equalization: Enhances the contrast of the image by equalizing its histogram.
- Image Rotation: Rotates the image by a specified angle.
- Image Scaling: Scales the image up or down in size while preserving its aspect ratio.
- Color Space Conversion: Converts the image from one color space to another, such as RGB to HSV.
- Global and Local Thresholding: Segments the image into binary regions based on pixel intensity thresholds.
- Image Compression: Compresses the image using JPEG compression.
- Median Filtering: Applies a median filter to the image to reduce noise.

2. Parallelization with MPI: MPI is used for parallelization to distribute image processing tasks across multiple computing nodes or processes. Key MPI functions utilized include:
    - MPI_Init: Initializes the MPI environment.
    - MPI_Comm_rank: Retrieves the rank of the current process within the communicator.
    - MPI_Comm_size: Retrieves the total number of processes in the communicator.
    - MPI_Bcast: Broadcasts data from one process to all other processes.
    - MPI_Scatter: Scatters data from one process to all other processes in chunks.
    - MPI_Gather: Gathers data from all processes into one process.
    - MPI_Wtime: Retrieves the current time for measuring execution time.
3. Image Loading and Manipulation: Images are loaded and manipulated using the OpenCV library. This includes reading images from files, converting images to grayscale, resizing, and applying various image processing filters.
4. Execution Time Measurement: The execution time of each image processing operation is measured using MPI_Wtime. This allows for the comparison of execution times between sequential and parallel implementations.
5. User Interface: A simple command-line interface allows users to choose which image processing algorithm to apply and select the number of MPI processes for parallel execution.

Functions and Modules

BroadcastImageDimensions(Mat& image, int& rows, int& cols, int my_rank);

Description

- This function is important for parallel image processing using OpenCV and MPI as it broadcasts the dimensions (rows and columns) of the input image to all processes. This ensures that each process has the necessary information about the image size before processing it.
- If the current process is rank 0, it retrieves the image dimensions and broadcasts them to all processes using MPI_Bcast.

calculateLocalHistogram(const Mat& image, vector<int>& localHist) ;

Description

- It initializes the local histogram vector with zeros and then iterates over the image pixels to calculate the frequency of each intensity value, updating the local histogram accordingly.
- Histogram calculation is a common operation in image processing for various tasks like contrast enhancement, equalization, and feature extraction.

HistogramEqualization(const Mat& image, vector<int>& localHist);

Description

- Histogram equalization is a technique used to improve the contrast of an input image using MPI for parallel processing. by redistributing the intensity values.

GaussianBlur (const Mat& image, vector<int>& localHist);

Description

1. It divides the image into segments and distributes them among MPI processes for parallel processing. Each process applies Gaussian blur to its segment of the image, and the results are gathered to reconstruct the final blurred image.

EdgeDetection (const Mat& image, vector<int>& localHist);

Description

- It function performs edge detection on an input image using the Canny edge detection algorithm. It divides the image into multiple parts based on the number of processes and applies edge detection locally to each part. The function then gathers the results to display the edge-detected image.

ImageRotation(const Mat& image, vector<int>& localHist);

Description

- It performs image rotation on an input image by rotating it 90 degrees clockwise. It divides the image into multiple parts based on the number of processes and rotates each part locally. The function then gathers the rotated parts to display the final rotated image.

ImageScaling(const Mat& image, vector<int>& localHist);

Description

- It performs image scaling by doubling the size of the input image. It divides the image into parts based on the number of processes and scales each part locally. The function then gathers the scaled parts to display the final scaled image.

ColorSpaceConversion(const Mat& image, vector<int>& localHist);

Description

- The function converts the color space of an input image from BGR to HSV (Hue, Saturation, Value). It divides the image into parts based on the number of processes and performs the color space conversion locally. The function then gathers the converted parts to display the final HSV image.

GlobalThresholding(const Mat& image, vector<int>& localHist);

Description

- The function applies global thresholding to an input image to create a binary image. It divides the image into parts based on the number of processes and performs thresholdi7ng locally. The function then gathers the thresholded parts to display the final thresholded image.

LocalThresholding(const Mat& image, vector<int>& localHist);

Description

- The function applies local thresholding to an input image to create a binary image. It divides the image into parts based on the number of processes and performs thresholding locally. The function then gathers the thresholded parts to display the final thresholded image.

Median (const Mat& image, vector<int>& localHist);

Description

- The function applies a median filter to an input grayscale image using parallel processing. It divides the image into parts based on the number of processes, applies the median filter locally to each part, and then gathers the filtered parts to display the final filtered image.

CompressImage(const Mat& image, vector<int>& localHist);

Description

- The function reads an image from a specified path, divides it into parts based on the number of processes, compresses each part using JPEG compression, and then gathers the compressed data back to process 0 to reconstruct the final compressed image.

Comparison of Local vs. Parallel Execution