1. **Title Page:**
   * Title of the report
   * Project name
   * Your name
   * Date
2. **Table of Contents:**
   * List of sections and subsections with corresponding page numbers.
3. **Abstract/Executive Summary:**
   * Brief overview of the project, its objectives, methods, key findings, and conclusions.
   * Typically 150-250 words.
4. **Introduction:**
   * Background information about the project.
   * Objectives and goals.
   * Scope and limitations.
5. **Methodology:**
   * Description of the methods and procedures used in the project.
   * Data collection techniques.
   * Tools or software utilized.
   * Any assumptions made.
6. **Results:**
   * Presentation of the findings from your project.
   * Use tables, charts, graphs, or other visual aids to enhance understanding.
   * Include both quantitative and qualitative results.
7. **Discussion:**
   * Interpretation of the results.
   * Comparison with initial objectives.
   * Addressing any unexpected outcomes.
   * Relate findings to existing literature or theories.
8. **Conclusion:**
   * Summarize the main findings.
   * Reiterate the significance of the project.
   * Discuss implications and potential future work.
9. **Recommendations:**
   * Suggest actions based on the findings.
   * Offer suggestions for further research or improvements.
10. **References:**
    * List of all sources cited in the report.
    * Follow a consistent citation style (e.g., APA, MLA).
11. **Appendices (if necessary):**
    * Additional supporting materials such as raw data, survey instruments, detailed calculations, etc.

**SHIV NADAR UNIVERSITY CHENNAI**

**KALAVAKKAM- 603110**

**HIGH POWER COMPUTING**

**PROJECT REPORT 2024**

**PARALLEL IMAGE FILTERING USING DEPEDENT AND INDEPENDENT FILTERS**

**Team Members:**

Vishwa Kumaresh 21011101055 (AI&DS-A)

Rhea Pandita 21011101100 (AI&DS-B)

**Third Year, CSE Department**

1. **Project Summary**

The aim of our project is to parallelize image filtering operations with MPI to optimize processing time for efficient image manipulation, evaluating dependent and independent filter applications. By comparing the serial and parallel runtimes for both filters, we quantify the improvements achievable through parallelization.

1. **Keywords**

MPI, Sobel Filter, Averaging Filter.

1. **Objectives:**

* Implement parallel processing techniques using MPI for image filtering tasks
* Applying dependent and independent filters to images in parallel.
* Quantify and compare the runtime performance between serial and parallel implementations for dependent and independent filters.

1. **Scope and Applications**

The project primarily focuses on implementing parallel processing techniques using MPI specifically for image filtering tasks. Both dependent and independent filters will be considered for parallelization to provide a comprehensive comparison of performance gains. Evaluation will primarily concentrate on runtime performance metrics such as speedup and efficiency.

Implementation of parallel image filtering techniques can be extended to other image processing tasks, for example, edge detection, noise reduction and feature extraction. The performance insights gained from this project can be used in further applications of parallel algorithms for image-related tasks.

1. **Dataset Descriptions**

A model like this one will typically require a large-scale dataset with atleast 50,000 images. The Fashion Product Images dataset was downloaded from Kaggle, containing 7000 training images. These images were duplicated until the dataset had 56,000 images.

1. **Implementation**

The code for this implementation has 8 important functions :

1. images\_from\_directory\_with\_filter ()
2. serial\_code()
3. images\_from\_directory\_with\_filter\_independent\_mpi()
4. parallel\_independent\_code()
5. images\_from\_directory\_with\_filter\_dependent\_mpi()
6. parallel\_dependent\_code()
7. run\_independent\_filter()
8. run\_dependent\_filter()
9. images\_from\_directory\_with\_filter ():
10. serial\_code():
11. images\_from\_directory\_with\_filter\_independent\_mpi():
12. parallel\_independent\_code():
13. images\_from\_directory\_with\_filter\_dependent\_mpi():
14. parallel\_dependent\_code():
15. run\_independent\_filter():
16. run\_dependent\_filter():
17. images\_from\_directory\_with filter() :

A screen shot of a computer code

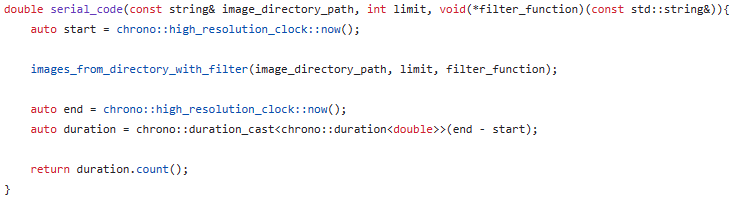
Description automatically generated

This function iterates through all the image files in the directory specified and applies a filter function to each image file.

The parameters of the function include the path to the directory, maximum number of image files to process, and a function pointer representing the filter function to be applied to each image file.

The function opens the specified directory using the ‘opendir’ function. If there an issue with loading the directory, an error message is printed, else it iterates through each image file using the ‘readdir’ function. The isImageFunction() function checks if the current file is an image or not. If it is, then it constructs the full path to the image file and reads the image into a cv::Mat object. If it is not, then prints an error message. If a filter function is provided, it applies it to the image. Then it closes the directory.

1. serial\_code()



This function measures the execution time of a serial image processing code that iterates through image files in a specified directory, applies a filter function to each image, and returns the duration of the entire process.

The parameters of the function take in the path of the image directory, maximum number of images files to process and a function pointer representing the filter function to be applied to each image.

‘chrono::high\_resolution\_clock::now()’ records the current time and assigns it to the start variable. Then the images\_from\_directory\_with\_filter() function is called to perform serial image filtering. After this the time is again recorded and assigned to the end variable. Subtracting start from end gives the duration of the operation. This value is returned back.

1. **A screenshot of a computer code

   Description automatically generated**images\_from\_directory\_with\_filter\_independent\_mpi()

A screenshot of a computer program

Description automatically generated

**A computer code with text

Description automatically generated with medium confidence**

This function implements parallel image processing using MPI for applying independent filter to images. It distributes image processing tasks among MPI processes, with each process responsible for applying the independent filter to a specific range of rows of the image.

First MPI is initialized with ‘MPI\_Init’. It retrieves the size of the communicator ‘comm\_size’ and the rank of the process within the communicator ‘rank’ using MPI\_Comm\_size and MPI\_Comm\_rank.

For root process, it opens the image directory and iterates through its contents to find image files. A vector ‘image\_paths’ is constructed containing the paths of the image files. It distributes the images among the available MPI processes by sending the image paths , start row, end row of each image to other processes. For remaining images, it directly applies the averaging filter to the specified range of rows.  
  
For other processes, the image path, start row and end row are received from the root process. Then the averaging filter is applied to the specified range of rows. Then MPI\_Finalize finalizes MPI.

D) parallel\_independent\_code()

A computer screen with text

Description automatically generated

This function measures the execution time of a parallel image processing code that applies an independent filter to images using MPI.

This function records the current time when the function starts executing. Then it executes the ‘images\_from\_directory\_with\_filter\_independent\_mpi’ function to perform parallel image filtering using MPI. Then the current time again is recorded to mark end of the operation. Duration is computed and returned back.

E)