SZABIST
UNIVERSITY

Final Project Report

Parallel Image Processing

GROUP MEMBER NAMES:

Hamza Jabbar (2212251)

Muhammad Osama (2212259)

Sadam Hussain (2212266)

Hasnain Magsi (2212252)

Abdul Rehman (2112258)

Danish (2012338)

Submitted by: Dr. Syed Samar Yazdani

Faculty of Computing and Engineering Sciences

Shaheed Zulfiqar Ali Bhutto Institute of Science and Technology, Karachi

Table of Contents

1. Introduction

This report presents a Python-based image processing system designed to apply common image filters using both sequential and parallel execution methods. The objective is to demonstrate how parallel processing can improve performance by dividing work into independent chunks and processing them concurrently, reducing processing time for large images.

2. Problem Statement

Applying image filters such as blur, edge detection, and brightness/contrast adjustment on large images is time-consuming when executed sequentially. This project aims to evaluate whether splitting an image into smaller chunks and processing them in parallel improves the performance compared to sequential processing.

3. Objectives

- Apply image filters (blur, edge detection, brightness/contrast) using Python (Pillow library).
- Implement both sequential and parallel processing methods for image processing.
- Split images into chunks with overlap handling to avoid visible seams.
- Measure execution time for both methods and calculate speedup.
- Visualize results using performance graphs.

4. Tools & Technologies

- Language: Python 3.x
- Libraries:
    - PIL (Pillow) for image manipulation.
    - Concurrent.futures (ThreadPoolExecutor) for parallel processing.
    - Matplotlib for generating performance graphs.
- Parallelism: Thread-based parallel execution using ThreadPoolExecutor.

5. Filters Used

5.1 Blur Filter

Smooths the image and reduces noise using Gaussian Blur.

5.2 Edge Detection Filter

Detects boundaries and outlines within the image, emphasizing edges.

5.3 Brightness & Contrast

Adjusts the lighting and contrast to enhance image visibility.

6. System Design

6.1 Overall Flow

The flow of the system is as follows:

1. Input Image: Load the image.
2. Split into Chunks: Divide the image into horizontal chunks for parallel processing.
3. Apply Filters: Apply the selected filter(s) to each chunk.
4. Parallel Threads: Each chunk is processed in parallel using threads.
5. Merge Chunks: After processing, chunks are reassembled.
6. Output Image: Save the processed image.

6.2 Chunking with Overlap

The image is split into horizontal chunks, and an overlap is added between the chunks to prevent visible seams or broken edges after processing.

7. Code Explanation

7.1 Filter Application Function

The apply_filter function applies the chosen filter (blur, edge detection, or brightness/contrast) to the image or image chunk.

Code:

```
def apply_filter(img, filter_name, blur=2.0, brightness=1.0, contrast=1.0):
    if filter_name == "blur":
        return img.filter(ImageFilter.GaussianBlur(radius=blur))
    if filter_name == "edges":
        return img.filter(ImageFilter.FIND_EDGES)
    if filter_name == "bright":
        out = ImageEnhance.Brightness(img).enhance(brightness)
        out = ImageEnhance.Contrast(out).enhance(contrast)
        return out
```

7.2 Sequential Processing

In sequential processing, the entire image is processed at once using a single CPU core.

Code:

```
def process_sequential(img, filter1, filter2, blur, brightness, contrast):
    out = apply_filter(img, filter1, blur, brightness, contrast)
    if filter2:
        out = apply_filter(out, filter2, blur, brightness, contrast)
    return out
```

7.3 Parallel Processing

In parallel processing, the image is split into chunks, and each chunk is processed by a separate thread to improve speed.

Code:

```
def process_parallel(img, filter1, filter2, workers=4, chunk_h=256, overlap=16, blur=2.0, brightness=1.0, contrast=1.0):
    out = Image.new(img.mode, img.size)
    chunks = build_chunks(img.height, chunk_h, overlap)

    with ThreadPoolExecutor(max_workers=workers) as pool:
        futures = [pool.submit(process_chunk, img, c, filter1, filter2, blur, brightness, contrast, overlap) for c in chunks]
        for f in futures:
            y0, strip = f.result()
            out.paste(strip, (0, y0))
    return out
```

8. Benchmarking & Performance Evaluation

Execution time is measured for both sequential and parallel modes over multiple runs, and the speedup is calculated using the formula:

8.1 Speedup Formula

Speedup = Sequential Time / Parallel Time

9. Output & Graphical Analysis

9.1 Output Images

After applying filters, two output images are generated for each input image:

- Sequential Output Image (<image_name>_seq.jpg)
- Parallel Output Image (<image_name>_par.jpg)

Both images visually appear identical, ensuring that parallel processing does not affect the correctness of the image processing, only the performance.

9.2 Output Images (Illustration)

Figure 1: Sequential Processed Image (Blur / Edge / Brightness applied)

Figure 2: Parallel Processed Image (Same filter applied using chunks & threads)

Observation: Both images look identical, confirming the correctness of parallel execution.

9.3 Graphical Output (Performance Graphs)

- Figure 3: Sequential vs Parallel Processing Time Graph
    - X-axis: Image names
- Y-axis: Processing time (seconds)

- Figure 4: Speedup Graph
    - This graph shows the speedup achieved by parallel processing.
    - Speedup = Sequential Time / Parallel Time
    - Observation: Speedup is greater than 1 for all images, proving the performance improvement.

9.4 Visual Output Observation

- Blur filter produces smooth images.
- Edge detection highlights boundaries clearly.
- Brightness/contrast improves image visibility.
- No visible artifacts due to correct overlap handling.

10. Results Summary

- Sequential processing is slower for large images.
- Parallel processing significantly reduces execution time.
- Speedup increases with the image size, demonstrating the advantages of parallel execution.

11. Real-World Applications

- Medical Image Analysis: Faster processing of medical scans.
- Satellite Image Processing: Efficient handling of large satellite images.
- AI Image Preprocessing Pipelines: Speeding up image data pipelines.
- Photo Editing Software: Real-time processing for large image editing.

12. Conclusion

This project demonstrates that parallel image processing using threads can significantly improve performance over sequential execution. By dividing images into chunks and processing them concurrently, we achieve better CPU utilization and reduced execution time, making the system suitable for large-scale image processing tasks.

13. Future Improvements

- Implement GPU-based parallel processing for further performance enhancement.
- Support additional advanced filters and processing algorithms.
- Extend the system to support video frames for real-time processing in video editing software.