2025

PROJECT REPORT

PARALLEL AND DISTRIBUTED COMPUTING

MUHAMMAD SABTAIN KHAN – B22F0827AI069

AI22-GREEN | PAF-IAST

Contents

Face Detection System for Parallel & Distributed Computing

Abstract

This project focuses on the performance comparison of sequential and parallel approaches for face detection using Python. The system processes a batch of images and detects human faces using OpenCV's Haar Cascade algorithm. The goal was to evaluate whether distributing the workload across multiple processes provides a speedup over traditional sequential execution on a low-spec machine. Results showed that for small datasets, the parallel approach introduced more overhead than benefit, resulting in slower performance. The project highlights important insights into when parallelism is effective and when it may not be ideal.

1. Introduction

Parallel and Distributed Computing (PDC) plays a key role in accelerating tasks that involve heavy computation or large datasets. Image processing is a common use-case where parallel execution is expected to improve performance by dividing tasks across multiple CPU cores.

This project implements a **face detection system** that runs in two modes:

1. **Sequential face detection**
2. **Parallel face detection using Python's multiprocessing**

The objective was to compare the execution times, analyze efficiency, and understand the behavior of parallel computing on a laptop with limited resources.

2. Problem Statement

Face detection is a compute-intensive task, especially when processing large number of images. While parallel processing can theoretically reduce the total execution time, overheads such as process creation, inter-process communication, and data loading may reduce performance on low-power systems.

Problem:

To determine whether parallel processing provides a real speedup over sequential execution for a face-detection pipeline on typical student-level hardware.

3. Objectives

The main objectives of this project are:

- To implement sequential and parallel versions of a face-detection system.
- To analyze and compare performance on CPU-bound tasks.
- To understand parallel overhead and when parallelism becomes beneficial.
- To measure total images processed, number of faces detected, and execution time for both modes.

4. Hardware Specifications

Testing was performed on the following system:

Component	**Specification**
Device Name	M-Sabtain-Khan
Processor	Intel Core i5-2450M @ 2.50 GHz (2 cores, 4 threads)
RAM	8 GB
Storage	233 GB HDD (Toshiba MK2576GSX)
Graphics	Intel HD Graphics 3000 (32 MB)
System Type	64-bit OS, x64-based processor

5. Tools & Technologies Used

Tools and libraries used in that project are following:

- **Python 3.13**
- **OpenCV (cv2)** for Haar Cascade face detection
- **Multiprocessing library** (parallel execution)
- **Pillow (PIL)** for image creation
- **Streamlit** for UI
- **OS module**, **time module**, **concurrent processing utilities**

6. Algorithms Used

This project uses:

Haar Cascade Face Detection

A machine-learning–based approach provided by OpenCV to detect faces using pre-trained classifiers.

Parallel Processing Algorithm

- Images divided into chunks
- Each process loads the Haar Cascade and performs detection independently
- Results aggregated at the end

Sequential Face Processing

- A single loop processes all images one by one

7. Methodology

The project follows these steps:

1. Dummy Image Generation

A create_dummy_images.py script generates test images with simple shapes acting as placeholders.

2. Sequential Processing

- Load image
- Apply Haar Cascade
- Detect faces
- Append results

3. Parallel Processing

- Images distributed across available CPU cores
- Each process independently performs detection
- Combine results and measure total time

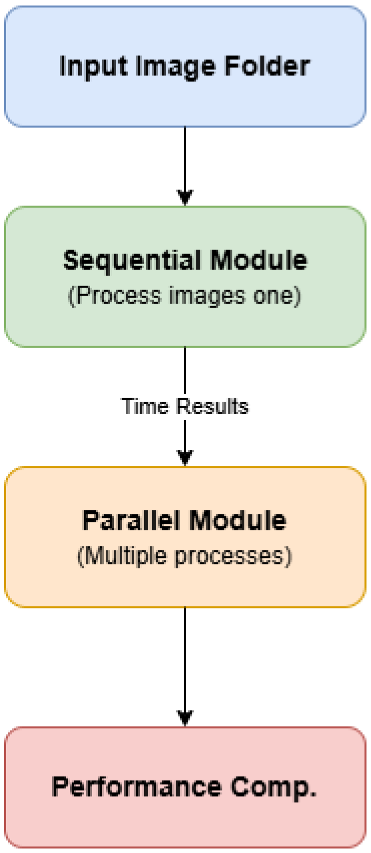
4. Performance Recording

The program prints:

- Number of images processed
- Total faces detected
- Total execution time

8. System Architecture

Below is a conceptual diagram representing the workflow:



9. Implementation Details

Main Features Implemented

- Batch image loading
- Face detection using Haar Cascade
- Multiprocessing Pool for parallel execution
- Streamlit interface for user interaction
- Time comparison between approaches

Code Structure

- *create_dummy_images.py* → Generates test images
- *main.py* → Core logic for sequential & parallel processing
- *streamlit_app.py* → UI for demonstration

- *requirements.txt* → Dependencies
- *readme.md* → Documentation

10. Performance Evaluation

Results Obtained:

- **Total images processed:** 2
- **Total faces found:** 11
- **Original number of faces:** 11
- **Execution Times**

Mode	Time
Sequential	**7.9273 seconds**
Parallel	**26.6973 seconds**

- **Parallel Speedup Calculation**

Speedup = Sequential Time / Parallel Time

Speedup = 7.92 / 26.69

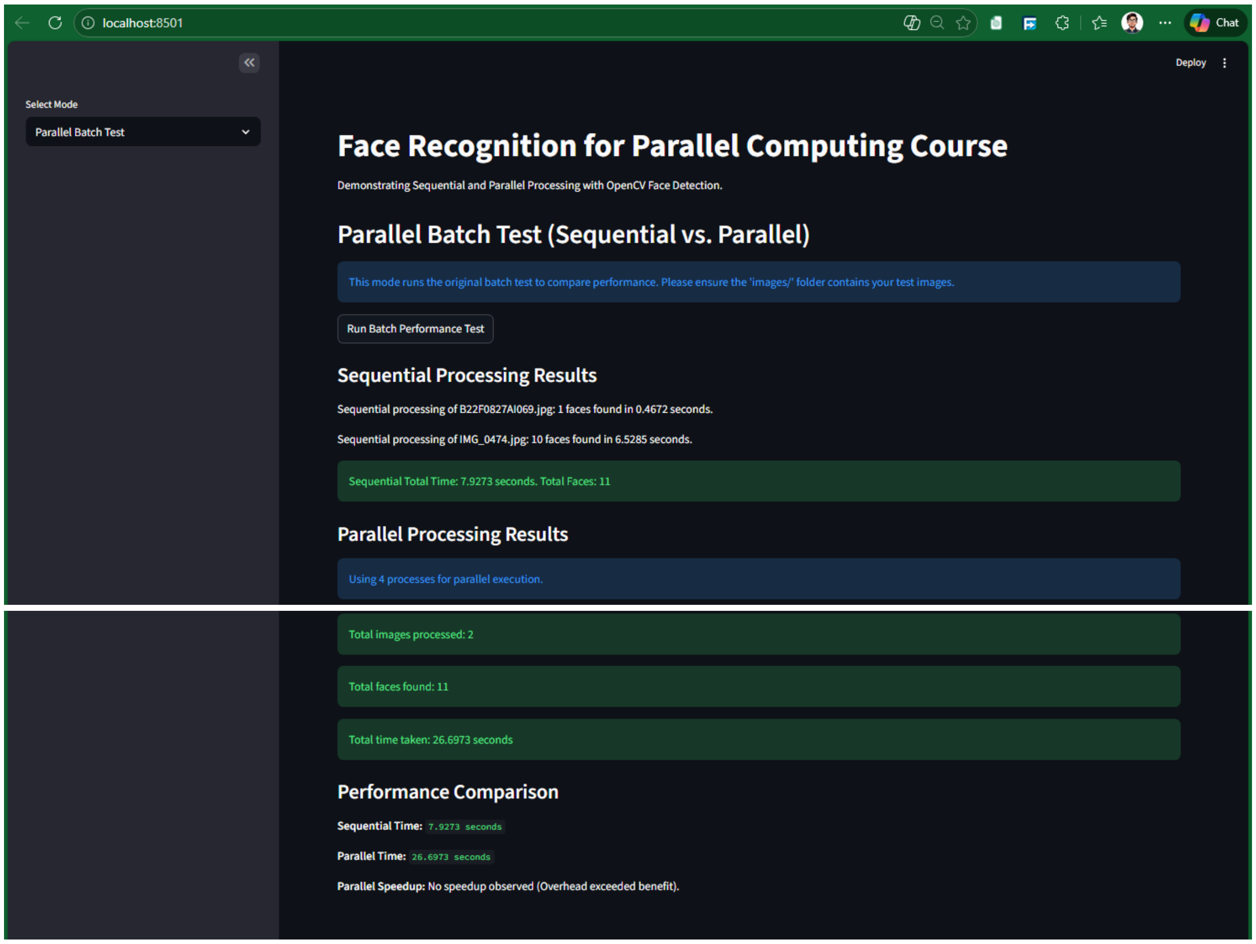
No speedup observed.

Reason for Slow Parallel Execution

- Small dataset (only 2 images)
- High overhead of creating parallel processes
- Image loading repeated per process
- Older dual-core CPU with limited multiprocessing efficiency
- HDD storage increases file loading time

Parallel execution becomes beneficial **only for large datasets**.

11. Project Visualization



Face Recognition for Parallel Computing Course

Demonstrating Sequential and Parallel Processing with OpenCV Face Detection.

Parallel Batch Test (Sequential vs. Parallel)

This mode runs the original batch test to compare performance. Please ensure the 'images/' folder contains your test images.

Sequential Processing Results

```
Sequential processing of B22F0827AI069.jpg: 1 faces found in 0.4672 seconds.
Sequential processing of IMG_0474.jpg: 10 faces found in 6.5285 seconds.
Sequential Total Time: 7.9273 seconds. Total Faces: 11
```

Parallel Processing Results

```
Using 4 processes for parallel execution.
Total images processed: 2
Total faces found: 11
Total time taken: 26.6973 seconds
```

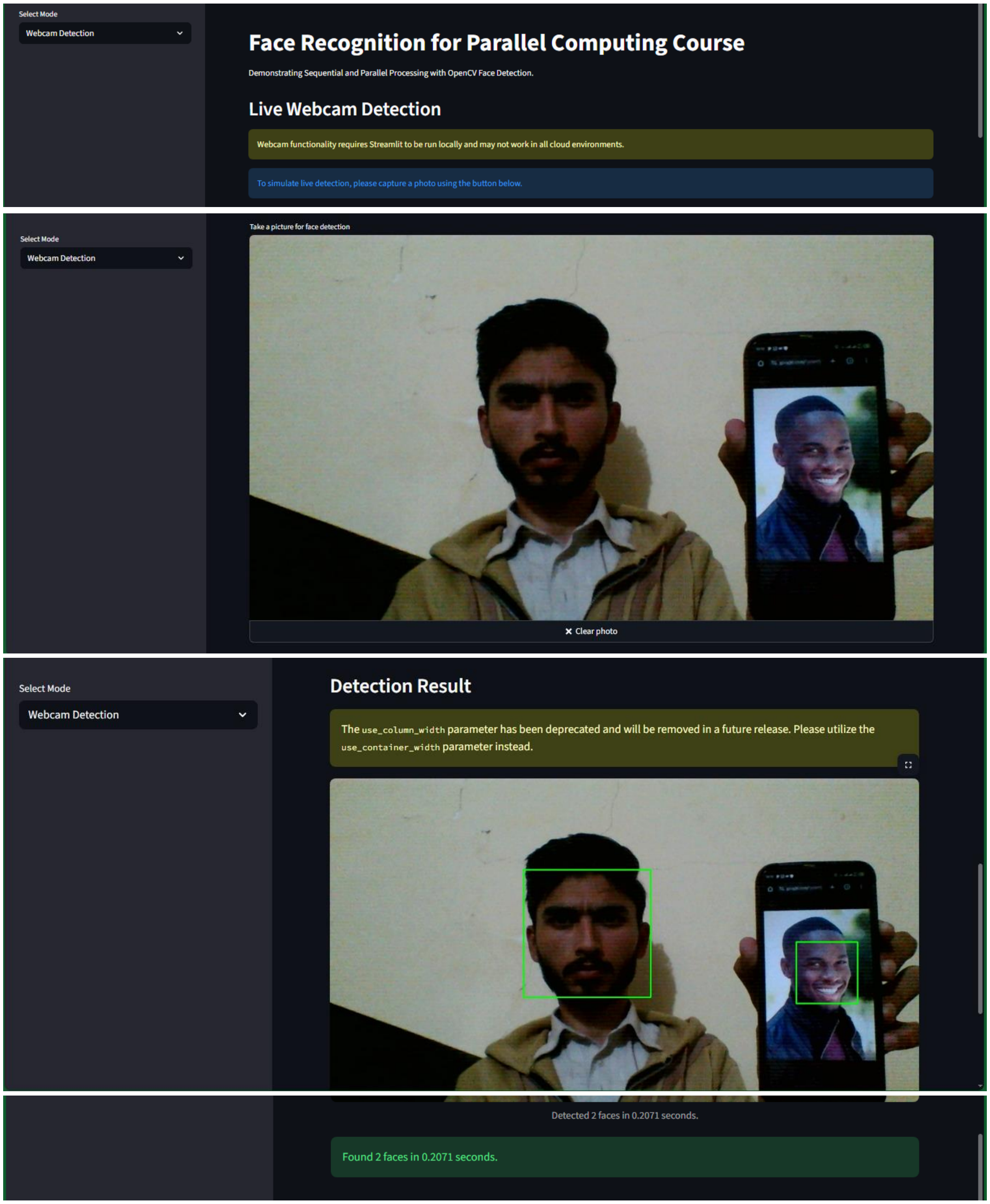
Performance Comparison

Sequential Time: `7.9273 seconds`

Parallel Time: `26.6973 seconds`

Parallel Speedup: No speedup observed (Overhead exceeded benefit).

- **Figure 1:** Sequential & Parallel processing output



Face Recognition for Parallel Computing Course

Demonstrating Sequential and Parallel Processing with OpenCV Face Detection.

Live Webcam Detection

Webcam functionality requires Streamlit to be run locally and may not work in all cloud environments.

To simulate live detection, please capture a photo using the button below.

Take a picture for face detection

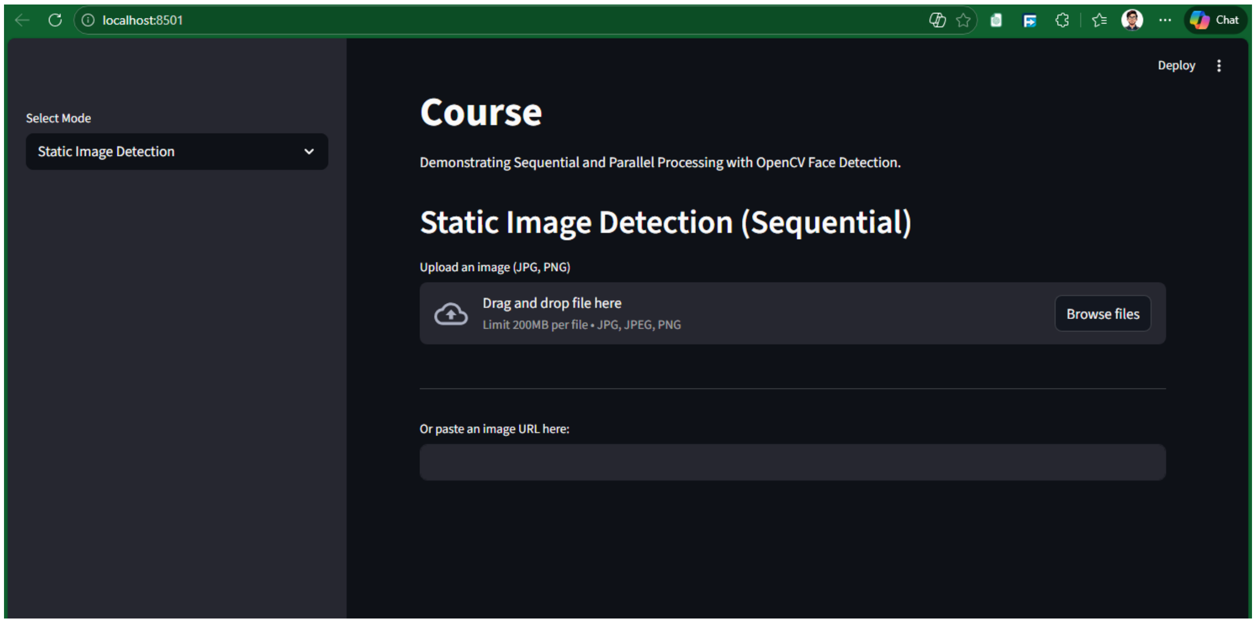
Detection Result

The `use_column_width` parameter has been deprecated and will be removed in a future release. Please utilize the `use_container_width` parameter instead.

Detected 2 faces in 0.2071 seconds.

Found 2 faces in 0.2071 seconds.

- **Figure 2: Webcam Detection**



Course

Demonstrating Sequential and Parallel Processing with OpenCV Face Detection.

Static Image Detection (Sequential)

Upload an image (JPG, PNG)

Drag and drop file here
Limit 200MB per file • JPG, JPEG, PNG

Or paste an image URL here:

- **Figure 3: Static Face Detection**

12. Conclusion

The project successfully demonstrates the comparison between sequential and parallel face-detection techniques. While parallelism is expected to reduce execution time, the results clearly show that for small datasets and low-spec hardware, parallel execution introduces overhead that outweighs performance gains. Sequential processing proved faster in this scenario.
However, the experiment provides valuable insight into how parallel computing behaves in real-world systems and highlights the importance of selecting the correct workload size before applying parallel techniques.

13. Future Work

- Test on **larger datasets** (e.g., 100+ images) to observe real speedup.
- Upgrade to **optimized multiprocessing techniques**, such as shared memory or vectorized operations.
- Experiment with **GPU-based detection** using deep learning models like SSD or YOLO for more accurate results.

14. References

- OpenCV Documentation – Haar Cascades

- Python Multiprocessing Library
- Streamlit Official Documentation
- Parallel & Distributed Computing course material