HUMAN DETECTION IN VIDEO WITH PARALLEL PROCESSING

ABSTRACT

The goal of this project is to develop an efficient and scalable solution for human face detection in videos by using parallel processing techniques. The implementation utilizes three parallelization approaches: OpenMP, MPI (Message Passing Interface), and MPI broadcast. The project involves a serial implementation, a parallelized version using MPI, and an optimized version with MPI broadcast. This report provides a detailed overview of the project, its components, methodologies, results, and future scope.

INTRODUCTION

Human face detection in videos is a task in computer vision. Processing videos for face detection can be computationally intensive, especially for large datasets. Parallel processing offers a solution to accelerate these tasks.

The objective of this project includes:

i. Implementing a serial version for human face detection in a video.
ii. Parallelizing the implementation using OpenMP and MPI (Message Passing Interface).
iii. Optimizing MPI version with MPI broadcast.
iv. Evaluating the performance of each approach.

BACKGROUND

YOLOv3 (You Only Look Once, Version 3) is a real-time object detection algorithm that identifies specific objects in videos, live feeds, or images. The YOLO machine learning algorithm uses features learned by a deep convolutional neural network to detect an object. Versions 1-3 of YOLO were created by Joseph Redmon and Ali Farhadi, and the third version of the YOLO machine learning algorithm is a more accurate version of the original ML algorithm.

METHODOLOGY

1. SERIAL IMPLEMENTATION

The implementation is based on YOLOv3 (You Only Look Once). This model has proven to be effective in real-time applications and is chosen for its balance between accuracy and speed.

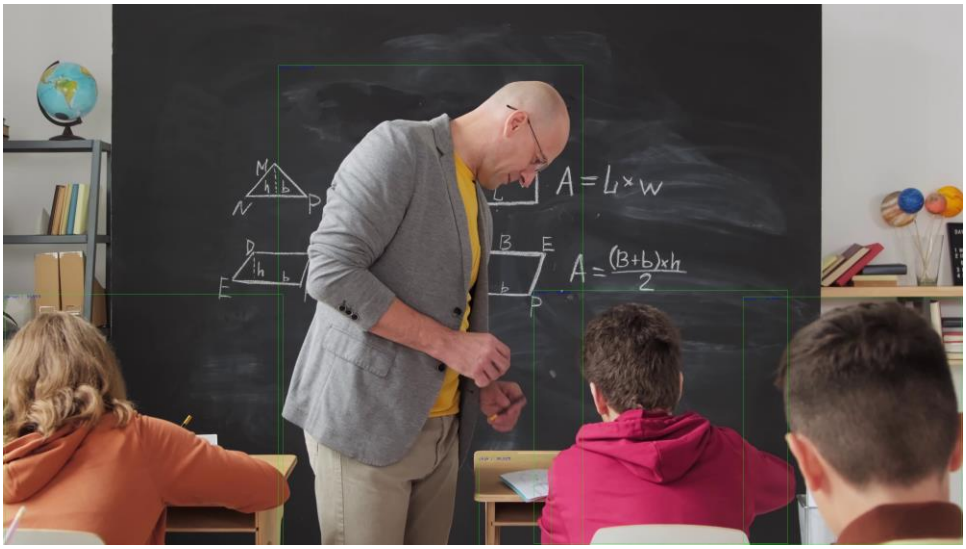
The serial implementation serves as the baseline for comparison and includes the following components:

1. Initialization: The implementation makes use of OpenCV, ImageAI library for object detection.
2. Object Detection setup:
    a. An instance of ObjectDetection class from ImageAI is created as ‘detector’.
    b. YOLOv3 is loaded from a pre-trained file ‘yolov3.pt’ and is configured for person detection.
3. Image Processing:
    a. The input video is passed to ‘preprocess_video’ function which converts the video into individual frames by calling ‘vid_to_img’. The input, ‘samplevideo’ is 18 seconds long and 491 frames are generated. These frames are named as frame{count}.jpg.

b. These frames then use ‘detect_person’ to perform person detection. It then saves the resulting images with bounding boxes around the detected persons in a folder.
c. The final video is created from the list of image file paths, which contains bounding boxes around the detected humans.

Below is the representation of a frame obtained from converting a video into a sequence of frames.

The following image is an example showing green colored boundaries around the persons detected.



2. PARALLEL IMPLEMENTATION

2.1 OpenMP

OpenMP follows a shared-memory parallelization model, where multiple threads share the same memory space. We used ‘multiprocessing’ module to use multiple cores, distributing the workload, and accelerating the frame processing.

A multiprocessing pool is created which uses the number of processes set. The ‘starmap’ method is used to apply ‘detect_person’ function to each image in parallel. After all parallel tasks are completed, the multiprocessing pool is closed. The processed frames, containing the detected persons are used to create a new video using OpenCV’s ‘Video Writer’. Each process in the pool works independently on a subset of frames, enhancing the efficiency of the video processing task.

2.2 MPI

The MPI framework enables collaborative execution among processes, with each process handling a portion of image files. We used ‘mpi4py’ python library to implement MPI.

Like the serial implementation, the YOLOv3 model is configured to detect persons in images, utilizing the ImageAI library. The script reads input video data, converts it into individual frames, and scatters these frames across MPI processes for concurrent analysis. Each process independently applies the YOLOv3 model to its allocated frames, detecting and saving human faces with bounding boxes.

After parallelized processing, the script gathers the results from all processes, combines them, filters the detections if ‘percentage probability’ is more than the confidence threshold (set to be 50%) and creates a video showcasing the detected faces with bounding boxes.

2.3 MPI BROADCAST

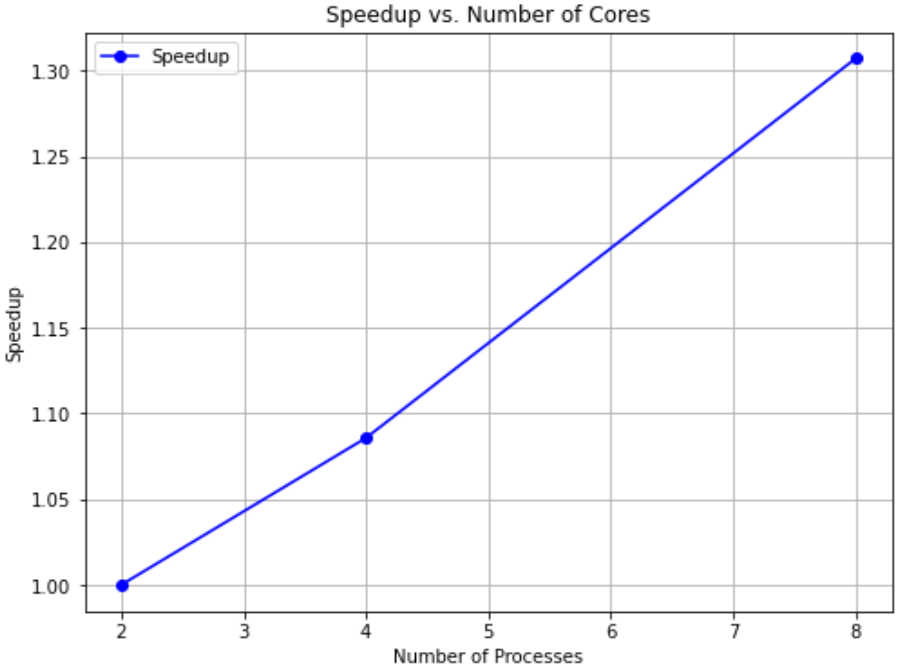
MPI broadcast optimization improves the performance by broadcasting the YOLOv3 model to all processes at the beginning, eliminating the need for each process to load the model independently. This reduces the redundancy of loading the model independently by each process, thereby avoiding the initialization overhead. By sharing the initialized model through MPI broadcast, the parallel processes gain faster access to the detector, resulting in more efficient execution times.

This optimization is particularly beneficial when dealing with computationally intensive tasks, such as object detection on video frames, where model initialization can be a bottleneck. Overall, MPI broadcast optimization contributes to a streamlined parallel execution, improving the overall efficiency of the distributed YOLOv3 model.

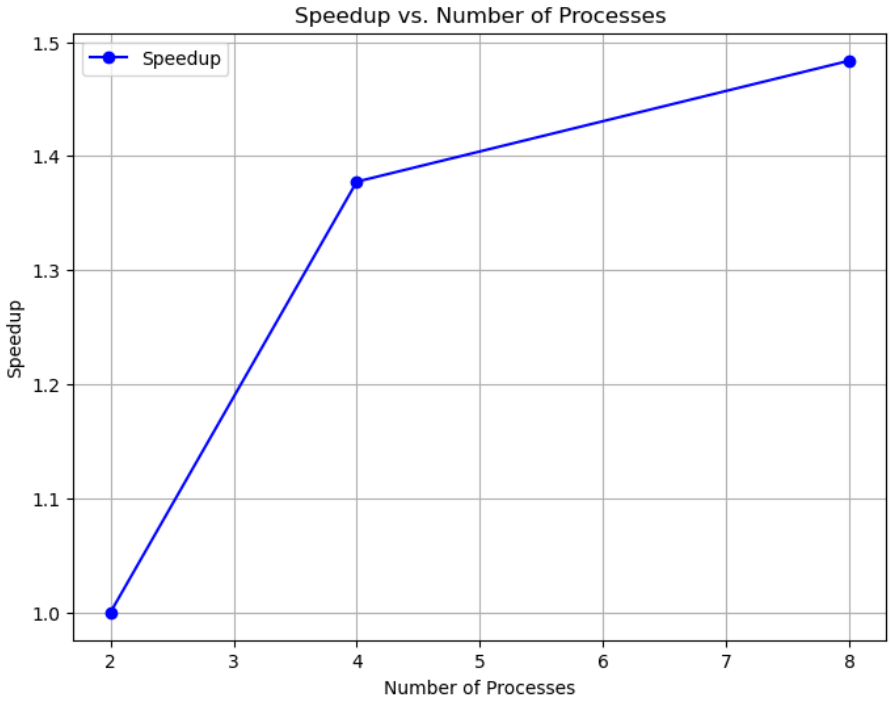
RESULTS

The code for OpenMP, MPI, and MPI broadcast have been run for processes 2, 4, and 8. The graphs for speedup vs number of processes is plotted as follows:

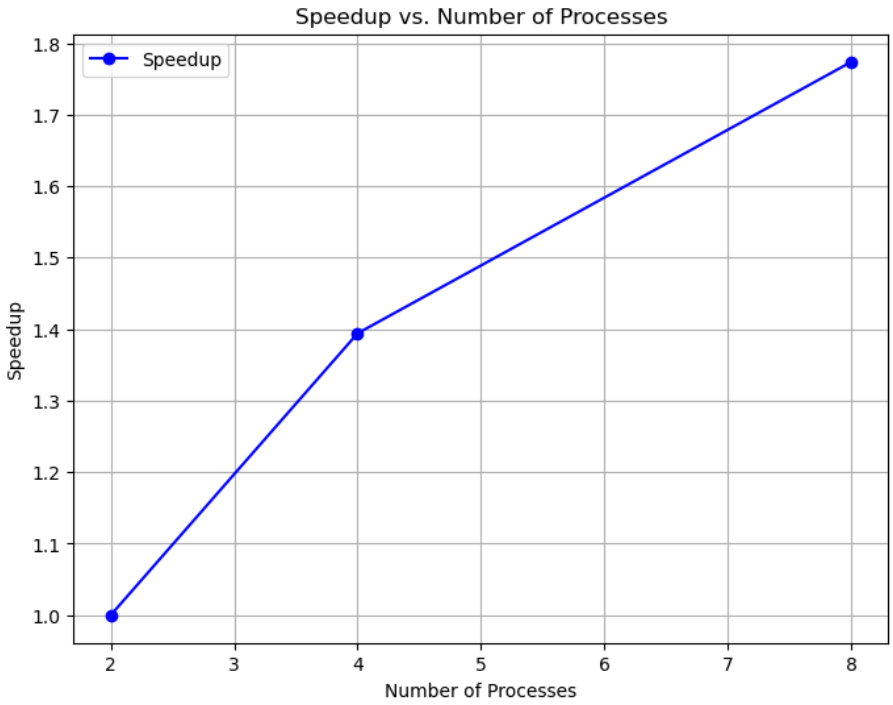
1.1 OMP


Speedup vs. Number of Cores
Speedup
1.30
1.25
1.20
1.15
1.10
1.05
1.00
Speedup
2
3
4
5
6
7
8
Number of Processes

1.2 MPI


Speedup vs. Number of Processes
Speedup
1.5
1.4
1.3
1.2
1.1
1.0
Speedup
2
3
4
5
6
7
8
Number of Processes

1.2 MPI BROADCAST



FUTURE SCOPE

We tried implementing a weak scaling approach, where we divided uneven file chunks to each process. The file chunk sizes were adjusted to increase with the number of processes. Unfortunately, this led to corruption in the generated video.

REFERENCES

1. https://imageai.readthedocs.io/en/latest/detection/
2. https://viso.ai/deep-learning/yolov3-overview/