Student Name : Sri Harsha Mudumba

Student ID : 462988584

CPRE/MATH 525 : High Performance Computing

Project Proposal: Video Processing for License Plate Detection

1. Project Title

Real-Time Video Processing and Analysis Using MPI

2. Introduction

The capacity to detect license plates efficiently and accurately in video streams is pivotal for applications such as automated toll collection, traffic monitoring, and security enforcement. With increasing video resolution and data volumes, the computational demands have also escalated, making traditional serial processing methods less effective for real-time applications. This project aims to investigate and demonstrate the efficacy of parallel computing techniques, specifically using the Message Passing Interface (MPI) on Nova clusters, to enhance the performance of video processing tasks involved in license plate detection. The primary goal of this project is to enhance processing efficiency by leveraging parallel computing resources, thereby reducing the time required to process video streams compared to traditional serial methods. The project also aims to ensure that the parallel processing method maintains or enhances the accuracy of license plate detection compared to serial methods, despite the complexities involved in managing distributed computations. Moreover, the project will provide a thorough comparative analysis of serial and parallel processing methods.

3. Methodology

To achieve these goals, the project will first implement a serial version of the video processing system to serve as a baseline for evaluating the effectiveness of the parallel approach. Subsequently, this serial approach will be adapted to a parallel processing model using MPI, involving the distribution of video processing tasks across multiple nodes within Nova clusters. The performance of both models will be measured and compared in terms of processing speed, accuracy of license plate detection, and resource utilization, with special attention given to the overhead introduced by parallelization and its impact on overall efficiency.

4. Technical Setup and process overflow

The technical execution will utilize the Nova clusters, which are equipped with MPI, to perform the parallel processing tasks. This setup will include configuring the computational environment, managing data distribution, and synchronizing processes to ensure consistent and reliable results across different nodes.

The initial step in the workflow involves loading the video into the system using OpenCV's cv2.VideoCapture() function. This function accesses the video file from a specified path, creating a video capture object that allows for frame-by-frame processing. Following the video load, each frame is processed sequentially. This crucial step involves applying various image processing techniques to accurately detect and interpret license plates. The operations performed on each frame include converting the frame to grayscale, applying filters and enhancements for image clarity, and detecting edges that indicate the presence of license plates.

After processing, the enhanced frames are compiled into a new video file, achieved using OpenCV's cv2.VideoWriter() function. This function assembles the processed frames back into a video format, adhering to specified parameters such as frame size, codec, and frame rate, which can either match the input video's specifications or be optimized according to the desired output quality.

To assess the performance and efficiency of processing methods, the code is implemented in both serial and parallel configurations. The serial implementation processes the video frames one at a time on a single processor. In contrast, the parallel implementation employs MPI to distribute the task of frame processing across multiple processors within the Nova clusters, potentially reducing overall processing time by handling multiple frames concurrently.

5. The functions used in the implementation

Grayscale Conversion

Purpose: The primary goal of converting color images to grayscale is to simplify the image data by reducing it from three color channels—Red, Green, and Blue—to a single intensity channel. This simplification significantly reduces the computational complexity required for subsequent processing steps, facilitating faster computations and reduced processing demands.

Mathematical Explanation: The formula used for grayscale conversion reflects the human eye's sensitivity to different colors, giving more weight to green, which is perceived more sensitively. The mathematical representation of this conversion is:

$$Y=0.299R+0.587G+0.114B \qquad \ldots(1)$$

Here, R,G,B denote the red, green, and blue components of a pixel, respectively, while Y represents the grayscale equivalent. This formula ensures that the transformation retains the most important visual information while reducing the data volume.

Gaussian Blur

Purpose: Gaussian blur is applied to smooth the image, an essential step in reducing noise and minor detail that could compromise the effectiveness of subsequent edge detection. By softening the image, Gaussian blur helps to mitigate the impact of insignificant variations like small imperfections or textures on the surface of the license plate.

Mathematical Explanation: The smoothing effect is achieved by convolving the image with a Gaussian kernel, a process mathematically defined by:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}} \qquad ...(2)$$

where x and y represent the distances from the kernel's center in horizontal and vertical directions, respectively, and σ is the standard deviation of the Gaussian distribution. This formula ensures that pixels closer to the kernel's center have more influence on the blurred output, resulting in a smooth, homogeneous area conducive to accurate edge detection.

Edge Detection (Canny Method)

Purpose: The Canny edge detection algorithm is utilized to identify significant changes in brightness across the image, which typically indicate edges—key markers for object boundaries such as those of license plates.

Mathematical Explanation: The Canny method uses the gradient of the image to detect edges. The gradient magnitude and direction are calculated as follows

$$Gradient\ Magnitude: G = \sqrt{G_x^2 + G_y^2} \qquad ...(3)$$

$$Gradient\ Direction, \Theta = tan-1(Gx/Gy) \qquad ...(4)$$

The algorithm then proceeds with non-maximum suppression to thin out the edges, followed by double thresholding and edge tracking to distinguish between true and false edges, ensuring that only the most relevant edges are retained for further analysis.

Morphological Operations (Dilation)

Purpose: Dilation is a morphological operation used to expand regions of foreground pixels in a binary image, enhancing features such as the edges detected in earlier stages. This operation is particularly useful for closing small holes and gaps within detected objects, which enhances the clarity and continuity of object boundaries.

Mathematical Explanation: Dilation involves applying a structuring element (kernel) to the image and expanding the shapes contained in the image based on this kernel. The operation is defined as:

$$(A \oplus B)(x,y) = \max_{(x',y') \in B} A(x-x', y-y') \qquad \ldots(5)$$

where A represents the binary image and B is the structuring element. The result is an image where small gaps and imperfections are filled, making subsequent analyses like character recognition more robust and reliable. Dilation not only enhances the visual characteristics of important features in an image but also plays a significant role in preparing the image for more complex analyses. It ensures that features such as edges and contours are pronounced and intact, which is essential for the accurate detection and recognition of objects within images.

Together, these image processing techniques form a comprehensive approach to license plate detection, each step building upon the last to ensure that the final image analysis is both efficient and accurate, suitable for real-world applications where speed and reliability are paramount.

Running Optical Character Recognition (OCR):

Purpose: OCR is a technology used to convert different types of documents, such as scanned paper documents or images captured by a digital camera, into editable and searchable data. For images, it involves detecting text regions and recognizing the characters within those regions.

6. Serial Implementation

This Python script processes a video to detect and recognize license plates using OpenCV for video handling and EasyOCR for text recognition, with GPU acceleration enabled for performance. It loads a video, iterates through each frame, and uses OCR to detect text within the frames. Detected text and their bounding boxes are drawn onto the frames, which are then saved to a designated directory. The script utilizes multi-threading to process multiple frames concurrently, enhancing efficiency. It tracks and outputs the total number of vehicles detected to a file and prints the count and processing status.

```
original_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
original_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
frame_rate = int(cap.get(cv2.CAP_PROP_FPS)) // 2  # Reduce the frame rate by half
```

```
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break

    # Resize the frame to 2x
    frame = cv2.resize(frame, dsize: (original_width * 2, original_height * 2))

    # Convert to grayscale
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Apply Gaussian blur
    blur = cv2.GaussianBlur(gray, ksize: (5, 5), sigmaX: 0)
```

```
for contour in contours:
    x, y, w, h = cv2.boundingRect(contour)
    aspect_ratio = w / float(h)
    if aspect_ratio > 2 and aspect_ratio < 5:  # Typical aspect ratio for license plates
        cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)  # Draw a green box
        plate_roi = frame[y:y+h, x:x+w]
        result = reader.readtext(plate_roi)
        if result:
            plate_text = result[0][1]
            print(f"Detected Number Plate: {plate_text}")
            cv2.putText(frame, plate_text, org: (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, fontScale: 0.8,
```

These are few code snippets from serial implementations showing the implementation of reducing the video speed by half and applying gaussian blur, morphological operations and edge detection as described in previous sections and finally this code draws green boarders across the number plate and print the detected texts to output.txt file.

7. Parallel Implementation using MPI (number of processors used 16)

Here are code snippets for MPI implementation :

#Initializing the environment

comm = MPI.COMM_WORLD

rank = comm.Get_rank() # identify each process

size = comm.Get_size() # total number of processes

#Division of tasks

frames_per_process = total_frames // size

*start_frame = rank * frames_per_process*

end_frame = start_frame + frames_per_process

#Synchronization

comm.Barrier() # making sure all the processes are in sync and to ensure processes have completed the tasks

#Inter process communication.

if rank == 0:

if frame_number != start_frame:

comm.send(frame, dest=(frame_number // frames_per_process), tag=77)

else:

if frame_number == start_frame:

frame = comm.recv(source=0, tag=77)

8. Outputs from serial and parallel implementation

Input video length	Serial implementation	Parallel Implementation
10.5 sec	602.4 sec	54.3 sec
1.5min	1002.34 sec	106.21 sec
2.45 min	2435.55 sec	132.4 sec

9. Issues faced

1. Input video perspective (angle)
2. Lighting conditions
3. Camera resolution

10. Conclusion

The project aimed at enhancing video processing for license plate detection through the implementation of parallel computing techniques using MPI on Nova clusters has demonstrated significant improvements in processing efficiency and scalability. By comparing serial and parallel implementations, it is evident that parallel processing drastically reduces the time required to analyze video data, making it feasible to handle larger video streams effectively.

The use of 16 processors within Nova clusters allowed for a substantial decrease in processing times, as shown by the performance metrics collected during the project. For instance, the processing time for a video segment of 2.45 minutes was reduced from 2435.55 seconds in a serial setup to just 132.4 seconds in a parallel setup. This represents a

dramatic improvement in speed, showcasing the power of distributing computational tasks across multiple processing units.

In conclusion, this project not only fulfilled its objective of improving the efficiency and scalability of video processing for license plate detection but also paved the way for future developments in real-time video analysis applications. The findings suggest that further exploration of parallel processing capabilities, especially in high-performance computing environments like Nova clusters, could lead to even greater advances in the field of computer vision and real-time video processing.

Github Link : https://github.com/sriharshamudumba/Real-Time-Video-Processing-and-Analysis-MPI-