# Project Proposal: License Plate Number Recognition

Junye Chen (junyec), Zhan Dong (zhand)

## 1 URL

https://patrickzhandong.github.io/ParallelPlateRecoginition/

## 2 Summary

For the final project, we want to write a parallel program for license plate number recognition, a very interesting topic in the field of computer vision. The process of plate number recognition involves various computer vision algorithms such as image segmentation, edge detection, etc. We will use CUDA (and maybe OpenMP), and try to have a good speedup against the benchmarks.

## 3 Background

License plate number recognition is a very popular topic in the field of computer vision. As more and more people in the world buy their own cars, the technology of identifying and recognizing numbers on the number-plates is also getting more and more handy. Plate number recognition can play a very important role in government or private security systems. Its also very interesting, in that the algorithm goes through some very classic image processing steps and ends up with character recognition, another classic problem in the field of computer vision. Thus, we believe that it is a very interesting project to parallelize.
In this project, we will form our plate number recognizer based on the method proposed in the paper by Mahesh Babu K and M V Raghunadh (http://ieeexplore.ieee.org/ielx7/7811913/7831584/07831610.pdf). The algorithm consists of the following steps:

1. Preprocessing the graph and Image enhancement.
2. Edge Detection that finds discontinuities in image and detects the number plate in the image
3. Image segmentation that partitions the number plates into bounding boxes that contain single character.
4. Character Recognition that matches the bounding boxes to the closest character or number.

From the above description, we observe that the algorithm requires a lot of computation, and therefore have the potential to be accelerated through parallelism. We believe that this algorithm has a large potential for different levels of parallelism. A very important reason for this is that a large proportion of the computation involved in this algorithm is both intensive and independent. For example, in the edge detection part, the calculation of the intensity gradients is independent from other intensity gradients, and therefore can be calculated in a parallel way. Therefore, we expect a good speedup through using the parallelization strategies we have learnt in this course. Besides, since the information we need to determine if a pixel belong to an edge normally comes from nearby pixels, if we can find a good way to assign work to threads, we expect a relatively low overall communication cost. It is also quite challenging since some of the steps such as character recognition requires looking at the entire bounding box, and find a similar pattern. This might break the locality.

# 4 Challenge

Some of the challenges we might face in this project are:

**1. Bad Locality and Cache Performance** In our computation, we will encounter memory access patterns that do not have good temporal localities. For example, if we store the image in row-major format, we might encounter severe cache misses during the calculation of some of the the numbers which require access to data in different columns. To alleviate these cache misses, we might come to a better way to store the data, or have some other approaches to improve cache hit rate.
Besides, the memory accesses of our algorithm can be very tricky. For example, in image segmentation, we are going to use k means clustering algorithm (more details below). The bounds of each cluster can be hard to determine. To look for optimal cluster centers, our memory accesses will be somehow random, so we cannot just look at nearby pixels. We need to look for a way to deal with this issue.

**2. Communication and Shared Space** Since we will be using CUDA for part of the parallelization, we also need to consider the communication. We observe from our previous assignment that a high communication cost will severely worsen the overall performance of the program. Therefore, we should come to good ways to do the communication, so that less elements need to be communicated. We should also think about what to be stored in the global space, and what to be stored in the shared space. A clever design of information saved in the shared space might also improve the performance.

**3. Synchronization** Another possible challenge we might face is synchronization. Since the algorithm we are using is a multi-stage algorithm, we need to make sure that the partial result after each step is complete and correct. There also might be some synchronization required in each step, and since frequent synchronization usually results in bad overall performance, we need to design the algorithm so that synchronizations are only done when necessary.

**4. Inherent Sequential Portion of the Algorithms** The algorithms we are using might not be perfectly parallelizable. For example, in image segmentation, k means clustering algorithm needs to iterate until it cannot find better centers for clustering. The iteration step is inherently sequential, and is impossible to speed up. Plus, inside each iteration, we need to find the optimal centers and let each point find its own cluster. Although the computation within each step is parallelizable, these two steps have to be done sequentially. In other words, all the points have to wait until the optimal centers are found before they look for their new clusters. We have to keep those in mind when we analyze the speedup and potential bottleneck in the algorithm.

# 5 Resource

We are going to follow the algorithm described in Babu and Raghumadh's paper. For each of the three main image processing step we are doing, we are going to look up algorithms to implement each of them. For example, for edge detection, we are going to follow Canny's algorithm described in this article (https://www.cse.unr.edu/ bebis/CS791E/Notes/EdgeDetection.pdf). For image segmentation, we are going to use the operations described in this article (http://www.sciencedirect.com/science/article/pii/S1877050915014143), which mainly uses k means

clustering algorithms to partition the image. In terms of parallelization techniques, we plan to use CUDA and possibly OpenMP to parallelize our solution, so we might use the latedays cluster.

# 6 Goals

## 6.1 PLAN TO ACHIEVE

Due to the large scalability of this project, we hope to implement the majority of the algorithm on our own and experiment various parallelization techniques on it. The algorithm is made up of three main parts, edge detection, image segmentation, and character recognition. We want to at least have a completely working and reasonably parallelized solution of edge detection and image segmentation at the end of the project.
For edge detection, we want to follow Canny's edge detection algorithm, which mostly uses intensity gradient in the image to detect edges. We are going to follow the algorithm detailed in this paper (https://www.cse.unr.edu/ bebis/CS791E/Notes/EdgeDetection.pdf), and we will look for any other tutorials for clarifications. We have looked for some existing libraries of edge detection, and we decide to use OpenCV's implementation of edge detection as a benchmark for both accuracy and efficiency.
For image segmentation stage, we are going to use k means clustering algorithm, which is not an easy algorithm to parallelize. It is hard to determine the bound for each cluster, so we will have to think carefully about the locality issue during parallelization.
We want to finish a mostly functional sequential version of edge detection and image segmentation by the checkpoint. Since the focus of this project is mainly on parallelization, we will see how the implementations of the above two algorithms turn out. Then we will determine if we have time to implement the character recognition step from scratch, because we will need a decent amount of time to come up with a good parallel solution for Canny's edge detection and image segmentation with k means clustering algorithm. We plan to experiment our parallel solution with various techniques, such as OpenMP, CUDA. We expect to achieve significant amount of speedup against our sequential implementation first, because we should observe an obvious speedup when we parallelize the algorithm on GPU. Then we are going to compare our implementation to some existing implementation in image processing libraries, such as OpenCV.

## 6.2 HOPE TO ACHIEVE

If we finish both algorithms, we will try implementing character recognition on our own and parallelizing it. If so, we theoretically have a working algorithm that can recognize license plate numbers! If the project doesn't work out too well, and the speedup is not very promising, we are probably going to spend more time improving the speedup of the two algorithms, so we will use some existing library for character recognition step. Therefore, we will still be able to demo our final product.

# 7 Demo

During poster session, we are going to demonstrate that our program can do a decent job in recognizing license plate numbers given some picture of a car, whether we have time to implement the character recognition step or not. Performance wise, we are going to display the timings for our sequential and parallel implementations, and explain the speedup (or bottleneck of the algorithm). Then we are going to show the timings for existing implementations (OpenCV, for example) of these algorithms, and compare them with ours.

## 8 Platform Choice

Since OpenCV has C++ implementations, we are going to use C++ for our project so that we minimize any lurking variables that affect speedup. We are also going to use CUDA and OpenMP for good parallelization. CUDA is a very common choice for achieving good speedup when manipulating images. In HW2, CUDA is able to speed up the parallel rendering very well. Although the memory copying in CUDA is very costly, if we can finish most of the computation inside CUDA kernel, we should take the most advantage of CUDA. CUDA also offers shared memory, which is definitely a good way to improve locality when computing in large arrays.

## 9 Schedule

**By November 8** Understand the algorithm of edge detection and image segmentation, start to write code for sequential implementation

**By November 20** Finish most of the sequential implementations for both algorithms, and sketch out the techniques to speed up the program

**By November 27** Finish the main skeleton of parallel implementation and should pass correctness checks. Compare the speedup against sequential benchmark and OpenCV benchmark (CPU and GPU). If the project goes well, start implementing a sequential version of character recognition.

**By December 5** Wrap up the implementations of edge detection and image segmentation. If the program achieves good speedup, explore the potential of parallelizing character recognition. If not, analyze any potential problems (bad locality, uneven distribution of work among threads, too much overhead of memory copying), try to improve, and look for some existing library of character recognition. Start the writeup

**By December 12** Make sure overall program works. Clean up code, and finish the writeup.

## 10 Reference

[1] https://en.wikipedia.org/wiki/Canny_edge_detector

[2] https://www.cse.unr.edu/ bebis/CS791E/Notes/EdgeDetection.pdf

[3] https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html

[4] http://ieeexplore.ieee.org/ielx7/7811913/7831584/07831610.pdf

[5] http://www.sciencedirect.com/science/article/pii/S1877050915014143