Presented to the College of Computer Studies

Computer Technology Department

De La Salle University - Manila

Term 3, A.Y. 2023-2024

In partial fulfillment of the course

MULTIPROCESSING AND PARALLEL COMPUTING

Milestone 3

Enhancing Traffic Sign Recognition Performance Using CUDA-Accelerated Hough Transform: A Comparative Study with C Implementation

Submitted by:

Group 4

CO, Sofia Bianca

DATARIO, Yasmin Audrey

EDRALIN, Philippe Nikos Keyan

PO, Aliexandra Heart

Submitted to:

UY, Roger Luis

July 30, 2024

I. ABSTRACT

This study enhances a Traffic Sign Recognition (TSR) application by applying the Hough Transform algorithm through CUDA parallel computing, with an emphasis on execution time comparisons against traditional C implementations. TSR is critical for the reliability of autonomous driving systems, necessitating precise and rapid detection of traffic signs. The problem addressed is the inefficiency of CPU-based TSR systems, which struggle with real-time processing due to limited parallelization capabilities. By leveraging CUDA's GPU-accelerated architecture, this project seeks to significantly reduce computation time by parallelizing the shape detection process. The methodology involves integrating CUDA with a MATLAB application that already performs HSV-based masking and extended border tracing, facilitating direct execution time comparisons between the CUDA-enhanced and C implementations for the Hough Transform Algorithm. This comparative analysis aims to demonstrate the potential of CUDA in accelerating TSR systems, potentially setting a new benchmark for future automated driving technologies.

II. INTRODUCTION

Traffic Sign Recognition (TSR) applications have become increasingly important in detecting and interpreting traffic signs. Traditional methods for TSR application predominantly utilize CPU-based implementations which, while effective, often need to catch up in real-time scenarios due to their limited parallel processing capabilities. To address these challenges, this project aims to enhance the performance of TSR software through the implementation of the Hough Transform using CUDA, with an emphasis on shape detection. The Hough Transform is a technique for detecting geometric shapes, such as circles and lines, which are seen in traffic signs. Through utilizing CUDA, the optimization achieved through this project is particularly valuable for automated driving systems. In such systems, the ability to quickly and reliably identify traffic signs is important to the overall performance and reliability of autonomous vehicles. Efficient

traffic sign recognition can also provide feedback to drivers regarding their blind spots, assuming they could not see the sign.
This project will involve the integration of CUDA with a MATLAB application that already performs HSV-based masking and edge tracing. This integration facilitates direct execution time comparisons between the CUDA-enhanced implementation, the C implementation, and the MATLAB Hough Transform function.

The primary objective of this project is to demonstrate the efficacy of CUDA in accelerating TSR systems. Through a comparative analysis, this project seeks to highlight the advantages of GPU acceleration in real-world traffic sign detection scenarios.

III. Comparison From Existing Implementations

Traffic Signs Detection (TSD) has seen various implementations over the years, each with different approaches, methodologies, and performance characteristics. Traditional methods typically employ edge detection algorithms like the Canny edge detector, followed by the Hough Transform, to identify geometric shapes such as circles and lines. While these methods are straightforward and effective under controlled conditions, they suffer from reduced accuracy in noise and varying lighting conditions (Shi & Lin, 2017). Traditional CPU-based methods process images sequentially, leading to slower performance and challenging real-time detection, especially with high-resolution images.

Several modifications have been made to improve the algorithm speed to the Hough Transform. For example, Yakimov and Fursov (2013) introduced a modified Generalized Hough Transform to handle noise better and efficiently detect traffic signs like triangles. These techniques improve upon traditional methods but still rely on CPU processing. Although they offer better noise handling and accuracy, the computational complexity remains high, limiting their real-time applicability (Yakimov & Fursov, 2013).

Initially, the group worked with a TSR system for a previous project using HSV filtering and Hough Transform implemented on MATLAB. This approach is similar to the traditional methods mentioned earlier. However, much like the other implementations with CPU processing, this method faced significant challenges in real-time processing due to the sequential nature of CPU operations.

Recognizing these limitations, we explored the possibility to enhance the TSR system's performance by leveraging GPUs' parallel processing capabilities through CUDA. The goal was to implement a CUDA-enhanced Hough Transform, allowing for the parallel processing of image segments, unlike its sequential counterpart. This approach takes advantage of the GPU's Single Instruction, Multiple Threads (SIMT) architecture, enabling simultaneous processing of multiple segments, which could possibly reduce computation time. To analyze the speedup and improvements, we planned a direct comparison between the traditional sequential C implementation, the CUDA-enhanced Hough Transform, and the built-in MATLAB implementation of Hough Transform.

IV. Salient Point Of The Project

Hough Transform is a technique to detect lines, circles and other shapes in an image. It maps points (peaks) from the image space to a parameter space where we find intersecting curves via the detection of collinear points.

We implemented Hough Transform in C and in CUDA, compared their time, then integrated it in matlab through MEX.

- **C IMPLEMENTATION**
  The mex function accepts a binary 2D array, width, and height. The binary 2D array is the edge-detected version of the image. It outputs an unsigned char accumulator array representing the Hough space, the max number of theta values, and the maximum rho value.

The accumulator is a 2D array where each cell represents a potential line in the image. The array dimensions are determined by the maximum rho value and the range of theta values (usually -90 <-> 89).

For each edge pixel (non-zero value) in the image, the algorithm iterates over possible theta values, calculates the corresponding rho value, and increments the accumulator cell that corresponds to these parameters.

The implementation is timed using 'QueryPerformanceCounter' to measure execution time over multiple runs (50 times).

- **CUDA IMPLEMENTATION**

  The mex function is more or less the same as the C implementation, but the difference is how we transform the image. The CUDA implementation parallelizes the Hough Transform using GPU threads. Each thread is responsible for processing a single pixel in the image.

  The kernel function calculates rho for each theta value for a given pixel and increments the corresponding accumulator cell using atomicAdd to prevent race conditions between threads.

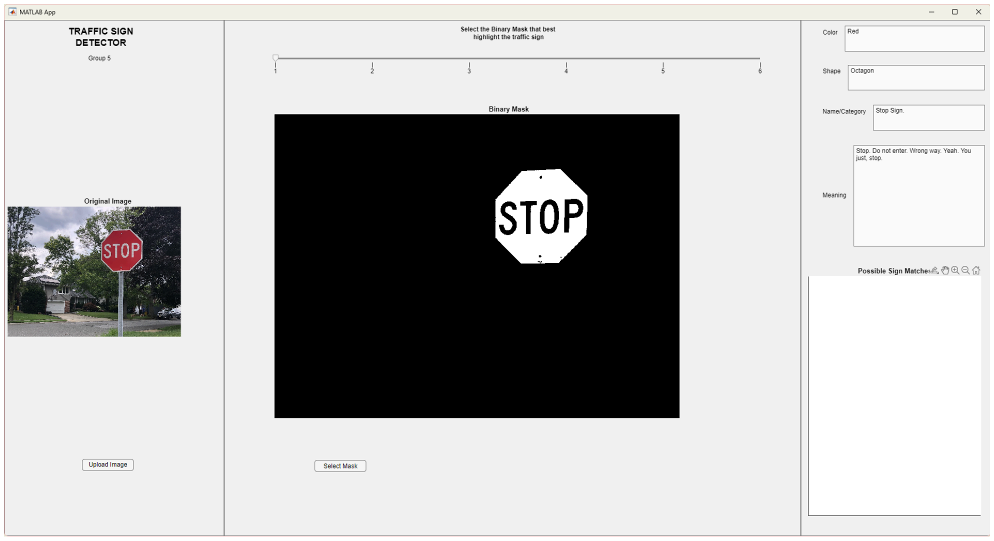
  The implementation is also timed using 'QueryPerformanceCounter' to measure execution time over multiple runs (50 times). This is compared with the C implementation to see if using CUDA really improves performance or not.

- **MATLAB INTEGRATION**

  Both implementations are integrated with MATLAB using MEX files, which lets us call the .c and .cu files in matlab as well as pass parameters and get outputs. The matlab code is also modified to fit our current implementation. Some changes include making the output of the C and CUDA implementations double-precision for a matlab function that required that specific data type since we used the output accumulator there, and we also transposed the array to fit some matlab function requirements.

Additionally, we included the built-in MATLAB Hough Transform function in our tests to compare with both the C and CUDA implementations. By comparing the three, we will be able to know if there is really a difference between the original implementation and our CUDA enhanced one.

V. Results and Conclusion



Color: Red

Shape: Octagon

Name/Category: Stop Sign.

Meaning: Stop. Do not enter. Wrong way. Yeah. You just, stop.

Figure 1. MATLAB Application of Traffic Sign Recognition with Image Input

For our testing, we tried different traffic signs with different shapes. Here, we used an image of a red stop sign. After masking it and getting the edges of the shape, we can now run our Hough Transforms.

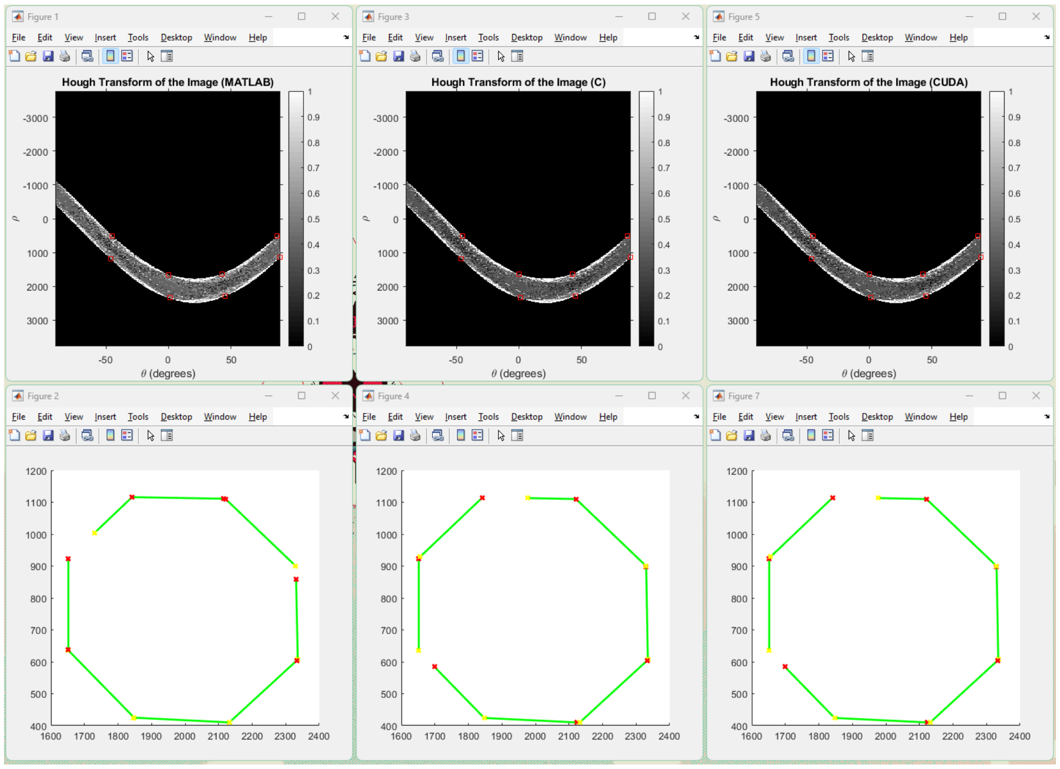
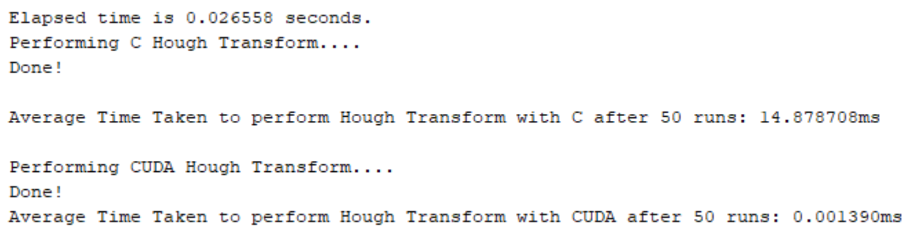


Figure 1
Hough Transform of the Image (MATLAB)
Figure 3
Hough Transform of the Image (C)
Figure 5
Hough Transform of the Image (CUDA)
θ (degrees)
ρ
Figure 2
Figure 4
Figure 7

Figure 2. MATLAB Hough Transform (leftmost); C Hough Transform (middle); CUDA Hough Transform (rightmost)

All three implementations produced the same Hough waveform, but the MATLAB Hough has a different line plot than the C and CUDA. While debugging, we found that the y coordinates for the houghpeaks of the C and CUDA implementations were offset by 1. This was enough to have a big difference in the way houghlines (a built-in matlab code) interpreted the houghpeaks. However, that does not mean that the C and CUDA implementations were wrong, it's just that the houghlines MATLAB function was built to be most compatible with the output matrix of the MATLAB Hough function. Unless we modify our code to output exactly the same as the built-in MATLAB Hough, this is the closest result we will have. On a positive note, our hough waveforms were identical.



```
Elapsed time is 0.026558 seconds.
Performing C Hough Transform....
Done!

Average Time Taken to perform Hough Transform with C after 50 runs: 14.878708ms

Performing CUDA Hough Transform....
Done!
Average Time Taken to perform Hough Transform with CUDA after 50 runs: 0.001390ms
```

Figure 3. Runtimes for all three functions

The CUDA implementation (0.00139ms) outperformed the C implementation (14.878708ms) timing-wise. While the C version processes each pixel sequentially, the CUDA version processes multiple pixels in parallel. This results in reduced computation time. Although the use of atomicAdd in CUDA is questionable, we found no alternative methods to achieve a similar result. The built-in MATLAB Hough Transform (26.558ms) ran the slowest out of the three, but was more accurate as established earlier.

VI. References

Ellahyani, A. (2016). Traffic sign detection using image processing. International Journal of Computer Applications, 975, 8887.

F. Shao, X. Wang, F. Meng, T. Rui, D. Wang, and J. Tang, "Real-Time Traffic Sign Detection and Recognition Method Based on Simplified Gabor Wavelets and CNNs," Sensors, vol. 18, no. 10, p. 3192, Sep. 2018, doi: https://doi.org/10.3390/s18103192.

In: Blanc-Talon, J., Kleihorst, R., Philips, W., Popescu, D., Scheunders, P. (eds) Advanced Concepts for Intelligent Vision Systems. ACIVS 2011. Lecture Notes in Computer Science, vol 6915. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-23687-7_55

P. Yakimov and V. Fursov, "Traffic Signs Detection and tracking using modified Hough transform," 2015 12th International Joint Conference on e-Business and Telecommunications (ICETE), Colmar, France, 2015, pp. 22-28.

Shi, J.-H., & Lin, H.-Y. (2017). A vision system for traffic sign detection and recognition. IEEE Transactions on Intelligent Transportation Systems, 17(6), 1623-1637.

Van Den Braak, GJ., Nugteren, C., Mesman, B., Corporaal, H. (2011). Fast Hough Transform on GPUs: Exploration of Algorithm Trade-Offs.

Y. Chen, Y. Xie, and Y. Wang, “Detection and Recognition of Traffic Signs Based on HSV Vision Model and Shape features,” Journal of Computers, vol. 8, no. 5, May 2013, doi: https://doi.org/10.4304/jcp.8.5.1366-1370.

Yam-Uicab, R., Lopez-Martinez, J.L., Trejo-Sanchez, J.A. et al. A fast Hough Transform algorithm for straight lines detection in an image using GPU parallel computing with CUDA-C. J Supercomput 73, 4823–4842 (2017). https://doi.org/10.1007/s11227-017-2051-5