Implementing and Parallelizing Real-time Lane Detection on Heterogeneous Platforms

Xiebing Wang
Department of Informatics
Technical University of Munich
Munich, Germany
wangxie@in.tum.de

Christopher Kiwus
Department of Informatics
Technical University of Munich
Munich, Germany
kiwusc@in.tum.de

Canhao Wu
Department of Computer Science
Sun Yat-Sen University
Guangzhou, China
wuch28@mail2.sysu.edu.cn

Biao Hu
College of Information Science & Technology
Beijing University of Chemical Technology
Beijing, China
hubiao@mail.buct.edu.cn

Kai Huang
Department of Computer Science
Sun Yat-Sen University
Guangzhou, China
huangk36@mail.sysu.edu.cn

Alois Knoll
Department of Informatics
Technical University of Munich
Munich, Germany
knoll@in.tum.de

***Abstract*—Lane detection is a cardinal functionality in state-of-the-art Advanced Driver Assistant Systems (ADAS). However, it is still not straightforward to fulfill the real-time performance demand of processing High Definition (HD) images with high robustness and scalability. To address this problem, we propose an improved lane detection algorithm based on top-view image transformation and two-stage RANdom SAmple Consensus (RANSAC) model fitting. By virtue of off-line affine homography matrix adaption to bound an adaptive Region Of Interest (ROI) for subsequent on-line Warp Perspective Mapping (WPM) transformation, the algorithm can analyze arbitrary on-road videos and generate adaptive ROI without priori knowledge about camera parameter. To ensure the scalability, we present a comprehensive parallel design of the application in a heterogeneous system consisting of multi-core CPU, GPU and FPGA. We show in detail how the potentially parallel task loads are implemented and optimized so that they can be mapped to the most suitable processor so as to achieve optimal performance. Experimental results reveal that our improved algorithm can robustly process the video streams with a higher accuracy. Moreover, the heterogeneous executions are capable of processing HD 1920×1080 images with runtime performance of 81.6 fps and 47.9 fps, respectively, on an AMD FirePro W7100 GPU and a Terasic Arria 10 FPGA.**

***Index Terms*—ADAS, Lane detection, multi-core CPU, GPU, FPGA, OpenCL**

I. Introduction

As a fundamental functionality in ADAS, lane detection is a well-studied algorithm which has attracted much research attention since mid-1980's. Typically for lane detection, camera is the most frequently used sensor type not merely because of its fairly low cost, but also taking into account that roads and lanes are designed to be perceived by human drivers and the perception of visual cues is essentially the same as for human eyes. However, using camera as source of information requires large computational power to handle the amount of data, particularly to meet the state-of-the-art requirement of real-time High Definition (HD) image processing.

Apart from the performance requirement, how to ensure the robust utilization of lane detection application across various on-road scenarios is still nontrivial. For instance, to reduce noise influence and computational complexity, it is typically a pre-step to extract a Region Of Interest (ROI) within the whole image before it is actually analyzed. However, how to obtain an efficient, stable and reliable ROI is often scenario-specific and this size needs to be self-tuned if external on-road condition changes drastically. What's more, given a specific lane detection algorithm, it is often not easy-to-evaluate when taking an arbitrary road video stream as input, since camera intrinsic and extrinsic parameters are often unknown and need to be calibrated. On the one hand, this hinders the promotion of the algorithm since the third-party users or other researchers do not always assume the highly practicability of the sample videos provided by the developer. On the other hand, this prolongs the development period since it is time-consuming to construct a technically sound benchmark.

The last important issue lies in the scalability of the application, i.e., how convenient for the end-users to consider the tradeoff between the algorithmic accuracy and execution speed, and how expensive for the developers to transplant the application back and forth, for instance, from simulation environment to real world, and from one platform to another. The bottleneck here is not merely hardware configuration and code modification, but also the burdensome performance optimization across different architectures.

To address the aforementioned problems, this paper proposes a comprehensive design of an improved lane detection algorithm across heterogeneous parallel platforms including Commercial Off-The-Shelf (COTS) multi-core CPU, GPU and FPGA. During the implementation, we present in detail how the potentially parallel workloads are elaborately implemented and optimized so that tasks are mapped to the most suitable processor. We choose Open Computing Language (OpenCL) [1] as the programming framework to ease the cross-platform deployment of the application. The original lane detection algorithm is proposed in [2]. However, the work in [2] assumed a pre-defined fixed ROI and guessed camera parameters to evaluate the vanishing point for subsequent Inverse Perspective Mapping (IPM) transformation of the ROI into a Bird-Eye View (BEV) image. We overcome these drawbacks by virtue of orientation map voted vanishing point estimation and Warp Perspective Mapping (WPM) using off-line calculated and on-line regulated homography matrix. In this way, the algorithm

978-1-5386-7479-6/18/$31.00 ©2018 IEEE

can analyze arbitrary on-road videos and generate adaptive ROI without priori knowledge about the camera parameters, thus facilitating its versatility. Experimental results demonstrate that our improved algorithm can achieve a higher accuracy with better robustness. The parallel implementations are capable of processing HD 1920×1080 images with a faster-than-real-time performance and accelerate the baseline single-core CPU execution with speedups of 8.87× and 5.21× when using GPU and FPGA, respectively. The main contributions of this paper can be summarized as follows:

- We present an improved lane detection algorithm which can efficiently analyze arbitrary on-road videos, by virtue of adaptive ROI extraction and camera-independent homography matrix estimation, and afterwards detect multiple curve lanes via two-stage RANdom SAmple Consensus (RANSAC) model fitting.
- We propose a comprehensive design of the lane detection algorithm across heterogeneous parallel platforms consisting of multi-core CPU, GPU and FPGA. We show how the parallel task loads are implemented and optimized so that they are mapped to the most suitable processor so as to achieve optimal performance.
- We conduct a detailed comparative study of using homogeneous and heterogeneous configurations to accelerate the application across different platforms. The experimental results throw light upon future deployment of ADAS applications with heterogeneous accelerators.

The remainder of this paper is organized as follows: Section II reviews state-of-the-art research on lane detection and parallel acceleration of ADAS applications. Section III illustrates the drawbacks and our improved design of the previous work. Section IV discusses the parallel implementation and optimization. Section V presents experimental results and discussion. Section VI concludes the paper.

II. Related work

A. Lane detection techniques

The algorithm in this paper pertains to the camera-based methods and therefore in the following we mainly review the approaches which only use camera as source of information. Basically camera-based approaches are characterized by similar execution procedures as follows. First the ROI within the camera-captured image is defined for further processing. This definition of the ROI size is often algorithm-dependent. Some approaches attempt to distinguish and extract the lanes from the whole image [3] [4] [5], while other researchers generate the ROI via either manually setting the ROI boundary [6] [7] [8] or dynamically calculating the lane area with inherent road properties and camera parameters [9] [10] [11].

After the ROI is generated, typical image processing methods are adopted to extract the features of the lane boundaries, such as color, gradients and edges, to distinguish between the markings and non-lane areas inside the image. In this step, the image on which the algorithm is operating can be raw or BEV images. By using raw image, the lane features can be directly extracted and mapped back to the image so as to highlight the candidate lanes. However, the perspective effect, i.e., the angle of view under which a scene is acquired and the distance of the objects from the camera, in fact must be taken into account in order to weigh each pixel according to its information content. Current researches often use two different approaches, namely IPM [3] [8] [9] [12] and WPM [13] [14], to compute a BEV image from the input image.

The models used to fit the candidate lanes are miscellaneous. The studies in [5] [6] [8] [11] used RANSAC model to perform straight line and spline fitting to refine the detect lanes. In [7] and [15], the authors proposed to use particle filter to predict lane markings after the previous lanes are detected, while the work in [9] adopted similar Kalman tracking.

B. Parallel acceleration of ADAS applications

State-of-the-art research does not witness too much on the acceleration of lane detection using parallel platforms. The authors in [5] used CUDA and TBB to optimize the processing pipeline of a RANSAC-based road lane detection application. They reported a processing speed of 34.8 ms per frame for 640×480 images, with a speedup of around 3.55×. The work in [16] proposed a lane departure warning system implemented on an Xilinx FPGA. They used SIMD structure to implement vanishing point-based parallel Hough transform and reported a real-time performance of 40 Hz. Another similar study is presented in [17], which gives performance result of 30 frames per second. The most related work with this paper is [7], in which the authors accelerated lane detection across a rich set of parallel platforms.

Meanwhile, there exist plenty of researches that propose to utilize COTS components to accelerate other ADAS algorithms, such as pedestrian detection [18], audio sensing [19], traffic sign detection [20], etc. While it is a potential trend to speedup ADAS applications with parallel processors, in this paper we present a comprehensive study of accelerating lane detection with OpenCL. To take full advantage of different parallel platforms, we propose in detail how the work loads can be optimized to leverage the optimal execution.

III. Algorithm design

The algorithm is based on the previous work in [2], which generates an IPM-based BEV image of a pre-defined, fixed ROI inside the input frame. To extract lanes in the IPM transformed image, this approach filters the image vertically by a smoothing Gaussian filter and horizontally by a second derivative Gaussian filter. Based on the filtered IPM values, a simplified Hough transformation is applied, which gives an initial guess about the position of the lane markings. Two RANSAC iterations are afterwards performed to refine the lane detection. The first step matches linear lines using previous Hough transformation data. After applying geometric checks, the second RANSAC iteration fits the lane to a third degree Bezier spline. The subsequent post-processing step tries to re-localize the matched splines in the original input image and to extend the relocated results.

Although this approach is illumination invariant and provides the capability to detect straight as well as curved lane markings, it still suffers from several crucial drawbacks: ① The pre-defined ROI only provides a limited section of the available lane marking information inside each image frame and neglects other relevant information; ② The proposed IPM resolution of the ROI is downscaled to 160×120 pixels, which further decreases the lane information; ③ As the top-view IPM transformation is based on unknown camera parameters, these values have to be guessed, resulting in an inaccurate vanishing point estimation; ④ The vanishing point does not take road changes into account by assuming constant street gradients.

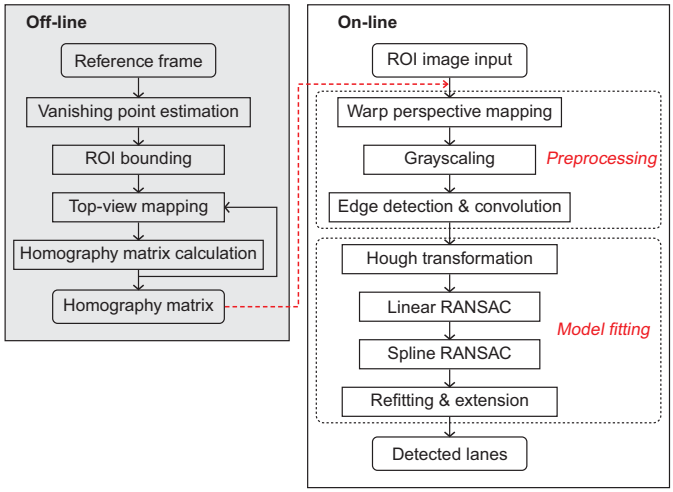
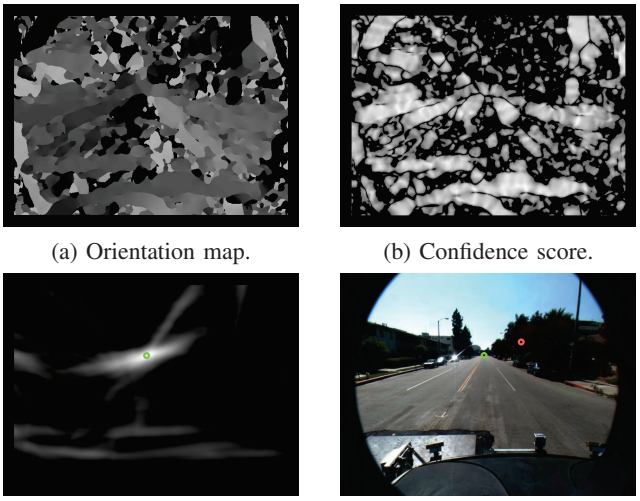


Fig. 1: Processing flow of the improved algorithm.



(a) Orientation map. (b) Confidence score.

(c) Vanishing point voting. (d) Estimated vanishing point.

Fig. 2: Vanishing point estimation.

The improved design tries to overcome the limitations mentioned above. As the camera parameters of the lane detection data set are unknown and have to be guessed, which results in an inaccurate transformation matrix, in order to increase the accuracy of this matrix, in this paper the top-view image is processed by a WPM instead, which is based on the reference frame of the video stream. The overall design of the improved algorithm is shown in Figure 1. The homography matrix used in WPM is first generated off-line by virtue of the vanishing point estimation and top-view mapping of the reference frame. Subsequently, each image is processed on-line via WPM to generate the BEV image. Then the top-view ROI is grayscaled and convoluted to extract the vertical as well as quasi-vertical lane markings. Afterwards, the lane marking is refined with a simplified Hough transformation and two RANSAC fitting, which is similar to the work in [2].

A. Vanishing point estimation

In contrast to IPM which uses intrinsic (focal length, optical center) as well as the extrinsic camera parameters (pitch angle, yaw angle, roll angle, height above the ground) to calculated the required transformation matrix, the WPM method is independent from the camera parameters. However, a minimum of four corresponding point pairs in the original image as well as the transformed image are required to compute an affine matrix mapping. As the perspective transformation matrix is unknown, these correspondences cannot be calculated directly. To get the corresponding points in the raw image, we use a vanishing point based mapping from an arbitrary image to its corresponding top-view on the reference frame. To estimate the vanishing point the approach in [21] is implemented. The texture orientations are estimated by convoluting the reference image with multiple Garbor filters over 36 orientations from $[0, \pi)$ quantized with a step size of 5°. After a confidence score is calculated for the 36 values obtained per pixel coordinate for every orientation, a voting map is obtained based on the confidence scores inside a lower semicircle around each coordinate centered at the currently processed pixel. Then the final vanishing point is proposed to correspond to the largest value in the voting map. Figure 2 shows the result of our detected vanishing point in the reference frame. As can be seen, the position of our estimated vanishing point (green circle in Figure 2d) is far more accurate than the one calculated by [2] (red circle in Figure 2d).

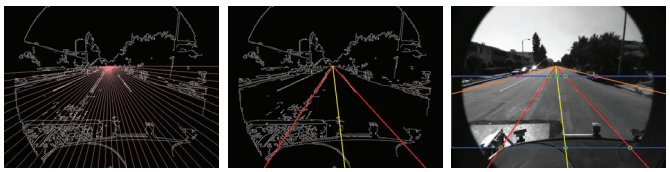
B. ROI bounding

We assume some reasonable environment-related properties in the reference frame:

1) A minimum of two lanes are included in the reference frame, one to the left side and one to the right side of the vanishing point.
2) The lane boundaries in the reference frame are parallel.
3) The intensity values of the lane markings are higher than the ones of the surrounding road surface.
4) The vanishing point lies within the reference frame.
5) The roll angle of the camera is close to zero and can be neglected.

Figure 3 gives the overall procedure of the ROI initialization. Based on Assumption (5), the vanishing line can be approximated as a line parallel to the x axis that intersects the vanishing point. With Assumption (1) and (3), Canny edge detector is applied to extract lane boundary points. To eliminate outliers from the binarized edge data, the image below the vanishing line is sampled with straight lines centered at the vanishing point within an angle of $[0, \pi)$ between the x axis and the sampling line with an empirically defined step size of 4° (shown in Figure 3a). As a result, lines that overlap with a lane boundary edge score best. To compensate small vanishing point errors, this procedure iterates over an offset of three pixels to the left and to the right side of the vanishing point. The final scoring for each line is defined as the maximum response over all offsets. Based on Assumption (1), the set of sampling lines is separated into two equally sized subsets, ranging from $[0, \pi/2)$ and $[\pi/2, \pi)$, respectively, to detect one lane boundary in the left and the right part of the image according to the x coordinate of the vanishing point. Each lane marking is defined as the highest score in the corresponding subset. Additionally, an upper and lower y axis boundary (blue lines shown in Figure 3c) is further imposed to neglect image parts without any road information as well as unreliable areas with perspective effects.

As observed in Figure 3c, the ROI is bounded by the upper and lower y axis limits (blue lines) and the image border



(a) Linear sampling. (b) Detected borders. (c) Initialized ROI.

Fig. 3: ROI extraction from the reference frame.

crossing lines (orange lines). The y-coordinates of the upper and lower boundary are calculated by

$$\begin{aligned} U_y &= V_y + \alpha \times (H - V_y) \\ L_y &= V_y + \beta \times (H - V_y) \end{aligned} \tag{1}$$

where V_y is the y-coordinate of the vanishing point, H is the image height and α, β are user-defined scale factors. The image border meeting points are defined by their equal y-coordinates, which is obtained from

$$B_y = U_y + \gamma \times (L_y - U_y) \tag{2}$$

where U_y and L_y are the previously defined upper and lower y axis boundaries and γ is a parameter ranging from $[0, 1]$ to adjust the upper left and right triangles in the original image, which are neglected in the BEV as they are supposed to contain irrelevant information about the environment.

With Equation (1) and Equation (2), the y-coordinate of the image border meeting points is calculated as

$$B_y = V_y + [\alpha + \gamma \times (\beta - \alpha)](H - V_y) \tag{3}$$

That is to say, given image height H, vanishing point y-coordinate V_y and empirically determined coefficients α, β and γ, the ROI is adaptive to the video streams.

C. Top-view mapping & Homography matrix adaption

The top-view mapping is separated into two steps. Initially, feature point coordinates are obtained from the reference frame to estimate corresponding points in the BEV image. Then the relations between the matching points are constrained by the underlying assumptions to compute the required homography mapping. First the bottom corner distance in the BEV image is assumed to be a fixed value v and thereupon the homography matrix is computed. As the distance between the top-view transformed lane boundary feature points and the medium x axis (x_m) in the BEV has to be equal for the upper and lower image border according to Assumption (2), the bottom corner distance correction scaling s_c is calculated as

$$s_c = \frac{1}{n} \sum_{i=0}^{n} \frac{d_u({}^tP_i, x_m)}{d_l({}^tP_i, x_m)} \tag{4}$$

where n is the number of feature points extracted from the input image and d_u, d_l are the upper and lower distances from the current feature point in BEV coordinates ${}^tP_i$ to x_m. The final bottom border distance d_f is computed as

$$d_f = 2 \times v \times s_c. \tag{5}$$

The corresponding points in the reference frame and the BEV image are used to compute the corrected homography matrix. To compensate vanishing point fluctuations due to changing road gradients, in each frame we adapt the homography matrix to the current street conditions. If the variance of the lane-to-vanishing point distance exceeds a certain threshold, then Equation (4) and Equation (5) are reused to adjust the homography matrix and restore parallel lane conditions.

D. On-line iterative processing

The input of the iterative detection is the cropped ROI based on the boundaries inferred in Section III-B. Given homography matrix generated off-line, the main steps performed on-line are WPM, image grayscaling (GS), edge detection (ED), image convolution (CONV), Hough transformation (HT), linear and spline RANSAC (LR, SR).

- **WPM** is implemented by mapping the pixel value at point P_c in camera coordinate system to its corresponding point P_g in BEV ground plane, with projection relationship as

$$P_g = \mathbf{H_a} P_c \tag{6}$$

  where $\mathbf{H_a}$ is the affine homography matrix.
- **Image grayscaling (GS)** in our design only takes into account one color channel of the input image, for the sake of efficiency, when calculating the mean image value.
- **Edge detection (ED)** of the grayscaled image is implemented by subtracting the intensity of each BEV coordinate by the mean image value to reduce the influence of non-lane marking points in further processing steps, whereby negative values are set as zero.
- **Convolution (CONV)** of an original image $\mathbf{I}$ into the convoluted image $\mathbf{J}$ with a kernel matrix $\mathbf{K}$ can be represented as

$$\mathbf{J}(x, y) = \mathbf{K} * \mathbf{I} = \sum_{i=-k_x}^{k_x} \sum_{j=-k_y}^{k_y} \mathbf{K}(i, j)\mathbf{I}(x-i, y-j) \tag{7}$$

  where k_x, k_y are half of the kernel size in x- and y-direction and x, y are the coordinates of the current processed pixel. In our design, the image is convoluted with a second order Gaussian derivative filter.
- **Hough transformation (HT)** is implemented by adding up the pixel values in each column of the convoluted image, assuming that the lane boundaries highlighted by the previous steps achieve the highest values.
- **RANSAC model fitting (LR, SR)** can be divided as two stages. First a random subset of the data points are used to compute the model parameters. Then the quality of the postulated model are tested and candidate data points that fit the model within a defined threshold according to a specific loss function are added into the consensus set. The iteration terminates if the cardinality of the consensus set exceeds a second threshold.

IV. Parallel implementation and optimization

A. Parallel implementation

The main steps illustrated in Section III-D can be divided into two categories, namely pixel-wise dependent ($task_{p\_d}$) and independent tasks ($task_{p\_ind}$). The pixel-wise independent tasks can be highly parallelized since data manipulations over each image pixel do not interfere with each other and the correctness of the final result can be guaranteed. However, the pixel-wise dependent tasks typically perform atomic data aggregation and therefore might block the parallel processing

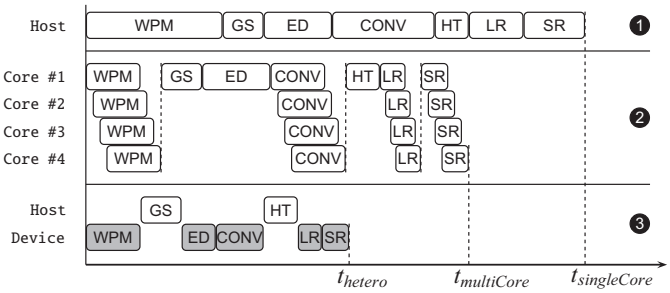


Fig. 4: Overview of the designs on different platforms.

pipeline, which poses a potential trade-off. The WPM, edge detection, image convolution and RANSAC model fitting steps belong to the pixel-wise independent tasks, while the rest two are pixel-wise dependent.

Figure 4 gives an overview of native and parallel implementations of the lane detection algorithm. For the native CPU design, we provide a single-core implementation as the baseline and propose a multi-core design to showcase the speedup as well. The *single-core implementation* (Case ❶) handles the aforementioned steps sequentially, while the *multi-core implementation* (Case ❷) processes WPM and image convolution with multiple threads and executes all RANSAC iterations for one vertical line in a single thread. To ensure the scalability of the parallel implementation, we use OpenCL to program on GPU and FPGA. In the *OpenCL-based implementation* (Case ❸), we use 4 kernels to consume WPM, edge detection, image convolution and two RANSAC model fitting workloads. Edge detection and image convolution are combined into one kernel so that the number of writing operations is halved as both steps are performed simultaneously without caching the mean corrected top-view image. Moreover, each of the two RANSAC kernels processes a single iteration of one vertical line per invocation. Each pixel operation on the BEV coordinate is mapped to one work item and all BEV pixel coordinates of the ROI are divided evenly among the work groups. With regards to the pixel-wise dependent tasks, we provide different implementations which either includes or excludes the atomic manipulations in order to test the tradeoff of on-device versus out-of-device data aggregation.

During the experiments, we found the computation of the RANSAC model fitting were fast enough when processed on CPU side, therefore we use different OpenCL implementations to further evaluate the RANSAC runtime performance. One approach computes the complete RANSAC algorithm on the parallel device, while the other processes the RANSAC model on CPU side and calculates the quality of the model on the device exclusively.

B. Optimization

The optimization of both GPU and FPGA kernels is two-fold. First pre-computed values are used in image convolution kernel to replace memory reading operations by a multiplication with a fixed constant, to avoid expensive global memory access. Secondly, each loop inside the kernels is manually unrolled to ensure the benefit of coalesced memory access.

The main optimization effort lies in the kernel configuration on the FPGA platform. As OpenCL kernel is hardware synthesized and then implemented as dedicated circuit blocks on FPGA, the board resource limits the actual performance and the resource utilization of each kernel needs to be coordinated. In our design the four OpenCL kernels are included into a single file before they are compiled as the executable binary file, taking into account the fact that kernel context switching overhead is significantly expensive and cannot be compensated by the full optimization of each kernel instead, as each frame is processed continuously in a loop manner.

In Intel FPGA SDK for OpenCL [22], several attributes can be used to guide the optimization of the kernel. To increase the data processing efficiency, `num_simd_work_items` can be used to specify the number of work-items within a work-group so that the kernel is executed in a Single Instruction Multiple Data (SIMD) manner. `num_compute_units` is another attribute that can instruct the off-line OpenCL compiler to generate multiple kernel compute units capable of executing multiple work groups simultaneously. In addition, `max_unroll_loops` can be indicated to decrease the number of executed iterations at the expense of increased hardware resource consumption. In our implementation, both `num_simd_work_items` and `max_unroll_loops` are set as 1 since each kernel is thread-ID dependent and each loop is manually unrolled.

The optimization of the aforementioned OpenCL kernels on FPGA is as follows: first we assume the most simplified configuration of each kernel, i.e., with `num_compute_units` set as 1, to guarantee that our design meets the board resource limitation. Afterwards, we step by step optimize each kernel by replicating more compute units to the most time-consuming task load. More details are referred to Section V-C.

V. Experiments and discussion

A. Experimental setup

We use a heterogeneous system as the test environment and the detail information about the hardware specification is shown in Table I. To evaluate the performance of the implemented algorithm, the runtime as well as detection quality are measured for the Caltech data set [23]. This data set consists of four clips on various urban street scenarios including straight and curve lane markings, shadows, other vehicles, reflections and street conditions to reflect real-world conditions.

B. Accuracy evaluation

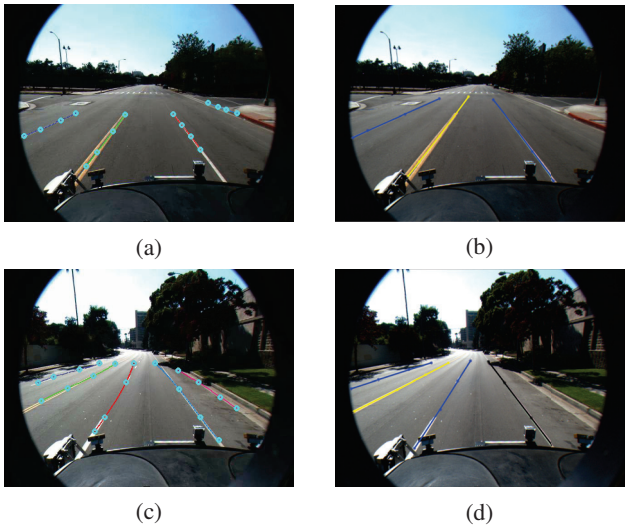
Table II gives the compared accuracy results of our implementation and the work in [2]. From the table we can see that

TABLE I: Detailed specification of the hardware platforms.

Platform	**Information**	
Host CPU	Intel Xeon E31225 @ 3.10GHz	
Thermal Design Power	95W	
PCIe generation	2.0	
Device	FPGA	GPU
Model	Terasic Arria 10	AMD W7100
Architecture	Arria 10 AX	FirePro
OpenCL SDK version	Intel FPGA SDK 16.0	AMD APP SDK 3.0
Peak GFLOPS	1366	3379.2
Peak board power (W)	95	150

TABLE II: Accuracy comparison of our design and [2].

Clip	**Total**	**Detected**	**Correct rate**		**False positive rate**	
			our design	[2]	our design	[2]
1	919	853	92.81%	91.62%	19.15%	5.66%
2	1048	932	88.93%	85.50%	41.60%	40.64%
3	1274	1230	96.55%	92.78%	26.22%	13.11%
4	931	886	95.17%	93.66%	31.58%	8.59%
Total	4172	3901	**93.50%**	90.89%	28.89%	17.38%



(a) (b) (c) (d)

Fig. 5: Comparison of the detected results and ground truth. Figure (a): Correct detection resulting in false positives. Figure (b): Wrongly labelled data excluding some lane markings. Figure (c): Correct detected rightmost lane marking. Figure (d): Wrongly excluded rightmost lane boundary.

the improved design outperforms the work in [2] in terms of the correct rate, which describes the number of lane markings found divided by the number of available ones. For all the clips our implementation increases the correct rate and the average accuracy is 93.50%. However, our method suffers from a nearly two times increased false positive rate compared to the work in [2]. The reason for this is as follows: ① The original data set is not labeled correctly and consistently. Several of the supposed false positive results are actually true positive, but not labeled in the original data due to unknown reasons (shown in Figure 5a and 5b). ② Because of the increased ROI of our approach, lane markings that are interrupted by crossroads are detected several frames earlier causing false positive results even if the lane markings are detected correctly. ③ The left and right road boundary, which should be considered as lane marking as well, is detected for most of the frames processed by the advanced implementation, but is excluded from the labeled ground truth over all video files (shown in Figure 5c and 5d). ④ The remaining actual false positives are mainly caused by sidewalks and noisy road surfaces.

C. Runtime evaluation

The whole algorithm is partitioned into four parts and each part is recorded with time stamps to obtain the final execution time. Table III details the information of each partition. **Partition I** includes the WPM transformation, image grayscaling and the computation of the image mean gray value. **Partition II** performs the edge detection, image convolution and the simplified Hough transformation. Subsequently, **Partition III and IV** perform the linear and spline RANSAC model fitting, respectively. The reason for this partitioning is straightforward, since each partition natively corresponds to one OpenCL kernel in the later design. For the runtime evaluation, each implementation is run multiple times and the final results are averaged. The comparison tests are performed on 640×480 videos, while the optimal designs are further evaluated on HD 1920×1080 images.

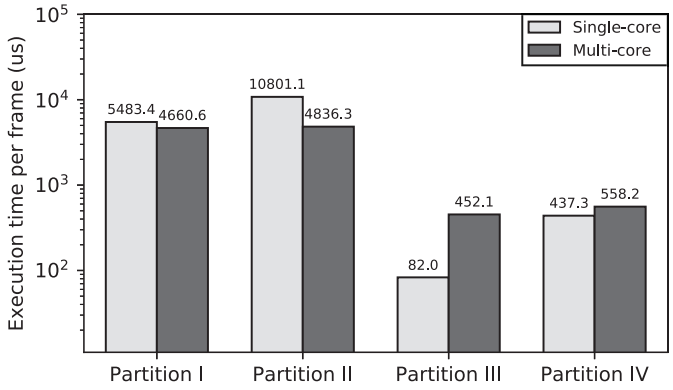


Fig. 6: Execution time of the homogeneous execution on CPU.

1) Homogeneous execution: We first evaluate the performance of the single- and multi-core executions on CPU. Figure 6 exhibits the execution time of each partition with single or multiple threads. Note the logarithmic scale of the y-axis. Here the multi-core implementation is tested with 8 threads. From the figure we can notice that the time costs of Partition III and IV are far less than that of Partition I and II. This can be explained by the reason that the computation load of the linear and spline RANSAC model fitting is rather small, as in practice the number of detected potential lane markings is always below 10. In this case, the benefit of concurrently performing RANSAC model fitting cannot compensate for the overhead of initializing new threads and additional data synchronization, resulting in the longer execution time of the multi-core implementation. However, the preprocessing of the ROI image is computation-intensive and therefore can be accelerated by the parallel implementation. In our design, the multi-core execution outperforms the single-core version and decreases the execution time of Partition I and II by 15.01% and 55.22%, respectively.

2) Heterogeneous execution: The OpenCL-based implementation follows the top-down design principle. First of all, the naive version wraps each of the aforementioned partition into one kernel. Afterwards, by gradually pruning lightweight and data aggregation task loads, the runtime performance of each kernel is re-evaluated to demonstrate the tradeoff of the parallelization of these lightweight workloads and atomic operations. The step-by-step optimization of both the CPU-GPU and CPU-FPGA heterogeneous executions is shown in Table IV. The steps which are marked with ticks are executed on the GPU (or FPGA), while the rest are run on the host CPU. First all the steps in Table III are assumed to be parallelized (*heteroGPU*$_1$, *heteroFPGA*$_1$), then the model computing in the RANSAC step is excluded to exhibit the pro and con of parallelizing this lightweight task load (*heteroGPU*$_2$, *heteroFPGA*$_2$). Subsequently, atomic data aggregation tasks are excluded to weigh the benefit of implementing them with

TABLE III: Partitions of the algorithm for runtime evaluation.

Module	Step	Partition	Category
Preprocessing	WPM	I	$task_{p\_ind}$
	Image grayscaling	I	$task_{p\_d}$
	Edge detection	II	$task_{p\_ind}$
	Image convolution	II	$task_{p\_ind}$
Model fitting	Hough transformation	II	$task_{p\_d}$
	Linear RANSAC	III	$task_{p\_ind}$
	Spline RANSAC	IV	$task_{p\_ind}$

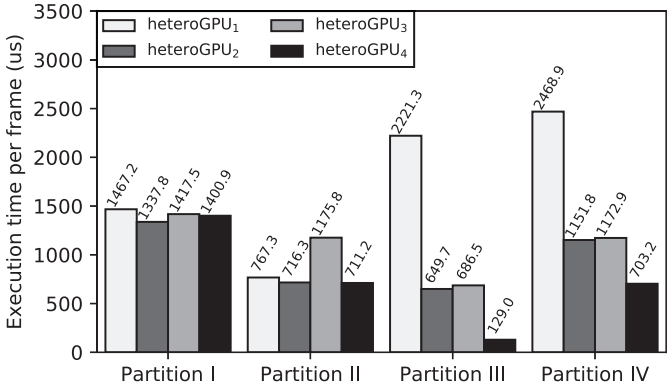


Fig. 7: Time cost of CPU-GPU heterogeneous execution.

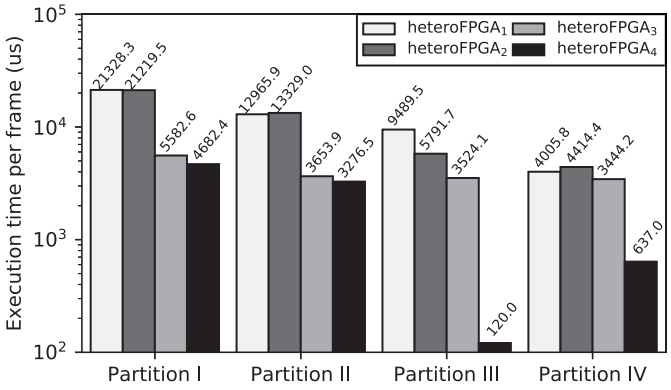


Fig. 8: Time cost of CPU-FPGA heterogeneous execution.

OpenCL (*heteroGPU*$_3$, *heteroFPGA*$_3$). At last, the optimal execution (*heteroGPU*$_4$, *heteroFPGA*$_4$) is inferred from the previous contrast tests.

Figure 7 gives the runtime results of the CPU-GPU heterogeneous implementations. It is clearly seen that in *heteroGPU*$_1$ the time cost of RANSAC model fitting (Partition III and IV) is nearly two times large as the image preprocessing (Partition I and II), although the input task size of the RANSAC part is rather small. As observed from *heteroGPU*$_1$ to *heteroGPU*$_2$, the execution time is decreased rapidly when shifting the RANSAC model computing part to the CPU side. However, this time cost is still large than that of CPU homogeneous execution (shown in Figure 6). Comparing the results of *heteroGPU*$_2$ and *heteroGPU*$_3$, we can see that the execution of the atomic operations consumes longer time when they are processed on the CPU side, which means the parallelization of these data aggregation tasks are worthwhile on GPU. As a consequence, the optimal *heteroGPU*$_4$ execution excludes RANSAC part and processes all the other steps on GPU side, which takes an overall time consumption of less than 3 ms.

The runtime evaluation of CPU-FPGA heterogeneous executions is shown in Figure 8. Note the logarithmic scale of the y-axis. The execution time of Partition III turns out an apparent decline when the model computation of the RANSAC fitting is processed on the host CPU (from *heteroFPGA*$_1$ to *heteroFPGA*$_2$), while there exists a minor increase in the time cost of Partition IV. From the executions of *heteroFPGA*$_2$ and *heteroFPGA*$_3$, it is clearly seen that FPGA suffers much more than GPU when performing the atomic operations on device side. Removing the data aggregation tasks can decrease the runtime cost of Partition I and II by 73.69% and 72.59%, respectively. Different to GPU, the optimal CPU-FPGA execution (*heteroFPGA*$_4$) excludes both the aggregation workloads

TABLE IV: Configurations of the heterogeneous execution.

Configuration	WPM	GS	ED	CONV	HT	RANSAC	
						MC[1]	ME[2]
$heteroGPU_1$	✓	✓	✓	✓	✓	✓	✓
$heteroGPU_2$	✓	✓	✓	✓	✓		✓
$heteroGPU_3$	✓		✓	✓			✓
$heteroGPU_4$	✓	✓	✓	✓	✓		
$heteroFPGA_1$	✓	✓	✓	✓	✓	✓	✓
$heteroFPGA_2$	✓	✓	✓	✓	✓		✓
$heteroFPGA_3$	✓		✓	✓			✓
$heteroFPGA_4$	✓		✓	✓			

[1] RANSAC model computation.
[2] RANSAC model evaluation.

(image grayscaling and simplified Hough transformation) and RANSAC model fitting from the OpenCL kernels.

3) Optimal execution: As observed in Table IV and Figure 8, the optimal CPU-FPGA heterogeneous execution contains two kernels and the WPM kernel consumes more time than that of the image convolution kernel. Consequently during the optimization more resources are allocated to the WPM kernel. Figure 9 presents the performance of the single- and multi-core implementation, as well as the optimal OpenCL-based designs. As for the configuration of the CPU-FPGA heterogeneous execution, the `num_compute_units` attribute of the WPM and convolution kernel is respectively set as 12 and 4, and the resource utilization information is detailed in Table V.

As depicted in Figure 9, all the 4 different implementations are capable of processing 640×480 videos real-timely. The CPU-GPU execution achieves the best performance of 169.8 fps, while the CPU-FPGA execution can process the video stream with time cost of 7.07 ms per frame. Nevertheless, the CPU-only implementations cannot handle 1920×1080 images with a performance demand of 30 fps. The heterogeneous executions can fulfill this requirement, resulting in a speedup of 8.87× and 5.21× over the single-core implementation, respectively on GPU and FPGA platforms.

D. Discussion

Experiments about the optimization of the OpenCL-based implementations reveal that GPU and FPGA exhibit different characteristics when performing the same task load. Our evaluation results show that FPGA is more sensitive to atomic data aggregation operations, while both GPU and FPGA are unsuitable to parallelize lightweight workloads. For lightweight workload, both platforms show a larger overhead when transferring data between the host and device, compared to the processing of the task itself. However, the atomic data aggregation operations turn out a severe performance bottleneck for the FPGA platform, which in the other way still deserves to being accelerated by GPU.

Compared to the previous work in [2], the improved design in this paper can robustly detect multiple lane markings with a

TABLE V: Resource usage of the OpenCL kernels on FPGA.

Resource	Used	Total	Utilization
ALMs	232952	427200	55%
Registers	481135	1708800	28%
RAM Blocks	2247	2713	83%
DSP Blocks	740	1518	49%

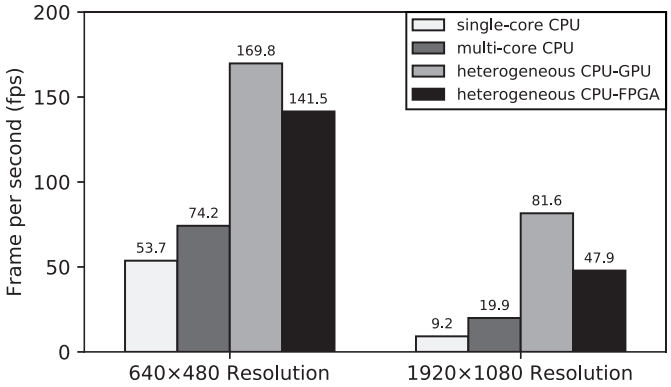


Fig. 9: Performance with different parallel configurations.

higher accuracy. Moreover, in our design the scalability of the algorithm can be easily demonstrated via tuning the number of RANSAC iterations, resulting in a tradeoff of accuracy and runtime performance. Last but not least, the heterogeneous parallel executions can deal with HD 1920×1080 video inputs while meeting the real time performance constraint.

One possible improvement of the accuracy lies in the WPM top-view transformation, which can be adapted by rotating the BEV image to achieve vertical and parallel lane markings. Thereby the initial guesses from the simplified Hough transformation would be more reliable and accurate.

VI. Conclusion

In this paper we propose an improved design of a lane detection algorithm which adopts RANSAC model fitting to detect multiple lanes on top-view transformed BEV image. We overcome the critical drawbacks in the previous work including inflexible ROI, susceptibility to drastic road gradient change and unknown camera parameters. By virtue of off-line homography matrix adaption to bound an adaptive ROI for subsequent on-line WPM transformation, the improved algorithm can robustly process arbitrary on-road videos with a higher accuracy. To address the scalability of this algorithm, we present a comprehensive parallel implementation of the application across multi-core CPU, GPU and FPGA platforms. Each potentially parallel task is elaborately implemented and optimized so that they are mapped to the most suitable processor so as to achieve optimal performance. The comparative experiments of using homogeneous and heterogeneous configurations indicate that GPU and FPGA exhibit significantly different runtime behaviours when dealing with atomic data aggregation tasks. Last but not least, our optimized parallel design can achieve a faster-than-real-time performance when processing HD images.

Acknowledgment

This work is supported in part by the scholarship from China Scholarship Council (CSC) under the Grant Number 201506270152 and Beijing University of Chemical Technology Talent Foundation (Grant No. BUCTRC201811).

References

[1] Khronos Group, "OpenCL Overview," https://www.khronos.org/opencl/, 2017.

[2] M. Aly, "Real time detection of lane markers in urban streets," in *IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2008, pp. 7–12.

[3] B. Dorj and D. J. Lee, "A precise lane detection algorithm based on top view image transformation and least-square approaches," *Journal of Sensors*, vol. 2016, 2016.

[4] J. Niu, J. Lu, M. Xu, P. Lv, and X. Zhao, "Robust lane detection using two-stage feature extraction with curve fitting," *Pattern Recognition*, vol. 59, pp. 225–233, 2016.

[5] S. Sakjiraphong, A. Pinho, M. N. Dailey, M. Ekpanyapong, and A. Tavares, "Real-time road lane detection with commodity hardware," in *Proceedings of the International Electrical Engineering Congress*. IEEE, 2014, pp. 1–4.

[6] S. Zhu, J. Wang, T. Yu, and J. Wang, "A method of lane detection and tracking for expressway based on ransac," in *IEEE 2nd International Conference on Image, Vision and Computing*. IEEE, 2017, pp. 62–66.

[7] K. Huang, B. Hu, J. Botsch, N. Madduri, and A. Knoll, "A scalable lane detection algorithm on cotss with opencl," in *Design, Automation and Test in Europe Conference and Exhibition (DATE)*. IEEE, 2016, pp. 229–232.

[8] H. Tan, Y. Zhou, Y. Zhu, D. Yao, and K. Li, "A novel curve lane detection based on improved river flow and ransa," in *IEEE 17th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2014, pp. 133–138.

[9] H. Xuan, H. Liu, J. Yuan, and Q. Li, "Robust lane-mark extraction for autonomous driving under complex real conditions," *IEEE Access*, 2017.

[10] J. Son, H. Yoo, S. Kim, and K. Sohn, "Real-time illumination invariant lane detection for lane departure warning system," *Expert Systems with Applications*, vol. 42, no. 4, pp. 1816–1824, 2015.

[11] H. Yoo, U. Yang, and K. Sohn, "Gradient-enhancing conversion for illumination-robust lane detection," *IEEE Transactions on Intelligent Transportation Systems (TITS)*, vol. 14, no. 3, pp. 1083–1094, 2013.

[12] W. Lu, S. A. Rodriguez F., E. Seignez, and R. Reynaud, "Monocular multi-kernel based lane marking detection," in *IEEE 4th Annual International Conference on Cyber Technology in Automation, Control, and Intelligent Systems*. IEEE, 2014, pp. 123–128.

[13] S.-N. Kang, S. Lee, J. Hur, and S.-W. Seo, "Multi-lane detection based on accurate geometric lane estimation in highway scenarios," in *IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2014, pp. 221–226.

[14] T. T. Duong, C. C. Pham, T. H.-P. Tran, T. P. Nguyen, and J. W. Jeon, "Near real-time ego-lane detection in highway and urban streets," in *IEEE International Conference on Consumer Electronics-Asia*. IEEE, 2016, pp. 1–4.

[15] R. Gopalan, T. Hong, M. Shneier, and R. Chellappa, "A learning approach towards detection and tracking of lane markings," *IEEE Transactions on Intelligent Transportation Systems (TITS)*, vol. 13, no. 3, pp. 1088–1098, 2012.

[16] X. An, E. Shang, J. Song, J. Li, and H. He, "Real-time lane departure warning system based on a single fpga," *EURASIP Journal on Image and Video Processing*, vol. 2013, no. 1, p. 38, 2013.

[17] R. Marzotto, P. Zoratti, D. Bagni, A. Colombari, and V. Murino, "A real-time versatile roadway path extraction and tracking on an fpga platform," *Computer Vision and Image Understanding*, vol. 114, no. 11, pp. 1164–1179, 2010.

[18] M. Hahnle, F. Saxen, M. Hisung, U. Brunsmann, and K. Doll, "Fpga-based real-time pedestrian detection on high-resolution images," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2013, pp. 629–635.

[19] P. Georgiev, N. D. Lane, C. Mascolo, and D. Chu, "Accelerating mobile audio sensing algorithms through on-chip gpu offloading," in *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 2017, pp. 306–318.

[20] W. Shi, X. Li, Z. Yu, and G. Overett, "An fpga-based hardware accelerator for traffic sign detection," *IEEE Transactions on Very Large Scale Integration Systems (VLSI)*, vol. 25, no. 4, pp. 1362–1372, 2017.

[21] H. Kong, J.-Y. Audibert, and J. Ponce, "General road detection from a single image," *IEEE Transactions on Image Processing (TIP)*, vol. 19, no. 8, pp. 2211–2220, 2010.

[22] Intel, "Intel® FPGA SDK for OpenCL™," https://www.altera.com/en_US/pdfs/literature/hb/opencl-sdk/aocl-best-practices-guide.pdf, 2017.

[23] M. Aly. Caltech lanes. Accessed: 2018-04-17. [Online]. Available: http://www.vision.caltech.edu/malaa/datasets/caltech-lanes