ADAS on COTS with OpenCL: A Case Study with Lane Detection

Kai Huang, Biao Hu, Long Chen, Alois Knoll, and Zhihua Wang

Abstract—The concept of autonomous cars is driving a boost for car electronics and the size of automotive electronics market is foreseen to double by 2025. How to benefit from this boost is an interesting question. This article presents a case study to test the feasibility of using OpenCL as the programming language and COTS components as the underlying computing platforms for ADAS development. For representative ADAS applications, a scalable lane detection is developed that can tune the trade-off between detection accuracy and speed. Our OpenCL implementation is tested on 14 video streams from different data-sets with different road scenarios on 5 COTS platforms. We demonstrate that the COTS platforms can provide more than sufficient computing power for the lane detection in the meanwhile our OpenCL implementation can exploit the massive parallelism provided by the COTS platforms.

Index Terms—Advanced driver assistance systems (ADAS), commercial of the shelf (COTS), OpenCL, FPGA, GPU

1 INTRODUCTION

AUTOMOTIVE IC sales had been estimated to represent 7.3 percent of the total $ 287.1 billion IC market in 2015 [13]. The IC Market Drivers Report states that from 2014 through 2019, the automotive electronics market is forecast to increase at an average annual growth rate of 6.7 percent, highest among six major end-use applications—computer, consumer, communications, automotive, industrial/medical, and government/military—and two points more than the 4.3 percent CAGR forecast for the total IC industry over the same time period. More specific, the automotive IC market is forecast to grow 7 percent to $ 22.2 billion in 2016 and continues increasing to $ 29.2 billion in 2019 [13]. The driving force of this boost in automotive electronics is the concept of autonomous cars. According to the consensus of most Tier-1 OEMs, fully autonomous cars will present to market by 2018-2020 [19]. Their arrival will help double by 2025 the size of todays automotive semiconductor market. In additional, according to Linley Gwennap, president of the Linley Group, an expected $ 10,000 premium for a self-driving car will be paid for lots of cameras and processing horsepower. The question here is how Tier- 2 and 3 suppliers, particular for car electronics suppliers, to benefit from this boost.

Freescale, one of the leading car electronics suppliers, states two critical issues on the path toward autonomous driving: the lack of open standards for Advanced Driver Assistance Systems (ADAS) development and safety guarantee of consumer-focused silicon chips for safety-critical autonomous applications [9].

- *K. Huang and L. Chen are with the School of Data and Computer Science, Sun Yat-sen University, Xiaoguwei Island, Panyu District, Guangzhou 510006, China. E-mail: {huangk36, chenl46}@mail.sysu.edu.cn.*
- *B. Hu and Z. Wang are with the Institute of Microelectronics, Tsinghua University, Beijing 100084, China. E-mail: hubiao_007@outlook.com, zhihua@tsinghua.edu.cn.*
- *A. Knoll is with the Institute of Robotics and Embedded Systems, Technical University Munich, Boltzmannstr. 3, Garching 85748, Germany. E-mail: knoll@in.tum.de.*

Manuscript received 25 Apr. 2016; revised 13 Aug. 2017; accepted 11 Sept. 2017. Date of publication 3 Oct. 2017; date of current version 19 Mar. 2018.
(Corresponding author: Biao Hu.)
Recommended for acceptance by D. Zeng.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TC.2017.2759203

Responding to the current lack of open standards and to reverse the trend toward closed, proprietary ADAS systems which inhibit development and design innovation, Freescale has announced it will soon introduce an Open Computing Language (OpenCL)-based automotive development environment, targeting its own silicon. The motivation of Freescale' move is to reduce ADAS R&D overhead and free their customers to focus more on ADAS innovation, and more importantly, ADAS safety, according to Bob Conrad, Freescale SVP and General Manager, Automotive MCUs [9]. OpenCL [12] is an open, royalty-free standard maintained by the non-profit technology consortium Khronos Group. As OpenCL is designed for cross-platform, parallel programming, and improving speed and responsiveness for a wide spectrum of applications in numerous markets, we consider as well OpenCL could be a suitable programming model for ADAS development.

Regarding the underlining platforms that execute ADAS applications, although Freescale claims that consumer-oriented silicon solutions designed to enhance gaming graphics or run smartphone apps are not safe enough to ensure autonomous driving-quality and reliability, we consider the mass-production commercial off-the-shelf (COTS) components will be the de-facto carriers for ADAS. The reasons are multi-fold. First, the time-to-market and development cost can be significantly reduced by using COTS, compared to traditional dedicated ECU/ASIC-implementations. Second, autonomous driving requires significantly higher computing power than those available via special-purpose controllers equipped with safety features. Nowadays COTS components, on the other hand, can provide massive computing power and the performance per price can be even a magnitude higher than automotive-specialized ones [11]. Third, COTS platforms start to support OpenCL for heterogeneous computing, which allows ADAS developers to exploit the computing power of COTS in the meanwhile trades off different design objectives, e.g., performance and energy consumption. Therefore, the use of high-performance commodity COTS in safety-critical systems becomes desirable. The safety issue that Freescale believes to hinder consumer-oriented silicon solutions for ADAS can and will be solved by the technology advance and a short survey on COTSs for safety-critical systems is presented in the related work section to justify this claim.

To test the feasibility of using OpenCL for ADAS development on COTS platforms, this article conducts a case study. We choose five different COTS platforms which are not specifically designed for automotive markets, i.e., Nvidia GeForce GTX 660 Ti and Quadro K600, Altera Stratix V A7 and a resource-restricted Cyclone V SoC FPGA, and Redmi Note 2 mobile phone. All these five platforms support OpenCL such that source code written in OpenCL can be executed on them without major modifications. This helps for a fair comparison. In addition, Nvidia Jetson TK 1, which is specially designed and targeted for automotive market, is used for comparison as well. With this case study, we try to answer following questions: 1) Can OpenCL be a standard for cross-platform development and applications developed with OpenCL be smoothly applied on different COTS platforms? 2) Can an ADAS application, which is developed using OpenCL, exploit the parallelism provided by COTS platforms? 3) Can COTS platforms provide enough computing power for ADAS applications? For the safety issue, it is not a focus of this article and we left for the related work for a brief discussion.

For ADAS applications, lane detection is chosen. Lane detection is one of the most basic functions for ADAS and it has grabbed significant attention in research since mid-1980's. New development on computer technologies, however, has allowed new perspectives on designing a good lane detection algorithm. For example, as the software in a car is expected to run up to 1 GB of software [3], it is preferable to design a scalable algorithm to leave rooms for multiple ADAS applications on single ECU [4]. The scalability here means the demanding power of the algorithm can be tuned to trade off

between the accuracy and speed. The reason is that lane detection, at certain circumstance, can be particularly computationally demanding due to, e.g., varying light conditions, traffic on the road that obstructs the lane markings, and the shadows cast by buildings or trees. Therefore, such scalability has practical need in real life. On the other hand, with this capability to scale the computing demand, we can use this algorithm to exploit the computational limits of the tested COTSs. The contributions are summarized as follows:

- With an OpenCL implementation, we tested our scalable lane detection [14], [15] on five different COTS platforms with 14 video streams from different data-sets representing different road scenarios. We demonstrate that our OpenCL implementation can exploit the massive computing power provided by the COTS platforms. With an average deviation fewer than 5 pixels, the average frame rates can reach about 600 fps on Nvidia GeForce GTX 660 Ti and Altera Stratix V A7 for the 14 test videos. The peak frame rates for certain videos on the GeForce GPU can reach over 1,000 fps. On the low-budgeted Quadro K600, the average frame rates can be more than 200 fps. Even on the resource-restricted Altera Cyclone V SoC FPGA and Redmi Note 2 mobile, the average frame rates can still reach real time at acceptable accuracy.
- We profiled and analyzed the execution of our application and show the detailed timing of the execution on the 5 COTSs platforms. In addition, we extended existing OpenCL runtime environment to support Altera FPGA and demonstrate the heterogeneous execution of our lane detection with a combination of Nvidia GeForce GPU and Altera Stratix FPGA.
- To compare the performance between COTSs and automotive-specific hardware, Nvidia TK 1 is chosen, which is specially designed and targeted to automotive market for ADAS applications. Since OpenCL is not supported by TK 1 for the moment, we re-implemented our lane detection with CUDA and ran all the test cases with the CUDA implementation on TK 1. By comparing the measurement results, we demonstrated the usefulness of COTSs platforms for ADAS applications.

The rest of this paper is organized as follows: The next section reviews related work in the literature. Section 3 provides a short introduction of OpenCL programming and runtime environment. Section 4 describes the experimental setup and Section 5 presents experimental results. Section 6 concludes the paper.

2 Related Work

Early discussion on building systems with COTSs can be dated back to [2], where, from a software engineering perspective, some basic understanding of how developing systems with COTS products were described and why and what new capabilities were being identified. From the perspective of hardware platforms, Kotaba et al. [18] systematically analyzed the effects of COTS multi-core platforms—the applications compete for resource access, typically arbitrated in a non-explicit manner by specific hardware implementation, which causes non-deterministic temporal delays on execution—and suggested mitigation techniques, where possible.

Inline with the aforementioned guideline, there is generic work on providing temporal isolation on COTS platforms without specifying concrete application domains. In [21], a run-time fixed-size weighted DMA transaction was suggested to provide temporal separation for hardware-based I/O virtualization on PCIe. With such virtualization, secure and safe sharing of I/O subsystems can be guaranteed for COTS multi-core systems. In [23], an integrated approach is presented for temporal partitioning and WCET analysis of COTS multi-core systems. Workload monitoring at basic block level at runtime is used to ensure that the respective process does not introduce further interference on the affected resource. In [11], a so-called software coded processing technique, i.e., detecting transient, permanent, and systematic hardware execution errors by arithmetic encoding of variables, constants, and operations, is deployed to demonstrate that COTS hardware can be used in safety-critical applications.

There is also work targeting specific safety-critical domain, e.g., automotive, railway, and aerospace. Shen et al. [30] demonstrated that a teleoperation system can be exclusively composed of low-cost COTS components yet still meet the high performance demands of remotely driving a car on the road with wireless communication over 3G/4G data networks. In [8], an attempt to design an automotive ECU using a Xilinx Zynq-7000 FPGA, rather than an MCU-based platform, which conforms to both the AUTOSAR and ISO 26262 standards, was presented. Cohen et al. [5] showed the feasibility of hard real-time and parallel execution of safety-critical tasks on the Xilinx Zynq COTS. A train signaling usecase provided by Alstom Transport was programmed in an extension of the synchronous dataflow language HEPTAGON and a software stack and composition methodology were designed to enable hard real-time control code to be isolated from timing interference. Further, a recent book [16] examines radiation effects for FPGA and GPU on aerospace applications. The authors concluded that even FPGA and GPU are very sensitive to radiation, such COTS platforms can still meet the reliability requirements with the help of certain fault-tolerance techniques.

There are also quite a few recent research projects working on deterministic multi-core architectures, e.g., MERASA [36], parMERASA [35], ACROSS [28], RECOMP [27] and T-CREST [32]. The MERASA architecture uses either round robin [26] or IABA arbitration [25] to guarantee an upper bounded for inter-core interference. The parMERASA approach presents time predictable on-chip ring architectures to achieve composable WCET on a many core architecture [24]. The RECOMP project uses IDAMC NoC [20] to support mixed critical applications. The ACROSS project employs time triggered NoC for time predictable access to the shared memory. Similar to ACROSS, the T-CREST project aims to build a time predictable NoC architecture [31], [33].

From semiconductor industry, Freescales latest Qorivva MPC5643L was the first multi-core to achieve a formal ISO 26262 certificate for ASIL-D functional safety capability by an independent third-party accredited certification body in the industry. Infineon in 2012 introduced a 32-bit TriCore multi-core processor to meet safety and powertrain requirements of automotive industry. In November 2015, Altera announced a lockstep solution for its Nios II embedded processors to enable the implementation of SIL-3 safety designs in Altera FPGAs in full compliance with functional safety standards IEC 61508 and ISO 26262 [34]. Nvidia and Huawei are also newcomers of automotive industry that are entering this market [6], [22] and we expect soon the announcement of their safety-critical solutions.

From all these R&D activities, We believe that the concern of Freescale on the safety issue of consumer-oriented silicon solutions for ADAS can and will be solved by the technology advance. COTS platforms will be the de-facto carriers for ADAS applications and thus we focus on the performance and programming issues in this article.

3 OpenCL Programming

Open Computing Language [12] is a standard for heterogeneous high performance computing managed by the non-profit technology consortium Khronos Group. It defines a high level abstraction layer for low level hardware instructions. This enables cross-platform execution from general purpose processors to massively parallel devices without changing the source code.

3.1 OpenCL Framework

The OpenCL specification provides specific view and rules to organize the execution of applications. An OpenCL application runs on

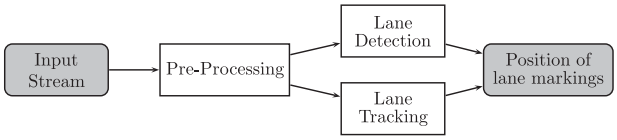

Fig. 1. Lane detection algorithm.

a *host* system which is connected to one or more accelerators *devices*. The host coordinates the execution and the devices are capable of executing C-like OpenCL code as accelerators. The computing acceleration by OpenCL is achieved by the means of the following four models in OpenCL framework.

Platform Model. OpenCL abstracts the combination of host and devices as a single *platform*. Any platform has only one host and one or more than one devices. The host is often a CPU of a system. The device can be any accelerators, such as Gpu and Fpga. A device is considered to be made of blocks of several *compute units* and each compute unit is made up of several processing element. According to this model, a processing element is the smallest entity and it resembles a *thread* in traditional parallel programming terminology.

Execution Model. There are two kinds of executions of OpenCL, i.e., *host execution* and *device execution*. The host execution is mainly responsible for assigning jobs to devices, and making use of results from devices. The device execution is called the *kernel* that is called by host and executed in parallel in device. The kernels are mostly written in the *OpenCL C* programming language, a dialect of the C99 standard and compiled with a vendor-specific compiler, but native kernels are optionally supported as well. They describe the sequence of instructions within a single execution instance, called a *work-item*. Work-items that are grouped in the same *work-group* are executed concurrently.

Memory Model. The memory hierarchy consists of four distinct regions for the kernels: *Global* memory that can be written and read by all work-items in all work-groups. *Constant* memory, a region of global memory that is initialized by the host and does not change during the execution. *Local* memory that is only accessible to work-items within the same work-group. *Private* memory owned by a single work-item and not visible by others.

Program Model. OpenCL supports *Data-Parallel* and *Task-Parallel* models. Data-Parallel model involves a kernel operating on multiple elements of a data structure concurrently. Task-Parallel model can be defined as a set of independent programs operating concurrently.

3.2 Scalable Lane Detection

Our lane detection algorithm contains three parts, namely video pre-processing, lane detection, and land tracking, as shown in Fig. 1. The video stream showing a road and the area surrounding it will be processed in two subsequent steps frame by frame. First, information on the lane markings is amplified and extracted from each frame in the pre-processing step. Then, depending on whether previous estimates of the position exist or not, the exact position of the lane markings is detected or tracked in a lane detection or lane tracking steps. Note that the proposed method is vision-based and requires no knowledge of any physical parameters like position and orientation of the camera.

All three parts displays a high potential for parallel computing. For instance in the pre-processing, Grayscaling and Thresholding can be applied to each pixel independently and the use of the Sobel Filter requires knowledge of eight neighbouring pixels. In detection and tracking, the sampling of the candidate lines, the calculation of their intensity weights, and the prediction and importance weight update of one particle can all be done in parallel. Therefore, three OpenCL kernels are built for three parts respectively. Detailed explanation of our approach is skipped due to space limit and is referred to [14], [15].

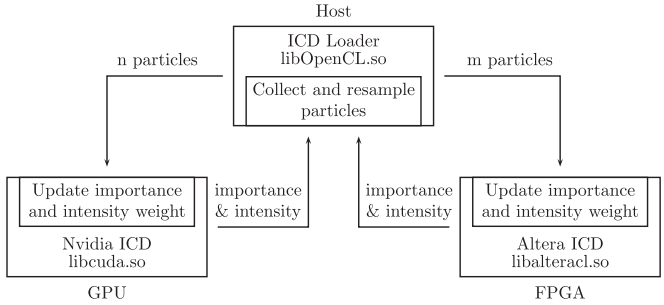

Fig. 2. Relation of Icd libraries in the case of lane tracking.

3.3 OpenCL Runtime

OpenCL has been implemented on a wide range of Gpu devices, such as Nvidia and ATi Gpu. In recent years, Fpga providers also start to provide tools to generate hardware design specification from OpenCL kernels, e.g., Altera Complete Design Suite [7] or Xilinx AutoESL [29]. To allow multiple OpenCL implementations from different vendors to co-exist on the same system, there is a `cl_khr_icd` extension[17]. It defines an *installable client driver (ICD) loader*, a unifying library that acts as a mediator between the different platforms. This enables the use of multiple heterogeneous devices in a single process without interference between implementations on different devices. Without this mechanism, the overheads of multiple processes and inter-process communication (IPC) between them are required.

The Icd loader libraries act as a proxy between the user program and the actual implementations. It employs the C programming language library *dl* that is used on Linux to load libraries at runtime. Due to the absence of Icd loader in Altera AOCL 13.0sp1, a self-developed Icd loader was developed and implemented. The relationships between the libraries are shown in Fig. 2, where different vendor implementations can be loaded at runtime without the symbol conflicts. This allows a heterogeneous execution of our application, i.e., distributing the workload around Gpu and Fpga during runtime.

The communication between the host and devices is organized in a server-client model, as shown in Fig. 2. The host distributes the workload between the devices and collects the results when they are ready. This distribution is adjustable at runtime because frames are processed one after another independently.

4 Evaluation Setup

This section describes the setup of our case study. Details of the chosen Cotss and test videos are depicted.

We choose five different Cots platforms, i.e., an Nvidia GeForce GTX 660 Ti Gpu and an Altera Stratix V A7 Fpga (Nallatech PCIe385n_a7 card) mounted on a desktop with an Intel Intel Xeon processor, an Nvidia Quadro K600 on another desktop with an Intel Core 2 Quad desktop, a standalone Altera Cyclone V SoC Fpga, and a Redmi Note 2 mobile phone. The Cyclone SoC is embedded with a Cortex-A9 dual-core as the host. The Redmi Note 2 mobile uses the MediaTek Helio X10 octa-core on 2 GHz [1], with an on-chip PowerVR G6200 Gpu. The detailed specifications of these platforms are listed in Table 1.

The choices of Cotss for evaluating the application performance are follows. First of all, the Cots platforms should be diverse to test the heterogeneity support of OpenCL. From this regard, Gpus and Fpgas, which represent two structurally different programming platforms, are chosen. In addition, a mobile Gpu is tested as well. Second, we intend to test the scalability of our approach and extract the limits for a wider range of platforms, from powerful high-end to low-budgeted low-cost platforms. For high-end platforms, GTX 660 Ti Gpu and Altera Stratix V A7 Fpga are chosen. For budgeted platforms, Quadro K600 Gpu and Altera Cyclone V

TABLE 1
Hardware Specification of the COTS Platforms

	FPGA1	FPGA2	GPU1	GPU2	GPU3
Model	Nallatech 385-D5	Cyclone V SoC	GeForce GTX 660 Ti	Quadro K600	PowerVR G6200
Architecture	Stratix V 5SGXA7	ST 5CSTD6	Kepler GK104	Kepler GK 107	Mediatek Helio X10
Driver version	13.1	13.1	304.88	340.58	Android 5.0
Host CPU	Intel Xeon E31225	Cortex-A9 dual-core	Intel Xeon E31225	Intel Core 2 Quad Q9300	Cortex-A53 Octa-core
Host connection	PCIe	AMBA AXI	PCIe	PCIe	MTK MCSI
PCIe Generation	3.0	N/A	3.0	2.0	N/A
PCIe lanes	8x	N/A	16x	16x	N/A
On-board memory	8 GB DDR3	1 GB DDR3	2 GB DDR5	1 GB DDR3	2 GB DDR3
Max. Freq. (MHz)	600	210	915	876	700
Max. GFLOPS	294.7	56	2,460	336.4	89.6
Max. Power (W)	25 (board)	~4 (chip)	150 (board)	41 (board)	~4 (chip)
OpenCL version	1.1	1.1	2.0	1.2	1.2

SOC FPGA are chosen. The Redmi Note 2 mobile is another type of low-budgeted platform, as the mobile itself costs around $100. In addition, Nvidia Jetson TK 1, which is specially designed and targeted to automotive market for ADAS applications, is chosen for further comparison.

To test our lane detection application on these five COTS platforms, 14 video streams from different data-sets with different scenarios are used. The detailed information of these video streams is listed in Table 2 in [15]. These videos can respectively represent different road situations, including night, blurred lane, and broken lanes. In these 14 video streams, the third one and the fourth one come from Caltech Lanes Dataset, while all others are self-recorded.

5 EVALUATION RESULTS

This section presents experimental results and analyzes these results from different aspects.

5.1 Accuracy

We evaluate the impact of numbers of candidate lines (denoted as N_c) and particles (denoted as N_p) on the overall accuracy. Since our approach is a stochastic approach, the lane markings detected from a $N_c = 2^{14}$ and $N_p = 2^{12}$ setting are used as baselines. We evaluate different N_p and N_c combinations. Each combination is tested with the 14 videos in [15]. The numbers of pixels deviated from the baselines are reported.

The results are shown in Fig. 4a . From the figure, the first observation is that the accuracy increases when N_p increases. With $N_p = 128$, the average deviation is already below 4 pixels. Considering the neighborhood N_n is set to 10 pixels, i.e., a lane marking is represented by 20 pixels, we can conclude that our approach can achieve high accuracy. The second observation is that when $N_p \geq 512$ the deviation is saturated to 3 pixels. The reason for the saturation at 3 pixels is that particle filter is a statistical result. Since every time particles are randomly distributed on the image, there exists inherent deviation between any two detection results. Although our reference baselines are calculated by a large N_c and N_p, this reference baselines cannot guarantee to be the real lane markings. It can happen that real lane markings are obtained with smaller N_c and N_p during the runtime. Besides, the real lane on the image is not a strictly straight line, which will also result in some deviations between the detected lane and the real lane.

5.2 Performance

We also report the frame rates for the five COTS platforms in Table 1 for all N_p and N_c combinations in Fig. 4a . For each N_p and N_c combination, 14 videos are checked and the corresponding frame rates are recorded for every platform that runs at default frequency from the manufactory. The boxplots of frame rates [36] are shown in Figs. 4b , 4c , 4d , 4e , and 4f . The frame rate of every video is different because the video shows the road of different scenarios (easy or hard) and the video format is also not the same. It can be seen that, the GeForce GPU achieves the highest frame rates, as expected among all COTSS (Fig. 4b). It can reach over 1,000 fps for certain videos when the numbers of particles (N_p) are small. With $N_p = 512$, i.e., average deviation less than 3 pixels, the average frame rate can still reach more than 550 fps. For the Stratix V FPGA (Fig. 4e , the frame rates are lower than the GeForce GPU but still super fast, about average 400 fps for $N_p = 512$ and almost 1,000 fps for smaller N_p. Considering the Stratix is running at 200 Mhz, such performance is magnificent. The budgeted Quadro K600 GPU (Fig. 4c) can also reach relatively high performance, e.g., over 200 fps for average frame rate at $N_p = 512$. For the ultra-low power resource-restricted Redmi PowerVR GPU (Fig. 4d) and Cyclone V SoC FPGA (Fig. 4f), our algorithm can still reach real time. Considering the power consumption for these two chips is just 4 W, our algorithm is also energy efficient.

In addition, we also investigate the influence of ROI size on the frame rate. Four different ROI sizes are tested and the results are shown in Fig. 3. In these cases, $N_p = 256$ and $N_c = 512$ are adopted. From the figure, It can be seen that the ROI size has big influence on the frame rate. In general, the drops of the frame rates are sub-linear to the increases of the size of the ROI for devices. Nevertheless, our algorithm can still reach 77 fps on Stratix V FPGA for large ROI.

Furthermore, Nvidia Jetson TK 1 is used as a comparison to evaluate the performance. TK 1 is developed as a small super

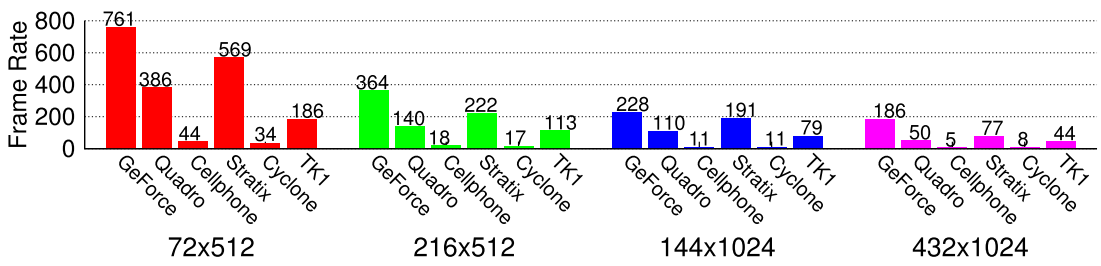

Fig. 3. Frame rate w.r.t. ROI.

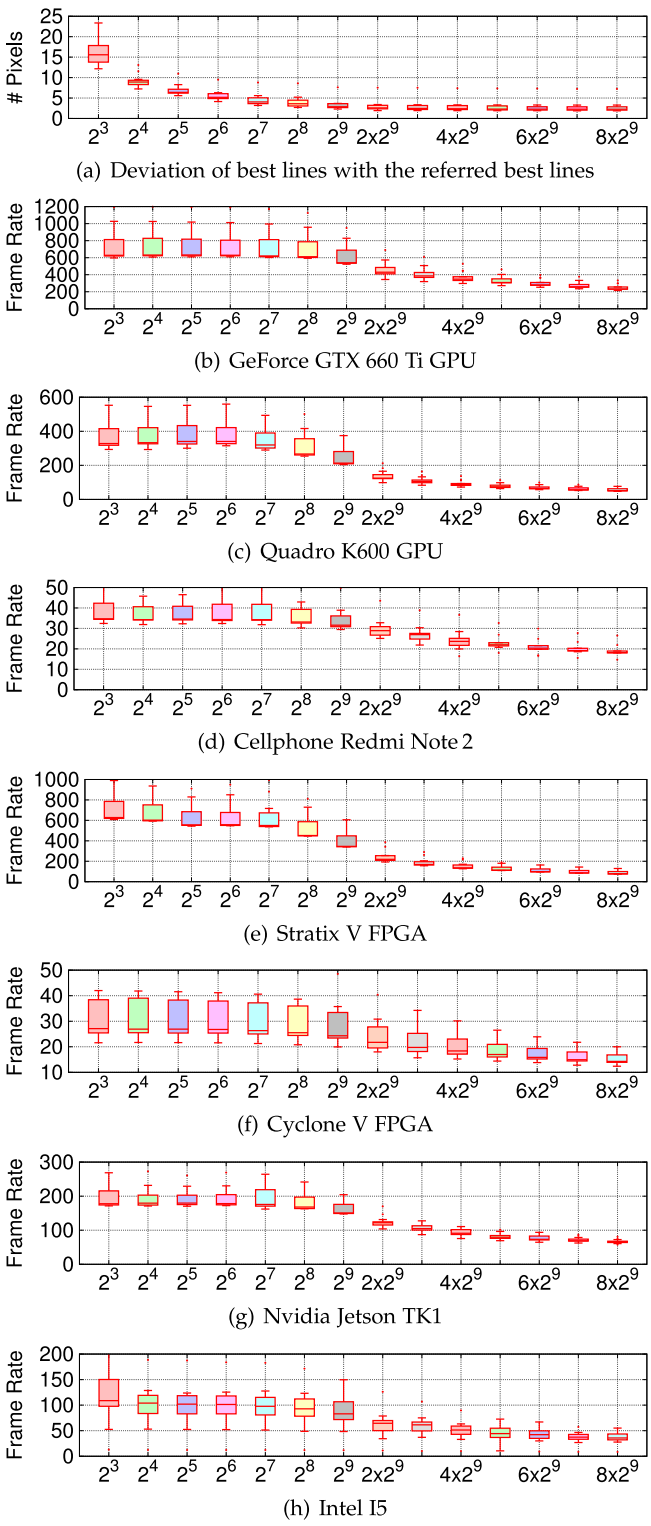

Fig. 4. Accuracy versus frame rate. The X axis is the number of particles N_p. For $N_p \leq 2^9$, $N_c = 2^9$, otherwise $N_c = 2^{12}$.

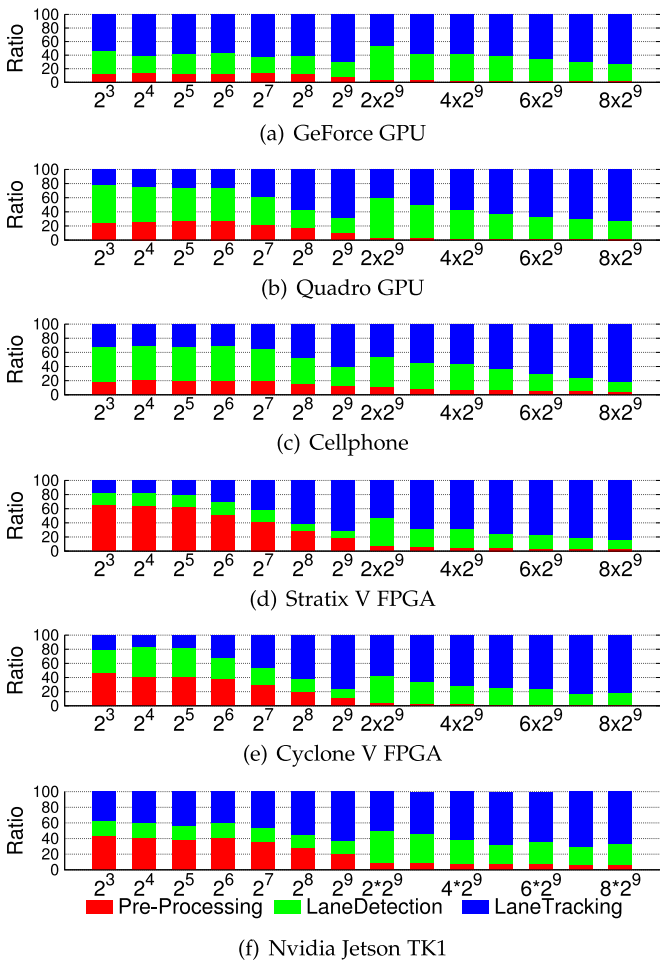

Fig. 5. Ratios of the average computation time of kernels on different platforms.

computer that aims at the use in autonomous driving and robot. Since it does not support OpenCL, our OpenCL code is ported into CUDA code and runs on TK 1. Compared to other GPUs in a PC platform, TK 1 architecture is optimized by using a share memory between CPU and GPU to save the time of data communication. Nevertheless, the frame rates (Fig. 4g) can only reach half of those on Quadro K600, though TK 1 and Quadro K600 have similar GFLOPS. Note that we ran the CUDA code on Quadro GPU as well to compare the OpenCL and CUDA implementations. From this comparison, we found the OpenCL implementation is only 10 percent slower than the CUDA implementation. This shows that our CUDA implementation is not the cause of the low frame rate on TK 1. We believe that the bottleneck of TK 1 is the host processor, a 32-bit quad-core Cortex A15 and its 64-bit memory data width.

5.3 Timing Analysis

In our approach, there are three OpenCL kernels, i.e., pre-processing, lane detection, and lane tracking. Therefore, we investigate the detailed timing of these three kernels for all the experiments in Fig. 4. The normalized average computation time of all N_c and N_p combinations is shown in Fig. 5. From this figure, we have following observations. In general, given fixed N_c, increasing N_p will increase the timing percentage of lane tracking. The reason is that there are more particles needed to be processed with larger N_p. Another reason is that with less particles, i.e., smaller N_p, the tracking is less accurate and need more times of detection. On the other hand, the workload for pre-processing is fixed for a given ROI. Therefore, the ratios for pre-processing drop further as the number of N_p increases. The second observation is the ratios are different for different platforms, given the same N_c and N_p combination. In general, the two FPGAS have higher percentages for pre-processing than the GPUS. The reason is that more floating point operations are involved during pre-processing.

Lets take a closer look and consider the case $N_p = 256$ and $N_c = 512$. The normalized average total execution time and execution time per frame are shown in Figs. 6 and 7, respectively. For the total execution time (Fig. 6), the major computation is expected spending on lane tracking. The reason is that average 95 percent

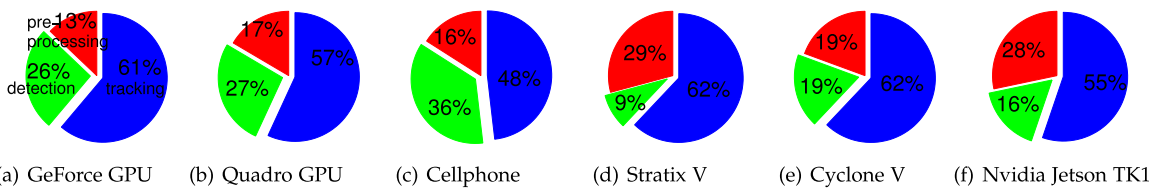

Fig. 6. Normalized average total execution time of OpenCL kernels for $N_p = 256$ and $N_c = 512$.

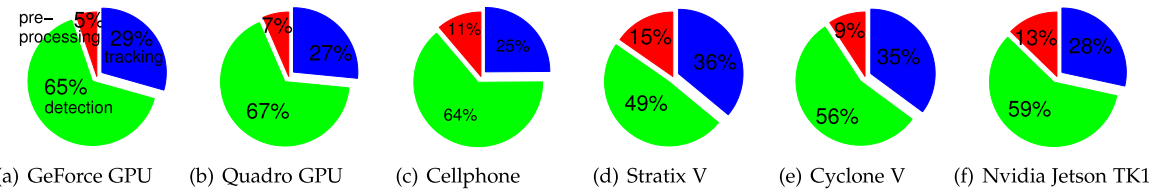

Fig. 7. Normalized execution time of OpenCL kernels for single frame for $N_p = 256$ and $N_c = 512$.

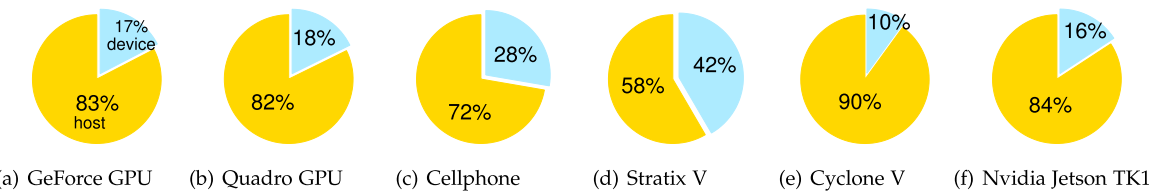

Fig. 8. Distribution of the computation time between host and devices.

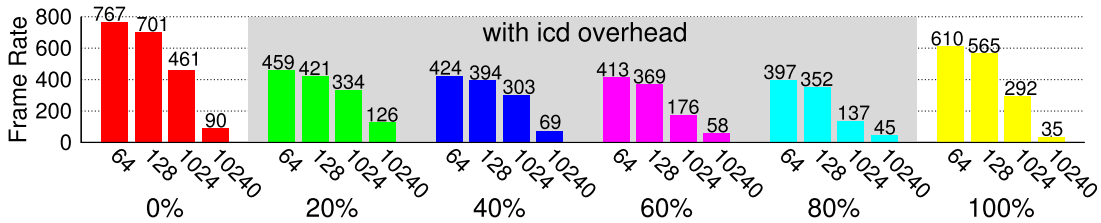

Fig. 9. Heterogeneous execution with Stratix FPGA and GeForce GPU, where the bottom numbers represent the percentage of particles on FPGA.

frames are conducted lane tracking. When we consider the computing time for individual frames (Fig. 7), lane detection consumes much more computing power than lane tracking, e.g., more than twice for GPUs, which justifies the need of a lane tracking step, rather than detection for every frames.

Fig. 8 shows the ratios of timing expense between the host CPU and devices. In general, the hosts have a bigger portion, i.e, about 80 percent. This means the hosts are always the bottleneck. The main reason for this bottleneck is the conversion of OpenCV structure of image frames into OpenCL peer. This conversion unfortunately cannot be paralleled with the devices. Comparing the GeForce GPU and Stratix V FPGA (Figs. 8a and 8d), who are mounted on the same machine with Intel Xeon processor, one can find the portion for the host is smaller (i.e., 58 percent) for Stratix V FPGA. The reason is that the Stratix V FPGA has much small computing resources than the GeForce GPU, i.e., slower. Although almost half computation time is spent on the Stratix V, the processing speed of Stratix V is still very fast as the frame rate for Stratix V is the second highest among all devices. The reason for the high percentage of Stratix V computation time is that the host processor is very powerful. For the case of Cyclone V, the host processor is a dual-core Cortex-A9, which results in the highest ratio for the host.

5.4 Heterogeneous Execution on Multiple Devices

This section presents experiments of heterogeneous execution of our lane detection with Altera's Stratix V FPGA and Nvidia's GeForce GTX 660 TI GPU in combination. The results are summarized in Fig. 9.

In this experiment, the number of sampled lines are set twice as the number of particles ($N_c = 2N_p$) and we tested four cases, each with different number of particles, i.e., N_p = 64, 128, 1,024, and 10,240. For each N_p, six scenarios are evaluated, i.e., the percentage of N_p computed on FPGA is set to be 0, 20, 40, 60, 80, and 100 percent. When this percentage is 0 and 100 percent, all computations are solely executed on GPU and FPGA, respectively. In these two cases, there is no ICD overhead during the test because the ICD is not used when only one device is used.

There are three main observations from Fig. 9. First, the frame rates decrease as more workload are on the FPGA for most of the cases. This is because the GeForce GPU has higher computer power than the Stratix FPGA, as shown in Figs. 4b and 4e. Second, the ICD overhead is non-trivial. Comparing scenarios 0 percent (without ICD) and 20 percent (with ICD) for cases 64, 128, and 1,024, the frame rates drop significantly, i.e., 40.16, 34.52, and 27.55 percent, respectively. While comparing 20 to 40 percent, 40 to 60 percent, and 60 to 80 percent, the drops on frame rates are not significant as 0 to 20 percent. On the other hand, the frame rates increase again from 80 percent (with ICD) to 100 percent (without ICD). Third, the speed-up in computation by distributing workload to both devices is effective only when particles number is sufficiently large where single device cannot cope with the required workload. Comparing scenarios 0 to 20 percent, only in the case of 10,240 particles, the frame rate increases from 90 to 126, while the frame rates drop for all other cases. It indicates that the benefit of distributing workload overcomes the ICD overhead and thus increases the frame rate. In other cases, the workload distribution cannot compensate the ICD overhead and only results in performance decrease.

6 Conclusion

This article presents a case study for running lane detection application on COTSS with OpenCL. From this case study, we draw

following conclusions. First, our lane detection developed with OpenCL can smoothly execute on both Nvidia GPUs and Altera FPGAs, individually and jointly. Although getting ideal performance on Redmi mobile needs customized optimization, such optimization does not affect the basic of the source code. Regarding the effort of coding in OpenCL, it took 6 months for a master student without any prior knowledge of parallel programming to develop a working prototype of the lane detection. It took another 4 months for a bachelor student without no knowledge of OpenCL to optimize the memory consumption as well as the FPGA execution. Therefore, we can confidently say OpenCL can be considered as a viable standard programming model for ADAS development. Second, as shown the experiments, all the examined COTSs can provide more than sufficient computing power for lane detection. With proper manners that can exploit the available parallelism, we believe COTS platforms can provide enough computing power to support ADAS applications, even multiple applications jointly on single platforms. In summary, Using OpenCL to develop parallel ADAS applications on COTS platforms is a viable solution for future ADAS development.

ACKNOWLEDGMENTS

This work is partly supported by SYSU fund: 46000-31650002.

REFERENCES

[1] GPU GFLOPS for Game Consoles/ARM, X86 SoC. 2016. [Online]. Available: http://kyokojap.myweb.hinet.net/gpu_gflops/
[2] L. Brownsword, D. Carney, and T. Oberndorf, "The opportunities and complexities of applying commercial-off-the-shelf components," *Crosstalk*, vol. 11, no. 4, pp. 4–6, 1998.
[3] M. Broy, I. Kruger, A. Pretschner, and C. Salzmann, "Engineering automotive software," *Proc. IEEE*, vol. 95, no. 2, pp. 356–373, Feb. 2007.
[4] M. Buechel, et al., "An automated electric vehicle prototype showing new trends in automotive architectures," in *Proc. Int. Conf. Intell. Transp. Syst.*, Sep. 2015, pp. 1274–1279.
[5] A. Cohen, V. Perrelle, D. Potop-Butucaru, M. Pouzet, E. Soubiran, and Z. Zhang, "Hard real time and mixed time criticality on off-the-shelf embedded multi-cores," in *Proc. 8th Eur. Congr. Embedded Real Time Softw. Syst.*, Jan. 2016, pp. 1–10.
[6] W. Cunningham, "BMW adopts Nvidia GPU for in-car displays," 2011. [Online]. Available: http://www.cnet.com/news/bmw-adopts-nvidia-gpu-for-in-car-displays/
[7] T. Czajkowski, et al., "From OpenCL to high-performance hardware on FPGAS," in *Proc. Int. Conf. Field Programmable Logic Appl.*, 2012, pp. 531–534.
[8] F. Fons and M. Fons, "FPGA-based automotive ECU design addresses AUTOSAR and ISO 26262 standards," *Xcell J.*, vol. 78, pp. 20–31, 2012.
[9] Freescale, "OpenCL ADAS development environment addresses safety concerns," 2015. [Online]. Available: http://automotive.electronicspecifier.com/
[10] J. Fritsch, T. Kuhnl, and A. Geiger, "A new performance measure and evaluation benchmark for road detection algorithms," in *Proc. 16th Int. IEEE Conf. Intell. Transp. Syst.*, 2013, pp. 1693–1700.
[11] M. Ghadhab, J. Kaienburg, M. Süßkraut, and C. Fetzer, "Is software coded processing an answer to the execution integrity challenge of current and future automotive software-intensive applications?" in *Advanced Microsystems for Automotive Applications*. Berlin, Germany: Springer, 2016, pp. 263–275.
[12] K. Group, "OpenCL," [Online]. Available: http://www.khronos.org/opencl/
[13] IC Market Drivers Report. 2016. [Online]. Available: http://www.icinsights.com/
[14] K. Huang, B. Hu, J. Botsch, N. Madduri, and A. Knoll, "A scalable lane detection algorithm on COTSs with OpenCL," in *Proc. Des. Autom. Test Eur. Conf. Exhib.*, Mar. 2016, pp. 229–232.
[15] K. Huang, B. Hu, L. Chen, A. Knoll, and Z. Wang, "ADAS on COTS with OpenCL: A case study with lane detection," Inst. Für Informatik, Technische Universität München, München, Germany, Tech. Rep. TUM-I1636, May 2016.
[16] F. Kastensmidt and P. Rech, Eds., *FPGAs and Parallel Architectures for Aerospace Applications*. Berlin, Germany: Springer, 2016.
[17] Khronos Group, "OpenCL Extension #5: Installable client driver (ICD) loader," 2010. [Online]. Available: https://www.khronos.org/registry/cl/extensions/khr/cl_khr_icd.txt. Retrieved on: Mar. 15, 2015.
[18] O. Kotaba, J. Nowotsch, M. Paulitsch, S. M. Petters, and H. Theiling, "Multicore in real-time systems–temporal isolation challenges due to shared resources," in *Proc. Workshop Ind.-Driven Approaches Cost-effective Certification Safety-Critical Mixed-Criticality Syst.*, 2014, pp. 1–6.
[19] T. Litman, "Autonomous vehicle implementation predictions," *Victoria Transport Policy Institute*, vol. 28, 2014.
[20] B. Motruk, J. Diemer, R. Buchty, R. Ernst, and M. Berekovic, "IDAMC: A many-core platform with run-time monitoring for mixed-criticality," in *Proc. IEEE 14th Int. Symp. High-Assurance Syst. Eng.*, Oct. 2012, pp. 24–31.
[21] D. Muench, M. Paulitsch, and A. Herkersdorf, "Temporal separation for hardware-based I/O virtualization for mixed-criticality embedded real-time systems using PCIe SR-IOV," in *Proc. 27th Int. Conf. Archit. Comput. Syst.*, Feb. 2014, pp. 1–7.
[22] P. Muyanözçelik and V. Glavtchev, "GPU computing in tomorrow's automobiles," 2008. [Online]. Available: http://www.nvidia.com/content/nvision2008/tech_presentations/Automotive_Track/NVISION08-GPU_Computing_in_Tomorrows_Automobiles.pdf
[23] J. Nowotsch, M. Paulitsch, A. Henrichsen, W. Pongratz, and A. Schacht, "Monitoring and WCET analysis in COTS multi-core-SoC-based mixed-criticality systems," in *Proc. Des. Autom. Test Eur. Conf. Exhib.*, Mar. 2014, pp. 1–5.
[24] M. Panić, G. Rodriguez, E. Quiñones, J. Abella, and F. J. Cazorla, "On-chip ring network designs for hard-real time systems," in *Proc. 21st Int. Conf. Real-Time Netw. Syst.*, 2013, pp. 23–32.
[25] M. Paolieri, E. Quiñones, F. J. Cazorla, G. Bernat, and M. Valero, "Hardware support for WCET analysis of hard real-time multicore systems," *SIGARCH Comput. Archit. News*, vol. 37, pp. 57–68, Jun. 2009.
[26] M. Paolieri, E. Quinones, F. J. Cazorla, and M. Valero, "An analyzable memory controller for hard real-time CMPs," *IEEE Embedded Syst. Lett.*, vol. 1, no. 4, pp. 86–90, Dec. 2009.
[27] P. Pop, L. Tsiopoulos, S. Voss, C. F. Oscar Slotosch, U. Nyman, and A. R. Lopez, "Methods and tools for reducing certification costs of mixed-criticality applications on multi-core platforms: The recomp approach," in *Proc. Workshop Ind.-Driven Approaches Cost-Effective Certification Safety-Critical Mixed-Criticality Syst.*, 2013, pp. 1–6.
[28] C. E. Salloum, M. Elshuber, O. Hoeftberger, H. Isakovic, and A. Wasicek, "The ACROSS MPSoC—A new generation of multi-core processors designed for safety-critical embedded systems," in *Proc. 15th Euromicro Conf. Digit. Syst. Des.*, Sep. 2012, pp. 105–113.
[29] K. Shagrithaya, K. Kepa, and P. Athanas, "Enabling development of OpenCL applications on FPGA platforms," in *Proc. IEEE 24th Int. Conf. Appl.-Specific Syst. Archit. Processors*, 2013, pp. 26–30.
[30] X. Shen, et al., "Teleoperation of on-road vehicles via immersive telepresence using off-the-shelf components," in *Proc. 13th Int. Conf. Intell. Auton. Syst.*, 2016, pp. 1419–1433.
[31] R. B. Sorensen, M. Schoeberl, and J. Sparso, "A light-weight statically scheduled network-on-chip," in *Proc. NORCHIP*, Nov. 2012, pp. 1–6.
[32] J. Sparso, "Design of networks-on-chip for real-time multi-processor systems-on-chip," in *Proc. 10th Int. Conf. Appl. Concurrency Syst. Des.*, 2012, pp. 1–5.
[33] J. Sparso, E. Kasapaki, and M. Schoeberl, "An area-efficient network interface for a TDM-based network-on-chip," in *Proc. Des. Autom. Test Eur. Conf. Exhib.*, 2013, pp. 1044–1047.
[34] K. Taylor, "Altera functional safety package combines FPGA flexibility with lockstep processor solution to reduce risk and time-to-market," 2015. [Online]. Available: http://newsroom.altera.com/press-releases/nr-altera-functional-safety-yogitech.htm
[35] T. Ungerer, et al., "parMERASA—Multi-core execution of parallelised hard real-time applications supporting analysability," in *Proc. Euromicro Conf. Digit. Syst. Des.*, Sep. 2013, pp. 363–370.
[36] T. Ungerer, et al., "Merasa: Multicore execution of hard real-time applications supporting analyzability," *IEEE Micro*, vol. 30, no. 5, pp. 66–75, Sep. 2010.