High-Speed Scene Flow on Embedded Commercial Off-the-Shelf Systems

Long Chen, *Member, IEEE*, Mingyue Cui, Feihu Zhang, Biao Hu, *Member, IEEE*, and Kai Huang, *Member, IEEE*

***Abstract*—Scene flow is an essential part of a stereo-based perception system for autonomous driving and mobile robotics. As in most of these platforms, the computing resource is limited but the computing requirement is high, embedded and parallelized algorithms are of vital importance for real-time tasks. This paper develops a cross-platform embedded scene flow algorithm by using an OpenCL (Open Computing Language) programming. Meanwhile, we propose a method to achieve a good performance by using a novel coarse-grained software pipeline for the embedded stream application. Experimental results show that the proposed algorithm can boost the average processing speed to 50 fps for different commercial off-the-shelf (COTS) hardware, including desktop graphics processing units (GPUs), field-programmable gate arrays (FPGAs), and mobile phone platforms. For certain GPUs, the peak frame rates can also reach 1000 fps. By comparing the efficiency among the serial platform, we illustrate that with the help of OpenCL programming, COTS platforms can provide enough computing resources for the stereo-based perception algorithm.**

***Index Terms*—Commercial off-the-shelf (COTS), FPGA, GPU, Open Computing Language (OpenCL), scene flow.**

I. Introduction

THREE-DIMENSIONAL (3-D) scene flow estimates objects 3-D geometry as well as its 3-D motion information directly from a sequence of stereoscopic images. 3-D scene flow estimation has a wide range of applications, including mobile robotics, self-driving, etc. It contains two basic subparts: stereo matching and optical flow computation, which have been researched for decades.

3-D scene flow, with the purpose of estimating both the 3-D geometry and 3-D motion from stereoscopic image sequences, is a key function for many important tasks of autonomous driving and mobile robotics where research contents include scene understanding, moving-object detection, and obstacle avoidance. The 3-D scene flow contains two basic subparts: stereo matching and optical flow computation, which have been researched for a long time [1]–[3]. In terms of flow density, scene flow can be categorized as sparse scene flow [4]–[6] or dense scene flow [7], [8]. In general, the sparse scene flow is an intermediate step of the dense scene flow, by offering the initial sparse estimation of some key points. For instance, the dense scene flow proposed in [9] needs the sparse scene flow [5] as initial input.

Nowadays, heavy workload versus limited resources is a critical contradiction in industries, especially in mobile robotics. To solve this problem, embedded and onboard algorithms have become popular. Additionally, the products of low-cost and powerful semiconductors accelerate the use of COTS in domain-specific applications. For reducing the overall manufacture cost, it is very attractive to use COTS instead of traditional dedicated application specific integrated circuit (ASIC) implementations. Moreover, different COTSs lead to different hardware platforms, such as FPGA, GPU, and CPU, or the combinations of them. In order to adapt to multiple COTSs easily, the best choice is to develop a cross-platform method. The technique using the COTS platforms to execute the algorithm could benefit its future ASIC [10], which has become a significant practical issue for industrial processes.

A coarse-grained software pipeline for the sparse scene flow was presented by us in [11]. We chose OpenCL because it is an industry standard for parallel computing, which supports heterogeneous hardware. Therefore, the source code that is written in OpenCL can be executed on GPUs, FPGAs, and multi-core CPUs, without major modifications. This is beneficial to design software running on heterogeneous COTS hardware for unmanned vehicles in the future. That will replace the traditional dedicated electronic control unit. The proposed method is tested on both GPUs and FPGAs with the OpenCL implementation. The average frame rate can achieve about 400 frames per second (fps) on the NVIDA graphics card GeForce GTX Titan in different road scenarios. An average frame rate of 950 fps and a maximum frame rate of 1000 fps can be obtained on NVIDIA graphics GTX 1070. The average frame rate can be still up to 25 fps on mobile devices. This paper extends our previous work [11] in the following three aspects.

1) We test on various platforms and reveal the influence of different parameters of the proposed algorithm.

Manuscript received March 17, 2017; revised September 7, 2017 and May 8, 2018; accepted July 24, 2018. Date of publication August 7, 2018; date of current version April 3, 2019. This work was supported by the National Natural Science Foundation of China under Grant 61301277. Paper no. TII-17-0478. *(Corresponding author: Kai Huang.)*

L. Chen, M. Cui, and K. Huang are with the School of Data and Computer Science, Sun Yat-sen University, Guangzhou 510275, China (e-mail: chenl46@mail.sysu.edu.cn; cuimy@mail2.sysu.edu.cn; huangk36@mail.sysu.edu.cn).

F. Zhang is with the School of Marine Science and Technology, Northwestern Polytechnical University, Xi'an 710072, China (e-mail: feihu.zhang@nwpu.edu.cn).

B. Hu is with the College of Information and Science, Beijing University of Chemical Technology, Beijing 100029, China (e-mail: hubiao@mail.buct.edu.cn).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TII.2018.2864173

2) We compare the serial and parallel experimental results and demonstrate the necessity and superiority of our algorithm.
3) A comprehensive analysis of the execution performance of our algorithm is also presented.

To sum up, the contributions of this paper are as follows.

a) A coarse-grained software pipeline for the scene flow method is designed, and we implement the method on different embedded COTS platforms with real-time performance.
b) The scene flow method is tested on nine COTS platforms with ten video streams, which represent different road scenarios with an OpenCL implementation. The experimental results show that the proposed implementation can take advantage of the parallelism provided by the COTS platform. With an average frame rate of 25 fps, even on mobile devices, it is still able to achieve real-time performance.
c) We verified the following two facts: First, stereo-based perceptual methods could be implemented on the COTS platform and even on mobile platforms; second, OpenCL can be viewed as a standard programming language for both cross-platform and parallel advanced driver assistance systems (ADAS).

The remaining parts of this paper are as follows. The next section reviews related works. Section III presents details of the proposed method. We describe the evaluation setup in Section IV. The experimental results are shown in Sections V, and Section VI concludes the paper.

II. Related Work

The term "scene flow" was first introduced by Vedula *et al.* [12], who defined it as a 3-D motion field of points in the world. This approach first estimated 2-D optical flow for all views, and then fitted the final 3-D flow to them. In the early period of scene flow, there were also several researchers recasting the problem as a 2-D disparity flow. Wedel *et al.* [7] estimated optical flow for a reference view and the disparity differences for the other views. Huguet *et al.* [13] first viewed the geometry and flow as an integrated procedure. Traditionally, the problem is formulated as a variational framework. Vogel *et al.* [14] designed a regularizer encouraging locally rigid motion.

There are also scene flow methods based on RGBD (Red, Green, Blue, Depth) sensors [7], [8], [15], [16]. Letouzey *et al.* [17] were the first to use Kinect to implement scene flow. Wedel *et al.* [7] explicitly dissociated the position and velocity estimation and calculated dense velocities using variational approach. A real-time dense scene flow was introduced in [18] with a primal-dual framework. However, being limited by the short measuring distance, RBGD-based methods are not suitable for outdoor perception.

Influenced by recent trends in the field of optical flow [19], [20], segment-based scene flow has attracted more and more attention. This kind of method explicitly modeled the discontinuities and inferred or refined the segmentation together with the scene parameters. Yamaguchi *et al.* [21] contributed to segment-based stereo by penalizing deviation. Recently, methods with an assumption of rigid moving planar scene have also achieved remarkable results [9]. In these approaches, the scene is segmented into planar segments with consistent motion and each pixel was with a planar segment. Then, the scene flow was modeled as a discrete-continuous optimization problem with continuous rigid 3-D motion parameters of the planar segment to be optimized. Vogel *et al.* [22] proposed a slanted plane model and used α-expansion to solve the optimization problem, and favorable results have been achieved on the KITTI benchmark used here, which is based on the work of stereo and optical flow benchmarks [23]. However, despite the achievements mentioned above, a drawback of this kind of method is that they are usually computationally expensive.

Training a convolutional neural network (CNN) to compare image patches was inspired by recent success of CNN [24], [25]. Outstanding results had been achieved in various tasks. LeCun *et al.* [26] proposed a stereo matching approach by training a CNN model to compare image patches, which could be introduced for scene flow computation potentially. However, CNN-based methods are too time-consuming to be practical.

To date, there is a lack of an embedded real-time scene flow method, which could be applied in outdoor environment. Geiger *et al.* [5] introduced an effective sparse scene flow method, which has nearly real-time performance on PC platform. Based on this work, we propose a cross-platform, parallel, real-time scene flow method, which achieves favorable performance on several different kinds of COTS embedded systems.

III. OpenCL Based Scene Flow

A. Scene Flow

The scene flow used here is based on the work of Geiger *et al.* [5], where feature detection, extraction, and feature matching are three core parts. For feature detection, we select 5×5 spots and corner markers [27] to filter the images. Afterwards, nonmaximum and nonminimum suppression (NMS) [28] are used to select feature points. The pixel corresponding to the local minimum and maximum values of the filter response is considered to be a feature point. Additionally, the image is further filtered by two Sobel cores [29], with each in vertical and horizontal directions, respectively. Feature descriptor consists of its pixel value and its 11×11 neighboring Sobel filters, which are represented by eigenvectors. The feature points and their corresponding descriptors are calculated from the left and right images between two consecutive frames. Similarly, in the matching step, we perform a circular match, as shown in Fig. 1. The characteristic feature points of the left image in the frame t (previous frame) is first tracked and matched by the $M \times M$ search window to the right image in the frame t. This procedure is repeated for the stereo images in frame $t+1$ (current frame) and ends with the left image in frame t. If the feature matches correctly in all of the above frames, the circle is closed and the characteristics in the last frame coincide with the characteristics in the first frame. If the feature matches fail, the circle will not close, and the feature point is rejected as an outlier. All

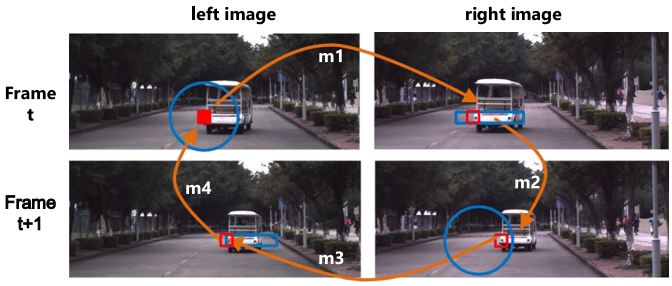


Fig. 1. Scene flow feature matching illustration.

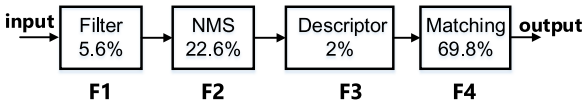


Fig. 2. Sequential flow of the scene flow algorithm and the distribution of computation time.

of the matching errors are represented by the sum of absolute differences (SAD).

B. Coarse Grained Software Pipeline Parallelization

Scene flows sequential computation contains four cascaded modules, i.e., the filtering, NMS approach, descriptor extraction, and feature matching (see Fig. 2). For simplicity, we name them as $F1$, $F2$, $F3$, and $F4$. During serial computing, if the search radius of NMS and matching are set to 9 and 200 pixels, percentages of each module in the total processing time are 5.6%, 22.6%, 2%, and 69.8%, respectively. Since the modules $F4$ and $F2$ totally take over 90% of the time, they need to be optimized thoroughly.

Here, we describe our parallelization method as well as the optimization framework. As mentioned above, there are four key kernels in our algorithm, i.e., $F1$, $F2$, $F3$, and $F4$; $F21$ is the output. Considering that the scene flow is a streaming program, we perform data parallelism first. The relationship of these four modules can be found in Fig. 3(a). We first segment the non-dependence data from the input, which is shown in Fig. 3(b). Then, we modify the logic of each module until they satisfy the requirements of parallelization. Especially for module $F4$, i.e., a serial match process with four points, which is the most time-consuming module, we design a coarse-grained software pipeline for parallel optimization. First, we split the tasks of F4 into $f4\_1$, $f4\_2$, $f4\_3$, $f4\_4$ after the data parallelization, where, $f4\_1$ represents the match between left image and right image in frame t, labeled as $m1$ in Fig. 1. 4_2, 4_3, and 4_4 are also defined analogously (see Fig. 1). The coarse-grained pipeline is shown in Fig. 3(c).

As mentioned above, the feature match is a circle fashion including m1, m2, m3, m4, so here we are going to take the four processes corresponding to four functions: $f4\_X1$, $f4\_X2$, $f4\_X3$, $f4\_X4$. Thus, we use the four layers pipeline structure and each time there are four parallel execution threads at most. For time 1 ($t1$), only $f4\_1$ for circle m1 ($f4\_11$) is executed, and the left and right image are available at this moment. For time 2, $f4\_2$ for circle m1 ($f4\_21$) is executed while the $f4\_1$ for circle m2 ($f4\_12$) is executed parallelly. In a similar case, $f4\_31$, $f4\_22$, and $f4\_13$ are executed parallelly at time 3. Every time we add a circle, the number of parallel times will be doubled. The first feature match ($f4\_1$) is ended at step 4, and the second feature match ($f4\_2$) is ended at time 5, and so on. The computing results are saved in device end before each execution of $f4\_X4$. The pseudocode of parallelization by coarse-grained software pipeline is shown in Algorithm 1. This reduces the consumption of data from the host to the device. In principle, operating efficiency of $F4$ could be improved for nearly four times by using this coarse-grained pipeline. Using coarse-grained pipeline, we can achieve real-time processing even though this pipeline is not real-time architecture itself.

Algorithm 1: Parallelization by Coarse-Grained Software Pipeline.

Input: Characteristic descriptors for the upper level pipeline
Output: New descriptors

function _Kernel_Matchimage_row, image_col
  $feature\_num \leftarrow Get\_Global\_Size()$
  $feature\_id \leftarrow Get\_Global\_Id()$
  $\_local\_\ int\ detect[feature\_num * detect\_size]$
  $\_Barrier\_$
  for $cnt = 0 \rightarrow detect\_size - 1$ **do**
    $detect[pos + cnt] \leftarrow image\_orig[resp\_pos]$
  end for
  $\_Barrier\_$
  $desptr[size] \leftarrow data\_from\_pipe()$
  $val \leftarrow CalSAD(desptr \rightarrow feature\_point)$
  for $cnt = 0 \rightarrow detect\_size - 1$ **do**
    $temp\_cal \leftarrow CalSAD$
    $(detect[feature\_pos + cnt])$
    if $val = temp\_val$ **then**
      $next\_desptr \leftarrow desptr(detect[pos + cnt])$
      $Send\_Data\_To\_Pipe(next\_desptr)$
    end if
  end for
end function

IV. Evaluation Setup

A. Qualitative Performance

We first test our algorithms performances on both KITTI and SYSU dataset, the results are shown in Fig. 4, where the arrow represents the motion direction. The direction, length, and color of the arrow represent the motion direction, the velocity and the depth of the point, respectively. Red color presents the pixel with large depth, and green means the pixel with small depth. The top eight figures show the experimental performance of the street scenario in Table II, and the others illustrate the test results of campus scenarios.

From the figure, we can see that from left to right, the number of feature points is getting less and less, which means the scene

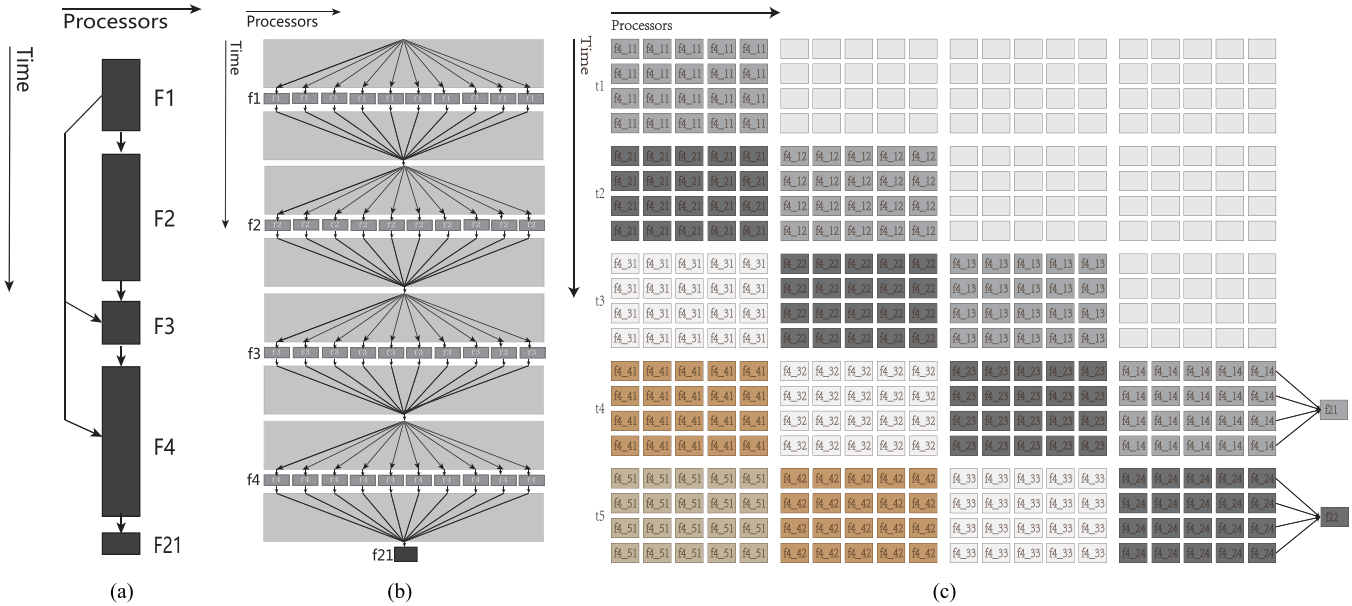


Fig. 3. Parallel execution models for stream programs and the coarse-grained pipeline strategy. (a) Sequential. (b) Data parallel. (c) Coarse-grained pipeline.

TABLE I
HARDWARE SPECIFICATIONS OF THE COTS PLATFORMS

	GPU1	GPU2	GPU3	GPU4
Model	GeForce GTX Titan	GeForce GTX 1070	GeForce GTX 760	Quadro K600
Architecture	Kepler GK 110	Pascal	Kepler	Kepler GK 107
Driver version	361.42	367	367	340.58
Host CPU	Intel Xeon E5-2620	Intel Xeon E5-2620	Intel Xeon E5-2620	I5-4460T
Host connection	PCIe	PCIe	PCIe	PCIe
PCIe Generation	3.0	3.0	3.0	2.0
PCIe lanes	16x	16x	16x	16x
On-board memory	6GB DDR3	8GB GDDR5	2GB GDDR5	1GB DDR3
Max. Freq. (MHz)	837	1506	1000	876
Max. GFLOPS	4,494	6500	2260	336.4
Max. Power (W)	250 (board)	150 (board)	170 (board)	∼41 (chip)
OpenCL version	2.0	2.0	2.0	2.0
GPU5	**GPU6**	**GPU7**	**GPU8**	**FPGA**
Qualcomm Adreno 530	Mali- T628MP4	Mali-T760MP8	Qualcomm Adreno 430	Nallatech 385-D5
Snapdragon 820	HiSilicon KIRIN 920	Exynos 7 Octa 7420	Snapdragon 810	Stratix V D5
Android 6.0.1	Android 5.1	Android 6.0	Android 6.0	15.0
Quad-core Kryo	Cortex-A15, Cortex-A7	Exynos 7420	Cortex-A57, Cortex-A53	Intel Xeon E5-2620
MTK MCSI	MTK MCSI	MTK MCSI	MTK MCSI	PCIe
3.0	2.0	N/A	N/A	N/A
N/A	N/A	N/A	N/A	8x
3GB DDR4	3GB DDR3	4GB DDR4	3GB DDR4	8 GB DDR3
624	600	772	600	600
407.4 498.5	70.4	189	N/A	388.8 408
N/A	N/A	N/A	N/A	∼25 (chip)
2.0	1.1	1.2	2.0	2.0

flow is getting sparser as the nms_n value increases. The more detailed analysis is given in next section.

Our method can successfully extract moving objects as well as their motion information from the SYSU dataset. In addition, we compare the original algorithm with the optimized algorithm, and the ratio of different feature points is shown in Fig. 5. By contrast, we can see that the differences in quality performance of the original and optimized code are very small and the ratio average only 1.2%. Our attention focuses on optimizing acceleration without changing the original performance.

B. Test Cases

We test our algorithm on nine different kinds of COTS platforms. The selection of COTSs are based on the following: First, GPUs (including mobile GPUs) and FPGAs are selected as the representation of two structurally different platforms. Second, we plan to test the scalability of our proposed method from high-end to low-cost platforms. Here, Titan GPU, GTX 1070, and GTX 760 are selected for high-end platforms. For budgeted platforms, Quadro K600 GPU and Altera Stratix V A7 FPGA

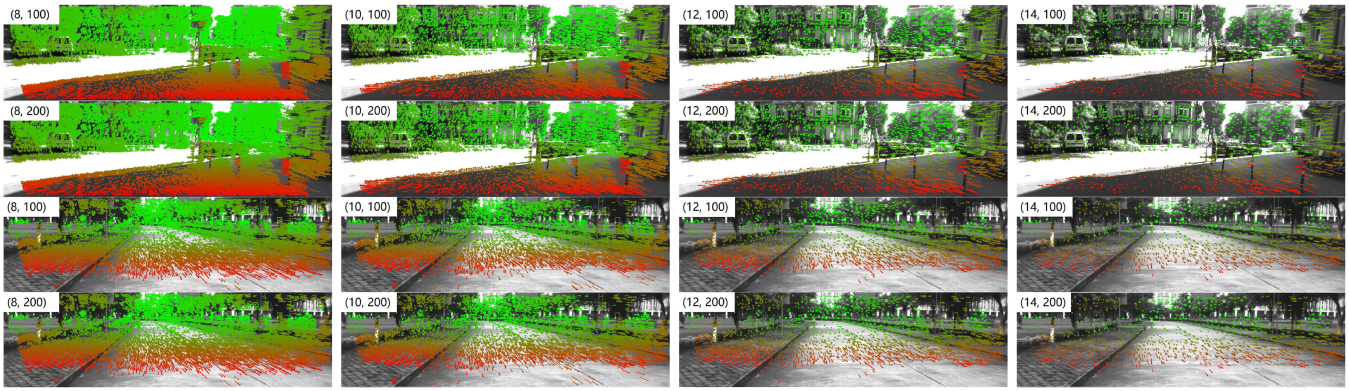


Fig. 4. Scene flow detection on the KITTI dataset (top rows) and the SYSU dataset (bottom rows) for different nms_n and $match\_radius$, which are marked on the upper left.

TABLE II
DETAILED INFORMATION OF TEST VIDEOS

Videos	1	2	3	4	5	6	7	8	9	10
Total frames	300	300	300	291	319	372	319	300	300	372
Resolution	1226X370	1241X376	1299X374	1299X374	1298X367	1299X374	1241X376	1241X376	1241X376	1241X376
Scenario	city	street	highway	highway	campus	jam	spacious	turning	countryside	street

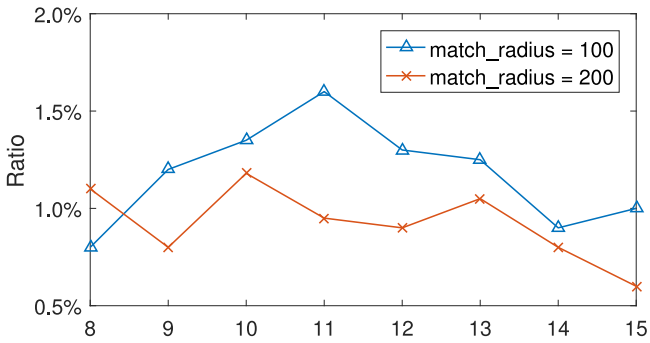


Fig. 5. Ratio of different feature points between the original algorithm and the optimized algorithm.

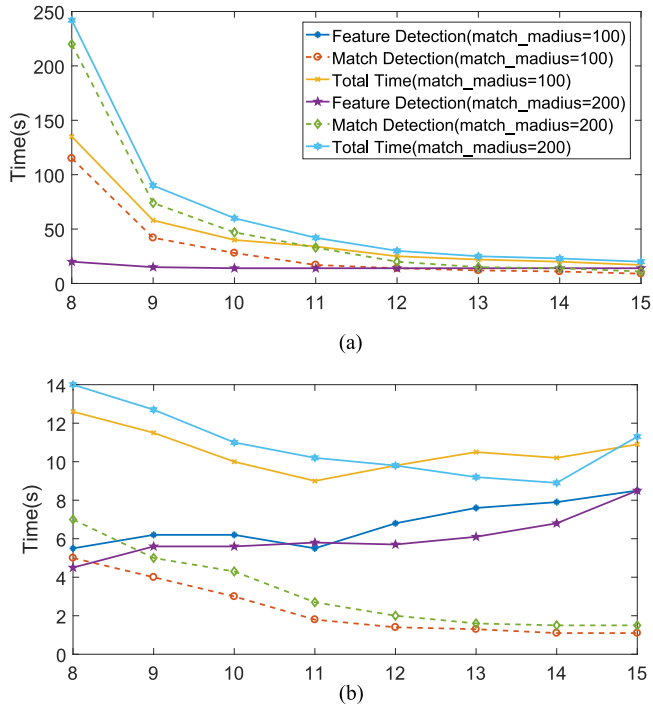


Fig. 6. Comparison of the execution time (in second) on the CPU and parallel MI5 mobile phone across different nms_n denoted on x-axis. (a) The performance of Intel Xeon E5-2620 CPU. (b) The performance of MI5 mobile phone.

are chosen. The MI5 mobile phone, Samsung Galaxy Note5 mobile phone, Sony Z5 mobile phone, and Honor6 mobile phone are also chosen as the other low-cost platforms.

We test ten videos on all these nine COTS platforms. The detailed information of these video streams is listed in Table II. These videos represent different road conditions, ranging from city, street to countryside. In these ten video streams, the third and fourth one come from the KITTI dataset [30], while the other videos are recorded in the Sun Yet-sen University campus, named the SYSU dataset.

V. EVALUATION RESULTS

A. Performance

There are four OpenCL kernels, i.e., $Filter$, NMS, $Descriptor$, and $Matching$, which have been mentioned before. Among these four kernels, there are two main parameters having great impact on the performance and efficiency in our algorithm. One is the nms_n which is the search range in the NMS kernel. The other is the search radius of match kernels termed as $match\_radius$.

We also compare the execution time of original serial scene flow with our parallelled one on KITTI dataset, which is shown in Fig. 6. The top one is the execution time of serial algorithm on Intel Xeon E5-2620 and the bottom one is the execution time

of parallel algorithm on MI 5 mobile phone. For the feature detection of different $match\_radius$, the result on Intel Xeon E5-2620 is very close, which makes the curve difficult to distinguish. Actually, the computing capability of Xeon E5-2620 is much better than MI 5. However, the experimental results are the following.

1) The execution time of serial algorithm on Intel Xeon E5-2620 cannot achieve real-time performance, but the execution time of our parallel algorithm on mobile phone platform can average 45 fps and 20 times faster.
2) As the nms_n increases and the number of feature points decreases, the matching time and total time decrease obviously.
3) The larger match_radius is, the more the execution time is.

Performances of frame rate on nine different COTS platforms are shown in Fig. 7. The videos in Table II are tested on every platform for each nms_n and $match\_radius$ combination. It can be seen that the GTX 1070 can reach up to 1000 fps and an average frame rate of 950 fps. The GeForce Titan can obtain an average frame rate of 400 fps and the maximum frame rate of certain videos is 450 fps. As a new generation of graphics cards, the frame rate on GTX 1070 is nearly twice of the frame rate on GTX Titan. The reason is that the GTX 1070 has a new Pascal architecture, which is with qualitative improvement both in the performance and processing efficiency. Previous generation GPUs used 28 nm CMOS techniques, and GTX 1070 uses 16 nm CMOS techniques. The performances of GTX 760 and GTX Titan are similar, but GTX 760 is more floating. For the K600 GPU, the frame rates are lower than the GeForce GPU, with an average frame rate of 80 fps. At the same time, we also tested the processing capacity of four models of mobile phone with different CPUs and GPUs.

In the platform of mobile, the Mali-T760MP8 from the Samsung Galaxy Note5 performs almost the same performance as the Quadro K600 GPU. It can reach up to 60 fps for certain videos and an average frame rate of 45 fps. And the Qualcomm Adreno 530 from Mi5 mobile is as good as the Mali-T760MP8, but its average frame rate is only 40 fps.

As the latest addition to the Qualcomm Adreno 530, the Qualcomm Adreno 430 on the Sony Xperia Z5 Premium mobile is able to delivers up to 65 fps with an average frame rate of 45 fps. It can be slightly better than the Qualcomm Adreno 430 performance, but it is a little volatile. Mali T628MP4 in Honor 6 mobile phone performance is relatively low, with the average frame rate of about 18 fps. Stratix V FPGA platform is also studied, but performance is not very satisfactory.

Among the most platforms, the frame rate exhibits parabolic performance and reaches the highest value when the nms_n is located between 11 and 12. When nms_n increases, the number of the matching points decreases, which makes the GPUcomputing time down. However, when the number of matching points reaches a certain level, the data transmission time becomes the main influencing factor, and the frame rate decreases. The frame rate of Sony Xperia Z5 Qualcomm Adreno 430 is in the range from 12 to 65 fps. The performance of Qualcomm Adreno 530 is better than that of 430, because Adreno

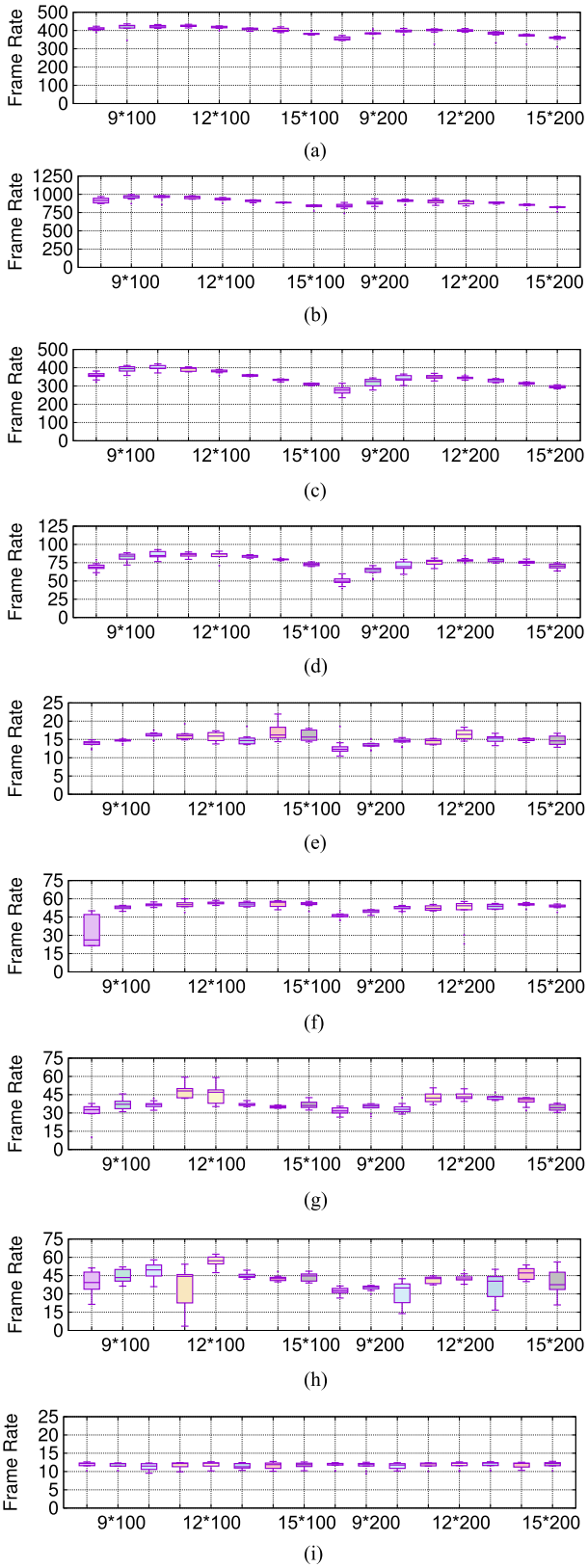


Fig. 7. Performance of frame rate on nine different COTS platforms under the different nms_n (from 8 to 15) and $match\_radius$. For the front half, $match\_radius = 100$. For the latter half, $match\_radius = 200$. (a) GTX Titan. (b) GTX 1070. (c) GTX 760. (d) Quadro GPU. (e) Cellphone Honor6. (f) Cellphone Note5. (g) Cellphone MI5. (h) Cellphone SONY Z5. (i) Stratix V FPGA.

530 has a higher computing power. Moreover, Sony Xperia Z5 itself has a stumbling block. The potential reasons are the heat dissipation and memory read speed. Although judging from the GFLOPS, the performance of Qualcomm Adreno 530 should be twice of Mali-T760MP8, but Mali-T760MP8 is a little better than Qualcomm Adreno 530 in the average frame rate. One possible reason is that Exynos 7420 in Note5 is better than Qualcomm Snapdragon 820 in MI5. For Stratix V, the parallel code and hardware configuration cannot be fully adapted due to the limitation of chip resource. As a result, part of the work is processed on the CPU and losing some of the parallelism. In addition, because of the resource constraints, many modules of coarse-grained pipeline strategies are executed with low efficiency. Consequently, one of our future work is to further optimize our algorithms and make rational use of computing resources for FPGA platform.

B. Timing Analysis

As mentioned above, nms_n and $match\_radius$ mainly affect the performance and speed of the proposed method. The nms_n increases with the decrease of the computing time of the algorithm, because the number of the detected feature points decreases. Meanwhile, the higher the $match\_radius$ is, the more the computing time will be, because more timing is required for bigger searching areas. For deeply evaluating the performance of the algorithm, we investigate the detailed timing of these four kernels for all the experiments.

In Fig. 8, the ratio of the average computation time of kernels with different nms_n and $match\_radius$ can be seen. Given a fixed $match\_radius$, the time of the $Match$ kernel will grow as the nms_n increases. The first reason is to deal with more particles with lesser nms_n. In addition, with fewer particles, the matching point is less accurate and increases the time of detection. Moreover, increasing the $match\_radius$ will require more matching time if nms_n is fixed.

The second experiment is on the processing time proportion of each module in different platforms. The mobile phone platforms have a higher Filter and Descriptor percentage than others. In four desktop GPUs, to some extent, with the stronger processing capacity, the proportion of Match kernel is lower. Finally, for the Stratix V, NMS takes most of the time because the Match is executed on the CPU. Limited by the resources, the computing ability of FPGA is not fully exploited.

Here, we take the case $nms\_n = 8$ and $match\_radius = 200$. The normalized average total execution time and the execution time single frame for $nms\_n = 8$ and $match\_radius = 200$ are shown in Figs. 9 and 10, respectively. The reason why we choose such a combination is that in this case, the number of feature points can reach the maximum, the qualitative performance is the best, and the computation is the largest in this case.

From Figs. 9 and 10, for most platforms, the proportion of match is greatly reduced, and the proportion of NMS is greatly improved. The reason is that $Match$ module uses the four-stage pipeline and for the execution time of single frame, it only complete 1/4. If the data transfer and intermediate results are ignored, the implementation efficiency of the Match module will be quadrupled.

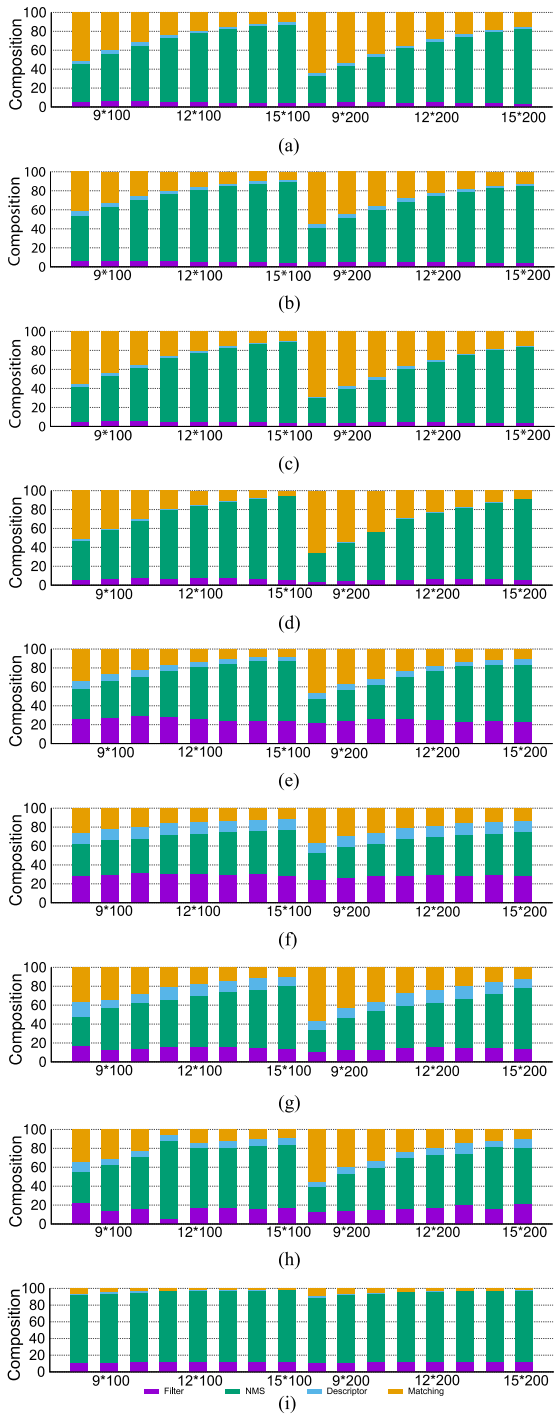


Fig. 8. Ratios of the average computation time of kernels on different platforms. (a) GTX Titan. (b) GTX 1070. (c) GTX 760. (d) Quadro GPU. (e) Cellphone Honor6. (f) Cellphone Note5. (g) Cellphone MI5. (h) Cellphone SONY Z5. (i) Stratix V FPGA.

Fig. 11 shows the distribution of the computation time between host and devices in those nine platforms. In order to make full use of the parallel computing ability of GPU, most of the work will be transferred to the GPU for parallel processing.

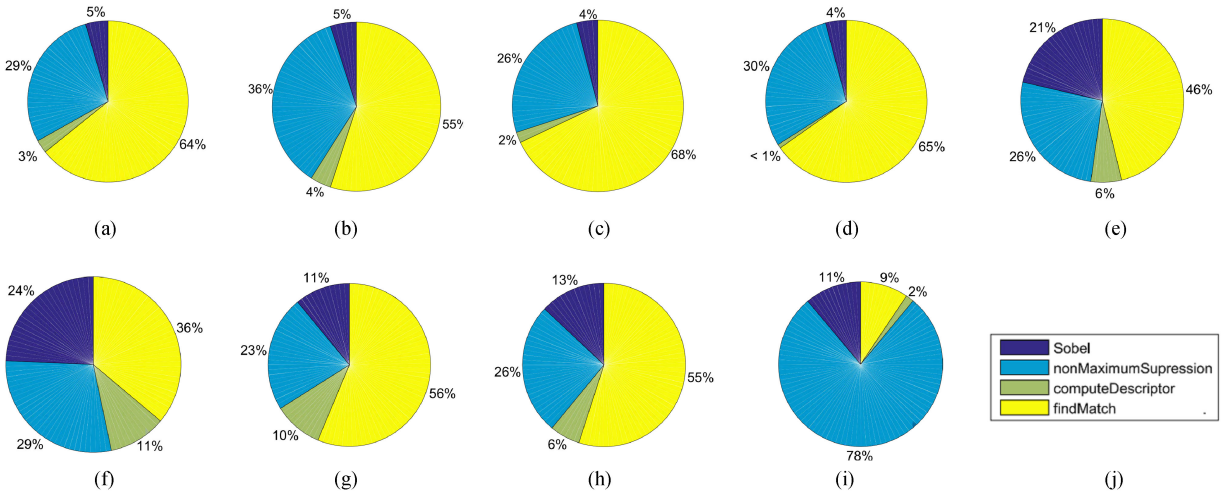


Fig. 9. Normalized average total execution time of kernels for $nms\_n = 8$ and $match\_radius = 200$. (a) Geforce GTX Titan. (b) Geforce GTX 1070. (c) Geforce GTX 760. (d) Quadro GPU. (e) Cellphone Honor6. (f) Cellphone Note5. (g) Cellphone MI5. (h) Cellphone SONY Z5. (i) Stratix V FPGA. (j) The legend.

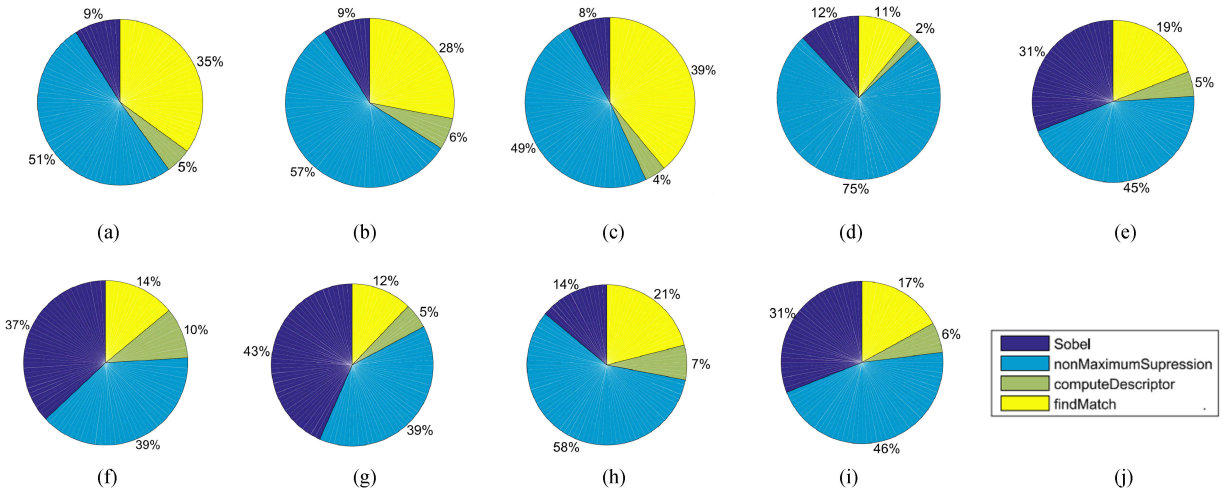


Fig. 10. Normalized execution time for single frame for $nms\_n = 8$ and $match\_radius = 200$. (a) Geforce GTX Titan. (b) Geforce GTX 1070. (c) Geforce GTX 760. (d) Quadro GPU. (e) Cellphone Honor6. (f) Cellphone Note5. (g) Cellphone MI5. (h) Cellphone SONY Z5. (i) Stratix V FPGA. (j) The legend.

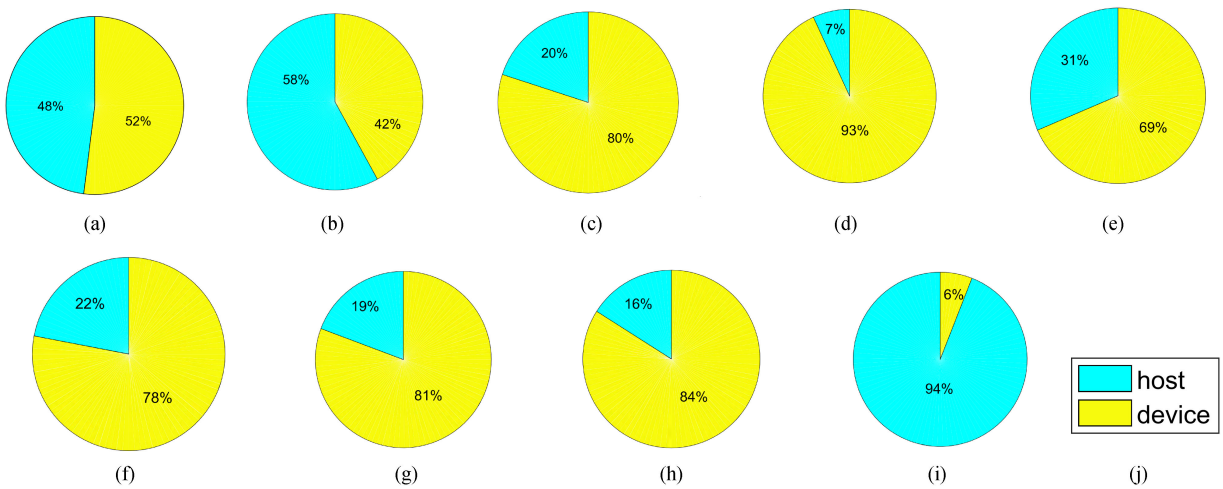


Fig. 11. Distribution of the computation time between host and devices. (a) Geforce GTX Titan. (b) Geforce GTX 1070. (c) Geforce GTX 760. (d) Quadro GPU. (e) Cellphone Honor6. (f) Cellphone Note5. (g) Cellphone MI5. (h) Cellphone SONY Z5. (i) Stratix V FPGA. (j) The legend.

For the CPU, it is only responsible for tasks, such as handling simple results, data transmission, and communication with the GPU.

VI. Conclusion

This paper presents an embedded real-time scene flow algorithm, which can be deployed in different types of COTS. We also provide a method to analyze the program framework and choose the appropriate optimization strategy, such as data parallelization, task parallelization, or their combination. Experimental results reveal that scene flow program can achieve a good performance by using a novel coarse-grained software pipeline, which will be helpful for other embedded stream programs. Our work shows that ADAS applications can be developed using OpenCL language and ADAS can be benefited from COTS, even from the mobile platforms. In the future, we are going to apply our method to other perception applications of autodriving [30] and further improve our algorithm by introducing more finegrained strategies.

References

[1] H. Hirschmuller, "Accurate and efficient stereo processing by semi-global matching and mutual information," in *Proc. IEEE Comput. Soc. Conf. Comput. Vision Pattern Recognit.*, 2005, vol. 2, pp. 807–814.

[2] K. Yamaguchi, D. McAllester, and R. Urtasun, "Efficient joint segmentation, occlusion labeling, stereo and flow estimation," in *Proc. Eur. Conf. Comput. Vision*, Springer, 2014, pp. 756–771.

[3] M. Lin, *et al.*, "Single view stereo matching," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit*, 2018, pp. 155–163.

[4] P. Lenz, J. Ziegler, A. Geiger, and M. Roser, "Sparse scene flow segmentation for moving object detection in urban environments," in *Proc. IEEE Intell. Veh. Symp.*, 2011, pp. 926–932.

[5] A. Geiger, J. Ziegler, and C. Stiller, "Stereoscan: Dense 3d reconstruction in real-time," in *Proc. IEEE Intell. Veh. Symp.*, 2011, pp. 963–968.

[6] G. Kuschk, C. Bailer, D. Stricker, R. Schuster, and O. Wasenmller, "Sceneflowfields: Dense interpolation of sparse scene flow correspondences," in *Proc. IEEE Winter Conf. Appl. Comput. Vision*, 2018, pp. 1056–1065.

[7] A. Wedel, C. Rabe, T. Vaudrey, T. Brox, U. Franke, and D. Cremers, "Efficient dense scene flow from sparse or dense stereo data," in *Proc. Eur. Conf. Comput. Vision*, Springer, 2008, pp. 739–751.

[8] J. Quiroga, T. Brox, F. Devernay, and J. Crowley, "Dense semi-rigid scene flow estimation from RGBD images," in *Proc. Eur. Conf. Comput. Vision*, Springer, 2014, pp. 567–582.

[9] M. Menze and A. Geiger, "Object scene flow for autonomous vehicles," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2015, pp. 3061–3070.

[10] K. Huang, B. Hu, L. Chen, K. Alois, and Z. Wang, "Adas on COTS with openCL: A case study with lane detection," in *Proc. IEEE Int. Conf. Intell. Robots Syst.*, vol. PP, von. 99, 2017, pp. 1–1.

[11] L. Chen, M. Cui, K. Huang, and Z. Xuanyuan, "Real-time scene flow on COTS embedded systems by coarse-grained software pipeline," in *Proc. IEEE Intell. Veh. Symp.*, 2017, pp. 1164–1169.

[12] S. Vedula, S. Baker, P. Rander, R. Collins, and T. Kanade, "Three-dimensional scene flow," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 1999, vol. 2, pp. 722–729.

[13] F. Huguet and F. Devernay, "A variational method for scene flow estimation from stereo sequences," in *Proc. IEEE Int. Conf. Comput. Vision*, 2007, pp. 1–7.

[14] C. Vogel, K. Schindler, and S. Roth, "3D scene flow estimation with a rigid motion prior," in *Proc. Int. Conf. Comput. Vision*, 2011, pp. 1291–1298.

[15] E. Herbst, X. Ren, and D. Fox, "RGB-D flow: Dense 3-d motion estimation using color and depth," in *Proc. IEEE Int. Conf. Robot. Automat.*, , 2013, pp. 2276–2282.

[16] M. Hornacek, A. Fitzgibbon, and C. Rother, "SphereFlow: 6 DoF scene flow from RGB-D pairs," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, , 2014, pp. 3526–3533.

[17] A. Letouzey, B. Petit, and E. Boyer, "Scene flow from depth and color images," in *Proc. Conf. Brit. Mach. Vision*, BMVA Press, 2011, pp. 46–1.

[18] M. Jaimez, M. Souiai, J. Gonzalez-Jimenez, and D. Cremers, "A primal-dual framework for real-time dense RGB-D scene flow," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2015, pp. 98–104.

[19] J. Wulff and M. J. Black, "Modeling blurred video with layers," in *Proc. Eur. Conf. Comput. Vision*, Springer, 2014, pp. 236–252.

[20] K. Yamaguchi, D. McAllester, and R. Urtasun, "Robust monocular epipolar flow estimation," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2013, pp. 1862–1869.

[21] K. Yamaguchi, T. Hazan, D. McAllester, and R. Urtasun, "Continuous Markov random fields for robust stereo estimation," in *Proc. Eur. Conf. Comput. Vision*, Springer, 2012, pp. 45–58.

[22] C. Vogel, K. Schindler, and S. Roth, "Piecewise rigid scene flow," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2013, pp. 1377–1384.

[23] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? The KITTI vision benchmark suite," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2012, pp. 3354–3361.

[24] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "DeepFace: Closing the gap to human-level performance in face verification," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2014, pp. 1701–1708.

[25] L. Chen, L. Fan, J. Chen, D. Cao, and F. Wang, "A full density stereo matching system based on the combination of CNNs and slanted-planes," in *IEEE Trans. Syst. Man. Cybernet. Syst.*, vol. PP, pp. 1–12, 2017.

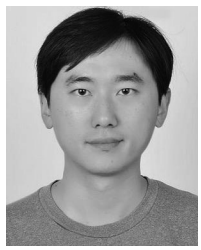
[26] J. Zbontar and Y. LeCun, "Stereo matching by training a convolutional neural network to compare image patches," *J. Mach. Learn. Res.*, vol. 17, pp. 2287–2318, 2016.

[27] D. Muller, M. Meuter, and S.-B. Park, "Motion segmentation using interest points," in *Proc. IEEE Intell. Veh. Symp.*, 2008, pp. 19–24.

[28] A. Neubeck and L. Van Gool, "Efficient non-maximum suppression," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2006, vol. 3, pp. 850–855.

[29] R. O. Duda *et al.*, *Pattern Classification and Scene Analysis*, vol. 3. Hoboken, NJ, USA: Wiley, 1973.

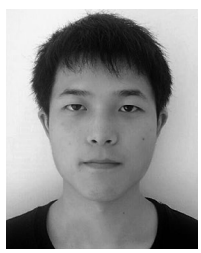
[30] Q. Li, L. Chen, M. Li, S. L. Shaw, and A. Nuchter, "A sensor-fusion drivable-region and lane-detection system for autonomous vehicle navigation in challenging road scenarios," *IEEE Trans. Veh. Technol.*, vol. 63, no. 2, pp. 540–555, Feb. 2014.



Long Chen (M'xx) received the B.Sc. degree in communication engineering and the Ph.D. degree in signal and information processing from Wuhan University, Wuhan, China, in 2007 and 2013, respectively. From October 2010 to November 2012, he was a co-trained Ph.D. Student with the National University of Singapore.

He is currently an Associate Professor with the School of Data and Computer Science, Sun Yat-sen University, Guangzhou, China. His research interests include autonomous driving, robotics, artificial intelligence, where he has contributed more than five publications.

Dr. Chen was the recipient of the IEEE Vehicular Technology Society 2018 Best Land Transportation Paper Award, the IEEE Intelligent Vehicle Symposium 2018 Best Student Paper Award, and the IEEE Intelligent Vehicle Symposium 2018 Best Workshop Paper Award. He is the Guest Editor for the IEEE Transaction on Intelligent Vehicle. He was an Associate Editor for IEEE Technical Committee on Cyber-Physical Systems newsletter.

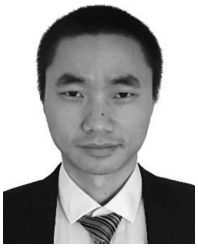


Mingyue Cui received the B.Sc. degree in software engineering from Chongqing Normal University, Chongqing, China, in 2014, and the M.Sc. degree in software engineering from Sun Yat-sen University, Guangzhou, China, 2017. He is currently working toward the Ph.D. degree at Sun Yat-Sen University, majoring in computer science. He is currently a student under the instruction of Kai Huang and Long Chen.

His research interests include the areas of high performance heterogeneous parallel computing and computer vision, where he is recently focused on high performance parallel algorithms of autonomous vehicle.

Feihu Zhang received the B.Sc. degree in automation and the M.Sc. degree in control theory and application from Xian Jiaotong University, Xi'an, China, in 2017 and 2010, respectively, and the Ph.D. degree in informatic from the Technical University of Munich, Munich, Germany, in 2016.

From 2014 to 2016, he was with the Data Fusion group at fortiss GmbH. He is currently an Associate Professor with the School of Marine Science and Technology, Northwestern Polytechnical University, Xi'an, China. His research interests include intelligent vehicle and robotics.



Biao Hu (M'xx) received the B.Sc. degree in control science and engineering from the Harbin Institute of Technology, Harbin, China, in 2010, the M.Sc. degree in control science and engineering from Tsinghua University, Beijing, China, in 2013, and the Ph.D. degree in computer science from the Technical University of Munich, Munich, Germany, in 2017.

He is currently an Associate Professor with the College of Information Science and Technology, Beijing University of Chemical Technology, Beijing, China. His research interests include the autonomous driving, OpenCL computing in heterogeneous system, the scheduling theory in real-time systems, and safety-critical embedded systems.

Dr. Hu is a Handling Editor for Elsevier *Journal of Circuits, Systems, and Computers*.

Kai Huang (M'xx) received the B.Sc. degree from Fudan University, Shanghai, China, in 1999, the M.Sc. degree from the University of Leiden, Leiden, The Netherlands, in 2005, and the Ph.D. degree from ETH Zürich, Zürich, Switzerland, in 2010.

He joined Sun Yat-sen University as a Professor in 2015. He was appointed as the Director of the Institute of Unmanned Systems, School of Data and Computer Science, in 2016. He was a Senior Researcher with the Department of Computer Science, the Technical University of Munich, Munich, Germany, from 2012 to 2015, and a research group leader in fortiss GmbH, Munich, Germany, in 2011. His research interests include techniques for the analysis, design, and optimization of embedded systems, particularly in the automotive and robotic domains.

Dr. Huang was the recipient of the Program of Chinese Global Youth Experts 2014 and was granted the Chinese Government Award for Outstanding Self-Financed Students Abroad 2010, and also the recipient of Best Paper Awards ESTC 2017, ESTIMedia 2013, SAMOS 2009, Best Paper Candidate ROBIO 2017, ESTMedia 2009, and General Chairs. Recognition Award for Interactive Papers in CDC 2009. He has served as a member of the technical committee on Cybernetics for Cyber-Physical Systems of the IEEE SMC Society since 2015.