# Optimization of driver monitoring ADAS algorithm for heterogeneous platform

Ognjen Kocić, Aleksandra Simić, Milan Z. Bjelica and Tomislav Maruna

***Abstract* — Rapid expansion of Advanced Driver Assistance Systems (ADAS) applications has resulted in development of many new algorithms that are applied in solving various challenging problems. These algorithms need to be implemented on existing ADAS platforms which are usually heterogeneous in order to maximize computing power, while minimizing power consumption. The problem becomes how to efficiently decouple the algorithm and map parts of it to heterogeneous hardware, often including CPU, DSP and GPU blocks. This paper gives some insight into efficient ADAS algorithms mappings and optimizations for these platforms. As an illustrative example, driver monitoring algorithm is optimized.**

***Keywords* — ADAS, driver monitoring, optimization, heterogeneous computing.**

## I. Introduction

NOWADAYS we are witnessing that more and more vehicles are equipped with a variety of passive ADAS systems that issue warnings and active systems that actually change car's behavior. In this fast paced environment many algorithms are being created and need to be adequately ported to target embedded systems.

In parallel with development of algorithms, semiconductor companies are shifting their focus from pure CPU systems to advanced heterogeneous systems containing multiple computing units as a part of the same SoC (System on Chip), including but not limited to CPU, GPU and DSP blocks. Rising complexity of both algorithms and target boards has created a problem when these algorithms need to be efficiently deployed. The deployment includes both the decomposition of algorithm, as well as mapping its parts to available computing units. There are many factors to be considered. While one solution may result in better performance, other might be significantly faster to develop.

This paper will present how to efficiently decompose, map and optimize the Driver Monitoring algorithm, which is among the essentials within ADAS environments.

Ognjen Kocić, Faculty of Mathematics, University of Belgrade, Studentski trg 16, 11000 Belgrade, Serbia (e-mail: ognjen@matf.bg.ac.rs).

Aleksandra Simić, Faculty of Mathematics, University of Belgrade, Studentski trg 16, 11000 Belgrade, Serbia (e-mail: ognjen@matf.bg.ac.rs).

Milan Z. Bjelica, Faculty of Technical Sciences, University of Novi Sad, Trg Dositeja Obradovića 6, 21000 Novi Sad, Serbia (e-mail: milan.bjelica@rt-rk.uns.ac.rs).

Tomislav Maruna is with RT-RK Institute for Computer Based Systems, Novi Sad, Serbia (email: tomislav.maruna@rt-rk.com)

## II. Related work

When it comes to driver's fatigue detection various approaches exist in literature [1]. In general fatigue can be detected by observing driving patterns, attaching driver to different sensors, or as in our case - visually tracking the driver using cameras. This approach has many variations both in computer vision methods applied and equipment used (some examples include [2] and [3]).

Various approaches are taken to build ADAS platforms nowadays, with focus being reliability, high performance, low cost and low power consumption. These platforms usually contain a few processing units with different purposes on the same SoC. One standard configuration could include one or more ARM cores, a few DSPs and a specialized GPU. Typical examples are Nvidia Tegra X1 and X2 [4] and TDA2XX of Texas Instruments [5].

Complex ADAS algorithms use different computer vision algorithms as their building blocks. Because of that performance optimizations for listed platforms are an important and well-studied problem [6-7]. Also, optimizations done on isolated computation units, for example DPSs [8], can be later integrated into the system consisting of several processing units. Finally, before any effort is invested in algorithm porting, it might be interesting to get a good estimate of optimal performance and some methods are developed for this purpose [9].

## III. The algorithm

Algorithm that has been optimized has several connected stages, whereas it can be divided to two main parts: (1) face detection and (2) eyes detection.

For a face detection implementation of widely adopted Viola-Jones framework was used. Eyes were detected with custom algorithm that uses weight matrices sums which are then compared against each other. Specifically, a weight matrix is applied on each pixel in eye region part of image and weighted sum for that pixel was calculated. Pixel for which this sum was minimal was used as an eye center.

Eye detection was performed by utilizing concentric weight matrices which are scaled depending on size of the eye region. In the matrix, values are distributed from smaller values to higher values, observing these values from outside towards center of matrix. Two adjacent matrix values, in different rows and columns, differ by factor of $k$ which is an algorithm parameter. An example of such a matrix is given in Fig. 1.

978-1-5090-4086-5/16/$31.00 ©2016 IEEE

0.028 0.028 0.028 0.028 0.028
0.028 0.056 0.056 0.056 0.028
0.028 0.056 0.111 0.056 0.028
0.028 0.056 0.056 0.056 0.028
0.028 0.028 0.028 0.028 0.028

Fig. 1. Normalized 5x5 concentric weight matrix with factor k=2.0.

Finally, algorithms used for the implementation of driver monitoring needed some camera input adjustments. Both face and eye detection were performed on a grayscale image which was cropped and downscaled to a target resolution. Detailed high level diagram of driver monitoring stages is given in Fig. 2.

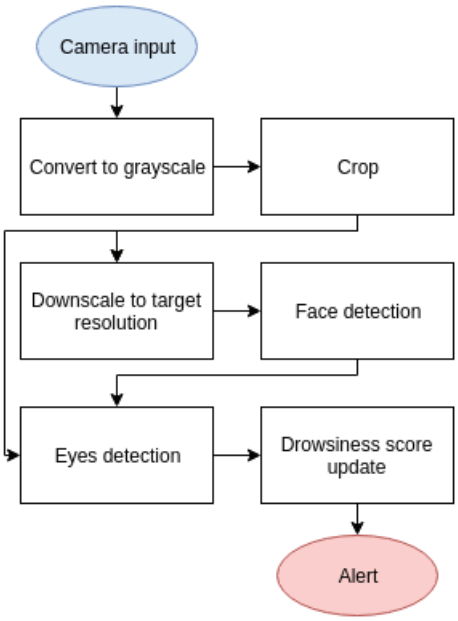


Fig. 2. High level organization of driver monitoring algorithm

## IV. Algorithm Implementation

This chapter gives a brief overview of algorithm implementation aspects.

### A. Short hardware overview

Hardware components that are widely used in ADAS boards can be classified into few categories in terms of how efficiently they process vectored input. This particular categorization is chosen because many ADAS algorithms are built on top of computer vision algorithms which use vectored input. For example, an image can be seen as a vector of pixels.

The most efficient hardware for vectored input is hardware with SIMD architecture. GPUs are highly available example of widely adopted hardware with SIMD architecture. Besides standard GPUs, in automotive there are some vendor specific hardware like Texas Instruments EVE computing unit [5], or EyeQ that is manufactured by MobilEye [10]. SIMD hardware is best utilized if it processes large amount of data in a same fashion. This is due to a fact that these devices have high throughput, but low latency. An example of a simple algorithm, which maps very well to a SIMD hardware, is matrix multiplication [11].

Next, VLIW and MIMD hardware architecture should be considered. Digital signal processors usually fall into this category. Unlike SIMD, components with VLIW and MIMD architectures can efficiently process more than one data unit at a time, while they can be easily programmed in a high level language such as C. DSPs can be very helpful with intermediate level processing algorithms like edge detection [12].

At last, CPUs are considered to be in last category, which consists of the least efficient processing units when considering vectored data. Not only they are not that efficient, they usually have other roles when used on ADAS platforms. When considering ADAS use case, CPUs are exploited as controlling units in a safety critical environment [13].

### B. Our solution for driver monitoring algorithm

Our solution was tested on ADAS board containing Qualcomm's quad core Kryo processor, Adreno 530 GPU and Hexagon DSP device, packed within a single SoC. From performance point of view, it can be observed that this particular board follows a standard pattern in terms of few ARM CPU cores, computer graphics specific hardware and a digital signal processor. ADAS board was equipped with Android OS, which is based on Linux kernel and when used in native mode presents a solid framework for ADAS development (at the time of conduction of this research Linux was not available for the target SoC).

The algorithm is decoupled and implemented in three stages, which are described in Fig. 3.

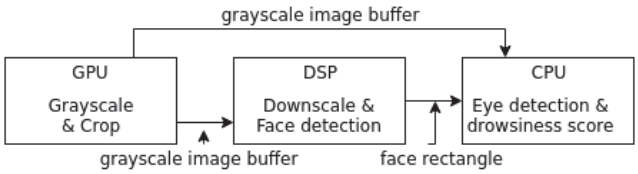


Fig. 3. Decoupling of the algorithm

First stage of the algorithm is image preprocessing. Converting from RGB color space to grayscale was combined with crop into single algorithm that was executed on GPU. Reasoning behind this decision uses the fact that color space transformation is the same for every pixel. Colored pixel $x$ was converted into grayscale pixel $y$ using formula:

$$y = x.r*0.59 + x.g*0.33 + x.b*0.11$$

Crop part was added to avoid unnecessary computations, which would require conversion of unnecessary, cropped parts of the image to grayscale.

Next, resizing image for face detection stage, as well as face detection stage, was executed on a DSP. These kind of intermediate complexity computations are ideal for a DSP. However, cascade classifier, which is a part of Viola-Jones framework, uses data that is commonly read from a file. These values have been hardcoded into static arrays instead. This was done in order to eliminate any file I/O

from DSP.

Finally, eyes detection and drowsiness score calculations have been mapped onto the CPU. Although this may seem to oppose the aforementioned proposition for best utilization of CPUs in ADAS context, however, this decision was a result of global algorithm pipeline optimization. We consider presented mapping to be advantageous because all hardware components were used in implementation. Moreover, eyes detection algorithm that was used is in early stage of development. As such, it was much easier for the algorithm to be prototyped on a CPU.

## V. Optimization of Implementation

In this section both high level and code level optimizations that were done in order to get real time processing of input frames will be described.

### A. Pipeline optimization

Let's observe processing of three adjacent frames ($f_i$, $f_{i+1}$ and $f_{i+2}$ in text that follows). While face detection is running on DSP using properly adjusted frame $f_i$ as input, GPU is free and preparation of next frame, $f_{i+1}$ can be executed on it. These steps can be parallelized, if frame $f_{i+1}$ is successfully delivered from camera by the time face detection started processing frame $f_i$. Furthermore, overlapped processing of all three frames can be expected only under the following assumptions (Fig. 4):

1. Frame $f_{i+2}$ is successfully obtained from camera;
2. Frame $f_{i+1}$ is prepared for face detection algorithm;
3. Frame $f_i$ was processed by face detection algorithm and it is ready for eyes detection algorithm;
4. Assumptions that are listed above all true in time point *t;*
5. None of the phases has significantly longer time of execution when compared to the rest of them.

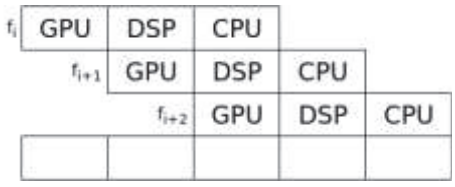


Fig. 4. Overlapping of frame processing

Frames have been time stamped as they were entering and leaving described phases in order to confirm our hypothesis.

### B. Memory optimizations

When algorithms are ported to heterogeneous platforms, one of the main performance issues comes from how these devices share data. It is a very common situation that different computation units do not share address spaces, hence they cannot access the same memory. If data must be copied from address space of one device, into address space of another device, performance benefits from using multiple computation units gets lost too often.

In this particular case, the same few buffers had to be shared between GPU, CPU and DSP (Fig. 3). Since OpenCL 2.0 was used to implement parts of our algorithm on GPU, we could have took advantage of that by exploiting memory sharing between OpenCL device and host that was added to 2.0 standard [14]. But this approach would solve only one part of the problem. Sharing data between DSP and the rest of hardware would still be an issue.

To overcome listed problems, ION allocator was made on Android platform [15]. In general, memory allocated on one of ION heaps can be shared between devices and no extra work is required. This includes both GPU and DSP. To benefit from the fact that our operating system was indeed Android, all buffers, which were exchanged in algorithm, were allocated directly on ION heap. Thus, memory copying was avoided.

### C. DSP optimizations

Since we have overlapped frames processing and optimized memory access, our main concern became optimization of the longest phase which was face detection running on a DSP. Algorithm that we optimized contains quadruple loop as the most expensive component, so we focused our attention on that.

When optimizing this part of code some of the well-known code optimization techniques were applied, including:

- Loop unrolling;
- Inline functions;
- Elimination of branching;
- DSP assembly instructions.

Function inlining, if not abused, can have a good impact on performance as it also had in our case. The key principle is elimination of auxiliary instructions that are used to prepare stack for function call and to do cleanup afterwards. Additionally, jump instructions are also removed in the process.

Last two techniques listed above were in fact very useful when combined together. In general, branches can be harmful for code performance in cases when branch predictor, static or dynamic one, selects wrong branch to be executed. As a consequence potentially long pipeline of instructions will collapse, resulting in the observable execution delay. Hexagon DSP (that was used) has a multiplexing instruction that can be used to choose one of two values depending on a given predicate. Moreover, code that was placed inside branch was not complicated and was mainly containing assignment and simple arithmetic instructions, so branch condition was used as predicate and correct values have been selected using the multiplexing instruction.

Loop unrolling had the smallest impact on performance of the algorithm.

## VI. Evaluation

Evaluation was performed in a lab environment, by monitoring a driver completing a driving simulation task on a demonstration setup (Fig. 5). The focus of the part of the research in this paper and therefore the evaluation was the performance, whereas accuracy was assessed in a separate work so it will be only briefly addressed it in this paper.

The results obtained present an early work and obtained

performances and techniques are not state of the art yet. However, in our opinion, solid results are achieved. The final fully optimized algorithm is able to process 25 frames per second (fps) on average with detected low of 20 fps and high of 28 fps. Main reason for the range between 20 and 28 fps is that eyes were not searched for if face was not already detected.

Fig. 5. Evaluation in the lab environment on a demonstrator

TABLE 1: TIME FOR EACH STAGE OF ALGORITHM.

| Preprocessing | Face detection | Eyes detection |
|---|---|---|
| 34ms | 45ms | 27ms |

Average time for every phase is given in milliseconds in Table 1. When summed up we get 106ms for a single frame processing, which corresponds to 9-10 fps. On the other hand, algorithm pipeline optimization had the biggest impact on performance due to phases overlapping and it was the major factor for greater performances to be reached. Fig. 5, which supports this claim, contains measurements for a single, three, six and twenty frames.

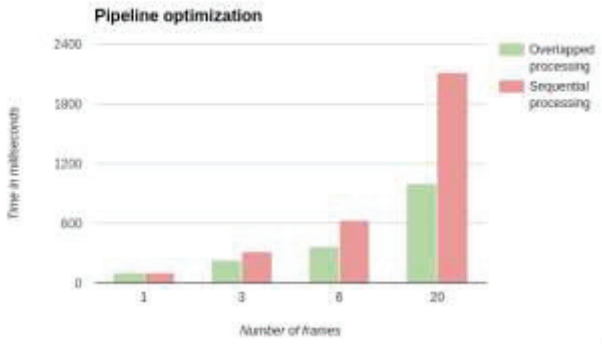


Fig. 5. Effect of pipeline optimization

Accuracy of the algorithm which was observed on a limited set of participants indicate a correlation between driver distraction and high distraction scores that were observed. Therefore, we see the developed algorithm a good starting base for the stated optimization assessment.

## VII. CONCLUSION

In this paper we proposed one approach how to efficiently map driver monitoring algorithm to ADAS board. Furthermore, we have applied a number of optimization techniques to certain parts of the algorithm. In our opinion, this work has shown importance of whole pipeline optimizations on heterogeneous platforms. Future work may include creation of load balancing engine that would be able to optimally choose one of available implementations (e.g. CPU or GPU code) for a single stage of algorithm during runtime. This would be done with goal to maximize both whole algorithm performance and hardware utilization.

## ACKNOWLEDGMENT

This work was partially supported by the Ministry of Education, Science and Technological Development of the Republic of Serbia, under grant number: TR32041.

## REFERENCES

[1] Qiong Wang, Jingyu Yang and Mingwu Ren, “Driver Fatigue Detection: A Survey”, in IEEE International Conference on Intelligent Control and Automation, 21-23. Jun 2006.

[2] Wen-Bing Horng, Chih-Yuan Chen and Yi Chang, “Driver fatigue detection based on eye tracking and dynamic, template matching” in IEEE International Conference on Networking, Sensing and Control, 21-23 March 2004.

[3] Qiang Ji, Zhiwei Zhu and Peilin Lan, “Real-time nonintrusive monitoring and prediction of driver fatigue”, in IEEE Transactions on Vehicular Technology Volume: 53, Issue: 4, July 2004.

[4] Nvidia Tegra X1 whitepaper. Available: http://international.download.nvidia.com/pdf/tegra/Tegra-X1-whitepaper-v1.0.pdf

[5] J. Sankaran and N. Zoran, “TDA2X, a SoC optimized for advanced driver assistance systems,” in Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE Int. Conf. on. IEEE, 2014, pp. 2204–2208.

[6] Pramod Poudel, Mukul Shirvaikar, “Optimization of computer vision algorithms for real time platforms”, in 42nd Southeastern Symposium on System Theory, 7-9 Mar. 2010.

[7] Marcos Nieto, Gorka Vélez, Oihana Otaegui, Seán Gaines and Geoffroy Van Cutsem, “Optimising computer vision based ADAS: vehicle detection case study”, in IET Intelligent Transport Systems, Volume: 10, Issue 3, 28. Mar. 2016.

[8] Turturici, M., Saponara, S., Fanucci, L., Franchi, E.: “Low-power DSP system for real-time correction of fish-eye cameras in automotive driver assistance applications”, Journal of Real-Time Image Processing, 2013, 9, pp. 463-478.

[9] Romain Saussard, Boubker Bouzid, Marius Vasiliu, Roger Reynaud, “Optimal Performance Prediction of ADAS Algorithms on Embedded Parallel Architectures”, in IEEE International Conference on High Performance Computing and Communications, 24-26 Aug. 2015.

[10] G. P. Stein, E. Rushinek, G. Hayun, and A. Shashua, “A computer vision system on a chip: a case study from the automotive domain,” in Computer Vision and Pattern Recognition-Workshops, 2005. CVPR Workshops. IEEE Comp. Soc. Conf. on. IEEE, 2005, pp. 130–130.

[11] José María Cecilia, José Manuel García, Manuel Ujaldón, “Parallel Computing: From Multicores and GPU’s to Petascale”, Amsterdam: IOS Press, 2010, pp 331 - 340.

[12] Zuwena Musoromy, Faycal Bensaali, Soodamani Ramalingam, and Georgios Pissanidis, “Comparison of real-time DSP-based edge detection techniques for license plate detection”, in IEEE International Conference on Information Assurance and Security (IAS), 23-25 Aug. 2010.

[13] Andrew Hopkins, “The Functional Safety Imperative in Automotive Design”. Available: http://www.arm.com/files/pdf/The_Functional_Safety_Imperative_in_Automotive_Design.pdf

[14] OpenCL specification. Available: https://www.khronos.org/registry/cl/specs/opencl-2.0.pdf

[15] John Stultz, “Integrating the ION memory allocator”. Available: https://lwn.net/Articles/565469/