Estimation of Time Behavior of Selected Autonomous Driving Algorithms using GPGPU-Sim

Christian Widerspick, Christian Hartmann and Dietmar Fey
Friedrich-Alexander University Erlangen-Nürnberg (FAU), Germany
E-Mail: {christian.widerspick,christian.hartmann,dietmar.fey}@fau.de

***Abstract*—In order to achieve the goal of autonomous driving, more and more computing power has to be installed in vehicles. In addition to the already existing architectures, GPU manufacturers are trying to ensure that their products are used in vehicles. In the area of estimating non-intrinsic properties, such as execution time, there is still room for improvement, especially affecting GPUs. This work is to examining whether it is sufficient in this domain, that the execution time determination is conducted by an functional emulation and simple instruction counting. For this reason gpgpu-sim is extended by this simple method.**

I. Introduction

Automated or future autonomous driving is one big direction in the current car development [1]. At present the driver needs to have a deep understanding of his environment. Basing on own experience and the monitoring of direction and speed of other traffic participants, the driver has the possibility to make proper decisions, in order to avoid accidents and reach the destination. The human driver gets this information mainly by visual observation [2]. This complex task has to be taken by the car itself, when driving autonomous. It is necessary that a car knows as much as possible about its environment. An exclusive use of visual information does not seem to be sufficient. The malfunction of the camera would expose the driver to great risks, because the driver needs seconds to understand the current situation he or she is faced with and to take control over his car again.

Every type of sensor has its own advantages and disadvantages and gets a different kind and quality of data in different circumstances. Already nowadays there is a large variety of different sensors in automobiles, supporting the driver to deal with that challenge. The combination of different sensors result in redundancy and thus in a greater failure safety. Today nearly all automotive manufacturers use at least a combination of camera and radar or lidar. There are some reasons for the use of GPUs in autonomous cars. When speaking about image processing, it is obvious that using a GPU is a choice. Most of time processing radar data is spent by calculating FFTs and Nvidia is advertising the speed of their implementation on GPUs. The increasing number of sensors is leading to more and more data, which have to be analyzed. GPGPU computing can help to accelerate that interpretation.

The usage of GPUs increases the heterogeneity of the overall system. Currently it is worked on a simulation framework for the Nvidia Drive PX2 platform, which should help to master heterogeneous systems. One part is the improvement of GPGPU-sim for applications in the automotive domain.

In a first step, algorithms, which are representative for the autonomous driving are selected. These have to be suitable for implementation on GPUs. Furthermore, it is necessary to look at the conditions in which a speedup can be achieved [3]. In a last step it is compared how exactly the time projected by emulation matches with the time measured on real hardware.

II. Related Work

Various work has been concerned with the emulation of GPUs with a stuttering result[4]. There are a number of projects that have dealt with functional simulation and estimation of non-intrinsic properties. The previous approaches are usually difficult to use, more difficult to adapt to new circumstances and poorly documented. Even the change within an architecture can render the results unusable [4]. As a rule, GPGPU-Sim [5] is used as a reference. Depending on the problem domain, it provides very accurate results for specific GPUs. However there are models for different GPUs with Tesla and Fermi architecture available.

There is a series of other simulators that work according to Eeckhout approach on a mechanistic model [6]. Another representative is the PMAC Framework [7]. It searches in source code for previously known access patterns. These are compared with previously collected microbenchmarks. The observable accuracy is low and the framework only works for memory-bound kernels. In the Eiger framework [8] the required parameters are collected in an automated, experimental test run and the obtained data is evaluated and combined into one model. It requires a deep understanding of statistical methods. Overall, the framework has a promising approach, but its use is difficult because of poor documentation [4]. A similar approach was chosen in the STARGAZER [9] framework. However, some characteristics of modern architecture are missing, therefore in some cases the prediction is bad. In addition, it depends on the accuracy of GPGPU-Sim which is used to determine some parameters.

III. Selection of Algorithms

In modern vehicles, a series of driver assistance systems has already been established. These give a certain insight into what sensors are necessary to let vehicles drive autonomously. Adaptive cruise control, for example, uses radar and camera data to keep the distance to the preceding vehicle [10]. For this reason, two algorithms were selected from which one is processing the image and the other one the radar data.

A. *Optical Flow*

The optical flow is used to detect movements within an image sequence. Therefore the moving-distance of each pixel between two pictures is calculated. It provides a two-dimensional velocity vector for each image point under consideration. This vector can also be understood as a translation vector, which moves one image pixel into another. The Movement of all Pixels can be described as a vector field. There are different kinds of algorithms under this name. Considered is the algorithm according to Horn and Schunk [11]. It is assumed that the brightness of each pixel remains constant after the shift. The problem is considered to be a minimization problem. The goal is the determination of the vector field and thus the minimization of the global brightness difference. This is a continuity equation which is sometimes found in physics. The resulting equation system can be solved with an iterative Jacobi method. An example is shown in Figure 1.

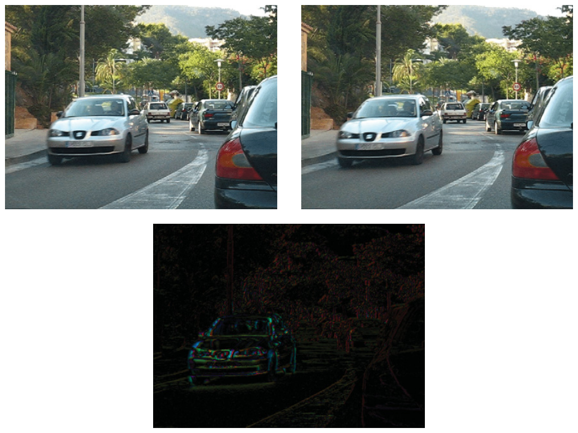

Fig. 1. Result of the Horn and Schunk algorithm. The calculated vectors are assigned to a color in the hsv color space

B. *Fast Fourier Transformation*

With one radar antenna the relative speed and distance of an object to the automobile can be determined, but not its position [12]. Basically, a stereo camera can also be used to determine the distance. The use of radar is much more robust against environmental influences, such as bad weather than a camera. With the use of two independent systems, the data quality can be increased. If, for example, the acquired data of the camera are unusable, this does not necessarily have to apply to the radar as well. Thus, to a certain degree, a fall-back level can be achieved. In order to calculate distance and speed, a Fourier transformation has to be calculated twice. Nvidia states that the speed of their implementation is up to ten times higher than on other architectures. To test this promise, measurements were carried out.

IV. Circumstances for the Use of a GPU

That the use of a GPU as an accelerator is worthwhile, certain framework conditions have to be fulfilled. Although massive problems can be resolved in parallel, there is also a large overhead. The data connected to the problem have to be decomposable into a lot of small parts and the individual workflow should execute the same control flow branch within a warp as often as possible. Before the calculation starts, the data has to be copied first to the GPU and in the end back again. The time for the copying process is not be underestimated and the performed calculations on GPU have to be big enough in relation to this.

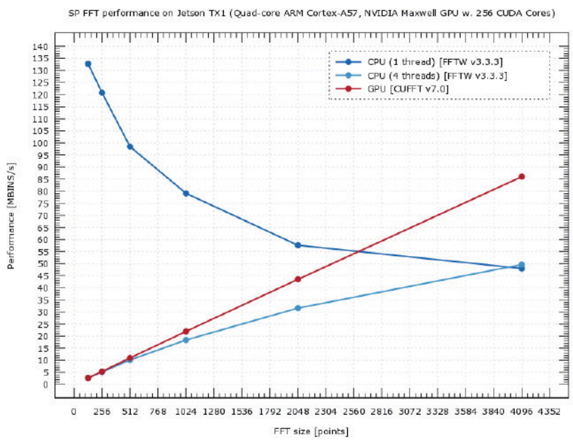

Fig. 2. Performance comparison of the FFT on different architectures

In Figure 2 it is shown how often the FFT of a certain size can be calculated on different architectures. The radar is transmitting electromagnetic waves. The reflected wave of each pulse is summarized in a so-called BIN. It can be seen that the performance on CPUs for small sizes is superior to GPUs, but the relationship changes from a certain size. The same applies to the implementation with several threads. The overhead for the fork-and-join process exceeds the profit by using multiple cores, analogous to the copying process of the GPU.

The advantage for the GPU is only found in the case of magnitudes, which are one to two orders above the present used. This leads to more precise results, but is in the range of noise. Another possible approach is the use of several antennas. N antennas produce N times as many data as one. By using more than one antenna, it is possible to determine, in addition to the distance and the speed, the angle of an object in relation to the direction of travel. In this way, the view of the surroundings can be refined. The accuracy of the angle depends on the number of antennas. A known algorithm for determining this angle is the MUSIC algorithm [13]. In this case, a computer-intensive eigenvalue decomposition is carried out, which can be well parallelized.

V. Comparison Between Real Hardware and Simulation

Within the scope of the research group FORMUS$^{\mathrm{i}}$IC, a framework is to be developed that maps a heterogeneous system in one application. The objective is an instructional simulator that takes non-intrinsic characteristics into account. The Nvidia Drive PX2 board serves as a template for the special needs of the automotive industry. It contains two Parker SoCs, consisted of four ARM Cortex A-57 and two Denver-2-cores, two discrete GPUs based on Pascal architecture and an Aurix TC 297 microcontroller [14]. Both presented algorithms were implemented accordingly and run on the discrete GPU with an ARM host. The fusion of the sensor data is still being worked on and is to be carried out on the Aurix.

To the predetermined structure a simulator is created. It combines Synopsys Platform Architect, which provides the model for the ARM processor and the Aurix Microcontroller. This is connected to the widely used GPGPU-Sim as functional GPU emulator, which was slightly adapted to run the programm. In this way, a uniform, heterogeneous simulator, which is intended to emulate the board, is established. By selecting the algorithms, each component is claimed. This allows a comprehensive comparison of simulated and real world in the end. To stay in reasonable bounds, only the GPU part is discussed in this work.

Between the Cuda language and the assembler code level there is an intermediate language called PTX ISA. The emulation of the application is based on this. The advantage of this intermediate language lies in the stability of several architectural generations. In order to determine the execution time of the kernel, a model, which represents the execution of the instructions, is required. This is a simple model which only counts the instructions. Each instruction is provided with a weight corresponding to the execution time in cycles. The total execution time thus results in:

$$T = \sum_{i=1}^{n} C_i W_i$$

$$\begin{aligned} \text{with T} &= \text{execution time} \\ \text{n} &= \text{number of different instructions} \\ \text{C} &= \text{number of calls of each instruction} \\ \text{W} &= \text{weight of instruction} \end{aligned}$$

This process obviously has a major weakness. Namely the independence of internal conditions always assumes that the instructions need always the same execution time and all data is available. But this method can be carried out very quickly since very few things have to be known and are regarded. It also provides the lowest speed loss compared to other methods. It is important to evaluate for which conditions this assumption is sufficient.

VI. Results

The emulation of the two selected algorithms provide a very different result in view of the accuracy of the prediction. At present the approach models the memory accesses very poorly. It is assumed that each access takes the same number of cycles and only one access can take place at the same time. In fact, several accesses can take place in parallel. For this reason, only kernels that have few accesses are considered.

A. Emulation of the Fast Fourier Transformation

In order to preserve the clarity, 5 sample kernels were randomly selected from the libcufft API-Call cufftExecC2C() (Table I). The elapsed cycles were measured using the clock() function.

TABLE I
Comparison of execution time between emulation and reality

name of kernel	cycles real	cycles emulated	% diff.
spRadix0004A_kernel1Tex	3358	3710	9.5
spRadix0003A_kernel1Tex	3440	3386	-1.6
spRadix0002A_kernel1Tex	2895	2750	-5.3
spRadix0002A_kernel3Tex	2226	2192	-1.6
spRadix0009A_kernel3Tex	6618	6434	-2.9

The emulation provides a useful result in this case. In most cases the predicted time is a little smaller than in reality. The maximum absolute error is less than 10%. In order to deepen the consideration of the execution time, it is possible to carry out a finer-granular measurement. Figure 3 shows a measurement from the first to the nth instruction of one of the selected sample kernels. For the emulated case, a monotonically increasing course, which would also be expected for the real course, is obtained. The insertion of a measuring point after each instruction for the real measurement would strongly influence the program sequence. Thus, only the possibility to measure number-of-instructions times remains. First, the duration of the first instruction is measured, second the duration of the first and the second instruction, then the instructions 1 to 3 and so on. Since the measurements are independent to each other, the resultant graph may fall, although you have a monotonously increasing course in reality. This is because the duration of the individual instructions has a certain range of variation.

It can be seen that the curves in the front and back part are very similar. In the middle there are greater deviations, which can be attributed to the fact that pipelining in this model is considered only implicit. The gap in the course of the curve results from the fact that not all paths are traversed.

B. Emulation of Optical Flow

The results obtained by the emulation of the optical flow are significantly worse. Here, the expectations are clearly below the measured real time (Table II).

Underestimated execution times are mainly in the range of 50 - 60 % and make this model unfeasible for this usecase. In this case the internal states of the GPU have a too great

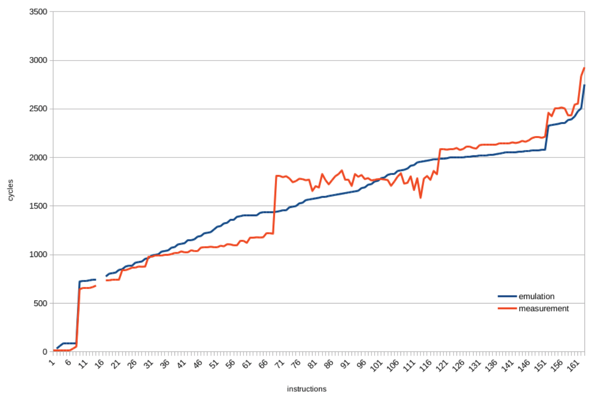

Fig. 3. Measurement of execution time from first to nth instruction of kernel spRadix0002A_kernel1Tex

TABLE II
COMPARISON OF EXECUTION TIME BETWEEN EMULATION AND REALITY

name of kernel	cycles real	cycles emulated	% diff.
norm	9833.6	4065	-59.5
bicubic_warp	10905.2	5866	-48.2
constants	5689,9	1820	-69.5
sorNormal	10584,5	10284	-6.8
AngularMagnitudeToHSV	11029.8	4687	-59

influence on the execution time. Obviously a much more extensive model presentation is necessary. The immediate feasibility of the instructions, which applied in the first case, seem to be no longer correct in this case. Much more seem the GPU idling. This hast to be regarded with the help of a more complex model.

VII. Conclusion and Future Work

In this paper, a brief insight into the algorithms that are relevant to the domains of autonomous driving has been given. In addition, it was compared how far the emulation of the execution time with the help of simple instruction counting fits with reality. It has been shown that for a FFT this simple procedure is sufficient, while for complex image processing algorithms a finer model is required. Two key points could be identified: Firstly it has to be identified which circumstances lead to the fact that this simple instruction counting is once providing acceptable results and once not. Due to the fact that in both cases memory accesses were minimized. The question remains open whether the consideration of each instruction leads to the wished results. Under circumstances, too little can be found about the internal structure, so that measuring coarser blocks produces better results. Alternatively, statistical approaches could also be followed [15]. Secondly, a model has to be found that predicts the duration of memory accesses. This has to take account of coalescence, cache behavior and memory bandwidth.

Acknowledgment

This work is supported by the Bavarian Research Foundation (BFS) as part of their research project ”FORMUS^{3}IC”.

References

[1] T. Luettel, M. Himmelsbach, and H. J. Wuensche, “Autonomous ground vehicles 2014;concepts and a path to the future,” *Proceedings of the IEEE*, vol. 100, no. Special Centennial Issue, pp. 1831–1839, May 2012, ISSN: 0018-9219. DOI: 10.1109/JPROC.2012.2189803.

[2] G. Underwood, “Visual attention and the transition from novice to advanced driver,” *Ergonomics*, vol. 50, no. 8, pp. 1235–1249, 2007, PMID: 17558667. DOI: 10.1080/00140130701318707. eprint: http://dx.doi.org/10.1080/00140130701318707. [Online]. Available: http://dx.doi.org/10.1080/00140130701318707.

[3] Nvidia Corperation. (2016). Cufft, [Online]. Available: https://developer.nvidia.com/cufft (visited on 02/10/2017).

[4] S. Madougou, A. L. Varbanescu, C. de Laat, and R. van Nieuwpoort, “An empirical evaluation of GPGPU performance models,” in *Euro-Par 2014: Parallel Processing Workshops - Euro-Par 2014 International Workshops, Porto, Portugal, August 25-26, 2014, Revised Selected Papers, Part I*, 2014, pp. 165–176. DOI: 10.1007/978-3-319-14325-5_15. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-14325-5_15.

[5] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong, and T. M. Aamodt, “Analyzing cuda workloads using a detailed gpu simulator,” in *Performance Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on*, Apr. 2009, pp. 163–174. DOI: 10.1109/ISPASS.2009.4919648.

[6] L. Eeckhout, *Computer Architecture Performance Evaluation Methods*, ser. Synthesis lectures in computer architecture. Morgan & Claypool Publishers, 2010, ISBN: 9781608454679. [Online]. Available: https://books.google.de/books?id=HtYuOaaNcL8C.

[7] A. Snavely, L. Carrington, N. Wolter, J. Labarta, R. Badia, and A. Purkayastha, “A framework for performance modeling and prediction,” in *Proceedings of the 2002 ACM/IEEE Conference on Supercomputing*, ser. SC ’02, Baltimore, Maryland: IEEE Computer Society Press, 2002, pp. 1–17. [Online]. Available: http://dl.acm.org/citation.cfm?id=762761.762785.

[8] A. Kerr, E. Anger, G. Hendry, and S. Yalamanchili, “Eiger: a framework for the automated synthesis of statistical performance models,” in *High Performance Computing (HiPC), 2012 19th International Conference on*, Dec. 2012, pp. 1–6. DOI: 10.1109/HiPC.2012.6507525.

[9] W. Jia, K. A. Shaw, and M. Martonosi, “Stargazer: automated regression-based gpu design space exploration,” in *Performance Analysis of Systems and Software (ISPASS), 2012 IEEE International Symposium on*, Apr. 2012, pp. 2–13. DOI: 10.1109/ISPASS.2012.6189201.

[10] K. P. Sreevishakh and S. P. Dhanure, “A review paper on automotive crash prediction and notiication technologies,” in *2015 International Conference on Computing*

Communication Control and Automation, Feb. 2015, pp. 999–1002. DOI: 10.1109/ICCUBEA.2015.197.

[11] B. K. Horn and B. G. Schunck, “Determining optical flow,” Cambridge, MA, USA, Tech. Rep., 1980.

[12] C. Cook, *Radar Signals: An Introduction to Theory and Application*, ser. Electrical science series. Elsevier Science, 2012, ISBN: 9780323146302. [Online]. Available: https://books.google.de/books?id=hB3CQQ7jBckC.

[13] R. Schmidt, “Multiple emitter location and signal parameter estimation,” *IEEE Transactions on Antennas and Propagation*, vol. 34, no. 3, pp. 276–280, Mar. 1986, ISSN: 0018-926X. DOI: 10.1109/TAP.1986.1143830.

[14] Nvidia Coperation, “Nvidia drive px 2 automotive product - data sheet,” Tech. Rep., 2016.

[15] K. Berezovskyi, L. Santinelli, K. Bletsas, and E. Tovar, “Wcet measurement-based and extreme value theory characterisation of cuda kernels,” in *Proceedings of the 22Nd International Conference on Real-Time Networks and Systems*, ser. RTNS ’14, Versaille, France: ACM, 2014, 279:279–279:288, ISBN: 978-1-4503-2727-5. DOI: 10.1145/2659787.2659827. [Online]. Available: http://doi.acm.org/10.1145/2659787.2659827.