# Work-in-Progress: DAPHNE - An Automotive Benchmark Suite for Parallel Programming Models on Embedded Heterogeneous Platforms

Lukas Sommer Florian Stock Leonardo Solis-Vasquez Andreas Koch
{sommer,stock,solis,koch}@esa.tu-darmstadt.de
Embedded Systems and Applications Group
TU Darmstadt, Germany

## ABSTRACT

Due to the ever-increasing computational demand of automotive applications, and in particular autonomous driving capabilities, the automotive industry and its suppliers are starting to adopt parallel and heterogeneous embedded computing platforms.

However, C and C++, the currently dominating programming languages in this industry, do not provide sufficient mechanisms to fully exploit such platforms. As a result, vendors have begun to employ true parallel programming models such as OpenMP, CUDA or OpenCL.

In this work, we report on a benchmark suite developed specifically to investigate the applicability of established parallel programming models to automotive workloads on heterogeneous platforms.

## 1 INTRODUCTION

The computational demands of automotive applications has increased steeply in recent years, in particular due to autonomous driving and advanced driver-assistance (ADAS) functionalities.

To meet this new computational demand, the automotive industry is starting to turn towards parallel and heterogeneous platforms. The multi-core processors and accelerators (e.g., GPUs) found on heterogeneous platforms typically require programming language support to explicitly express parallelism. However, C and C++, the currently dominating programming languages in the automotive field, lack sufficient mechanisms. As a consequence, the automotive industry is keenly interested in parallel programming models.

While a number of well-established standards for parallel and heterogeneous programming exist in the HPC community, the embedded target platforms used in automotive applications differ significantly from the HPC systems these programming models were originally tailored for.

In this work, we present the DAPHNE (Darmstadt Automotive Parallel HeterogeNEous) open-source benchmark suite [1], comprising multiple kernels from automotive applications, together with parallel implementations in different programming models and for different embedded computing platforms. This suite allows to study the applicability of parallel programming models to different automotive workloads and their performance on embedded, heterogeneous systems. The insights can also be used to establish a set of best practices, potential adaptions and possible extensions of the investigated parallel programming models.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
*EMSOFT'19 Companion, October 13–18, 2019, New York, NY, USA*
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6924-4/19/10…$15.00
https://doi.org/10.1145/3349568.3351547

| Kernel | LoC | Input [MB] | Output [MB] | Data Sets |
|---|---|---|---|---|
| points2image | 347 | 19 000.000 | 4 300.000 | 2500 |
| euclidean_clust. | 956 | 45.000 | 54.000 | 350 |
| ndt_mapping | 1394 | 4 200.000 | 0.008 | 115 |

**Table 1: Kernel statistics**

## 2 BENCHMARK SUITE

Our goal for the development of the benchmark suite was to extract actual compute-intensive automotive workloads into easy-to-analyze standalone kernels for parallelization.

For the serial implementations of the automotive workloads that serve as the basis for our benchmarks suite, we reviewed multiple open-source frameworks for ADAS. In this process, we found Autoware [3] to be the most promising candidate as the source of the serial implementations.

Autoware contains algorithms for all steps of an AD/ADAS application, including sensing, perception, planning, decision making, and actuation. As such, the modules contained in Autoware are representative for the kind of computations found in real-world automotive applications.

Using the test dataset acquired from a real test drive and provided by Autoware, we used profiling to identify execution hotspots. As our aim was to accelerate code with a parallel execution model, only those code parts that seemed promising for parallelization were considered for extraction.

Currently, our benchmark suite contains three different kernels. Each kernel was provided in a skeleton for standalone execution outside of the complex ROS-based Autoware framework. References to third-party libraries were either inlined or replaced with custom implementations. To make sure that we did not introduce any artificially slow sections in this process, we compared the performance of our implementation to that of the original Autoware implementation (for an example, see Fig. 1).

Additionally, we extracted five data sets of increasing size with input- and reference- data for each kernel, which we provide together with the benchmark suite, see Table 1 for statistics.

After extraction, we parallelized each kernel using different programming models. We chose OpenMP, CUDA and OpenCL for parallelization, because these standards are supported on most embedded, heterogeneous platforms. For each kernel, we provide an

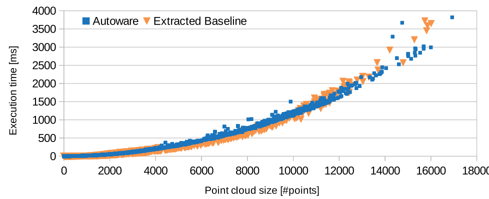


**Figure 1: Comparison of different radiusSearch implementations.**

implementation in each of the programming models. In the case of OpenMP we also provide implementations using the new device offloading features. For the points2image-kernel, we additionally created an implementation for Xilinx Zynq Ultrascale+ MPSoC using OpenCL with Xilinx SDAccel.

## 2.1 points2image Kernel

The points2image-routine gets as input a point cloud (which originates from a LIDAR) with intensity and range information. The points are then projected onto a given 2D view.

With thousands of points being projected at the same time, this seemed like a very good candidate for parallelization. However, in practice, multiple of the 3D LIDAR points may end up being projected to the same point in the 2D view. This leads to race conditions in parallel execution, and can result in incorrect results when not handled with care.

The original Autoware code prioritizes the point that is *closer* to the projected view. This behavior is implemented in our parallel versions by using atomic min functions in the threads accessing the 2D view. In case of accessing the same 2D element, the atomic min function guarantees that the correct thread writes its value.

Similar atomic and synchronization functions can be used to keep the array with the corresponding intensity up to date. While these atomic operations diminish the resulting speed up, the accelerators were still able to improve the run time by up to a factor of 3.5x.

## 2.2 euclidean_clustering Kernel

Similar to points2image, the euclidean_clustering-kernel also operates on a point cloud. The algorithm clusters points that are close together, to identify objects.

The original implementation uses a *k*d-tree to compute the distances between points. This was provided by the Point Cloud Library (PCL, [4]). To remove the dependency on the PCL and enable stand-alone, library-less compilation and execution, a custom implementation was created. As the distances from *all* points to *all* others were required, our replacement for the radiusSearch function from PCL employs a look-up of the distances in a precomputed table. The table does not need to hold the distances between two points, but just a boolean indicating if the distance between those points is below the given threshold. This reduces the memory required for the table significantly.

To verify that this standalone kernel has a similar performance to the original one, it was benchmarked against the original Autoware version. Fig. 1 shows that the performance of the two implementations is almost identical.

## 2.3 ndt_mapping Kernel

The ndt_mapping kernel is a SLAM (simultaneous location and mapping) algorithm that works with *Normal Distribution Transformations* (NDT, [2]). As input, two point clouds are accepted, one representing the current LIDAR scan, and a second one representing the map built during the drive so far. The newly discovered points from the LIDAR scan are then added to the map by *aligning* them with the existing map.

The computational result is a transformation consisting of a rotation and a translation. The transformation is the best transformation that fits the newly acquired LIDAR point cloud to a pre-existing map point cloud using NDT.

To make the kernel code library free, we replaced the complex SVD algorithm, with a much simpler Gaussian solver. In general the SVD solver deals better with singular matrices, but for our purpose, the precision of the replacement turned out to be sufficient.

Beyond that, we again replaced a radius search within a *k*d-tree. In this case, the search was not used to compute the distances from all points to each other within the *k*d-tree. Instead it was used to actually find the *closest* point inside the tree to a point outside the tree. This search was implemented as a linear search within the point cloud. Our evaluation showed that neither of these changes slows down the sequential baseline. On the contrary, they actually *improve* the runtime of the sequential baseline, and thus do not artificially inflate the speedups we achieve through parallelization.

# 3 CONCLUSION AND OUTLOOK

We presented the DAPHNE open-source [1] benchmark suite, comprising three different workloads typical for automotive applications. For each of the kernels we provided parallelizations with CUDA, OpenCL and OpenMP, all qualified for execution on actual embedded computing platforms.

The benchmark suite can be used to assess the applicability of established parallel programming models to automotive workloads, or to evaluate new compute platforms for the AD/ADAS domains.

In the future, we plan to extend the benchmark suite by adding more kernels and parallelization with other programming models, such as SYCL. Besides that, we will use the benchmark suite to investigate the performance of established parallel programming models on embedded heterogeneous platforms and how these models can be adapted to better meet the needs of the automotive industry.

# ACKNOWLEDGMENTS

This project was funded by VDA FAT as part of the research of AK31. The authors would also like to thank Xilinx for supporting this work by donations of hard- and software, as well as Renesas and Codeplay for their help with the acquisition of the V3M and the timely support for using their programming tools.

# REFERENCES

[1] 2019. DAPHNE Benchmark Suite. https://github.com/esa-tu-darmstadt/daphne-benchmark. Accessed: 2019-08-01.
[2] P. Biber and W. Strasser. 2003. The normal distributions transform: a new approach to laser scan matching. In *Intl. Conf. on Intelligent Robots and Systems (IROS 2003)*.
[3] S. Kato, E. Takeuchi, Y. Ishiguro, Y. Ninomiya, K. Takeda, and T. Hamada. 2015. An Open Approach to Autonomous Vehicles. *IEEE Micro* 35, 6 (Nov 2015), 60–68.
[4] Radu Bogdan Rusu and Steve Cousins. 2011. 3D is here: Point Cloud Library (PCL). In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. Shanghai, China.