Accelerating Automotive Analytics: The M2DC Appliance Approach

Giovanni Agosta[1], Carlo Brandolese[1], William Fornaciari[1], Nicholas Mainardi[1], Gerardo Pelosi[1], Federico Reghenzani[1], Michele Zanella[1], Gaetan Des Courchamps[2], Vincent Ducrot[2], Kevin Juilly[2], Sébastien Monot[2], and Luca Ceva[3]

[1] DEIB – Politecnico di Milano {name.surname}@polimi.it
[2] AS+ Groupe Eolen {name.surname}@eolen.com
[3] Vodafone Automotive Telematics
luca.ceva@vodafone.com

Abstract. The Modular Microserver DataCenter (M2DC) project provides low-energy, configurable, heterogeneous servers for applications that focus on the elaboration of large data sets, but can take advantage of performance enhancement provided by transparent acceleration techniques. In this paper, we exemplify the M2DC approach through one of the project's use cases, namely automotive Internet of Things analytics. We present the main goals of the use case and we show how an appropriate M2DC microserver can be used to accelerate the application without significant modifications to its code.

Keywords: Data Analytics · Embedded Systems · Compilers · Applied Cryptography · Automotive

1 Introduction

Analytics applications extracting valuable knowledge from the large amount of data collected by Internet of Things connected devices are already becoming a killer application for data centres. In particular, the automotive application domain is expected to expand vastly in the next few years. The 20% of the circulating cars will be connected vehicles by 2020 and this number will double by 2024. This makes the market of automotive telematics viable, paving the way for a large number of different applications. Consequently, the market space for fleet management and added value services will skyrocket, with a compound annual growth rate of 65% [4].

To cope with the requirements of value-added applications leveraging automotive telematics data, data-centres need to update their technologies. The Modular Microserver for Data Centers (M2DC) project aims at prototyping a highly efficient and customizable cost-optimized server architecture [8,21]. These servers are composed of microserver computing resources, which can be tailored to a specific application domain. In the M2DC approach, an application developer can select from a range of turnkey appliances that are easily configured, deployed, and maintained. The appliance is composed

[4] http://analysismason.com

of a collection of hardware resources and of a software stack needed to exploit them in a transparent way.

In this work, we present an example of the M2DC customised microserver, tailored for an automotive data analytics scenario provided by Vodafone Automotive Telematics, and leveraging specific accelerators – called System Efficiency Enhancements (SEE) in the M2DC parlance – developed by Politecnico di Milano and AS+ Groupe EOLEN. The SEEs focus on accelerating key portions of the data analytics application, namely the decryption of source data and the computational kernels of the analysis. The complete appliance provides a key demonstrator for the M2DC microservers.

Organization of the paper The rest of the paper is organized as follows. Section 2 introduces the M2DC approach to provide turnkey appliances for data centres. Section 3 describes the automotive use case and its requirements, while in Section 4 we describe the main System Efficiency Enhancements which will be embedded in the IoT Analytics appliance, obtaining an high-performance, low-energy solution for the automotive use case. Finally, Section 5 and 6 present the roadmap towards higher technology readiness levels for the use cases and the conclusions that can be drawn from this work.

2 The M2DC Approach

The M2DC project [13] aims at developing turnkey appliances for specific application domains, leveraging a novel modular server architecture featuring heterogeneous processing elements, including GPGPU and custom-developed reconfigurable accelerators. Such appliances are going to achieve a lower Total Cost of Ownership (TCO), thanks to improved energy-efficiency, dependability, customisation, scalability, and integration. The M2DC servers are composed of a baseboard which connects up to 16 microservers with a dedicated high-speed, low-latency communication network, which instead supports also connection to storage and I/O extensions. Microservers can feature Intel x86_64 or ARM Aarch64 server processors, as well as low-power solutions (e.g., based on NVIDIA Jetson), GPGPUs, or FPGAs, allowing individual appliances to tailor the architectures to their specific energy/performance trade-offs. The M2DC appliances can hinge upon a set of so called System Efficiency Enhancements (SEE), which are low-power accelerators for common tasks in the data-centres scenario, such as bulk data encryption or pattern matching. Given the requirements of M2DC microservers, in particular achieving a low TCO for appliances and allowing the flexible allocation of a task among different nodes, ASIC implementations are not considered in the M2DC project. Full details of the server architecture and its possible configurations can be found in [21].

3 The Automotive IoT Scenario: Driver Identification

Willing to assess the effectiveness of M2DC infrastructure in providing high throughput and energy efficiency to applications addressing real world data analytics scenarios, we focused on a common task in the automotive area: the processing and the analysis of data gathered by on-vehicle monitoring devices.

3.1 The Driver Identification Scenario

The adoption of embedded devices in vehicles, equipped with a variety of sensors, has recently been observed as an emerging trend, especially among insurance companies. The data collected are often used to: (a) develop more complex and adaptive Driver Assistance System (DAS) or (b) profile drivers, in particular for anti-fraud purposes. In this context, there are mainly two similar problems in literature: Driving Style Classification and Driver Identification. In the first case, the goal is to classify a driver according to predefined driving styles (e.g. calm/aggressive, lawful/unlawful) [20] in order to provide feedback to the driver with the extent of optimizing the energy usage of the car or improving the ride comfort. In the second case, the goal is to uniquely identify the driver for a given trip, which is useful for insurance purposes and anti-theft methods [1,14]. In particular, the IoT application, presented in this work, addresses the problem of identifying the number of people usually driving a vehicle, a problem closer to the latter case. To deal with this type of problem, three main approaches have been identified in a comprehensive survey of the literature [20]: rule-based, model-based and learning-based.

In particular, data-driven machine learning algorithms became one of the promising solutions due to the increasing volume of data gathered by sensors. Nevertheless, there are two relevant limitations in most of the previous works: they rely on *input data* mainly retrieved with invasive methodologies [9,11,18] and they leverage supervised techniques [9,18,24] (e.g., SVM, Random Forest Classifier, Neural Network). The former requires the reading from the vehicle Electronic Computed Board or the CAN bus, leading to compatibility issues between different car manufactures and safety-related concerns. The latter approaches require a labelled training data, which is generally harder to be obtained in real world application scenarios.

Our proposed approach overcomes the aforementioned issues through a data analytic workflow specifically designed to tackle the identification of number of drivers problem by using *unsupervised* technique on data collected with *non-invasive* methodologies. In this section, we present this workflow, which will be accelerated on the M2DC infrastructure. Further details of this workflow can be found in [19].

Driver Identification Workflow. In the context of the M2DC project, Vodafone automotive provided some data collected on real vehicles. This dataset contains motion data (speed of the vehicle and accelerations on the three axis), geolocalization data (position and altitude) and some additional information, such as the type of road run across by the vehicle (highway, urban etc.) and its speed limit. These data are sampled at 1 Hz rate. A set of such measurements from engine switch-on to engine switch-off is denoted as *trip*. To devise our workflow, we hinge upon a reasonable assumption: there is only one driver for each trip, hence we can perform driver identification trip-wise instead of sample-wise.

The idea of our workflow is to represent a trip with a set of features characterizing the driving style; we expect that trips of the same driver are quite similar, hence by clustering all the trips of the same vehicle, the trips of the same driver form a cluster. Therefore, the workflow estimates the number of usual drivers as the number of clusters found in the set of trips of the same vehicle. The first challenge of this approach

resides in the design of a set of features able to characterize the driving style. To this extent, we discard the geolocalization data from the trips, since these information are not related to the driving style. Then, we devise a set of features computed from the remaining data of a trip. These features are either statistical measures (mean, variance, skewness, kurtosis) of motion data or specific values representing a particular aspect of the driving style (number of speed infringements, number of acceleration peaks to denote a nervous driving style).
A second challenge of our workflow is the need to reduce the number of features characterizing each trip. This avoids the well-known *curse of dimensionality* issue: points in high dimensional spaces are generally too far from each other to get any cluster. In our case, a trip characterized by m features is considered as a point in m dimensional space by the clustering algorithm, hence we want to keep this number m low. This requirement is not satisfied by the set of features of our application scenario, whose size is $m{=}40$. Therefore, the well-known Principal Component Analysis (PCA) is used to shrink the dimensionality of the data and fulfill this requirement for the clustering algorithm. Indeed, PCA is able to represent the dataset in new components, such that the first few ones are sufficient to retain an high percentage of statistical significance of the original data. In our case, by retaining 75% of statistical significance, we are able to shrink the dimensionality of the data from 40 to $5{-}7$ components, depending on the specific dataset of a vehicle.
After dimensionality reduction given by PCA, the trips are ready to be clustered. Among the variety of clustering approaches, given that specifying the number of clusters to be formed in advance would make the results of our workflow meaningless, we choose to employ density-based clustering. In particular, we employ the *dbscan* [10] algorithm, whose input parameters are estimated according to heuristics proposed in [10, Section 4].

Data Confidentiality Issues. Data processed by the IoT application concern the driver's personal behavior, such as speed infringements and GPS locations. Therefore, if these data are stored on a remote server, their confidentiality against possible security breaches needs to be guaranteed. To address this relevant issue, we are going to add a decryption layer to the application, allowing the storage of encrypted data. The addition of this layer should be straightforward, since there already exists an R wrapper for the well-known OpenSSL cryptographic library. Furthermore, among the System Efficiency Enhancements (SEE) developed in the context of M2DC project, there is an FPGA accelerator for bulk data encryption. Thus, we are planning to employ this Cryptographic SEE available in the project (see Section 4.1) to speed-up data decryption needed by the IoT application.

3.2 Implementation Details and Preliminary Experimental Results

A baseline implementation of the proposed workflow is a single core application, employing the R programming language. This choice is motivated by two factors: (a) R is widely employed for data analytics tasks, since it provides several packages and functionalities commonly required in these applications; (b) R is supported by the heterogeneous compilation toolchain developed by AS+, which enables the possible heteroge-

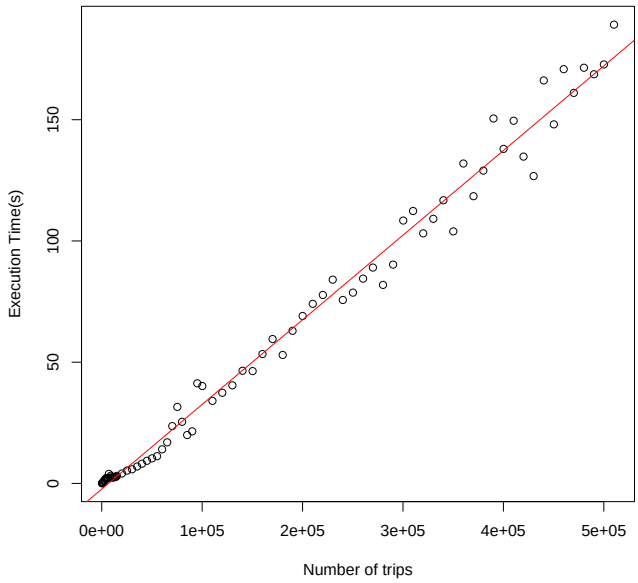

Fig. 1. Execution time of PCA and clustering w.r.t. the number of trips. The red line represents the linear regression model: $Time \approx 3.490 \cdot 10^{-4} \cdot n$.

neous platform exploitation running the algorithm both on CPU and on GPU, improving both the performances and the energy consumption of the application (see Section 4.2). Hinging upon this baseline implementation, we evaluate the application workflow focusing on two aspects: (a) the estimation of the algorithm accuracy; (b) the analysis of the computational complexity and scalability of the data analytics portion of the application, i.e. PCA and clustering. Conversely, since feature extraction is mainly I/O bound, we will not mention its performance evaluation.

Accuracy analysis. At this step we aim to evaluate if the proposed approach produces realistic results. Unfortunately, the dataset provided by Vodafone Automotive is unlabelled: we do not know the actual number of drivers for each vehicle. This hardens the validation of our workflow, since we do not know if the number of drivers identified is correct. To overcome the aforementioned limitations and maintain consistency with the geosocial conditions of the dataset, we derive the average number of drivers per vehicle for UK, the country where Vodafone Automotive actually gathered these data. From the publicly available government data [12] we get the average number of adult people per vehicle and the number of drivers per adult people, thus obtaining an average value of drivers per vehicle of 1.095. Our workflow identifies more than one driver in around 10% of the vehicles, which is in line with the aforementioned deduced geosocial data.

Scalability analysis. The goal of this step is providing a scalability evaluation in terms of execution time with respect to the number of trips n of the same vehicle. First

of all, we expect that the DBSCAN algorithm, which has a complexity of $O(n \log n)$, dominates the average-case complexity of the overall workflow, given the linear execution time of PCA. Then, in order to measure the actual execution times, we setup a server machine, equipped with AMD Opteron-8435 cores and 128 GB of RAM, running the single-core R implementation of the application. The experimental results, shown in Figure 1, highlight that the value of execution time can be approximated by a linear trend, due to the minimal impact of the $\log n$ term. Regarding the absolute execution times, we observe that the current application is able to perform PCA and clustering for a vehicle with $500k$ trips in approximately 3 minutes. We remark that this analysis is expected to be performed for thousands of customers, in turn requiring several days of computation. Therefore, in order to improve the scalability of the application with the number of vehicles, we will hinge upon the R compilation toolchain developed by AS+ (described in Section 4.2), which provides the following benefits and capabilities: (a) the application could be run on a single GPU instead of several high-end CPUs, in turn allowing the decrease of both TCO and energy consumption; (b) the R application is compiled instead of being interpreted, thus it is executed directly from machine code; (c) the compilation toolchain is able to perform several optimizations which are generally hard to be directly applied on R code (e.g., exploiting data parallelism and the vectorized operations available in recent CPUs).

4 The IoT Analytics Appliance

The M2DC appliance employed for the IoT application will be tailored to the performance and energy requirements of the application. As the IoT application focuses on the processing of large quantities of data, with a computational effort mostly devoted to small set of kernels, the plan is to use x86_64 servers attached to accelerators suitable for two types of kernels: encryption primitives employed to access the securely stored data and acceleration of R primitives. Specifically, for the encryption acceleration, the appliance will include an FPGA module based on the Altera Stratix 10 SX[5] series. The FPGA will be exploited via a dedicated OpenSSL engine. For the acceleration of R primitives, the appliance will enclose a parallel R optimized implementation, obtained with the R compilation toolchain, which performs data analytics computation of our workflow.

In the rest of this section, we provide an overview of these two acceleration techniques, including initial experiments carried out in isolation. During the final phase of the project, we will integrate the application on the final testbed.

The resulting appliance will be highly reusable, thanks to the seamless integration of the accelerators in standard components: the cryptographic SEE employs OpenSSL as its interface, whereas the data analytics accelerating toolchain is installed as part of the R compiler. Thus, the application can be run with or without the accelerators with minimal modifications – essentially, it is sufficient to specify if the application employs default OpenSSL implementations or the ones provided by our SEE to perform the cryptographic operations.

[5] The accelerator was developed on an Arria 10 GX board, but Stratix 10 is the expected target for the deployment of the accelerator in the M2DC context

Table 1. Comparison of the energy efficiency of the OpenCL implementations for FPGA accelerator of the nine ISO standard block ciphers, on a Intel Xeon E5-1505M v6 CPU based host with DDR4-2133 DRAM. The CPU AES implementation employs the dedicated `AES-NI` instructions

Block Cipher	Platform	Throughput (MB/s)	Power (W)	Energy Efficiency (GB/J)
AES	FPGA	1020.8	15.02	67.96
	`AES-NI`	1678.4	25.79	65.09
HIGHT	FPGA	824.0	21.87	37.68
	CPU	20.0	26.01	0.77
DES	FPGA	764.8	12.62	60.61
	CPU	85.0	22.24	3.82
CAST5	FPGA	814.2	13.63	59.75
	CPU	132.0	23.14	5.70
SEED	FPGA	930.8	14.49	64.23
	CPU	102.3	22.46	4.54
Camellia	FPGA	957.6	15.06	63.59
	CPU	198.0	22.68	8.73
MISTY1	FPGA	1006.0	25.59	39.31
	CPU	22.0	26.07	0.84
CLEFIA	FPGA	1202.0	22.60	53.19
	CPU	3.0	26.38	0.11
Present	FPGA	979.0	25.57	38.29
	CPU	11.2	23.87	0.46

4.1 Accelerating Cryptography

One of the fundamental application domains for datacenters is represented by bulk data encryption and decryption, as it has to be performed on the data being stored as well as on data being transmitted or received. For this purpose, fast disk encryption software support has been explored using GPGPUs [2].

Cryptography SEE proposed in M2DC [7] exploits the OpenCL programming model to realize high-performance FPGA accelerators. This way provides a viable and more versatile alternative to the use of ad-hoc cryptographic accelerators, currently available in high-end server CPUs only. OpenCL provides functional portability across a wide range of platforms, which have recently been extended to support FPGAs. However, it is well known that performance portability is much harder to achieve. A domain-specific analysis for symmetric encryption highlighted that even within the architectural class of GPGPUs there are major differences, requiring an almost entire rewriting of the encryption code to achieve optimal performance [3]. We therefore analyzed the programming practices to exploit the High Level Synthesis (HLS) toolchains available

to deploy OpenCL programs onto FPGA based accelerators, identifying a set of best-practices to design OpenCL kernels for FPGA. In particular, we found out that single work-item, single work-group implementations are more suitable for FPGAs. We also identified other best-practices related to loop optimization, caching techniques, memory coalescing, I/O latency hiding and host-side synchronization, whose details can be found in [7].

By exploiting the best-practices identified, we were able to obtain high throughput and energy efficient implementations for all the nine ISO standard ciphers implemented by the Cryptographic SEE. The comparison between our FPGA implementations and CPU ones in terms of throughput and energy efficiency is reported in Table 1. We observe that the throughput achieved on the FPGA by all ciphers is between 750 MB/s and 1200 MB/s, with a speed-up over software implementations ranging from $5\times$ to $400\times$. It is worth noting that the throughput of the FPGA implementations is currently limited by the PCI-Express channel employed in our host. At the moment this limits the communication to be half-duplex, introducing significant idle time in FPGA kernels due to the time spent on waiting data transfer completion. In particular, we believe throughput could even been doubled in case a full-duplex communication channel is available. From the point of view of energy efficiency, our FPGA implementations achieve an average improvement of $22.78\times$ (geometric mean) and $79\times$ (arithmetic mean) in terms of GB encrypted per Joule spent. Finally, we remark that our AES implementation is more energy efficient than Intel `AES-NI`, which is an hardware AES implementation included in Intel CPUs usable via a specific assembly instruction added to the ISA. Indeed, our implementation exhibits a comparable throughput and lower power consumption.

Currently, the cryptographic SEE is embodied in an ad-hoc OpenSSL engine, making its usage quite transparent to the application. Indeed, OpenSSL library exhibits a standard interface for each cryptographic operation; OpenSSL engines provide different implementations of these interfaces. Therefore, an application simply needs to specify which engine must be employed to implement interfaces for several cryptographic operations, requiring a minimal modification to the application code. Therefore, shipping the SEE in an OpenSSL engine enables an easy integration at application level, which is a desirable feature to boost the usage of the cryptographic SEE.

As discussed in Section 3, we will introduce an encryption layer on the IoT application to ensure data confidentiality and we will hinge upon this SEE to accelerate the cryptographic operations required by the IoT application. In this scenario, where an R wrapper exists for OpenSSL, the availability of the SEE as an OpenSSL engine greatly aids its integration. Nevertheless, the interface exposed by this wrapper appears insufficient for the integration of the SEE. Indeed, the wrapper provides only AES algorithm as a symmetric cipher and there are no functions to manage OpenSSL engines. Nevertheless, we analyzed the wrapper and we identified the modifications required in order to handle OpenSSL engines and other symmetric encryption algorithms directly from the R application; thus, we will obtain such an enriched wrapper without a significant effort through these simple modifications.

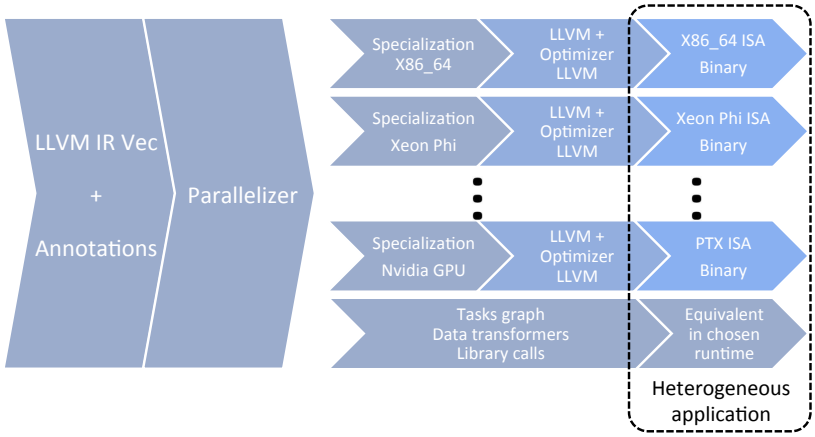

Fig. 2. Back-end architecture for the compiler

4.2 Accelerating Data Analytics R Application

R is a publicly available implementation of the high-level S language for statistical computing. The R language is now widely used for statistical calculations in various application fields such as biology, medicine, financial applications. R language is not widely regarded as a platform for developing scalable, high-performance codes: indeed, it is an interpreted language and the execution of programs typically involves dynamic allocation of large data structures, particularly arrays. However, several approaches have emerged to address both performance and scalability of R programs, which have however showed some relevant limitations. In particular, distributed approaches rely on process-based distribution framework such as MPI and Map Reduce. These approaches imply that each process is a separate instance of the R interpreter without any shared memory. These approaches include RMPI[6], an implementation of MPI bindings in R which enables master-slave like scheduling, and Snow Extension (Simple Network Of Workstations)[7], which provides a further level of abstraction above existing communication frameworks, masking communications details and enabling the creation of different kinds of *clusters*. Although they partially address the scalability issues of R, these approaches do not solve the overhead introduced by the interpretation layer and hardly target heterogeneous systems. Just In Time (JIT) compilers minimize performance degradation of interpreted languages by caching the code once it is translated. There are two R packages providing JIT compilation[8,9], which have been deprecated since 2011. Lastly, full blown compilers, such as RCC [22], are advanced optimizing compiler for R or S programs. RCC is based on three successive stages: static analysis of R program to extract call graphs and dataflows, translation of R programs to C and

[6] https://cran.r-project.org/web/packages/Rmpi/index.html
[7] https://cran.r-project.org/web/packages/snow/index.html
[8] http://stat.ethz.ch/R-manual/R-devel/library/compiler/html/compile.html
[9] http://www.milbo.users.sonic.net/ra/jit.html

eventually optimization of translated C code. This approach suffers from the complexity brought by using the C language as intermediate representation and by the resulting static analysis stage which aims to retrieve high-level constructs already existing in the original language.
Conversely, our R compilation toolchain addresses the aforementioned limitations: it is a full compiler from R to machine code, which also targets heterogeneous architectures and which is able to take into account parallel constructs in its intermediate representation. In the next paragraphs, we present some of the main features of our toolchain.

Data Flow Execution Model. The approach of a data flow runtime for transparent heterogeneous execution is already known and available nowadays in some runtime implementations, such as starPU [6], StarSS [15] or Quark [25]. However, programming on top of those runtimes is complex and when heterogeneity is involved, one code must be written for each target. This problem is common also to parallel programming models such as OpenCL [3,5,4]. Our approach aims to provide transparent compilation for heterogeneous targets, with direct support of data flow runtime, for an high-level DSL like R. To ease the achievement of these goals, we define our own programming model, which relies on the following concepts:

- all data are either managed by the runtime or used for control flow and simple parameter of a function;
- a program is a bunch of actions on numerous data, where the control of ordering the actions is done by evaluating data dependency;
- a task is a function with two kind of parameters: managed data references and control parameter which are copied argument and cannot be reaccessed outside of the task .

Therefore, we need a runtime compatible with these concepts and that handles multiple implementation of a task. It also requires an efficient heuristic to choose the place of execution and the best implementation of a task on this place. StarPU fits in those constraints, so we chose it as our effective execution runtime.

A modular design. The full toolchain and runtime support is designed as a modular framework by leveraging the modular property of LLVM [16] based compilers. Following the design principles of LLVM framework, the compilation workflow is split into two main stages: the input program is first translated by the front-end into an Intermediate Representation (IR), which is further translated and optimized for the target architecture by the back-end. Therefore, the front-end is programming language specific, while the back-end is related to a specific architecture. Our toolchain strongly decouples these stages, allowing to independently choose the front-end and back-end to be employed. In order to support multiple architectures from one representation, we did extend the LLVM IR at some points. These IR extensions are then handled by specialized LLVM passes in order to generate a complete program on top of starPU. For instance, vectorization is easier to be performed in the front-end, thus we need a way to express vectors operation without the knowledge of vector unit size. To this extent, we introduce an unknown sized vector type to handle manipulation of data in an abstract

way.

Code Parallelization. Despite R programming model is sequential, our toolchain aims at building parallel program from a standard R program. In particular, we can induce parallelism if the functions are using non overlapping data at some point; the analysis of these data dependencies is easy since each task announces both the data it uses and the intent on them.

Task based runtime support. Our toolchain, when compiling the task code itself, generates multiple implementations for each possible target as well as an initialization function which creates the runtime multi-implementation structure binding each implementation to the runtime. For instance, we generate multiple versions of one task to handle different vector unit sizes on an x86 processor (128, 256 and 512 bits wide). We also generate the test functions to execute the best possible implementation depending on runtime parameters. The calls for an extracted task are then transformed into a runtime call for further submission to the runtime. The runtime itself is built on top of starPU, with some wrapping function to ease code generation and support function for managed data manipulation. Another interesting feature of our toolchain is the availability of a runtime interface to prepare plugin development for supporting other runtimes without any modification of the compiler itself, provided that this new runtime is able to implement the interface to our toolchain. In conclusion, by hinging upon our toolchain, it is possible to obtain a parallel heterogeneous (CPU/GPU) application from a sequential single-core R one without significant modifications to the application code.

5 Towards Advanced Applications

In this section, we highlight some open issues in the IoT automotive data collection and analysis scenario; these problems will not be investigated in the context of M2DC project, but their solution provide some interesting research and innovation directions that are planned for the next future.

Data Privacy. The collection and usage of data from private car and fleets managers can open new business avenues, including insurance price customisation based on driver profiling. Inertial and geo-located data can be collected from a vehicle through aftermarket telematic boxes, endowed with GPRS communication capabilities as well as sensing capabilities. However, such data, which are sufficient for basic applications such as crash detection, are insufficient for more complex ones such as the driver profiling performed in the M2DC scenario. In this case, an integration with third party information, such as meteorological or speed limit data, acquired commercially or from (reliable) open data sources, is needed. Furthermore, in a real-world scenario, different telematic boxes may collect different data, or use different format, becoming a further source of heterogeneity for the assembled data.

Given the nature of the data, privacy is particularly important: simple anonymization by removing the contract holder name and replacing it with a unique identifier can be easily proved insufficient. As an enlightening example, let us consider the scenario of a

malicious employer willing to acquire personal information on his commuting employees. By selecting contracts that consistently show trips ending at or near the company premises during the work start times, the employer can identify a set of employees. Then, by identifying the most common destination of the last trips for each day of the identified employees, the employer can identify an approximate location of their residence, which can be easily compared with data legitimately held by the employer to associate the contract ID with the employee identity, thus foiling the anonymization. Needless to say, the employer can now easily check if one of the employees has, e.g., regular visits to a local hospital, and thereby infer information about the employee's health status he has no right to be privy to.

It is important to understand, however, that the personal information items cannot be easily separated from the non-personal, but useful, elements of the information. For instance, the data could be more strongly anonymized by removing georeferences, but then they would become much less useful to insurance companies willing to customize prices based on the locations usually traveled and the frequency of accidents on those routes.

Scalability. Another key challenge is scalability. Whereas for small fleets trip information can be stored at a fine grain in data centers, scaling up would easily produce a veritable data deluge. In this case, it is necessary to rethink the data collection phase, the processing, and the storage. A viable solution would be to move part of the computation, namely the feature extraction, from the cloud server nearer to the sensor [17]. Since the sensor itself has limited computation capabilities, this implies adding an edge node to the system. The edge node may be endowed with reasonably powerful processing elements at a limited cost, thus enabling complex feature extraction to take place before the data transmission. However, although such an edge node would not require a significant amount of energy to affect the overall vehicle system, key-off operation would be much less viable, unless the edge node is turned off. As a result, the best scenario would be to enable a flexible runtime configuration of the IoT system (sensor–edge node–cloud) such that the edge node can be excluded when operating off the vehicle battery and turned back into operation when the vehicle is running, as potentially discussed in [26].

6 Conclusions

In this article, we presented the automotive IoT scenario employed in the M2DC project to demonstrate the effectiveness of the microserver approach, combined with some of the SEEs developed. We showed how we plan to achieve significant speedups against our preliminary implementation, without significant impact on the cost of development of the application, thanks to the transparent embedding of the SEEs in the existing software stack.

During the next months, we will finalize the integration of CryptSEEs to fully expose their cryptographic capabilities to the R OpenSSL wrapper and we will address the scalability issues by hinging upon the R compilation toolchain. In the longer term, the application will be enriched with additional components aiming at providing increased data privacy so that third party applications can leverage the data set, as well as improv-

ing scalability to larger vehicle fleets through a flexible allocation of feature extraction (currently performed on the data centre servers) to edge and Fog nodes and integration of the driver identification in a more complex, multi-application scenario [23].

Acknowledgements

Work supported by the EU's H2020 programme (grant n.688201), Modular Microserver DataCentre (M2DC).

References

1. Agosta, G., Barenghi, A., Brandolese, C., Fornaciari, W., Pelosi, G., et al.: V2i cooperation for traffic management with safecop. In: 2016 Euromicro Conference on Digital System Design (DSD). pp. 621–627 (Aug 2016). https://doi.org/10.1109/DSD.2016.18
2. Agosta, G., Barenghi, A., De Santis, F., Di Biagio, A., Pelosi, G.: Fast Disk Encryption through GPGPU Acceleration. In: 2009 International Conference on Parallel and Distributed Computing, Applications and Technologies. pp. 102–109 (Dec 2009). https://doi.org/10.1109/PDCAT.2009.72
3. Agosta, G., Barenghi, A., Di Federico, A., Pelosi, G.: OpenCL performance portability for general-purpose computation on graphics processor units: an exploration on cryptographic primitives. Concurrency and Computation: Practice and Experience **27**(14), 3633–3660 (2014). https://doi.org/10.1002/cpe.3358
4. Agosta, G., Barenghi, A., Pelosi, G., Scandale, M.: Towards Transparently Tackling Functionality and Performance Issues across Different OpenCL Platforms. In: 2nd Int'l Symp. on Computing and Networking (CANDAR). pp. 130–136 (Dec 2014). https://doi.org/10.1109/CANDAR.2014.53
5. Agosta, G., Fornaciari, W., Massari, G., Pupykina, A., Reghenzani, F., Zanella, M.: Managing Heterogeneous Resources in HPC Systems. In: Proc. of PARMA-DITAM '18. pp. 7–12. ACM (2018). https://doi.org/10.1145/3183767.3183769
6. Augonnet, C., Thibault, S., Namyst, R., Wacrenier, P.A.: Starpu: a unified platform for task scheduling on heterogeneous multicore architectures. Concurrency and Computation: Practice and Experience **23**(2), 187–198 (2011)
7. Barenghi, A., Madaschi, M., Mainardi, N., Pelosi, G.: OpenCL HLS Based Design of FPGA Accelerators for Cryptographic Primitives. In: Proceedings of the 13th International Workshop on Security and High Performance Computing Systems (SHPCS 2018). pp. 3:1–3:8. IEEE Computer Society (July 2018)
8. Cecowski, M., Agosta, G., Oleksiak, A., Kierzynka, M., et al.: The M2DC Project: Modular Microserver DataCentre. In: 2016 Euromicro Conference on Digital System Design (DSD). pp. 68–74 (Aug 2016). https://doi.org/10.1109/DSD.2016.76
9. Enev, M., Takakuwa, A., Koscher, K., Kohno, T.: Automobile driver fingerprinting. Proceedings on Privacy Enhancing Technologies **2016**(1), 34–50 (2016). https://doi.org/10.1515/popets-2015-0029
10. Ester, M., Kriegel, H., Sander, J., Xu, X.: A density-based algorithm for discovering clusters in large spatial databases with noise. In: Proceedings of the 2^{nd} International Conference on Knowledge Discovery and Data Mining (KDD-96), Portland, Oregon, USA. pp. 226–231 (1996), `http://www.aaai.org/Library/KDD/1996/kdd96-037.php`
11. Fugiglando, U., Massaro, E., Santi, P., Milardo, S., Abida, K., Stahlmann, R., Netter, F., Ratti, C.: Driving behavior analysis through CAN bus data in an uncontrolled environment. CoRR **abs/1710.04133** (2017), `http://arxiv.org/abs/1710.04133`

12. Government, U.: Driving licence holding and vehicle availability (nts02) (2016), national Travel Survey
13. Kierzynka, M., Oleksiak, A., Agosta, G., Brandolese, C., Fornaciari, W., Pelosi, G., et al.: Data centres for iot applications: The m2dc approach (invited paper). In: 2016 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS). pp. 293–299 (July 2016). https://doi.org/10.1109/SAMOS.2016.7818361
14. Kwak, B.I., Woo, J., Kim, H.K.: Know your master: Driver profiling-based anti-theft method. CoRR **abs/1704.05223** (2017), http://arxiv.org/abs/1704.05223
15. Labarta, J.: Starss: A programming model for the multicore era. In: PRACE Workshop 'New Languages & Future Technology Prototypes' at the Leibniz Supercomputing Centre in Garching (Germany) (2010)
16. Lattner, C., Adve, V.: Llvm: A compilation framework for lifelong program analysis & transformation. In: Code Generation and Optimization, 2004. CGO 2004. International Symposium on. pp. 75–86. IEEE (2004)
17. Li, C., Xue, Y., Wang, J., Zhang, W., Li, T.: Edge-oriented computing paradigms: A survey on architecture design and system management. ACM Comput. Surv. **51**(2), 39:1–39:34 (Apr 2018). https://doi.org/10.1145/3154815
18. Ly, M.V., Martin, S., Trivedi, M.M.: Driver classification and driving style recognition using inertial sensors. In: 2013 IEEE Intelligent Vehicles Symposium (IV). pp. 1040–1045 (June 2013). https://doi.org/10.1109/IVS.2013.6629603
19. Mainardi, N., Zanella, M., Reghenzani, F., et al.: An unsupervised approach for automotive driver identification. In: Proceedings of the 1st ACM International Workshop on Intelligent Embedded Systems Architectures and Applications. ACM (2018). https://doi.org/10.1145/3285017.3285023
20. Martinez, C.M., Heucke, M., Wang, F.Y., Gao, B., Cao, D.: Driving style recognition for intelligent vehicle control and advanced driver assistance: A survey. IEEE Transactions on Intelligent Transportation Systems **19**(3), 666–676 (March 2018). https://doi.org/10.1109/TITS.2017.2706978
21. Oleksiak, A., Kierzynka, M., Piatek, W., Agosta, G., et al.: M2dc – modular microserver datacentre with heterogeneous hardware. Microprocessors and Microsystems **52**, 117 – 130 (2017). https://doi.org/https://doi.org/10.1016/j.micpro.2017.05.019
22. Qi, F., Zhang, X., Wang, S., Mao, X.: Rcc: A new programming language for reconfigurable computing. In: 2009 11th IEEE International Conference on High Performance Computing and Communications. pp. 688–693 (June 2009). https://doi.org/10.1109/HPCC.2009.74
23. Sansottera, A., Zoni, D., Cremonesi, P., Fornaciari, W.: Consolidation of multi-tier workloads with performance and reliability constraints. In: 2012 International Conference on High Performance Computing Simulation (HPCS). pp. 74–83 (July 2012). https://doi.org/10.1109/HPCSim.2012.6266893
24. Vaitkus, V., Lengvenis, P., Žylius, G.: Driving style classification using long-term accelerometer information. In: 2014 19th International Conference on Methods and Models in Automation and Robotics (MMAR). pp. 641–644 (Sept 2014). https://doi.org/10.1109/MMAR.2014.6957429
25. Yarkhan, A., Kurzak, J., Dongarra, J.: Quark users' guide. Electrical Engineering and Computer Science, Innovative Computing Laboratory, University of Tennessee (2011)
26. Zanella, M., Massari, G., Galimberti, A., Fornaciari, W.: Back to the future: Resource management in post-cloud solutions. In: Proceedings of the Workshop on INTelligent Embedded Systems Architectures and Applications. pp. 33–38. INTESA '18, ACM, New York, NY, USA (2018). https://doi.org/10.1145/3285017.3285028, http://doi.acm.org/10.1145/3285017.3285028