A Hardware/Software Framework for the Integration of FPGA-based Accelerators into Cloud Computing Infrastructures

Fritjof Steinert* , Philipp Kreowsky* , Eric L. Wisotzky* , Christian Unger† and Benno Stabernack*‡
*Fraunhofer Institute for Telecommunications, Heinrich Hertz Institute
Berlin, Germany
fritjof.steinert@hhi-extern.fraunhofer.de, {philipp.kreowsky, eric.wisotzky, benno.stabernack}@hhi.fraunhofer.de
†CPU 24/7 GmbH Potsdam, Germany
c.unger@cpu-24-7.com
‡University of Potsdam, Embedded Systems Architectures for Signal Processing
Potsdam, Germany

***Abstract*—The need for high computing power has increased enormously in recent years, particularly in the field of image signal processing and machine learning applications, very powerful computing systems are required. It has been shown that homogeneous architectures in data centers work very inefficiently regarding these special applications, showing high latency of the response times and providing a very poor power efficiency. In order to integrate FPGA (Field Programming Gate Array)- as well as GPU-based accelerators into cloud computing infrastructures as compute nodes we present a generic hardware/software framework for using heterogeneous computing systems. A real industrial image processing application shows the acceleration achieved.**

***Index Terms*—Heterogeneous Computing, Accelerator, FPGA, Cloud Computing, Resource Management**

I. Introduction

In order to integrate FPGAs (Field Programming Gate Array) and GPUs into cloud computing environments typically found in data center installations, it is necessary to provide a generic way for the usage of the different accelerators. Vendor or application-specific frameworks, such as those used to integrate machine learning accelerators, are usually offered on a Software-as-a-Service (SaaS) basis which does not provide the freedom to use the environment for different workloads. Besides application specific frameworks, the direct opposite can be found where FPGA- and GPU-based compute resources are offered on a IaaS (Infrastructure as a Service) basis e.g. Amazon Web Services (AWS) EC2 F1-Instances [1]. This has the disadvantage that the user gets access to an environment, whereas the installation of all required software is on his own responsibility. With the hardware (HW)/software (SW) framework presented in this paper we are aiming at a scalable, generic and power efficient way to use the particular hardware resource of a cloud computing system on an AaaS (Accelerator as a Service) basis. Our framework consists of a generic hardware architecture providing a simple, standardized approach to integrate application-specific hardware accelerators for FPGAs as interchangeable IP cores, which can be reconfigured at run-time or as a static implementation at design time. In addition to FPGAs, we are targeting both GPUs and SW-based accelerators, both managed by our run-time resource management system utilizing the Docker container virtualization concept. Our main contributions are three-fold:

- A hardware accelerator framework for flexible usage of FPGAs in high performance clouds.
- A software framework for utilizing the FPGA accelerators together with other accelerators in a heterogeneous cloud computing architecture as AaaS.
- An industrial image processing application that is accelerated by the hardware and software framework. For this we present runtime and energy measurements.

The structure of the paper is as follows: Section II gives related work, Section III gives a short overview of the overall cloud computing architecture that has been used for our experiments. Section IV describes in detail the hardware framework that we have developed to use FPGA-based accelerator add-on cards, whereas Section V provides an overview of the run-time resource management software, that has been implemented for orchestrating the different possible hardware accelerators. In Section VI we provide a description of an actual industrial use case that we have accelerated using the described hardware and software components. The rest of the paper is dedicated to the presentation of the achieved results and a conclusion.

II. Related Work

Ranging from embedded systems equipped with specialized co-processors over desktop computing systems, heterogeneous computing has reached the cloud computing domain as well. Comparable systems are available as rentable cloud computing systems (e.g. AWS). Besides GPU-based systems, FPGAs are increasingly offered as application-specific computing accelerators in data centers. The AWS cloud lacks a runtime resource management system for FPGAs, only an IaaS is provided [1].

This work was funded by BMWi (Federal Ministry of Economic Affairs and Energy) in the M3D project.

978-1-7281-6547-9/20/$31.00 ©2020 IEEE
DOI 10.1109/SmartCloud49737.2020.00014

In [2] a comprehensive overview of FPGA cloud installations is given. Regarding the presented use cases, the accelerators here are primarily used for the acceleration of search engines [3], data bank applications using memcached [4] and the acceleration of Mapreduce-based applications as described in [5]. The majority of the presented systems do not offer the freedom to use the accelerator computing resources for other applications and are tightly bound to their target application. However, the free use of FPGAs for different accelerators in a data center is necessary to ensure that resources are used at a high utilization rate. In [6] a virtualization concept for FPGA-based accelerators is presented which provides infrastructure for direct communication between accelerators as well. The presented HW/SW system is limited to the management of FPGA accelerators and does not provide support for GPUs. The authors in [7] present a comprehensive OpenStack based cloud integration of virtualized FPGA resources, which shows good results, but also lacks the GPU integration.

III. Cloud Computing Architecture

Our targeted scalable and dedicated high performance computing cloud architecture, depicted in Fig. 1, consists of three basic components: cloud management, cloud storage and compute nodes with heterogeneous hardware accelerators (e.g. CPUs, GPUs, FPGAs). The cloud management services (e.g. user identification and authentication, network management and telemetry) are managed from dedicated controller nodes. Performance-uncritical communication for management purposes (e.g. Intelligent Platform Management Interface) is performed over 1G Ethernet networks. An open-source software-defined storage (Ceph) is used as backend for object storage as well as network-attached block storage for compute resources like bare-metal nodes and Podman containers providing fast data exchange. All communication between performance-critical compute components as well as storage traffic (e.g. I/O, data replication) is handled by a low-latency (approx. 90 ns) and high-bandwidth (up to 100 Gbit/s) dual-rail InfiniBand EDR interconnect. The designed cloud architecture is characterized by a high level of security and privacy. The encrypted Amazon S3 RESTful (Representational State Transfer) API is used for incoming and outgoing data transfers from the Internet as well as for internal data transfers to the object storage. For management purposes SSH with multi-factor authentication is used. Only the relevant protocols gain access to the cloud through the used firewall.

IV. Heterogeneous Compute Nodes

To meet the requirements specified for various applications, such as high data throughput, low latency or high energy efficiency (see Sec. VI), a large number of different types of computing nodes should be provided in the data center. In a heterogeneous cloud data center with CPUs (x64- and ARM-based architectures) and accelerators such as FPGAs and GPUs, the ideal processing node can be selected for each application based on performance and energy efficiency

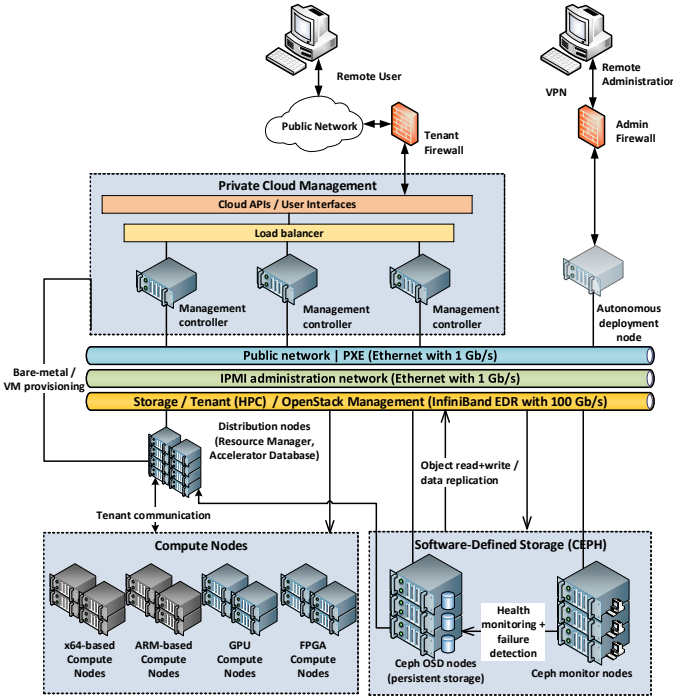

Fig. 1. Cloud computing infrastructure architecture

TABLE I
Used compute nodes in the heterogeneous data center

Node	CPU	RAM	Used Accelerator
CPU	Intel Xeon Silver 8 Cores@2.2 GHz	8 GByte DDR4-2666	none
GPU	Intel Xeon Silver 8 Cores@2.2 GHz	8 GByte DDR4-2666	4 Tesla V100 16GB
FPGA	Intel Xeon Silver 8 Cores@1.8 GHz	8 GByte DDR4-2666	Nallatech 385A

requirements. For our application we have used the compute nodes listed in Table I.

All add-on cards are integrated into rackmount server systems via standard PCIe interfaces using the provided vendor specific operating system driver.

A. FPGA Accelerator Framework

For a simple and rapid design process for FPGAs, we propose a generic hardware framework called Accelerator Framework, which can be used for any type of FPGA-based accelerator. An exemplary framework design is depicted in Fig. 2. The static framework supports a generic number of reconfigurable accelerators and implements necessary basic functions for operation, such as connection to the host machine via PCIe, memory access to the external memories, and partial reconfiguration of the various Accelerator Sockets. Each accelerator within an Accelerator Socket is unambiguously identified by a Universally Unique Identifier (UUID), which is assigned during the development of the specific accelerator. Different Accelerator Sockets can be used simultaneously for various purposes. When using several sockets, access via the

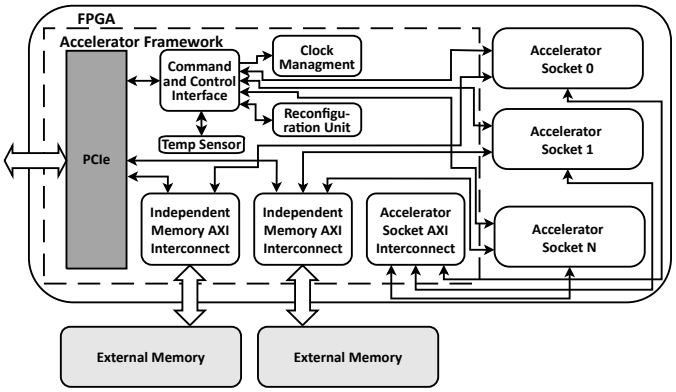

Fig. 2. Example of an FPGA Accelerator Framework with multiple Accelerator Sockets

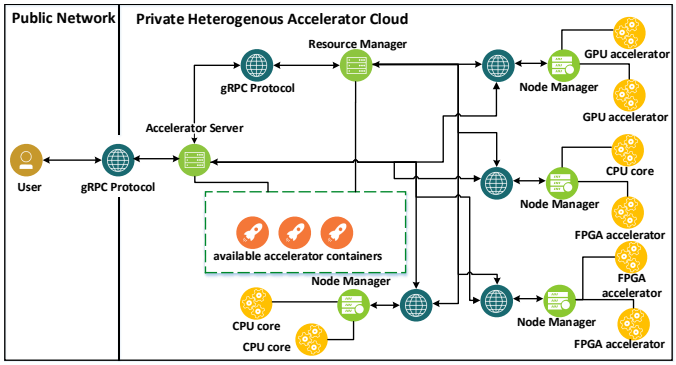

Fig. 3. Used software framework for resource management and task distribution.

common PCIe interface must be scheduled. If several sockets share an external memory, the memory accesses affect each other.

Management: The PCIe-based bus Command and Control Interface (CCI) is used to manage the framework and the accelerator. Within the framework the AXI (Advanced eXtensible Interface Bus) Lite-based [8] CCI is used to select corresponding CCI slaves. Due to the resource-efficient implementation, only one blocking connection between a master and a slave is possible at any time. The UUID for each socket can be read using the CCI. In addition, the UUID of the framework and other status information such as the current FPGA temperature can be read out. To determine the address offset between the CCI slaves, the offset can be read out via CCI. This enables the corresponding software to work easily with different Accelerator Framework versions.

Data Exchange: The Accelerator Sockets are connected to the external memories via a flexible, high bandwidth AXI4 [8] network. Our framework supports an arbitrary number of independent memories per socket. Alternatively, the sockets can use a shared memory to enable data exchange. Simultaneous access to different slaves is possible due to a parallel crossbar. An exclusive access of an accelerator to a memory slave is carried out with up to 14.9 GB/s using a 512 bit data bus. A PCIe DMA controller with a generic number of independent DMA channels is also connected via the AXI network to the memories for high speed data transfer with up to 4.9 GB/s between the host and accelerator. The use of DMA minimizes the workload of the host CPU.

Reconfiguration: The Reconfiguration Unit (RCU) is used for partial reconfiguration of the Accelerator Sockets. The reconfiguration data are transferred to the RCU via PCIe after the accelerator is stopped. This ensures that there is no running communication at the interface to the appropriate accelerator at the time of reconfiguration. A PLL (phase-locked loop), reconfigurable via CCI, supplies each accelerator with an individual core frequency so that the highest possible clock frequency is maintained even after partial reconfiguration.

The concept of Accelerator Sockets allows a free use of the existing interfaces and an arbitrary interpretation of the received or sent data within each accelerator. This enables the use of the Accelerator Framework in a wide range of applications. The resource usage of the framework is shown in Table II. To minimize the resource overhead of the framework, we aim for a small number (up to 4) of Accelerator Sockets. For the design of an accelerator for an Accelerator Socket, predefined components and generic templates are used.

V. Run-time Resource Management

The flexible use of heterogeneous computing accelerators, especially FPGAs, in cloud data centers as distributed systems is complicated by the difficult integration into existing structures, regardless of all advantages such as improved energy efficiency, low latency and overall increased computing power. In order to enable straightforward and efficient sharing of various accelerators in a data center, a software framework is required that manages the existing heterogeneous resources and assigns them different tasks. The proposed software framework consists of the Acceleration Server, the Resource Manager, a private container repository and various node managers, as shown in Fig. 3. The various components communicate via gRPC (gRPC Remote Procedure Calls), which is based on remote function calls and provides secure authentication and encryption as well as high scalability [9].

The software framework should act as a thin layer to the hardware for applications which demand high computing power and low latencies. At the same time, applications need to be encapsulated for management and security reasons. Since virtual machines have a long startup time, the software framework uses containers [10]. From outside the cloud, access to the accelerator is only possible via the Acceleration Server according to the AaaS concept. The chosen virtualization architecture with containers for individual computing tasks allows the compute nodes to be shared. Thus, various tasks can be processed on different accelerator sockets of an FPGA.

A. Acceleration Server

The Acceleration Server is used to select the optimal accelerator according to the criteria availability of an algorithm, run-time, energy efficiency and availability of the corresponding resource. The Acceleration Server can be accessed via an

API and thus serves as an easy to use interface of the private AaaS cloud to the outside world. The server authenticates the requesting user and queries the Resource Manager for the available resources, which selects the desired resources according to the relevant criteria. At the same time, the availability of the algorithm is checked in the Accelerator Database. The selected resources will be reserved at the Resource Manager. The Acceleration Server directly configures the Node Managers of the selected resources, transfers necessary data to the Node Managers either directly or via an object storage server and waits for the results. Any data can be transmitted. After the calculation has been completed, the server releases the used resources at the Resource Manager.

B. Resource Manager

The Resource Manager administers all available accelerator resources in a database along with information about their specific capabilities. The suitable algorithms for the different accelerator types (CPU, GPU and FPGA) are kept within Podman containers for immediate use in a private repository [11]. Podman containers are containers compatible to Docker, which among other things allow a rootless execution on the compute node. The private repository is used to avoid external traffic outside the accelerator cloud network. An executable version of the algorithm for the corresponding accelerator is installed inside a container. For FPGA accelerators, a bit stream for partial reconfiguration is installed in the container in addition to the device driver. This bit stream is used to reconfigure the corresponding Accelerator Socket with the accelerator when the container is started. The containers can be created for the appropriate accelerator algorithm and added to the private repository via a web interface.

Each Node Manager registers itself at the Resource Manager with a list of usable resources. The Resource Manager returns a unique Node ID. When the Acceleration Server queries for available resources for a given application, the Resource Manager provides it with a list of usable accelerator resources. In case of several requests for an accelerator service, they are collected in a queue and processed one after the other. After the Acceleration Server has selected the resources for use, they are marked as used in the database and reserved for exclusive use by the application at the corresponding Node Manager. After the Acceleration Server provided a notification of the completion of the calculations, the resources used in the database and the corresponding Node Managers are released for reuse. In order to detect unforeseen unavailability of individual nodes, the Resource Manager regularly asks for a sign of life from the respective Node Manager.

C. Node Manager

The Node Manager runs permanently on every bare-metal compute node and is used to manage and control the accelerator resources as well as for secure data transfer to and from the accelerator. After starting a Node Manager, the number of available resources of the node like CPUs, GPUs and FPGAs is delivered to the Resource Manager, which returns a unique Node ID. An update will be sent if the usable resources of a node changes during run-time. When a resource is selected for use by the Acceleration Server, the appropriate Node Manager is configured. The respective Podman container is loaded and started by the node manager from the repository. Each container has access to one or more FPGA or GPU accelerators in the compute node. In case of GPU usage, remote GPUs can also be accessed via rCuda for better resource utilization [12]. If an FPGA accelerator is used, then a partial reconfiguration is performed if the socket is not already configured with the suitable accelerator. The data for the algorithm is either sent directly from the Accelerator Server to the Node Manager and then passed on to the accelerator, or the exchange takes place via an object storage server. After the Resource Manager notifies that the algorithm has been completed, the Node Manager terminates the started container. When the compute node is shut down, the Node Manager logs off from the Resource Manager first.

Orchestration frameworks such as Kubernetes can be used to simplify the distribution of the Node Manager as well as the Acceleration Server or the Resource Manager.

VI. Industrial Use Case

We have used our framework to accelerate the following computationally intensive problem in an industrial context: The supply management of industrial parts is often challenging. Many of the industrial machines that require spare parts consist of thousands of different parts. In the maintenance process, a technician has to identify the correct spare part on site to replace it. Especially in the mobility sector, most of the maintained objects have a long lifetime and are adapted by the operator during its maintenance. Additionally, due to the long lifetime, the required spare parts are not clearly identifiable. Up to now, no automatic framework exist to support the identification of spare parts, due to the following technical challenges. Most of the industrial produced parts exhibit primarily large homogeneous, structureless surfaces and are either made of metal or chrome-plated and therefore highly reflective or made of plastic, usually black, and therefore light absorbing. These characteristics complicate both an image-based object identification as well as a photometric three-dimensional (3D) reconstruction for object detection, identification and database search using uncalibrated images. 3D scene reconstruction has been an area of important research since the structure-from-motion (SfM) problem became more robust. SfM yields good quality if there are many matching correspondences between multiple images, which requires a well structured and nonuniform scene [13]. To handle the reconstruction of homogeneous and structureless objects, several approaches exist, but each approach is limited to its specific setup and cannot be used for accurate industrial object reconstruction with identification afterwards. Including edge information in the reconstruction process seems to be a promising option as it generally has no restrictions to a specifically defined setup [14].

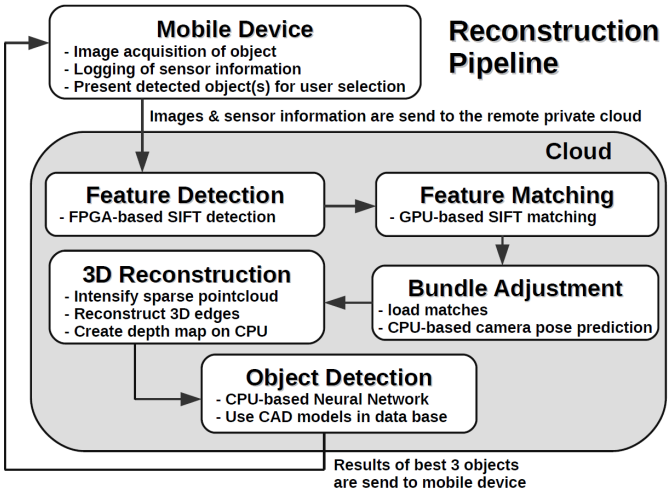

Fig. 4. Object identification processing pipeline

Our proposed processing pipeline, which is shown in Fig. 4 reconstructs homogeneous and structureless objects by combining local object features with edge information. The extracted edge information can improve the camera positions, previously estimated by the framework, and form a wire-frame model which can be combined with the sparse point cloud of the scanned object. In order to achieve a high reconstruction quality, which is necessary for industrial components, a high number of images is necessary. In addition, the time factor plays a decisive role in maintenance work and therefore low-latency 3D reconstruction is an essential aspect. For this reason, the reconstruction pipeline is moved from the mobile acquisition system into the cloud to ensure a fast response within a few minutes by using a large number of compute nodes supported by application-specific accelerators. Based on comprehensive profiling runs of the overall processing pipeline it has been shown, that the generation (SIFT) and processing (matching) of the feature points are the most demanding steps regarding computational complexity.

To use an AaaS SIFT (Scale-invariant feature transform) or matching accelerator, the appropriate API must be addressed by the user program. By including the API into a mobile application, the Acceleration Server is addressed, which then controls the further use of the service according to the description in Sec. V. The processing pipeline is started after the images are transferred to the Ceph storage server in the cloud. The Acceleration Server requests accelerator services from the Resource Manager based on the number of images. The server selects the desired resources. The Resource Manager then starts the relevant containers. These are configured by the Acceleration Server. Each Node Manager fetches the necessary data for its calculations from the Ceph storage server and stores the results there. After completion of a sub-calculation, the resource is used for further calculations. After all calculations are completed, the results are transferred from the Acceleration Server to the application and the resources are released for new AaaS requests.

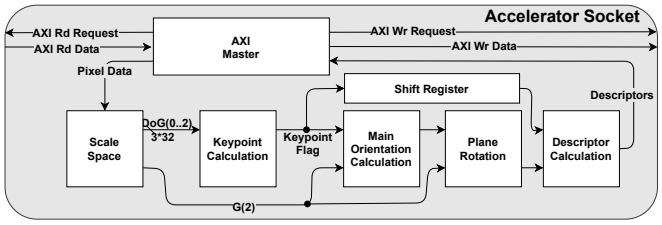

Fig. 5. SIFT accelerator

TABLE II
FPGA RESOURCE UTILIZATION FOR A SIFT ACCELERATOR

Entity	ALM	Block RAM	M20K	DSP
available	427 k	55.6 Mbit	2713	1518
SIFT Core	102 k	17.6 Mbit	1269	301
Accelerator Framework	27.7 k	1.8 Mbit	153	0
- Vendor IP Cores	23.8 k	959 kbit	89	0
- PCIe App Layer	409	526 kbit	29	0
- Memory Interconnect	2950	288 kbit	30	0
- CCI Interconnect	574	2416 bit	5	0
Total	**130 k**	**19.4 Mbit**	**1422**	**301**

A. SIFT Accelerator

To extract SIFT feature points from a given image we have developed a high performance, low latency (below one frame) and energy efficient FPGA accelerator which is depicted in Fig. 5 running at 200 MHz. Due to the high BRAM (Block RAM) requirements of the accelerator we used a framework with a single Accelerator Socket and a single external memory. The accelerator calculates keypoint locations according to [15]. The calculated keypoints trigger the main orientation calculation entity whose output is used to rotate the input data for the keypoint descriptor calculation entity. Table II shows the resources usage for a single SIFT accelerator in an Arria 10 FPGA. As can be seen, with 6.5% utilization of the ALMs (Adaptive Logic Module), 5.6% of the M20K blocks and 0% of the DSPs, the framework again leaves most of the FPGA resources available for the accelerator.

B. Matching Accelerator

In the algorithm, the matching of two images represents a computationally intensive subproblem. To solve this problem we have implemented a brute-force matching GPU accelerator for SIFT descriptors based on OpenCV. For each feature point the Squared Euclidean distance to all other feature points is calculated. The eight nearest neighbors are selected for further processing. After the first matching, the matched feature points can be used to calculate a point cloud. Therefore the latency depends on the processing time of each image pair.

VII. RESULTS

To compare the accelerators used with standard software, we performed the algorithm with 36 HD test images showing the object from different directions. In the process, we measured the run-time and energy consumption both for the complete compute node and for each PCIe accelerator card. For this

TABLE III
COMPARISON OF DIFFERENT ACCELERATORS FOR SIFT DETECTION

Node	Run-time	Server Energy	PCIe Card Energy
CPU Node	87.6 s	15261 J	
GPU Node	14.4 s	2014 J	469 J
FPGA Node	1.9 s	232.2 J	51.2 J

TABLE IV
COMPARISON OF DIFFERENT ACCELERATORS FOR MATCHING

Node	Run-time	Server Energy	PCIe Cards Energy
CPU Node	16 min 17 s	154971 J	
GPU Node	2 min 49 s	316341 J	188002 J

purpose we have developed a power measurement system for all relevant supply voltages. However, a detailed presentation of this system would go beyond the scope of this paper. For each image, the feature points must be extracted by the SIFT algorithm so that 36 detections are executed. The results are shown in Table III. It can be clearly seen that the necessary computing time and the used energy is greatly reduced by using an accelerator for the SIFT algorithm. This is especially true for the FPGA accelerator. It is also evident that a large part of the used energy is consumed by the host platform, which in the case of the FPGA accelerator is mainly used to provide data.

The detected feature points of an image are compared to all detected points of 10 adjacent images to determine the nearest neighbors. Therefore, 36 images in total require 36*10 = 360 matchings for high quality object detection. The results are shown in Table IV. By using GPU accelerators, the calculation of the matching is accelerated by a factor of 5.8. Due to the minimized run-times, the service technician gets a result promptly. It is also evident that a significant amount of energy is used only for the host platform.

VIII. CONCLUSION AND FUTURE WORK

In this paper we presented a generic HW/SW framework for the seamless integration of FPGA- and GPU-based accelerators into existing data center architecture on an AaaS basis. It has been shown that our hardware framework abstracts the complexity of basic hardware functions like external communication via PCIe. With our virtualization concept based on sockets, different types of accelerators can be operated in parallel with ease. A hardware designer can use the standardized interfaces to design his own accelerators faster. The presented software framework gives easy access to normally hard to use accelerator architectures and is a key enabler for paving the way to a broad application of FPGA- and GPU-based accelerators. In our use case, we show that offloading compute-intensive algorithms is easily accomplished with the proposed approach. We show that, in contrast of running an object classification on a mobile device, which could take several hours, we have achieved a significant reduction down to minutes for the same task. By accelerating object recognition in the cloud, the service technician gets an exact classification of the photographed component within a few minutes, which leads to reduced downtimes.

Our ongoing and future work includes the extension of the presented concept to use FPGA-based network-attached accelerators (NAA) to get rid of compute nodes which are mainly used as PCIe integration systems. This will show a major increase in power efficiency as has been shown by other similar developments. We are aiming to use the NAA architecture as a basis for a machine learning cluster which is accessible as a cloud computing resource. We will develop the NAA with authentication and encryption, taking security aspects into account. We also want to investigate the use of the HW/SW framework on an IaaS like the AWS cloud.

REFERENCES

[1] (2020) Amazon Web Services FPGA. [Online]. Available: https://github.com/aws/aws-fpga

[2] C. Kachris and D. Soudris, "A survey on reconfigurable accelerators for cloud computing," in *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, Aug 2016, pp. 1–10.

[3] A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, J. Gray, M. Haselman, S. Hauck, S. Heil, A. Hormati, J. Kim, S. Lanka, J. Larus, E. Peterson, S. Pope, A. Smith, J. Thong, P. Y. Xiao, and D. Burger, "A reconfigurable fabric for accelerating large-scale datacenter services," *IEEE Micro*, vol. 35, no. 3, pp. 10–22, May 2015.

[4] S. R. Chalamalasetti, K. Lim, M. Wright, A. AuYoung, P. Ranganathan, and M. Margala, "An FPGA memcached appliance," in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, ser. FPGA 13. New York, NY, USA: Association for Computing Machinery, 2013, p. 245254.

[5] Y. Shan, B. Wang, J. Yan, Y. Wang, N. Xu, and H. Yang, "FPMR: Mapreduce framework on FPGA," in *Proceedings of the 18th Annual ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, ser. FPGA 10. New York, NY, USA: Association for Computing Machinery, 2010, p. 93102.

[6] S. A. Fahmy, K. Vipin, and S. Shreejith, "Virtualized FPGA accelerators for efficient cloud computing," in *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*, Nov 2015, pp. 430–435.

[7] S. Byma, J. G. Steffan, H. Bannazadeh, A. L. Garcia, and P. Chow, "FPGAs in the cloud: Booting virtualized hardware accelerators with openstack," in *2014 IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines*, May 2014, pp. 109–116.

[8] ARM Limited, "AMBA AXI and ACE Protocol Specification: AXI3, AXI4, and AXI4-Lite ACE and ACE-Lite," ARM Limited, Cambridge, United Kingdom, Specification D, 2011.

[9] (2020) gRPC. [Online]. Available: https://grpc.io/

[10] K.-T. Seo, H.-S. Hwang, I.-Y. Moon, O.-Y. Kwon, and B.-J. Kim, "Performance comparison analysis of linux container and virtual machine for building cloud," in *Advanced Science and Technology Letters. Vol.66 (Networking and Communication 2014)*, 2014, pp. 105–111.

[11] (2020) Podman. [Online]. Available: https://podman.io/

[12] S. I. F. Silla, J. Prades and C. Reao, "Remote GPU virtualization: Is it useful?" *2016 2nd IEEE International Workshop on High-Performance Interconnection Networks in the Exascale and Big-Data Era (HiPINEB)*, pp. 41–48, 2016.

[13] D. C. Blumenthal-Barby and P. Eisert, "High-resolution depth for binocular image-based modeling," *Computers & Graphics*, vol. 39, pp. 89–100, 2014.

[14] B. Micusik and H. Wildenauer, "Structure from motion with line segments under relaxed endpoint constraints," *International Journal of Computer Vision*, vol. 124, no. 1, pp. 65–79, 2017.

[15] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vision*, vol. 60, no. 2, p. 91110, Nov. 2004.