A novel framework for the Seamless integration of FPGA accelerators with Big Data analytics Frameworks in Heterogeneous data centers

Ioannis Stamelos, Elias Koromilas, Christoforos Kachris and Dimitrios Soudris
Institute of Communication and Computer Systems (ICCS)
National Technical University of Athens
Athens, Greece

***Abstract*—To face the increased network traffic in the cloud, data center operators have started adopting an heterogeneous approach in their infrastructures. Heterogeneous infrastructures, e.g. based on FPGAs, can provide higher performance and better energy-efficiency compared to the contemporary processors. However, FPGAs lack of an easy-to-use framework for the efficient deployment from high-level programming frameworks. In this paper, we present a novel framework that allows the seamless integration of FPGAs from high-level programming languages, like Java and Scala. The proposed approach provides all the required APIs for the utilization of FPGAs from these languages. The proposed scheme has been mapped on Amazon AWS f1 infrastructure and a performance evaluation is presented for two widely used machine learning algorithms.**

***Index Terms*—heterogeneous computing, reconfigurable computing, FPGA, data center, accelerator, machine learning.**

I. Introduction

Emerging applications in cloud computing, IoT and big data analytics have created the need for more powerful data centers that can sustain the huge amounts of the data that are generated by these applications. To face the exponential increase of this data traffic, data centers need to adopt more efficient architectures and solutions that will be able to face the increased traffic without consuming excessive amounts of power. Until recently, data center operators were relying on the scaling of the data centers by the utilization of more commodity server processors as most of the workload can be parallelized efficiently. However, this approach increases significantly the cost and the power consumption of the data center operators.

Therefore, during the last few years, data center operators have started adopting new heterogeneous architectures based on energy-efficient accelerators such as FPGAs and GPUs. Hardware accelerators offer higher performance and better energy efficiency compared to the typical server processors. However, several constraints prevent them from the wide adoption from data center operators and data analytics frameworks, such as the high programming efficiency, the lack of a transparent interface, the low resource-utilization in the infrastructure, as well as the limited virtualization and isolation. Several efforts have been performed to increase the programming efficiency of the accelerators, such as HLS and OpenCL languages. Nevertheless, one of the main barrier for the widespread adoption of the FPGAs in the data centers is the lack of an easy to use interface that will allow the communication with the processor without significant programming effort.

In this work we present a novel scheme that allows the seamless and efficient utilization of hardware accelerators from high-level programming frameworks, like Python, Java and C++ based ones. The proposed scheme provides all the required APIs for the tight integration of the hardware accelerators with such programming frameworks, thus enabling their effortless deployment. The proposed framework has been evaluated in the emerging FPGA-as-a-service (Faas) platforms that allow the utilization of the FPGAs in the cloud. Specifically, the proposed framework has been evaluated in the Amazon AWS f1 platform for two widely used machine learning techniques.

Specifically, the main contributions on this paper are the following:

- An integrated framework for the effortless deployment of FPGAs from high-level programming frameworks.
- Transparent APIs for the utilization of FPGAs from Java, Scala and Python.
- Efficient mapping on Cloud FPGAs based on the Amazon AWS f1 platform.
- Performance evaluation on two widely-used machine learning algorithms.

In Section II, we describe the related work in the domain of programming frameworks for the efficient deployment of FPGAs in heterogeneous data centers. In Section III, we describe the integrated framework for the transparent use of FPGAs from high-level programming frameworks. In Section IV, we describe the efficient mapping on the FPGAs hosted in the Amazon AWS and in Section V we present the performance evaluation of the proposed framework on two widely-used machine learning techniques, that have been implemented and mapped on the Amazon FPGAs.

II. Related Work

The use of heterogeneous systems comes at a significant cost: the increase in programming complexity. To overcome this problem, new programing frameworks, that hide the complexity of the heterogeneous systems without affecting the

overall system performance, must be developed. The programming of the FPGAs can be overcome by using *High Level Languages* such as OpenCL or C [1][2][3][4]. Therefore, in the last years, there are several hardware accelerators that have been proposed that targeting cloud computing applications for data centers [5]. However, there are still several issues that need to be resolved on deploying the FPGAs in the data centers. For example, the main issues are the virtualization and the partitioning of the hardware resources, the configuration of the FPGAs, and the scheduling of the hardware accelerators, based on the applications requirements. In the last couple of years, there are several research efforts towards an efficient framework for the deployment of the FPGAs in the data centers.

A. FPGAs in the Cloud, IBM

In [6], a general framework is proposed for integrating FPGAs into the cloud by IBM. The framework proposes an accelerator pool (AP) that abstracts FPGA as a consumable resource while avoiding hardware dependencies of current FPGA technologies. In the AP abstraction, each FPGA has several pre-defined accelerators slots in which the hardware accelerators can be mapped. By utilizing the partial reconfiguration mechanism of the FPGAs, each slot can be considered a virtual resource that can be assigned for specific tasks. A cloud tenant can submit either pre-defined hardware accelerators that are hosted in central repository or can submit his own designs. However, in the latter case the cloud owner should perform the synthesis, place and route and generate the bitstreams for the FPGA slots.

A prototype of the framework is implemented on an x86-based Linux-KVM environment with attached FPGAs and deployed in a modified OpenStack cloud environment. Four different accelerators are used for prototyping: Encryption (AES), Hashing (SHA), Stereo matching and Matrix-Vector Multiply. The performance evaluation shows that proposed framework allows the efficient utilization of the FPGA resources by the cloud tenants with less than 4 microseconds latency overhead of the visualization.

B. Virtualized Hardware accelerators, University of Toronto

In [7], a novel approach for integrating virtualized FPGA-based hardware resources into cloud computing systems with minimal overhead. The proposed framework allows cloud users to load and utilize hardware accelerators across multiple FPGAs using the same methods as the utilization of Virtual Machines. The reconfigurable resources of the FPGA are offered to the users as a generic cloud resources through OpenStack.

An agent is introduced in this framework that implements the resource management of the OpenStack. The proposed framework splits the FPGA into several reconfigurable regions, each managed as a single resource. Therefore, instead of a single FPGA bitstream, a collection of partial reconfigurable bitstreams corresponding to the user hardware is passed to the agent. Again, as in the case of the IBM, the cloud provider must generate the partial bitstream for each accelerator and for each partially reconfigurable slot since the current technology requires specific bitstreams for each region of the FPGA.

C. FPGAs in Hyperscale Data Centers, IBM Zurich

In [8], a framework is proposed by IBM Zurich that allows cloud users to combine multiple FPGAs in the programmable fabric. This allows cloud operators to offer an FPGA to users in a similar way as a standard server. In the proposed framework multiple user applications can be hosted on a single physical FPGA, somehow similar to multiple VMs running on the same hypervisor. Each user can get a partition of the entire user logic and uses it to implement its own applications. This partitioning is achieved by utilizing partial reconfiguration. With partial reconfiguration it is possible to dynamically reconfigure a portion of the FPGA while the rest of the regions remain untouchable.

The proposed architecture has been integrated into the OpenStack framework and allows the renting of the FPGA resources on the cloud. The same architecture also allows the possibility to distribute their applications on a large number of FPGAs through an FPGA fabric. The integrated framework with multiple FPGAs is compared with a typical data center based on commodity processors. It is shown that if each system is based on 2048 module the FPGA-based system can provide 958 TFLOPS compared with the 442 TFLOPS offered by the commodity processors.

D. RC3E: Reconfigurable Cloud Computing Environment

In [9], a cloud hypervisor is proposed by Technical University of Dresden that integrates virtualized FPGA-based hardware accelerators into the cloud environment. The hypervisor allows users to implement and execute their own hardware designs on virtual FPGAs. The hypervisor has access to a database containing all physical and virtual FPGA devices in the cloud system and their allocation status. Each device is assigned to its physical host system (node). The user can allocate a complete physical FPGA, which has to be marked separately in the device database or can allocate portion of the vFPGA. In the case of vFPGA allocation, the configuration is performed by utilizing partial reconfiguration (PR).

The proposed framework supports the required security by protecting the device files using access rights. This additional virtualization layer allows concurrent users to interact with their allocated devices without influencing each other.

E. Virtualized FPGA accelerators, University of Warwick

In [10], a novel framework is presented that integrates reconfigurable accelerators in a standard server with virtualized resource management and communication. The proposed framework integrates a PCIe based FPGA board into a standard data center server. The FPGA is partitioned into separate accelerator slots. Accelerator functions are either stored in a library on the host machine as partial bitstreams or can be uploaded by the user.

In this framework, a hypervisor is implemented for the configuration and the scheduling of the user logic in the FPGA

resources. When an accelerator is to be configured in the FPGA, the hypervisor decides on the optimal partial reconfiguration regions (PRRs) to host it and initiates reconfiguration. The hypervisor also maintains a list of the available PRRs and configured accelerators to avoid unnecessary reconfiguration when a required accelerator is already present in the FPGA and not in use.

III. Transparent Utilization of Accelerators

For the transparent utilization of hardware accelerators, we had to create a new framework that would embed all the functionality of the old ones, serving high-level APIs in Python, Java, Scala etc., while supporting tight communication with the FPGA and that would be able to handle requests for any FPGA resources, or transfers data between the host and the FPGA memory. For evaluation purposes, the proposed software stack was used to build an accelerated machine learning library, that allows the seamless utilization of the available hardware resources. In the following paragraphs, we are going to explain the aspects of our approach based on that use case, however any type of acceleration scenarios could be supported.

Figure 1 shows the layers of the controllers developed in VINEYARD for the efficient communication between the FPGAs and the programming frameworks. The controllers that we developed support both the Xilinx FPGA platforms (Kintex KU3, Virtex VU9P) and the Intel (Xeon+QPI+FPGA) HARP platforms. The latter platforms, use an extended OpenCL API for the communication with the FPGA. Xilinx platforms provide communication via PCIe lanes, while Intel platform offers low-latency Shared Virtual Memory between the CPU and the FPGA. That diversity led us into creating a new unified software stack tailored to our needs. The core of this stack, the FPGA driver API, is packed in a shared object library and can be used in a transparent way, hiding all the low level details. What is more, we implemented top level APIs in C/C++, Java, Python and Scala that offer ease of use, while are also easily maintained, since the middle layer, our dynamic library, remains the same for all of the above. In other words, we implemented a 3-tier style software stack. The top level hosts the users' applications, the middle layer hosts our libraries and the lower layer hosts the OpenCL API, which is used to actually invoke the accelerators. This 3-tier scheme has a lot of advantages, in which we will go through in more detail at the next paragraphs. It is important to note here that a similar structure is used both for the Intel and the Xilinx platforms, as is also depicted in Figure 1.

A. Application Layer

This layer hosts users' applications. The applications can be executed natively using C/C++, using Java or Python. Users are able to perform a plethora of methods (i.e. train(), test(), load() etc.) on their machine learning models. Users that already have their applications running, do not need to change a single line of code, except from the imported or included library. Apart from that, changes in the lower layers of our stack won't affect this one. This way we are able to make changes, optimize and add stuff or functionality to our libraries and drivers, without affecting any top-level applications.

B. Acceleration Layer

This layer hosts the whole functionality of our framework. The key elements of this layer are the implemented shared library (**libVine.so**) and the Java/Scala library (**libVine.jar**). The jar file serves as the Java/Scala API, containing all the classes for writing an application in Java or Scala. It hosts the implementation of each machine learning model in .java and .scala files. Furthermore, it hosts all the functions and methods needed to communicate with C/C++ through the Java Native Interface (JNI), which is used as an intermediate step for invoking an accelerator. Any JNI request from the Java environment is handled from the shared library. It is noteworthy that to speed up Python applications, all the implemented classes invoke the Java virtual environment using the *pyjnius* package. In other words, Python applications use the same Java API mentioned above for the whole execution of any class function.

The shared library combines a lot of functionality. It hosts the C/C++ API for any top-level applications, the JNI functions executing native C/C++ code and the FPGA driver that is used to communicate with the OpenCL API.

The C/C++ API offers the same methods and functions the Java/Scala API does. Any machine learning models can be trained in the exact same way.

The VineNI (Vineyard Native Interface) implements three different basic functions:

- **acceleratorInit**(),
- **acceleratorEnd**() and
- **acceleratorRun**()

Each one invokes the FPGA driver to perform the corresponding task. In more detail, *acceleratorInit* is used to initialize the FPGA board, to download the bitstream and allocate any buffers, *acceleratorEnd* is used to free any allocated memory and *acceleratorRun* is used to transfer the data to and from the FPGA board along with performing any necessary computations. For the Intel platform two extra calls have been implemented, *mallocSVM* and *freeSVM*, that allow the easy manipulation of the shared memory.

The FPGA driver is consisted of two parts: A library that wraps the OpenCL API and is used to invoke the OpenCL API in every implemented kernel, and the driver for each kernel.

- The OpenCL Wrappers & Helper Functions library supports error handling but also can lead to $\tilde{8}0\%$ reduction of each accelerator driver, in terms of code lines. It is packed as a unified library for both Vendors, and someone can use it in both platforms only by defining the Vendor name (**#define XILINX** or **#define ALTERA**).
- Each driver is kernel specific, meaning that any input arguments and procedures are tightly intertwined to the accelerator's functionality. For example, an accelerator may have 1 input and 1 output argument while another

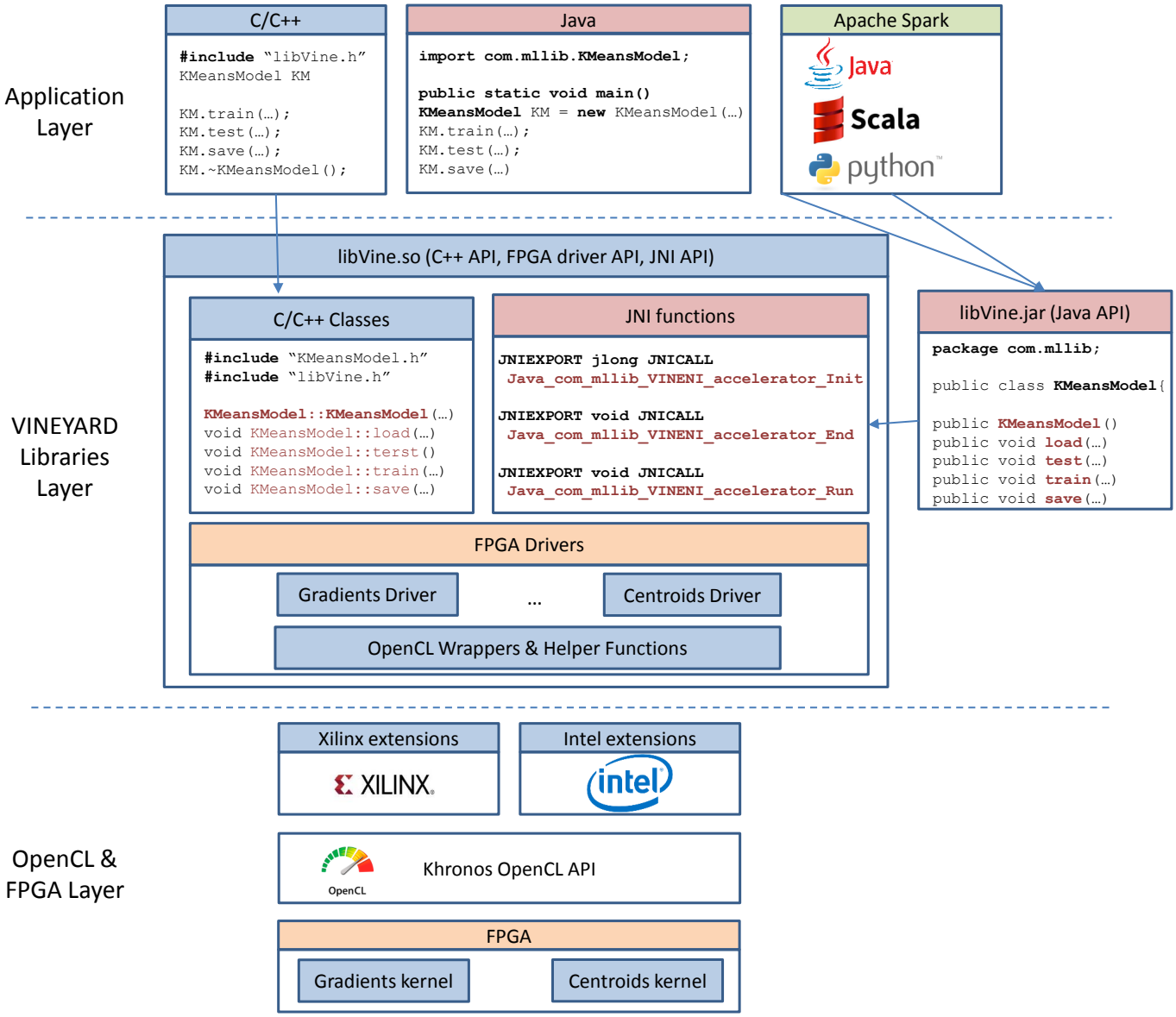

Fig. 1. Software stack for the Seamless Integration of FPGA accelerators.

may have a lot more or even different types of arguments etc.

C. OpenCL - FPGA layer

The bottom layer, that serves as the FPGA runtime, is consisted of the OpenCL library that is provided by each Vendor and contains Vendor-specific extensions, complementary to Khronos OpenCL function calls. This layer has remained unchanged as it hosts the primitive elements for communicating with the FPGA board through the PCIe or the QPI interfaces.

IV. Hardware Accelerators - Use Cases

To begin with, Amazon Web Services was the first major cloud service provider that started deploying FPGAs in its cloud platform. Amazon EC2 f1 instances are compute instances that contain Xilinx FPGA boards, connected on PCIe slots, allowing users to program them and accelerate their applications. Amazon has built a formidable position in Infrastructure-as-a-Service (IaaS) with Amazon Web Services Elastic Compute Cloud (AWS EC2), and it is hoping to extend that position to applications that benefit from acceleration using GPUs and FPGAs. FPGAs are relatively difficult to program, and the cloud leader is laying the foundation to simplify FPGA adoption by creating a marketplace for accelerated applications built on Xilinx FPGAs. Other data center operators Baidu and Tencent also have adopted Xilinx FPGAs as a service this year. AWS is relying on OpenCL FPGA programming to reach more developers in future, although this still requires a lot of expertise and isnt necessarily a

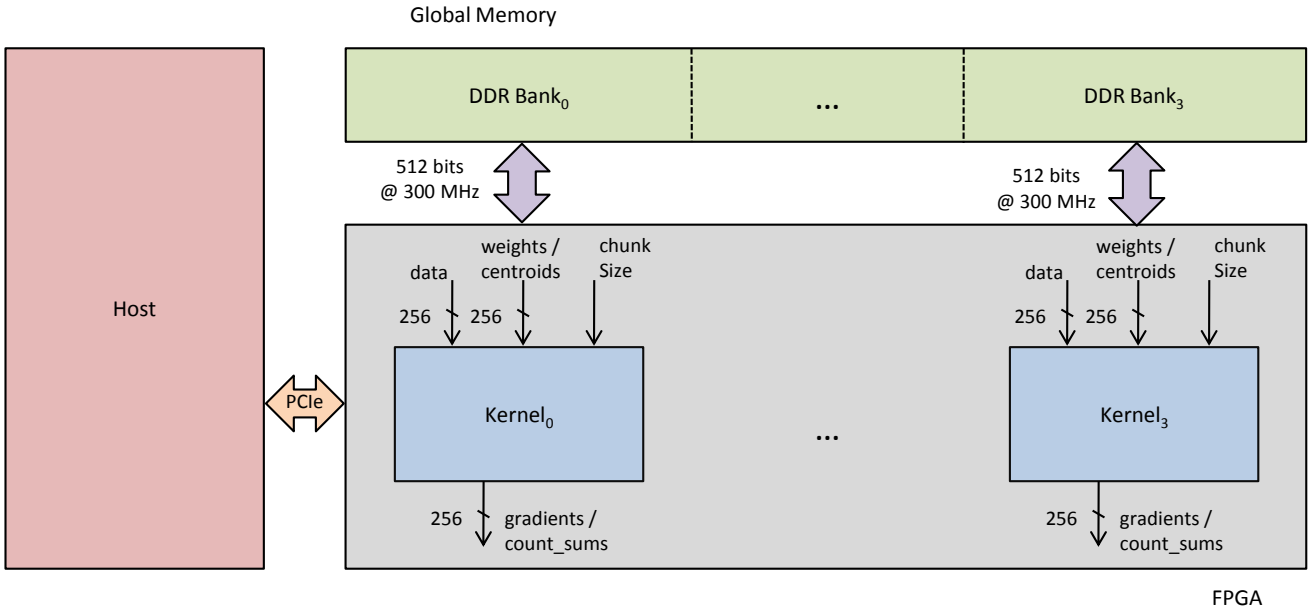

Fig. 2. Amazon f1 Memory Model and Accelerators' Architecture.

good match for the FPGA programming model. However, there are several issues that prevent the widespread adoption of the FPGAs-as-a-service concept in the cloud computing community. Therefore, a library-based design flow is required that will allow the wide adoption of hardware accelerators for data analytics applications.

The machine learning algorithms that we chose to accelerate on Amazon f1, through the proposed scheme, are Logistic Regression and KMeans. For both of them, a proof of concept implementation based on Zynq (ARM+FPGA) architecture [11], allowed us to focus on the migration of specific kernels to this real heterogeneous cloud architecture, as well as explore more efficiently the available design space and modify the previously implemented kernels to meet the new optimal performance.

A. Accelerator Features

This paragraph describes the final accelerators design for maximizing performance and throughput. At a kernel level, we went through all the possible optimizations. At first, we put a lot of effort into unrolling and pipelining most of the loops. By reordering the loops and reading batches of the input dataset, we finally managed to break any loop carried dependencies and achieve an initiation interval of value 1 for every pipelined loop. Further on, we optimized the kernel to global memory data transfers by taking advantage of the 512bits bandwidth provided by the Amazon f1 FPGA boards. In our case, both accelerators have two input M_AXI interfaces (*data* and *weights* for gradients kernel and *data* and *centroids* for the centroids one), one output M_AXI interface (*gradients*, *count_sums* for gradients and centroids kernel respectively) and an AXI lite input port (*chunkSize*) for setting the number of given data. After extensive profiling and trying to balance the factor of availably used FPGA resources and performance gains, we finally set the bandwidth of every M_AXI interface to 256bits. This way every M_AXI interface is able to transfer at the same time eight (8) floating point numbers rather than one (1). In addition, to maximize throughput we implemented four kernels of the same IP in every accelerator. This was found to be a tough procedure, since to figure out the optimal kernel number we had to experiment with many different combinations on this number and the number of the compute-units. Implementing four kernels, we were able to almost fully utilize all of the available FPGA resources and take advantage of the four (4) DDR banks provided. Each one of the four kernels is assigned a different DDR bank and sends or receives any data through it. The whole architecture of the implemented accelerators is shown in Figure 2.

TABLE I
FPGA RESOURCE UTILIZATION (%).

Amazon FPGA Image	LUT	LUTMem	REG	BRAM	DSP
Logistic Regression	28.15	2.17	16.20	57.71	26.36
KMeans	27.84	2.03	20.38	76.28	46.51

Finally, as far as the specifications of each accelerator are concerned, the gradients accelerator supports up to 10 classes and 784 features while the centroids one supports up to 14 clusters and 784 features.

V. PERFORMANCE EVALUATION ON AMAZON AWS

For the performance evaluation of the proposed framework, we used the exact same algorithm implementations for the software and hardware executions. For the software execution

we used an Amazon c4.8xlarge instance which has the same price tag (per hour) as f1.2xlarge does (∼1.6/h). Every Amazon c4.8xlarge instance comes with 36 vCPUs and 60 GiB of RAM while the f1.2xlarge instances host 8 vCPUs and 122GiB of RAM.

An F1 vCPU is a single thread of an Intel Xeon E5-2686 v4 @ 2.3 GHz processor, while a C4 vCPU stands for a single thread of an Intel Xeon E5-2666 v3 @ 2.9 GHz processor. To compare the accelerated and software-only implementations, we created two different execution scenarios.

A. Native Execution

The first one concerns a native execution of the same application in software and hardware. The aim was to compare a simple application implementation to the accelerated one, without using OpenMP or any other libraries for parallel execution. Therefore, the software execution ran on a single CPU thread, ending up in a 1 CPU thread to 1 FPGA accelerator comparison. This way, the rest of the c4s CPU threads (35) remained idle, while on the f1 all of the eight threads remained idle since the CPU intensive parts were executed on the FPGA board. Figure 3 depicts the kernel execution time for both Logistic Regression and KMeans algorithms in software (written in C++, Java and Scala) normalized to the hardware execution using the Amazon vu9p FPGA. Results show a 19.6x speedup compared to the software execution for the Logistic Regression algorithm and a 21.35x speedup for the KMeans algorithm.

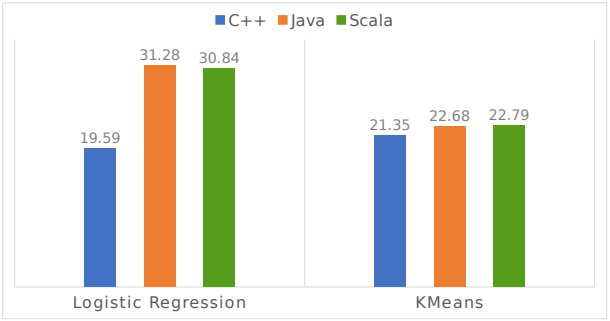

Fig. 3. Native Execution Speedup.

B. Spark Standalone Execution

To take advantage of all the c4s CPU threads, we ran the aforementioned algorithms on a Spark Standalone cluster. For the software execution we again used a c4 Amazon instance. Spark framework was launched with 36 workers, therefore using all the available CPU threads. The memory of each worker was set accordingly in order to also use all of the available DRAM memory. As far as the f1 instance is concerned, a Spark cluster of 8 workers was launched dividing again the amount of the available memory to each worker. OpenCL, which is used for the invocation of the hardware accelerator, does not support shared memory allocation for the context, program or any other OpenCL-created structures. This restriction do not allow us to access the context from every worker, since each one has his own virtual memory space. To surpass this problem, we initially perform any data (RDD) transformations partitioning the input dataset across all the available workers. Further on, we coalesce the transformed data to one partition and only one of the available workers takes over the task of communicating with the FPGA board to download the bitstream, send or receive data and free any allocated memory. At this point there is the trade-of between underusing the available f1 resources and being able to accelerate our applications. Figure 4 depicts the execution times of Logistic Regression and KMeans algorithms in software and hardware. Results prove a 2.3x speedup compared to the hardware execution for the Logistic Regression and a 2.5x lower execution time for the KMeans accelerated implementation.

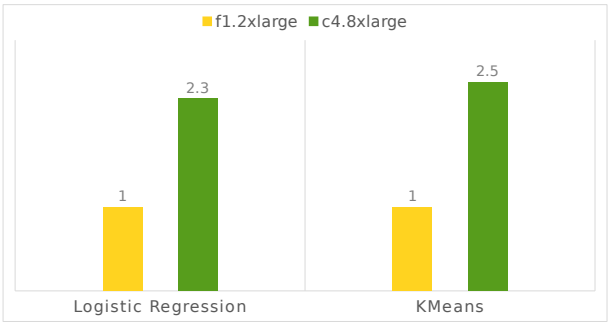

Fig. 4. Spark Standalone Speedup.

VI. Conclusion

Hardware accelerators can improve significantly the performance of machine learning applications. However, currently data analytics frameworks like Spark do not support the transparent utilization of such acceleration modules. In this study, we demonstrate a novel scheme for the seamless utilization of hardware accelerators in high level programming frameworks, like Spark, in Amazon AWS.

For the evaluation we have implemented two hardware accelerators, one for Logistic Regression and one for KMeans, and we have efficiently integrated it with Spark, while we have also packed them in a native library. The results show that the proposed scheme can be used in high performance systems to reduce up to 2.5x the execution time as well as the Amazon infrastructure operating cost. The results also show that the proposed framework can be utilized to support any kind of hardware accelerators in order to speedup the execution time of computational intensive machine learning applications, and prove that hardware acceleration and thus SW/HW co-design is in fact a valid solution when software acceleration techniques meet their limits.

As Future Work, in order to solve the OpenCL shared memory restriction and to be able to efficiently manipulate the available FPGA resources, an FPGA Manager capable of handling accelerator requests in a Client-Server scheme

could be implemented. This architecture would also give us the ability to perform also better processing and scheduling of such requests, improving in that way both the make-span and the throughput of the system.

ACKNOWLEDGMENT

The authors are sincerely grateful to the anonymous reviewers for their valuable comments. This work was supported by the VINEYARD project, which has received funding from the European Unions Horizon 2020 research and innovation programme under grant agreement No 687628 - VINEYARD: Versatile Integrated Heterogeneous Accelerator-based Data Centers.

REFERENCES

[1] S. Windh, X. Ma, R. J. Halstead, P. Budhkar, Z. Luna, O. Hussaini, and W. A. Najjar. High-level language tools for reconfigurable computing. *Proceedings of the IEEE*, 103(3):390–408, March 2015.

[2] David Bacon, Rodric Rabbah, and Sunil Shukla. Fpga programming for the masses. *Queue*, 11(2):40:40–40:52, February 2013.

[3] Oren Segal, Philip Colangelo, Nasibeh Nasiri, Zhuo Qian, and Martin Margala. Sparkcl: A unified programming framework for accelerators on heterogeneous clusters. *CoRR*, abs/1505.01120, 2015.

[4] O. Segal, M. Margala, S. R. Chalamalasetti, and M. Wright. High level programming framework for fpgas in the data center. In *Field Programmable Logic and Applications (FPL), 2014 24th International Conference on*, pages 1–4, Sept 2014.

[5] Christoforos Kachris and Dimitrios Soudris. A survey on reconfigurable accelerators for cloud computing. In *Field Programmable Logic and Applications (FPL), 2016 26th International Conference on*, pages 1–10. IEEE, 2016.

[6] Fei Chen, Yi Shan, Yu Zhang, Yu Wang, Hubertus Franke, Xiaotao Chang, and Kun Wang. Enabling fpgas in the cloud. In *Proceedings of the 11th ACM Conference on Computing Frontiers*, CF '14, pages 3:1–3:10, New York, NY, USA, 2014. ACM.

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

[8] Jagath Weerasinghe, Francois Abel, Christoph Hagleitner, and Andreas Herkersdorf. Enabling fpgas in hyperscale data centers. In *2015 IEEE International Conference on Cloud and Big Data Computing (CBDCom 2015)*, May 2015.

[9] Oliver Knodel and Rainer G. Spallek. RC3E: provision and management of reconfigurable hardware accelerators in a cloud environment. In *2nd International Workshop on FPGAs for Software Programmers*, 2015.

[10] Suhaib A. Fahmy, Kizheppatt Vipin, and Shanker Shreejith. Virtualized FPGA accelerators for efficient cloud computing. In *7th IEEE International Conference on Cloud Computing Technology and Science, CloudCom 2015, Vancouver, BC, Canada, November 30 - Dec. 3, 2015*, pages 430–435, 2015.

[11] C. Kachris, E. Koromilas, I. Stamelos, and D. Soudris. Spynq: Acceleration of machine learning applications over spark on pynq. In *2017 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS)*, July 2017.