VineTalk: Simplifying Software Access and Sharing of FPGAs in Datacenters.

Stelios Mavridis, Manolis Pavlidakis, Christi Symeonidou, Christos Kozanitis, Nikolaos Chrysos, and Angelos Bilas
Institute of Computer Science (ICS)
Foundation for Research and Technology – Hellas (FORTH)

Ioannis Stamoulias, Christoforos Kachris, and Dimitrios Soudris
Institute of Communication and Computer Systems (ICCS)
National Technical University of Athens (NTUA)

***Abstract*—FPGA-based accelerators are becoming a first class citizen in data centers. Adding FPGAs in data centers can lead to higher compute densities with improved energy efficiency for latency critical workloads, such as financial applications. However deployment of FPGAs in datacenters is hindered, as both developers and cloud providers face difficulties. Application writers need to deal with FPGA interfacing as well as application logic/algorithms. On the other hand, cloud providers are reluctant to deploy FPGAs in large scale, due to the FPGAs lack of sharing that results in reduced utilization and questionable ROI. In this paper, we introduce *VineTalk*, a framework that reduces the programming effort associated with FPGA-based accelerators and FPGA virtualization. We integrate *VineTalk* with the Xilinx SDAccel development framework and we map it to the Kintex UltraScale FPGA. Our preliminary evaluation with a use-case of financial applications shows that *VineTalk* can offer effective FPGA sharing introducing less than 4% overhead to application execution time.**

***Index Terms*—hardware accelerator, virtualization, cloud computing, FPGAs, data centers.**

I. Introduction

The cost-effectiveness of commodity hardware has led to horizontal scaling of modern datacenter applications: Whenever an application challenges the limits of a single server, one can simply get more performance by using more servers in parallel. As a result, today most large scale datacenter applications are hosted by scale-out deployments of commodity servers, either in private datacenters or in public clouds. However, recent trends, both in application requirements and technology, challenge the sustainability of this scale-out approach. First, applications are becoming more computationally intensive. There is an increasing use of compute intensive workloads, such as machine learning and deep learning algorithms where matrix multiplications dominate [1]. Second, with the current projections for unprecedented data growth, applications will need to continue on the scale-out path. As a result, CPU processing is becoming a main limitation in datacenters [2].

Recently, a response to these trends has been the use of application-specific accelerators in datacenters. The use of such acceleration units can increase performance and efficiency of datacenters significantly, while maintaining power and hardware costs under control. In the last couple of years there have been several reconfigurable architectures proposed for the acceleration of cloud computing applications in datacenters. However, there are three main challenges that limit the use of FPGA resources in the cloud.

The first challenge is the high programming effort. To address this issue in hardware design, FPGA vendors have already modernized hardware design procedures by replacing traditional HDL languages, such as VHDL and Verilog, with higher level languages, such as OpenCL and C++. Despite this progress however, hardware design still remains a challenge for software engineers.The second main challenge is the interfacing of FPGA resources to software applications. For each application, typically there is a complex and platform specific communication layer that needs to be considered carefully to transfer data to the FPGA, allocate and release memory, specify which routine to run, and get the results back. The third challenge is the lack of virtualization mechanisms in FPGAs, which allows operators to achieve better utilization of their hardware. However, virtualization of FPGA resources presents two main challenges. First, it requires dynamic reconfiguration, which is hard to achieve transparently. Second, even when applications use the same configuration on a FPGA, it is a challenge to coordinate the shared access to the FPGA.

In this paper we present *VineTalk*, a software layer between FPGAs and applications that reduces the complexity of communication between application software and accelerator hardware, and allows sharing of FPGA across applications. Furthermore, *VineTalk* allows applications to transparently access FPGA accelerators, regardless of whether they run natively on a server, within virtual machines, or containers. In this work we focus on FPGA sharing among different applications that use the same configuration, and we leave as a future work the complete virtualization of FPGAs with dynamic reconfiguration on top of our device sharing infrastructure. Our system consists of two major components: a Communication Layer that implements *VineAccelerator*s and *VineBuffer*s, and a Software Controller that runs as a user-level process and controls/schedules all accesses to the accelerator.

The main contributions of this paper are the following:

1) A software interface, which exposes FPGA accelerators as task queues to applications.
2) A hardware interface, which simplifies the addition of new kernels by hardware developers.
3) A Software Controller that facilitates sharing of acceleration resources.
4) An integration with the SDAccel framework of Xilinx.

We implement *VineTalk* for Linux servers using 4700 lines of

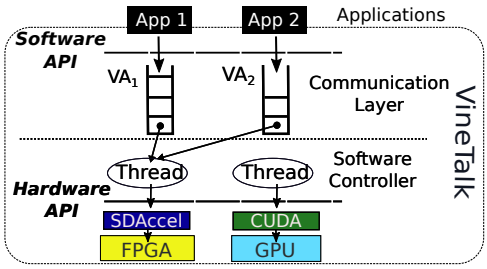

Fig. 1. Design overview of *VineTalk*. VA represent *VineAccelerator*

C code. We demonstrate *VineTalk*'s FPGA abstraction capabilities with a financial application running on a Xilinx ADM-PCIE-KU3 FPGA device. The application uses a hardware implementation of the *Black&Scholes* algorithm, and accesses the FPGA using the Xilinx SDAccel development environment. The simplicity of the *VineTalk* API reduces applications' code complexity by 30% in terms of lines of code. Our results show that applications that user *VineTalk* to access the SDACCel framework have a performance overhead from 0.9% up to 4% compared to their native execution. Moreover, for applications that share the same accelerator, the overhead is negligible.

II. Related work

The use of heterogeneous systems comes at a significant cost: the increase in programming complexity at different levels. To overcome issues related to programming the FPGA itself, developers can employ *High Level Languages,* such as OpenCL or System C [3], [4]. However, little has been done to reduce the effort in incorporating the use of FPGAs in applications and services. Recent research has examined various techniques to partition FPGAs so they can be used by multiple applications. Our work is orthogonal to these efforts since our goal is to allow applications to share simultaneously each partition. In [5], a novel framework is presented that integrates reconfigurable accelerators in a standard server with virtualized resource management and communication. Unlike their approach, *VineTalk* places the accelerator controller in the host, completely avoiding dependencies with a hypervisor. In [6], a runtime system has been proposed to simplify the FPGA application development process by providing a high-level API and a simple execution model that supports both software and hardware execution. *VineTalk* uses a host core for scheduling tasks/accelerators, which leaves all FPGA/accelerator resources available to applications.

III. The *VineTalk* Framework

Figure 1 presents the design of *VineTalk* for a single server setup. Our design consists of a Software-facing API(Section III-A), Hardware-facing API(Section III-D), a communication layer (Section III-B) based on shared memory and a Software Controller (Section III-C).

A. Software Facing API

Our Software-facing API replaces the multitude of all platform-specific acceleration APIs, all of which provide functions that handle memory management and data and task transfers between applications and hardware accelerators. The implementation of the API is completely decoupled from accelerator details. *VineTalk* achieves this by using three main abstractions: *VineAccelerator*s, *VineTask*s, and *VineBuffer*s.

A *VineAccelerator* is a virtual accelerator that can execute a specific kernel. When a *VineAccelerator* is created by an application, the application specifies the kernel that it needs to execute. Then, the system identifies the physical accelerators that can execute the required kernel and maps the *VineAccelerator* to one or more physical accelerators. *VineAccelerator*s can be shared across threads but are private to applications. Each application can allocate as many *VineAccelerator*s as it desires. *VineTalk* uses a repository of kernels to instantiate them on an FPGA at system initialization. The FPGA may support multiple partitions. Although *VineTalk* can remove unused kernels from the FPGA and instantiate new kernels as requested by applications, we do not explore this further. After a *VineAccelerator* has been allocated, an application can issue *VineTask*s. *VineTask*s are not statically mapped to a physical accelerator. Consequently, a single physical accelerator can achieve higher utilization by executing *VineTask*s from multiple *VineAccelerator*s. *VineBuffer*s are used to handle transparently the transfer of the data between an app's address space to the physical accelerator. *VineTalk* transparently moves data between application memory and physical accelerators.

The Software-facing API enables applications to access these abstractions through a set of methods as in Table I.

B. Communication Layer

VineTalk's communication layer implements and manage *VineAccelerator*s and *VineBuffer*s. With *VineTalk*, applications run as separate processes (or VMs) from the Software Controller. Therefore, *VineBuffer*s and *VineTask*s need to be transported across address spaces within the server. To achieve this, we use a shared memory-based transport. *VineTalk* uses shared memory to store all *VineTask*s and *VineBuffer*s. The advantage of shared memory within a server is that after the setup phase there is no need to use system calls. Our shared memory approach currently introduces two additional copies to the shared memory segment, when sending/receiving data to/from the accelerator memory. This transport approach relies on shared segments that can be mapped across native processes, containers, and VMs.

C. Software Controller

The Software Controller is a process that controls all accesses to the underlying hardware. It monitors *VineAccelerator*s for issued *VineTask*s and utilizes *VineBuffer*s to retrieve the inputs and store the outputs. Moreover, it implements the accelerator sharing, since it offloads multiple *VineAccelerator*s to the same accelerator. The Software Controller assigns a unix thread (i.e. accelerator thread) to each physical accelerator existing in the system. This accelerator thread, firstly selects the *VineAccelerator* that is going to serve, based on a scheduling policy (currently round-robin). Secondly, it pops the first *VineTask*from the selected *VineAccelerator*, and executes it to

TABLE I
MAIN METHODS OF THE SOFTWARE-FACING API.

Method	Description
vine_kernel_get()	This is a management call that (re) configures a partition of an FPGA with a kernel from a repository.
vine_va_get()	Allocates a *VineAccelerator* that is capable of executing a specific kernel.
vine_buffer_init()	Creates a *VineBuffer* which describes the input and output data for the kernel in the app's address space.
vine_task_issue()	Invokes a kernel to a *VineAccelerator*, using one or more *VineBuffer*s.
vine_task_wait()	Waits for a task to complete. After completion, *VineBuffer*s are updated with computation results.

its physical accelerator. Then, it copies the result to the share memory segment, and serves the next *VineAccelerator*.

D. Hardware Facing API

VineTalk allows hardware designers to incorporate new kernels by using two main functions: *VT2Accel()* and *Accel2VT()*. Additionally, *VineTalk* already provides a number of different implementations of this simple API to cover kernels for different accelerators, including FPGAs and GPUs. For FPGA devices, *VineTalk* implements this API in OpenCL and SDAccel, whereas for GPU devices *VineTalk* implements this API in OpenCL and CUDA. Porting applications to *VineTalk* consists of two steps; First, to create a *VineTalk* library for each kernel and for each *VineTalk* library to create a function that contains the kernel invocation. Second, the modification of the application to replace all accelerator related functions with the corresponding methods from the Software-facing API. *VT2Accel()* prepares the input data for an accelerator kernel. A hardware designer is expected to provide this method for each new kernel. The function allocates input *VineBuffer*s on the accelerator memory and copies the contents of *VineBuffer*s of the communication layer. The method will then be used prior to kernel execution by the host controller. Similarly, *Accel2VT()* is called once after the kernel execution finishes. Its goal is to send the output back to *VineBuffer*s and to releases the reserved acceleration memory.

IV. INTEGRATION WITH SDACCEL

Xilinx has recently released the SDAccel framework, which is a development environment for OpenCL applications that targets Xilinx FPGA-based accelerator cards. It provides an interface between software applications and FPGA devices. The application consists of a host program written in C/C++ and one or more accelerated kernels written in C, C++, or the OpenCL language that run on the underlying FPGA board.

VineTalk intervenes between the application software side and the hardware side of SDAccel and simplifies the development of applications that use FPGA accelerators as well as incorporating new FPGA kernels to applications. The SDAccel specific implementation of the Hardware-facing API (see Section III-D) allows any SDAccel kernel to be used with applications using *VineTalk*, with no hardware dependencies.

To evaluate the coding effort benefits of *VineTalk*, we port three financial applications, *Black&Scholes*, *Black-76* and *Binomial*. We use SDAccel to build three hardware accelerated variations of the aforementioned algorithms. We also write and evaluate a simple application, which interfaces with those kernels, submits tasks and data, and reports the results.

In order to port a SDAccel application we create a *VineTalk* library for each kernel. For each library we use the Hardware-facing API to simplify the kernel invocation. Moreover in the application side, we replace all SDAccel specific functions with the corresponding methods from *VineTalk*'s Software-facing API. The resulting application consists of 30% fewer lines of code and it uses semantically much simpler routines.

V. PERFORMANCE EVALUATION

A. Experimental Setup

For our experiments, we use one Intel(R) Core(TM) i5-4590 machine running at 3.3GHz, with 16 GBytes of DRAM, and one ADM-PCIE-KU3 FPGA Alpha Data board, with 16 GB DDR3, connected to PCI Express Gen3 x8. The system runs CentOS 7 with SDAccel version 2016.4. In our evaluation we

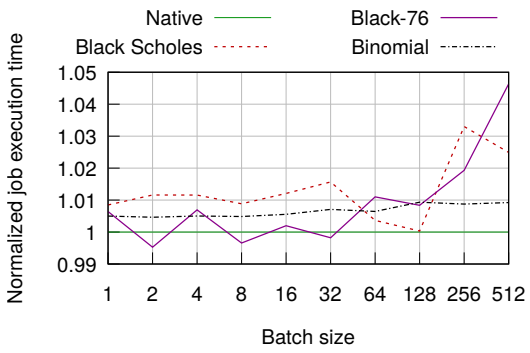

Fig. 2. Performance comparison between *VineTalk*-integrated applications, and their standalone execution over SDAccel. The x-axis is the stock batch size, the y-axis is the normalized job execution time in msec.

use three finacial kernels: *Black&Scholes*, *Black-76*, and *Binomial*. *Black&Scholes* gives a theoretical estimate of the price of European-style options and can also be used for American-style call options. *Black-76* is a variant of the *Black&Scholes* model. *Binomial* option pricing quantizes time and price of an underlying asset, and maps both to a binary tree. We perform each experiment with 2000 options and with varying batch size between 1 and 512. The batch size represents the number of consecutive stocks that are transfered from the application space to the FPGA's memory in a single transfer. We exclude from our results the FPGA reconfiguration overhead, which amounts to 6.15 sec.

Black&Scholes and *Black-76* have four inputs and one output per stock, and with a batch size of 1, the input size of a batch is 16 bytes (4 x 4 bytes) and the output size is 4 bytes. While, *Binomial* uses 5 inputs and 1 output, and for batch size of 1, the input size of each batch is 20 bytes (5 x 4 bytes) and the output size is 4 bytes.

B. VineTalk *overhead*

We compare the execution of the aforementioned applications with *VineTalk* versus the standalone SDAccel execution (Native), to identify *VineTalk*'s overhead. Figure 2 presents the normalized job execution time in milliseconds. Job execution times are averages over 20 experiments, after removing the minimum and maximum values. *VineTalk* adds negligible overheads for small batch sizes while it adds a slight penalty for larger ones as shown in Figure 2. For *Black&Scholes* and *Black-76*, the overhead added from *VineTalk* is between 0.5% and 4% when the batch size is 512, while for batch sizes between 1 and 32 the overhead is negligible. *Binomial* has an overhead between 0.45% and 0.9% for all batch sizes. In all cases, the main source of the overhead are the two additional data copies (inputs and outputs) required by *VineTalk* in the shared memory segment. However, although the overhead of those transfers is constant for most experiments, as they transfer similar amounts of data in aggregate, the impact to the execution of each experiment varies. Runs with larger batch sizes take significantly (by up to two orders of magnitude) less time to execute, and thus, they become more sensitive to the overhead of memory transfers. Table II demonstrates this difference in the execution time for various batch size. The table summarizes the overall application execution time for two batch sizes, 1 and 512, for Native and *VineTalk*. As the batch size increases, the application performance increases significantly for both systems. *VineTalk* incurs the lowest overhead with the *Binomial* application because it has longer task execution times (and thus lower communication to computation ratio) when compared to the other kernels. *Black&Scholes* and *Black-76* are less compute intensive than the *Binomial* kernel. Which results to greater transfer to compute ratio of *Black&Scholes* and *Black-76* than the *Binomial*. Consequently, the extra copies have a stronger effect on the execution time of *VineTalk*.

TABLE II
OVERALL APPLICATION EXECUTION TIME (SECONDS) AND OVERHEAD (%) WITH 2000 STOCKS AND BATCH SIZES 1 AND 512.

	VineTalk (sec)		Native (sec)		Overhead (%)	
Benchmark	**Batch 1**	**Batch 512**	**Batch 1**	**Batch 512**	**Batch 1**	**Batch 512**
Black&Scholes	0.39	0.00123	0.38	0.00120	2.56	2.43
Black-76	0.808	0.00187	0.0803	0.00179	0.618	4.27
Binomial	256	0.514	254.7	0.509	0.50	0.97

C. *Accelerator sharing*

TABLE III
COMPARE THE JOB EXECUTION TIME OF 1 AND 2 CONCURRENT *VineTalk* APPLICATION(S) WITH APPLICATIONS RUNNING DIRECTLY ON THE FPGA (I.E. NATIVE).

	Black&Scholes			Black - 76			Binomial		
Job number	**Native (ms)**	**VineTalk (ms)**	**Ratio**	**Native (ms)**	**VineTalk (ms)**	**Ratio**	**Native (ms)**	**VineTalk (ms)**	**Ratio**
1	9.912	9.88	0.99	18.849	18.47	0.98	5128	5176	1.009
2	9.952	9.68	0.97	18.935	18.39	0.97	5113	5218	1.02

To evaluate the impact of accelerator sharing, we run concurrently up to two instances (jobs) of each application. Limitations of the current testbed, and more specifically the number of cores, does not allow us to run more concurrent instances. Concurrent job execution is possible only with *VineTalk*, which can interleave tasks belonging to different applications. For Native, we execute the 2 jobs sequentially, one after the other. In each experiment, all application instances invoke the same kernel. Each run (i.e. the sum of one or two applications) consists of 2000 stocks and batch size 50. In the first run (with one job), the job consists of 2000 stocks, whereas in the second run the two concurrent jobs consists of 1000 stocks. Table III compares the total serialized execution time (i.e. Native) with *VineTalk*-powered total execution time. We also present the Native's to *VineTalk* total execution time ratio.

For all applications the execution time ratio is very close to 1, consequently the overhead added from *VineTalk* is negligible. For Black & Scholes and Black-76, the *VineTalk* to Native job execution time ratio, for 2 concurrent jobs is 0.97, and 1.02 for the Binomial application. Thus multiplexing applications with *VineTalk* does not introduce overheads.

VI. CONCLUSIONS

In this paper we examine how FPGAs can be used transparently in datacenter servers. In particular, we examine how FPGAs can be shared by multiple applications. We design and implement *VineTalk*, a system that provides a hardware-agnostic abstraction and in fact, is designed to be used with different accelerators, including FPGAs and GPUs. *VineTalk* uses an RPC-like API and a communication channel based on shared memory to allow low-overhead, shared access from applications to accelerators. Our approach is orthogonal to FPGA partitioning and can allow multiple applications to share each partition in an FPGA. Our results show that *VineTalk* reduces both programmer effort at the application level by reducing lines of code related to kernel invocation by about 30% with significantly simpler semantics and introduces overhead between 0.9% and 4% compared to native application execution over the FPGA. Finally, *VineTalk* provides the ability of accelerator sharing from consolidated applications, with less than 2% overhead.

VII. ACKNOWLEDGMENTS

This work has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 687628[1].

REFERENCES

[1] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. E. Jerger, and A. Moshovos, "Cnvlutin: Ineffectual-neuron-free deep neural network computing," in *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*, 2016.

[2] F. Kruger, "Cpu bandwidth - the worrisome 2020 trend," https://itblog.sandisk.com/cpu-bandwidth-the-worrisome-2020-trend, 2016.

[3] 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*, 2015.

[4] O. Segal, P. Colangelo, N. Nasiri, Z. Qian, and M. Margala, "Sparkcl: A unified programming framework for accelerators on heterogeneous clusters," *CoRR*, 2015.

[5] S. A. Fahmy, K. Vipin, and S. Shreejith, "Virtualized fpga accelerators for efficient cloud computing," in *7th IEEE International Conference on Cloud Computing Technology and Science.*, 2015.

[6] M. Asiatici, N. George, K. Vipin, S. A. Fahmy, and P. Ienne, "Virtualized execution runtime for fpga accelerators in the cloud," *IEEE Access*, 2017.

[1] VINEYARD: Versatile Integrated Accelerator-based Heterogeneous Data Centers.

We thank Xilinx University Program for the kind donation of the Software and hardware platforms.