Galapagos: A Full Stack Approach to FPGA Integration in the Cloud

Naif Tarafdar
University of Toronto

Nariman Eskandari
University of Toronto

Varun Sharma
University of Toronto

Charles Lo
University of Toronto

Paul Chow
University of Toronto

Field-Programmable Gate Arrays (FPGAs) have shown to be quite beneficial in the cloud due to their energy-efficient application-specific acceleration. These accelerators have always been difficult to use and at cloud scale, the difficulty of managing these devices scales accordingly. We approach the challenge of managing large FPGA accelerator clusters in the cloud using abstraction layers and a new hardware stack that we call Galapagos. The hardware stack abstracts low-level details while providing flexibility in the amount of low-level access users require to reach their performance needs.

Computing today has evolved towards using large-scale data centers that are the basis of what many call the *cloud*. Within the cloud, large-scale applications for Big Data and machine learning can use thousands of processors. At this scale, even small efficiency gains in performance and power consumption can translate to significant amounts. In the data center, power consumption is particularly important as 40 % of the costs are due to power and cooling [1]. Accelerators, which are more efficient for classes of computations, have become an important approach to addressing both performance and power efficiency. The increasing requirements for computation can no longer be serviced by single large processors and require multi-node clusters. Today, these multi-node clusters can scale into many thousands of nodes in data centers. The efficiency gained at this scale is not just pure performance (minimal latency or high throughput) but performance per watt of energy consumed.

The need for power efficient performance acceleration has led us to investigate the integration of Field-Programmable Gate Arrays (FPGAs) as programmable accelerators in the data center. FPGAs provide the ability to create custom circuitry for a given application on a reconfigurable logic fabric. This flexibility makes FPGAs particularly attractive in the data center environment.

The significant benefit of using FPGAs in the data center can be highlighted by the flagship example of Microsoft's Catapult project [2] [3]. In this work, they have shown that with a 10 % increase in power, they are able to almost double the performance of their Bing search engine. The surge in interest in FPGAs is further evidenced by Intel's acquisition of Altera, the second largest FPGA company at the time [4].

Our solution to the growing challenge of FPGA integration is the introduction at scale of a stack to support FPGA-based acceleration that we call Galapagos. This hardware stack is created with several layers of abstraction, with each layer providing an interface to the layer above. This philosophy is similar to the OSI model, which is used to abstract networking protocols and allows users to make the tradeoff between transparency and ease of use. The stack we introduce is responsible for creating FPGA clusters from a pool of shared resources. Galapagos includes the physical setup of the FPGAs and CPUs in the data center, as well as the provisioning and orchestration of these resources. CPUs and FPGAs in Galapagos are all directly connected to the network and can use multiple network protocols for communication. The work presented in [3] statically configures a multi-FPGA cluster for the specific application.

THE NEED FOR A HARDWARE STACK

In this section, we briefly explain the need for abstractions by explaining FPGAs, the current state of integration of FPGAs in the data center, sharable cloud infrastructures of homogeneous and heterogeneous resources, and other current abstractions provided for single FPGAs and multi-node CPU clusters.

Field-Programmable Gate Arrays (FPGAs)

Field-Programmable Gate Arrays (FPGAs) are programmable devices that enable the mapping of hardware circuits onto a reconfigurable logic fabric. Modern FPGAs include fixed logic for DSP functions, blocks of internal memory, and other I/O interfaces such as PCIe and ethernet. Some FPGAs now also include ARM processors. The low-level flexibility provided by FPGAs gives the user the ability to create energy-efficient circuitry as the user can create custom logic for their given application. This low-level nature also adds a level of difficulty as traditional FPGA design requires the user to create all components, including the cores to communicate to the I/O interfaces.

FPGA Virtualization

To alleviate the difficulties of using FPGAs, several works have looked at abstracting away the I/O interfaces. This abstraction gives the user a simple handle to I/O such as the network and PCIe interfaces. Some commercial examples of abstractions are the Xilinx SDx and Intel's FPGA SDK for OpenCL [5] [6]. In both approaches, the FPGA is provisioned as an offload engine to the CPU. The platform handles copying data between the host memory and the accelerator kernel, notifying the CPU host when execution of the kernel completes. This model does not scale for large FPGA clusters.

Abstraction Layers for Compute Clusters

The creation of abstraction layers for software homogeneous compute clusters on CPUs is a common technique to increase productivity. These layers are important as coordinating the communication, data distribution and computation load balancing can become exceedingly difficult at a large scale. Some software examples of such layers include several Apache Frameworks such as Spark [7]. These frameworks handle the communication of CPU clusters and abstract the underlying hardware away from the user. Users then view their problem as a logical one that gets mapped automatically to the underlying infrastructure. This abstraction is even more important in a heterogeneous cluster as even an individual accelerator is difficult to use today, thus motivating the development of Galapagos.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/MM.2018.2877290, IEEE Micro

THE GALAPAGOS HARDWARE STACK

The Galapagos hardware stack is shown in Figure 1, which spans from the physical hardware at the bottom to the communication layer at the top. Each layer of the stack provides an interface for the layer below and the implementation of each layer is isolated. If the interface is maintained, layers can be changed independently, providing modularity and flexibility. The layer divisions of the stack come from different perspectives of usability and have analogous software layers. The analogous software stack can be found at [8]. At the very bottom layer, the Physical Hardware refers to no abstraction provided for the device. The layers above such as the Hypervisor in hardware and the OS/Hypervisor Layer in software represents the abstraction of a single device. In a software stack if there are multiple Operating Systems on a Physical Device (virtual machines) then this can be implemented as two layers as there will be multiple Operating Systems on a single Hypervisor. The Cloud Provisioning Layer represents the provisioning of these abstracted single devices from a pool of shared cloud resources. The Middleware Layer represents the abstraction of clusters of one kind of resource, such as an FPGA cluster, this is analogous to an Orchestration Layer that can provision multiple CPU nodes (either virtual machines or physical machines). The Communication Layer represents an easy to use abstraction for communicating in a heterogeneous cluster.

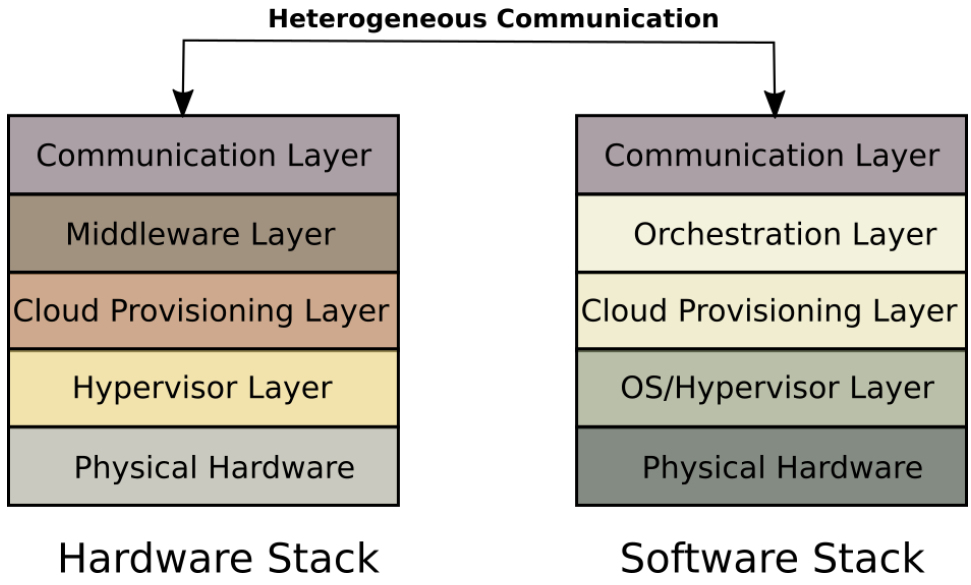

Figure 1: The Galapagos hardware and software Stack where hardware and software communicate through the use of a Communication Layer.

Physical Hardware and Network Layer

The Physical Hardware Layer refers to how we have connected our heterogeneous nodes to the network and to each other. Our system has FPGAs and CPUs connected to the Top of Rack (ToR) switch. Our FPGA boards are either connected via PCIe to an x86 CPU or are integrated with an ARM CPU via AXI buses in a System-on-Chip (SoC). The primary connection for moving data to and from the FPGA is through the network connection to the ToR switch, which is the same as the CPU servers. There is a control path for the FPGA that comes from a CPU, which is either through PCIe or an AXI interface in an SoC.

This control path facilitates the setup of a multi-node cluster and once the setup is complete, the CPUs and FPGAs communicate via the network. Users wanting to use this physical layer of abstraction would be required to create a PCIe or AXI interface to communicate to the CPU, and a

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/MM.2018.2877290, IEEE Micro

network interface to communicate to other FPGAs. Furthermore, they would also need to configure the network switch with the appropriate routing information to handle multi-node (CPU and/or FPGA) communication. This is cumbersome with even one node and extremely impractical with many nodes. Users that would like to change the underlying architecture of how the nodes are connected within the data center would be working in this layer of the stack.

Hardware Hypervisor/Shell

There are two main goals of this layer. First, this layer simplifies the interface between the physical I/O (PCIe and network) and the user application. Second, it standardizes the interface of the user application across multiple platforms so that an application only sees one common interface across all platforms to improve portability. While the line between operating system and hypervisor can be blurry, we consider this layer to be a hypervisor as it does not provide any memory virtualization. This hypervisor is shown in Figure 2.

Our current hypervisor layer has four main interfaces. One is a memory-mapped interface between the CPU and the user region (PCIe for PCIe-connected boards and AXI for the SoC FPGA boards). The second interface is a serial JTAG connection that is necessary for checking debug probes within the FPGA. Third, a controller is added to interface with off-chip DDR memory and the last abstracted interface is a streaming network connection that connects the FPGA to the data path. A typical user in this layer of the Galapagos stack would be someone who would want an easy interface to use a single FPGA. This user would not have to worry about the details of interfacing with the I/O of a specific FPGA.

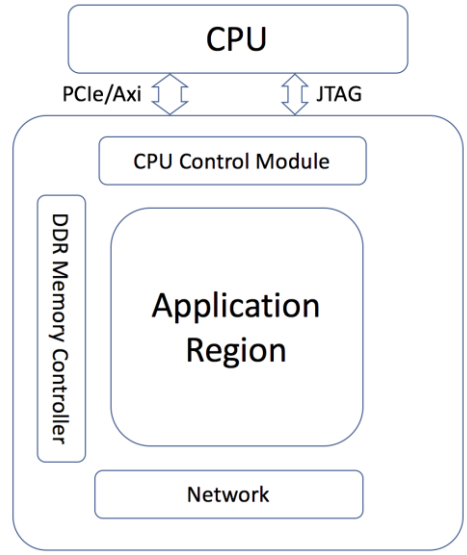

Figure 2: The hypervisors in our hardware stack. The hypervisor abstracts a control path either through PCIe or AXI and the data path is through the network connection. The USB JTAG connection is used for programming and debugging in the hardware.

Hardware Cloud Provisioning Layer

The management of our cloud is through OpenStack [8]. OpenStack maps user requests of virtual machines (VM) to an underlying cluster by placing them on hypervisors within the physical cluster. OpenStack also keeps track of the physical location of all provisioned VMs, the underlying physical nodes, and network connections. Initially, our FPGA environment was a VM or the bare-metal server itself with a connected FPGA. More recently, we support containers in

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/MM.2018.2877290, IEEE Micro

OpenStack with connected FPGAs as the preferred environment for users. Containers provide OS-level virtualization. As with VMs, different users can be isolated from one another but with a lower performance overhead. Unlike VMs, containers share the underlying host kernel and don't need to emulate an entire OS or translate instructions. Granting hardware access to virtual environments requires passing through the appropriate devices. The provisioned virtual FPGA may be an entire physical FPGA, in which case buses such as PCIe or AXI are passed through to the virtual environment directly. In the case of multiple tenants sharing a physical FPGA, a portion of the FPGA would be provided instead.

As described in our hardware layer, we have a control path through a CPU and a data path through the network. The data path requires provisioning an extra network port for each FPGA (as this is not handled by the default provisioning of a virtual machine). We use OpenStack to dynamically allocate a new network port (network address not assigned to any other node in the data center) and assign it to the physical port number where the FPGA connects to the ToR switch. Once this assignment is complete, any network connected device can send packets to the FPGA by communicating to the assigned network address. For details on the provisioning and network assignment refer to [9]. A user that would like to provision a single FPGA from a shared resource pool and use with the FPGA abstractions highlighted in the Hypervisor section would work in this layer.

Hardware Middleware Layer

The Cloud Provisioning layer in Galapagos enables the provisioning of single nodes from a sharable pool. The Hardware Middleware Layer refers to the provisioning and orchestration of entire clusters. We have implemented multiple Hardware Middleware layers on top of the Cloud provisioning layer and Hypervisor layer to show the modularity of our layer abstractions. These works are described in [9] [10] and illustrated in Figure 3. In both works, the user describes a cluster of heterogeneous devices, consisting of both CPUs and FPGAs. Both works also include clusters consisting of streaming FPGA kernels. The Hardware Middleware layer in [9] describes a packet switched network where all kernels can address any other kernel in the cluster. Each output packet from any kernel in this cluster contains metadata to specify the destination kernel. The Hardware Middleware layer in [10] describes a circuit switched network where a user defines a path (that can be modified dynamically at run-time as well) for packets to take. This approach could be used to chain virtual network functions [10]. In this model, there is no metadata as the user fixes the paths through heterogeneous devices. Both layers are implemented on the same Cloud Provisioning layer in Galapagos. A user that would like to orchestrate a cluster of FPGAs from a shared resource pool would be working in this layer. The inter-FPGA connections is configurable and can be implemented by either TCP/IP or Ethernet. With a simple change in a configuration file the user can select the network protocol the Middleware would connect the FPGA cluster.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/MM.2018.2877290, IEEE Micro

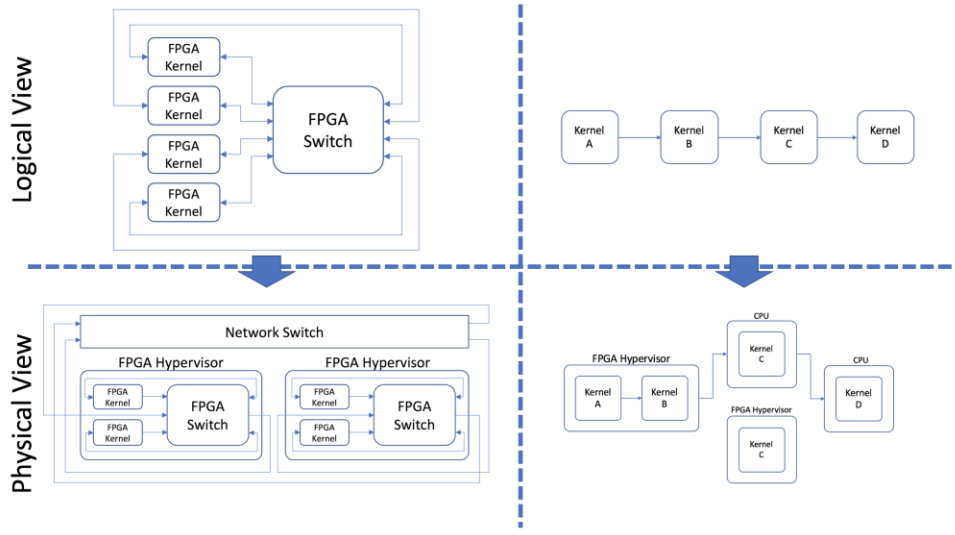

Figure 3: The two middleware layers built in our infrastructure. The left maps a logical cluster onto a physical packet switched network and on the right maps a logical chain onto a heterogeneous chain of CPUs and FPGAs.

Communication Layer

One communication layer that we are building in Galapagos is a subset of the Message Passing Interface (MPI) abstraction layer. The MPI programming model is commonly used in high-performance computing applications. The user only needs to write using the MPI abstraction and Galapagos will provision the FPGAs, make the network connections, build the FPGA logic with the appropriate network hardware and program the FPGAs. The user is simply returned a handle to the cluster. This is one example of a communication layer built on top of Galapagos. There can be other communication layers built for other domains where a different model would be appropriate (e.g. a shared memory model using RDMA). Changing the implementation of a communication layer would only require providing the IP blocks to bridge packets adhering to a communication layer to and from the network. We created hooks in the Hardware Middleware Layer for the user to insert a bridge to their layers above. A bridge will encapsulate data with a new header for a given layer.

TOOL FLOW

This entire project is open-source and can be downloaded from https://github.com/UofT-HPRC/galapagos. We have inserted a Makefile that is responsible for creating a project from the Hardware Middleware Layer and below. It also accepts an input for bridges that a user would need for upper layers when building a programming layer. The automation flow partitions the cluster into separate FPGAs, creates FPGA projects, and builds FPGA programming bitstreams. The bitstreams are built using the hypervisor that is downloaded during the build process from our online server.

EVALUATION

In the following section, we evaluate various layers of our stack. Our main benefit is productivity though that is not exactly quantifiable. Here we show that the penalty we pay for productivity is minimal in terms of resources and performance. Table 1 shows the resource overheads in ab-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/MM.2018.2877290, IEEE Micro

solute numbers and relative percentage of what is available on the FPGA of a few individual layers within the Galapagos stack. These results are from implementing the Galapagos layers on the Alpha Data 8K5 FPGA Board, which uses an Ultrascale Kintex FPGA (XCKU115-1157) [12]. We have also implemented the stack on other boards and the utilization results are similar.

Table 1 FPGA Resource Utilization

Galapagos Layer	**Logic (LUTs) [663360 Total]**	**Registers [1326720 Total]**	**Block RAM [2160 Total]**
Hypervisor [9] [10]	57778 (8.7 %)	57720 (4.3 %)	341 (7.8 %)
Middleware Layer [9]	18292 (2.8 %)	8582 (0.64 %)	0 (0 %)
MPI Communication Layer [11]	469 (0.06 %)	861 (0.06 %)	0 (0 %)

As Table 1 shows, the hardware resource overhead of our layers is minimal (at most 8.7 % logic). Furthermore, the performance overhead of these layers is also very minimal. All layers, including the Hypervisor, Middleware layer and MPI programming layer all perform at line-rate with a few-cycle penalty. The observed bandwidth of an MPI connected cluster over TCP/IP is 7.3 GB/s and 9 GB/s using Ethernet. The bandwidth limitations are due to the limitation of the TCP/IP, Ethernet IP core as well as the header overheads.

CONCLUSION

Abstraction layers increase productivity by hiding details that can get overwhelming. Furthermore, having a stack of these abstractions allows users to flexibly choose their intended level of abstraction. Our hardware stack adds little overhead but introduces a large potential increase in productivity as we automate the creation of a heterogeneous cluster: from the provisioning of resources, to connecting the nodes within the cluster, to providing an easy programming model to communicate to your cluster.

ACKNOWLEDGMENTS

The authors would like to thank the SAVI testbed, NSERC, Xilinx, CMC and Huawei Technologies Canada for providing the infrastructure and funding this research.

REFERENCES

[1] Z. Song, X. Zhang and C. Eriksson, "Data Center Energy and Cost Saving Evaluation," *Energy Procedia*, vol. 75, pp. 1255-1260, 2015.

[2] A. Putnam et al., "A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services," in *International Symposium on Computer Architecture (ISCA), Proceedings of the 41st Annual*, Minneapolis, 2014.

[3] A. Caulfield et al., "A Cloud-Scale Acceleration Architecture," in *Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture*, 2016.

[4] D. Clark, "Intel Completes Acquisition of Altera," Wall Street Journal, 28 December 2015. [Online]. Available: https://www.wsj.com/articles/intel-completes-acquisition-of-altera-1451338307. [Accessed 5 May 2018].

[5] Xilinx, "Programmable Abstactions," Xilinx, [Online]. Available: https://www.xilinx.com/products/design-tools/all-programmable-abstractions.html.

[6] Intel, "Intel FPGA SDK for OpenCL," Intel, [Online]. Available: https://www.altera.com/products/design-software/embedded-software-developers/opencl/overview.html.

[7] Spark Framework, "Spark," [Online]. Available: http://sparkjava.com/.

[8] OpenStack Foundation, [Online]. Available: https://www.openstack.org/software/.

[9] N. Tarafdar, T. Lin, E. Fukuda, H. Bannazadeh, A. Leon-Garcia and P. Chow, "Enabling Flexible Network FPGA Clusters in a Heterogeneous Cloud Data Center," in *Field-Programmable Gate Arrays (FPGA '17), Proceedings of the 2017 ACM/SIGDA International Symposium on*, Monterey, 2017.

[10] N. Tarafdar, T. Lin, N. Eskandari, D. Lion, A. Leon-Garcia and P. Chow, "Heterogeneous Virtualized Network Function Framework for the Data Center," in *Field Programmable Logic and Applications (FPL), 2017 27th International Conference on*, Ghent, 2017.

[11] Alpha Data, "ADM-PCIE-8K5," [Online]. Available: https://www.alpha-data.com/dcp/products.php?product=adm-pcie-8k5.