The Future of FPGA Acceleration in Datacenters and the Cloud

CHRISTOPHE BOBDA and JOEL MANDEBI MBONGUE, University of Florida
PAUL CHOW, MOHAMMAD EWAIS, NAIF TARAFDAR, and JUAN CAMILO VEGA, University of Toronto
KEN EGURO, Microsoft
DIRK KOCH, Manchester University
SURANGA HANDAGALA and MIRIAM LEESER, Northeastern University
MARTIN HERBORDT and HAFSAH SHAHZAD, Boston University
PETER HOFSTE, IBM POWER Systems Performance
BURKHARD RINGLEIN, IBM Research Europe
JAKUB SZEFER, Yale University
AHMED SANAULLAH, Red Hat, Inc
RUSSELL TESSIER, University of Massachusetts Amherst

In this article, we survey existing academic and commercial efforts to provide Field-Programmable Gate Array (FPGA) acceleration in datacenters and the cloud. The goal is a critical review of existing systems and a discussion of their evolution from single workstations with PCI-attached FPGAs in the early days of reconfigurable computing to the integration of FPGA farms in large-scale computing infrastructures. From the lessons learned, we discuss the future of FPGAs in datacenters and the cloud and assess the challenges likely to be encountered along the way. The article explores current architectures and discusses scalability and abstractions supported by operating systems, middleware, and virtualization. Hardware and software security becomes critical when infrastructure is shared among tenants with disparate backgrounds. We review the vulnerabilities of current systems and possible attack scenarios and discuss mitigation strategies, some of which impact FPGA architecture and technology. The viability of these architectures for popular applications is reviewed, with a particular focus on deep learning and scientific computing. This work draws

Authors' addresses: C. Bobda and J. Mandebi Mbongue, University of Florida, Gainesville, FL 32611-6200; emails: {cbobda, jmandebimbongue}@ece.ufl.edu; P. Chow, M. Ewais, N. Tarafdar, and J. C. Vega, University of Toronto, Toronto, Ontario M5S 3G4 CANADA; emails: pc@eecg.toronto.edu, mewais@ece.utoronto.ca, naif.tarafdar@mail.utoronto.ca, camilo.vega@mail.utoronto.ca; K. Eguro, Microsoft, Microsoft Corporation, One Microsoft Way, Redmond, Washington, 98052, United States of America; email: eguro@microsoft.com; D. Koch, University of Manchester, Oxford Road, M14 9LP Manchester; email: dirk.koch@manchester.ac.uk; S. Handagala and M. Leeser, Northeastern University, 360 Huntington Avenue, Boston, Massachusetts 02115; emails: s.handagala@northeastern.edu, mel@ece.neu.edu; M. Herbordt and H. Shahzad, Boston University, 333 Photonics Center Building, 8 Saint Mary's Street Boston, MA 02215; emails: herbordt@bu.edu, NGL@zurich.ibm.com; P. Hofste, IBM POWER Systems Performance, 11400 Burnet Rd Austin, TX 78758; email: hofstee@us.ibm.com; B. Ringlein, IBM Research Europe, Zurich Research Laboratory, Säumerstrasse 4 CH-8803 Rüschlikon, Switzerland; email: NGL@zurich.ibm.com; J. Szefer, Yale University, 10 Hillhouse Avenue New Haven, CT 06511; email: jakub.szefer@yale.edu; A. Sanaullah, Red Hat, Inc; email: asanaull@redhat.com; R. Tessier, 134 Marston Hall, University of Massachusetts, 130 Natural Resources Road Amherst, MA 01003-9293; email: tessier@umass.edu.

1936-7406/2022/02-ART34 $15.00
https://doi.org/10.1145/3506713

from workshop discussions, panel sessions including the participation of experts in the reconfigurable computing field, and private discussions among these experts. These interactions have harmonized the terminology, taxonomy, and the important topics covered in this manuscript.

CCS Concepts: • **Hardware** → **Reconfigurable logic and FPGAs**; • **Computer systems organization** → *Cloud computing*; • **Security and privacy** → *Systems security;*

Additional Key Words and Phrases: Cloud, datacenter, FPGA, virtualization, security

ACM Reference format:
Christophe Bobda, Joel Mandebi Mbongue, Paul Chow, Mohammad Ewais, Naif Tarafdar, Juan Camilo Vega, Ken Eguro, Dirk Koch, Suranga Handagala, Miriam Leeser, Martin Herbordt, Hafsah Shahzad, Peter Hofste, Burkhard Ringlein, Jakub Szefer, Ahmed Sanaullah, and Russell Tessier. 2022. The Future of FPGA Acceleration in Datacenters and the Cloud. *ACM Trans. Reconfigurable Technol. Syst.* 15, 3, Article 34 (February 2022), 42 pages.
https://doi.org/10.1145/3506713

1 INTRODUCTION

Since Microsoft published their work on Catapult in 2014 [115], **Field-Programmable Gate Arrays (FPGAs)** have become more than an exotic and niche technology that can only be tamed by the magicians of hardware. Major players such as Alibaba, Amazon, Baidu, Huawei, and Tencent now expose FPGAs to application developers in their datacenter infrastructures. Others use FPGAs to provide applications as a service (Microsoft [27] and Nimbix) or for internally developed applications. Furthermore, a growing number of projects are underway across the globe, in academia and other research organizations, to provide the benefit of acceleration and flexibility remotely to users. While FPGAs are increasingly available in datacenters, their long-term adoption in these venues is not guaranteed. Several technologies must be provided for FPGAs to compete with their peers—**central processing units (CPUs)**, **graphics processing units (GPUs)**—and the flood of **artificial intelligence (AI)**-specific processors currently deployed and in development.

Architecture and integration at the board and the system level must ensure that applications harness the strengths of reconfiguration and mitigate its weaknesses. The provisioning of hardware resources requires adequate middleware, hardware virtualization, and domain separation mechanisms. Efficient and flexible hardware resource provisioning increases the possibility of seamless integration, ideally with no redesign, in existing cloud management infrastructure. The spatial sharing of FPGAs increases the complexity of cloud schedulers and resource managers. Fortunately, the large body of work developed in the past decade can be leveraged in cloud task schedulers currently in use in datacenters, such as the list-scheduling used in Amazon cloud.

With multiple users temporally or spatially sharing FPGAs in datacenters, the security of designs must be guaranteed. It has recently been demonstrated that FPGAs can be used covertly for various types of cybersecurity attacks in the cloud [117, 118, 145, 146]. Mitigation strategies have been proposed [83, 113] and many more are currently in development. However, despite all best efforts, current FPGA architectures may still be resistant to protection against cloud-based attacks.

Without applications, any architecture incorporating FPGAs will be of no use. Programming languages and high-level design environments as well as efficient hardware/software mapping strategies are necessary to facilitate the transition for software engineers. In this article, experts from academia and industry evaluate the road previously traveled and the successes and failures of current solutions. We open a window on the future of FPGA-accelerated datacenters, identify opportunities and challenges, and discuss the path to success and broad adoption of FPGA technology for computing.

The remainder of the article is organized as follows: The next section briefly describes the landscape of FPGA use in cloud infrastructure. A taxonomy of FPGA architecture integration is provided in Section 3 followed by a detailed discussion of current architectural development and trends. The integration of FPGA resources into **operating systems (OS)** and software is discussed in Section 4. Topics include shell interfaces, middleware, and virtualization. In Section 5, we explore challenges in hardware and system security in single-user (tenant) and multi-tenant cloud compute environments. A critical review of current FPGA architecture is provided and suggestions to increase resiliency are offered. In Section 6, we review applications that can benefit from cloud implementation. Deep learning is the main candidate, but image processing, sorting, and database operations can also benefit. We conclude the article in Section 7 with a summary of the opportunities and challenges of the paradigms discussed.

2 THE LANDSCAPE

After decades of single-thread-driven performance growth for computing platforms, followed by more than a decade where the majority of performance growth resulted from increases in the numbers of hardware threads, we have now entered a period where the majority of hardware-based computational performance growth is expected to come from specializing hardware [50, 68, 127]. The most power-constrained platforms, such as cell phones, already have multiple types of specialized processing elements including GPUs, neural processors, and specialized signal processing, media, and security engines. While some think of ARM [10] as an architecture that has **instruction set architecture (ISA)**-driven efficiency, the main reason ARM is gaining traction beyond cell phones and embedded devices is that ARM provides a **system-on-chip (SoC)** ecosystem that is thus far unmatched [53], enabling the rapid construction of more specialized and efficient SoCs.

One of the earliest successes in using FPGAs in the cloud was the Catapult project at Microsoft being used for Bing searches [115]. This is an example of "provider application as a service" where the user is not aware that FPGAs are being used to accelerate their applications. Subsequently, Microsoft provided acceleration for Azure customers by offloading machine learning and host networking to hardware [47]. In the past few years, FPGAs in the cloud have been used for a variety of different applications (see Section 6), including the acceleration of networking, privacy and security, and machine learning, and data analytics. As processing becomes more heterogeneous, FPGAs stand out as accelerators that can process data directly from the network and provide benefits to users with or without their knowledge. Thus, their use is likely to grow dramatically in the future.

Of course, specialization is challenging primarily, because it intrinsically trades off flexibility for efficiency, primarily measured in reduced chip area and improved energy per computation at equivalent performance. This approach requires allocating hardware resources to the "right" priorities for the platform and also requires ensuring that the platform comes with a software development and runtime ecosystem that ensures the specialized elements are used to a sufficient degree to justify allocating the resources to these specialized rather than general-purpose functions. Platforms that provide more extensive and prescriptive application development environments will be in the lead in providing such specialization.

To date, most cloud providers have focused on more general-purpose platforms, renting out standardized scalable infrastructure. Offerings are primarily differentiated on the number of (virtual) cores and the amount of memory and network bandwidth per CPU and storage. Offerings with more specialized hardware, primarily GPUs, are mostly separate specialized offerings, e.g., for **high-performance computing (HPC)**, rather than elements that provide the broad underpinnings of a cloud platform architecture.

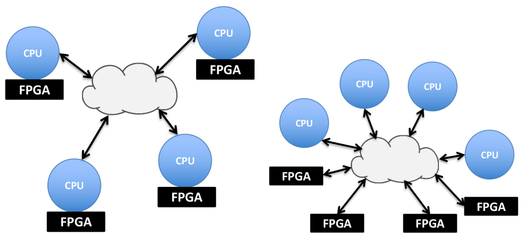
A number of cloud providers have started to use specialized hardware (including FPGAs) [27] and even started to develop their own SoCs [50, 127] rather than rely on vendor silicon, which opens the door to increased hardware-based platform differentiation. It is envisioned that as cloud infrastructure progresses from **infrastructure as a service (IaaS)** to virtualized platforms and then to microservices and **function as a service (FaaS**), cloud computing may also provide the software ecosystems that enable the increased introduction of specialized hardware.

In the creation of specialized hardware, reconfigurable logic occupies an interesting middle ground. Reconfigurable logic can be leveraged to create a wider set of specialized computational elements from the same hardware that each can outperform and improve efficiency over a general-purpose processor for a specific task or set of tasks. Many systems combine reconfigurable offerings with highly flexible I/O, allowing a very large number of system configurations. At the same time, for any fixed configuration of the FPGA supporting a fixed set of tasks, hardware that is not (as) reconfigurable can be implemented significantly more efficiently.

Even if reconfigurable logic becomes ubiquitous, it still may not be programmed by a wide audience. This does not necessarily mean the logic is not regularly reprogrammed. The logic configuration is provided by the system vendor or cloud provider, and not by independent software vendors or end users. There are a number of well-known reasons for this: the difficulty of providing and maintaining a software ecosystem, difficulty of ensuring system integrity, difficulty of ensuring security, and the skills required to program the logic.

Before delving more into considerations specific to the deployment of reconfigurable logic in clouds and datacenters, we take stock of some of the different ways in which reconfigurable logic can be deployed and introduce some of the challenges within these contexts. Perhaps the tightest way to integrate reconfigurable logic in a processor is to use it as a reprogrammable execution unit sharing a register file with the remainder of the core. While not a common use of reconfigurable logic today, contemplating such a use illustrates some of the challenges. For example, a unit could be configured to implement new instructions before they are directly supported in custom hardware. In this case, the software ecosystem is essentially the same as for any other new (set of) instruction(s). A more ambitious step would be to allow reprogramming and changing of the performance profile but not the set of supported instructions. This approach requires a much more complex software ecosystem, as compilers tend to be highly tuned to performance specifics. This approach is also likely to require OS-level coordination, as schedulers would likely have to become aware of these profile changes to ensure the right tasks are scheduled to the processors with the appropriate performance profile. A next step might include support for different profiles with different (vendor-defined) sets of instructions. This step now puts an even larger burden on the software ecosystem but still leaves the responsibility with the hardware vendor. A final step would be to allow cloud provider or end-user reprogramming of the embedded reconfigurable logic to create user-defined instructions. Architectures such as RISC-V and OpenPOWER explicitly allow for the introduction of implementation-specific instructions. It is not difficult to contemplate a cloud-specific use of such a capability, for example to improve the performance of a specific encryption or decryption algorithm that does not already have direct hardware support.

A first approach might be to treat a processor with a customer-specific instruction as a customer-specific processor to be verified in its entirety by the customer as if they had built a new processor with a customer-specific instruction. Such an approach might not be satisfactory, but with cloud providers now building their own SoCs it is perhaps not unrealistic. Beyond the software ecosystem and system functionality, system integrity and system security are also issues, even for this very restricted way of introducing reprogrammable functions. For example, one will want to ensure that reprogramming cannot result in physical damage to the system or affect reliability. The issue is not far-fetched, as FPGAs can be configured with collections of ring oscillators with local power densities that exceed normal design constraints [113]. Security could also be a concern, as one would have to be very careful that a custom execution unit does not observe (and store) any information from another user context or a context at a different privilege level. Clearly, much



(a) Accelerator Model (b) Peer Model

Fig. 1. Example FPGA placement in datacenters.

work remains before general-purpose user-reprogramming of a reconfigurable unit integrated at this level can be commonplace.

Next, we consider a reconfigurable unit that shares memory with other processors in the system, either physically integrated in the same chip or on the same module or provided separately. Bus protocols range from architecture-specific (QPI or UPI for example) to open standards such as OpenCAPI, AXI [9], CCIX, and CXL. Standards at the protocol level have to be accompanied by software standards at the system level. One common approach is for the native OS (or hypervisor) to retain control over system resources and manage memory access to shared memory. This provides a clear foundation for system organization, scheduling and security, as it can build on the knowledge from other systems with heterogeneous processing elements. However, there are special considerations for processing elements built from reconfigurable logic. To maintain system integrity, the address translation and/or bus interface units typically must remain the responsibility of the processor or system vendor to ensure the accelerator operates in a virtual (effective) address space instead of having physical addressing capabilities. This goal can be achieved if the interfaces to the reconfigurable logic operate in a virtual (effective) address space, or any reconfigurable logic provided by someone not responsible for the physical infrastructure is combined (and verified to be combined) with logic that is. This can also raise an issue of trust. For example, if the only way to guarantee integrity is to allow the system provider to do extensive checking of the logic during the build process, then an end-user or independent software vendor who wants to differentiate based on the reconfigurable logic they provide may be forced to expose that logic to the system provider.

Another level is integration as a device, either for computational acceleration or to enhance network or storage capabilities. A standard device does not have access to all system resources, and it is easier to delineate the responsibilities. CXL supports device and shared memory accelerators by providing a variety of protocol profiles. A typical use of reconfigurable logic in this context might be the creation of "smart" storage or network devices, providing functions such as security, data compression or decompression, or filtering. Smart networking or smart storage devices that allow the combination of user-created reconfigurable logic within a network or storage controller do again raise a set of concerns that must be addressed.

Finally, perhaps the most ambitious approach would be to develop a datacenter that consists of (essentially) only FPGAs. Such an approach is made easier because FPGAs integrate increasingly powerful CPU cores, (stacked) DRAM memory, and high-speed I/O. The software foundation for such an infrastructure would build on the conventional notions underpinning conventional system design.

3 FPGA CLOUD ARCHITECTURES

The computational capability and flexibility of a reconfigurable fabric, coupled with the performance of ASICs for high speed I/O (e.g., SERDES units), allows FPGA-based systems to be deployed almost anywhere in the cloud to accelerate compute, networking, and storage. An example of this versatility is illustrated in Figure 1, which shows two common, contrasting deployment models. However, while the potential for arbitrary deployment certainly exists, FPGAs cannot be randomly placed as part of any cloud infrastructure. The need for cost-effectiveness leads to an emphasis on size, power, cooling, compatibility, and in-place upgradability—all while ensuring that the specific performance, memory, server capability, and network connectivity needs of cloud workloads are met. Thus, based on a system's requirements, certain types of cloud FPGA architectures can have substantially more benefit than others.

This section discusses architectural choices for the integration of FPGAs into cloud computing infrastructures. Current architectures in use in academia and industry have distinctive characteristics, the most important of which are listed below, that have helped shape them over the past two decades.

- Very large scale, with (potentially) millions of nodes;
- Mature, having evolved through many generations;
- Highly competitive, with several vendors offering similar products, yet with high value in differentiation;
- Diverse evolving workloads, but with some emerging high-performance applications;
- System upgrades are always partial placing a premium on compatibility through several generations;
- Heterogeneity, with different sets of nodes for different types of users.

We hypothesize that future organizations will only slightly evolve from existing architectures, depending on the devices and integration technology. A radical break is not expected. In the next subsection, we begin with a taxonomy of system organizations, followed by a review and discussion of existing architectures.

3.1 Taxonomy

This subsection presents a set of high-level attributes that define cloud FPGA systems. These taxonomic categories provide an effective means to not only analyze the relationships between design choices and system constraints, but also to describe and compare different cloud FPGA architectures. We continue with an overview of mainstream FPGA cloud systems, both in production and research, and conclude by analyzing architectural trends in these systems in the context of the proposed taxonomy.

The taxonomy is based on the critical aspects of any cloud FPGA system: **(A) Type of FPGA boards**, **(B) Placement of FPGAs in the system**, **(C) Network Connectivity**, **(D) Intra-node Connectivity**, and **(E) Use Cases**. Note that these taxonomic categories are neither mutually exclusive nor comprehensive: It is possible for systems to have multiple sub-categories and for new sub-categories to be added later to encapsulate future innovation.

Figure 2 illustrates common architectures. These architectures, and a sample benefit, are as follows:

(1) **Bump-in-the-Wire:** Large-scale compute, network and storage acceleration;
(2) **Co Processor:** Local compute acceleration;
(3) **Storage attached:** Local storage acceleration;
(4) **Back-end:** Ultra low latency, rack scale FPGA-FPGA communication;
(5) **Smart NIC:** Local network acceleration;
(6) **Network HW:** Flexible routing/switching protocols;
(7) **Local Cluster:** Multi-accelerator system;

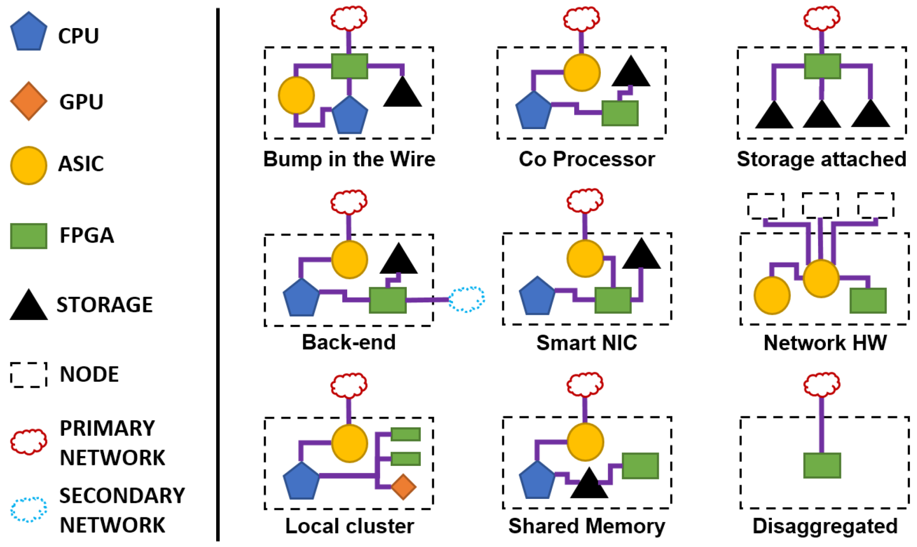


Fig. 2. Common FPGA architectures. Note that Accelerator Model and Peer Model from Figure 1(b) are illustrated here as Co Processor and Disaggregated, respectively.

(8) **Shared Memory:** Cache coherent acceleration;
(9) **Disaggregated:** High infrastructure utilization.

3.1.1 *Type of FPGA Boards.* Given that cloud providers do not currently create their own FPGAs, the smallest unit of differentiation is the FPGA board; both **(a) Off-the-shelf** and **(b) Custom** are possible. Economic advantages depend on the scale of deployment and provider development infrastructure. Given the latter, custom boards still have higher start-up and upgrade costs, but may be cheaper in large quantities. But the advantage of scale also affects off-the-shelf economics, as the provider has the market power to affect price and board features.

With custom FPGA boards, just about any attribute can be varied, such as number/types/bandwidths of I/O ports, FPGA family, off-chip memory type and size, form factor, and other on-board devices. This ensures that the boards closely match the specifications/requirements of the target system, from computation to cooling. Nonetheless, off-the-shelf boards are available for every (currently) sizable usage domain, including SoCs, node-level networking, NoCs, data center switches, and storage.

3.1.2 *Placement of FPGAs.* FPGAs can be placed in either a **(a) distributed** or **b) (centralized** manner. Having a distributed FPGA placement means that compute/storage nodes have their own FPGAs, and thus do not have to compete for the resource. This leads to more offload capability, greater reliability, since FPGA failure does not affect other compute/storage nodes, and reduces security concerns, since offloads for different nodes can be isolated. It is also possible to place FPGAs in a centralized manner, typically inside the networking nodes (e.g., in switches as ASIC-FPGA, CPU-FPGA, or FPGA only circuits). Substantially fewer FPGAs are needed for such a deployment; this typically translates to easier management, lower power consumption, lower **total cost of ownership (TCO)**, smaller average node sizes, and potentially higher performance, since expensive high-end FPGAs can be used (and upgraded more frequently).

3.1.3 Network Connectivity. Within each node, it is possible for FPGAs to be **(a) not connected to any network** or connected to **(b) the primary network** and/or **(c) a secondary network**. Being connected to the primary data center network enables FPGAs to intercept/accelerate network traffic to the node, as well as achieve data-center-wide scalability for FPGA workloads, since multiple FPGAs can directly communicate with each other. However, the circuitry needed to support this FPGA position can consume a significant portion of FPGA resources. This includes circuits to support high resiliency, since the FPGAs can be a single point-of-failure: An entire node can become unstable in the case of an FPGA error. In the case of secondary network connectivity, FPGAs can communicate across nodes with significantly more flexibility in the topology used (e.g., mesh, torus, switched), as well as the communication protocol, all of which can lead to ultra-low latencies. However, using a custom network configuration means that complex router hardware, routing algorithms, and switch arbitration policies may need to be implemented on each FPGA. Moreover, complex cabling may be required, which can add a significant burden to the overall data center architecture [27].

3.1.4 Intra-node Connectivity. FPGAs within a node can be **(a) not connected to any other significant device** (i.e., be a *Disaggregated* resource) or connected to one or more devices; **(b) connected to CPUs**, e.g., through PCIe and possibly with cache coherence using interconnects such as CCIX, CXL, or CAPI; **(c) connected to other FPGAs**, e.g., through a PCIe switch and/or using direct and programmable interconnects; **(d) connected to GPUs**, e.g., through a PCIe switch; **(e) connected to ASICs**, e.g., through multiple potential forms of connectivity depending on the ASIC and nature of coupling such as a NIC or tensor processor; **(f) connected to storage devices** through the device-specific interface, e.g., SPI for flash and DDR for SDRAM.

3.1.5 Use Cases. Use cases have a substantial impact on architecture, since cloud providers must ensure workload requirements are met (e.g., performance) without compromising on core aspects (e.g., security, reliability). Here, we look at some common cloud FPGA use cases. **(a) Customer applications:** Customers can develop, simulate, debug and compile their custom FPGA logic, as well as scale their infrastructure and change resources according to their workload demands. A wide pool of applications can be deployed, e.g., in genomics, financial analytics, computational fluid dynamics, video processing, transcoding, and security. Several development environments are available so users do not need to write their own HDL code. **(b) Provider Application as a Service (AaaS):** The cloud provider supports a limited set of customer applications by developing the FPGA design themselves: Only the necessary APIs and high-level design parameters are exposed. This model ensures high performance and resilience at the expense of reducing customer access to the entire FPGA. **(c) Provider applications:** In this case, cloud providers use FPGAs to accelerate their internal workloads, e.g., **software defined networking (SDN)**, as well as save CPU resources that can then be rented to the customer.

3.2 Production Architectures

In this subsection, we discuss what can be referred to as *core* or *mainstream* production cloud FPGA systems that are either in widespread or large-scale use.

Perhaps the most unique and widely deployed production system is Microsoft's Catapult v2 [27], which has FPGAs in most Azure and Bing **stock keeping units (SKUs)** in a *bump-in-the-wire* configuration: An FPGA sits between the **top-of-rack switch (TOR)**, **network interface card (NIC)**, ASIC, and CPU, hence enabling data-center wide communication within tens of microseconds of latency. These hundreds of thousands of FPGAs (or more) are used for both internal (e.g., network packet processing and Bing search [33]) and external workloads (e.g., machine learning inference as a service [33]).

Another type of a widespread production system is the single node accelerator model, which leverages FPGAs in either a *Co-Processor* configuration, or as a *Local Cluster* where devices are connected either via a PCIe switch or using direct FPGA-FPGA interconnects. A number of cloud providers such as AWS, Huawei, Baidu, Tencent, Nimbix, and Alibaba use this model. These systems are used by customers to run a wide pool of cloud native applications such as genomics, financial analytics, data acquisition, computational fluid dynamics, video processing, image processing, transcoding, security, and AI workloads. There are also examples of these FPGAs being used by providers for their own workloads. Baidu uses FPGAs to accelerate its cloud-based storage, SQL queries, data security, search engine, and AI workloads. FPGA-based AI chips—such as Baidu's Kunlun for AI, Alibaba's Ouroboros for speech recognition, and Alibaba's Hanguang 800 for inference operations—are deployed in their cloud data centers [43]. Alibaba has reported 75% savings in TCO by using FPGAs to oversee product images on its e-commerce site [161]. In 2018 it reported over $30 billion retail on its website in a single day (compared to $5 billion on all US online and in-store retail on Black Friday 2017); this was possible with its data center FPGAs being used to accelerate transactions and provide recommendations to users [48].

There are also systems that are widely deployed, but where there is insufficient publicly available information for analysis. Amazon has announced **AQUA (Advanced Query Accelerator**) nodes for its Redshift data warehouse, available through the RA3.16XL and RA3.4XL instances. These nodes use FPGAs to accelerate dataset filtering and aggregation. Baidu uses Smart NICs to improve virtualization and workload performance. OVHCloud also uses Smart NICs, but for network packet processing to mitigate **distributed-denial-of-service (DDoS)** attacks in its cloud traffic. Scaleflux CSD2000 SSDs are deployed by over 40 data centers globally. An example is the Alibaba cloud, which uses Scaleflux CSD20004 in place of traditional **solid state drives (SSDs)** on their storage nodes to accelerate applications such as MySQL, Aerospike, Oracle, and PostGreSQL. Samsung Smart SSDs are deployed in the Nimbix cloud where they accelerate Apache Spark, running queries up to 6× faster when using software from Bigstream. Eideticom's computational storage processor has been implemented in Barreleye G2 servers on Rackspace.

3.3 Research Architectures

In this subsection, we discuss systems that are presently in research and development and represent the most technologically plausible candidates for widespread future deployment.

One of the most commonly used research architectures is the cluster of *Back-End* tightly coupled FPGAs that deploy a secondary network using direct and programmable interconnects to connect FPGAs across nodes. Microsoft's Catapult v1 was a back-end system that connected multiple nodes in 6×8 tori [115]. It was demonstrated using Microsoft's Bing workloads; it is not clear whether it was ever part of a production cloud. Other research examples include Maxwell [14], Novo-G# [55], Noctua system at the Paderborn Center for Parallel Computing [112], and Albireo nodes of the Cygnus supercomputer system at University of Tsukuba. Although this approach can substantially reduce FPGA-FPGA latency, it is difficult to scale beyond a single rack due to wiring requirements; in the general case it also requires each FPGA to implement a router to support the communication. Currently no such example can be found operating in the production cloud.

Another research area proposed in References [51, 80] involves ***Channel-over-Ethernet*** **(CoE)**; a back-end, inter-FPGA Ethernet communication network using the OpenCL kernel programming. The results demonstrate the feasibility of such a configuration as the system achieves a latency of 0.99 μs for inter-FPGA communication via the secondary Ethernet switch as compared to 29.03 μs

Table 1. Classification for Production and Research FPGA Cloud Architectures Based on the Taxonomic Categories Discussed in Section 3.1

	Board Type	**Placement**	**Network Connectivity**	**Intra-node Connectivity**	**Use Case**
PRODUCTION SYSTEMS					
Alibaba	Custom	Distributed	None	FPGA, CPU, Storage	Customer, Provider
Baidu	Custom	Distributed	None	CPU, Storage	Customer, Provider
Microsoft Catapult v2 [27]	Custom	Distributed	Primary	CPU, ASIC, Storage	AaaS, Provider
Amazon AWS F1	Custom	Distributed	None	FPGA, CPU, Storage	Customer
Huawei	Custom	Distributed	None	FPGA, CPU, Storage	Customer, Provider
Nimbix	Off-the-shelf	Distributed	None	CPU, Storage	AaaS
Tencent	Off-the-shelf	Distributed	None	CPU, Storage	Customer
RESEARCH SYSTEMS					
Microsoft Catapult v1 [115]	Custom	Distributed	Secondary	CPU, Storage	Research
Enzian	Custom	Distributed	Secondary	CPU, Storage	Research
Cygnus	Off-the-shelf	Distributed	Secondary	FPGA, CPU, GPU, Storage	Research
IBM CloudFPGA [3]	Custom	Distributed	Primary	Storage	Research
Maxwell [14]	Off-the-shelf	Distributed	Secondary	CPU, Storage	Research
NARC [35]	Off-the-shelf	Distributed	None	CPU, Storage	Research
Noctua	Off-the-shelf	Distributed	Secondary	FPGA, CPU, Storage	Research
Novo-G [54]	Off-the-shelf	Distributed	None	FPGA, CPU, Storage	Research
Novo-G# [55]	Off-the-shelf	Distributed	Secondary	FPGA, CPU, Storage	Research
IBM Power8 + CAPI	Off-the-shelf	Distributed	None	FPGA, CPU, GPU, Storage	Research
IBM SuperVessel	Off-the-shelf	Distributed	None	CPU, Storage	Research
SAVI [86]	Off-the-shelf	Distributed	Primary	CPU, Storage	Research

Though all systems place FPGAs in a "Distributed" manner, the placement column is still shown to highlight this trend. It is further discussed in Section 3.4.2.

via the host CPU. A drawback is that data are sent as packets and that there is additional overhead, such as IP addresses and flags, that reduce the effective data rate [152].

Other research architectures include systems that support a *Local Cluster*, but where the communication scaling via direct interconnects is limited to a single node. An example includes Novo-G (a former version of Novo-G#) [54]. Other examples include the research systems currently deployed at the IBM SuperVessel Cloud and the IBM Power8+CAPI cluster at the University of Texas, Austin that use a *Shared Memory* cache coherency model.

A different approach is to directly connect FPGAs to the datacenter network as a standalone resource. Each FPGA can be accessed by a CPU or another FPGA resulting in good scalability. CloudFPGA by IBM Zurich Research Lab is one example [121, 122, 157–159]. The authors have built a prototype with multiple chassis for data center scale, capable of hosting 1024 FPGAs per rack [3]. A drawback of such an architecture may be that FPGA-CPU communication is necessarily among separate nodes and has high latency. Another consideration is the increase in the number of TOR connections.

The University of Toronto SAVI testbed connects FPGAs to the primary network [140]. The authors in References [26, 138] have demonstrated that virtualizing FPGA resources on the SAVI testbed enables multiple regions within an FPGA device to support different designs using APIs such as OpenStack. Enzian at ETH Zurich employs an FPGA as a node connected to the network on one end and coherently attached to a large server-class SoC on another node. Unlike Microsoft's *bump-in-the-wire*, this system allows CPUs to either connect directly to the network or via the FPGA. Unlike other cache coherent systems, it allows the FPGA side of the cache coherency protocol to be extended and tailored [6]. The **Gator Reconfigurable Cloud infrastructure (GatorRecc)** provides a platform to explore multi-tenancy FPGA usage in cloud applications, using virtual instances (CPU+FPGA) and managed by Openstack[100].

3.4 Architectures Trends

Table 1 classifies production and research cloud FPGA architectures based on the taxonomy in Section 3.1. As we can see, research systems generally explore different varieties of cloud architecture options. While production systems are bounded by factors such as **total-cost-of-ownership (TCO)**, **power-usage-effectiveness (PUE)**, performance, resilience, modularity, scalability, and security, research systems tend to enjoy greater degrees of freedom. To effectively analyze these architectures, we highlight trends in their architectures based on the following categories: (1) Boards, (2) Placement, (3) Network Connectivity and Use Cases, and (4) Intra-node Connectivity.

3.4.1 Boards. Table 1 shows that a majority of production cloud vendors have used custom boards in their deployments. For Microsoft in particular, this was necessary, since requirements for placing FPGAs in special HPC SKUs "constrained power to 35W, the physical size to roughly a half-height half-length PCIe expansion card (80mm × 140 mm), and tolerance to an inlet air temperature of 70 °C at 160 lfm airflow" [27]. Custom boards are not required, however: Nimbix and Tencent both use off-the-shelf boards. For research systems, custom boards are preferred if the proposed systems are *Disaggregated*, network attached (e.g., Enzian), and like IBM CloudFPGA. Also, for earlier *Back-end* systems, such as Catapult v1 and Novo-G#, a customized board allowed the system to increase transceiver count. However, we can see that recent *Back-end* and *Local Cluster* systems mostly use off-the-shelf boards. Systems with no inter-node communication network almost always use off-the-shelf boards.

3.4.2 Placement. Table 1 shows that all these systems deploy FPGAs in a distributed fashion. This is because: (i) FPGA resources are easier to orchestrate, (ii) FPGAs can be offered as bare-metal resources, which simplifies the tooling needed, and (iii) FPGA failure only affects local resources, as opposed to potentially millions of nodes.

3.4.3 Network Connectivity - Use Cases. Table 1 shows two important trends. First, none of the production systems uses a secondary network. This is likely because of: (i) the cost and complexity of wiring a second network for potentially millions of nodes and additional networking hardware, (ii) the potentially limited scalability if direct FPGA-FPGA connectivity is supported, and (iii) high chip resource usage for building routers and securing the system. The second important trend is the relationship between network connectivity and use cases. Specifically, due to security and reliability constraints, systems that allow customers to offload their own applications do not support any direct network connectivity. Rather, this connectivity is only available if workloads are either internal or if the offering is an application where only a limited set of APIs are exposed to the customer. Research systems are distributed evenly across the different network connectivity options. We also note that newer research systems almost always have network connectivity, either primary or secondary. This helps scale the application across multiple FPGAs and achieve lower latency.

3.4.4 Intra-node Connectivity. Table 1 shows four major trends. First, in all systems with the exception of IBM CloudFPGA, FPGAs communicate with the CPU over the PCIe slot. This emphasizes the role of the CPU as being the core computational resource, whereas the FPGA is a complexity offload engine managed by the CPU. Second, FPGAs are almost always connected to some form of off-chip storage; typically, a DDR memory chip on the same board. Third, no production system currently offers instances with FPGA-GPU connectivity. To the best of our knowledge, none of the cloud providers has placed GPUs and FPGAs within the same node. For research systems, GPUs are being employed on the same node as FPGAs, especially for highly parallel, SIMD-like workloads and communication occurs over a PCIe switch. Fourth, in terms of FPGA-ASIC connectivity, only Microsoft v2 supports this approach, since the FPGA must transparently process packets for the traditional NIC. None of the research architectures connect an ASIC with an FPGA on the same node.

3.5 Potential Future Innovation

We identify areas of potential novelty that can be derived by traversing the categories in the taxonomy and by comparing different sub-categories with what is already present in Table 1.

Type of Boards: Potential novelty here is with modular boards that lie at the intersection of custom and commodity. Similar to what is commonly done with micro-controllers, semi-custom boards can be built by buying and connecting together off-the-shelf modules for different FPGA chips, memory chips, and interfaces (QSFP+, PCIe, etc.). This would allow providers to tailor boards to their specific requirements, reduce the penalties of designing a custom board (development costs, upgrade costs, probability of failure, time to market), and easily replace specific modules as needed (due to hardware failure or for regular upgrades).

Placement of FPGAs: While FPGAs have been used in high-end network switches, their role is typically limited to providing the performance and flexibility needed to support changing protocols. However, there is currently no system that leverages TOR switches where FPGAs are responsible for implementing the entire switch hardware.

Supporting such an architecture has a number of benefits. (i) Customer offloads: Customers could use these TOR FPGAs to compute in the network, e.g., for doing collective operations such MPI All-Reduce and Broadcast. (ii) Provider offloads: Providers could leverage these FPGAs to implement services such as metering, accounting, analytics, and packet filtering. (iii) Flexible networking: By combining FPGA based TORs with Bump-in-the-Wire FPGAs, a data-center-wide network could be created that does not rely on a standard protocol for communication. As a result, the communication latency could be reduced substantially. Alternatively, it may be possible to dynamically switch between different standard protocols based on the target workload.

Network Connectivity: A potential novelty here would be to support both Primary and Secondary network connectivity, either within the same FPGA, or through multiple tightly coupled FPGAs within the same node. This would effectively combine key benefits of Microsoft's Catapult v1 and v2, i.e., having ultra-low latency for rack scale communications through custom interconnects and still supporting data-center scale FPGA-FPGA connectivity.

Intra-node connectivity and Use cases: The connectivity between FPGAs and CPUs is typically done using the PCIe bus. This is because existing use cases define the role of the FPGA as an offload engine for the CPU. However, a potential novelty here is supporting sufficient low-level electrical coupling, such as the FPGA has read-modify-write access to the CPUs Baseboard Management Controller and firmware. This would effectively turn the FPGA into a management and security controller for the CPU and enable new *system administrator* use cases such as CPU firmware attestation.

4 USABILITY, SCALABILITY, AND PORTABILITY

Datacenters have become the enablers of many technologies and services. Search engines, personal assistants, streaming, video conferences, 5G, and telecommunications, along with the newly popular **infrastructure as a service (IaaS)** are all examples of such services. With the scale of these applications and services, deployment, provisioning of resources, and isolation all become tricky, especially with the loads of these services becoming more dynamic. For example, in cellular networks there may be a significant increase in peer-to-peer telephone or SMS traffic during holidays, but then during large sporting events there may be a spike in video streaming and cellular data use. Even if the total load is the same, different types of traffic must be processed differently making the load on individual functions dynamic. Furthermore, the principle of elastic computing assumes that instantaneous load is not constant over time [91], creating opportunities to scale down compute resources, putting excess resources in a low power state, or quickly scale up when high usage resumes. Similar examples of dynamic usage profiles exist in most datacenter use cases and have led to multiple studies into load balancing, traffic control, and usage forecasting to improve performance and minimize energy consumption [13, 76, 108, 131].

We start this section with an overview of the singularity of integrating FPGAs for elastic computing in datacenters. We briefly survey existing commercial systems, then dive into the details of the tools needed to program and deploy FPGA in the cloud. We discuss programmability of single node, integration of design in vendor cloud using shell, then virtualization. We then survey current development and discuss the outlook.

4.0.1 Resource Provisioning and Reconfigurability. The problem of datacenter provisioning has motivated deployment and orchestration platforms that allow datacenter managers to monitor their network and modify in real time the amount of compute resources given to each function. They make use of platforms such as Openstack [129] and Kubernetes [16], which instantiate, provision, and reconfigure datacenter resources in real time. These tools orchestrate the entire datacenter, but require individual components on each node to provision its resources. These tools implement resource provisioning in the cloud in several steps, including admission control, capacity allocation, load balancing, energy optimization, and quality of service guarantees. Unfortunately, these tools are currently only limited to provisioning CPU resources. For FPGA-accelerated clouds, they must be extended or re-engineered to include FPGA resource management. Besides the spatial nature of computing in FPGAs, reconfiguration along with bitstream management must be taken into account.

FPGAs are becoming more popular within the datacenter, as discussed earlier. But a critical aspect of FPGAs is their configurability, which could play a major role in making them suitable for accelerating such dynamic tasks, as their reconfigurability could naturally allow for the deployment and scaling these applications within the datacenter. Extending the typical datacenter orchestration tools with compatibility to FPGAs is thus natural, as it minimizes the required changes datacenter operators have to sustain. However, the bulk of the work needed for this support will happen on the FPGA virtualization side to make FPGAs compatible with orchestration tools, especially to support FPGAs as primary resources in the datacenter like CPUs, rather than just accelerators. This includes the ability to remotely provision the FPGA resources, split and isolate them between tenants, configure their network interfaces, and virtualize these network connections as needed.

FPGA programming is typically performed via an attached CPU. This is suitable with the current programming model of FPGAs as function accelerators for CPUs but is not suitable for standalone FPGA compute nodes as we envision in a datacenter. Depending on the FPGA type, the current programming interface is usually either JTAG, popular among PCIe attached FPGAs, or through an AXI memory mapped interface, as is typically the case in Xilinx MPSoC configurations. If FPGAs are to be primary compute resources in the datacenter, then their programming model requires more flexibility.

In a datacenter architecture where part of, or entire, racks are made up of FPGA boards, while others are made up of CPU based servers (Figure 1), it should be possible for independent FPGAs to receive partial or full bitstreams over the network. This ability is not new and has been previously proposed [69, 122, 150], though this would come with, among other things, security challenges that have not yet been overcome, and many of these alternate programming approaches come at the cost of area and or performance.

FPGA network configuration and virtualization can be achieved using a variety of ways, each with its benefits and downsides. The simplest solution would be to statically plan the network connections between FPGAs in a deployment, along with assigning their correct network configurations (i.e., IPs and MAC addresses). Static network configuration is used by various current datacenter FPGA implementations such as Microsoft's Catapult [27, 115] or Galapagos [139–141]. This requires prior knowledge about the deployment and lacks the ability to scale the deployment up or down if needed, and the flexibility of deploying these applications alongside others, which may cause conflicts between the network configurations, for example. Another possible solution is to use some common chaining protocols like VXLAN [94], MPLS [12], and **Segment Routing (SR)** [2]. These protocols may or may not (in the case of VXLAN) require the active participation of the switch as well as requiring modifications to the network interfaces at the compute nodes. These protocols can create virtual network connections over top the existing physical network architecture. This approach is reconfigurable, allowing a change of the virtual network whenever needed. However, a few drawbacks of these approaches are that they apply additional routing restrictions, latency overheads, and they require larger packet headers affecting throughput. These protocols could be merged into the underlying FPGA Shells used in the datacenter, but allowing these Shells (along with the switches, if needed) to receive and apply the virtual network configuration will be required. A third approach is similar to that followed by orchestration tools with CPUs today, which configures the network interfaces in a non-virtualized way, but instead has to keep each resource informed about the network interfaces of other resources (i.e., using a DNS table) and update them in the case of redeployment or scaling. This approach is not restrictive and allows any resource to connect to any other resource within its deployment. However, this approach is more complex in terms of implementation and integration in an FPGA Shell. Regardless of the approach adopted, MPSoC-based FPGAs are very appealing candidates, since the CPU on the MPSoC can act as an accelerator to the FPGA itself, offloading most of the network configuration tasks, along with interfacing with the central datacenter orchestration.

Existing datacenters circumvent these issues in a variety of ways. Microsoft Catapult, for instance, rebuilt the database infrastructure from scratch fitting FPGAs into the picture [27, 115]. However, this system lacks the quick reconfigurability that is possible in other scenarios, and complete rebuilding of datacenters, as they have done, is extremely costly and may be infeasible. Also, the wiring is built and optimized for their physical layout with no virtualization in the node functions or their networking. Instead, it would be ideal to be able to add FPGAs to existing infrastructures and reap their benefits without the upfront cost of a datacenter overhaul for widespread FPGA adoption to work. While this showed significant performance benefits without the orchestration or virtualization, this improvement is limited to one type of application and can cost more in the long run for changing the supported applications.

Amazon's AWS adds FPGAs to an existing infrastructure. It does not directly connect FPGAs to their datacenter network. Packets instead travel through the software layers of a PCIe-attached CPU before being forwarded to the network, and incoming packets need to traverse this stack in reverse. Similar frameworks are utilized by Huawei and Alibaba cloud infrastructures [155]. This approach resolves all the networking and virtual routing issues, as well as some security issues with network attached FPGAs, although this practice heavily limits throughput and increments latency. Even in full pass-through mode where the CPU is not asked to perform any processing or encapsulation, these architectures have network throughputs capped at about 40 Gbps [155]. However, it is possible to reconfigure what each FPGA compute node is capable of doing and can even offer FPGAs as a service by temporarily lending FPGA nodes.

4.0.2 FPGA Programming. Designing for FPGA consists of creating hardware circuits that will operate on the FPGA, using basic resources available on the device. The process uses **Hardware Description Languages (HDL)** such as VHDL and Verilog. It involves substantial hand and tedious work to implement all of the I/O functionality for the target board for data exchange. The migration of a design to a different FPGA board requires a complete redesign of the I/O handling.

There have been many early attempts to create software-like programming environments that were commercially available circa 2000, such as *Handel-C* and *Impulse C* [142]. These environments used a subset of C with some language extensions, data types, and functions that could be synthesized to hardware running on supported FPGA boards. They provided an abstraction that hides most of the hardware complexities from the application developer. Maxeler developed its **`Data Flow Engine (DFE)`** data flow programming model based on an extended Java called `MaxJ` and provided a full stack from the hardware layer to a software runtime and application layer interfaces. Again, the goal was to provide a software development flow that runs on CPU and FPGA platforms. The FPGA vendors have now developed their own programming environments, such as those using OpenCL, and are now moving towards more integrated environments that provide support for several languages and libraries for application spaces such as `Vitis` from Xilinx and Intel's oneAPI Toolkit. There are also efforts to build more of an open middleware stack such as the Intel `Open FPGA Stack` and the **`Open Programmable Acceleration Engine (OPAE)`** `Technology`.

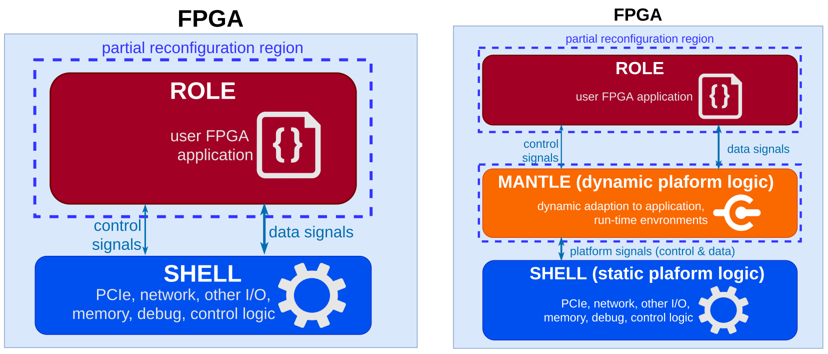
Supporting computing environments for FPGAs and making them more usable for application developers are important steps, but they still do not enable the use of FPGAs at the scale of clouds and datacenters.

4.0.3 Design Deployment: The Shell-role Architecture. The emergence of FPGAs as compute accelerators in the Cloud and other multi-user environments inevitably led to a split of the FPGA design into a user application programmed by the developers and a platform-specific part controlled by the infrastructure provider. This split of FPGA logic into a **cloud service provider (CSP)** controlled *Shell* and a user-controlled *Role*[1] allows the necessary introduction of different privilege levels within an FPGA design and potentially improves re-usability of user applications [122].

The general principle of the **Shell Role Architecture (SRA)** is depicted in Figure 3(a). Usually, the Shell contains all necessary I/O and control logic for data exchange between the user's hardware design and the rest of the system. On the one hand, this abstraction prevents the user from dealing with cumbersome I/O details. On the other hand, the central control of the infrastructure by the CSP simplifies the management and allows the CSP to implement resource allocation, isolation, and necessary security guarantees. Consequently, SRAs allow the CSP to stay in control of the Shell, while application logic is controlled by the user [122]. This SRA pattern is widely used across different FPGA platforms [27, 44, 122], each with its own level of details.

However, despite their simplicity and wide-usage, SRAs generate new problems when used at scale: The important character of the interface between Shell and Role "freezes" this interface after its release. To tackle this strong dependency while preserving the advantages of SRAs, recent research developed the notion of a three-layer architecture [121]. The general idea is to introduce an "adapter" layer between Shell and Role to provide forward and backward compatibility between Shell and Role. The authors call this middle layer *Mantle* and their overall idea is sketched in Figure 3(b).

[1]The terms *Shell* and *Role* were first introduced by Putnam et al. [115] and are now commonly used. Sometimes, the term *Hypervisor* is used instead of *Shell* because of a loose correspondence to the functionality of hypervisors in the software systems stack.


FPGA
partial reconfiguration region
ROLE
user FPGA application
control signals
data signals
SHELL
PCIe, network, other I/O, memory, debug, control logic
FPGA
partial reconfiguration region
ROLE
user FPGA application
control signals
data signals
MANTLE (dynamic plaform logic)
dynamic adaption to application, run-time envronments
platform signals (control & data)
SHELL (static plaform logic)
PCIe, network, other I/O, memory, debug, control logic

(a) The *Shell Role Architecture* principle

(b) The *Mantle Architecture* principle, to relax the strong dependency of the Role to low-level details of the Shell [190].

Fig. 3. Shell Role and Mantle architecture principles.

4.1 FPGA Resource Virtualization

While the shell-role architecture can be seen as a bare-metal deployment of design in FPGAs, the integration of resource management, including enforcement of elasticity in datacenter, requires some form of resource virtualization, particularly FPGA resources. This section discusses the issue of virtualization in datacenters that use FPGA. We discuss virtualization from two different perspectives. The first one is the hardware resource virtualization on FPGA devices, and the second one is the integration in hypervisors used to span new virtual instances in the cloud.

Virtualization essentially consists of creating abstraction layers over hardware components such as processors, memory, storage, and so on, with the goal of enabling resource sharing at the software level. While the introduced abstraction layers may decrease native performance, they hide hardware complexity, improve the flexibility and programmability of the underlying hardware and make the integration in cloud manage tool.

Processor virtualization is well investigated and currently relies either on instruction set translation or hardware support with technologies such as Intel VT [106]. Memory and disk virtualization essentially map the virtual space to physical locations in the physical space [132]. FPGAs differ from the previous types of hardware in that they do not execute sequential programs, instructions after instruction, but rather implement parallel circuits. Traditional virtualization techniques used in software are therefore not applicable to FPGAs. Abstracting FPGA resources at higher levels requires a new approach. FPGAs consist of configurable computing resources, storage, and interconnect, all immersed in a programmable interconnect infrastructure. Programming is achieved with HDLs. The long synthesis time makes it impossible to achieve binary translation needed in most virtualization frameworks today.

Abstractions are provided in FPGAs using two different approaches: Slot-based allocation and FPGA overlays. Slot-based allocation divides the FPGA into blocks where pre-synthesized circuits can be mapped at runtime, while overlays use processor-like configuration, which makes them more convenient for traditional virtualization.

4.1.1 Slot-based Allocation. Slot-based FPGA resource management [22] has been around since the early days of partial reconfiguration. The goal is to allocate FPGA area for the execution of pre-synthesized circuits at runtime through partial reconfiguration. The FPGA is divided into fixed areas, also called virtual regions, that will accommodate pre-synthesized circuits at runtime through

Table 2. Study of Recent Research in FPGA Virtualization Architectures for Cloud Infrastructure

	Shell Area Overhead	# FPGA regions	Spatial Sharing	On-chip Comm. Support	Data Width (bits)	Fmax (MHz)	Access Method	PCIe Version	Network Specification
Fahmy et al. [45]	7%	4	Yes	No	256	250	PCIe	Gen 3x8	-
Vesper et al. [151]	18%	4	Yes	Yes	32–256	variable	PCIe	Gen 3x8	-
Weerasinghe et al. [159]	21.7%	1	No	No	64	156.25	Network	-	10 GbE
Tarafdar et al. [140]	27%	1	No	No	43	125	Network	-	10 GbE
Mbongue et al. [98]	1%	4	Yes	Yes	-	227	PCIe	Gen3x16	-
Catapult [115]	23%	1	No	No	16–48	200	PCIe & Network	-	10 Gb SAS
Byma et al. [26]	19%	-	Yes	No	256	160	Network	-	10 GbE
Feniks [167]	13%	-	Yes	No	-	-	PCIe	Gen3x8	-
Mandebi et al.[103]	0.1%	6	Yes	Yes	32–256	1500	PCIe & Network	-	Fast Ethernet
Chen et al. [29]	6.46%	4	Yes	No	-	100	PCIe	Gen2x8	-
Asiatici et al. [11]	-	3	Yes	No	-	-	PCIe	-	-

partial reconfiguration. The slot-based allocation is extended for cloud FPGA resource management in References [4, 36, 141, 168]. These works leverage partial reconfiguration at the FPGA level and provide packages for integration in cloud management infrastructure. Table 2 lists some of the recently published research work in this direction. Companies such as VMAccell and Inaccel provide commercial frameworks to achieve the same goal.

The table classifies the architectures based on the shell area overhead, the number of virtualized regions per FPGA, whether spatial sharing is enabled or not, the presence or absence of on-chip communication support, the data width, the maximum frequency, and the access method to the virtual resources on the FPGA. For example, Chen et al. [29] divide each FPGA into four locations or "virtual FPGAs." The architecture multiplexes FPGAs in space and provisions hardware resources over PCIe. However, this architecture is limited in that it only allows the use of pre-built hardware functions without support for direct on-chip communication between accelerators. This results in middleware copy overhead for data movement between the accelerators of a user.

To minimize the data movement overhead, an on-chip interconnect can be used between virtualized hardware regions as a support to hardware elasticity [98, 103]. It increases the on-chip throughput and enables pipelined processing within the hardware domain of cloud and datacenter applications. Reference [101] evaluates the IO bottleneck in multi-tenant cloud FPGAs. Weerasinghe et al. [159] observe that network-attached FPGAs may offer lower latency and higher throughput compared to accessing accelerators over PCIe. Catapult [115], Tarafdar [140], and Weerasinghe [159] do not support spatial sharing of FPGA components between the workloads of multiple cloud users. They focus on FPGA time sharing. Table 2 also shows that the hardware footprint of the resource overhead that supports the virtualization is less than 30%. Since Shells occupy a small area on the FPGA, and considering the increasing size of FPGA devices, this overhead will become negligible.

4.1.2 FPGA Overlays. FPGA overlays have been developed as a promising way to increase FPGA programmability and productivity [133]. The approach commonly consists of designing a layer above the fine-grained FPGAs resources, thus hiding the complexity associated with programming low-level hardware components such as **look-up tables (LUTs), flip flops (FFs)**, signal processing blocks (DSPs), and so on. Moreover, it considerably increases productivity as it mitigates the long compilation time inherent in FPGA design flows. In general, FPGA overlays use **coarse-grained reconfigurable arrays (CGRA)** of processors that are programmed at runtime through software-level function calls. At the architecture level, FPGA overlays usually implement interconnect topologies allowing parallel processing and data exchange among processing cores [25, 65, 77, 85, 95, 97]. The software programmability of the coarse-grained processors makes it possible to develop efficient compilers for automatic mapping of sequential applications to overlays

[1, 92, 99, 160]. Beyond standalone utilization of FPGAs and FPGA overlays, cloud and datacenter architectures offer the opportunity to exploit FPGAs at a higher level of abstraction. FPGA overlays are most often immersed in the Shell to provision reconfigurable resources to cloud users.

4.1.3 Operating System Extensions for FPGA Virtualization. The integration of hardware resource, including those on FPGAs into virtual instances, assumes some sort of operating system support resources and protocols. There is a sizeable body on FPGA and integration in conventional operating system. While these work do not directly address cloud and datacenter, the low-level protocols developed would facilitate the integration of FPGA resources in hypervisor and virtual machines. Ma et al. [90] propose OPTIMUS, a hypervisor for a scalable shared system between cloud FPGAs and host CPUs. It implements efficient virtual DMA isolation via page table slicing, resulting in up to 7× improved throughput. FPGAVirt uses VirtIO for efficient communication between virtual machine and FPGA through in-kernel network stack [98]. Korolija et al. explore the application of traditional operating system resource abstraction to FPGAs [82]. The authors implement Coyote, an open source, portable, and configurable shell for FPGAs. It supports secure spatial and temporal FPGA multiplexing, virtual memory, communication, and memory management inside an uniform execution environment. Hategekimana et al. extends **Security-Enhanced Linux (SELinux)** security context to hardware IPs on multi-tenant cloud FPGAs [67]. This capability allows system security policies to propagate access control privileges expressed at the hypervisor level down to individual FPGA accelerators at runtime. Though operating system-level concerns such as abstraction and scheduling are being investigated in the literature, there is still need for operating system support for FPGA multi-tenancy in the cloud. Therefore, Khawaja et al. detail AmorphOS as a solution [78]. It enables cross-domain protection and replaces fixed slot-based allocation with elastic resource management to increase utilization and throughput. Moreover, recent research also addresses the compilation and execution of FPGA accelerators in cloud-based systems. As example, Landgraf et al. present SYNERGY, an FPGA compiler tool capable of generating controls to software execution, necessary to support core virtualization primitives such as suspend, resume, and program migration on FPGA [84]. Fumero et al. introduce TornadoVM, a virtual machine for applications acceleration on heterogeneous hardware at runtime [52]. It relies on JIT compilation to map kernels to adequate hardware accelerators. Using computer vision workloads, the authors show that TornadoVM allows achieving 7.7× speedup in an heterogeneous platform provisioning CPUs, FPGAs, and GPUs.

4.2 Using and Scaling of FPGA Clusters

Besides providing easy-to-use abstractions or virtualization of a single FPGA in a datacenter environment, the problem of programming and deploying multiple FPGAs is important as well for making FPGAs a scaling-up solution. The first problem of programming a cluster of FPGAs includes the question of how to abstract and use the communication links between the FPGAs and also how to link the kernels on multiple FPGAs together. The second problem of deploying an FPGA cluster contains the question of how to distribute, orchestrate, and manage an multi-FPGA application once it is built. In the following, we summarize major academic and commercial approaches tackling these problems.

4.2.1 The Galapagos Research Project: Providing a Middleware for Communication, Build, and Deployment. The Shell–Role interface simplifies using a single FPGA. At a cloud level, IP core placement within and across devices is needed. The Galapagos infrastructure [139–141] provides multiple layers of abstraction including a Middleware. The responsibility of the Middleware is to provide an easy abstraction for users to deploy an application across many FPGAs. This entails providing an abstraction for users to describe clusters, scale and replicate IP cores, as well as

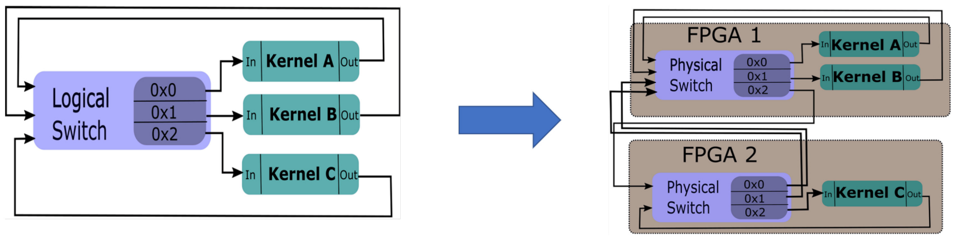


Fig. 4. The logical view of the cluster from the user's perspective is transformed with the Galapagos Middleware into a multi-FPGA implementation [140].

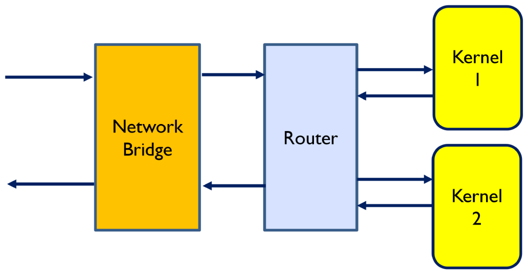


Fig. 5. Bridges and routers generated by the Middleware placed on each FPGA.

to make the routing simple between IP cores, agnostic to their placement. The agnostic routing removes the headache the user would have when designing IP cores and also gives the Middleware layer the flexibility to place IP cores anywhere. This flexibility is leveraged to provide reliability, as IP cores can be migrated between different physical locations, scalability as these IP cores can be replicated, and even ease of development as users can implement their IP core (that is part of a large multi-FPGA deployment) in software and incrementally migrate parts of their multi-FPGA application from software to hardware.

The Galapagos Middleware standardizes the interface between IP cores through AXI-Stream [9], a standard streaming protocol used by ARM and Xilinx IP cores. This streaming protocol is typically used within single FPGA systems in the Xilinx environment, with an optional bit-width configurable destination side-channel. This side-channel is used to route packets between on-chip AXI-stream IP cores. Galapagos Middleware takes this established streaming protocol for on-chip IP cores and expands this to multiple FPGAs. By leveraging standard AXI-stream, IP-cores do not need to be aware of the location of their destination IP cores. Figure 4 represents the user's view of the cluster versus what is actually implemented. The user-provided IP cores are unaware of the physical placement of these kernels.

The logical to physical translation is implemented by first performing a partitioning of IP cores across different FPGAs. The first iteration of this partitioning is simple, as a greedy algorithm is used. It attempts to fit as many IP cores on one FPGA before allocating another FPGA. This partitioner is a first implementation and can be improved independent of the rest of the Galapagos stack. Once a partitioning of all IP cores in the cluster has been established, the middleware generates the routers and bridges to connect all the IP cores within the cluster. Figure 5 shows the bridges used to establish on-chip and off-chip connections between IP cores. Since the formation of the bridges is automated through the middleware, user-provided IP cores can remain agnostic to the placement of all the other IP cores within the cluster. The autogenerated router on each

FPGA is aware of the placement of all IP cores in the cluster and can route packets to an IP core on the same FPGA or to a different FPGA through a network bridge. A network bridge translates AXI-stream packets to off-chip network packets. The network protocol is configurable to be any off-chip protocol the user wants as long as there is a bridge available to translate AXI-stream packets into network packets. Currently, bridges are available for Layer 2 Ethernet, UDP, and TCP/IP. However, as long as the user supplies the bridge this can support any off-chip protocol.

To support heterogeneous interaction between CPUs and FPGAs, a software library, called *libGalapagos* [137] has been created. The library provides software interfaces that exactly match the streaming interfaces used in hardware Galapagos kernels. Any software kernel that uses these interfaces can interact directly with any hardware Galapagos kernels, making it easy for a software kernel on a CPU to communicate with a hardware kernel on an FPGA. One benefit of this capability is that if a Galapagos kernel is described using HLS-synthesizable C++, then that HLS code can be wrapped using libGalapagos so it can be run purely in software. This enables a co-simulation environment where all the hardware kernels can be first tested in software to debug functionality. Once functionality is achieved, the HLS kernels can then be synthesized and run as hardware without any code changes. Whether such a kernel is run in software or in hardware is determined by specifying the version in a Galapagos configuration file.

4.2.2 The ZRLMPI Prototype: Providing a Programming Model for FPGA Clusters. Besides abstracting and managing the communication between different FPGA kernels, the behavior of this type of concurrency needs to be programmed as well, so a programming model for FPGA clusters is needed. In the past decade, there have been numerous approaches for this question (among others, References [37, 44, 55]), but there is no convergence to a common standard yet.

The trend towards network-attached FPGAs and thus the development of the FPGA from a co-processor to an “equal” stand-alone node, caused the community to revisit this search for a programming model for FPGA clusters. The ZRLMPI prototype [123, 124] tries to end this search by proposing to port MPI, the de facto standard for HPC, to FPGA clusters.

Bringing MPI to a heterogeneous cluster of FPGAs and CPUs involves two steps: First, a compiler is needed to compile a given program to different hardware. Second, a runtime environment is required, which implements the MPI APIs, start, stop, and synchronizes the execution between the nodes. ZRLMPI provides both. On the one hand, it provides a transpiler—or cross-compiler—that splits and optimizes an input MPI program to multiple programs, one for each physical node. This split breaks with the **“Single Program, Multiple Data” (SPMD)** notion of MPI, but this is necessary due to two reasons: First, in opposition to software, unused program parts for individual nodes results in wasted logic for FPGAs. Second, CPU and FPGA parts of the program need different compilers to build the binaries. On the other hand, ZRLMPI implements a runtime environment for FPGA and CPU nodes that ensures the synchronous message processing independent of the executing hardware. Hence, the CPU-FPGA cluster can be programmed and managed like a pure CPU cluster and existing programs could be ported easily to such heterogeneous clusters.

4.2.3 Commercial Programming Model: Maxeler’s Dataflow Engines. Maxeler, a pioneer in FPGA-based high-performance computing, provides an entire infrastructure consisting of hardware and software for dataflow applications. Its MaxCloud provides cloud implementation of a high-performance dataflow computing system. MaxCloud runs on Maxeler MPT compute nodes that consist of multi-core x86 CPUs with multiple Maxeler **Dataflow Engines (DFEs)**, large memory systems, and fast disks. It runs an industry standard Linux distribution. The DFEs are FPGA-boards attached to the CPU through PCIe.

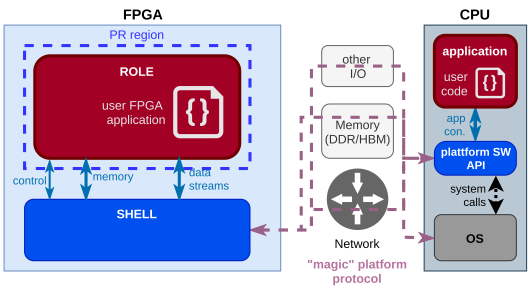


Fig. 6. Shell Role Architecture shares many common concepts.

4.2.4 Commercial Deployment Platforms: InAccel and VMAccel. InAccel offers a general-purpose accelerator orchestrator capable of integrating FPGAs from various vendors to simplify the deployment, scaling, and resource management of FPGA clusters. Their Coral software contains high-level APIs in C/C++, Java, and Python and a unified engine that supports every heterogeneous multi-accelerator platform. InAccel also provides a runtime specification that vendors can use to advertise system hardware resources to Coral. It aims to specify the configuration, and execution interface for the efficient management of any FPGA hardware resource, without customizing the code for Coral itself. In Coral, client applications may call accelerators on a local FPGA or remotely with their arguments and configuration parameters. Coral orchestrates the execution on a variety of execution environments, from bare-metal Linux servers to containers inside a Kubernetes cluster.

VMAccel, a company in stealth before January 2021, operates a large, multi-tenant FPGA IAAS cloud platform based on Openstack, Docker, and Kubernetes. This platform supports virtual machines, bare metal, and containerized FPGA acceleration options. VMAccel's cluster consists of over 1,000 FPGAs and is designed from the ground up with FPGAs in mind. VMAccel removes restrictions on hardware and offers direct access to JTAG and QSFP interfaces for clients; their offering is designed to be customizable and flexible to accommodate varying user needs and workloads. The company plans to implement disaggregation technology by Q4 2021 to offer users even more flexibility in hardware composition.

4.3 Cloud FPGA Usability – Current Needs and Outlook

Considering the usability, scalability, and portability, we notice the progress that has been made in the past years to decrease the barriers of leveraging FPGAs in datacenter and cloud environments, but we also see the need to further improvements. In particular, we identify two areas: First, to enable "seamless" portability, the application interfaces within the FPGAs need to be standardized. Second, FPGAs should be integrated further into classical software stacks and debugging of (distributed) FPGA applications has to become simpler. Both areas are addressed below.

4.3.1 Enabling Portability: Standardizing the Application Interface. From a general point of view, FPGA applications do not exist in empty space: They are part of a more or less complex communication schema with other FPGAs or CPUs, as discussed in Sections 3 and 4.0.2. If we analyze all the earlier mentioned SRAs from this perspective, then we find some general patterns, besides the pure split of logic into a Shell and a Role. This high-level generic design of SRA platforms is shown in Figure 6. Every Role is controlled by some configuration and control registers or signals such as `start` and `stop`, `node_id` or virtual I/O pins. Besides this, two types of communication channels can be found:

(1) Address-based communication: Every data belongs to an address in an address space. This address space is at least partially shared with the CPU. So, the content of the data can be interpreted based on the address the data belongs to. This communication type is used by PCIe-attached or memory-synchronized FPGAs such as openCAPI systems or OpenCL platforms.
(2) Stream-based communication: Here, every data belongs to a stream of data without further meta data such as addresses. In some cases, the data stream is accompanied by other streams that indicate the origin or destination of this stream. Consequently, the data can be interpreted only with the knowledge of the specific physical connection the data came from and to some extent, the content of parallel meta data streams. This type of communication is used by network-attached FPGAs—such as the IBM cloudFPGA project [3] or the Galapagos cluster [44]—or for some fabric abstractions.

The synchronization between these FPGAs and the other devices, e.g., the CPU is up to the *"magic"* protocol of this platform, as sketched in the middle of Figure 6. "Magic" in the sense that the details of this synchronization are not important for FPGA application developers. FPGA application developers are more concerned with the high-level capabilities (what can be done) as opposed to the low-level implementation details (how the synchronization between FPGAs and CPUs is done).

Therefore, since most FPGA cloud platforms share the same architectural pattern with the SRA and due to the observation that interfaces of the data path are more important to users of cloud FPGAs than the details of how the data path ends up in the FPGA, we think it is possible to describe a standard interface for cloud FPGA applications at this level and leave the details of the "magic" to each individual platform. The SRA architecture has been around since the early days of FPGA integration in desktop systems. Examples include the ESM [22], Walder and Platzner OS [153], and the Celoxica series 1000PP and 2000 [21], and many more. This standard interface can also apply to extensions like the Mantle architecture [121], because the adapter could be "below" or "above" a common interface. There is no reason to doubt that progress will continue in this direction and that future systems will rely on a more portable, standardized SRA as interface for seamless deployment of design in the cloud.

Every system designer has implemented a variation with a library on the hardware and software side to facilitate the development of hardware and software systems. A semblance of standardization has been developed over the years by FPGA vendors in the form of board support packages and libraries. While these packages simplify the design and integration of a single FPGA in a desktop or standalone mode, extension to cloud computing platforms has yet to be achieved. Increased adoption of FPGA infrastructure requirements in the cloud will help achieve this goal. In this direction, we are starting to see new CSP players such as VMAccell and Inaccel that provide tools and architectures to seamlessly expose FPGAs to users in numerous possible cloud computing paradigms. This progress will continue with companies offering more flexible, efficient solutions that increase resource usage, with single or multiple tenants. The Cyborg project discussed in Section 4.3.2 is an effort by the opensource community to facilitate the integration of non-conventional processors such as FPGA, GPUs in datacenters. We expect these developments to gain traction and lead to better design tools and efficient management frameworks that will accelerate adoption of FPGA in datacenters.

4.3.2 Cyborg Project: Managing FPGA Accelerators in OpenStack Clouds. OpenStack is a free and open standard for cloud computing platforms. It allows easy deployment and management of infrastructure-as-a-service for public and private clouds. The development of OpenStack is organized into projects. The Cyborg project specifically addresses the integration and general management of hardware accelerators such as FPGAs, GPUs, SoCs, and so on, in OpenStack, enabling the provision of hardware acceleration as a service. Cyborg provides REST APIs to list, create, update, and delete devices hardware devices in a cloud infrastructure. Enabling FPGA acceleration in OpenStack opens opportunities for researchers to explore use cases of FPGA technology in the cloud. In addition, it enables cloud providers with the ability to provision heterogeneous architectures with minimal effort and cost. However, services still need to be developed to program devices at runtime, manage bitstreams, update and migrate shells, enable debugging designs with the insertion of probes, monitor on-chip sensors, and so on.

4.3.3 Simplifying Usability: Increase FPGA Integration and DevOps Support. Besides the limited portability of FPGA applications, which leads to strong dependencies to specific hardware and cloud vendors [121], FPGA applications need still be developed from scratch, most of the time. Usually, one needs to write a new FPGA application in a tool that depends on the used FPGA platform. In addition, debugging is often still done with debug-probes at "wire-level." This is in strong contrast to the usability of other accelerator platforms, where in the best case, the application developer does not notice the usage of heterogeneous hardware by the toolchain. For example, if using today's deep-learning frameworks, then GPUs are seamlessly integrated and can be used with one or two lines of code [143]. The usage of one or many FPGAs must become as simple as the usage of GPUs in this examples.

As discussed in above sections, there are many efforts that explore this research direction. For example, it is possible to deploy a given FPGA binary on a multitude of cloud FPGA platforms with a single line of code or to compile and deploy an application of a given framework on a specific platform [123]. However, to reach the desired level of simplicity, both categories need to be combined. It is not sufficient to deploy a fixed functionality on a flexible platform or a flexible functionality on a fixed platform. For FPGAs, to be of use for complex scientific and commercial applications, it must be possible to develop, debug, and deploy complex distributed applications easily and independently of specific platforms. This includes the development of vertically integrated design environments [63, 104, 107, 111] and supporting more debugging possibilities [116]. We notice that this problem is subject of many active research projects and we appreciate that this focus is starting to be reflected in academic conferences and workshops.

5 SECURITY OF FPGA ACCELERATION IN CLOUD AND DATACENTER ENVIRONMENTS

The availability of FPGAs in cloud datacenters has opened up unprecedented levels of application flexibility and performance, although security is an important consideration. The development of Spectre [81] and Meltdown [87] demonstrated how hardware vulnerabilities can be leveraged at the software level to successfully launch attacks on infrastructure. FPGAs add another infrastructure resource to the cloud that must be free of vulnerabilities to prevent malicious activities. Unfortunately, a user's ability to implement any logic function provides unique avenues for malicious attacks on other cloud users' applications and data and the cloud infrastructure itself [74, 148, 165]. In this section, current and future threats to cloud FPGA security are discussed with an eye towards prevention of existing and future potential attacks.

5.1 Cloud FPGA Security Overview

Many of the threats facing cloud FPGAs are similar to threats that have long been prevalent in microprocessor-based cloud computing [34, 109, 136]. Multi-tenancy can lead to side channels in which confidential user data is stolen [117, 169], or covert channels in which otherwise logically isolated users are able to transmit sensitive information between each other [58]. Software and interface manipulation can also result in a denial of service for cloud users or even the infrastructure. Malicious bitstreams uploaded to the FPGAs in the data center can lead to faults or incorrect results being generated [62, 113]. Unlike previous software-based attacks, cloud FPGA attacks often include digital hardware manipulation whereby the FPGA logic is configured in a malicious way that can lead to accelerated device wear-out, short circuits [8, 15], or performance degradation. In addition to the existing single-tenant deployments already available from vendors such as AWS, multi-tenant cloud FPGAs are being actively researched. In multi-tenant FPGAs, different users share the same FPGA, making attacks between users more dangerous. The use of multi-tenant cloud FPGAs in the future will expand attack surfaces, necessitating more comprehensive security plans.

The threats associated with cloud FPGAs include malicious FPGA tenants in a single-tenant setting, co-tenants in a multi-tenant setting, nefarious cloud service providers, malicious and compromised tools, or intellectual property cores that may try to attack or steal information [74]. The most striking of these threats is related to malicious co-tenants in the multi-tenant setting.

In this section, we review cloud and datacenter FPGA security threats. Vulnerabilities broadly target the FPGA chip, the interfaces between the FPGA chip and other components on the FPGA board or the server where the FPGA boards are located, and the system software used to manage the FPGA bitstreams (e.g., loading designs onto the FPGA) and access data generated by the FPGA. Other infrastructure components related to FPGAs, such as the shared power distribution within the server, can also be a source of security vulnerabilities. Current and potential future attacks are reviewed in this section in addition to existing and potential countermeasures.

5.2 Cloud FPGA Usage Models

In this subsection, we categorize FPGA usage models in the context of the architectures and interfaces introduced in Sections 3, 4.0.2, and 4.3.1. In many respects, the level of security available to cloud FPGAs is dependent on the architectural model used and the associated interfaces.

Unlike in software-based **virtual machine (VM)** instances, with FPGA-accelerated instances, users naturally engage with the hardware at a lower level. In software-based VMs, a hypervisor provides an encapsulated environment for user execution that largely consists of virtualized CPUs, memory, storage, and network interfaces. This encapsulation facilitates memory protection and network isolation. Today's commercial single-tenant cloud-based FPGAs do not abstract or virtualize fundamental logical resources such as LUTs, block memory, or multiply-accumulate units. This limitation is currently present for two reasons. First, abstracting the underlying FPGA fabric with a template incurs non-trivial overhead in terms of usable FPGA logic density. Since many of the performance benefits of using an FPGA hinge upon exploiting maximal parallel processing, the efficient use of the available logic is very important. Second, FPGA virtualization is difficult. For example, adding logic to stop an arbitrary circuit mid-computation, save the intermediate state, and restore it at a later point presents design challenges and is expensive to implement in terms of FPGA resources and performance. For example, although block memory and flip flop contents in Xilinx UltraScale+ FPGAs can be streamed out of the device at 6.4 Gbps using a configuration interface, the largest devices contain nearly 500 Mbit of RAM and 3.5 million flip flops [162]. Also, pipeline registers in multiply-accumulate blocks cannot be easily retrieved via the interface.

Difficulties in implementing FPGA virtualization affect how they can be offered in the cloud. For example, unlike traditional software-based servers, the fine-grain context switching of user Roles is not practical. Rather, today's commercial cloud providers assign an entire FPGA or more than one entire FPGA to a user for the duration of their computation, forming a single-tenant setting. For example, in the case of an AWS F1 instance, a user is assigned a VM on a given server that has exclusive access to N FPGAs, with the VM resources and number of FPGAs scaled to the size of the purchased instance. The user first programs their circuit(s) onto the FPGA(s) and then can utilize them at will, within the constraints of what the Shell will allow. When the user's instance terminates, or the cloud provider elects to reclaim the instance, the software-side and FPGA resources will be returned to the allocation pool to be re-assigned to a different user, who will then gain exclusive access to the FPGA(s) for themselves. The provider is expected to properly erase all the resources before allocating them to the next user. For instance, the DDR memories provided on the FPGA boards in AWS F1 instances get overwritten before an instance is allocated to a new user.

In the near future, it is likely that large FPGAs that could be shared by multiple users will be installed in datacenters. Sharing of the same FPGA by multiple users in the multi-tenant setting is an example of the multi-tenant spatial model. In this case, FPGAs will have enough resources to subdivide each device into multiple regions that can be assigned to independent users and a single Shell would manage multiple Role regions, using dynamic partial reconfiguration to install a given user's Role at runtime. As a result, a single device would need to connect to multiple different VMs and communication to the device would need to be multiplexed from a specific VM to a particular Role. Since the circuits for different users reside on the same FPGA simultaneously in different Role regions, significant new security issues may arise (e.g., side-channel attacks, fault injection attacks, attacks on shared memory and I/O infrastructure [74]).

In addition to accelerator-focused deployments, such as AWS F1 instances, in which users access the FPGAs via PCIe as a co-processor, there are other models of intra-node FPGA communication. As discussed in Section 3, bump-in-the-wire architectures directly connect FPGAs to a network so all network communication flows through FPGAs. This gives faster access to network data, but, effectively, data from many users may pass through an FPGA instance controlled by a malicious user.

A further possible deployment includes the CPU and FPGA together on the server, possibly as different chips on the same motherboard or with the CPU and FPGA sharing the same **system-on-chip (SoC)**. With the acquisitions of Altera by Intel and Xilinx by AMD, FPGAs may become directly integrated into large servers as fundamental computers, not just as accelerator cards. It is also expected that FPGAs will become parts of the processor dies themselves. This opens up even further security threats, as, for example, the FPGA may have access to the memory bus and the coherence messages being sent between CPUs.

5.3 Cloud and Datacenter FPGA Attack Categories

The co-location of multiple tenants inside cloud-based FPGAs, or having multiple dedicated FPGAs per user but sharing a server between different users, leads to numerous resource sharing threats. Resource sharing can occur within an FPGA in a multi-tenant setting, between an FPGA and associated host CPU and DRAM, or between different FPGA boards within the server in a back-end configuration. Figure 7 provides a summary of resources that can be shared. The vulnerable resources include the system and individual chip **power distribution network (PDN)**, FPGA logic and wiring resources, Shell logic provided by the cloud provider, DRAM memory on the FPGA board, and the host CPU. Further, a system bus in a storage-attached system may be shared with the host server, which can include vulnerable DRAM memory that can be accessed by multiple users via **direct memory access (DMA)**.

The following subsections detail current and future threats that are rooted in the shared resources. These threats have been broadly grouped into the following categories:

(1) Single-tenant attacks;
(2) Multi-tenant attacks;
(3) Node-level attacks;

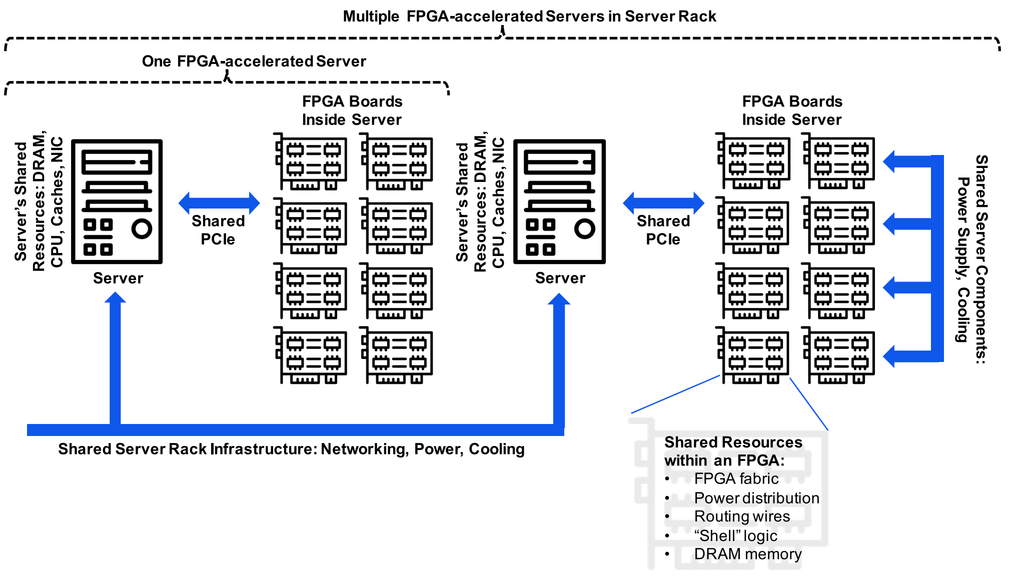

Multiple FPGA-accelerated Servers in Server Rack
One FPGA-accelerated Server
FPGA Boards Inside Server
Server's Shared Resources: DRAM, CPU, Caches, NIC
Server
Shared PCIe
FPGA Boards Inside Server
Server's Shared Resources: DRAM, CPU, Caches, NIC
Server
Shared PCIe
Shared Server Components: Power Supply, Cooling
Shared Server Rack Infrastructure: Networking, Power, Cooling
Shared Resources within an FPGA:
• FPGA fabric
• Power distribution
• Routing wires
• "Shell" logic
• DRAM memory

Fig. 7. Diagram of hardware resources that are shared in FPGA-accelerated servers and among servers in a server rack; each shared resource could be a source of a possible attack or information leak.

(4) Datacenter system-level attacks;
(5) Bump-in-the-wire attacks;
(6) FPGA and CPU on same die, motherboard, or SoC attacks.

5.4 Single-tenant Attacks

These attacks affect FPGA computation in which one user at a time is using the FPGA. Today's cloud-based FPGA deployments exclusively follow such a single-tenant model.

5.4.1 Single-tenant Temporal Channels. Allowing users to load custom FPGA designs opens up the opportunity for malicious users to sense the environment on or around an FPGA. In particular, temperature affects operation of many electronic circuits: It affects signal delay in wires, transistor switching, and capacitor decay (used in DRAM memories, for example).

Previous work [145] has shown that it is possible to create thermal covert channels for communication between users. In particular, even in a single-tenant FPGA setting, users share the FPGAs in time: Once a user is done using an FPGA, it is assigned to a different user. Other temporal attacks may be possible, for example one FPGA user may generate substantial PCIe traffic, which reduces the bandwidth quota assigned to the FPGA. When the next user design is loaded onto the FPGA, he or she may still experience a reduced bandwidth quota from the prior user. Any system component that maintains state could be abused for temporal attacks if the state is not properly erased. The state could be physical (e.g., thermal state) or logical (e.g., PCIe bandwidth allocation). For example, old DRAM values may be read by subsequent users. In practice, this attack is not currently feasible, since vendors (e.g., AWS F1) actively clear DRAM between users.

5.5 Multi-tenant Attacks

Multi-tenant attacks impact FPGA computation in which multiple users access the FPGA or shared resources (e.g., memory) at the same time. Most current research focuses on this setting, as resource sharing can lead to a diverse set of attacks.

5.5.1 Multi-tenant Voltage Attacks. FPGAs in the multi-tenant spatial model are particularly vulnerable to supply voltage attacks. All logic implemented in a commercial FPGA shares a **power distribution network (PDN)**. As a result, the PDN can be manipulated to perform a broad spectrum of attacks, including the implementation of covert channels [58], the side channel theft of encryption keys [128, 169] or other information, and fault injection [62, 113]. In extreme cases, large power waster circuits can consume sufficient power to drive the FPGA board into reset [114]. A full summary of multi-tenant voltage attacks is available [74].

Although multi-tenant voltage attacks have been widely explored, there are several additional vectors of interest. It may be possible to localize failures to other specific devices on a board through clever on-FPGA voltage manipulation. Remediation to prevent such attacks is also needed. AWS F1 includes a power monitor that will shut down an entire FPGA that consumes excessive power [61]. It has been shown that voltage sensor networks instantiated in an FPGA can quickly detect attacks [113]. This information could be used to quickly suppress offending attack circuitry. Finally, it may be possible to architect FPGAs so PDN regions are isolated or a global "kill" signal can quickly suppress an FPGA region suspected of malicious activity.

5.5.2 Multi-tenant Attacks on Shared Block RAMs. Allowing a user to intentionally cause write collisions in FPGA dual-port block RAMs can induce voltage and temperature fluctuations and result in circuit faults [5]. The faults may affect other users. Although unexplored, it may be possible to abuse other modules (e.g., communication blocks, DSP blocks) within an FPGA to cause faults or unintended behavior for other users.

5.5.3 Multi-tenant Crosstalk Attacks. The crosstalk of routing channel wires in multi-tenant FPGAs has been extensively explored [60, 117]. In particular, static values on routing wires (logical 1 vs. 0, which corresponds to a charged or discharged routing wire) affect the propagation delay on immediately adjacent channel wires. It is possible to sense the state of the adjacent wire and use the information as a covert channel or side channel. Crosstalk has been used to steal AES [117] and RSA [118] cryptographic keys.

Crosstalk could also be used to target Shell logic, or wires associated with communication buses, such as the AXI bus within the FPGA used to connect different modules. The shared internal FPGA bus could potentially be snooped to extract data from different users at different times. The existence of crosstalk necessitates physical separation so different users' modules or the Shell are spatially far apart to prevent crosstalk. User-level control of logic placement may need to be prevented in multi-tenant scenarios so attackers cannot chose custom locations for their logic, e.g., next to the Shell's communication buses.

5.5.4 Multi-tenant Attacks on DRAM. Each cloud FPGA board typically has its own attached DRAM that augments the shared DRAM on the server. Both sets of DRAM storage could be susceptible to Rowhammer-style attacks [79] in which one user attempts to modify the DRAM of another user. This issue is particularly worrisome for multi-tenant FPGAs in which multiple spatial co-tenants have immediate hardware access to the same DRAM modules, although this has not yet been demonstrated in practice.

Since cloud FPGAs contain multiple DRAM chips (e.g., AWS F1 includes four DRAM modules, or chips, per FPGA), it may be possible to assign data from different tenants to different DRAM chips, mitigating Rowhammer attacks by tenants. If more tenants than DRAM chips exist in a bank, then tenants could be assigned to non-adjacent DRAM locations to prevent attacks. The development of a DRAM controller and DRAM allocation and access policies for multi-tenant FPGA settings could also be considered.

5.6 Node-level Attacks

These attacks influence computing components located outside of the FPGA. Many of these attacks are possible already in single-tenant cloud settings.

5.6.1 Node-level Attacks on Server's DRAM. An FPGA Role could influence memory access for software components that are not running on the FPGA. An FPGA Role or compromised Shell could induce Rowhammer-style attacks on the server's DRAM, since the FPGA has DMA access to the memory on the server. FPGA accelerators typically use memory mapped buses to access DRAM. Cloud providers must implement ad hoc memory protection units to prevent malicious accesses, especially when resources are shared by another user.

While current FPGA node implementations typically include accelerator cards, FPGAs will eventually be more tightly integrated into the server. Once integrated, FPGAs may have direct access to coherence messages and the ability to read and write DRAM, bypassing the PCIe and DMA used today. This opens up the possibility of attacks on the coherence messages or the modification of the caches' contents, since FPGA operations will affect the CPU's memory.

5.6.2 Node-level Attacks on a Server's Power Distribution Network. Shared power distribution networks are not only a concern for FPGA co-tenants, but also for components that share a system-level power supply. Significant changes in FPGA power usage on a data center board can be sensed by an FPGA on a different board if both boards use the same power supply [59]. While covert communication using a shared power supply has been demonstrated in a lab setting, it has not been shown using commercial public cloud hardware. A natural extension may be the creation of a covert channel between servers or between server racks, although these channels may be blocked by power supply filtering circuits.

It has also been shown that the encryption key of a cryptographic algorithm running on a microprocessor can be extracted using an FPGA if both devices share a supply voltage [169]. This approach could also be used to create a covert communication channel. Similar channels could be developed from CPU to FPGA or from GPU to FPGA, as previously demonstrated [59]. In both cases the FPGA could be the receiver, as it is able to measure voltage changes.

5.6.3 Node-level Bus Attacks. The Shell could be attacked by an FPGA Role to stall the bus connecting the FPGA to the host server, creating a denial-of-service attack. This attack could potentially be achieved by manipulating messages sent to the bus controller on the FPGA side or by causing voltage attacks that in turn create faults in the bus controller that stall the bus.

5.7 Datacenter System-level Attacks

These attacks affect more than one node in the datacenter.

5.7.1 FPGA Instance Fingerprinting. Users can create malicious FPGA designs that steal cloud infrastructure information. For example, it is possible to fingerprint FPGA instances using DRAM **physical uncloneable functions (PUFs)** [146]. Given the assumption that a given physical DRAM module will not be moved from one FPGA board to another, fingerprinting DRAM modules is equivalent to fingerprinting the attached FPGAs. An FPGA may also be fingerprinted using dedicated PUF circuits inside the device [149]. Any circuit with a delay path, such as **time-to-digital converter (TDC)** circuit or existing hard **intellectual property (IP)** modules for **digital signal processing (DSP)**, could potentially be used as a PUF. Recent work [144] has shown that PCIe bus contention can also be used to fingerprint cloud infrastructures. Such contention-based attacks could be extended to also cover network cards, which are attached to the bus.

Fingerprinting attacks allow users to learn the number of FPGAs that are available and their physical co-location within servers. Although fingerprinting attacks have focused on individual FPGAs and servers, they could be extended to fingerprint server racks or whole data centers, possibly using server thermal or voltage fluctuation patterns.

5.7.2 Potentially Destructive Attacks. Any attack that can physically damage the data center hardware is very serious. Most destructive attacks would likely use excessive power, although, as mentioned eariler in this section, power usage is already checked by cloud providers such as AWS, and a user's design is stopped if an FPGA crosses a power usage threshold. In general, destructive power attacks may not have an instantaneous effect. Accelerated hardware degradation via less aggressive power attacks may take longer but can be equally destructive and harder to detect.

5.8 Bump-in-the-wire Attacks

Risks associated with cloud FPGAs impact system architecture and user access models. Bump-in-the-wire architectures have unique security challenges, since FPGAs are directly connected to the network and process network packets from many users. Exposing FPGA resources in this manner allows reconfigurable hardware application developers to exploit the natural ability of the hardware to process at network line rates. Furthermore, bump-in-the-wire (or Smart-NIC) architectures can support new capabilities while minimizing the impact of host software.

Malicious hardware in a bump-in-the-wire FPGA can easily and quickly flood the network with nuisance packets. Although modern datacenters have systems to both monitor for problems and take corrective actions, which limit the potential scope of issues that do arise, there is an inherent delay between the start of a potential problem and the implementation of a solution. Thus, network-connected FPGAs can, at the very least, create transitory local service issues. In a datacenter where network-connected FPGAs are ubiquitous, a well-coordinated attack could have a systemic impact. This issue is further complicated in bump-in-the-wire architectures, since blocking traffic from an ill-behaved FPGA also blocks access to the host server.

Future research in bump-in-the-wire architectures could consider the moderation of network packet injection and the careful scanning of network requests. Effectively, a broad class of network security approaches could be applied to this class of cloud FPGA architecture.

5.9 Comparison with Existing CPU- and GPU-based Threats in Cloud Computing

Most hardware-related CPU threats faced by cloud providers involve information leakage and side-channel or covert-channel attacks. The information leakage is usually due to resource sharing (e.g., via shared processor caches) [135]. These sharing-based attacks have some similarity to possible shared bus [144] and DRAM attacks involving multiple FPGA tenants. For users that demand security, cloud providers can allocate full CPU cores (to avoid level 1 cache sharing and related attacks, for example) or even allocate full servers to a single user. This limits any sharing of CPU resources, but also clearly prevents resource sharing that could lower costs or improve utilization. Researchers have also demonstrated GPU-based cache attacks that can leak information between users due to shared resources such as caches [39]. FPGAs also suffer from resource sharing and contention, but also have new types of FPGA threats due to ability to create malicious FPGA circuits that can measure temperature, measure voltage changes, observe decay of the DRAM modules, and so on.

5.10 Cloud FPGA Security – Current Countermeasures and Needs

Looking forward, security for cloud FPGAs can be categorized into four axes of protection: FPGA intellectual property protection, system-level security, runtime user support, and FPGA device architecture improvements. Each of these axes is addressed below.

5.10.1 FPGA Intellectual Property Protection. Although FPGA design disclosure to the cloud vendor helps limit the loading of malicious bitstreams to the cloud, it exposes a user's intellectual property to the cloud vendor. AWS and Microsoft, for example, do not allow users to directly create FPGA bitstreams for use in the cloud. Zeitouni et al. [166] have developed a multi-step procedure to protect cloud infrastructure from damage by a malicious FPGA without disclosing the user's netlist to the cloud provider. This goal is accomplished using a trusted execution environment, trusted Shell, and mutual authentication using a PUF for key generation. This approach could be extended by considering FPGA architecture enhancements for trusted execution and key storage.

5.10.2 Preventing Attacks Using System-level Security. Protection for FPGAs, memory, transport buses, and OS support are needed to create a secure system. These aspects of cloud FPGAs use could potentially be vulnerable, and a security approach must consider how to protect the whole system. System-level integration increases the difficulty of securing an entire system in the face of remote hardware attacks.

The integration of FPGAs in cloud systems induces temporal and/or spatial interaction among cloud tenants. Without guarantees of isolated execution, this sharing can lead to scenarios where shared accelerators act as potential covert channels among software guests that reside in different security contexts. Remote attacks have been demonstrated by manipulating FPGA hardware (e.g., the PDN), as explained earlier in this section. Domain isolation in hardware can be addressed from a software perspective using an operating system. Emphasis can be placed on a system's ability to limit privileges associated with executing processes to contain the scope of damage that can result from the exploitation of application vulnerabilities [88, 125]. Access control used in SoC and FPGA-based embedded systems have assumed the availability of *reference monitors* to manage hardware access [23, 71, 119]. Similar approaches could be provided in a multi-tenant cloud FPGA scenario.

A comprehensive solution to domain isolation that cuts across hardware and software has been proposed [66, 83, 96, 102] in which an FPGA is shared among virtual machines. The isolation framework guarantees that hardware modules execute and reside in the same security context as the "caller" VM. Similar operating systems and hypervisor approaches could form the basis of multi-tenant isolation.

5.10.3 Runtime User Support. The Shell, cloud software, and cloud management systems used for cloud FPGAs will need to significantly change in the coming years to support the multi-tenant spatial model. This prospect raises multiple new FPGA-specific questions for cloud providers. Starting at the beginning of the work flow, a user's Role must be dynamically programmed to a specific Role region. However, the cloud provider may not elect to expose an API for the user's VM to perform this programming for itself. Doing so may unnecessarily expose a vital part of the system, the dynamic partial reconfiguration port, to potential abuse (e.g., a denial of service attack that prevents or slows other Roles from being programmed onto the device). Rather, the cloud provider may modify their management system to program the Role region in advance, only starting the user's VM after this operation is complete. This model also simplifies the Shell, software drivers, and VM facilities that must be provided, since no safeguards are needed to ensure that a given VM can only reprogram a specific Role region. In this case, the cloud management software will send commands to the hypervisor, similar to what occurs when a VM is allocated. This model may depart from the experience that some users may expect or need. For example, disallowing a user from reprogramming the FPGA directly changes the potential use-cases that can be supported and complicates the recovery process when a Role that has become unresponsive due to a bug in the user's code.

Once the user's Role is active on the device, the user's VM must have exclusive access to the Role. Although the VM, driver, and Shell modifications needed to implement this access may be similar to SR-IOV for devices that use PCIe, this approach may require different techniques for devices that do not, e.g., network-attached standalone FPGAs. Furthermore, the Shell has responsibilities that extend beyond communication with a user's software—it must virtualize all external resources. In the case of a resource like external memory, the Shell not only needs to segment the address space for different Roles, it also must make fairness or quality of service considerations between Roles. In the case of a resource like a network connection, this action likely requires support for virtual networking to enforce isolation across the network and quality of service considerations from both Roles and upstream switches.

Put together, the basic requirements needed to support the multi-tenant spatial model of execution indicate that all parts of the cloud eco-system will need to become more sophisticated. However, this added complexity presents new issues. For example, adding complexity naturally also increases the potential for security vulnerabilities. In the case of the Shell, the necessary changes also affect utilization. For example, the value proposition of the multi-tenant spatial model is the availability of hardware for different users, thereby minimizing idle FPGA space. However, if significant additional logic must be added to the Shell to support the multi-tenant spatial model, then any potential advantage may be nullified.

5.10.4 FPGA Architecture Changes. In general, commercial FPGAs were not designed for simultaneous use by multiple independent users (the multi-tenant spatial model). To prevent attacks, additional security measures may need to be added to FPGA devices. One possible architectural enhancement is the isolation of PDNs to FPGA regions to avoid on-FPGA voltage attacks. Such isolation, which is common in multi-core microprocessors, may also allow for more flexible voltage scaling on a per-region basis. FPGAs could also be instrumented with additional hardened voltage and temperature monitors that can be read quickly. This information could then be used to isolate and suppress voltage, thermal, and other attacks before they can cause damage. Finally, new techniques to dynamically reconfigure FPGA regions and save state more quickly may be needed to fully support rapid on-demand FPGA use in the cloud.

Improved bitstream security may spur interest in FPGA use as a root-of-trust in the data center. Although modern FPGAs contain PUFs used in key management, and specialized blocks for hashing and decryption of bitstreams, full breaks of modern FPGA bitstreams are commonplace [41]. Increased bitstream security will allow for a wider role for FPGAs in boot-strapping systems, incorporating roles that have historically used **trusted platform modules (TPMs)** to validate firmware.

6 APPLICATIONS

In this section, we focus on customer applications that in most cases are run on public cloud services. We divide these applications into two categories, namely, infrastructure-based and other applications.

6.1 Infrastructure

FPGAs have traditionally been used within networking equipment such as routers and switches. In addition to the base functionality provided by the equipment vendor, more flexible switching and routing capabilities were later introduced with protocols such as OpenFlow. An OpenFlow switch [105] was implemented using NetFPGA hardware [156] to handle traffic going through the electrical engineering and computer science building at Stanford University. Subsequently, NetFPGA SUME was introduced for prototyping 10, 40, and 100 Gbps applications [170]. FPGAs have also been used as disaggregated computing resources by directly connecting to the datacenter network as standalone resources [3]. The infrastructure of this cluster can be separated into privileged and non-privileged regions using partial reconfiguration [122]. In recent years, computing requirements on the data plane have increased significantly with the emergence of applications such as server virtualization and overlay tunneling. Performing such functions in software causes significant CPU overhead, and SmartNICs were used to implement highly complex server side data plane functions. Microsoft introduced FPGA-based SmartNICs to provide accelerated cloud services for Azure customers by offloading host networking to hardware [47]. In this architecture, an FPGA is placed in series with the network connection of the blade server (bump-in-the-wire) that enables pre- and post-processing of data. Several other cloud service providers have also started using SmartNICs in their acceleration applications [89, 163]. A framework [42] has been proposed that provides a software abstraction and an FPGA-based hardware runtime for accelerating general purpose applications in multi-tenant systems.

6.2 Other Applications

Numerous data center applications can benefit from using FPGA acceleration in areas such as database acceleration, big data, machine learning, security, and privacy, some of which are described below.

6.2.1 Data Analytics. The core functionality of Memcached, a powerful open-source in-memory key-value store for small chunks of data, has been implemented on an FPGA [28]. This architecture has tightly integrated compute, network, and memory resources so it achieves significant benefits in terms of power consumption and performance compared to a baseline server. Xilinx has also built two versions of Memcached on FPGA [18, 19]. A methodology has been presented for integrating FPGAs into Apache Spark [31], an open source analytics engine used for big data processing. The cluster used in this work consists of a master node and six worker nodes, each of which is equipped with a PCIe-attached Alpha Data ADM-PCIE-7V3 FPGA board. It has been shown that the performance of Spark-FPGA integration scales well up to six worker nodes. A challenge of using Java-based analytics platforms like Spark is being able to efficiently share the application data managed by Java with the custom accelerator inside the FPGA [56, 57]. Samsung has proposed a near storage accelerator for database sort using a SmartSSD, which consists of an integrated FPGA that has a direct connection (P2P) to the storage device [126]. Having a direct connection obviates the need to transfer data through the host memory, which incurs significant latency in addition to occupying host resources. The implementation of a k-means algorithm using multiple FPGAs in a heterogeneous computing cluster has been presented [32]. Researchers have shown the ability to sort large amounts of data on an FPGA to support non-volatile memory [75], an application that could be useful for cloud architectures in the future. Similarly, researchers have shown that FPGAs can accelerate analytics for relational databases using a single node [93], an approach that could be implemented in the cloud. Network-attached FPGAs have been shown to significantly accelerate functions useful for database transactions, such as atomic broadcast in hardware, which has been demonstrated on a distributed key-value store implementation [73] that demonstrates low latency and high throughput. FLOEM is a system for accelerating network applications where reconfiguration is part of the NiC. FLOEM also uses key-value stores as a demonstrator of its efficacy [110].

6.2.2 Deep Learning. Computational requirements for processing AI workloads have grown significantly over the past few years, and cloud operators have opted to use FPGAs in applications involving DNNs because of their high power efficiency and performance compared to CPUs and GPUs. For some of the implementations, they were motivated by research outcomes in the field of machine learning. DNNWeaver [130] generates custom, synthesizable accelerators for deep neural networks that best match the needs of the DNN while providing high performance and efficiency gains for the target FPGA. Notably, DNNWeaver is FPGA-agnostic and has been demonstrated on both Intel and Xilinx FPGAs. The automatically generated accelerators deliver superior performance and efficiency compared to multicore CPU and GPU implementations. The redundancy associated with floating point number representation in CNNs used by CPUs and GPUs can be removed by quantizing weights and activations and retraining the neural network. FINN is a framework developed by Xilinx Research Labs to design flexible quantized neural network inference accelerators on FPGAs [20]. This work shows that simple arithmetic operations in **binary neural networks (BNNs)** are well suited for energy-efficient inference by fitting parameters entirely in **on-chip memory (OCM)**, which enables high computational performance. Microsoft's Brainwave [33, 49] is a hardware architecture developed to perform real time AI calculations using a soft-core **neural processing unit (NPU)** implemented on an Intel Stratix 10 FPGA. This NPU uses reduced precision proprietary floating point formats and enables competitive levels of performance and energy efficiency compared to hard NPUs without compromising the accuracy of the model. Brainwave was used as a cloud service [38] to accelerate ResNet-50 image classification for particle physics computing applications. An enhancement has been made to the Brainwave's NPU [24] to efficiently use the tensor blocks in an FPGA, and its performance has been compared against Nvidia's T4 and V100 GPUs. An FPGA acceleration platform using a supertile-based design method for general-purpose CNNs and image/video inference applications has also been introduced [164]. An interleaved task dispatching method has been used to scale up **supertile units (SUs)** to efficiently perform different types of convolutions. Training CNNs using an FPDeep framework has also been demonstrated [154]. Although the proposed architecture is not cloud-ready yet, the authors have provided some insights onto a possible future cloud deployment using a scalable FPGA cluster. The use of multiple FPGAs to implement more complex 3D CNNs is of research interest. Such implementations have demonstrated higher inference throughput while preserving the end-to-end latency of a single FPGA implementation [17, 134].

6.2.3 *Security and Privacy.*

An important application area for FPGAs in the cloud is the security and privacy of user data undergoing computation. Early research identified the necessity of having FPGAs secure client data from attackers and untrusted system administrators [40]. A virtualization framework was proposed and prototyped [29] that allows for the abstraction of cloud FPGA resources using **accelerator pools (APs)**. **Garbled circuits (GC)** [46] provide a promising approach for the hardware acceleration of privacy preserving computation. This work demonstrated the implementation of a generic reconfigurable coarse-grained FPGA overlay architecture for secure function evaluation. GC has been deployed using AWS [70] and applied to privacy-preserving machine learning on cloud servers using FPGAs [72]. A garbler circuit [70] can achieve a 15× speedup compared to a software implementation. The acceleration of homomorphic encryption is also being investigated [147]. Here, a domain-specific co-processor architecture is used to accelerate homomorphic function evaluation on encrypted data using AWS F1 FPGA instances. Researchers at Microsoft have demonstrated homomorphic encryption on FPGAs and demonstrated over 100× speedup compared to previous implementations [120].

6.3 Future Applications

FPGAs are increasingly being made available in user programmable cloud systems including those from Alibaba [171], AWS [7], and the Open Cloud Testbed [64]. The trend is to have these FPGAs be directly attached to the network to remove the latency incurred when data is transferred to and from the host. At the same time, FPGAs continue to grow in capacity and to incorporate larger and diverse memory elements. The applications of the future will take advantage of these properties to deliver graph processing [30], machine learning, security and privacy applications on large data, as well as accelerating a host of scientific applications and applications not yet imagined.

7 CONCLUSIONS

This article has discussed the recent and potential future evolution of FPGAs in cloud computing platforms and datacenters from the perspective of experts in this field of research. The article focuses on the architectural organization, resource management, security, and application deployment of FPGAs used in cloud and datacenter environments. These disciplines must be well understood for **cloud service providers (CSP)** to effectively provide resources to users, and for users to efficiently use the resources. The evolution of these FPGA systems has been chronicled in the article, and current research provides insight into future development trajectories.

Towards GPUs and CPUs, FPGAs have the advantage of low latency, high throughput, and energy efficiency. While FPGAs will not replace CPU and GPUs, they will be a complementary alternative in the cloud in applications such as computer vision, AI, security, crypto processing, and more.

We believe that future FPGA-inclusive cloud systems and datacenters will consolidate the vast selection of choices available today into a smaller range of architectures, programming models, and security-enhanced deployments. This consolidation may mirror the integration of CPUs and FPGAs in desktop and embedded systems that has taken place over the past two decades. With increased available FPGA resources per device, many applications in the future will use only a portion of an entire FPGA, leading to a need for resource sharing and system multi-tenancy. As a result, FPGA architectures for cloud computing will require physical resource and protocol isolation to ensure user security across different tenants, and the containment of faults and attacks that can disrupt system operation. Enhancements in architecture, programming models, and security will lead to a broader range of applications for FPGA-based cloud deployment and growing market share in the years to come.

The main obstacle for a broad adoption of FPGAs in datacenters is the lack of the software stack that allows easy deployment, management, and scaling of FPGAs on the cloud. Providing a software stack similar to the one that has led to the success of CPUs/GPUs will increase architecture diversification in the cloud, with FPGAs operating as equal partners on the side of CPUs and GPUs. The foundations for broad adoption of FPGA in datacenters are currently being developed. Innovative companies such as VMAccel, Inaccel, as well as the opensource initiatives such as the Cyborg project whose goal is the seamless integration of accelerator in OpenStack, are paving the way.

REFERENCES

[1] Mohamed S. Abdelfattah, David Han, Andrew Bitar, Roberto DiCecco, Shane O'Connell, Nitika Shanker, Joseph Chu, Ian Prins, Joshua Fender, Andrew C. Ling, et al. 2018. DLA: Compiler and FPGA overlay for neural network inference acceleration. In *28th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 411–4117.

[2] A. Abdelsalam, F. Clad, C. Filsfils, S. Salsano, G. Siracusano, and L. Veltri. 2017. Implementation of virtual network function chaining through segment routing in a Linux-based NFV infrastructure. In *IEEE Conference on Network Softwarization (NetSoft)*. 1–5. DOI: https://doi.org/10.1109/NETSOFT.2017.8004208

[3] Francois Abel, Jagath Weerasinghe, Christoph Hagleitner, Beat Weiss, and Stephan Paredes. 2017. An FPGA platform for hyperscalers. In *IEEE 25th Annual Symposium on High-Performance Interconnects*. 29–32. DOI: https://doi.org/10.1109/HOTI.2017.13

[4] Amran A. Al-Aghbari and Muhammad E. S. Elrabaa. 2019. Cloud-based FPGA custom computing machines for streaming applications. *IEEE Access* 7 (2019), 38009–38019.

[5] Md Mahbub Alam, Shahin Tajik, Fatemeh Ganji, Mark Tehranipoor, and Domenic Forte. 2019. RAM-Jam: Remote temperature and voltage fault attack on FPGAs using memory collisions. In *Workshop on Fault Diagnosis and Tolerance in Cryptography*. 48–55.

[6] Gustavo Alonso, Timothy Roscoe, David Cock, Muhsen Owaida, Kaan Kara, Dario Korolija, Zeke Wang, et al. 2020. Tackling hardware/software co-design from a database perspective. In *6th Biennial Conference on Innovative Data Systems Research (CIDR).*

[7] Amazon.com, Inc. 2021. Amazon EC2 F1 Instances. Retrieved from https://aws.amazon.com/ec2/instance-types/f1/.

[8] R. Amerson, R. J. Carter, W. B. Culbertson, P. Kuekes, and G. Snider. 1995. Teramac-configurable custom computing. In *IEEE Symposium on FPGAs for Custom Computing Machines.* 32–38.

[9] ARM. 2010. *AMBA 4 AXI4-Stream Protocol Specification.* Technical Report. ARM.

[10] ARM. 2011. *AMBA AXI and ACE Protocol Specification.* Technical Report. ARM.

[11] Mikhail Asiatici, Nithin George, Kizheppatt Vipin, Suhaib A. Fahmy, and Paolo Ienne. 2017. Virtualized execution runtime for FPGA accelerators in the cloud. *IEEE Access* 5 (2017), 1900–1910.

[12] Daniel O. Awduche. 1999. MPLS and traffic engineering in IP networks. *IEEE Commun. Mag.* 37, 12 (1999), 42–47.

[13] Hitesh Ballani, Paolo Costa, Thomas Karagiannis, and Ant Rowstron. 2011. Towards predictable datacenter networks. In *ACM SIGCOMM Conference (SIGCOMM'11).* Association for Computing Machinery, New York, NY, 242–253. DOI: https://doi.org/10.1145/2018436.2018465

[14] R. Baxter, S. Booth, M. Bull, G. Cawood, J. Perry, M. Parsons, A. Simpson, A. Trew, A. McCormick, G. Smart, R. Smart, A. Cantle, R. Chamberlain, and G. Genest. 2007. Maxwell – A 64 FPGA supercomputer. In *2nd NASA/ESA Conference on Adaptive Hardware and Systems (AHS'07).* 287–294. DOI: https://doi.org/10.1109/AHS.2007.71

[15] Christian Beckhoff, Dirk Koch, and Jim Torresen. 2010. Short-circuits on FPGAs caused by partial runtime reconfiguration. In *International Conference on Field Programmable Logic and Applications.* 596–601.

[16] D. Bernstein. 2014. Containers and cloud: From LXC to Docker to Kubernetes. *IEEE Cloud Comput.* 1, 3 (2014), 81–84.

[17] Saman Biookaghazadeh, Pravin Kumar Ravi, and Ming Zhao. 2021. Toward multi-FPGA acceleration of the neural networks. *ACM J. Emerg. Technol. Comput. Syst.* 17, 2 (2021), 1–23.

[18] Michaela Blott, Kimon Karras, Ling Liu, Kees Vissers, Jeremia Bär, and Zsolt István. 2013. Achieving 10Gbps line-rate key-value stores with FPGAs. In *5th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud'13).* USENIX Association, San Jose, CA. Retrieved from https://www.usenix.org/conference/hotcloud13/workshop-program/presentations/blott.

[19] Michaela Blott, Ling Liu, Kimon Karras, and Kees Vissers. 2015. Scaling out to a single-node 80gbps memcached server with 40terabytes of memory. In *7th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage'15).* USENIX Association, Santa Clara, CA. Retrieved from https://www.usenix.org/conference/hotstorage15/workshop-program/presentation/blott.

[20] Michaela Blott, Thomas B. Preußer, Nicholas J. Fraser, Giulio Gambardella, Kenneth O'Brien, Yaman Umuroglu, Miriam Leeser, and Kees Vissers. 2018. FINN-R: An end-to-end deep-learning framework for fast exploration of quantized neural networks. *ACM Trans. Reconfig. Technol. Syst.* 11, 3 (2018), 1–23.

[21] Christophe Bobda. 2007. *Introduction to Reconfigurable Computing: Architectures, Algorithms, and Applications* (1st ed.). Springer Publishing Company, Incorporated.

[22] C. Bobda, A. Majer, A. Ahmadinia, T. Haller, A. Linarth, and J. Teich. 2005. The Erlangen slot machine: Increasing flexibility in FPGA-based reconfigurable platforms. In *IEEE International Conference on Field-Programmable Technology.* 37–42. DOI: https://doi.org/10.1109/FPT.2005.1568522

[23] Marc Boulé and Zeljko Zilic. 2008. Automata-based assertion-checker synthesis of PSL properties. *ACM Trans. Des. Autom. Electron. Syst.* 13, 1 (Feb. 2008). DOI: https://doi.org/10.1145/1297666.1297670

[24] Andrew Boutros, Eriko Nurvitadhi, Rui Ma, Sergey Gribok, Zhipeng Zhao, James C. Hoe, Vaughn Betz, and Martin Langhammer. 2020. Beyond peak performance: Comparing the real performance of AI-Optimized FPGAs and GPUs. In *International Conference on Field-Programmable Technology (ICFPT).* 10–19. DOI: https://doi.org/10.1109/ICFPT51103.2020.00011

[25] Alexander Brant and Guy G. F. Lemieux. 2012. ZUMA: An open FPGA overlay architecture. In *IEEE 20th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM).* IEEE, 93–96.

[26] Stuart Byma, J. Gregory Steffan, Hadi Bannazadeh, Alberto Leon Garcia, and Paul Chow. 2014. FPGAs in the cloud: Booting virtualized hardware accelerators with openstack. In *IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines.* IEEE, 109–116.

[27] A. M. Caulfield, E. S. Chung, A. Putnam, H. Angepat, Jeremy Fowers, Michael Haselman, Stephen Heil, Matt Humphrey, Puneet Kaur, Joo-Young Kim, Daniel Lo, Todd Massengill, Kalin Ovtcharov, Michael Papamichael, Lisa Woods, Sitaram Lanka, Derek Chiou, and Doug Burger. 2016. A cloud-scale acceleration architecture. In *49th IEEE/ACM International Symposium on Microarchitecture.* 1–13.

[28] Sai Rahul Chalamalasetti, Kevin Lim, Mitch Wright, Alvin Au Young, Parthasarathy Ranganathan, and Martin Margala. 2013. An FPGA memcached appliance. In *ACM/SIGDA International Symposium on Field Programmable Gate Arrays.* 245–254.

[29] Fei Chen, Yi Shan, Yu Zhang, Yu Wang, Hubertus Franke, Xiaotao Chang, and Kun Wang. 2014. Enabling FPGAs in the cloud. In *11th ACM Conference on Computing Frontiers.* ACM, 3.

[30] Xinyu Chen, Hongshi Tan, Yao Chen, Bingsheng He, Weng-Fai Wong, and Deming Chen. 2021. ThunderGP: HLS-based graph processing framework on FPGAs. In *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 69–80.

[31] Yu-Ting Chen, Jason Cong, Zhenman Fang, Jie Lei, and Peng Wei. 2016. When Spark meets FPGAs: A case study for next-generation DNA sequencing acceleration. In *8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud'16)*.

[32] Yuk-Ming Choi and Hayden Kwok-Hay So. 2014. Map-Reduce processing of k-means algorithm with FPGA-accelerated computer cluster. In *IEEE 25th International Conference on Application-Specific Systems, Architectures and Processors*. 9–16.

[33] Chung Eric, Fowers Jeremy, Ovtcharov Kalin, Papamichael Michael, Caulfield Adrian, Massengill Todd, Liu Ming, Lo Daniel, Alkalay Shlomi, and Haselman Michael. 2018. Serving DNNs in real time at datacenter scale with project brainwave. *IEEE Micro* 38, 2 (2018), 8–20.

[34] Catalin Cimpano. 2020. Vast majority of cyber-attacks on cloud servers aim to mine cryptocurrency. Retrieved from https://www.zdnet.com/article/vast-majority-of-cyber-attacks-on-cloud-servers-aim-to-mine-cryptocurrency/.

[35] Chris Conger, Ian Troxel, D. Espinoza, Vikas Aggarwal, and A. George. 2005. NARC: Network attached reconfigurable computing for high performance, network based applications. In *8th Annual International Conference on Military and Aerospace Programmable Logic Devices (MAPLD'05)*.

[36] Guohao Dai, Yi Shan, Fei Chen, Yu Wang, Kun Wang, and Huazhong Yang. 2014. Online scheduling for FPGA computation in the cloud. In *International Conference on Field-Programmable Technology (FPT)*. IEEE, 330–333.

[37] Tiziano De Matteis, Johannes de Fine Licht, Jakub Beránek, and Torsten Hoefler. 2019. Streaming message interface: High-performance distributed memory programming on reconfigurable hardware. In *International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–33.

[38] Javier Duarte, Philip Harris, Scott Hauck, Burt Holzman, Shih-Chieh Hsu, Sergo Jindariani, Suffian Khan, Benjamin Kreis, Brian Lee, Mia Liu, et al. 2019. FPGA-accelerated machine learning inference as a service for particle physics computing. *Comput. Softw. Big Sci.* 3, 1 (2019), 1–15.

[39] Sankha Baran Dutta, Hoda Naghibijouybari, Nael Abu-Ghazaleh, Andres Marquez, and Kevin Barker. 2021. Leaky buddies: Cross-component covert channels on integrated CPU-GPU systems. In *ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 972–984.

[40] Ken Eguro and Ramarathnam Venkatesan. 2012. FPGAs for trusted cloud computing. In *22nd International Conference on Field Programmable Logic and Applications (FPL)*. 63–70.

[41] Maik Ender, Amir Moradi, and Christof Paar. 2020. The unpatchable silicon: A full break of the bitstream encryption of Xilinx 7-series FPGAs. In *29th USENIX Security Symposium*.

[42] Haggai Eran, Lior Zeno, Maroun Tork, Gabi Malka, and Mark Silberstein. 2019. NICA: An infrastructure for inline acceleration of network applications. In *USENIX Annual Technical Conference (USENIX ATC 19)*. 345–362.

[43] Dieter Ernst. 2020. *Competing in Artificial Intelligence Chips: China's Challenge Amid Technology War*. Centre for International Governance Innovation, Special Report.

[44] Nariman Eskandari, Naif Tarafdar, Daniel Ly-Ma, and Paul Chow. 2019. A modular heterogeneous stack for deploying FPGAs and CPUs in the data center. In *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'19)*. ACM, New York, NY, 262–271. DOI: https://doi.org/10.1145/3289602.3293909

[45] Suhaib A. Fahmy, Kizheppatt Vipin, and Shanker Shreejith. 2015. Virtualized FPGA accelerators for efficient cloud computing. In *IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, 430–435.

[46] Xin Fang, Stratis Ioannidis, and Miriam Leeser. 2017. Secure function evaluation using an FPGA overlay architecture. In *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 257–266.

[47] Daniel Firestone, Andrew Putnam, Sambhrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian Caulfield, Eric Chung et al. 2018. Azure accelerated networking: SmartNICs in the public cloud. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. 51–66.

[48] Forbes. 2018. Xilinx FPGAs: The Chip Behind Alibaba's Singles Day. Retrieved from https://www.forbes.com/sites/moorinsights/2018/11/29/xilinx-fpgas-the-chip-behind-alibabas-singles-day/?sh=5f2294e27e3b.

[49] Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Todd Massengill, Ming Liu, Daniel Lo, Shlomi Alkalay, Michael Haselman, Logan Adams, Mahdi Ghandi, Stephen Heil, Prerak Patel, Adam Sapek, Gabriel Weisz, Lisa Woods, Sitaram Lanka, Steven K. Reinhardt, Adrian M. Caulfield, Eric S. Chung, and Doug Burger. 2018. A configurable cloud-scale DNN processor for real-time AI. In *ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. 1–14. DOI: https://doi.org/10.1109/ISCA.2018.00012

[50] Karl Freund. 2017. Google Cloud TPU: Strategic Implications for Google, NVIDIA and the Machine Learning Industry. Retrieved from https://www.forbes.com/sites/moorinsights/2017/05/22/google-cloud-tpu-strategic-implications-for-google-nvidia-and-the-machine-learning-industry/?sh=69d2f5a13af7.

[51] Norihisa Fujita, Ryohei Kobayashi, Yoshiki Yamaguchi, and Taisuke Boku. 2019. Parallel processing on FPGA combining computation and communication in OpenCL programming. In *IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 479–488.

[52] Juan Fumero, Michail Papadimitriou, Foivos S. Zakkak, Maria Xekalaki, James Clarkson, and Christos Kotselidis. 2019. Dynamic application reconfiguration on heterogeneous hardware. In *15th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*. 165–178.

[53] William Gayde. 2020. How Arm Came to Dominate the Mobile Market and It's Coming for More, Much More. Retrieved from https://www.techspot.com/article/1989-arm-inside.

[54] Alan George, Herman Lam, and Greg Stitt. 2010. Novo-G: At the forefront of scalable reconfigurable supercomputing. *Comput. Sci. Eng.* 13, 1 (2010), 82–86.

[55] A. D. George, M. C. Herbordt, H. Lam, A. G. Lawande, J. Sheng, and C. Yang. 2016. Novo-G#: Large-scale reconfigurable computing with direct and programmable interconnects. In *IEEE High Performance Extreme Computing Conference (HPEC)*. 1–7. DOI: https://doi.org/10.1109/HPEC.2016.7761639

[56] Ehsan Ghasemi and Paul Chow. 2016. Accelerating Apache Spark big data analysis with FPGAs. In *International IEEE Conferences on Ubiquitous Intelligence Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld)*. 737–744. DOI: https://doi.org/10.1109/UIC-ATC-ScalCom-CBDCom-IoP-SmartWorld.2016.0119

[57] Ehsan Ghasemi and Paul Chow. 2019. Accelerating Apache Spark with FPGAs. *Concurr. Comput.: Pract. Exper.* 31, 2 (2019), e4222. DOI: https://doi.org/10.1002/cpe.4222 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.4222.

[58] Ilias Giechaskiel, Kasper Rasmussen, and Jakub Szefer. 2019. Reading between the dies: Cross-SLR covert channels on multi-tenant cloud FPGAs. In *IEEE International Conference on Computer Design*. 1–10.

[59] Ilias Giechaskiel, Kasper Rasmussen, and Jakub Szefer. 2020. CAPSULe: Cross-FPGA covert-channel attacks through power supply unit leakage. In *IEEE Symposium on Security and Privacy*. 1728–1741.

[60] Ilias Giechaskiel, Kasper B. Rasmussen, and Ken Eguro. 2018. Leaky wires: Information leakage and covert communication between FPGA long wires. In *Asia Conference on Computer and Communications Security (ASIACCS)*. 15–27.

[61] AWS GitHub. 2020. AFI Power. Retrieved from https://github.com/aws/aws-fpga/blob/master/hdk/docs/afi_power.md.

[62] Dennis R. E. Gnad, Fabian Oboril, and Mehdi B. Tahoori. 2017. Voltage drop-based fault attacks on FPGAs using valid bitstreams. In *International Conference on Field Programmable Logic and Applications (FPL)*. 1–7.

[63] Christoph Hagleitner, Dionysios Diamantopoulos, Burkhard Ringlein, Constantinos Evangelinos, Charles Johns, Rong N. Chang, Bruce D'Amora, James A. Kahle, James Sexton, Michael Johnston, Edward Pyzer-Knapp, and Chris Ward. 2021. Heterogeneous computing systems for complex scientific discovery workflows. In *Design, Automation Test in Europe Conference Exhibition (DATE)*. 13–18. DOI: https://doi.org/10.23919/DATE51398.2021.9474061

[64] S. Handagala, M. Herbordt, and M. Leeser. 2021. OCT: The open cloud FPGA testbed. In *31st International Conference on Field Programmable Logic and Applications (FPL)*.

[65] Reiner Hartenstein. 2001. Coarse grain reconfigurable architecture (embedded tutorial). In *Asia and South Pacific Design Automation Conference*. ACM, 564–570.

[66] Festus Hategekimana, Joel Mandebi Mbongue, Md Jubaer Hossain Pantho, and Christophe Bobda. 2018. Inheriting software security policies within hardware IP components. In *IEEE International Symposium on Field-Programmable Custom Computing Machines*. 53–56. DOI: https://doi.org/10.1109/FCCM.2018.00017

[67] Festus Hategekimana, Joel Mandebi Mbongue, Md Jubaer Hossain Pantho, and Christophe Bobda. 2018. Secure hardware kernels execution in CPU+ FPGA heterogeneous cloud. In *International Conference on Field-Programmable Technology (FPT)*. IEEE, 182–189.

[68] Nicole Hemsothd. 2017. First in-depth look at Google's new second-generation TPU. Retrieved from https://www.nextplatform.com/2017/05/17/first-depth-look-googles-new-second-generation-tpu/.

[69] Edson L. Horta, John W. Lockwood, David E. Taylor, and David Parlour. 2002. Dynamic hardware plugins in an FPGA with partial run-time reconfiguration. In *39th Annual Design Automation Conference (DAC'02)*. Association for Computing Machinery, New York, NY, 343–348. DOI: https://doi.org/10.1145/513918.514007

[70] Kai Huang, Mehmet Gungor, Xin Fang, Stratis Ioannidis, and Miriam Leeser. 2019. Garbled circuits in the cloud using FPGA enabled nodes. In *IEEE High Performance Extreme Computing Conference (HPEC)*. 1–6.

[71] Ted Huffmire, Brett Brotherton, Nick Callegari, Jonathan Valamehr, Jeff White, Ryan Kastner, and Tim Sherwood. 2008. Designing secure systems on reconfigurable hardware. *ACM Trans. Des. Autom. Electron. Syst.* 13, 3 (July 2008), 44:1–44:24.

[72] Siam U. Hussain, Bita Darvish Rouhani, Mohammad Ghasemzadeh, and Farinaz Koushanfar. 2018. Maxelerator: FPGA accelerator for privacy preserving multiply-accumulate (MAC) on cloud servers. In *55th Annual Design Automation Conference*. 1–6.

[73] Zsolt István, David Sidler, Gustavo Alonso, and Marko Vukolic. 2016. Consensus in a box: Inexpensive coordination in hardware. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI'16)*. 425–438.

[74] Chenglu Jin, Vasudev Gohil, Ramesh Karri, and Jeyavijayan Rajendran. 2020. Security of cloud FPGAs: A survey. *arxiv* arXiv:2005.04867 (2020).

[75] Sang-Woo Jun, Shuotao Xu, and Arvind. 2017. Terabyte sort on FPGA-accelerated flash storage. In *IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 17–24. DOI: https://doi.org/10.1109/FCCM.2017.53

[76] Abdul Kabbani, Balajee Vamanan, Jahangir Hasan, and Fabien Duchene. 2014. FlowBender: Flow-level adaptive routing for improved latency and throughput in datacenter networks. In *10th ACM International Conference on Emerging Networking Experiments and Technologies (CoNEXT'14)*. Association for Computing Machinery, New York, NY, 149–160. DOI: https://doi.org/10.1145/2674005.2674985

[77] Nachiket Kapre and Jan Gray. 2015. HopLite: Building austere overlay NOCs for FPGAs. In *25th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 1–8.

[78] Ahmed Khawaja, Joshua Landgraf, Rohith Prakash, Michael Wei, Eric Schkufza, and Christopher J. Rossbach. 2018. Sharing, protection, and compatibility for reconfigurable fabric with Amorphos. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI'18)*. 107–127.

[79] Yoongu Kim, Ross Daly, Jeremie S. Kim, Chris Fallin, Jihye Lee, Donghyuk Lee, Chris B. Wilkerson, Konrad K. Lai, and Onur Mutlu. 2014. Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. In *International Symposium on Computer Architecture*. 361–372.

[80] Ryohei Kobayashi, Yuma Oobata, Norihisa Fujita, Yoshiki Yamaguchi, and Taisuke Boku. 2018. OpenCL-ready high speed FPGA network for reconfigurable high performance computing. In *International Conference on High Performance Computing in Asia-Pacific Region*. 192–201.

[81] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. 2019. Spectre attacks: Exploiting speculative execution. In *40th IEEE Symposium on Security and Privacy (S&P'19)*.

[82] Dario Korolija, Timothy Roscoe, and Gustavo Alonso. 2020. Do OS abstractions make sense on FPGAs? In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI'20)*. 991–1010.

[83] S. Kumar Saha and C. Bobda. 2020. FPGA accelerated embedded system security through hardware isolation. In *Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*. 1–6. DOI: https://doi.org/10.1109/AsianHOST51057.2020.9358258

[84] Joshua Landgraf, Tiffany Yang, Will Lin, Christopher J. Rossbach, and Eric Schkufza. 2021. Compiler-driven FPGA virtualization with SYNERGY. In *26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 818–831.

[85] Xiangwei Li, Abhishek Jain, Douglas Maskell, and Suhaib A. Fahmy. 2016. An area-efficient FPGA overlay using DSP block based time-multiplexed functional units. *arXiv preprint arXiv:1606.06460* (2016).

[86] Thomas Lin, Byungchul Park, Hadi Bannazadeh, and Alberto Leon-Garcia. 2015. Savi testbed architecture and federation. In *Future Access Enablers of Ubiquitous and Intelligent Infrastructures*. Springer, 3–10.

[87] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. 2018. Meltdown: Reading kernel memory from user space. In *27th USENIX Security Symposium (USENIX Security'18)*.

[88] Peter Loscocco and Stephen Smalley. 2001. Meeting critical security objectives with security-enhanced Linux. In *Ottawa Linux Symposium*. 115–134.

[89] Layong Larry Luo and T. E. G. Tencent. 2018. In *Towards Converged SmartNIC Architecture for Bare Metal and Public Clouds at Tencent Scale 2nd Asia-Pacific Workshop on Networking (APNet 2018) August 2-3 2018, Beijing, China.*

[90] Jiacheng Ma, Gefei Zuo, Kevin Loughlin, Xiaohe Cheng, Yanqiang Liu, Abel Mulugeta Eneyew, Zhengwei Qi, and Baris Kasikci. 2020. A hypervisor for shared-memory FPGA platforms. In *25th International Conference on Architectural Support for Programming Languages and Operating Systems*. 827–844.

[91] M. Ma and V. W. S. Wong. 2019. An optimal peak hour content server cache update scheduling algorithm for 5G hetnets. In *IEEE International Conference on Communications (ICC)*. 1–6. DOI: https://doi.org/10.1109/ICC.2019.8761705

[92] Sen Ma, Zeyad Aklah, and David Andrews. 2015. A run time interpretation approach for creating custom accelerators. In *25th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 1–4.

[93] Divya Mahajan, Joon Kyung Kim, Jacob Sacks, Adel Ardalan, Arun Kumar, and Hadi Esmaeilzadeh. 2018. In-RDBMS hardware acceleration of advanced analytics. *Proc. VLDB Endow.* 11, 11 (July 2018), 1317–1331.

[94] Mallik Mahalingam, Dinesh G. Dutt, Kenneth Duda, Puneet Agarwal, Lawrence Kreeger, T. Sridhar, Mike Bursell, and Chris Wright. 2014. Virtual eXtensible local area network (VXLAN): A framework for overlaying virtualized layer 2 networks over layer 3 networks. *RFC* 7348 (2014), 1–22.

[95] Pongstorn Maidee, Alireza Kaviani, and Kevin Zeng. 2017. LinkBlaze: Efficient global data movement for FPGAs. In *International Conference on ReConFigurable Computing and FPGAs (ReConFig)*. IEEE, 1–8.

[96] Joel Mandebi Mbongue, Sujan Kumar Saha, and Christophe Bobda. 2021. Domain isolation in FPGA-accelerated cloud and data center applications. In *Great Lakes Symposium on VLSI*. 283–288.

[97] Joel Mandebi Mbongue, Danielle Tchuinkou Kwadjo, and Christophe Bobda. 2018. FLexiTASK: A flexible FPGA overlay for efficient multitasking. In *Great Lakes Symposium on VLSI*. ACM, 483–486.

[98] Joel Mbongue, Festus Hategekimana, Danielle Tchuinkou Kwadjo, David Andrews, and Christophe Bobda. 2018. FPGAVirt: A novel virtualization framework for FPGAs in the cloud. In *IEEE 11th International Conference on Cloud Computing (CLOUD)*. IEEE, 862–865.

[99] Joel Mandebi Mbongue, Danielle Tchuinkou Kwadjo, and Christophe Bobda. 2019. Automatic generation of application-specific FPGA overlays with Rapidwright. In *International Conference on Field-Programmable Technology (ICFPT)*. IEEE, 303–306.

[100] Joel Mandebi Mbongue, Danielle Tchuinkou Kwadjo, Alex Shuping, and Christophe Bobda. 2021. Deploying multi-tenant FPGAs within Linux-based cloud infrastructure. *ACM Trans. Reconfig. Technol. Syst.* 15, 2 (2021), 1–31.

[101] Joel Mandebi Mbongue, Sujan Kumar Saha, and Christophe Bobda. 2021. Performance study of multi-tenant cloud FPGAs. In *IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 168–171.

[102] Joel Mandebi Mbongue, Sujan Kumar Saha, and Christophe Bobda. 2021. A security architecture for domain isolation in multi-tenant cloud FPGAs. In *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 290–295.

[103] Joel Mandebi Mbongue, Alex Shuping, Pankaj Bhowmik, and Christophe Bobda. 2020. Architecture support for FPGA multi-tenancy in the cloud. In *IEEE 31st International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 125–132.

[104] Thierry Moreau, Tianqi Chen, Luis Vega, Jared Roesch, Eddie Yan, Lianmin Zheng, Josh Fromm, Ziheng Jiang, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. 2019. A hardware-software blueprint for flexible deep learning specialization. *IEEE Micro* 39, 5 (2019), 8–16. DOI: https://doi.org/10.1109/MM.2019.2928962

[105] Jad Naous, David Erickson, G. Adam Covington, Guido Appenzeller, and Nick McKeown. 2008. Implementing an OpenFlow switch on the NetFPGA platform. In *4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*. 1–9.

[106] Gil Neiger, Amy Santoni, Felix Leung, Dion Rodgers, and Rich Uhlig. 2006. Intel virtualization technology: Hardware support for efficient processor virtualization. *Intel Technol. J.* 10, 3 (2006).

[107] Jonas Ney, Dominik Loroch, Vladimir Rybalkin, Nico Weber, Jens Krüger, and Norbert Wehn. 2021. HALF: Holistic auto machine learning for FPGAs. In *st IEEE International Conference on Field-Programmable Logic and Applications (FPL)*. DOI: https://doi.org/10.1109/FPL53798.2021.00069

[108] M. Noormohammadpour and C. S. Raghavendra. 2018. Datacenter traffic control: Understanding techniques and tradeoffs. *IEEE Commun. Surv. Tutor.* 20, 2 (2018), 1492–1525. DOI: https://doi.org/10.1109/COMST.2017.2782753

[109] Opeyemi Osanaiye, Kim-Kwang Raymond Choo, and Mqhele Dlodlo. 2016. Distributed denial of service (DDoS) resilience in cloud: Review and conceptual cloud DDoS mitigation framework. *J. Netw. Comput. Applic.* 67 (May 2016), 147–165.

[110] Phitchaya Mangpo Phothilimthana, Ming Liu, Antoine Kaufmann, Simon Peter, Rastislav Bodik, and Thomas Anderson. 2018. FLOEM: A programming system for NIC-accelerated network applications. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI'18)*. 663–679.

[111] Christian Pilato, Stanislav Bohm, Fabien Brocheton, Jeronimo Castrillon, Riccardo Cevasco, Vojtech Cima, Radim Cmar, Dionysios Diamantopoulos, Fabrizio Ferrandi, Jan Martinovic, Gianluca Palermo, Michele Paolino, Antonio Parodi, Lorenzo Pittaluga, Daniel Raho, Francesco Regazzoni, Katerina Slaninova, and Christoph Hagleitner. 2021. EVEREST: A design environment for extreme-scale big data analytics on heterogeneous platforms. In *Design, Automation Test in Europe Conference Exhibition (DATE)*. 1320–1325. DOI: https://doi.org/10.23919/DATE51398.2021.9473940

[112] Christian Plessl. 2018. Bringing FPGAs to HPC production systems and codes. In *H2RC'18 Workshop at Supercomputing (SC'18)*. DOI: https://doi.org/10.13140/RG.2.2.34327.42407

[113] George Provelengios, Daniel Holcomb, and Russell Tessier. 2019. Characterizing power distribution attacks in multi-user FPGA environments. In *International Conference on Field Programmable Logic and Applications (FPL)*. 194–201.

[114] George Provelengios, Daniel Holcomb, and Russell Tessier. 2020. Power wasting circuits for cloud FPGA attacks. In *International Conference on Field Programmable Logic and Applications (FPL)*.

[115] 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. 2014. A reconfigurable fabric for accelerating large-scale data center services. In *ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*. 13–24. DOI: https://doi.org/10.1109/ISCA.2014.6853195

[116] Arzhang Rafii, Welson Sun, and Paul Chow. 2021. Pharos: A multi-FPGA performance monitor. In *31st International Conference on Field-Programmable Logic and Applications (FPL)*. 257–262. DOI: https://doi.org/10.1109/FPL53798.2021.00048

[117] Chethan Ramesh, Shivukumar B. Patil, Siva Nishok Dhanuskodi, George Provelengios, Sébastien Pillement, Daniel Holcomb, and Russell Tessier. 2018. FPGA side channel attacks without physical access. In *IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 45–52.

[118] Kasper Rasmussen, Ilias Giechaskiel, and Ken Eguro. 2019. Leakier wires: Exploiting FPGA long wires for covert-and side-channel attacks. *ACM Trans. Reconfig. Technol. Syst.* 12, 3 (2019), 11:1–11.29.

[119] Sandip Ray and Yier Jin. 2015. Security policy enforcement in modern SoC designs. In *IEEE/ACM International Conference on Computer-Aided Design*. 345–350.

[120] M. Sadegh Riazi, Kim Laine, Blake Pelton, and Wei Dai. 2020. Heax: An architecture for computing on encrypted data. In *25th International Conference on Architectural Support for Programming Languages and Operating Systems*. 1295–1309.

[121] B. Ringlein, F. Abel, D. Diamantopoulos, B. Weiss, C. Hagleitner, M. Reichenbach, and D. Fey. 2021. A case for function-as-a-service with Disaggregated FPGAs. In *IEEE 14th International Conference on Cloud Computing (CLOUD'21)*. 333–344. DOI: https://doi.org/10.1109/CLOUD53861.2021.00047

[122] Burkhard Ringlein, Francois Abel, Alexander Ditter, Beat Weiss, Christoph Hagleitner, and Dietmar Fey. 2019. System architecture for network-attached FPGAs in the cloud using partial reconfiguration. In *29th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 293–300. DOI: https://doi.org/10.1109/FPL.2019.00054

[123] B. Ringlein, F. Abel, A. Ditter, B. Weiss, C. Hagleitner, and D. Fey. 2020. Programming reconfigurable heterogeneous computing clusters using MPI with transpilation. In *IEEE/ACM International Workshop on Heterogeneous High-performance Reconfigurable Computing (H2RC)*. IEEE, 1–9. DOI: https://doi.org/10.1109/H2RC51942.2020.00006

[124] B. Ringlein, F. Abel, A. Ditter, B. Weiss, C. Hagleitner, and D. Fey. 2020. ZRLMPI: A unified programming model for reconfigurable heterogeneous computing clusters. In *IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 220. DOI: https://doi.org/10.1109/FCCM48280.2020.00051

[125] John M. Rushby. 1982. Proof of separability a verification technique for a class of security kernels. In *International Symposium on Programming*. 352–367.

[126] Sahand Salamat, Armin Haj Aboutalebi, Behnam Khaleghi, Joo Hwan Lee, Yang Seok Ki, and Tajana Rosing. 2021. NASCENT: Near-storage acceleration of database sort on SmartSSD. In *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'21)*. 262–272.

[127] Grigory Sapunov. 2021. Will ASIC Chips Become the Next Big Thing in AI? Retrieved from https://moorinsightsstrategy.com/will-asic-chips-become-the-next-big-thing-in-ai/.

[128] Falk Schellenberg, Dennis R. E. Gnad, Amir Moradi, and Mehdi B. Tahoori. 2018. An inside job: Remote power analysis attacks on FPGAs. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 1111–1116.

[129] Omar Sefraoui, Mohammed Aissaoui, and Mohsine Eleuldj. 2012. OpenStack: Toward an open-source solution for cloud computing. *Int. J. Comput. Applic.* 55, 3 (2012), 38–42.

[130] Hardik Sharma, Jongse Park, Divya Mahajan, Emmanuel Amaro, Joon Kyung Kim, Chenkai Shao, Asit Mishra, and Hadi Esmaeilzadeh. 2016. From high-level deep neural models to FPGAs. In *49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 1–12.

[131] S. B. Shaw, C. Kumar, and A. K. Singh. 2017. Use of time-series based forecasting technique for balancing load and reducing consumption of energy in a cloud data center. In *International Conference on Intelligent Computing and Control (I2C2)*. 1–6. DOI: https://doi.org/10.1109/I2C2.2017.8321782

[132] Jim Smith and Ravi Nair. 2005. *Virtual Machines: Versatile Platforms for Systems and Processes*. Elsevier.

[133] Hayden Kwok-Hay So and Cheng Liu. 2016. FPGA overlays. In *FPGAs for Software Programmers*. Springer, 285–305.

[134] Mengshu Sun, Pu Zhao, Mehmet Gungor, Massoud Pedram, Miriam Leeser, and Xue Lin. 2020. 3D CNN acceleration on FPGA using hardware-aware pruning. In *57th ACM/IEEE Design Automation Conference (DAC)*. 1–6.

[135] Jakub Szefer. 2019. Survey of microarchitectural side and covert channels, attacks, and defenses. *J. Hardw. Syst. Secur.* 3, 3 (Sept. 2019), 219–234.

[136] David Talbot. 2009. Vulnerability Seen in Amazon's Cloud-Computing. Retrieved from https://www.technologyreview.com/2009/10/23/208662/vulnerability-seen-in-amazons-cloud-computing/.

[137] Naif Tarafdar and Paul Chow. 2019. libGalapagos: A software environment for prototyping and creating heterogeneous FPGA and CPU applications. In *6th International Workshop on FPGAs for Software Programmers (FSP'19)*.

[138] Naif Tarafdar, Nariman Eskandari, Thomas Lin, and Paul Chow. 2017. Designing for FPGAs in the cloud. *IEEE Des. Test* 35, 1 (2017), 23–29.

[139] N. Tarafdar, N. Eskandari, V. Sharma, C. Lo, and P. Chow. 2018. Galapagos: A full stack approach to FPGA integration in the cloud. *IEEE Micro* 38, 06 (Nov. 2018), 18–24. DOI: https://doi.org/10.1109/MM.2018.2877290

[140] Naif Tarafdar, Thomas Lin, Eric Fukuda, Hadi Bannazadeh, Alberto Leon-Garcia, and Paul Chow. 2017. Enabling flexible network FPGA clusters in a heterogeneous cloud data center. In *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 237–246.

[141] Naif Tarafdar, Thomas Lin, Daniel Ly-Ma, Daniel Rozhko, Alberto Leon-Garcia, and Paul Chow. 2019. Building the infrastructure for deploying FPGAs in the cloud. In *Hardware Accelerators in Data Centers*. Springer, 9–33.

[142] Impulse Accelerated Technologies. 2021. Retrieved from https://web.archive.org/web/20110904033728/http://www.impulseaccelerated.com/.

[143] The Apache Software Foundation / TVM community. [n.d.]. Quick Start Tutorial for Compiling Deep Learning Models. Retrieved from https://tvm.apache.org/docs/tutorial/relay_quick_start.html.

[144] Shanquan Tian, Ilias Giechaskiel, Wenjie Xiong, and Jakub Szefer. 2021. Cloud FPGA cartography using PCIe contention. In *IEEE International Symposium on Field-Programmable Custom Computing Machines*.

[145] Shanquan Tian and Jakub Szefer. 2019. Temporal thermal covert channels in cloud FPGAs. In *International Symposium on Field-Programmable Gate Arrays (FPGA)*.

[146] Shanquan Tian, Wenjie Xiong, Ilias Giechaskiel, Kasper Rasmussen, and Jakub Szefer. 2020. Fingerprinting cloud FPGA infrastructures. In *International Symposium on Field-Programmable Gate Arrays (FPGA)*.

[147] Furkan Turan, Sujoy Sinha Roy, and Ingrid Verbauwhede. 2020. HEAWS: An accelerator for homomorphic encryption on the Amazon AWS FPGA. *IEEE Trans. Comput.* 69, 8 (2020), 1185–1196.

[148] Furkan Turan and Ingrid Verbauwhede. 2020. Trust in FPGA-accelerated cloud computing. *Comput. Surv.* 53, 6 (Dec. 2020), 28:1–28:128.

[149] Mohammad Usmani, Shahrzad Keshavarz, Eric Matthews, Lesley Shannon, Russell Tessier, and Daniel E. Holcomb. 2019. Efficient PUF-based key generation in FPGAs using per-device configuration. *IEEE Trans. VLSI Syst.* 27, 2 (Feb. 2019), 364–375.

[150] Juan Camilo Vega, Qianfeng Clark Shen, Alberto Leon-Garcia, and Paul Chow. 2019. Introducing ReCPRI: A field reconfigurable protocol for backhaul communication in a radio access network. In *IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. 329–336.

[151] Malte Vesper, Dirk Kocha, and Khoa Phama. 2017. PCIeHLS: An OpenCL HLS framework. In *4th International Workshop on FPGAs for Software Programmers*. VDE, 1–6.

[152] Hasitha Muthumala Waidyasooriya and Masanori Hariyama. 2019. Multi-FPGA accelerator architecture for stencil computation exploiting spacial and temporal scalability. *IEEE Access* 7 (2019), 53188–53201.

[153] Herbert Walder and Marco Platzner. 2004. A runtime environment for reconfigurable hardware operating systems. In *Field Programmable Logic and Application*, Jürgen Becker, Marco Platzner, and Serge Vernalde (Eds.). Springer Berlin, 831–835.

[154] Tianqi Wang, Tong Geng, Ang Li, Xi Jin, and Martin Herbordt. 2020. FPDeep: Scalable acceleration of CNN training on deeply-pipelined FPGA clusters. *IEEE Trans. Comput.* 69, 8 (2020), 1143–1158.

[155] X. Wang, Y. Niu, F. Liu, and Z. Xu. 2020. When FPGA meets cloud: A first look at performance. *IEEE Trans. Cloud Comput.* (2020), 1–1. DOI: https://doi.org/10.1109/TCC.2020.2992548

[156] Greg Watson, Nick McKeown, and Martin Casado. 2006. NetFPGA: A tool for network research and education. In *2nd Workshop on Architectural Research Using FPGA Platforms (WARFP)*, Vol. 3.

[157] J. Weerasinghe, F. Abel, C. Hagleitner, and A. Herkersdorf. 2015. Enabling FPGAs in hyperscale data centers. In *IEEE 12th International Conference on Ubiquitous Intelligence and Computing and IEEE 12th International Conference on Autonomic and Trusted Computing and IEEE 15th International Conference on Scalable Computing and Communications and Its Associated Workshops (UIC-ATC-ScalCom)*. 1078–1086. DOI: https://doi.org/10.1109/UIC-ATC-ScalCom-CBDCom-IoP.2015.199

[158] J. Weerasinghe, F. Abel, C. Hagleitner, and A. Herkersdorf. 2016. Disaggregated FPGAs: Network performance comparison against bare-metal servers, virtual machines and Linux containers. In *IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. 9–17. DOI: https://doi.org/10.1109/CloudCom.2016.0018

[159] J. Weerasinghe, R. Polig, F. Abel, and C. Hagleitner. 2016. Network-attached FPGAs for data center applications. In *International Conference on Field-Programmable Technology (FPT)*. 36–43. DOI: https://doi.org/10.1109/FPT.2016.7929186

[160] David Wilson and Greg Stitt. 2019. Seiba: An FPGA overlay-based approach to rapid application development. In *International Conference on ReConFigurable Computing and FPGAs (ReConFig)*. IEEE, 1–8.

[161] Xilinx Case Study. [n.d.]. Xilinx Powers Alibaba Cloud FaaS with AI Acceleration Solution for E-Commerce Business. Retrieved from https://www.xilinx.com/publications/powered-by-xilinx/xilinx-alibaba-case-study.pdf.

[162] Xilinx Corporation 2021. *Virtex UltraScale+ FPGA Data Sheet: DC and AC Switching Characteristics*.

[163] Wang Xu. 2018. Hardware acceleration over NFV in China Mobile. *OPNFV Plugfest.*

[164] Xiaoyu Yu, Yuwei Wang, Jie Miao, Ephrem Wu, Heng Zhang, Yu Meng, Bo Zhang, Biao Min, Dewei Chen, and Jianlin Gao. 2019. A data-center FPGA acceleration platform for convolutional neural networks. In *29th International Conference on Field Programmable Logic and Applications (FPL)*. 151–158.

[165] Shaza Zeitouni, Ghada Dessouky, and Ahmad-Reza Sadeghi. 2020. SoK: On the security challenges and risks of multi-tenant FPGAs in the cloud. *arxiv* arXiv:2009.13914 (2020).

[166] Shaza Zeitouni, Jo Vliegen, Tommaso Frassetto, Dirk Koch, Ahmad-Reza Sadeghi, and Nele Mentens. 2021. Trusted configuration in cloud FPGAs. In *IEEE International Symposium on Field-Programmable Custom Computing Machines*.

[167] Jiansong Zhang, Yongqiang Xiong, Ningyi Xu, Ran Shu, Bojie Li, Peng Cheng, Guo Chen, and Thomas Moscibroda. 2017. The Feniks FPGA operating system for cloud computing. In *8th Asia-Pacific Workshop on Systems*. 1–7.

[168] Ke Zhang, Yisong Chang, Mingyu Chen, Yungang Bao, and Zhiwei Xu. 2019. Computer organization and design course with FPGA cloud. In *50th ACM Technical Symposium on Computer Science Education*. ACM, 927–933.

[169] Mark Zhao and G. Edward Suh. 2018. FPGA-based remote power side-channel attacks. In *IEEE Symposium on Security and Privacy (S&P)*. 229–244.

[170] Noa Zilberman, Yury Audzevich, G. Adam Covington, and Andrew W. Moore. 2014. NetFPGA SUME: Toward 100 Gbps as research commodity. *IEEE Micro* 34, 5 (2014), 32–41.

[171] Xiantao Zxt, Zhengxiao Zx, and Justin Song. 2020. High-density multi-tenant bare-metal cloud with memory expansion SoC and power management. In *IEEE Hot Chips 32 Symposium (HCS)*. 1–18. DOI: https://doi.org/10.1109/HCS49909.2020.9220447

Received July 2021; revised October 2021; accepted December 2021