Coyote v2: Towards Open-Source, Reusable Infrastructure and Abstractions for FPGAs

Benjamin Ramhorst*
benjamin.ramhorst@inf.ethz.ch
Systems Group, ETH Zurich
Zurich, Switzerland

Maximilian J. Heer*
maximilian.heer@inf.ethz.ch
Systems Group, ETH Zurich
Zurich, Switzerland

Gustavo Alonso
alonso@inf.ethz.ch
Systems Group, ETH Zurich
Zurich, Switzerland

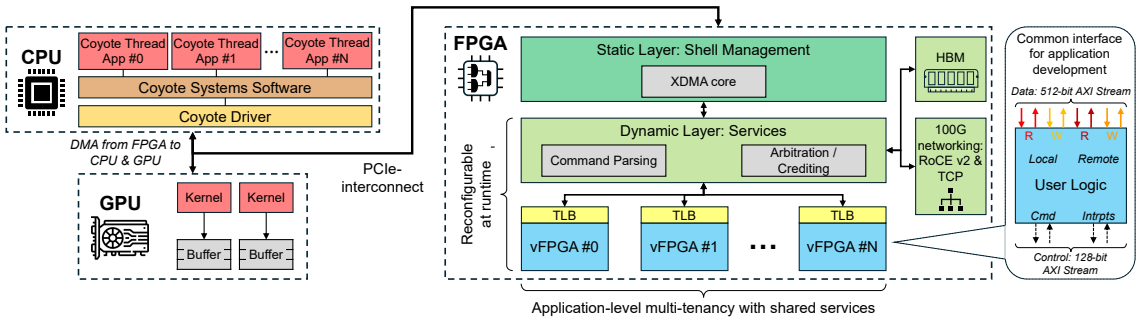

Figure 1. Architecture of the Coyote v2 framework in a heterogeneous setup with a CPU and GPU.

1 Introduction

As the demand for cloud and datacenter computing grows, traditional CPUs are reaching their performance limits. Consequently, cloud providers are increasingly investing in heterogeneous hardware systems, including GPUs, DPUs, and FPGAs [21]. FPGAs, in particular, are used as both application accelerators [5, 9, 11, 16] and SmartNICs for offloading network functions [8, 17]. In research, FPGAs have successfully been used to accelerate ML and database workloads [7, 14, 19], networking functions [18, 20] and cache-coherent systems [6]. However, a common issue in these projects is that they require extensive FPGA infrastructure; e.g., for data movement, high-speed networking or multi-tenancy. Existing tools and platforms do not provide the necessary functions and abstractions, often causing developers to spend significant time on infrastructure plumbing rather than application development and performance tuning. As an example, consider the base shell, AVED [1], for AMD's most recent and powerful FPGA, Alveo V80. While AVED facilitates data movement and card management, it lacks support for multi-tenancy, partial reconfiguration, and the card's powerful 800Gbps networking hardware. This forces many projects to start almost from scratch and reinvent the software stack needed to operate in a datacenter environment. As another example, recent projects have explored the use of FPGAs with other accelerators (e.g., GPUs [22] and smart storage [12]), further complicating the infrastructure requirements, since they are not supported by existing shells. Through our extensive experience with Coyote v1 [15], we have often encountered similar pitfalls, leading to application-specific patches that were not reusable. In this paper we identify a number of requirements for future FPGA shells and describe Coyote v2[1], our first step towards a unified FPGA platform for cloud and datacenter acceleration, providing support similar to a conventional OS on a CPU.

2 Coyote v2

We identify four key requirements an open-source FPGA shell should include. Alongside them, we also explain the approach taken in Coyote v2.

Transparent and generic user interface: The primary purpose of an FPGA shell should be able to facilitate application deployment and data movement. Therefore, each user should have a clear and easy-to-use interface on both the hardware- and software-level. On the hardware side, this interface can be thought of as an application binary interface (ABI), exposing a set of interfaces for data, control, and interrupt signals. Moreover, with the improvements in FPGA programming, the hardware interface should accommodate both hardware description languages (Verilog/VHDL) and High-Level Synthesis (HLS) kernels. On the software side, the interface should transparently enable users to move data between memories, devices, and the network, launch the

*Both authors contributed equally to this research.

LATTE '25, Rotterdam, Netherlands

[1]GitHub: https://github.com/fpgasystems/Coyote

aforementioned kernel, and wait for results. **Coyote v2 approach:** On the hardware, Coyote v2 is built around industry-standard AXI4 streams, which are used for local and remote reads/writes, shell control, and user interrupts. By using standard AXI4 streams, users are free to choose between HLS and RTL. On the software side, Coyote v2 includes a user-facing API written in C++ which facilitates data movement, networking, and reconfiguration. Under the hood, the software issues calls to a custom device driver which communicates with the FPGA through PCIe and an XDMA core (see Figure 1).

Services and libraries: Standard software libraries significantly simplify development. As an example, GPU vendors provide a rich set of libraries for networking, mathematics, and memory management. However, on FPGAs, these so-called "libaries" are sparse and rarely portable from project to project. A more complete FPGA shell should include support for TCP/IP and RDMA networks, DDR/HBM memory controllers, and offer the possibility to include additional hardware modules with minimum overhead. **Coyote v2 approach:** Thanks to its generic interfaces and modular design, we easily equipped Coyote v2 with our own, widely used and open-source, 100G network stack [4], supporting both TCP/IP and RDMA. To capture the full benefits of HBM, Coyote v2 automatically instantiates the necessary controllers and interfaces. Furthermore, the users are free to choose the number of memory channels, providing a trade-off between throughput and logic density. With the current shift to distributed and heterogeneous computing, recent contributions successfully extended Coyote with collective communications [10] and a DMA engine between FPGAs and GPUs [3].

Multi-tenancy and multi-threading: Recent FPGAs often exceed the requirements of a single application. For example, the recent AMD Alveo V80 is equipped with over 2.5 million logic elements (LUTs), 32 GB HBM supporting 800 GBps bandwidth and networking hardware supporting up to 100 GBps bandwidth, which is unlikely to be saturated by one application. In addition, given the shared nature of infrastructure in a cloud environment, it is imperative to support multi-tenancy. Furthermore, due to the deeply pipelined nature of FPGAs, a single hardware application can process multiple inputs simultaneously, which should be appropriately exposed to the user-space. **Coyote v2 approach:** As illustrated in Figure 1, the FPGA is spatially partitioned into multiple *virtual FPGAs* (vFPGAs). Each vFPGA represents a single user application with interfaces to the host and card memory, networking stacks, interrupts etc. Furthermore, Coyote adopts a virtual memory model, ensuring data separation between vFPGAs. To ensure fair access to the aforementioned services (networking, memory etc.), Coyote v2 also includes arbiters and schedulers, ensuring no vFPGA is using up all the bandwidth for itself. Within the user-facing software stack, a core component is the *Coyote thread* (cThread), which is associated with a specific vFPGA.

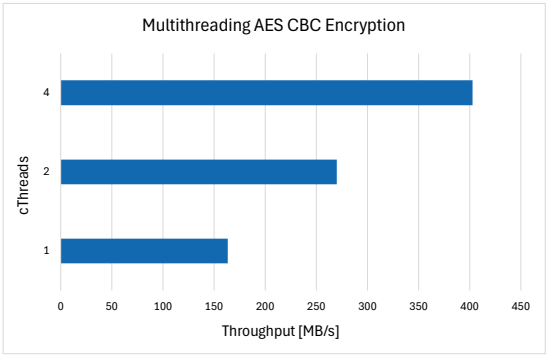

Figure 2. Throughput scaling with the number of cThreads for AES encryption using the same vFPGA.

However, due to the pipelined nature of FPGA applications, it is possible to assign multiple cThreads to the same vFPGA, thus improving throughput (see example in Figure 2).

Dynamic reconfiguration: A key concept in cloud computing is the ability to reuse the same hardware for different users and applications. As such, an FPGA shell should include the ability to reconfigure, at run-time, both the user applications (vFPGAs) and services (e.g., the network stack or the memory controller). **Coyote v2 approach:** Coyote v2 enables the reconfiguration of both the services (e.g. from RDMA to TCP/IP) and vFPGAs. The only fixed component in Coyote is the static layer, responsible for interactions with the host. By leveraging the Internal Configuration Access Port (ICAP) [2] and linking against the same static layer, Coyote v2 is able to load partial bitstreams into the FPGA configuration memory. On the host side, the reconfiguration is handled by the driver, which on one hand interacts with the ICAP through the XDMA, and on the other hand, exposes a set of *ioctl* calls to the user-facing software stack.

3 Future work

In future work, we plan to extend Coyote in multiple ways. First, we plan to investigate the integration and suitable abstractions with other hardware, such as SmartSSDs. Second, we plan to investigate suitable fallback mechanisms for packet processing on the host when the target protocol is not implemented on the FPGA to enable full SmartNIC capabilities. Finally, while previous work has shown the ability to run Coyote on a number of platforms (Alveo U55C, U280, U250, Enzian [6]), we also plan to investigate the portability to more recent FPGAs, such as the Alveo V80.

Acknowledgments

We would like to thank AMD for their continuous support in the development of Coyote and the donation of the Heterogeneous Accelerated Compute Cluster (HACC) which was used for Coyote v2 development and testing. We would like to especially thank Dario Korolija, whose dissertation [13] formed the basis for the development of Coyote v2.

References

[1] AMD. 2023. AMD Alveo Versal Example Design (AVED) Documentation — xilinx.github.io. https://xilinx.github.io/AVED/. [Accessed 27-01-2025]. (2023).

[2] AMD. 2012. AXI Hardware ICAP — xilinx.com. https://www.xilinx.com/products/intellectual-property/axi_hwicap.html. [Accessed 27-01-2025]. (2012).

[3] AMD. 2024. GitHub - Xilinx/ACCL at P2P — github.com. https://github.com/Xilinx/ACCL/tree/P2P. [Accessed 30-01-2025]. (2024).

[4] Systems Group at ETH Zurich. 2016. GitHub - fpgasystems/fpga-network-stack: Scalable Network Stack for FPGAs (TCP/IP, RoCEv2) — github.com. https://github.com/fpgasystems/fpga-network-stack. [Accessed 27-01-2025]. (2016).

[5] Eric S. Chung, Jeremy Fowers, Kalin Ovtcharov, et al. 2018. Serving dnns in real time at datacenter scale with project brainwave. *IEEE Micro*, 38, 2, 8–20. doi:10.1109/MM.2018.022071131.

[6] David A. Cock, Abishek Ramdas, Daniel Schwyn, et al. 2022. Enzian: an open, general, CPU/FPGA platform for systems software research. In *ASPLOS '22: 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Lausanne, Switzerland, 28 February 2022 - 4 March 2022*. Babak Falsafi, Michael Ferdman, Shan Lu, et al., (Eds.) ACM, 434–451. doi:10.1145/3503222.3507742.

[7] Jonas Dann, Daniel Ritter, and Holger Fröning. 2023. Non-relational databases on fpgas: survey, design decisions, challenges. *ACM Comput. Surv.*, 55, 11, 225:1–225:37. doi:10.1145/3568990.

[8] Daniel Firestone, Andrew Putnam, Sambrama Mundkur, et al. 2018. Azure accelerated networking: smartnics in the public cloud. In *15th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2018, Renton, WA, USA, April 9-11, 2018*. Sujata Banerjee and Srinivasan Seshan, (Eds.) USENIX Association, 51–66. https://www.usenix.org/conference/nsdi18/presentation/firestone.

[9] Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, et al. 2018. A configurable cloud-scale DNN processor for real-time AI. In *45th ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2018, Los Angeles, CA, USA, June 1-6, 2018*. Murali Annavaram, Timothy Mark Pinkston, and Babak Falsafi, (Eds.) IEEE Computer Society, 1–14. doi:10.1109/ISCA.2018.00012.

[10] Zhenhao He, Dario Korolija, Yu Zhu, et al. 2024. ACCL+: an fpga-based collective engine for distributed applications. In *18th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2024, Santa Clara, CA, USA, July 10-12, 2024*. Ada Gavrilovska and Douglas B. Terry, (Eds.) USENIX Association, 211–231. https://www.usenix.org/conference/osdi24/presentation/he.

[11] Amazon Web Services Inc. 2024. Amazon EC2 F1 Instances. [Accessed 16-03-2024]. (2024). https://aws.amazon.com/ec2/instance-types/f1/?nc1=h_ls.

[12] Hongsun Jang, Jaeyong Song, Jaewon Jung, et al. 2024. Smart-infinity: fast large language model training using near-storage processing on a real system. In *IEEE International Symposium on High-Performance Computer Architecture, HPCA 2024, Edinburgh, United Kingdom, March 2-6, 2024*. IEEE, 345–360. doi:10.1109/HPCA57654.2024.00034.

[13] Dario Korolija. 2024. Abstractions for modern heterogeneous systems. (2024). doi:10.3929/ETHZ-B-000671565.

[14] Dario Korolija, Dimitrios Koutsoukos, Kimberly Keeton, et al. 2022. Farview: disaggregated memory with operator off-loading for database engines. In *12th Conference on Innovative Data Systems Research, CIDR 2022, Chaminade, CA, USA, January 9-12, 2022*. www.cidrdb.org. https://www.cidrdb.org/cidr2022/papers/p11-korolija.pdf.

[15] 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 2020, Virtual Event, November 4-6, 2020*. USENIX Association, 991–1010. https://www.usenix.org/conference/osdi20/presentation/roscoe.

[16] Miriam Leeser, Suranga Handagala, Michael Zink, et al. 2021. Fpgas in the cloud. *Comput. Sci. Eng.*, 23, 6, 72–76. doi:10.1109/MCSE.2021.3127288.

[17] Matt McInnes, Melissa Hollingshed, Cynthia Nottingham, et al. 2024. Overview of Azure Boost — learn.microsoft.com. https://learn.microsoft.com/en-us/azure/azure-boost/overview. [Accessed 29-01-2025]. (2024).

[18] Mario Ruiz, David Sidler, Gustavo Sutter, et al. 2019. Limago: an fpga-based open-source 100 gbe TCP/IP stack. In *29th International Conference on Field Programmable Logic and Applications, FPL 2019, Barcelona, Spain, September 8-12, 2019*. Ioannis Sourdis, Christos-Savvas Bouganis, Carlos Álvarez, et al., (Eds.) IEEE, 286–292. doi:10.1109/FPL.2019.00053.

[19] Ahmad Shawahna, Sadiq M. Sait, and Aiman El-Maleh. 2019. Fpga-based accelerators of deep learning networks for learning and classification: A review. *IEEE Access*, 7, 7823–7859. doi:10.1109/ACCESS.2018.2890150.

[20] David Sidler, Zeke Wang, Monica Chiosa, et al. 2020. Strom: smart remote memory. In *EuroSys '20: Fifteenth EuroSys Conference 2020, Heraklion, Greece, April 27-30, 2020*. Angelos Bilas, Kostas Magoutis, Evangelos P. Markatos, et al., (Eds.) ACM, 29:1–29:16. doi:10.1145/3342195.3387519.

[21] Neil C. Thompson and Svenja Spanuth. 2021. The decline of computers as a general purpose technology. *Commun. ACM*, 64, 3, 64–72. doi:10.1145/3430936.

[22] Zeke Wang, Hongjing Huang, Jie Zhang, et al. 2022. Fpganic: an fpga-based versatile 100gb smartnic for gpus. In *Proceedings of the 2022 USENIX Annual Technical Conference, USENIX ATC 2022, Carlsbad, CA, USA, July 11-13, 2022*. Jiri Schindler and Noa Zilberman, (Eds.) USENIX Association, 967–986. https://www.usenix.org/conference/atc22/presentation/wang-zeke.