Enzian: An Open, General, CPU/FPGA Platform for Systems Software Research

David Cock
Systems Group, D-INFK, ETH Zurich
Zurich, Switzerland

Abishek Ramdas
Systems Group, D-INFK, ETH Zurich
Zurich, Switzerland

Daniel Schwyn
Systems Group, D-INFK, ETH Zurich
Zurich, Switzerland

Michael Giardino
Systems Group, D-INFK, ETH Zurich
Zurich, Switzerland

Adam Turowski
Systems Group, D-INFK, ETH Zurich
Zurich, Switzerland

Zhenhao He
Systems Group, D-INFK, ETH Zurich
Zurich, Switzerland

Nora Hossle
Systems Group, D-INFK, ETH Zurich
Zurich, Switzerland

Dario Korolija
Systems Group, D-INFK, ETH Zurich
Zurich, Switzerland

Melissa Licciardello
Systems Group, D-INFK, ETH Zurich
Zurich, Switzerland

Kristina Martsenko
Systems Group, D-INFK, ETH Zurich
Zurich, Switzerland

Reto Achermann
University of British Columbia
Vancouver, Canada

Gustavo Alonso
Systems Group, D-INFK, ETH Zurich
Zurich, Switzerland

Timothy Roscoe
Systems Group, D-INFK, ETH Zurich
Zurich, Switzerland

ABSTRACT

Hybrid computing platforms, comprising CPU cores and FPGA logic, are increasingly used for accelerating data-intensive workloads in cloud deployments, and are a growing topic of interest in systems research. However, from a research perspective, existing hardware platforms are limited: they are often optimized for concrete, narrow use-cases and, therefore lack the flexibility needed to explore other applications and configurations.

We show that a research group can design and build a more general, open, and affordable hardware platform for hybrid systems research. The platform, Enzian, is capable of duplicating the functionality of existing CPU/FPGA systems with comparable performance but in an open, flexible system. It couples a large FPGA with a server-class CPU in an asymmetric cache-coherent NUMA system. Enzian also enables research not possible with existing hybrid platforms, through explicit access to coherence messages, extensive thermal and power instrumentation, and an open, programmable baseboard management processor.

Enzian is already being used in multiple projects, is open source (both hardware and software), and available for remote use. We present the design principles of Enzian, the challenges in building it, and evaluate it with a range of existing research use-cases alongside other, more specialized platforms, as well as demonstrating research not possible on existing platforms.

ASPLOS '22, February 28 – March 4, 2022, Lausanne, Switzerland

ACM ISBN 978-1-4503-9205-1/22/02…$15.00
https://doi.org/10.1145/3503222.3507742

CCS CONCEPTS

• **Hardware → Reconfigurable logic and FPGAs**; • **Software and its engineering → Operating systems**; • **Computer systems organization → Heterogeneous (hybrid) systems**.

KEYWORDS

FPGAs, heterogeneous systems, cache coherence.

ACM Reference Format:
David Cock, Abishek Ramdas, Daniel Schwyn, Michael Giardino, Adam Turowski, Zhenhao He, Nora Hossle, Dario Korolija, Melissa Licciardello, Kristina Martsenko, Reto Achermann, Gustavo Alonso, and Timothy Roscoe. 2022. Enzian: An Open, General, CPU/FPGA Platform for Systems Software Research. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '22), February 28 – March 4, 2022, Lausanne, Switzerland.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3503222.3507742

1 INTRODUCTION

In this paper, we tackle how to facilitate relevant and far-reaching systems software research in an unstable and rapidly churning hardware landscape.

Until recently, research in operating systems, data management, networked systems, storage, and cloud computing has relied on commodity PC servers and associated peripherals as the underlying hardware platform. This has allowed considerable variation in functionality and performance while the commonality and general-purpose nature of this platform has made it easy to agree on relevant challenges, build on prior work, and apply ideas quickly in real-world scenarios.

This is no longer the case. Specialized architectures are increasingly deployed in place of less efficient conventional systems [70]. While the trend started with the proliferation of GPUs, today there

is a plethora of specialized processors and hardware accelerators used in both mobile devices and at all levels the cloud stack: smart storage, smart caches, smart NICs, programmable switches, TPUs, Field Programmable Gate Arrays (FPGAs), and so on.

In this work, we focus on "hybrid" systems combining CPUs and FPGAs, as they both cover a wide range of existing and future applications, and also provide greater potential for systems software research. Such systems are becoming widely deployed today, particularly in the cloud.

However, even for these existing systems, there is no consensus on the best long-term design choices for these new hardware platforms. Current designs are often cost-optimized for narrow use cases. We survey this space in Section 2.1, and show that even basic questions like whether cache-coherence between CPU and FPGA is a good idea have no clear answers.

Many such issues would benefit from software systems research insights. However, to date, academic research in systems has been tightly constrained by the available hardware. Creatively exploring the large hardware design space from a systems perspective is practically impossible.

Each CPU/FPGA platform today comes with its own quirks and limitations. One must first choose an existing platform (not designed for exploratory long-term research), climb the learning curve for programming it, fight its limitations for applications outside the intended use-case, and try to derive useful results about future software designs based on this. The situation is made worse by the closed, proprietary, and/or poorly-documented nature of many of these platforms not to mention the added complexities associated with FPGA development.

Not only does this make it difficult to explore the design space, but it is also impossible to compare results across different platforms, or transfer ideas from one platform to another. In Section 2.2 we survey the considerable recent work on "operating systems" for FPGAs, which illustrates the constraints researchers face when testing and deploying their designs.

To address this, we took inspiration from Mogul *et al.* [45] and designed and built an open, hybrid computing platform *informed by system software experience* but also *optimized for longer-term research*. The result, Enzian (Figure 1), provides both *coverage* (it can be used to explore a superset of the hardware design space of existing systems) and *openness* (practically all the system is available for access and modification).

Enzian is a 2-socket coherent Non-Uniform Memory Access (NUMA) system where one node is a large FPGA, and can be configured to imitate a wide range of hybrid computing platforms on the market, with comparable performance. This makes it possible to evaluate new ideas at scale, with real workloads, and compare the advantages and disadvantages of different designs and hardware/software options. However, Enzian goes beyond current systems in how it allows memory, network, storage, cache coherence, and other resources to be configured, and thus not only fully covers but greatly extends the "convex hull" of existing systems.

Enzian is an EATX format [74] (305×330mm) server board (Figure 1). Since it is built as a research computer, it also incorporates functionality often lacking in commodity, cost-optimized parts. It is heavily instrumented for power and thermal management. The

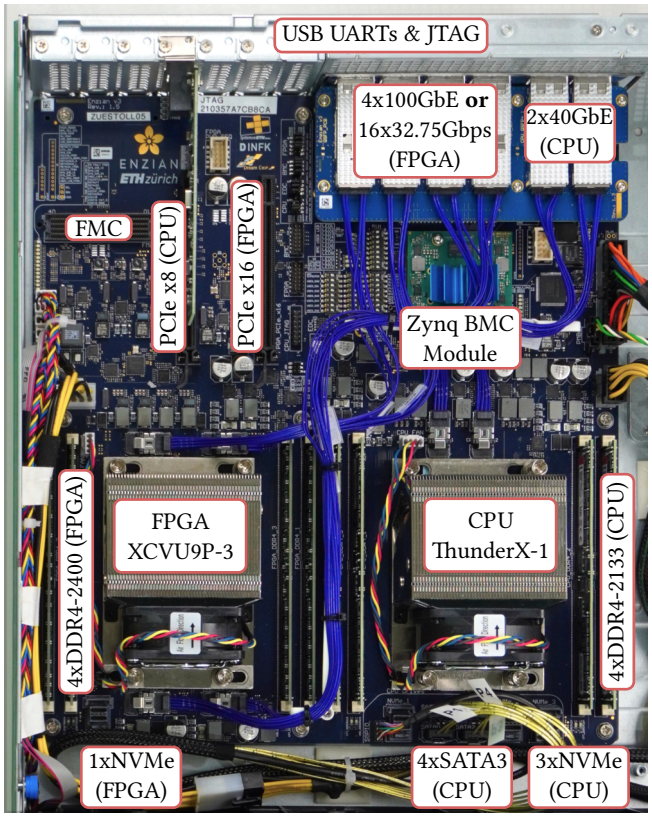

Figure 1: A complete Enzian board

open nature of the platform means that the Baseboard Management Controller (BMC), generally hidden on modern servers and accelerator cards, is fully available for programming. Direct, low-level access to cache coherence messages in the FPGA open up a range of research applications in system tracing, cache management, and high-performance I/O. Remarkably, Enzian achieves this at reasonable unit cost, comparable to a medium-to-large 2-socket rack-mount server.

Enzian makes several key research contributions. First, it shows that a single research platform can not only cover but extend the convex hull of all the existing platforms for hybrid systems research, and do so without excessive unit cost, which demonstrates that the specializations we see in other platforms are, in fact, narrow cost-based compromises and not technically grounded.

Second, we show research results unobtainable with existing platforms, such as a server-class application-specific memory controller. More broadly, we show the benefits (in both raw performance and in functionality) of expanding the design space for hybrid systems by eschewing standards like PCI Express (PCIe) in favor of lower-level interconnects more closely coupled with the CPU.

Thirdly, the process of building Enzian and the openness of the resulting platform (in particular the board management firmware) expose, and facilitate addressing, fundamental problems previously unaccessible to the research community that arise in the construction of modern servers.

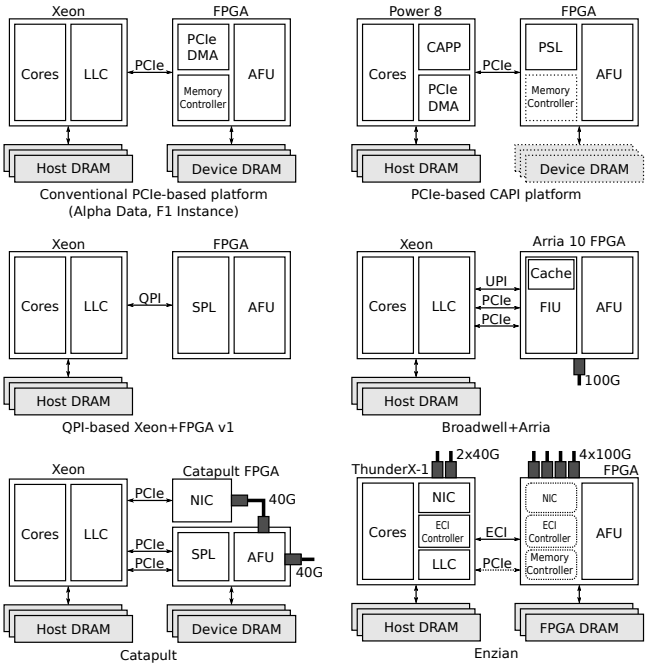

Figure 2: CPU+FPGA topologies (adapted from Choi *et al.* [13])

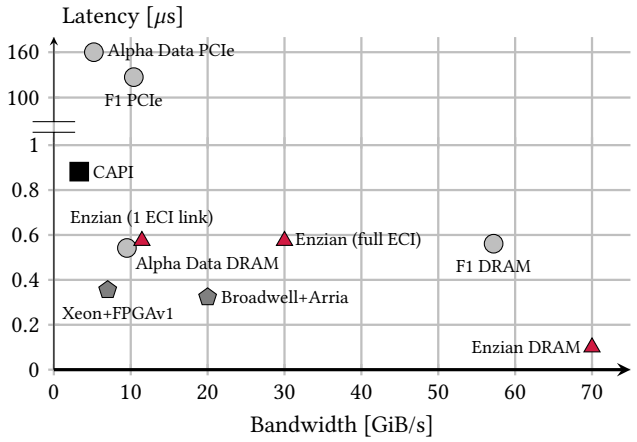

Figure 3: CPU-FPGA performance summary (adapted from Choi *et al.* [14]), with Enzian added for reference

Following our surveys in Section 2, in Section 3 we discuss the goals of Enzian and the design principles (and constraints) that guided its development. Section 4 then describes the platform implementation in detail. Section 5 shows that Enzian can subsume many existing designs for hybrid systems, while also adding new functionality of value to researchers. We demonstrate Enzian acting in the role of a range of different other systems, and show that its performance is highly competitive. We then show research use cases not possible with existing hardware platforms. Finally, we supplement this large-scale evaluation with both micro-benchmarks and demonstrations of Enzian's instrumentation.

In Section 7 we conclude and lay out our future plans for making Enzian widely available as a research platform.

2 BACKGROUND AND RELATED WORK

In this section we motivate and provide the necessary background for Enzian as well as covering related work.

2.1 The Changing FPGA Platform Landscape

We begin by describing the wide range of hybrid CPU/FPGA platforms available, and by examining their features and limitations from a research perspective, support our argument for a common, open research platform.

Choi *et al.* [14] surveyed and illustrated the diversity of hybrid CPU/FPGA platforms, and in particular the relationship between the CPU and FPGA. We use their classification to position Enzian in today's landscape. Figure 2 compares the configuration of Enzian with that of other systems, and Figure 3 compares its DRAM and interconnect performance to that of these systems (the performance data for Enzian are discussed in Section 5.1). Existing platforms fall into several categories, though some systems span multiple areas.

The first consists of **PCIe-based accelerators** in conventional servers, including modern offerings like the Intel D5005 [30], Alpha Data [3], Xilinx Alveo [76, 77], and the often-proprietary hardware deployed in cloud-based solutions from Amazon [4] or Alibaba [1].

Card-based acceleration (e.g., Amazon F1 [4]) is often modeled after GPUs with a programming interface based on batch-oriented programming models like OpenCL, where data is copied *en-masse* onto the card's memory for computation, and the results copied back to host memory using PCIe Direct Memory Access (DMA).

PCIe has sufficient bandwidth for such bulk transfers, but the limitations of this model are known to be a performance problem for fine-grained workloads. Efforts like the Open Coherent Accelerator Processor Interface (OpenCAPI) [66] and its predecessor the Coherent Accelerator Processor Interface (CAPI) [65], have sought to add more complex memory functionality into the accelerator, still viewing the FPGA as a PCIe-attached peripheral but adding a hardware protocol layer for cache coherence, fault handling, and virtual address translation. More recent protocol efforts to include similar capabilities over current or future versions of PCIe include Cache Coherent Interconnect for Accelerators (CCIX) [11] and Intel Compute eXpress Link (CXL) [19].

The second category eschews PCIe (or augments it) in favor of a **full cache coherency protocol**, as in Intel HARP systems [29, 51, 57], and more recently Intel's Broadwell+Arria architecture, which combines cache-coherent Universal Path Interconnect (UPI) and also PCIe links between CPU and FPGA. This results in very different software designs. Even PCIe-based OpenCL kernels ported to Intel HARPv2 have derived significant benefits from shared virtual memory versus explicit reads and writes to the FPGA [7]. Compression accelerators developed specifically for this architecture obtain dramatically higher performance [57]. Dagger [39] implements Remote Procedure Call (RPC) on the FPGA to use it as a smart NIC, taking advantage of the network connection available in the Arria 10 FPGA. Kona [8] uses the coherent interconnect to implement

remote memory by exposing "fake" physical memory backed by memory on other Kona machines.

Cache coherent FPGAs to date, however, also impose limitations on research. The design is skewed towards fine-grained acceleration, and the FPGA often has no local memory except a small, fixed cache. The coherence protocol remains closed, with the FPGA "shell" enforcing specific usage models for the interconnect via a wrapper. There are sound engineering decisions for this from the hardware vendors' perspective, but research use-cases remain tightly constrained.

A third category centers on a network interface, using the FPGA as the basis of a **smart NIC**. Dedicated FPGA-based smart NICs like Mellanox Innova [44] connect an FPGA to the NIC through an internal PCIe bus. Because there is no direct connection between FPGA and CPU, these smart NICs can only be used to process network packets. This permits exploring hardware acceleration for network protocols, and the NetFPGA [48] project has created a series of open-source FPGA-based NICs to facilitate this. NetFPGA strongly influenced our work in its goals and motivation, but our goal is much broader than networking research. In contrast to these approaches, more modern FPGA accelerators increasingly provide some network interface, allowing greater flexibility and probably serving better as smart NICs [39, 64].

Of particular relevance are Microsoft's series of Catapult FPGA systems for datacenters, where the FPGA is connected to the CPU through both a PCIe link and an Ethernet "bump in the wire" connection [10, 24, 56]. Such hardware has been used to accelerate applications ranging from key-value stores [40] to machine learning [15], and illustrates some of the advantages of FPGAs over (rather faster) ASICs in time-to-market and evolvability — for example, through the use of custom numeric formats for machine learning. Catapult, however, remains a closed system, with relatively few on-board resources like memory (due to niche in cloud networking).

A final category is the **single-chip hybrid platform or Multiprocessor System-on-a-Chip (MPSoC)**. These (e.g. [18]) integrate CPU cores onto an FPGA die. Access to the cache protocol on the chip in these systems allows a use-case not possible with Intel HARP: the FPGA acting as part of the CPU's memory system, enabling research into new remote memory protocols [8, 9] and cache management [58]. However, such systems are tiny in comparison with server machines, typically using 4 low-power ARM cores with limited DRAM. This makes it hard to examine either future on-die solutions or even contemporary datacenter workloads running on mid-range servers, which is unfortunate since these are the scenarios typically targeted by such architectures.

Ultimately, all these platforms work well for their intended use-cases but are limited in their own ways, constraining the research possible with them. Enzian aims to cover this whole space and more: native coherence between CPU and FPGA, ample memory and network bandwidth on the FPGA, and open, low-level access to all aspects of the system.

2.2 System "Software" for FPGAs

The rise of FPGA deployments has led researchers to examine what resource management means in the context of hybrid systems, which so far lack the basic abstractions provided by a traditional OS: memory management, scheduling, services like the network, etc. [38]. This is partly due to the rather dated concept of FPGAs as discrete acceleration units for manually offloading processing, independent of the primary CPU and its memory. The persistence of this concept in the available hardware platforms therefore constrains any attempts to implement and evaluate the equivalent of an OS for a hybrid system. In some cases, the resulting designs seem to be mostly a consequence of the underlying hardware.

AmorphOS [34] tackles the challenge of multi-tenancy in FPGAs by transforming the problem of sharing the FPGA fabric into a compilation/synthesis problem. Applications are either pre-compiled with multiple versions that can be assigned to different FPGA regions, or merged into a single circuit deployed as a unit. AmorphOS views the FPGA is an isolated device with no network connection or access to host memory, and the design follows from this.

Optimus [42] also tackles multi-tenancy on FPGA accelerators. In contrast to AmorphOS, it provides sophisticated, controlled access to host memory, including virtual addressing. However, Optimus focuses on PCIe-attached boards and therefore does not address cache coherency, access to FPGA memory from the CPU, or networking since these are features rarely found together in existing hardware.

Even Coyote [38], which probably implements the most microkernel-like set of abstractions and offers the greatest flexibility in memory addressing and protection, still restricts access to what one would expect in a PCIe card, rather than a full NUMA system in which the FPGA is a first-class participant in the memory system.

These and other recent systems make their choices for good reasons, and are solving complex and difficult problems. Our goal is not to criticize them, but to point out that they are primarily a product of the concrete configuration and features available in the hardware platforms they on which they run [26].

A fully-fledged analysis of OS design for hybrid systems needs a full-featured platform where one can argue generality and expandability, and also where different designs (perhaps emphasizing different trade-offs) can be compared directly.

Today, a valid comparison of, say, Dagger with Optimus is infeasible due to the radically different choices adopted by Intel's HARP-derived product and Xilinx' PCIe cards.

3 ENZIAN APPROACH AND DESIGN PRINCIPLES

We experienced these challenges first-hand in our previous research, and in response we decided to design and build our own *research-oriented hardware platform* and make it available to the research community. We took inspiration from past projects to create common research platforms and infrastructure, for example Berkeley Unix, Emulab [73], PlanetLab [54], and in particular the NetFPGA project [48]. We were also encouraged by Mogul *et al.* [45] to be critical of existing hardware designs, and try to design hardware from a software perspective.

The result is Enzian, a server-class machine designed for systems *research* into hybrid computing, typically overengineered for any single application, but which emphasizes as much flexibility as possible in how it can be used in research.

A flexible platform optimized for research has many advantages. It can include much more *instrumentation* than commercial products (e.g. for detailed power measurements). It reduces the *learning curve* for students, since one platform can be used across a range of projects. Moreover, it can act as *reference point* for comparing ideas. By being open and overengineered, it encourages *research applications not yet envisioned* when it was designed. If Enzian can be *shared across the research community*, it will allow much greater composition of projects across institutions and sharing of new tools, techniques, and applications.

Whereas a commercial product is aimed at a ready market and set of existing application use-cases, a research machine can be viewed from several different perspectives.

Firstly, like existing products it is a platform for running applications, and can be used to design, prototype, and evaluate new and better ways to exploit FPGAs as part of a complete system. Much of our evaluation of Enzian in Section 5 is from this perspective, showing that it can be used in place of existing products for the same applications.

Secondly, however, it also is a means of prototyping ideas that may eventually find their way into dedicated hardware like ASICs. Enzian is thus also a way to help system software researchers explore the broad hardware design space. In Section 5.4 we demonstrate an example using the FPGA to emulate a custom memory controller.

Thirdly, we can also view one part of Enzian as a way to instrument and control the rest of the platform. In Section 5.5 we show this with detailed power measurements, but there are other ways of partitioning Enzian into a "system under test" and "test harness" that we do not explore in this paper, such as detailed cache tracing using the coherence protocol, or runtime verification by feeding processor trace records in real time to the FPGA.

A research-optimized platform like Enzian can only function if it meets two important criteria:

- It must provide maximal *coverage*: it must be capable of replacing existing, more specialized platforms, and cover as much of the potential hardware design space as possible.
- *Performance* must be adequate. It is unlikely to outperform more specialized platforms, but performance should be good enough to be in the same ballpark.

In the rest of this paper we show that Enzian meets these requirements: it provides a superset of the functionality of existing platforms, with performance that is close enough to make strong claims for research projects implemented using it.

In addition, Enzian had other requirements. First and foremost, we had to design and build it on an academic research budget. It also had to be as usable and easily programmable as possible.

We faced a long series of challenges in designing and building Enzian. However, three are particularly relevant to its intended use as a flexible computer for research. The first was simply deciding on the high-level specification of the machine. We were guided by a set of *design principles* and also a set of pragmatic *compromises* to make the project feasible.

Our design principles were as follows; note that most are the exact opposite of good industrial product design practice:

- **Don't worry about unit cost:** While anxious to keep the project budget under control, we made no specific effort to minimize the *unit cost* of the eventual Enzian boards. This is the opposite practice to product development, where upfront design investment can yield more cost-effective end products. As a research machine, Enzian will only exist in relatively small numbers in labs, and so it is better to increase flexibility and ease of design at the cost of price-per-unit.
- **If in doubt, overengineer:** A research platform should limit researchers as little as possible. We exposed as much functionality of our components to the programmer as we could, and used the highest-specification components available. For example, we used the largest, and fastest, Xilinx FPGA avilable at the time, and used almost all the FPGA's high-speed transceivers for NVM Express (NVMe), networking interfaces, etc. Overengineering, or "maxing everything out", optimizes the flexibility of Enzian at the expense of unit cost; it can always be used to emulate a less capable board.
- **Favor bandwidth over capacity:** Many tradeoffs were of this nature. For example, we could have supported more DRAM capacity at the cost of bandwidth. We chose to favor bandwidth (one DDR4 DIMM per channel) since it is easier to scale experiment workloads by data volume, and higher performance facilitates easier experimentation.
- **Avoid the limitations of standards:** A research platform does not need to support an extensive industry ecosystem, but should allow experimentation beyond the restrictions of existing standards where practical. For example, we did not build Enzian around the PCIe connection used by most heterogeneous machines, instead using the CPU's native cache coherence protocol. PCIe is fast, and so this choice makes little difference for a class of use-cases derived from GPU-based workloads. However, it pays dividends when viewing the FPGA as a NUMA node, a custom memory controller, or a way to instrument the CPU's cache, for example.
- **Instrument as much as possible:** We were able to instrument power and clock regulators in a way not possible with commercial boards, and we took full advantage of this, much of it facilitated by us having to write our own BMC firmware.
- **Don't just think in single units:** A platform like Enzian is as much a building block for larger systems as it is a single discrete machine, and the flexible design should reflect this. For example, one reason that Enzian has such large network bandwidth (480 Gb/s) is to enable, e.g., many boards to be connected together into a single, large multiprocessor (with or without cache coherence), or to allow prototyping novel network switches or middleboxes.

At the same time, we faced practical limitations in what we could build, through limited resources, skill-sets, and time. The major scoping decision we took was selecting the CPU.

We decided early to focus on platform-level architecture research rather than trying to support work on processor design (for example, instruction set extensions or closely-coupled coprocessors), at full clock speed. This would have required designing and building a new processor (something beyond our group's skill and time), and is an area already well-served by the RISC-V initiative.

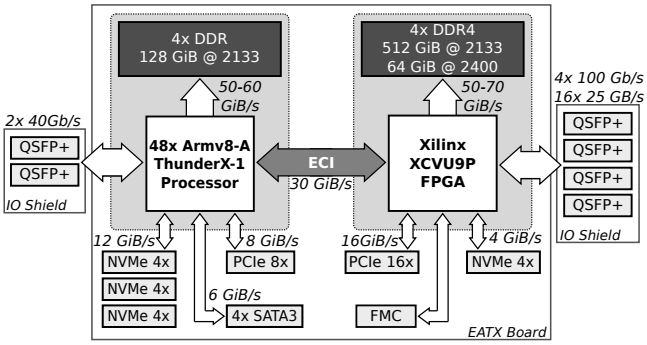

Figure 4: Enzian block diagram

We therefore needed an existing server-class CPU part supporting multisocket NUMA configurations, with a cache coherence protocol we could interoperate with on the FPGA. When we started, only one available SoC met these requirements, and that remains mostly true today. The Enzian CPU part trades off single-thread performance (it is mostly in-order) for parallelism. Even so, with 48 64-bit ARM cores at 2.0 GHz, performance is respectable.

We now describe in detail the Enzian that resulted from these design principles.

4 DESIGN

Enzian (see Figure 4) is a two-socket asymmetric NUMA system in which one node is a 48-core Marvell Cavium ThunderX-1 ARMv8 CPU [49, 50] clocked at 2.0 GHz. This System-on-Chip (SoC) has several accelerators (crypto, NIC, etc.) on-die, and the "networking" variant we use also has a programmable match-action table switch. The CPU has 128 GiB of DRAM on four DDR4 channels. The second node is a Xilinx XCVU9P Ultrascale+ FPGA [78]. We use the highest-speed variant of the FPGA, and in practice it runs at clock speeds between 200 and 300 MHz, depending on the loaded bitstream.

The FPGA has four DIMM slots collectively supporting up to 1 TiB DDR4 DRAM (current systems use 512 GiB or 64 GiB), and is connected to the CPU over the latter's native coherent interconnect using 24 10 Gb/s lanes with total *theoretical* bandwidth of 30 GiB/s in each direction.

Following the philosophy of "maxing everything out", both nodes have ample network bandwidth: 2×40 Gb/s Ethernet interfaces on the CPU SoC and 16×25 Gb/s serial lines on the FPGA, configurable as e.g. 4×100 Gb/s or 16×25 Gb/s Ethernet links. The motivation for such bandwidth on the FPGA is not simply to allow use as a smart network interface adaptor (NIC) (as in Section 5.2) or to cover the use cases for Microsoft Catapult [10, 24, 25, 56]. It also allows experimentation with custom fabric protocols and permits multiple Enzian boards to be connected into a single multiprocessor (possibly extending cache coherence across machines), a topic we intend to investigate in future work.

The FPGA also has a PCIe x16 slot and a single NVMe connector, to complement $3 \times$ NVMe, $4 \times$ Serial ATA (SATA), and a single PCIe x8 slot on the CPU. With the interconnect and DRAM, this accounts for all the high-speed pins available on the FPGA (even though we chose the XCVU9P to maximize the available transceivers). Additional components such as High-Bandwidth Memory (HBM) can be connected to the FPGA using an adapter to the FPGA Mezzanine Card (FMC) or PCIe slot.

4.1 Coherent Interconnect

A key feature of Enzian in comparison with commercial hybrid CPU/FPGA server platforms is that CPU and FPGA are connected via the CPU's *native* inter-socket cache coherence protocol, rather than PCIe. This was a major design decision, and indeed a key motivation to build Enzian in the first place. While modern PCIe implementations provide excellent theoretical bandwidth for GPU-style applications, they impose performance constraints on other usage models that would have limited the scope of Enzian. That said, it is possible to connect the two nodes via PCIe using a crossover cable, allowing reproduction of PCIe-based experiments or out-of-band communication from the inter-socket interconnect.

The need to use a "native" inter-socket protocol determined the choice of CPU, which had to support server-class multi-socket configurations via a protocol which we could practically implement ourselves on the FPGA. At the time, the only candidate was the Cavium (now Marvell) ThunderX-1 series. Interestingly, none of our use-cases for the coherence protocol so far involve implementing a significant cache on the FPGA, which in any case would have limited performance.

Our implementation, the Enzian Coherence Interface (ECI), is a MOESI-based protocol with 128-byte cache lines that in principle allows a line to be cached on the home or requesting node. It also supports non-cached small I/O reads and writes, and inter-processor interrupts. The system's physical address space is statically partitioned between the CPU and FPGA.

Getting ECI to interoperate with the ThunderX-1's CCPI protocol was a significant challenge despite much help and support from the vendor in terms of documentation and conversations with the designers. Recent standardization efforts [11, 19] aside, coherence protocols are not designed for interoperability, but instead for two identical silicon parts to talk to each other, and are typically documented accordingly.

For ECI, it helped that the lower layers of the protocol closely resembled existing standards, and were implemented on the CPU with attention to robustness. The CPU-side implementation had extensive diagnostic hardware, could be controlled from the Board Development Kit (BDK) command line before the processor fully booted, and dialed up and down in lanes and speed, allowing us to bring up our implementation gradually.

Nevertheless, significant effort was required. We extensively used built-in logic analyzers on the FPGA side to debug the implementation. We also took protocol traces of a 2-socket CPU system booting for reference, and wrote a Wireshark plugin to decode the coherence protocol's upper layers.

We then defined our own serialization format for the messages on ECI's various virtual circuits. This not only allowed us to store and analyze traces in a nice format, but also served as an interoperability standard for various software tools [43]. For example, we built a simulation environment which glued together a model we wrote of the CPU's L2 cache (running as part of ARM's "FAST models"

simulation suite) and a Verilog simulator for the FPGA hardware running on a different machine over a network [80].

We also formally specified several layers of the protocol, and generated formatters and assertion checkers from the specifications. This work has yielded many benefits, both for the Enzian (which can cover a much wider range of research use-cases, as we show in Section 5.4 and Section 5.5), and for us as a research group: the work we did on specification of the protocol layers has become a research line in its own right.

4.2 Baseboard Management Controller

Another challenge in building Enzian was managing power, clock, and temperature on the board itself. A significant engineering task for us was writing all the board firmware for Enzian.

Nearly all modern servers include hidden processors known as BMCs whose purpose is to control power and clock distribution for all components on the board. Far from being a simple Complex Programmable Logic Device (CPLD) or microcontroller, these processors often run a full operating system such as Linux or Minix [72]. The research community has paid very little attention to rigorously engineering hardware and software for BMCs in spite of the fact that the BMC has nearly complete control over the server, is connected to the network, and is unaccountable to the host operating system, CPU firmware, or hypervisor.

The Enzian BMC is an off-the-shelf Xilinx Zynq 7000 [18]-based System-on-Module (SoM), currently running OpenBMC [23] over embedded Linux. It provides more than a powerful, modern interface for controlling the server board. The addition of an FPGA on the BMC not only allowed for reconfiguration and rerouting of signals during the development process, but is an avenue for exploring reliable, real-time, reconfigurable baseboard controllers. The BMC is powered on whenever the case PSU is plugged in and has its own gigabit Ethernet interface and USB Type-A port.

The firmware running on the BMC needs to control all voltage regulators and clock generators correctly and turn them on and off in the right order. Although the regulators on Enzian came with fairly good datasheets, we had to learn much about how the power and clock distribution hardware works by trial and error, with nothing more than an oscilloscope for debugging. Experiments had to be conducted very carefully as mistakes in a regulator's configuration could trigger a short circuit on a high current (over 150 Amps) line and fatally damage one of the few board prototypes existing at the time. This was particularly challenging during the COVID-19 pandemic, with almost all of the team working remotely and coordinating via chat and video conferencing.

This challenge in implementing the firmware started another research direction in verifiable BMC design. Given the precise thresholds and sequencing requirements of the system components, finding a correct sequence and configuration for the 25 regulators requires non-trivial engineering. To bring assurance to this process, we developed a technique of declarative power sequencing in which powering requirements are specified, and then a solver is used to generate a provably correct sequence [60].

However, this is only one portion of the stack. Further research has gone into a verified, modular Inter-Integrated Circuit (I^2C) stack [27], and an ongoing effort looks to port sel4 [35] to the BMC, giving a verified, secure base from which to build our trustworthy and high-assurance control software.

Implementing the firmware ourselves allows us to make extensive instrumentation of Enzian available to users. Unlike in a regular server, the BMC is also wide open for use in experimentation (subject to the above caveats). In particular, it can export information from all the various voltage, current, and thermal sensors in the system over a network without impacting the main CPU.

4.3 Instrumentation

Modern computer platforms require a large number of discrete voltage rails, often with multiple voltages required for a single component. For example, a CPU may require three separate voltage domains [5] while each channel of DDR4 DRAM requires two [46]. This means that for a basic two socket server system, there are a minimum of 14 required voltage regulators. This essential control network introduces significant complexity in design, test, and operation, but can provide a unique opportunity for researchers: fine-grained control and power measurements of the numerous subsystems.

Enzian has 25 discrete voltage regulators supplying 30 voltage rails, each of which can be controlled and queried for some combination of voltage, current, and temperature. The majority of regulators are controlled via Power Management Bus (PMBus) [69], a superset of System Management Bus (SMBus) [68], which is in turn built on I^2C [61], a widely-used low-speed serial communication standard. Each regulator can be independently controlled or queried in approximately 5 ms.

As well as being essential for us when trying to bring of Enzian, the ability to independently monitor and control voltage regulators at fine granularity makes Enzian a worthy experimental platform for examining the undervolt behavior of FPGAs [59], CPUs [71], and DRAM [12]. Detailed power measurements coupled with application and CPU performance metrics can be used to develop models for power-aware resource scheduling and allocation. In Section 5.5 we show the BMC's capabilities for monitoring the power regulators on the board under a multi-phase diagnostic and stress-test workload.

4.4 Boot Sequence and System Software

The Enzian power-on sequence mostly follows that of a commercial server, with a few variations. The BMC powers up and boots, and then turns on power and clock to the rest of the system including FPGA and the CPU, which is held in reset. It then loads the FPGA with an initial bitstream, for example, the static component of a "shell" like Coyote [38]. It then takes the CPU out of reset.

The CPU loads the BDK which, in turn, loads the ARM Trusted Firmware (ATF) [55] and UEFI environment. The BDK is interesting in that it allows extensive configuration of the CPU and associated hardware. For example, the BDK is responsible for bringing up the ECI protocol, and can be used to limit bandwidth, number of lanes, or clock frequency to many parts of the system (indeed, early debugging of ECI was done with 4 lanes rather than the full 24). This degree of control is also useful for "scaling" the performance of some parts of the system, in order to simulate a platform with different performance characteristics.

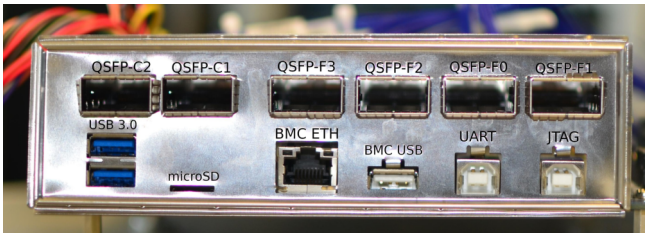

Figure 5: The Enzian rear panel

From UEFI, the CPU can boot Linux. No modifications were necessary to the Linux kernel, but Enzian requires a special Device-Tree [20, 41] specification since, of the two NUMA nodes, only one actually has CPU cores and the other may or may not appear to have memory. We currently use stock Ubuntu 20.04 LTS, which conveniently has drivers for most of the CPU devices, including the 40 Gb/s NICs.

4.5 FPGA "Shell"

Programmable FPGA-based systems typically load a "shell" or "donut" at startup, and then partially reconfigure the rest of the FPGA with the desired application(s). Enzian follows this pattern: the BMC loads an initial image into the FPGA, but Enzian's open nature allows this image to be user-specified.

To enable ECI, the initial image must exist on the FPGA before the CPU starts to boot, since CPU firmware attempts to detect the other NUMA node, train the links, etc. at startup. All the shells we use for Enzian therefore include the lower levels of ECI functionality.

Beyond that, however, Enzian offers a wide range of possibilities. Our default environment is a port of the open-source Coyote [38] shell. This allows the rest of the FPGA to be dynamically reconfigured by the CPU over ECI. Moreover, it provides a kernel of basic functionality (memory protection, address translation, spatial and temporal multiplexing, and a standard execution environment) plus additional services (virtualized DRAM controllers, network stacks, etc.) to applications each running in a Virtual FPGA (vFPGA).

Porting Coyote to Enzian was relatively straightforward, mainly replacing the PCIe DMA-based interface between the FPGA and CPU with one using ECI and dealing in cache lines rather than PCIe transactions. Other minor changes are due to the different Enzian pinout, and the presence of more Ethernet interfaces and DDR4 memory controllers than Coyote's original Alveo platform.

However, for the range of research use-cases we target, Coyote is most certainly not a panacea. For example, Enzian can be used to build custom memory controllers providing coherent "logical views" of DRAM to the CPU's cache, allowing applications with sparse but predictable access patterns to pack data structures densely into cache lines (as we show in Section 5.4), or for cache coherence to be extended across a network fabric. We also see Enzian as an attractive environment for exploring the design of alternative OSes for FPGAs in hybrid systems.

4.6 The Complete Board

A working Enzian system is shown in Figure 1. It is a standard EATX-format (305 × 330 mm) board which fits in a 2U rackmount case with a PMBus-enabled 1200 kW CRPS redundant power supply. The 6 QSFP-28 cages are presented on a standard ATX rear panel. Half-height PCIe cards fit in a standard 2U case, and full-height cards can be used via a PCIe riser. While we specified the board in considerable detail (including most of the components), we lacked the expertise in PCB design and signal integrity to design and build a large, 18-layer board with 15 GHz signals, and so this part together with board assembly was outsourced to a company, Dream Chip Technologies GmbH.

Building our own system for research allowed us to expose functionality that would often be hard to access in a regular machine. For example, Enzian has a number of serial consoles or UARTs: two from the CPU SoC, one from the FPGA, and one from the BMC processor. Since our BMC is overengineered, we used the Zynq's FPGA to route all four to a serial-to-USB converter connected to a USB type-B socket on the board (see Figure 5). This means an Enzian board can be plugged into a PC with a regular USB cable and an OS developer can access all four serial consoles without additional hardware, a feature that (with the BMC Ethernet) proved indispensable for doing software bringup during COVID lockdown.

Similarly, each of the primary components (CPU, FPGA, and BMC) have a JTAG port, a standard interface for debugging, testing, and reflashing. These are multiplexed and passed to a Digilent JTAG-to-USB converter for output via another USB type-B socket. Because all daisy-chained JTAG devices must be powered for the chain to work, we also provide bypass and external pinouts for obtaining individual JTAG interfaces.

For thermal management, each socket has a large fanned heat-sink with 4 additional ports for case fans. To facilitate low-level debugging, the power rails have indicator LEDs and test-points with silk-screened identification on the board. There are also two banks of user-definable DIP switches, and banks of LEDs controllable from either CPU or BMC.

5 EVALUATION

Our evaluation of Enzian is intended to determine whether it achieves its goal: to be a realistic general-purpose platform for heterogeneous systems research. To do so, Enzian must have similar performance to commercial systems specialized to particular use-cases, and must support use-cases similar to those proposed in the systems research literature to date. We start with evaluating key aspects of Enzian's performance (cache coherent interconnect and networking) and then show it can be used to support diverse use-cases (machine learning accelerators, memory controllers, and advanced instrumentation).

5.1 Cache Coherent Interconnect

We first compare throughput and latency of Enzian's ECI with that of the PCIe links used in commercial FPGA accelerators.

A feature of ECI inherited from the CPU implementation is that the 24 lanes (each with a theoretical bandwith of 10 Gb/s) are organized in two *links* of 12 lanes each. Transactions can in principle use either of these links, and the load-balancing strategy used by the CPU when it initiates transactions can be configured at boot time. For clarity, in this experiment we restrict all traffic on Enzian to *only one* of the two ECI links.

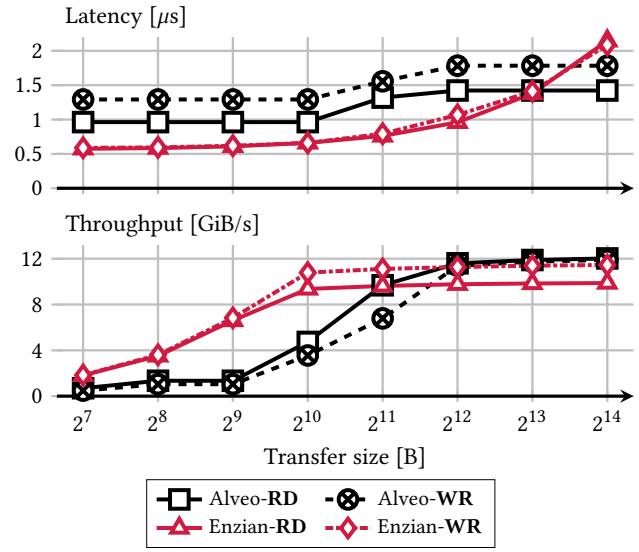

Figure 6: Link performance: ECI (one link) vs. PCIe x16 Gen3

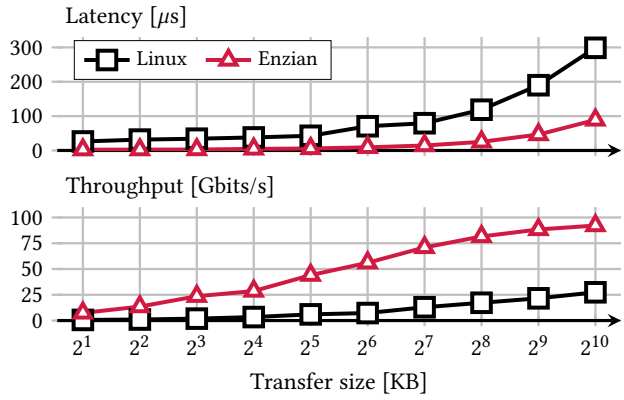

Figure 7: FPGA TCP stack performance, Enzian (1 flow) vs. CPU/Linux kernel stack (1 flow)

In a real application achieved bandwidth could be *up to double* this (theoretically, 30 GiB/s), but in practice will be rather less due to load-balancing effects. We expect the actual policy for using the two links will be defined by the developer of the FPGA application.

We benchmark the FPGA reading and writing (using uncached, coherent, cacheline-sized transactions) over ECI to host (CPU) memory. We compare Enzian with a Xilinx Alveo u250 data center card [76] in an Intel Xeon server using 16-lane PCIe Gen3, giving a maximum theoretical bandwidth of 16 GiB/s per direction. We measure achieved data throughput and latency for various transfer sizes, each averaged over 10000 runs.

Figure 6 shows a single ECI comfortably matches the Alveo's PCIe performance for large transfers, and this one link has significantly higher throughput for transfers under 2 KiB than Alveo's PCIe. Perfect balancing across both ECI links would double these figures for Enzian, but would be hard to achieve in practice.

Read performance for ECI is slightly lower than for writes; we conjecture that the limiting factor here is the performance of the ThunderX-1's L2 cache subsystem, which handles all the transfers on the CPU side.

Moreover, latency (time to last byte) for ECI is about *half* that of PCIe, except for the case of a large transfers over 8 KiB. Note that all large ECI transfers are simply a sequence of essentially independent low-latency cache line transactions.

The superior performance of a single ECI link, despite slightly lower theoretical bandwidth, is due to differences in protocol design. As a coherence protocol, ECI is optimized for latency of 128-byte cache line transfers. PCIe, in contrast, is designed for throughput with an upfront cost to set up a large transfer.

Significantly, however, many real applications require transfer sizes in the range where ECI is much more efficient than PCIe, something we confirm in Section 5.3. Superior performance for cache-line sized transfers is particularly of interest for applications that do not resemble GPU computations, for example, prototyping custom memory controllers as in Section 5.4.

This illustrates some of the advantages that can be gained from a change of design perspective, unconstrained by attachment to existing standards. By treating the FPGA as a processor on an equal footing to the CPU, we can achieve very different performance tradeoffs that can significantly benefit many classes of application.

To give a reasonable upper bound on achievable performance, we also measured the performance of cache coherence between two ThunderX-1 processors in a commercial 2-socket NUMA server, with hardware load-balancing across both links. We saw 19 GiB/s of achievable throughput, with a latency of 150ns. This is substantially lower latency than our ECI implementation, partly due to the lower clock frequency on the FPGA (300 MHz here), but suggests our throughput is comparable with the full hardware implementation.

5.2 Network (TCP/IP and RDMA)

An important feature of Enzian is its powerful network capability. We now explore the performance of networking using both TCP/IP and Remote Direct Memory Access (RDMA) stacks and discuss Enzian as a realistic platform for emulating a smart NIC, using the example of a high-performance TCP stack terminated in the FPGA.

We ported an open-source FPGA TCP/IP network stack [63] to Enzian as a Coyote service, and connected two Enzian machines through their FPGA-side 100 Gb/s Ethernet links via a conventional network switch. We compare this via `iperf` with the performance we achieve between two Intel Xeon Gold 6248 machines connected using 100 Gb/s Mellanox NICs.

Figure 7 shows that Enzian can saturate a single 100 Gb/s TCP connection with an MTU as low as 2 KiB, whereas a modern high-end machine requires multiple threads and connections to fully exploit the network link. For clarity, we omit the data for additional flows in the Xeon/Mellanox configuration; in our experiments, 4 flows are needed using the CPU to saturate the link. The stack used in Enzian has a single processing pipeline shared between all TCP connections, and so its performance is independent of the number of flows [62].

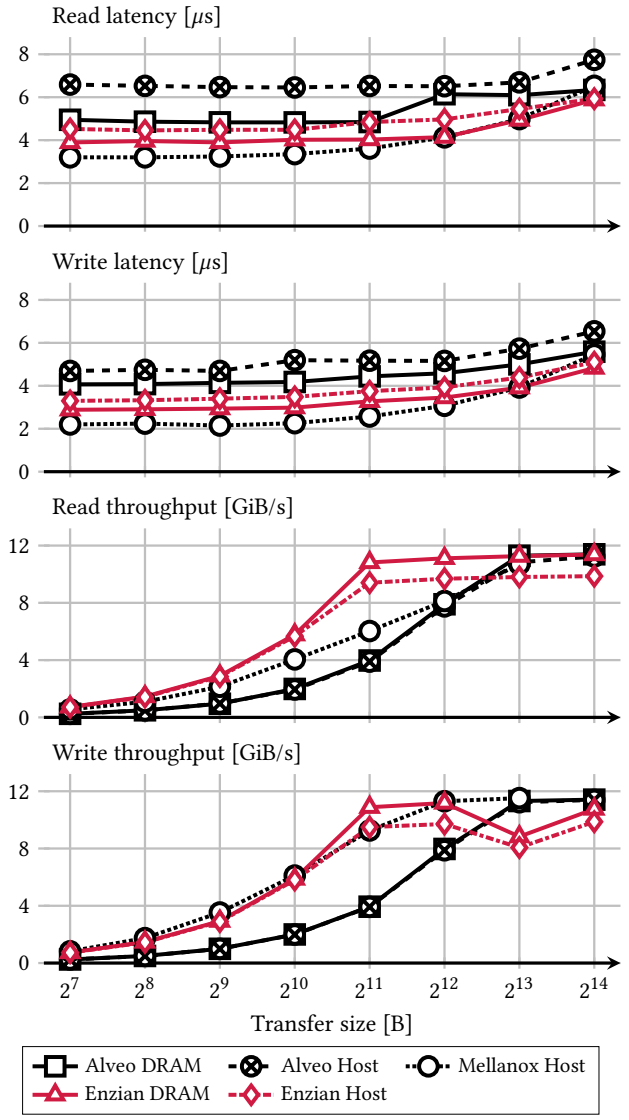

Figure 8: RDMA performance

Next, to explore how Enzian can be used as a smart NIC, we run an open source, extensible RDMA stack (StRoM [64]) on both Enzian and a commercial Xilinx Alveo 280 card in an Intel Xeon server, using the same Mellanox NIC as before as a baseline. We use a Xilinx VCU118 board to generate RDMA 1-sided copy requests over a 100 Gb/s Ethernet link.

Figure 8 shows the results of Enzian and the Alveo card accessing both "Host" (CPU) memory and DDR4 connected directly to the FPGA ("DRAM"). Note that in the "host" case, RDMA reads and writes on Enzian traverse ECI and are therefore coherent with the CPU's L2 cache.

As the figure shows, the performance of Enzian is highly competitive with both the Alveo and Mellanox platforms, and Enzian has superior throughput and latency when accessing the 512 GiB

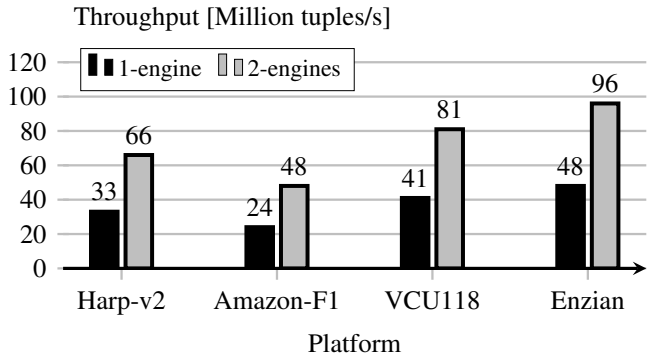

Figure 9: Gradient boosting decision trees

of DDR4 on the FPGA side. As in Section 5.1, write throughput is constrained by the use of a single ECI link.

This demonstrates that Enzian is a viable platform for prototyping smart NIC functionality. This encompasses a wide range of current research in Operating Systems, such as FlexNIC [33], Prism [6], or Dagger [39]. It also shows how Enzian can be used to implement, e.g., hardware-accelerated key-value stores [40].

Finally, we note that Enzian can also subsume the use-case for Microsoft Catapult (with equivalent performance) by connecting an additional networking cable between one of the 100 Gb/s interfaces on the XCVU9P (clocked at 10 GHz rather than 25 GHz) and one of the ThunderX-1's 40 Gb/s NICs.

5.3 PCIe Accelerator-Style Applications

Our first macro benchmark compares Enzian with commercial accelerators running a real-world workload previously published in the research literature, demonstrating that Enzian can be used in place of these platforms for such workloads.

We compare Enzian to existing results on inference over gradient boosting decision tree ensembles [52, 53]. Decision trees [47] are a popular form of supervised machine learning, and we reproduce the same experiment reported in the Coyote paper [38], together with our own results using Enzian.

This workload resembles many used on FPGA accelerators, particularly in its GPU-derived pattern of loading data from the host machine, computing on it, and copying the results back to host memory. Double-buffering is used to overlap data copying and computation, efficiently hiding latency. The design can be deployed as a single engine or as two parallel engines and we provide results for both configurations. The workload is primarily compute-bound, and uses no more than 4 GB/s of bandwidth between the FPGA and host (CPU-side) memory. We compare the performance of Enzian by reproducing existing published figures on Amazon F1 and Xilinx VCU118 [75] boards, as well as published numbers on Intel's HARPv2 platform (Broadwell+Arria in Figure 2 and Figure 3).

Figure 9 shows that for this application, Enzian outperforms all existing boards. The reason for this is that while the same FPGA is used in Enzian, F1, and the VCU118 board, Enzian employs the part variant with the highest speed available (following our principles of overengineering at the cost of unit price).

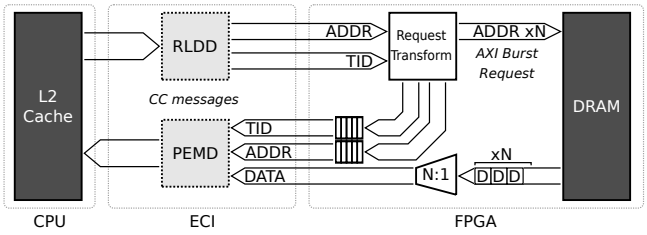

Figure 10: Coherent data reduction pipeline

This is only one workload, and we make no strong claims about Enzian being faster than commercial solutions for this or other accelerator-type applications. However, we do conclude that for fairly conventional use-cases, Enzian is highly competitive with current PCIe-based accelerators, despite the considerable additional functionality and flexibility – for example, the larger memory can be used to perform inference in much larger tree ensembles than in any currently-available system.

5.4 The FPGA as a Custom Memory Controller

This next experiment demonstrates the suitability of Enzian for prototyping custom memory controllers, such as PBerry [9], general physical memory manipulation, e.g., Cache Bleaching [58], as well as more general near-data processing. We demonstrate offloading a compute-intensive data reduction task: the colorspace transform and quantization part of a computer vision pipeline. Exploiting the cache coherence of the FPGA allows us to realistically explore the design's tradeoffs and benefits.

The offload engine uses the pipelined structure shown in Figure 10. It interacts with the raw coherence protocol packet interfaces, receiving refill requests from the CPU's L2 cache which it transforms into larger sequential burst reads from DRAM. The burst data is then fed to the data reduction module, which performs an RGB to luminance conversion (RGB2Y) and optionally quantizes to 4 bits per pixel, packing the result into a single cache line which is then returned to the CPU as a response to the refill request. The pipeline is this invisible to the CPU beyond an increase in latency. Loads appear exactly like NUMA-remote L2 refills in a 2-socket system would.

We evaluate the converter by integrating it into a machine vision pipeline, which performs RGB2Y followed by a 3×3 gaussian blur. The blur has roughly 5× the arithmetic intensity of the conversion. The FPGA is substituted for the soft RGB2Y stage: Pointing the input of the blur filter at the FPGA-backed addresses rather than the software output buffer makes the swap. Nothing else needs to be changed. We then compare the performance of both configurations.

Input data is uncompressed 1024x576 RGB video frames with 8 bits per channel pixels padded to 32 bits, preloaded into FPGA-side DRAM. We measure throughput in pixels-per-second, and additionally collect interconnect utilization, memory-dependent CPU stall cycles, and L1 refills, while varying the number of active CPU cores from 1 to 48.

Results for all configurations are plotted in Figure 11. Performance for the baseline (soft RGB2Y) scales linearly to 48 cores at 33×10^6px/s/core (125 MiB/s raw data).

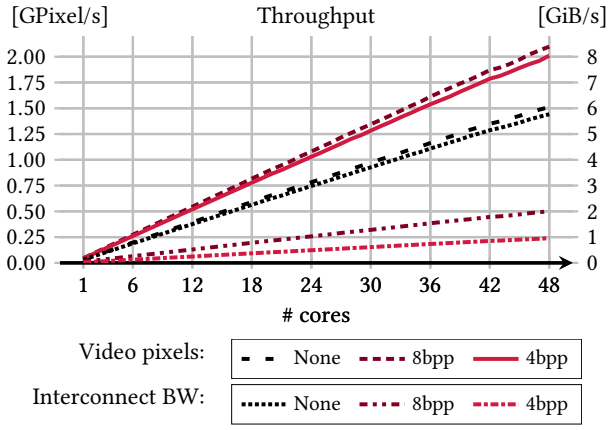

Figure 11: Pipeline throughput against active core count

Table 1: Pipeline PMU counts (48 threads)

Reduction	None	8bpp	4bpp
Memory stalls per cycle	0.025	0.005	0.005
Cycles per L1 refill ($/10^3$)	1.84	5.16	10.50

Enabling hardware RGB2Y increases throughput per core by 33% with quantization to 4 bits, or 39% without. As can be seen from the interconnect curves, the increase is due to reduced inter-socket bandwidth and thus expensive remote L2 refills (each 128-byte cache line is 32 pixels in the baseline, 128 with hard RGB2Y, and 256 with quantization).

By 8bpp the workload saturates, as the plateauing memory stall rates in Table 1 show, with no further improvement with a higher reduction ratio. The 4x data reduction from 32bpp to 8bpp translates into a 3× reduction in interconnect bandwidth relative to the previously memory-bound workload, indicating that previously idle compute cycles are now utilised, corresponding to the observed 39% throughput increase ($1.39/4 \approx 1/3$). The further 2× reduction to 4bpp gives a 2× bandwidth reduction, indicating that the workload is now compute bound. The slight throughput reduction is likely due to the increasing L2 refill latency, since we need to read 1 KiB from DRAM at this point for each cache line).

By referencing the pixels-per-second curves to the bandwidth axis (scaled to read 4B per pixel i.e. raw size), we see that moving the RGB2Y step across the interconnect (and thus making the pipeline compute bound) allows the application to increase its DRAM utilisation from 6 to 8 GiB/s.

This experiment demonstrates that Enzian is ideal for prototyping custom memory controller/near-data processing research. The open platform permits the replacement of arbitrary components, and low-level integration with the CPU's coherency protocol. Real benchmark code runs in real time exercising user-supplied custom coherent components on realistically-sized workloads (hundreds of gigabytes, tens of GB/s). Extensive instrumentation makes it straightforward to measure and investigate the interaction of tasks such as data reduction with interconnect and memory bandwidth,

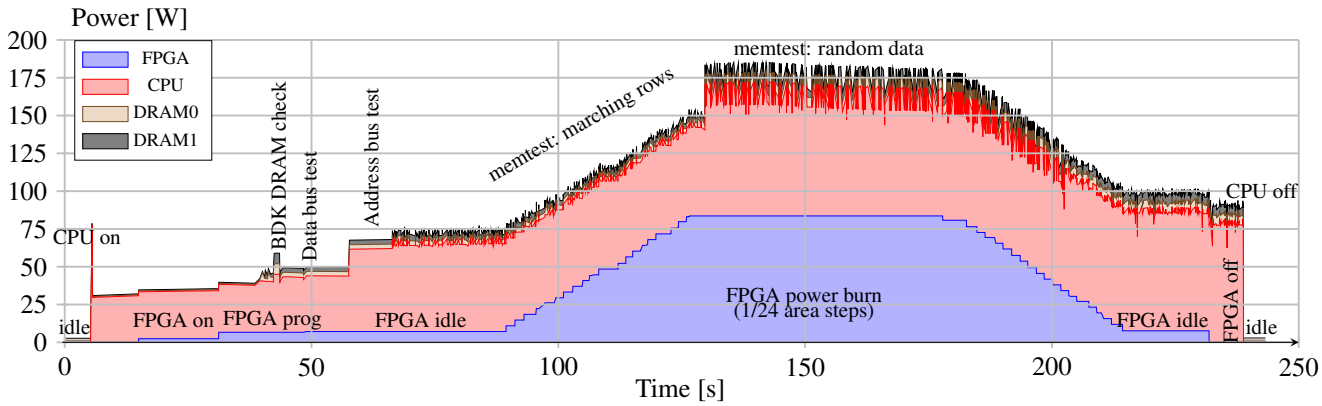

Figure 12: Power measurements of primary components during a boot, diagnostic, and stress test.

and thus make more realistic predictions about the final performance of research ideas actually implemented in silicon.

5.5 Instrumentation

Finally, we show how Enzian can perform fine-grained power monitoring, as one illustration of the benefits mentioned in Section 4.3 of a system with open hardware design and BMC.

We used the BMC to monitor the primary power regulators for the CPU and FPGA cores and the CPU-side DRAM channels, sampling each every 20 ms and collecting the data using our dbus-based telemetry service.

Figure 12 shows a time series of this power data as Enzian boots (with a power spike as the CPU is powered on), checks DRAM, runs a series of memory tests on the CPU, and then initiates an FPGA stress test by switching blocks of flip-flops on every clock cycle.

As can be seen, Enzian allows both high sampling rates and detailed per-domain measurements, allowing researchers to examine, in real time, the energy performance of hybrid applications and systems software. In practice, all the power and clock regulators in the system, together with a dozen temperature sensors, can be monitored in this way. The smaller FPGA on the BMC can also provide hardware support for such measurement, facilitating targeted and precise power and performance research.

6 FURTHER USE-CASES

We expect that the flexibility of Enzian will open up a wide range of opportunities for research into hybrid systems where an FPGA efficiently complement the CPU in different ways, beyond those we have surveyed in Section 5.

Viewing the FPGA as an *application accelerator*, we have preliminary work on relational database engines that take advantage of FPGAs as accelerators [2, 32]. These results are often limited by the PCIe bandwidth on the FPGA, whereas with the much higher bandwidth of ECI changes the balance between what can be done on the CPU and on the FPGA changes, especially when the cost and latency of data movements is reduced.

Similarly, the limited memory capacity on existing commercial FPGAs is an issue for applications where large data volumes are involved (such as data analytics and machine learning models), since the limited storage causes more data movement to and from the accelerator. With up to a terabyte of DRAM per FPGA, Enzian applications do not have to resort to host memory for large data sets. We have initial results for inference on recommendation systems [31, 79] where the models are large and where Enzian can show the advantage of keeping all the data in memory accessible to the FPGA while still consistent with CPU host memory.

We can alternatively view the FPGA as an *I/O device*. In addition to a smart NIC, the FPGA side of Enzian can also be used as a smart programmable storage controller, either with persistent storage connected via the NVMe connector or PCIe x16 slot, or instead using the large DRAM to emulate non-volatile memory. This enables experimentation at high performance with "in-storage" functionality (e.g. [36]) and also hardware support for Tiered Memory [67], to take but two examples.

Alternatively, the FPGA can function as an *instrument* for observing the CPU and its software in real-time. For example, we perform runtime verification of a combined hardware/software system at scale with zero overhead, by using the FPGA to process events from the program trace units on the ThunderX-1 cores, and compiling temporal logic assertions about the behavior of the hardware, OS, and application software into reconfigurable logic [17].

In this paper we have focused entirely on single system deployments. However, the ample bandwidth available suggests a variety of research projects enabled by a *cluster of Enzian boards* connected by a high-performance interconnect. There need not be Ethernet-based either: the 4×100 Gb/s links (or 16×25 Gb/s links) could just as easily use a novel wire protocol instead, perhaps more tailored to a programmable switch like Tofino [28].

To take one example, we have recent work on smart disaggregated memory [37] where the DRAM of the FPGA is made available as network attached memory and accessible either through RDMA, or on Enzian by extending the cache coherency protocol via a "bridge" implemented on the FPGA.

This disaggregated memory can be used, for example, as a database buffer cache with operator off-loading and push down directly to the memory. As set of Enzian nodes can be integrated into a cluster offering terabytes of network-attached DRAM with ample

network bandwidth to explore how application designs much less constrained by the limits of currently available hardware.

A final valuable feature of Enzian is simply the insight it gives into the way that modern servers are constructed, and the challenges this poses.

As discussed in Section 4.2, we are porting the sel4 microkernel [35] to the BMC. This would allow for the critical control stack to be implemented in a verifiable manner on a verified base, and non-critical components to be placed in a virtualized instance of OpenBMC [16]. Creating a correct SMBus/PMBus/I^2C stack is another area of research, building up a modular, model-checked I^2C implementation [27]. Adding a correct sequence [60] and generated device drivers (possibly in the FPGA itself) would allow for a completely verified stack of control software, eliminating not only bugs, but significant effort by engineers.

More broadly, it has been a long time since the systems software research community has had complete full-stack access to a modern, performant server. As we discovered, programming a completely open modern server platform like Enzian radically changes one's perspective on operating systems and server hardware.

7 DISCUSSION AND CONCLUSION

As shown by the examples throughout the paper, Enzian provides ample coverage of the hardware design space for prototyping and research on hybrid systems. It is overprovisioned, feature-rich, highly instrumented, and easily reconfigured from a software perspective. Since it is designed as a research platform rather than a commercial product, it not only provides an attractive application development platform itself, but can also be used to prototype potential future hardware in a realistic operating system setting. Moreover, it can be used to conduct experiments and instrument software in ways not possible on existing platforms.

At this point, the question naturally arises as to the useful lifetime of Enzian as a platform. It took us longer than we expected to get Enzian to its current state, and it will take longer before it becomes widely available to the research community. There is a real risk that it will be obsolete by the time this happens.

Our position on this has two aspects. First, as we have shown in this paper, Enzian remains at time of publication highly competitive with, and in many cases superior to, leading-edge available commercial hardware. The ThunderX-1 processor is today by no means state-of-the-art, but remains viable in a mid-range server, and the Xilinx XCVU9P remains very large by industry standards. Indeed, a noticeable trend in FPGAs is to incorporate more specialized "hard IP" functionality at the expense of reconfigurable logic, suggesting that the CPU/FPGA combination in Enzian, and its performance tradeoffs, will be relevant for some time to come.

Secondly, Enzian provides for the first time to the academic research community a fully open design for a complete server system. Ideas conceived, designed, implemented, and validated using Enzian are highly likely to be applicable to later hardware platforms, and Enzian provides an environment for experimenting with, for example, high-assurance board firmware that is unavailable elsewhere. The insight we gained into how a modern server is built today (with or without an FPGA) and the lurking problems therein have strongly influenced our current research directions, and we expect will do the same for others.

We noted in Section 3 that we had deprioritized unit cost in favor of overengineering and reducing design cost. However, while Enzian differs considerably from a regular server, the bill of materials is not too different. The unit cost of any 2-socket server is dominated by the DRAM and CPUs (including the FPGA in our case). The somewhat surprising consequence is that, given favorable pricing on the FPGA, an Enzian board only costs a small amount more than a regular 2-socket server with similar memory capacity.

At the time of writing, there are 9 working Enzian systems. We use them for our research, and also have active collaborations with other universities. Our goal is to make many more available (including for artifact evaluation), either remotely accessible or sold at cost in partnership with a hardware or systems vendor (as with the NetFPGA project [48]). All hardware design files have been open-sourced [22]. We also intend to open-source related software as much as possible.

We built Enzian to provide ourselves, and others in the systems research community, with a better platform for system software research. Thinking longer-term, Enzian is also a means for systems researchers to engage more fully with the hardware configuration and architecture, providing insights that are sometimes lacking in hardware design [45]. We might speculate further that, in the long term, if hybrid CPU/FPGA platforms are here to stay, then the current diversity of designs (with the associated problems we discussed in Section 2) is transitory, and a consensus will emerge on a single architectural pattern that is "good enough" for almost all applications, as happened with the horizontalization of the PC as a universal platform for server computing. We make no claim that Enzian is such a platform, but its design principles point in that direction.

ACKNOWLEDGEMENTS

We thank the reviewers and our shepherd, Dejan Kostic, for their helpful remarks. We are very grateful to Xilinx and Marvell for their technical help and donations of evaluation systems and components, to ETH Zurich and VMware for finanical support, and Dream Chip for board design. Many people at ETH Zurich have contributed to Enzian and its associated tools, and we have tried to list them all on the Enzian web site at https://enzian.systems. Anastasiia Ruzhanskaia and Lukas Humbel also helped with the experiments for this paper. Finally, we would like to gratefully acknowledge the feedback and encouragement we have received during the course of building Enzian from many members of the research community, both from academia and industry.

A ARTIFACT APPENDIX

A.1 Abstract

The submitted artifact consists of two major parts: the complete collection of board design files necessary for manufacturing an Enzian [22], and all bitstreams and benchmarks necessary to reproduce the evaluation results presented in Section 5 of the paper [21]. This artifact includes a detailed guide for booting and programming the machines. While the authors encourage other research groups to make use of the CAD documents to build new and better Enzians,

given the limited time and money available for artifact evaluation, we advise reviewers not to attempt to manufacture their own Enzians, and instead to proceed with the second artifact, found at Zenodo [21]. The provided bitstreams and benchmarks will allow reviewers to reproduce all presented evaluation data visualized in Figures 5, 6, 7, 8, 10, 11 and Table 1.

A.2 Artifact Check-List (Meta-Information)

- **Program: All: Linux (tested Ubuntu 20.04), provided custom kernel modules, 5.4: ffmpeg**
- **Compilation: gcc (tested 9.3.0), Xilinx Vivado (version 2020.1)**
- **Binary: 5.1-5.5 each have one or more Vivado-generated bitstreams**
- **Hardware: 5.1, 5.3-5.5: one Enzian, 5.2: two Enzians connected with 100G Ethernet cable**
- **Run-time state: The bitstreams must be loaded before booting Linux**
- **Metrics: memory throughput, latency, tuples/s, pixel/s, stall cycles, instruction retired, cycles, L1 refills, current, voltage**
- **Output: 5.1-5.3: collected data is presented in human-readable log files, 5.4-5.5: data is presented with scripts for processing into plots**
- **Experiments: 5.1: microbenchmarks, 5.2: TCP/IP, RDMA performance, 5.3: decision trees, 5.4: custom memory controller for video encoding, 5.5: power instrumentation**
- **How much disk space required (approximately)?: 600 MB for software and bitstreams**
- **How much time is needed to prepare workflow (approximately)?: 10 minutes per experiment for loading bitstream and booting machine**
- **How much time is needed to complete experiments (approximately)?: 5.4 may take 3-4 hours per configuration (3 in total) while the remaining experiments should take on average approximately 10 minutes per loaded bitstream**
- **Publicly available?: Yes**
- **Code licenses (if publicly available)?: Creative Commons Attribution 4.0 International**
- **Archived (provide DOI)?: Design files: 20.500.11850/517619, Bitstreams and benchmarks: 10.5281/zenodo.5729175**

A.3 Description

A.3.1 How to Access. The primary artifact, the design files of the system itself, can be found in the ETH Research Collection [22]. The bitstreams and benchmarks used for reproducing the data presented in the paper can be found on Zenodo [21].

A.3.2 Hardware Dependencies. All experiments must be run on one or more Enzian machines. As the authors are in possession of all nine extant Enzians, reviewers will have to program and access the machines via ssh through our gateway. There are unpublished comparative results presented in the paper that use one or two Xilinx Alveo U280 cards. The XACC-ETHZ cluster (https://xilinx.github.io/xacc/) can be used to reproduce this data.

A.3.3 Software Dependencies. Xilinx Vivado (2020.1) is required for loading the necessary bitstreams, however the entire toolchain is available on the gateway machine used for accessing the Enzian cluster.

A.4 Installation

All of the necessary software will be installed on all Enzian machines. The bitstreams will be available on the gateway server that contain the Vivado toolchain necessary for programming the FPGAs. If reviewers would like to install the artifact, it can be unzipped into the user's home directory and specific details of necessary scripts and benchmarks are in the `README.md` files in their respective directories.

A.5 Experiment Workflow

All experiments follow a similiar initialization procedure, outlined in detail in the Enzian Quickstart Guide found in the root directory of the artifact. Moreover, detailed instructions for running each experiment are included in the `README.md` included in each experimental directory. The basic outline of the workflow is as follows:

(1) From the gateway server, take the BMC and CPU consoles of the machine under test using the commands `console zuestollXX-bmc` and `console zuestollXX-console`, respectively.
(2) From the BMC power manager, power on the PSU using `common_power_up()`.
(3) Power on the CPU using `cpu_power_up()`.
(4) On the CPU console, the BDK boot menu will appear. Break the boot by pressing `B`.
(5) From the gateway server, program the bitstream for the experiment using the included tcl script.
(6) When the bitstream is programmed, the CPU boot process can resume, bringing the ECI link up, and then booting into Linux.

Experiments 5.1, 5,2, 5.3, and 5.4 require the bitstream to be programmed, the ECI link to come up, and the CPU to boot into Linux. 5.5 does not make use of ECI and all CPU activities are initiated from the BDK.

A.6 Evaluation and Expected Results

A.6.1 Microbenchmarks (Section 5.1). The microbenchmarks measure the throughput and latency of a single ECI link for both reads (CPU to FPGA) and writes (FPGA to CPU) and compares with those of PCIe x16 Gen3 interconnect. Two sets of software and bitstream are available to compare performance. The ECI performance can be measured by programming the provided bitstream and running the software on an Enzian machine. It can be compared with the performance of PCIe by programming the corresponding bitstream on an Alveo card and running its software. Detailed instructions along with the results used in the paper are in the `README.md` file.

A.6.2 Network Performance (Section 5.2). These experiments evaluate two network benchmarks, an TCP/IP FPGA stack, and RDMA.

For the TCP/IP experiment, two enzian FPGAs are connected to a 100 Gbps switch. One FPGA is the server and another is the client. We perform a ping-pong experiment between two FPGAs with the client sending a pre-defined amount of data to the server. Once the expected amount of data is received, the server sends back the same amount of data back to the client. The single trip data transfer latency is measured as the half of the total transfer latency.

RDMA measurements can be taken between two Enzian boards and two Alveo cards. In the case of Enzians, the memory being accessed is over ECI (CPU DRAM) while the Alveo case accesses memory directly attached to the FPGA.

A.6.3 Decision Trees (Section 5.3). The experiment performs gradient boosting decision tree inference on a given model in three steps. The first step is offloading the model and is not part of measurements. The second step is inference where data is offloaded in a streaming fashion into a pipeline where, as the third step, results are computed and written back to the host memory. The experiment uses 64KB of tuples to hit saturation point.

A.6.4 Custom Memory Controller (Section 5.4). The series of experiments in this section evaluates the use case of Enzian as a custom memory controller for video processing. The application has RGB data stored in the FPGA memory and performs RGB to luminance conversion, blur and edge detect (optional). The RGB to luminance conversion can be offloaded to the FPGA where the FPGA can provide luminance either at 8bits per pixel or 4 bits per pixel. By quantizing data, packing them and providing the luminance "view" to the CPU, the FPGA effectively reduces the number of memory operations that need to be performed by the CPU.

There are three experiments in this section:

- RGBA - No processing on the FPGA, CPU reads RGB data from FPGA memory and performs all operations.
- Y8 - FPGA converts RGB data to 8bit luminance per pixel and provides it to the CPU. The CPU performs rest of the operations.
- Y4 - FPGA converts RGB data to 4bit luminance per pixel and provides it to the CPU. The CPU performs rest of the operations.

Each experiment has a separate bitstream that has to be programmed onto the FPGA to run the experiment. Once the FPGA is programmed, the application can be launched on CPU to collect performance numbers and reproduce graphs. Each experiment takes approximately 3 hours to complete and detailed instructions are provided in the `README.md` in the experiment directory. The logs for results used in the paper are available along with the artifacts.

A.6.5 Power Instrumentation (Section 5.5). In the experiment we log measurements of various subsystem regulators, run varied diagnostic and stress tests, and use the current data in order to calculate the power consumed. A subset of measurements can be collected in a log at 10 ms intervals, or manually measured using the `print_current_all()` from the BMC power manager.

A.7 Notes

When using the BMC, please only use commands found in the instructions above and in the Quickstart Guide.

A.8 Methodology

Submission, reviewing and badging methodology:

- https://www.acm.org/publications/policies/artifact-review-badging
- http://cTuning.org/ae/submission-20201122.html
- http://cTuning.org/ae/reviewing-20201122.html

REFERENCES

[1] Alibaba Cloud Services. 2020. Compute optimized instance families with FPGAs. https://www.alibabacloud.com/help/doc-detail/108504.htm.

[2] Gustavo Alonso, Timothy Roscoe, David Cock, Mohsen Owaida, Kaan Kara, Dario Korolija, David Sidler, and Zeke Wang. 2020. Tackling Hardware/Software co-design from a database perspective. In *Proceedings of the 6th biennial Conference on Innovative Data Systems Research (CIDR)*. Amsterdam, Netherlands.

[3] Alpha Data 2019. *ADM-PCIE-7V3 Datasheet*. Alpha Data. https://www.alpha-data.com/pdfs/adm-pcie-7v3.pdf

[4] Jeff Barr. 2016. *Developer Preview – EC2 Instances (F1) with Programmable Hardware*. Technical Report. Amazon AWS. https://aws.amazon.com/blogs/aws/developer-preview-ec2-instances-f1-with-programmable-hardware/

[5] Thomas Burd, Noah Beck, Sean White, Milam Paraschou, Nathan Kalyanasundharam, Gregg Donley, Alan Smith, Larry Hewitt, and Samuel Naffziger. 2018. Zeppelin: An SoC for Multichip Architectures. *IEEE Journal of Solid-State Circuits* 54, 1 (2018), 133–143.

[6] Matthew Burke, Sowmya Dharanipragada, Shannon Joyner, Adriana Szekeres, Jacob Nelson, Irene Zhang, and Dan R. K. Ports. 2021. PRISM: Rethinking the RDMA Interface for Distributed Systems. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles* (Virtual Event, Germany) *(SOSP '21)*. Association for Computing Machinery, New York, NY, USA, 228–242. https://doi.org/10.1145/3477132.3483587

[7] Anthony M. Cabrera and Roger D. Chamberlain. 2019. Exploring Portability and Performance of OpenCL FPGA Kernels on Intel HARPv2. In *Proceedings of the International Workshop on OpenCL* (Boston, MA, USA) *(IWOCL'19)*. Association for Computing Machinery, New York, NY, USA, Article 3, 10 pages. https://doi.org/10.1145/3318170.3318180

[8] Irina Calciu, M. Talha Imran, Ivan Puddu, Sanidhya Kashyap, Hasan Al Maruf, Onur Mutlu, and Aasheesh Kolli. 2021. Rethinking Software Runtimes for Disaggregated Memory. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (Virtual, USA) *(ASPLOS 2021)*. Association for Computing Machinery, New York, NY, USA, 79–92. https://doi.org/10.1145/3445814.3446713

[9] Irina Calciu, Ivan Puddu, Aasheesh Kolli, Andreas Nowatzyk, Jayneel Gandhi, Onur Mutlu, and Pratap Subrahmanyam. 2019. Project PBerry: FPGA Acceleration for Remote Memory. In *Proceedings of the Workshop on Hot Topics in Operating Systems* (Bertinoro, Italy) *(HotOS '19)*. Association for Computing Machinery, New York, NY, USA, 127–135. https://doi.org/10.1145/3317550.3321424

[10] Adrian M. Caulfield, Eric S. Chung, Andrew Putnam, Hari 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 *The 49th Annual IEEE/ACM International Symposium on Microarchitecture* (Taipei, Taiwan) *(MICRO-49)*. IEEE Press, Article 7, 13 pages.

[11] CCIX Consortium and others. 2019. Cache Coherent Interconnect for Accelerators (CCIX). http://www.ccixconsortium.com.

[12] Kevin K. Chang, A. Giray Yağlıkçı, Saugata Ghose, Aditya Agrawal, Niladrish Chatterjee, Abhijith Kashyap, Donghyuk Lee, Mike O'Connor, Hasan Hassan, and Onur Mutlu. 2017. Understanding Reduced-Voltage Operation in Modern DRAM Devices: Experimental Characterization, Analysis, and Mechanisms. *Proc. ACM Meas. Anal. Comput. Syst.* 1, 1, Article 10 (June 2017), 42 pages. https://doi.org/10.1145/3084447

[13] Young-kyu Choi, Jason Cong, Zhenman Fang, Yuchen Hao, Glenn Reinman, and Peng Wei. 2016. A Quantitative Analysis on Microarchitectures of Modern CPU-FPGA Platforms. In *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)* (Austin, TX, USA). IEEE Press, 1–6. https://doi.org/10.1145/2897937.2897972

[14] Young-Kyu Choi, Jason Cong, Zhenman Fang, Yuchen Hao, Glenn Reinman, and Peng Wei. 2019. In-Depth Analysis on Microarchitectures of Modern Heterogeneous CPU-FPGA Platforms. *ACM Trans. Reconfigurable Technol. Syst.* 12, 1, Article 4 (Feb. 2019), 20 pages. https://doi.org/10.1145/3294054

[15] Eric Chung, Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Adrian Caulfield, Todd Massengill, Ming Liu, Mahdi Ghandi, Daniel Lo, Steve Reinhardt, Shlomi Alkalay, Hari Angepat, Derek Chiou, Alessandro Forin, Doug Burger, Lisa Woods, Gabriel Weisz, Michael Haselman, and Dan Zhang. 2018. Serving DNNs in Real Time at Datacenter Scale with Project Brainwave. *IEEE Micro* 38 (March 2018), 8–20. https://www.microsoft.com/en-us/research/publication/serving-dnns-real-time-datacenter-scale-project-brainwave/

[16] Darren Cofer, Andrew Gacek, John Backes, Michael W. Whalen, Lee Pike, Adam Foltzer, Michal Podhradsky, Gerwin Klein, Ihor Kuz, June Andronick, Gernot Heiser, and Douglas Stuart. 2018. A Formal Approach to Constructing Secure Air Vehicle Software. *Computer* 51, 11 (2018), 14–23. https://doi.org/10.1109/MC.2018.2876051

[17] Lukas Convent, Sebastian Hungerecker, Torben Scheffel, Malte Schmitz, Daniel Thoma, and Alexander Weiss. 2018. Hardware-Based Runtime Verification with Embedded Tracing Units and Stream Processing. In *Runtime Verification*, Christian Colombo and Martin Leucker (Eds.). Springer International Publishing, Cham, 43–63.

[18] Louise Helen Crockett, Ross Elliot, Martin Enderwitz, and Robert Stewart. 2014. *The Zynq Book: Embedded Processing with the Arm Cortex-A9 on the Xilinx Zynq-7000 All Programmable SoC*. Strathclyde Academic Media.

[19] CXL Consortium. 2020. Compute Express Link. https://www.computeexpresslink.org/.

[20] devicetree.org. 2020. *Devicetree Specification* (release v0.3 ed.). `devicetree.org`. https://www.devicetree.org/specifications.

[21] Enzian Team. 2021. ASPLOS 2022 - Artifact. https://doi.org/10.5281/zenodo.5729174.

[22] Enzian Team. 2021. Enzian Board Design Files. https://doi.org/10.5281/zenodo.5802292.

[23] Tian Fang. 2014. *Introducing "OpenBMC": an open software framework for next-generation system management.* Technical Report. Facebook Engineering. https://engineering.fb.com/open-source/introducing-openbmc-an-open-software-framework-for-next-generation-system-management/

[24] Daniel Firestone, Andrew Putnam, Sambhrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian Caulfield, Eric Chung, Harish Kumar Chandrappa, Somesh Chaturmohta, Matt Humphrey, Jack Lavier, Norman Lam, Fengfen Liu, Kalin Ovtcharov, Jitu Padhye, Gautham Popuri, Shachar Raindel, Tejas Sapre, Mark Shaw, Gabriel Silva, Madhan Sivakumar, Nisheeth Srivastava, Anshuman Verma, Qasim Zuhair, Deepak Bansal, Doug Burger, Kushagra Vaid, David A. Maltz, and Albert Greenberg. 2018. Azure Accelerated Networking: SmartNICs in the Public Cloud. In *Proceedings of the 15th USENIX Conference on Networked Systems Design and Implementation* (Renton, WA, USA) *(NSDI'18)*. USENIX Association, USA, 51–64.

[25] 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 *Proceedings of the 45th Annual International Symposium on Computer Architecture* (Los Angeles, California) *(ISCA '18)*. IEEE Press, 1–14. https://doi.org/10.1109/ISCA.2018.00012

[26] Sara Hooker. 2021. The Hardware Lottery. *Commun. ACM* 64, 12 (nov 2021), 58–65.

[27] Lukas Humbel, Daniel Schwyn, Nora Hossle, Roni Haecki, Melissa Licciardello, Jan Schaer, David Cock, Michael Giardino, and Timothy Roscoe. 2021. A Model-Checked I^2C Specification. In *International Symposium on Model Checking Software*. Springer, 177–193. https://doi.org/10.1007/978-3-030-84629-9_10

[28] Intel 2016. *Tofino*. Intel. https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-series/tofino.html

[29] Intel 2019. *Intel Acceleration Stack for Intel® Xeon® CPU with FPGAs Core Cache Interface (CCI-P) Reference Manual*. Intel. MNL-1092, https://www.intel.com/content/www/us/en/programmable/documentation/buf1506187769663.html.

[30] Intel 2019. *Intel FPGA Programmable Acceleration Card D5005 Data Sheet* (ds-1058 ed.). Intel. https://www.intel.com/content/www/us/en/programmable/documentation/cvl1520030638800.html.

[31] Wenqi Jiang, Zhenhao He, Shuai Zhang, Kai Zeng, Liang Feng, Jiansong Zhang, Tongxuan Liu, Yong Li, Jingren Zhou, Ce Zhang, and Gustavo Alonso. 2021. FleetRec: Large-Scale Recommendation Inference on Hybrid GPU-FPGA Clusters. In *KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*.

[32] Kaan Kara, Jana Giceva, and Gustavo Alonso. 2017. FPGA-based Data Partitioning. In *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017*.

[33] Antoine Kaufmann, Simon Peter, Naveen Kr. Sharma, Thomas Anderson, and Arvind Krishnamurthy. 2016. High Performance Packet Processing with FlexNIC. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems* (Atlanta, Georgia, USA) *(ASPLOS '16)*. Association for Computing Machinery, New York, NY, USA, 67–81. https://doi.org/10.1145/2872362.2872367

[34] 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 *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation* (Carlsbad, CA, USA) *(OSDI'18)*. USENIX Association, USA, 107–127.

[35] Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. 2009. SeL4: Formal Verification of an OS Kernel. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles* (Big Sky, Montana, USA) *(SOSP '09)*. Association for Computing Machinery, New York, NY, USA, 207–220. https://doi.org/10.1145/1629575.1629596

[36] Jinhyung Koo, Junsu Im, Jooyoung Song, Juhyung Park, Eunji Lee, Bryan S. Kim, and Sungjin Lee. 2021. Modernizing File System through In-Storage Indexing. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*. USENIX Association, 75–92. https://www.usenix.org/conference/osdi21/presentation/koo

[37] Dario Korolija, Dimitris Koutsoukos, Kimberly Keeton, Konstantin Taranov, Dejan Milojicic, and Gustavo Alonso. 2022. Farview: Disaggregated Memory with operator Off-loading for Database Engines. In *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*. Santa Cruz, CA, USA.

[38] 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). USENIX Association, 991–1010. https://www.usenix.org/conference/osdi20/presentation/roscoe

[39] Nikita Lazarev, Shaojie Xiang, neil Adit, Zhiru Zahng, and Christina Delimitrou. 2021. Dagger: Efficient and Fast RPCs in Cloud Microservices with Near-Memory Reconfigurable NICs. In *ASPLOS*.

[40] Bojie Li, Zhenyuan Ruan, Wencong Xiao, Yuanwei Lu, Yongqiang Xiong, Andrew Putnam, Enhong Chen, and Lintao Zhang. 2017. KV-Direct: High-Performance In-Memory Key-Value Store with Programmable NIC. In *SOSP*.

[41] Linux Kernel Documentation. 2019. *Device Tree Source Format* (version 1.0 ed.). https://git.kernel.org/pub/scm/utils/dtc/dtc.git/plain/Documentation/dts-format.txt, accessed 30. December 2019.

[42] 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 *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems* (Lausanne, Switzerland) *(ASPLOS '20)*. Association for Computing Machinery, New York, NY, USA, 827–844. https://doi.org/10.1145/3373376.3378482

[43] Jakob Meier. 2020. *Tools for Cache Coherence Protocol Interoperability*. Master's thesis. Department of Computer Science, ETH Zurich.

[44] Mellanox. 2020. Mellanox Innova™-2 FlexOpen Programmable SmartNIC. https://www.mellanox.com/files/doc-2020/pb-innova-2-flex.pdf.

[45] Jeffrey C. Mogul, Andrew Baumann, Timothy Roscoe, and Livio Soares. 2011. Mind the Gap: Reconnecting Architecture and OS Research. In *Proceedings of the 13th USENIX Conference on Hot Topics in Operating Systems* (Napa, California) *(HotOS'13)*. USENIX Association, USA, 1.

[46] Janani Mukundan, Hillery Hunter, Kyu-hyoun Kim, Jeffrey Stuecheli, and José F. Martínez. 2013. Understanding and Mitigating Refresh Overheads in High-Density DDR4 DRAM Systems. In *Proceedings of the 40th Annual International Symposium on Computer Architecture* (Tel-Aviv, Israel) *(ISCA '13)*. Association for Computing Machinery, New York, NY, USA, 48–59. https://doi.org/10.1145/2485922.2485927

[47] Alexey Natekin and Alois Knoll. 2013. Gradient Boosting Machines, A Tutorial. *Frontiers in neurorobotics* (2013), 21.

[48] NetFPGA. 2021. The NetFPGA Project. https://netfpga.org/.

[49] Cavium (now Marvell). 2017. *Cavium ThunderX CN88XX, Pass 2*. Document number CN88XX-HM-2.7P.

[50] Cavium (now Marvell). 2017. ThunderX Family of Workload Optimized ARMv8 Processors. https://www.marvell.com/server-processors/thunderx-arm-processors/.

[51] Neal Oliver, Rahul R. Sharma, Stephen Chang, Bhushan Chitlur, Elkin Garcia, Joseph Grecco, Aaron Grier, Nelson Ijih, Yaping Liu, Pratik Marolia, Henry Mitchel, Suchit Subhaschandra, Arthur Sheiman, Tim Whisonant, and Prabhat Gupta. 2011. A Reconfigurable Computing System Based on a Cache-Coherent Fabric. In *Proceedings of the 2011 International Conference on Reconfigurable Computing and FPGAs (RECONFIG '11)*. IEEE Computer Society, USA, 80–85. https://doi.org/10.1109/ReConFig.2011.4

[52] Muhsen Owaida and Gustavo Alonso. 2020. Distributed Inference over Decision Tree Ensembles. https://github.com/fpgasystems/Distributed-DecisionTrees.

[53] Muhsen Owaida, Amit Kulkarni, and Gustavo Alonso. 2019. Distributed Inference over Decision Tree Ensembles on Clusters of FPGAs. *ACM Trans. Reconfigurable Technol. Syst.* 12, 4, Article 17 (Sept. 2019), 27 pages. https://doi.org/10.1145/3340263

[54] Larry Peterson, Tom Anderson, David Culler, and Timothy Roscoe. 2003. A Blueprint for Introducing Disruptive Technology into the Internet. *SIGCOMM Comput. Commun. Rev.* 33, 1 (Jan. 2003), 59–64. https://doi.org/10.1145/774763.774772

[55] Trusted Firmware Project. [n.d.]. Trusted Firmware-A (TF-A). https://developer.arm.com/tools-and-software/open-source-software/firmware/trusted-firmware/trusted-firmware-a. Previously ARM Trusted Firmware.

[56] Andrew Putnam, Adrian M. Caulfield, Eric S. Chung, Derek Chiou, Kypros Constantinides, John Demme, Hadi Esmaeilzadeh, Jeremy Fowers, Gopi Prashanth Gopal, Jan Gray, Michael Haselman, Scott Hauck, Stephen Heil, Amir Hormati, Joo-Young Kim, Sitaram Lanka, James Larus, Eric Peterson, Simon Pope, Aaron Smith, Jason Thong, Phillip Yi Xiao, and Doug Burger. 2014. A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services. *SIGARCH Comput. Archit. News* 42, 3 (June 2014), 13–24. https://doi.org/10.1145/2678373.2665678

[57] Weikang Qiao, Jieqiong Du, Zhenman Fang, Libo Wang, Michael Lo, Mau-Chung Frank Chang, and Jason Cong. 2018. High-Throughput Lossless Compression on Tightly Coupled CPU-FPGA Platforms. In *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (Monterey, CALIFORNIA, USA) *(FPGA '18)*. Association for Computing Machinery, New York, NY, USA, 291. https://doi.org/10.1145/3174243.3174987

[58] Shahin Roozkhosh and Renato Mancuso. 2020. The Potential of Programmable Logic in the Middle: Cache Bleaching. In *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 296–309. https://doi.org/10.1109/RTAS48715.2020.00006

[59] Behzad Salami, Osman Unsal, and Adrian Cristal. 2018. Fault Characterization Through FPGA Undervolting. In *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*. 85–853. https://doi.org/10.1109/FPL.2018.00023

[60] Jasmin Schult, Daniel Schwyn, Michael Giardino, David Cock, Reto Achermann, and Timothy Roscoe. 2021. Declarative Power Sequencing. *ACM Trans. Embed. Comput. Syst.* 20, 5s, Article 84 (sep 2021), 21 pages. https://doi.org/10.1145/3477039

[61] NXP Semiconductors. 2014. I2C-bus specification and user manual. https://www.nxp.com/docs/en/user-guide/UM10204.pdf. Rev. 6.

[62] David Sidler, Gustavo Alonso, Michaela Blott, Kimon Karras, Kees Vissers, and Raymond Carley. 2015. Scalable 10Gbps TCP/IP Stack Architecture for Reconfigurable Hardware. In *Proceedings of the 2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM '15)*. IEEE Computer Society, USA, 36–43. https://doi.org/10.1109/FCCM.2015.12

[63] David Sidler, Monica Chiosa, Zhenhao He, Mario Ruiz, Kimon Karras, and Lisa Liu. 2020. Scalable Network Stack supporting TCP/IP, RoCEv2, UDP/IP at 10-100Gbit/s. https://github.com/fpgasystems/fpga-network-stack.git.

[64] David Sidler, Zeke Wang, Monica Chiosa, Amit Kulkarni, and Gustavo Alonso. 2020. StRoM: Smart Remote Memory. In *Proceedings of the Fifteenth European Conference on Computer Systems* (Heraklion, Greece) *(EuroSys '20)*. Association for Computing Machinery, New York, NY, USA, Article 29, 16 pages. https://doi.org/10.1145/3342195.3387519

[65] Jeffrey Stuecheli, Bart Blaner, CR Johns, and MS Siegel. 2015. CAPI: A Coherent Accelerator Processor Interface. *IBM Journal of Research and Development* 59, 1 (2015), 7:1–7:7. https://doi.org/10.1147/JRD.2014.2380198

[66] J. Stuecheli, W. J. Starke, J. D. Irish, L. B. Arimilli, D. Dreps, B. Blaner, C. Wollbrink, and B. Allison. 2018. IBM POWER9 Opens up a New Era of Acceleration Enablement: OpenCAPI. *IBM J. Res. Dev.* 62, 4–5 (July 2018), 8:1–8:8. https://doi.org/10.1147/JRD.2018.2856978

[67] K. Sudan, K. Rajamani, W. Huang, and J. B. Carter. 2012. Tiered Memory: An Iso-Power Memory Architecture to Address the Memory Power Wall. *IEEE Trans. Comput.* 61, 12 (dec 2012), 1697–1710. https://doi.org/10.1109/TC.2012.119

[68] System Management Interface Forum. 2018. System Management Bus (SMBus) Specification. http://www.smbus.org/specs/index.html. v3.1.

[69] System Management Interface Forum (SMIF), Inc. 2020. PMBus^{TM} Power System Management Protocol Specification, revision 1.2. http://www.powersig.org/.

[70] Neil C. Thompson and Svenja Spanuth. 2021. The Decline of Computers as a General Purpose Technology. *Commun. ACM* 64, 3 (Feb. 2021), 64–72. https://doi.org/10.1145/3430936

[71] Konstantinos Tovletoglou, Lev Mukhanov, Georgios Karakonstantis, Athanasios Chatzidimitriou, George Papadimitriou, Manolis Kaliorakis, Dimitris Gizopoulos, Zacharias Hadjilambrou, Yiannakis Sazeides, Alejandro Lampropulos, Shidhartha Das, and Phong Vo. 2018. Measuring and Exploiting Guardbands of Server-Grade ARMv8 CPU Cores and DRAMs. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. 6–9. https://doi.org/10.1109/DSN-W.2018.00013

[72] Steven J. Vaughan-Nichols. 2017. MINIX: Intel's hidden in-chip operating system. Online. https://www.zdnet.com/article/minix-intels-hidden-in-chip-operating-system/. Accessed: 09.10.2020.

[73] Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. 2002. An Integrated Experimental Environment for Distributed Systems and Networks. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (Copyright Restrictions Prevent ACM from Being Able to Make the PDFs for This Conference Available for Downloading)* (Boston, Massachusetts) *(OSDI '02)*. USENIX Association, USA, 255–270.

[74] Wikipedia. 2022. ATX. https://en.wikipedia.org/wiki/ATX.

[75] Xilinx. 2018. VCU118 Evaluation Board User Guide. https://www.xilinx.com/support/documentation/boards_and_kits/vcu118/ug1224-vcu118-eval-bd.pdf.

[76] Xilinx 2020. *Alveo U200 and U250 Data Center Accelerator Cards Data Sheet* (v.1.3.1 ed.). Xilinx. https://www.xilinx.com/products/boards-and-kits/alveo/u250.html.

[77] Xilinx 2020. *Alveo U280 Data Center Accelerator Card Data Sheet* (v.1.3 ed.). Xilinx. https://www.xilinx.com/products/boards-and-kits/alveo/u280.html.

[78] Xilinx. 2021. UltraScale Architecture and Product Data Sheet: Overview. https://www.xilinx.com/support/documentation/data_sheets/ds890-ultrascale-overview.pdf. DS890 (v4.0).

[79] Yu Zhu, Zhenhao He, Wenqi Jiang, Kai Zeng, Jingren Zhou, and Gustavo Alonso. 2021. Distributed Recommendation Inference on FPGA Clusters. In *31st International Conference on Field-Programmable Logic and Applications, FPL 2021*.

[80] Patrick Ziegler. 2020. *A Unified Approach to Simulation of Hybrid CPU/FPGA systems*. Bachelor Thesis. Department of Computer Science, ETH Zurich.