Trading Latency for Compute in the Network

Pietro Bressana
Università della Svizzera italiana
pietro.bressana@usi.ch

Noa Zilberman
University of Oxford
noa.zilberman@eng.ox.ac.uk

Dejan Vucinic
Western Digital Research
dejan.vucinic@wdc.com

Robert Soulé
Yale University
robert.soule@yale.edu

ABSTRACT

This paper proposes a new heterogeneous approach to programmable architecture that extends the capabilities of programmable switch ASICs with FPGAs. It identifies the key challenges in building a heterogeneous network architecture, and presents a concrete design and implementation based around a proof-of-concept data deduplication application. Our prototype demonstrates the use of a programmable network switch and FPGAs to accelerate storage fingerprinting, running at 10G and 100G at line rate. Our approach is modular, scalable and generalizes to a wide range of applications.

CCS CONCEPTS

• **Networks** → **Network architectures**; • **Computer systems organization** → **Heterogeneous (hybrid) systems**.

ACM Reference Format:
Pietro Bressana, Noa Zilberman, Dejan Vucinic, and Robert Soulé. 2020. Trading Latency for Compute in the Network. In *Workshop on Network Application Integration/CoDesign (NAI'20), August 14, 2020, Virtual Event, NY, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3405672.3405807

1 INTRODUCTION

The end of Moore's law and the limits of Dennard scaling have increased interest in the use of domain-specific processors as an alternative to general purpose compute devices. Domain-specific processors offer advantages not just because of the particular capabilities of the hardware, but also by virtue of *where* the hardware is placed. For example, computations might be performed in the network on data in-flight.

This has led to a blurring of the traditional division of labor between network and applications, as researchers have explored *in-network computing* (INC) [8, 11, 12] as a way to accelerate applications and services. Although in-network computing is still in its infancy, early research results are promising, often providing an order-of-magnitude (or more) increase in throughput and significant reductions in latency.

However, these performance improvements involve a trade-off. On the one hand, link speeds are fundamentally different between software, FPGA, and ASIC, with a gain of about 10× at each step. But, on the other hand, the expressiveness and flexibility of the hardware decreases inversely to the performance, strictly limiting the classes of applications that can be accelerated. So, while load-balancing [16] or consensus [8, 12] might be reasonable candidates for acceleration, in-network computing would unlikely be able to accelerate, for example, a Monte Carlo simulation.

It is therefore natural to ask: *is the exchange between performance and expressivity a characteristic attribute of in-network computing, or is there a way to navigate the trade-off?* We argue for the latter, by proposing a "compute hierarchy" for in-network computing. Concretely, we present a new heterogeneous approach to programmable architecture that extends the capabilities of programmable switch ASICs with FPGAs, thereby providing a path for accelerating a greater diversity of applications in the network.

Designing such an heterogeneous architecture is hard, due to the mismatch between different technologies. The mismatch is not only in expressiveness and flexibility, but also in performance and resources. Consequently, it is required not only that switch ASIC and an FPGA will *interoperate*, but also that they will *compensate* for the difference in performance, and *orchestrate* it in a manner that guarantees mandatory networking functionality remains unaffected.

There is likely no one-size-fits-all architecture that will work for all applications. It is therefore important that system designers understand the key design decisions that follow these challenges. In this paper, we identify and elaborate these choices in depth. Moreover, we implement and evaluate one approach that we think will be broadly applicable.

As a proof-of-concept, we have implemented a fingerprint computation—i.e., mapping a large data item to a smaller bit string—on our architecture, realized with a Barefoot Tofino [17] ASIC and a NetFPGA SUME [22] FPGA. Our evaluation shows that the prototype runs at line-rate with a minimal impact on resources. Moreover, we demonstrate that using more recent FPGA technology and 100GE ports would scale up in throughput, while providing a reduction in latency.

In summary, this paper makes the following contributions:

- It identifies benefits and limitations of programmable network ASICs and FPGA-based network devices (§2).

NAI'20, August 14, 2020, Virtual Event, NY, USA

ACM ISBN 978-1-4503-8044-7/20/08…$15.00
https://doi.org/10.1145/3405672.3405807

- It argues for creating a compute hierarchy for INC (§3).
- It identifies key challenges in building a heterogeneous network architecture (§4).
- It provides a concrete implementation and proof-of-concept data deduplication application (§5).
- It validates the proposed approach in terms of performance, resource utilization, and power consumption (§6).

2 BACKGROUND

Programmable Switch ASICs. Programmable switch ASICs are optimized for packet processing at very high rate. To program these devices, developers use high-level, domain-specific programming languages, in a way that is similar to software development. Compared to other hardware, ASICs provide the best performance and power efficiency, measured as number of network operations computed per watt [20]. On the other hand, meeting such strict performance requirements limits the flexibility of these devices and the amount of resources they offer. Programmable ASICs usually offer a small number of stages in their pipeline (i.e., tens of stages [17]), limited amount of memory per stage (in the form of network tables and SRAM banks), and simple functional blocks supporting a very limited instruction set.

Stateful operations and off-chip memory are limited in most programmable switches. Due to the fixed, vendor-specific architecture implemented by programmable network ASICs, developers cannot implement custom functions on a switch. However, they can leverage existing functions, as long as the silicon vendor has built those into the switch in advance.

FPGA-based Network Devices. FPGAs offer significant flexibility, and allow the implementation of any application that can fit within the available chip resources. FPGAs today are equipped with various types of resources, including microcontrollers, microprocessors, DSPs and on-board memory. The I/O on the FPGA can support different interconnects and protocols. FPGAs are suitable for accelerating highly parallelized tasks, potentially implementing many different computing cores within the same design. enable network programmability on FPGA hardware (e.g., [9]). The flexibility of FPGA comes at the cost of performance; device frequency is significantly lower than switch ASIC, and resource usage is not as well optimized. Furthermore, FPGAs offer much lower performance-per-watt compared to ASICs.

3 BUILDING COMPUTING HIERARCHIES

Although several research prototypes have demonstrated the potential for in-network computing [8, 11, 12], we have yet to see wide spread adoption of switches as computing platforms. One reason is that in-network computing depends on platform-specific function modules (externs), meaning that a program can not be easily (or at all) ported between different types of devices. Another reason is that in-network computing consumes resources required for native networking functionality, meaning that a switch can not act as both a networking-device and an accelerator at the same time. Last, and possibly most importantly, switches do not have the right computing resources to run certain applications.

Benefits of Compute Hierarchy. To overcome the computation limitations of switches, we adopt a hierarchical approach, similar to CPU architecture. In CPUs, memory hierarchy allows one to trade memory space for latency. This means that access to registers is immediate, access to first level cache has a few clock cycles latency, and every additional level in the hierarchy provides more memory, for the price of higher access time. Our approach trades computation for latency; computation within the programmable switch-ASIC is almost immediate but limited in capabilities, computation within an FPGA provides more opportunities, but takes more time and so on. Figure 1 shows the equivalences between the two approaches.

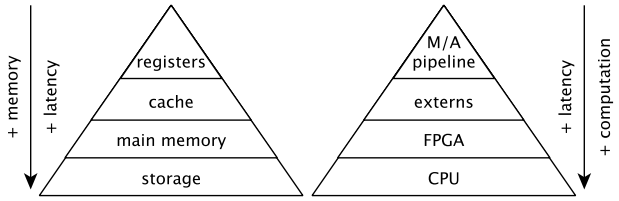

Figure 1: CPU hierarchy vs. network hierarchy.

In-network computing has been successfully adopted for caching use cases, but switches can cache only in the order of 64k [12]-128k entries, while applications such as memcached require millions to billions of unique keys [3]. Applications that require complex mathematical operations, ranging from multiplication to exponents and logarithms, are also divorced from switches. While look up tables can be used as an alternative to some operations [21], the resources required may be too extensive. For example, the multiplication of two 16bit operands will require a table of 2^{32} entries.

Using FPGAs as a second level of computation behind switches attends to many of these concerns. FPGAs have been extensively used as accelerators for scientific applications [4], meaning that their computations capabilities are not only more extensive than switches, but also more mature. Furthermore, FPGA's interfaces are configurable, allowing the FPGA to serve as a control and management layer between a switch and a memory or a storage device, providing access to additional resources. Moreover, many switch-boxes already include an FPGA, serving as a management device or, for example, for co-processing statistics.

Cost of Compute Hierarchy. There are, however, two prices to pay when using FPGA. First, the FPGA uses one (or more) of the switch ports. These ports achieve much higher data transfer rate than a PCI-express interface between the switch and a local CPU, but decrease the overall available port count. We argue that this is a better choice than dedicating switch I/O to memory or storage interfaces, as the number of pins required is smaller (per transfer rate), and the same switch-silicon flexibility supports a large number of configurations without design or manufacturing changes.

The second price, which can not be redeemed, is latency. While the latency through a switch is sub-microsecond, going through an FPGA and back, doubles or more the latency within a switch-box. A packet goes not only through the switch pipeline, but also to the FPGA and back, and potentially through part or all of the switch pipeline again, before going out to the user. We argue, however, that this trade-off of latency for computation is beneficial.

Performance. Applications are offloaded to the network only when there is a potential performance benefit. For example, if a caching application can serve 100× requests per second running within the network. While unlikely, it may be, in certain designs, that going through the FPGA will end up with the same request-reply latency as sending the request to a host. We assert, however, that if for the same overall latency, the throughput gain is 100×, then there is a benefit.

Network Latency. Having an FPGA attached to a switch provides the lowest bound on the latency of a network-attached accelerator. Using a remote smartNIC or FPGA means that any computation is further delayed by the latency of the connecting link or switches along the way. While this may be negligible in a ToR-to-NIC configuration, it is not if the processing ToR is located next to the user (e.g., edge termination), while the smartNIC is located next to the server, on the other side of the network. Furthermore, our scheme means that packets from the switch to the FPGA are not delayed by congestion within the network.

Network Load. Any computation sourced at a switch and offloaded to a remote FPGA or smartNIC increases the load on the network. Consider the case where a switch capable of processing ten billion requests a second has 1% cache miss ratio, and that this 1% is looked up remotely. That means a hundred million extra packets through the network every second, in each direction (query and result), potentially leading to congestion. Using an FPGA attached to the switch means that the network does not experience the extra load.

Power Efficiency. Tokusashi et al. [20] demonstrated that running an application on an FPGA is more power efficient than running on a host, and running on a switch is more power efficient than running on FPGA. Combining the switch and FPGA may increase the power consumption of a single switch box, but increases the overall end-to-end power efficiency of a data center, when considered holistically.

4 CHALLENGES AND OBSERVATIONS

Building a heterogeneous system with programmable network ASICs and FPGAs introduces a number of challenges. In this section, we discuss both primary challenges that must be addressed, and secondary challenges that would be nice to address.

4.1 Primary Challenges

Maintaining Performance. With 1-2 orders of magnitude gap in performance between switch-ASIC and FPGA, it must be guaranteed that switch throughput is not throttled by FPGA throughput. This means, for example, that a design where all incoming packets to the switch go through the FPGA is infeasible, even if there is sufficient I/O on both devices. Two potential mitigation techniques are i) compute on only a fraction of packets ii) send only a fraction of the data to the FPGA (e.g. 64B for every 8KB of data).

Blocking Architectures. Switch-ASICs achieve high performance through pipelining, moving the data all the time without stalling. In contrast, FPGA (often) achieve performance through parallelism, operating on multiple data-units at the same time, while each unit may be delayed. This conflict in approaches can lead both to performance loss and to congestion indications from the FPGA to the switch, which need to be avoided. An heterogeneous switch-FPGA architecture needs to be non-blocking, allowing the switch side not to stall on packets, while still allowing the FPGA to benefit from parallelism.

Encoding Information. It is not straight forward to send and receive packets between a switch and an FPGA. First, as it is desired to increase FPGA throughput by minimizing the processing required from it (e.g., header parsing) and as the interconnect between the switch and FPGA is a bottleneck. Second, as the switch needs to be able to distinguish between new incoming traffic and packets returning from the FPGA. Potential solutions include traffic encapsulation, bus adaptation and revised switch-traffic routing.

Multiple Switch Pipelines. Many switch-ASIC today use multiple pipelines to achieve target performance. This, however, creates a challenge connecting to FPGA. Each switch pipeline can connect independently, through a dedicated port, to the FPGA, thus maintaining the context of a transaction[1], but wasting ports. Alternatively, all pipelines can connect to the FPGA through a single egress port, assuming the required bandwidth is sufficient, but the FPGA needs to differentiate between source pipelines, and mark the reply for the right pipeline. None of these options is ideal.

4.2 Secondary Challenges

Matching Traffic Rates. Matching traffic rates is more than just ensuring that no performance is lost. The aim is to find the maximum amount of traffic that can be processed by the FPGA without leading to congestion. While in some cases this rate may be fixed, in others it may be workload dependent (e.g., with packet size distribution), and the switch needs to adapt to such dynamic conditions.

Merging Traffic. Consider a scenario where some packets are processed in the switch, and some in the FPGA. A combined design needs to ensure that the division of work is transparent to the user. This means, for example, that reordering of packets, where reordering is harmful, should not happen. This challenge becomes harder if there are also latency constraints, or if one tries to avoid buffering packets in the switch (e.g., due to memory limitations).

Processing Results. Processing computations' results by the switch is not trivial. First, as the switch may need to associate the results with previous requests. Second, as we would like to avoid processing the same packet twice. Not only to save resources and maintain throughput, but also to correctly process data and avoid running the same computation over and over again.

Design and Engineering. A heterogeneous architecture also brings engineering challenges, such as the need to integrate two parts of a program running on different types of devices. This means, for one, finding the optimal division of processing between switch and FPGA. It may also slow down the design cycles, as different languages and design tools are used by each component in the system.

5 FINGERPRINT APPLICATION AND PROTOTYPE IMPLEMENTATION

The previous section identified a number of design challenges for building a heterogeneous architecture. Choosing the best design

[1] I.e.,"which pipeline requested this computation?"

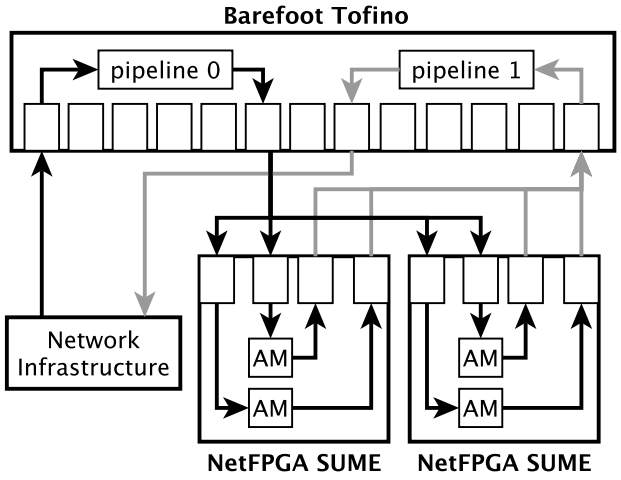

Figure 2: Prototype Implementation.

for navigating these challenges will depend on the specific requirements of the accelerated application. Below we describe a sample application, data deduplication, that benefits from a heterogeneous system. We also describe the concrete implementation motivated by the requirements of data deduplication.

Test Application: Data Deduplication. Data deduplication is used to reduce storage overhead by identifying and eliminating duplicate copies of data. A common approach for data deduplication, called "similarity-based deduplication" [2], consists of three phases. The first phase computes a hash, called the "fingerprint", for each block in the data stream. The fingerprints are then compared to a set of reference fingerprints in order to identify similarities between the new blocks and the ones already stored, through a process called "index searching". Finally, if a match is found, only a pointer to the reference block is stored. Otherwise, the new block is stored.

Data deduplication is typically computed at the storage node, just before the data is written to the storage devices. This approach, known as "target-based deduplication" is inefficient in two ways. First, it requires computing resources at the storage node to be reserved for data deduplication. Second, it increases the overall storage latency, by introducing an additional processing phase before accessing the storage medium. Leveraging network programmability, similarity-based deduplication can be improved by computing data deduplication while the data moves through the network, thus saving computing resources at the storage node and reducing the overall latency of the system.

Our proof-of-concept implementation only accelerates the first phase of similarity-based deduplication–the fingerprint computation. However, we believe that both the second and the third phases of similarity-based deduplication could be easily accelerated, without making substantial changes to the underlying prototype architecture.

Implementation. Figure 2 illustrates our prototype implementation, which uses a 32 ports Barefoot Networks' Tofino switch, and two Xilinx NetFPGA SUME cards. The network interfaces of the switch are configured for running at 40Gbps, by aggregating four 10Gbps network channels. Each FPGA is connected via two 10Gpbs interfaces to two switch pipelines. One pipeline processes packets before the FPGA, and one processes packets after the FPGA. Each FPGA includes two accelerator modules (AM), for a total of four. Each accelerator module is based on an implementation of a Rabin_16 hash function for computing fingerprints. The switch data plane logic is written in P4 [5].

Although out prototype is focused on the fingerprint computation, the design is flexible enough to accelerate generic computations. Both the network connections and the topology of the accelerators can be changed, based on the workload; the switch provides tens of network channels, that would allow us to connect many more accelerators of different types. The functionality implemented by each of the accelerator modules can be programmed independently through either hardware description languages, or high-level synthesis flows. Similarly, the pipelines of the switch can be reprogrammed in P4.

We note that each pipeline in the Tofino switch is independent from the others and data cannot be shared among pipelines. Due to this constraint, we were unable to share header data between input packets and output packets. Therefore, we decided to populate the headers of the output packets with some pre-defined, fixed data. A workaround for this limitation needs to be found to make routing flexible, until data sharing among pipelines will be available in programmable network switches.

6 EVALUATION

Fingerprint-based data deduplication is a good example of an application that benefits from global perspective that in-network acceleration provides. It also realizes an application that would be impossible to implement on programmable ASICs alone. Our evaluation quantifies the cost of this additional computing power, in terms of overhead of throughput, latency, resource utilization, and power consumption.

Experimental Environment. Our evaluation uses a Barefoot Network Tofino ASIC switch, which is connected to two NetFPGA SUME boards, as described in Section 5. To generate the workload, we used OSNT [1], running on top of a third NetFPGA board, hosted within a server, and generating test packets through four 10Gbps channels in parallel. To collect packets, the aggregated 40Gbps packet stream is returned to a NIC on the same server.

Latency. Each Rabin_16 module introduces a latency of 512 clock cycles. Considering that the clock frequency of the FPGA-based accelerators is 200MHz, the latency introduced by each Rabin_16 module is 2.56μs. Assuming that the latency contribution of the network interfaces is ~1μs (round-trip) and considering the submicrosecond latency added by the programmable switch, the total latency of the system is ~5.5μs.

Baseline Comparison. To provide context for our results, we refer to prior work by Li et al. [15], who report that the Rabin_16 cores achieve 5× performance improvement by running on FPGA rather than on a host. Our implementation uses sixteen of the same Rabin_16 cores, running on a newer FPGA.

Throughput. With traffic stimulus of 40Gbps, each Rabin_16 module is able to generate a new fingerprint every 2.56μs, or 390.6K

Resource	NetFPGA SUME	Xilinx VCU1525
Lookup Tables (LUTs)	16.5K (3.8%)	33K (2.7%)
Flip-Flops (FFs)	13.3K (1.5%)	26.7K (1.1%)
Block RAMs (BRAMs)	0 (0%)	0 (0%)

Table 1: FPGA Resource utilization on SUME and VCU1525 (one card, excluding the network interfaces).

fingerprints per second (Kfps). With four parallel Rabin_16 modules in each fingerprint module, the throughput is 1.56Mfps. In our evaluation, we used two NetFPGA SUME cards, each running two fingerprint modules in parallel, providing a total throughput of 6.25Mfps. The overall performance of the system is limited by the number of available 10Gbps ports on the FPGA platforms.

Resource Utilization. As shown in Table 1, the impact on FPGA resources is very limited. The fingerprint modules take less than 4% of the resources on each NetFPGA SUME card. If we include the network interfaces, the impact on the FPGA resources is less than 10% on average, thus leaving plenty of space for accelerating additional functionalities in the hardware. Only a fraction of the resources available on the switch is used in this implementation: each of the two pipelines employs a table and two actions. All the remaining resources are available for implementing more complex functionalities on the switch.

Power Consumption. The power consumed by each of the two FPGA-based accelerators is ~6.5W. Due to limitations in Barefoot's software, we were unable to measure the power consumption of the Tofino chip alone. But, the overall power consumption of the switch, measured at the plug, is ~100W.

Alternative FPGAs. We port our design to the Xilinx VCU1525 platform, based on Ultrascale+ FPGA and equipped with two 100Gbps ports. To match incoming data rates, we use eight Rabin_16 modules (×2 increase), a data path width of 1024b and clock frequency of 322MHz. This leads to latency decrease of 62.5% per Rabin_16 module, to 1.6μs, and to end-to-end system latency of ~4.5μs. Each module's throughput is increased by 60%, to 625Kfps. As shown in Table 1, the impact on FPGA resources is negligible, below 3%, thus leaving plenty of space for implementing additional functionalities.

7 DISCUSSION

Further Acceleration of Fingerprinting. In our prototype, the entire computation is done in the FPGA. However, fingerprinting can theoretically be further accelerated by breaking the process: running the index searching phase in the switch, and the fingerprint computation and encoding in the FPGA.

Bridging the Gap Between ASICs and FPGA. There are three important advantages to ASICs over FPGA: more high speed I/O, higher clock frequency and wider internal bus widths[2]. These lead in turn to the higher ASIC throughput. However, for computation purposes, the FPGA does not need to match those. It only needs to match the computation throughput requirements of the data arriving on a given port. With a tailored architecture, this is feasible on FPGA. The requirement thus becomes being able in the switch ASIC to partition and load balance computations across FPGA ports, which is an addressable requirement.

Limitations. Not all the challenges described in Section 4 are applicable for all applications. In this paper we did not attend to some of the challenges, including matching traffic rates, and design and engineering. We leave these challenges to future work, using different use cases. We also don't discuss management aspects of the system, such as master-slave relationship and bootstrapping. Our experience shows these will be system specific, while broader observations are beyond the scope of this paper.

[2]Limited by resources and timing constraints

8 RELATED WORK

Switch ASICs with FPGAs. Switch boxes that encompass switch-ASIC and FPGA have existed for a long time. In most cases, the FPGA is used for board control and management, and in some cases as a co-processor. The latter is used for offloading statistics collection, e.g., when the FPGA can connect to off-chip memory and help poll statics less often. However, these FPGA devices are used as termination point, and not as part of the processing pipeline.

Switch ASICs with other ASICs. Switch-ASIC also sometimes connects to other ASIC devices [6], used to extend on-chip tables. These devices are the evolution of off-chip TCAM, and do not offer the same level of flexibility as FPGA.

In-Network Computing with Switch ASICs. The last few years we have witnessed the rise of in-network computing usage for caching [12], query processing [14], consensus [8], storage [19], and many other applications. However, all these examples were limited by on-chip switch resources, or traded performance for functionality (e.g., through recirculation). Kim *et al.* [13] combined switch ASIC and remote memory on RDMA NIC as a means to overcome switch memory limitation. Our solution attends also to functionality limitations, and as there is no network traversal, offers the prospect of lower latency and lower network load.

In-Network Computing with FPGAs. Using FPGA as the main computing component for in-network computation has attracted further research interest [7, 10, 18], but an FPGA device performance is 1-2 orders of magnitude less than a switch ASIC [20].

FPGA-based Fingerprint Computation. Our work has been partially inspired by Li *et al.* [15], that provided FPGA-based fingerprint computation for high-throughput data storages. However, our contribution focuses on the concept of network computing hierarchy, and the challenges of integrating ASIC and FPGA.

9 CONCLUSION

We have presented a flexible and scalable network computing platform that extends programmable switches with FPGA-based accelerators. We have built a prototype architecture and validated it by accelerating fingerprint computation in the network. We believe our pioneering effort in making new programmable networks heterogeneous will pave the way to many future research projects in this field.

Acknowledgments. We acknowledge the support from the Swiss National Science Foundation (SNF) (project 407540_167173).

REFERENCES

[1] Gianni Antichi, Muhammad Shahbaz, Yilong Geng, Noa Zilberman, Adam Covington, Marc Bruyere, Nick Mckeown, Nick Feamster, Bob Felderman, Michaela Blott, Andrew W. Moore, and Philippe Owezarski. 2014. OSNT: open source network tester. *IEEE Network* 28, 5 (2014), 6–12.

[2] Lior Aronovich, Ron Asher, Eitan Bachmat, Haim Bitner, Michael Hirsch, and Shmuel T. Klein. 2009. The Design of a Similarity Based Deduplication System. In *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference (SYSTOR '09)*. Association for Computing Machinery, New York, NY, USA, Article 6, 14 pages. https://doi.org/10.1145/1534530.1534539

[3] Berk Atikoglu, Yuehai Xu, Eitan Frachtenberg, Song Jiang, and Mike Paleczny. 2012. Workload Analysis of a Large-scale Key-value Store. In *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '12)*. ACM, New York, NY, USA, 53–64. https://doi.org/10.1145/2254756.2254766

[4] Mariette Awad. 2009. FPGA supercomputing platforms: A survey. In *2009 International Conference on Field Programmable Logic and Applications*. 564–568.

[5] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. 2014. P4: Programming Protocol-Independent Packet Processors. *SIGCOMM Comput. Commun. Rev.* 44, 3 (July 2014), 87–95. https://doi.org/10.1145/2656877.2656890

[6] Broadcom. 2020. Knowledge-based processors. (2020). https://www.broadcom.com/products/embedded-and-networking-processors/knowledge-based/nla88650

[7] 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 *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 1–13.

[8] Huynh Tu Dang, Marco Canini, Fernando Pedone, and Robert Soulé. 2016. Paxos Made Switch-y. *SIGCOMM Comput. Commun. Rev.* 46, 2 (May 2016), 18–24. https://doi.org/10.1145/2935634.2935638

[9] Stephen Ibanez, Gordon Brebner, Nick McKeown, and Noa Zilberman. 2019. The P4->NetFPGA Workflow for Line-Rate Packet Processing. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '19)*. Association for Computing Machinery, New York, NY, USA, 1–9. https://doi.org/10.1145/3289602.3293924

[10] Zsolt István, David Sidler, Gustavo Alonso, and Marko Vukolic. 2016. Consensus in a Box: Inexpensive Coordination in Hardware. In *Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation (NSDI'16)*. USENIX Association, USA, 425–438.

[11] Theo Jepsen, Masoud Moshref, Antonio Carzaniga, Nate Foster, and Robert Soulé. 2018. Life in the Fast Lane: A Line-Rate Linear Road. In *Proceedings of the Symposium on SDN Research (SOSR '18)*. Association for Computing Machinery, New York, NY, USA, Article 10, 7 pages. https://doi.org/10.1145/3185467.3185494

[12] Xin Jin, Xiaozhou Li, Haoyu Zhang, Robert Soulé, Jeongkeun Lee, Nate Foster, Changhoon Kim, and Ion Stoica. 2017. NetCache: Balancing Key-Value Stores with Fast In-Network Caching. In *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP '17)*. Association for Computing Machinery, New York, NY, USA, 121–136. https://doi.org/10.1145/3132747.3132764

[13] Daehyeok Kim, Yibo Zhu, Changhoon Kim, Jeongkeun Lee, and Srinivasan Seshan. 2018. Generic External Memory for Switch Data Planes. In *Proceedings of the 17th ACM Workshop on Hot Topics in Networks (HotNets '18)*. Association for Computing Machinery, New York, NY, USA, 1–7. https://doi.org/10.1145/3286062.3286063

[14] Alberto Lerner, Rana Hussein, and Philippe Cudré-Mauroux. 2019. The Case for Network Accelerated Query Processing. In *CIDR*.

[15] Dongyang Li, Qing Yang, Qingbo Wang, Cyril Guyot, Ashwin Narasimha, Dejan Vucinic, and Zvonimir Bandic. 2015. A Parallel and Pipelined Architecture for Accelerating Fingerprint Computation in High Throughput Data Storages. In *2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines*. 203–206.

[16] Rui Miao, Hongyi Zeng, Changhoon Kim, Jeongkeun Lee, and Minlan Yu. 2017. SilkRoad: Making Stateful Layer-4 Load Balancing Fast and Cheap Using Switching ASICs. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '17)*. Association for Computing Machinery, New York, NY, USA, 15–28. https://doi.org/10.1145/3098822.3098824

[17] Barefoot Networks. 2018. Barefoot Tofino. (2018). https://barefootnetworks.com/products/brief-tofino/

[18] Muhsen Owaida, Gustavo Alonso, Laura Fogliarini, Anthony Hock-Koon, and Pierre-Etienne Melet. 2019. Lowering the Latency of Data Processing Pipelines through FPGA Based Hardware Acceleration. *Proc. VLDB Endow.* 13, 1 (Sept. 2019), 71–85. https://doi.org/10.14778/3357377.3357383

[19] Yi Qiao, Xiao Kong, Menghao Zhang, Yu Zhou, Mingwei Xu, and Jun Bi. 2020. Towards In-Network Acceleration of Erasure Coding. In *Proceedings of the Symposium on SDN Research (SOSR '20)*. Association for Computing Machinery, New York, NY, USA, 41–47. https://doi.org/10.1145/3373360.3380833

[20] Yuta Tokusashi, Huynh Tu Dang, Fernando Pedone, Robert Soulé, and Noa Zilberman. 2019. The Case For In-Network Computing On Demand. In *Proceedings of the Fourteenth EuroSys Conference 2019 (EuroSys '19)*. Association for Computing Machinery, New York, NY, USA, Article 21, 16 pages. https://doi.org/10.1145/3302424.3303979

[21] Zhaoqi Xiong and Noa Zilberman. 2019. Do Switches Dream of Machine Learning? Toward In-Network Classification. In *Proceedings of the 18th ACM Workshop on Hot Topics in Networks (HotNets '19)*. Association for Computing Machinery, New York, NY, USA, 25–33. https://doi.org/10.1145/3365609.3365864

[22] N. Zilberman, Y. Audzevich, G. A. Covington, and A. W. Moore. 2014. NetFPGA SUME: Toward 100 Gbps as Research Commodity. *IEEE Micro* 34, 5 (2014), 32–41.