applied system innovation

MDPI

Article

RETRACTED: Express Data Processing on FPGA: Network Interface Cards for Streamlined Software Inspection for Packet Processing

Sunkari Pradeep [1,*], Yogesh Kumar Sharma [2], Chaman Verma [3,*], Gutha Sreeram [4] and Panugati Hanumantha Rao [4]

1 Department of Computer Science and Engineering, Malla Reddy Engineering College for Women (UGC)–Autonomous Institution Maisammaguda, Secunderabad 500100, Telangana, India
2 Department of Computer Science and Engineering, Koneru Lakshmaiah Education Foundation, Vaddeswaram, Guntur 522302, Andhra Pradesh, India
3 Department of Media & Educational Informatics, Faculty of Informatics, Eotvos Lornad University, 1053 Budapest, Hungary
4 Department of Computer Science and Engineering, Vignana Bharathi Institute of Technology, Aushapur, Ghatkesar, Hyderabad 501301, Telangana, India
* Correspondence: pradeep.sunkari87@gmail.com (S.P.); chaman@inf.elte.hu (C.V.)

Abstract: Modern computers' network interface cards (NICs) are undergoing changes in order to handle greater data rates and assist with scaling problems caused by general-purpose CPU technology. The inclusion of programmable accelerators to the NIC's data channel is one of the ongoing improvements that is particularly intriguing since it gives the accelerator the chance to take on a portion of the CPU's network packet processing duties. Accelerators are frequently developed using platforms like field-programmable gate arrays because packet processing operations have severe latency requirements (FPGAs). When implementing packet processing activities, FPGAs' gain for through put is the number of data packets being successfully sent per second and latency is the actual time those packets take. However, due to their restricted resources, programming may need to be shared throughout a variety of applications. We provide hXDP, a software solution for FPGAs that targets the Linux eXpress Data Path and performs packet processing functions outlined with the eBPF technology. While maintaining performance on par with top-tier CPUs, hXDP only uses a tiny portion from the field programmable gate arrays, which are semiconductor devices that are based around a matrix of configuration logic blocks (CLB) connected over programmable interconnects. However, we demonstrate that when aiming towards a purpose-built FPGA architecture, many extended Berkeley packet filters (eBPF) allow programmers to use Berkeley packet filter byte code that makes use of certain kernel resources and instruction set architecture, to collocate and even eliminate, with considerably productivity and effectiveness. On an FPGA NIC, we implement hXDP and test its effectiveness using authentic eBPF programmes from the real world. Our version consumes 15% of the FPGA resources and operates at 156.25 MHz. This can constantly change and lead to the act of identification, inspection, extraction, and manipulation so that a network may make more intelligent management decisions.

Keywords: architecture; eBPF technology; FPGA; NIC

check for updates

Citation: Pradeep, S.; Sharma, Y.K.; Verma, C.; Sreeram, G.; Hanumantha Rao, P. RETRACTED: Express Data Processing on FPGA: Network Interface Cards for Streamlined Software Inspection for Packet Processing. *Appl. Syst. Innov.* **2023**, *6*, 9. https://doi.org/10.3390/asi6010009

Academic Editor: Eirini Eleni Tsiropoulou

Received: 4 November 2022
Revised: 31 December 2022
Accepted: 2 January 2023
Published: 9 January 2023
Retracted: 27 October 2023

1. Introduction

The hardware component used for processing data traffic arrivals, without which a computer cannot be connected over a network, consumes a significant portion of the CPU resources. A few examples of the duties carried out by these systems include enforcing security, for instance, the process of keeping our network firewalls secure by analysing key firewall monitoring parameters such as traffic, bandwidth, utilization, and sessions.

This paper has been retracted. A retraction notice was published on 27 October 2023 in *Applied System Innovation* **2023**, *6*, 101. https://doi.org/10.3390/asi6060101

Since a CPU performance cannot be further scaled and NIC port speeds are reaching over 100 Gigabit/s (Gbps), the best solutions for meeting the increasing energy are developed.

One of the possible methods for freeing the CPU from resource-intensive packet processing chores and reserving its valuable cycles for operations that cannot be performed elsewhere is the addition of programmable accelerators to the NIC [1]. The Flex network interface programming model can improve packet processing performance by reducing the memory system pressure at fast networks speeds [2,3]. As a result, different technologies, such as ASIC for a particular kind of transmission protocol, FGPA—based on a matrix of configurable logic blocks—and many more interconnected technologies are used to implement accelerators in the modern world.

Both a hierarchy of reconfigurable interconnects and a variety of programmable logic components are found in FPGAs [4]. The interconnects allow these blocks to be assembled in a variety of ways after they have been manufactured [5]. FPGAs offer a combination of programmability and performance when compared to other processors, as seen in Table 1.

Table 1. Performance description of processor.

S No.	Abbreviation	Processor	Description
1	ASIC s	Application Specific Integrated Circuits	The most efficient devices are those that are specifically designed, like Google's Tensor Processor Units (TPU). They cannot be changed to suit your changing demands.
2	GPU s	Graphics Processing Units	A common option for AI calculations. GPUs are faster at rendering images than CPUs because they can perform parallel processing.
3	CPU s	Central Processing Unit	CPUs are general-purpose processes but have subpar performance for processing graphics and video.
4	FPGA s	Field-Programmable Gate Arrays	The performance of FPGAs is comparable to that of ASICs, such as those offered on Azure. In order to include new logic, they are also adaptable and re-configurable throughout time.

With fewer FPGA resources and seamless integration with current operating systems, our approach is towards a universal and simple use of the method of programming by enabling the offloading of expensive packet processing. We achieve this using info, a group of technologies that makes it possible for the file system eXpress Data Path (XDP) [6,7] to be executed successfully on FPGA. The Linux community uses XDP extensively in production contexts because eBPF technology is used in the Linux kernel to provide secure programmable packet processing. The whole XDP feature of hXDP enables users to dynamically load content.

eBPF is a revolutionary technology with origins in the Linux kernel that can run sandboxed and was initially intended as an ordered list of something with an efficacious performance of operated devices where energy efficiency is key, such as in a register machine. The problem further proceeds for running XDP applications highly integrated in a field program [8]. In other words, extended Berkeley was made with high clock rates of the control unit in the server with a capacity of extended Berkeley instructions. As opposed to high-end CPUs, a field programme gate array has a widely parallel execution paradigm of 5 to 10 times lower with clock speeds [9].

Firstly, we take on the task by undertaking a thorough examination of the abstract model of a computer that defines the eBPF Instruction Set Architecture (ISA) and the current express data path programmes in order to identify as well as seize optimization opportunities. In the beginning, it lists the eBPF instructions that, when they are not being executed in the context of the Linux kernel, can be removed without risk.

Secondly, the extensions of the extended Berkeley instruction is going to introduce 3 mathematical operation architecture and there is also a program that stores a new parameterized program.

Finally, we influence instruction level uniformness in extended Berkeley for complier design techniques and processor executing numerous eBPF instructions concurrently by conducting a symbolic execution of the programmes at build time [10]. By minimizing pointless PCIe transfers, for example, we are able to fully utilise the on-NIC execution environment and integrate these efficiencies in the way we construct hXDP.

Here are the following features of our design:

A. A compiler for the enhanced hXDP ISA that converts the bytecode of XDP applications;
B. An independent is a block of logic or data that is used in making field programme gateway arrays or is application specific with various low-level optimizations;
C. The tool chain mechanism, which is dynamic at run time, loads a library and communicates for data path programmes, handles the acceleration tasks, and passes to NIC devices for additional processing.

This is an open source of Net FPGA logic blocks that implements a multiple level fan in gates, which gives it more impact on the design compared to an implementation [11]. The evaluation of our results shows real world applications of basic firewalls, which are going to load balancing with a re-engineered forwarding plane that takes advantage of kernel engineering as well as the XDP sample programmes provided by the Linux source code [12,13]. While offering a 10× lower forwarding latency, the Linux express data path will be equal, which refers to the capability of switching to forwarding packets in packet per second [14]. This is accomplished while using less than 15% of the FPGA resources and the frequency of prototypes with low clock speeds.

Conception

The main objective is making it possible to show data path applications on FPGA, which handles the acceleration tasks and passes packets to the NICs effectively, utilizing the least amount of FPGA hardware resources possible, as shown in Figure 1.

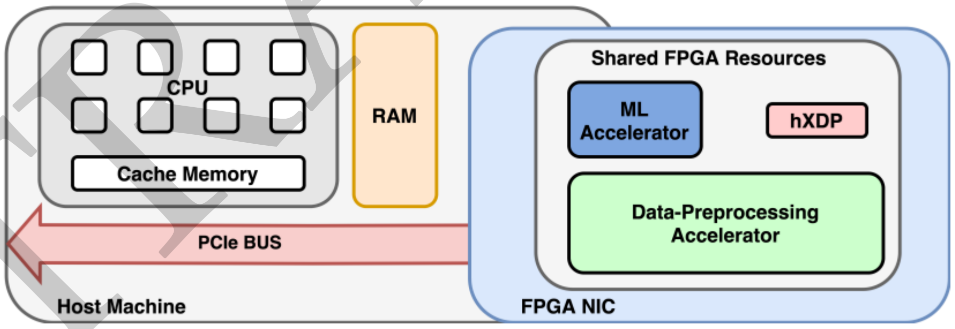


Figure 1. Specific accelerators of FPGA NIC resources and hXdp network accelerator.

As it allows for additional consolidation by fitting many application accelerators on the same FPGA, minimal utilization of the FPGA resources is particularly significant [15].

Instead, the decision to adopt XDP is driven by two advantages that the technology offers [16]. Because it offers an on-NIC optimization method that is widely known to a sizable community for Unix coders, programmers are spared from having to learn novel development tools like those offered by P4 or Flow Blaze It also makes NIC offloading for XDP programmes that have previously been deployed simple [17,18].

Requirements: From the discussion above, we list three high-level requirements for hXdp:

a. It must run compiled XDP programmes without modification and, therefore, must support the provided coverage foundations' tool chain, providing immense contribution to reloading and user-mode access to maps.
b. Performance for processing packets ought to be at least on par with top-tier processing packets.
c. It can be utilized only for the small portion of the hardware in the FPGA.

We will now provide a quick history of XDP before providing a more thorough explanation of the hXDP idea.

2. Materials and Methods

2.1. Xdp

By using express data path, inputs will be taken in code network interface, where they will be performed over the network packet and forwarded to the Linux network stack plan of action for XDP, which is built using eBPF technology from Linux [19]. A kernel virtualization provided by eBPF enables the trusted execution operation of code snippets within the core environment.

Extended Berkeley packet filter VM currently includes 64 b registers $\rightarrow$ 11

(a) r0 stores the return value from in-kernel programmes.
(b) r1, r2, r3, r4, r5 function as the parameters supplied to in-kernel functions.
(c) Holding address throughout control and arguments at r6, r7, r8, r9.
(d) Accessing the stack using the frame pointer, r10.
(e) The standard instruction set architecture that made over a hundred packed frame lengths is present in the eBPF virtual machine.

The kernel memory locations known as maps, or internal memory locations, which fundamentally re-arrange tables, are likewise accessible to eBPF programmes. For instance, arrays and hash tables can be implemented using maps in eBPF systems. A lookup on a map that has been set up as a hash table is one example of how an eBPF programme could interact with the locations on a map using an address reference for an unstructured information authorization or by looking over the communication of the function helper [20]. Since maps are the only way to maintain the programme state of the programme output for distributing information over eBPF programmes, they are particularly crucial.

2.2. Challenges

We will now take the sample of an XDP programme, which takes the inputs to a straightforward firewall to verify the formation of bi-directional TCP or UDP flows in order to gain instinctive knowledge over the structure complexity considered in implementing a data path on a field programme. It takes 71 eBPF instructions to generate a C programme that describes this straightforward firewall function.

We generate a rough estimate of the theoretical special case of this function for executing a field programme gate array based eBPF executor [21], if we assume that each eBPF instruction executes in a single clock cycle, that the clock cycles are not consumed for any additional processes, and that the field programme has a clock rate of 156 MHz, which is typical for a field programme gate array NIC.

2.3. hXDP Overview

By using a software–hardware co-design methodology, hXDP resolves the highlighted difficulty. Specifically, hXDP offers the necessary hardware module in addition to a compiler. The compiler utilises hXDP's hardware module features [22], which are provided to make it easier to take use of such chances, to take advantage of eBPF ISA optimization potential. By creating a new ISA that explicitly targets the execution of XDP programmes, we effectively extend the eBPF ISA.

The eBPF instruction level modifications performed by the compiler optimizations include the parallelization of instruction execution, the removal of superfluous instructions, and the replacement of existing instructions with newly designed more succinct ones [23,24]. The target hardware complexity is decreased since all optimizations are carried out at the time of compilation, shifting most of the complexity to the software compiler.

As a result, there is an exploit instruction level parallelism for performing and implementing thehXDP hardware module, which also implements a system for running up to four instructions simultaneously. No runtime programme optimization, such as instruction reordering or branch prediction, is offered by the VLIW soft processor [25].

In the end, the hXDP hardware component is implemented in the FPGA as a standalone module that is part of a larger device and system. The structure can be connected to additional processing modules, if necessary, and the NIC port as well as its PCIe controller addressing the core network can be connected to this by simply inserting a spike in the wire connecting them. Everything needed to use hXDP with the Linux operating system is provided by the hXDP software tool chain, which also contains the compiler. A compiled eBPF programme might, therefore, be executed in-kernel or on the FPGA alternately from the perspective of visualization, as shown in Figure 2.

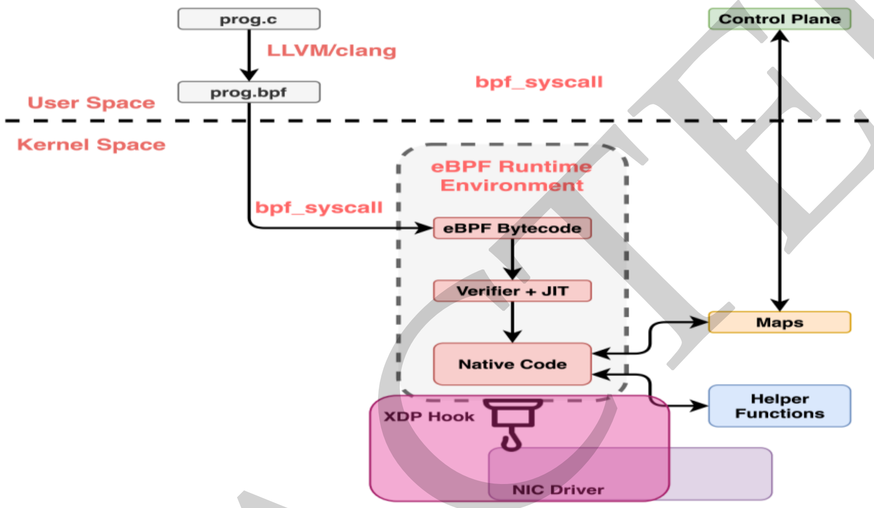
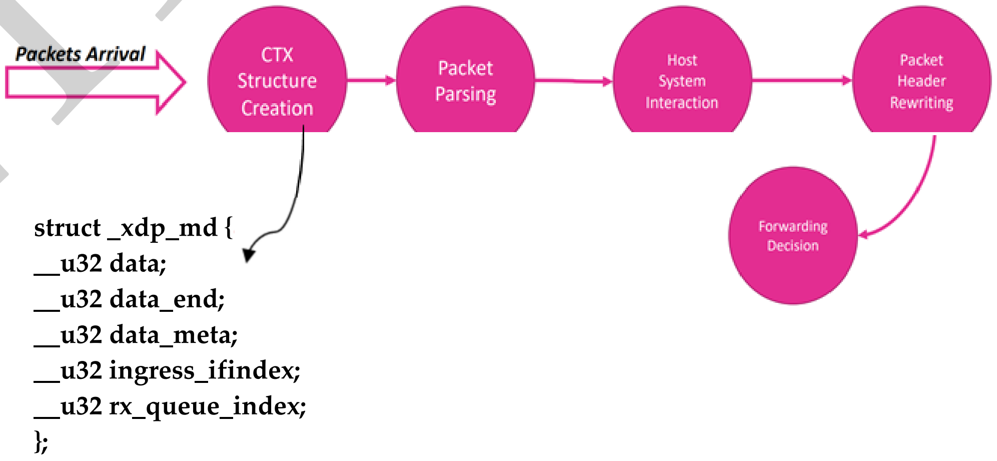


Figure 2. XDP workflow and architecture.

2.4. *hXdp Complier*

Programs must contain a count of additional inputs that are verified by the internal operating system in order for programmes using the eBPF technology to be able to execute within the Linux kernel [26]. When aiming for a specialized eBPF executor built on FPGA, a number of these operations could have been safely removed and returned with less expensive integrated hardware checks. Guidelines for memory boundary checks and memory zeroing are two pertinent examples, as shown in Figure 3.



```
struct _xdp_md {
__u32 data;
__u32 data_end;
__u32 data_meta;
__u32 ingress_ifindex;
__u32 rx_queue_index;
};
```

Figure 3. XDP program lifecycle.

2.5. *Hardware Module*

Figure 4 presents the following memory maps of hXdp hardware design:

(a) Input Queue (PIQ);
(b) eBPF runtime environment;

(c) Embedded packet capture (epc);
(d) Sephirot hardware and software;
(e) Helper functions.

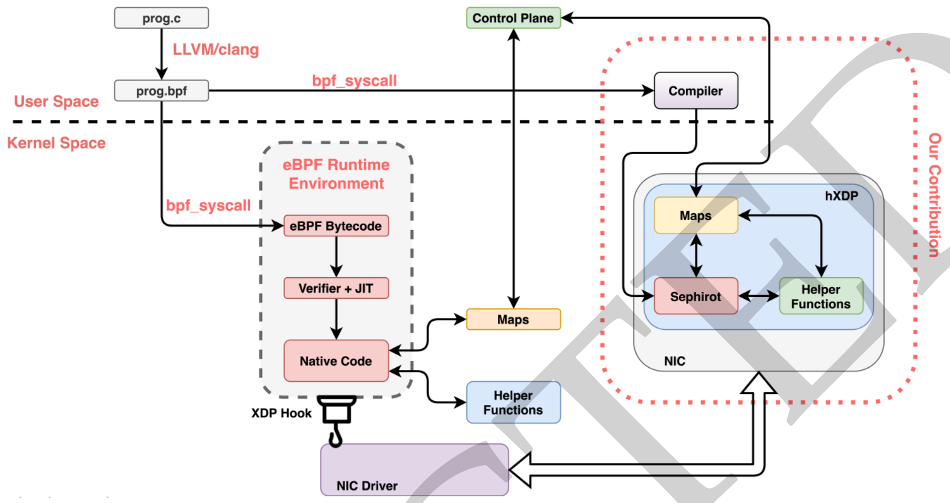


Figure 4. hXdp hardware design of logic architecture.

The four extended eBPF instructions that make up a VLIW instruction are read at processor start-up and their appropriate execution lanes are statically assigned [27]. The operations are downloaded in the background from the register file at this point, as four execution lanes work simultaneously to complete the final three pipeline steps. In case there is a need of eBPF instructions to load in memory, locations are pre-fetched during ID, and consuming the pre-defined values, the appropriate subunit is activated during IE. These include the control unit, memory unit (MU), and arithmetic and logical unit. Utilizing the hardware module characteristics of hXDP, which are designed to make the exploitation of such chances easier, the compiler takes advantage of eBPF ISA's better chance of receiving. In essence, we create a brand new ISA that enhances the eBPF ISA with a focus on running XDP programmes.

2.6. PIPE LINE

Self-resetting programme state: The variables that eBPF applications are going to use may be zeroed. We offer a programme initialization feature that automatically resets the stack and the registers. We can exclude any such zero-ing command from the programme thanks to this low-cost hardware feature that increases security.

Concurrent Branching: With architectures lacking branch prediction, speculative execution, and out-of-order execution, the inclusion of branch instructions may result in performance issues. This necessitates the serialization of the branch instructions for Sephirot. However, in XDP applications [28], particularly during header parsing, there are frequently a number of branches that are closely spaced apart. By putting on the priority channel, ordering a combination of hardware and software, we facilitated the parallel execution of these branches.

Processor Exit: When an exit instruction is carried out, the CPU terminates. The processor pipeline can be stopped in prior and the balance three clock cycles are saved, since outcome of inputs can be detected at the IF phase. This improvement enhances the performance advantage made possible by expanding the ISA to include configurable quit commands. In reality, before invoking an exit [29], XDP programmes typically shift the input to r0 to specify dynamic action. The Sephirot pipeline must always be traversed in order to set a value in a register [30] instead, since the value is already included in a freshly specified unconditional end exchange of information between a calling program and subroutine parameter passing eliminates the requirement for setting the input to r0.

The Linux data path (hXDP), an intellectual property core, utilizes 8.91% of the available programme gate array logic resources, and of 3.1% field programme gate array FPGAs, 2.3% address registers resources. APS and Sephirot require more resources in terms of logic because of their complexity compared to the other components. It is interesting to note that even more complicated helper functions, such as the one used to do a hash map lookup, only make a small contribution to the required logic, showing that to include them in the hardware design can be done for very little money while still providing good performance gains. The cumulative resource exploitation increases to 19.22%, 7.2%, and 14.22%.

3. Hardware Functionality

We contrasted hXDP with both the server-based version of XDP and the XDP programme loads directly on the NIC and execute without using CPU offered by a Netronome's sixth generation NFP-4000 multi-threaded Smart-NIC built on a SoC [31]. The 60 micro engines in the NFP 4000 run at an 800 MHz clock speed. Table 2 shows the processor utilization in server machines.

Table 2. Model description.

S No.	Type	Model	Processor
1	Intel	XL 710	2.1
2	Intel	I40 e	3.7
3	Xeon	X5670	3.6

We used a variety of CPU frequencies during the tests; we covered a wider range of deployment circumstances. In fact, a lot of deployments choose CPUs with lower frequency and more cores. We measured throughput and latency using a DPDK packet generator.

It was an easy way to run high performance packet processing programs without the hassle of kernel bypass technique. The design of the network flow processors used for intelligent flow processing in networks and communications of Netronome are all linked back-to-back to the packet generator, which can provide a 40-Gbps throughput with any packet size [32]. A traffic generator was used to put traffic onto a network for other machines to consume and also the round-trip time and perform delay measurements using hardware packet times tamping. A single network flow's 64B-sized packets were used for all testing, unless otherwise specified, in order to maintain consistency. The burden was difficult for the systems being tested.

4. Evaluation and Results

To carry out the hXDP corrections, we counted the Linux XDP example applications as well as two actual applications. Table 3 describes the Linux examples. The straightforward firewalls mentioned in the Katran server load balancer are the real-world applications.With the use of a weighted scheduling mechanism and per-flow consistency, in Katran, virtual addresses are transformed into real server addresses by a faster and with better load balancer. Katran collects a variety of flow data and also does IPinIP packet encapsulation.

Table 3. Evaluation of test in express data path.

Program	Description
xdp1	Parse pkt headers up to IP and XDP _DROP
Xdp2	Parse pkt headers up to IP and XDP _TX
Xdp_adjust_tail	Receive pkt, modify pkt into ICMP pkt and XDP _TX
Router _ipv4	Parse pkt headers up to IP, look up in routing table and forward (redirect)
Rxq_info(drop)	Increment counter and XDP _DROP
Rxq_info(tx)	Increment counter and XDP _TX
tx_ip_tunnel	Parse pkt up to L4, Encapsulate and XDP_TX
Redirect(map)	Output pkt from a specified interface (redirect)

We assessed how a compiler tries to minimally or maximally affect some attributes and the number of instructions in the programmes and the level of parallelism attained using these applications. As efficiency is computed over Net FPGA, implementation was then assessed. We also compared the performance of express data prototype with a Netronome-4000 Smart NIC architecture using micro benchmarks, as shown in Figure 5. Despite the fact that the two devices aim to address various deployment circumstances, this can shed further light on how the hXDP design decisions have affected the final product. We were unable to conduct a thorough review because the Netronome Flow processors-4000 only provides minimal extended Berkeley packet filter (eBPF) support. This section concludes with a summarized explanation of the output after we compared hXDP to alternative FPGA NIC programming methods.

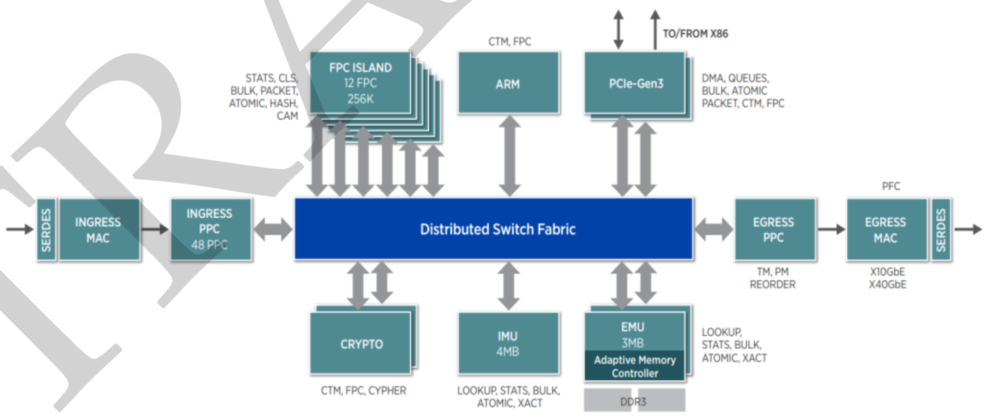

TO/FROM X86
CTM, FPC
STATS, CLS, BULK, PACKET, ATOMIC, HASH, CAM
FPC ISLAND 12 FPC 256K
ARM
PCIe-Gen3
DMA, QUEUES, BULK, ATOMIC PACKET, CTM, FPC
PFC
SERDES
INGRESS MAC
INGRESS PPC 48 PPC
Distributed Switch Fabric
EGRESS PPC
EGRESS MAC
SERDES
TM, PM REORDER
X10GbE X40GbE
CRYPTO
IMU 4MB
EMU 3MB
Adaptive Memory Controller
LOOKUP, STATS, BULK, ATOMIC, XACT
CTM, FPC, CYPHER
LOOKUP, STATS, BULK, ATOMIC, XACT
DDR3

Figure 5. Netronome 4000 Micro architecture.

4.1. Test Results

We report that the X86 instruction set that refers to the set of instructions that X-86 compatible microprocessor supports produced the extended Berkeley kernel programmability, which provides secure and high-performance kernel programmability from the operating system for compiler along with the decrease initiated by each optimization in the form of a stacked column, as shown in Figure 6. The benefit from parallelization to increase available computation power for fatter application processing and problem-solving gains attained by anticipating instructions from control comparable blocks is shown in the graph in Figure 7. We can see that our generator has the ability to produce a set of inputs that permits simpler, cheaper, high-performance implementations that are smaller than the set of instructions in the original programme. Observe how, in contrast, the JIT compiler for x86 typically produces an output that increases the number of instructions, as shown in Table 4.

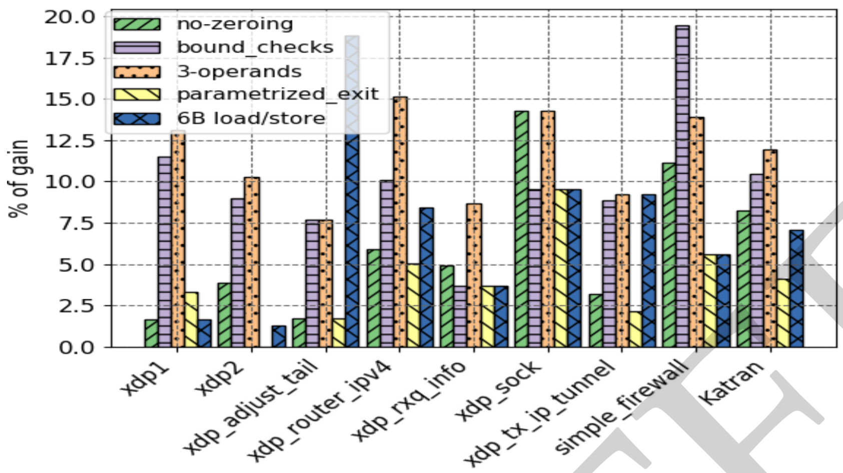


Figure 6. Impact of code optimization eBPF bytecode.

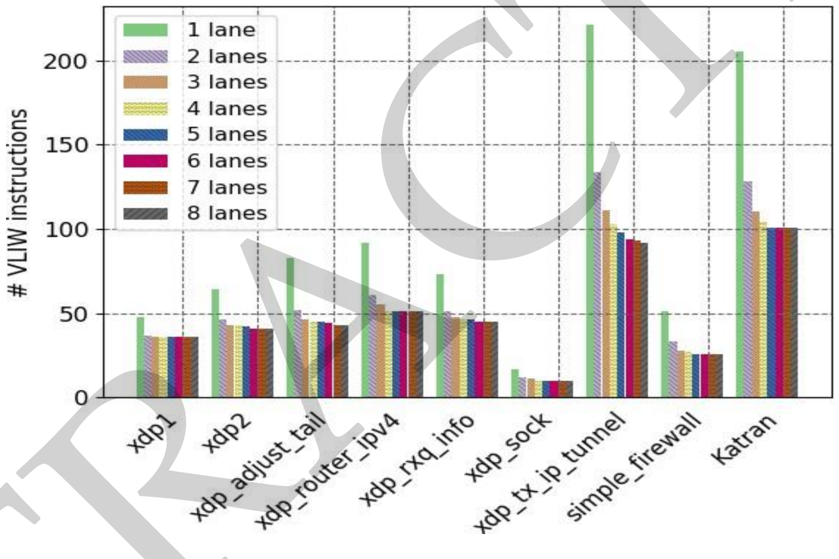


Figure 7. Very long instruction word inputs varying with no. of execution lanes.

Table 4. Instruction per cycle and IPC.

S No.	Program	Instruction	X86	hXdp
1	Xdp1	77	2.18	1.65
2	Xdp 2	60	2.22	1.69
3	Xdp tail	115	2.32	2.69
4	Ipv4	117	2.31	2.31
5	rxq info	80	2.74	1.75
6	Ip	281	2.34	2.76
7	Katran	267	2.31	2.61
8	Firewall	70	2.28	2.55

4.2. Application Performance

We stated that 2.8 Mpps show a limit in the mechanism of a straightforward firewall. The identical number of requests processed per minute per server instance achieves a throughput of 6.35 Mpps utilising a complier that tries to minimize or maximize some attributes of an executable program. For all packet sizes, hXDP offers a 10× reduction in processing delay (see Figure 8). Since hXDP does not cross the PCI or provide lower latency and higher data transfer rates than parallel buses such as PCI and PCI-X, this is the situation. The header and metadata are delivered to a flow processing, while the payload is forwarded to one of the memories in the displayed programmed express data path software, which solely conducts relaying of packets from one segment to another by nodes in the network. Despite the fact that it cannot be executed, the eBPF software offload solution provides a foundation for high performance, kernel complaint firewalls, DDoS protection, and load balancing. Even in this scenario, it is clear that hXDP offers a transmission capacity delay, particularly at smaller packet sizes. Figures 8–13 show the performance of express data path and network flow processing for base line, performance, latency, and speed.

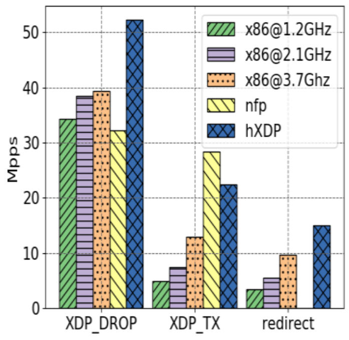


Figure 8. Baseline output measurements for basic X-data path.

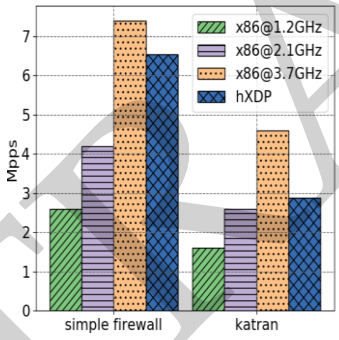


Figure 9. Comparable performance to an x86@2.1GHz for programs.

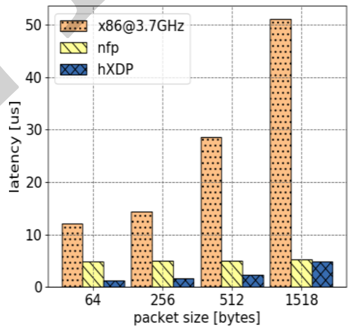


Figure 10. Performance evaluation latency measurements.

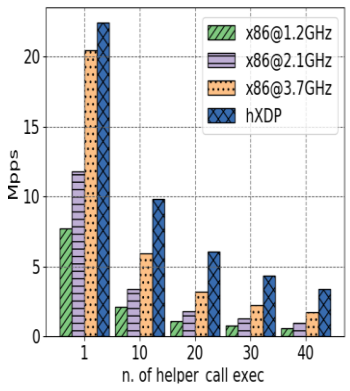


Figure 11. Forwarding output when calling a helper function.

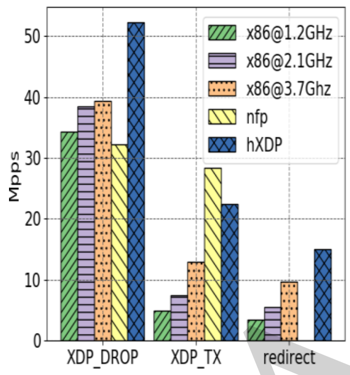


Figure 12. Baseline output when calling a helper function.

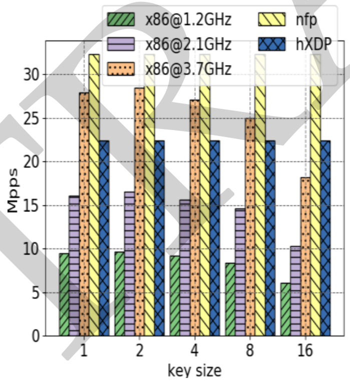


Figure 13. Impact on forwarding output on map accesses.

5. Conclusions

The express data system, which enables Linux's data path programmes to execute on field programme gate arrays with NIC, was designed and implemented in this work. Field programme express data can execute unmodified data path XDP applications with performance comparable to a top-tier system, over 2.2 GHz x86 CPU core. While we think the performance results for a design running at 156 MHz are already amazing, we also highlighted several areas for further improvements. Designing and implementing an express data path required considerable research and engineering efforts that included the creation of a processor and its compiler. In fact, this can be considered as the foundation express data path and application level to develop future connections among computer systems, programs, and network adapters. We provide access to various solutions for the scientific community to encourage work throughout connectivity.

Author Contributions: Authors contributed equally to the development of this work. Conceptualization, Formal Analysis, Data Curation, and Validation—S.P.; Methodology, Writing Original Draft, Validation—S.P., Y.K.S., C.V., G.S. and P.H.R.; Software, Writing-Review-Editing—S.P., Y.K.S., C.V., G.S. and P.H.R.; Supervision—Y.K.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: We would like to thank Y Madhavee Latha–Principal of Malla Reddy Engineering College for Women (UGC-Autonomous Institute), Hyderabad-500014, Telangana, India for supporting and enhancing the work which is committed for further. We would like to thank you the reviewers for careful verification and approval of our manuscript. The work of Chaman was supported by the Faculty of informatics, Eötvös Loránd University, Budapest, Hungary.

Conflicts of Interest: The Authors have declared no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

hXdp	Linux Express Data Path
XDP	Express Data path
FPGA	Filed programmed gateway array
NIC	Network Interface Cards
VERY LONG INSTRUCTION WORD	Very Long Instruction Word
Ebpf	Extended Berkeley Packet Filter

References

1. P4-NetFPGA. Available online: https://github.com/NetFPGA/P4-NetFPGA-public/wiki14-oct-21 (accessed on 25 December 2022).
2. Bernstein, A.J. Analysis of programs for parallel processing. *IEEE Trans. Electron. Comput.* **1966**, *15*, 757–763. [CrossRef]
3. Bosshart, P.; Daly, D.; Gibb, G.; Izzard, M.; McKeown, N.; Rexford, J.; Schlesinger, C.; Talayco, D.; Vahdat, A.; Varghese, G.; et al. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Comput. Commun. Rev.* **2014**, *44*, 87–95. [CrossRef]
4. Bosshart, P.; Gibb, G.; Kim, H.-S.; Varghese, G.; McKeown, N.; Izzard, M.; Mujica, F.; Horowitz, M. Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn. In Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM, SIGCOMM '13, Hong Kong, China, 12–16 August 2013; Association for Computing Machinery: New York, NY, USA, 2013; pp. 99–110.
5. Brunella, M.S.; Pontarelli, S.; Bonola, M.; Bianchi, G. V-PMP: A VERY LONG INSTRUCTION WORD packet manipulator processor. In Proceedings of the 2018 European Conference on Networks and Communications (EuCNC), Ljubljiana, Slovenia, 18–21 June 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 1–9.
6. Caulfield, A.M.; Chung, E.S.; Putnam, A.; Angepat, H.; Fowers, J.; Haselman, M.; Heil, S.; Humphrey, M.; Kaur, P.; Kim, J.; et al. A cloud-scale acceleration architecture. In Proceedings of the 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), Taipei, Taiwan, 15–19 October 2016; pp. 1–13.
7. Chen, T.; Moreau, T.; Jiang, Z.; Zheng, L.; Yan, E.; Shen, H.; Cowan, M.; Wang, L.; Hu, Y.; Ceze, L.; et al. TVM: An automated end-to-end optimizing compiler for deep learning. In Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18), Carlsbad, CA, USA, 8–10 October 2018; USENIX Association: Berkeley, CA, USA, 2018; pp. 578–594.
8. Chiou, D. The microsoft catapult project. In Proceedings of the 2017 IEEE International Symposium on Workload Characterization (IISWC), Seattle, WA, USA, 1–3 October 2017; p. 124.
9. Dumitru, M.V.; Dumitrescu, D.; Raiciu, C. Can we exploit buggy p4 programs? In Proceedings of the Symposium on SDN Research, SOSR '20, San Jose, CA, USA, 3 March 2020; Association for Computing Machinery: New York, NY, USA, 2020; pp. 62–68.
10. Facebook. Katran Source Code Repository. 2018. Available online: https://github.com/facebookincubator/katran (accessed on 25 December 2022).
11. Firestone, D.; Putnam, A.; Mundkur, S.; Chiou, D.; Dabagh, A.; Andrewartha, M.; Angepat, H.; Bhanu, V.; Caulfield, A.; Chung, E.; et al. Azure accelerated networking: Smartnics in the public cloud. In Proceedings of the 15th USENIX Symposium on

Networked Systems Design and Implementation (NSDI 18), Renton, WA, USA, 9–11 April 2018; USENIX Association: Berkeley, CA, USA, 2018; pp. 51–66.

12. FlowBlaze. Repository with FlowBlaze Source Code and Additional Material. Available online: http://axbryd.com/FlowBlaze.html (accessed on 25 December 2022).
13. Forencich, A.; Snoeren, A.C.; Porter, G.; Papen, G. Corundum: An open-source 100-Gbps NIC. In Proceedings of the 28th IEEE International Symposium on Field-Programmable Custom Computing Machines, Boulder, CO, USA, 29 April–1 May 2020.
14. Gautschi, M.; Schiavone, P.D.; Traber, A.; Loi, I.; Pullini, A.; Rossi, D.; Flamand, E.; Gürkaynak, F.K.; Benini, L. Near-threshold risc-v core with dsp extensions for scalable iot endpoint devices. *IEEE Trans. Very Large Scale Integr. VLSI Syst* **2017**, *25*, 2700–2713. [CrossRef]
15. Hazelwood, K.; Bird, S.; Brooks, D.; Chintala, S.; Diril, U.; Dzhulgakov, D.; Fawzy, M.; Jia, B.; Jia, Y.; Kalro, A.; et al. Applied machine learning at Facebook: A datacenter infrastructure perspective. In Proceedings of the High Performance Computer Architecture (HPCA), Vienna, Austria, 24–28 February 2018; IEEE: Piscataway, NJ, USA, 2018.
16. Heinz, C.; Lavan, Y.; Hofmann, J.; Koch, A. A catalog and in-hardware evaluation of open-source drop-in compatible risc-v softcore processors. In Proceedings of the 2019 International Conference on ReConFigurable Computing and FPGAs (ReConFig), Cancun, Mexico, 9–11 December 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 1–8.
17. Hennessy, J.L.; Patterson, D.A. A new golden age for computer architecture. *Commun. ACM* **2019**, *62*, 48–60. [CrossRef]
18. Hohlfeld, O.; Krude, J.; Reelfs, J.H.; Rüth, J.; Wehrle, K. Demystifying the performance of XDP BPF. In Proceedings of the 2019 IEEE Conference on Network Softwarization (NetSoft), Paris, France, 24–28 June 2019; pp. 208–212.
19. Høiland-Jørgensen, T.; Brouer, J.D.; Borkmann, D.; Fastabend, J.; Herbert, T.; Ahern, D.; Miller, D. The express data path: Fast programmable packet processing in the operating system kernel. In Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies, CoNEXT '18, Heraklion, Greece, 4–7 December 2018; Association for Computing Machinery: New York, NY, USA, 2018; pp. 54–66.
20. Intel. 5G Wireless. 2020. Available online: https://www.intel.com/content/www/us/en/communications/products/programmable/applications/baseband.html (accessed on 25 December 2022).
21. Iseli, C.; Sanchez, E. Spyder: A reconfigurable Very Long Instruction Word processor using FPGAs. In Proceedings of the IEEE Workshop on FPGAs for Custom Computing Machines, Napa, CA, USA, 5–7 April 1993; IEEE: Piscataway, NJ, USA, 1993; pp. 17–24.
22. Jones, A.K.; Hoare, R.; Kusic, D.; Fazekas, J.; Foster, J. An fpga-based Very Long Instruction Word processor with cust om hardware execution. In Proceedings of the 2005 ACM/SIGDA 13th International Symposium on Field-Programmable Gate Arrays, FPGA '05, Monterey, CA, USA, 20–22 February 2005; Association for Computing Machinery: New York, NY, USA, 2005; pp. 107–117.
23. Kaufmann, A.; Peter, S.; Anderson, T.; Krishnamurthy, A. Flexnic: Rethinking network DMA. In Proceedings of the 15th Workshop on Hot Topics in Operating Systems (HotOS XV), Kartause Ittingen, Switzerland, 18–20 May 2015; USENIX Association: Berkeley, CA, USA, 2015.
24. Kicinski, J.; Viljoen, N. eBPF hardware offload to SmartNICs: Cls bpf and XDP. *Proc. Netdev* **2016**, *1*, 1–6.
25. Michel, O.; Bifulco, R.; Rétvári, G.; Schmid, S. The programmable data plane: Abstractions, architectures, algorithms, and applications. *ACM Comput. Surv.* **2021**, *54*, 1–36. [CrossRef]
26. NEC Building an Open vRAN Ecosystem White Paper. 2020. Available online: https://www.nec.com/en/global/solutions/5g/index.html (accessed on 25 December 2022).
27. Netronome. AgilioTM CX 2x40GbE Intelligent Server Adapter. Available online: https://www.netronome.com/media/redactor_files/PB_Agilio_CX_2x40GbE.pdf (accessed on 25 December 2022).
28. Pontarelli, S.; Bifulco, R.; Bonola, M.; Cascone, C.; Spaziani, M.; Bruschi, V.; Sanvito, D.; Siracusano, G.; Capone, A.; Honda, M.; et al. Flowblaze: Stateful packet processing in hardware. In Proceedings of the 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19), Boston, MA, USA, 26–28 February 2019; USENIX Association: Berkeley, CA, USA, 2019; pp. 531–548.
29. Sultana, N.; Galea, S.; Greaves, D.; Wojcik, M.; Shipton, J.; Clegg, R.; Mai, L.; Bressana, P.; Soulé, R.; Mortier, R.; et al. Emu: Rapid prototyping of networking services. In Proceedings of the 2017 USENIX Annual Technical Conference (USENIX ATC 17), Santa Clara, CA, USA, 12–14 July 2017; USENIX Association: Berkeley, CA, USA, 2017; pp. 459–471.
30. Wang, H.; Soulé, R.; Dang, H.T.; Lee, K.S.; Shrivastav, V.; Foster, N.; Weatherspoon, H. P4fpga: A rapid prototyping framework for p4. In Proceedings of the Symposium on SDN Research, SOSR '17, Santa Clara, CA, USA, 3–4 April 2017; Association for Computing Machinery: New York, NY, USA, 2017; pp. 122–135.
31. Xilinx. 5G Wireless Solutions Powered by Xilinx. 2020. Available online: https://www.xilinx.com/applications/megatrends/5g.html (accessed on 25 December 2022).
32. Zilberman, N.; Audzevich, Y.; Covington, G.A.; Moore, A.W. NetFPGA SUME: Toward 100 Gbps as Research Commodity. *IEEE Micro* **2014**, *34*, 32–41. [CrossRef]