xPUBench: Scalable and Energy-Efficient GPU and DPU-Accelerated Network Functions

Maxime Vanliefde[1][0009-0001-5506-0743](✉), Romain Van Hauwaert[1], Nikita Tyunyayev[1][0000-0002-9838-9645], Clément Delzotti[1][0009-0009-0538-5556], Elena Agostini[2], and Tom Barbette[1][0000-0003-1269-2190]

[1] ICTEAM, UCLouvain, Belgium
maxime.vanliefde@uclouvain.be
nikita.tyunyayev@uclouvain.be
clement.delzotti@uclouvain.be
tom.barbette@uclouvain.be
[2] NVIDIA, USA
eagostini@nvidia.com

Abstract. The rapid increase in network speeds makes packet processing on general-purpose CPUs increasingly challenging. At 100 Gbps and beyond, CPUs struggle to sustain complex network functions without dedicated acceleration. This trend motivates the exploration and measurement of alternative compute platforms such as GPUs and embedded CPUs in Network Interface Cards (NICs). Modern NICs provide tighter integration with GPUs, with the ability to write received packets directly to GPU memory. SmartNICs, also known as Data Processing Units (DPUs), further feature embedded ARM or RISC-V cores capable of offloading NFV packet processing entirely.

In this work, we introduce xPUBench, a benchmarking environment that systematically measures the performance and energy efficiency of packet processing across CPUs, GPUs, and DPUs. We evaluate several (co-)processing models relevant to Network Function Virtualization, including CPU+GPU hybrid, DPU-only, and GPU-only approaches. Our measurements show that, for a computation-heavy workload, current CPU-only implementations manage to handle up to 50% of the 100 Gbps NIC rate. In contrast, GPU implementations can saturate it. We also show that the DPUs' most powerful embedded cores can replace the main CPU for some traditional packet processing, alleviating the load on the host, which can now be entirely dedicated to running applications. We finally propose a novel energy-efficiency dimension, showing that DPUs outperform traditional CPUs for low-throughput processing, requiring only 24 W to sustain 10 Gbps, and that GPUs outperform CPUs for high-throughput processing. Our findings emphasize the need to assess both performance and energy in heterogeneous packet-processing pipelines, given the growing diversity of "xPUs" in networked systems.

Keywords: NFV · GPU · DPU · DPA · Energy Efficiency · Packet Processing

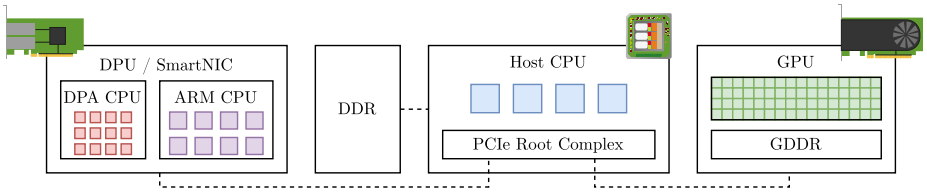


Fig. 1: Modern server architecture with multiple heterogeneous processing units

1 Introduction

With enterprises increasingly moving their infrastructure to the cloud, the emergence of high-bandwidth connectivity for end users, such as 5G and fiber-to-the-home, increases heavily the demand for high-speed network packet processing. Network Interface Cards (NICs) speeds have risen dramatically in the last decades. Interfaces offering up to 100 Gbps are now a commodity in datacenters [59], while cards with speeds of 400 Gbps can today be found on the consumer market [53]. Data centers also constantly improve their hardware to improve response times and reduce congestion [61].

Network functions such as firewalls, Network Address Translation (NAT), AI-enhanced malware detection, WAN optimizers, and even Radio Access Networks (RANs) have recently followed the approach of Network Function Virtualization (NFV), implemented primarily on software for flexibility and the ability to be offloaded on generic cloud machines [2, 7, 47].

With such high speeds, it can be difficult to keep up with the rate at which packets are received and processed on flexible and programmable platforms like Central Processing Units (CPUs). At 100 Gbps, processing a 64 B packet should happen as fast as 6.72 ns (i.e., 430 CPU cycles for a 16-cores 4 GHz CPU). Sustaining such a packet rate for even basic network functions already consumes much of today's CPU. Executing complex network functions at 100 Gbps is even beyond the capacity of current CPUs, as will be shown in Section 4.3.

To keep up the pace, multiple previous works have been conducted to lighten the load on CPUs, by porting network packet processing to other hardware units like Graphics Processing Units (GPUs) and Data Processing Units (DPUs) as highlighted in Figure 1.

GPUs can be used to offload some of the processing from the CPU. Their many-core parallel architecture and high-bandwidth memory make them well suited for parallel computations on numerous packets. The literature has mostly focused on GPU co-processing as GPUs were not able to manage the NIC on their own until recently [28, 54]. PacketShader [22] was the first to use discrete GPUs to accelerate network processing; APUNet [19] reviewed the findings using integrated GPUs; GPU-Ether [28] finally allows running GPU-only applications by using GPU threads to perform the packet I/O, completely bypassing the CPU.

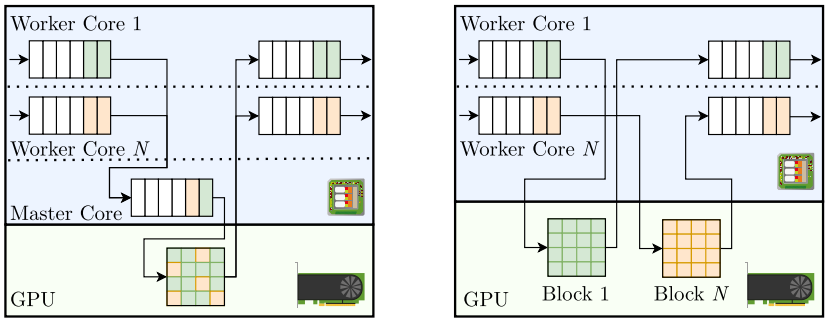
DPUs, also called SmartNICs, designate NICs with some on-NIC CPU cores. They have also been proposed as candidates to process traffic outside the CPU, following recent interest from the community [17,24,42,45]. They are becoming a part of today's datacenters and increase the computational power available on a single machine [26,31]. While it is common for SmartNICs to embed ARM cores running a full operating system, a recent trend also pushes for the integration of weaker but simpler RISC-V cores, such as the Data-Path Accelerator (DPA) on NVIDIA ConnectX-8 and BlueField-3 ASIC [56] or other FPGA-based architectures [12,23,33,58]. In this work, we consider DPUs general-purpose cores to offload network functions from the host. This allows for processing packets even before they reach the host, thereby reducing the load on the CPU and freeing up CPU cycles for other tasks. This model is relevant for datacenter operators, implementing network interposition in the SmartNIC they control, freeing up CPU cores for their tenants. In addition to complete offloading, DPUs enable a hybrid approach, offloading only some parts of the chain, more suitable for the less flexible NIC ASIC [42,43].

Nowadays, the interconnections between the NIC, the CPU and the GPU are much faster than in most of these previous works. In about 10 years, NIC speeds have multiplied by 40 [55] and PCIe speeds by 4 [57]. GPU memory bandwidth has doubled and is still much faster than both. With such increased speeds, GPUs and DPUs are poised to become more useful than ever for network processing.

Given these order-of-magnitude parameter changes, we ask the following question: *Are flexible (as in "easy to program") platforms like GPUs and CPU-based DPUs candidates to process packets at a rate of 100 Gbps?*

In this work, we present **xPUBench**, a benchmarking framework designed to evaluate and compare CPU-, GPU-, and DPU-based packet processing approaches in terms of throughput and energy efficiency. xPUBench is the first tool offering an environment that reproduces and measures the performance of a plethora of previously proposed (co-)processing models and of various devices, with a widely available baseline of 100 Gbps and similar applications. Based on a literature review on GPU network packet processing, xPUBench also integrates a novel processing model to enable harnessing the full potential of a GPU, resulting in a new, more scalable hybrid CPU-GPU framework. xPUBench is finally the first to propose the comparison of these (co-)processing models regarding energy efficiency. Notably, it explores how tuning the core and uncore CPU frequency and the core count is critical, even when the CPU is bypassed entirely. We open source xPUBench, so that the community can benefit from simple-to-use implementations and build upon our work.

Using xPUBench, we demonstrate that GPU-based methods provide higher throughput than CPU-only methods, particularly for heavy computational workloads; however, the optimal GPU processing model depends on the application to be run. The GPU-only approach (without CPU involvement) is the highest-performing implementation (up to 6.3× the throughput of the CPU-only approach and 2.7× that of the hybrid CPU-GPU approach). We also show that



(a) Single-Master – Multiple-Workers, as in PacketShader [22] and APUNet [19]

(b) Parallel Workers, as in Snap [69]

Fig. 2: CPU-GPU processing models

DPU-based methods are able to sustain the line rate for basic network functions, while providing lower latency compared to a CPU-only implementation, and that they are the most energy efficient, drawing up to 3× less power than a CPU-only equivalent approach.

The rest of this paper is organized as follows. Section 2 presents the background on GPU and DPU (co-)processing models and related works. Section 3 provides the design and implementation of xPUBench. Section 4 measures and discusses the performance and energy consumption of a cross-combination of the "xPUs" and processing models. Finally, Section 5 summarizes the main findings of this work.

2 Background

In the following, we review existing models for packet processing with GPU, first the larger body of work on CPU-GPU co-processing, and then GPU-only models. We then review the DPU literature and, finally, the energy efficiency aspects of NFV on xPUs. We first present processing models and then summarize the state of the art.

2.1 CPU-GPU (Co-)Processing

Processing Models for CPU-GPU Co-Processing. All hybrid implementations use shared queues to communicate between CPU and GPU. These queues reside in CPU memory, hold pointers to packets' payloads, and can be accessed by both the CPU and the GPU. Upon reception, packets are enqueued as batches.

Master-Worker Architecture. As shown in Figure 2a, in the master-worker architecture proposed in PacketShader [22] and APUNet [19], workers are responsible for the packet I/O, while the master is the only thread communicating with the

GPU. The master and workers are each associated with one CPU thread running on its own CPU core. In this model, GPU utilization is independent of the number of CPU cores used. Using more cores only helps to spread the load of received packets using Receive Side Scaling (RSS), but every packet will, in the end, transit through the same master core.

Parallel Workers Architecture. The parallel model, shown in Figure 2b, supports multiprocessing by completely duplicating the processing pipeline per CPU core, as proposed by Snap [69] and NBA [34]. Each pipeline is processed by a different GPU block (group of GPU threads) with a one-to-one matching between a GPU thread and a packet in a batch. Multiple CPU cores are used to receive packets, and each packet is processed on the core that received it.

Memory Models for CPU-GPU Co-Processing. In a naïve approach, packets can be received on CPU, then copied to GPU memory (Figure 3 A). This approach is simple to implement but can waste a lot of PCIe bandwidth, especially when only a portion of the packet is needed for processing.

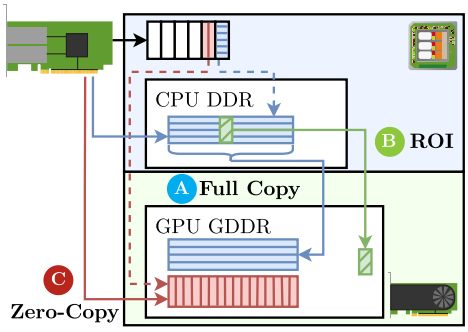


Fig. 3: GPUs can access packets' payloads using 3 different methods. A solid arrow indicates a data copy, and a dashed arrow indicates data access.

Region of Interest. A number of Virtual Network Functions (VNFs) only use a portion of the packet to process it. In particular, many only need a protocol header. For example, IP lookups only need the destination IP address, and Ethernet mirroring only needs the source and destination MAC addresses. Some previous work [22, 69] proposed to only send the data that is actually needed to the GPU, as shown in Figure 3 B. The contiguous byte range of the packet that is used is called a Region of Interest (ROI). GPUs read from global memory by section of 32 B [50]. It is beneficial to coalesce Regions of Interest (ROIs) of packets if the ROI is smaller than the size of a single read. This way, multiple GPU threads can be served with a single GPU memory access. For example, for an IPv4 lookup whose ROI size is 4 B, one memory transaction would serve $32/4 = 8$ threads all in one.

Zero-Copy (Z-C). If packets are to be entirely processed on GPU, NICs can directly write the payloads to GPU memory, without passing them through CPU memory, as shown in Figure 3 C. The host still receives the packets' metadata (including results of offloaded classification) to decide which application should be scheduled on the GPU. APUNet [19] alleviates the problem using the unified memory of APUs (combined CPU-GPU chips) to share memory between CPU and GPU. We do not consider APUs in this study as they are not common in servers. GASPP [71] calls zero-copy the idea that the GPU directly reads CPU

Table 1: Comparison of Related GPU Packet Processing Works.

	MW*	Parallel ROI†	Z-C‡ (CPU)	Z-C‡ (GPU)	GPU-only	Max. Speed§	Available	Last Maintained
PacketShader	✓	✗	✓	✗	✗	✗	4*dual-10 Gbps	✗	Not Available
Snap	✗	✓	✓	✗	✗	✗	4*10 Gbps	✓	2014
APUNet	✓	✗	✗	~	~	✗	dual-40 Gbps	✗	Not Available
NBA	✗	✓	✗	✗	✗	✗	dual-40 Gbps	✗	Not Available
GASPP	✗	✓	✗	✓	✗	✗	2*dual-10 Gbps	✗	Not Available
GPU-Ether	N/A	N/A	N/A	N/A	✓	✓	10 Gbps	✗	Not Available
xPUBench	✓	✓	✓	✓	✓	✓	1*100 Gbps	✓	2025

*Single Master, Multiple Workers. †Region of Interest. ‡Zero-Copy.
§Maximum speed achieved by the original work.

memory, which is different from the zero-copy considered by our work, as well as by Romein, J. W. [64], where the packets are directly written to GPU memory by the NIC. Using CPU memory, the packet is actually copied *once more* from CPU to GPU memory, meaning it crosses the PCIe bus twice. To avoid ambiguity, we explicitly specify either Zero-Copy on CPU or on GPU.

Kernel Models for CPU-GPU Co-Processing. When using the Zero-Copy memory model, we can also take advantage of persistent GPU thread execution, as shown in APUNet [19]. The use of explicit transfers between the CPU and GPU prohibits this approach, as we first initiate the copy request and then the kernel launch for each batch on a stream. Such a kernel defines the function executed in parallel by each GPU thread. Persistent GPU threads can reduce the latency of processing for lightweight applications whose processing is shorter than the kernel launch.

GPU-Only Bypassing CPU. GPU-Ether [28] introduces a framework designed to enable the development of GPU-only applications. As shown in Figure 4, incoming packets are directly received in GPU memory, and GPU threads manage network traffic. It does not use any CPU core after initialization. This technique is now standard to NVIDIA NICs, commercially known as GPUNetIO [54].

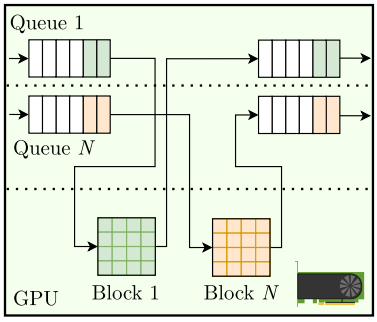


Fig. 4: GPU-Only, as in GPU-Ether [28]

Summary of Models in the Literature. Previous work explored different combinations of those models to separate CPU processing from GPU processing in GPU-accelerated frameworks, as shown in Table 1. While all these works present working solutions, speeds delivered nowadays by network cards call for a revision of the processing models with a common baseline.

Batching. Batching is a technique whereby multiple packets are processed at once. Multiple packets are received from the NIC and processed as a batch on the CPU. Similarly, the GPU and CPU communicate packets in batches. Previous work has shown that batching is essential to achieve high performance in packet processing. On the I/O side, batching reduces the overhead of accessing the NIC [5,36,63]. On the GPU side, using batches makes better use of the parallelism offered by the GPU [34, 69]. At the same time, batching increases average latency. Finding the optimal batching size thus involves a trade-off and depends on multiple factors, such as packet size, network bandwidth, and GPU processing power. Previous proposals [36, 71] support a dynamic batch size, whereby the batch is processed after a certain amount of time even if it is not full.

2.2 Other GPU-Related Work

Some work focuses on specific packet processing algorithms tailored to GPU, such as for IP lookup [38,68], NDN lookup [74], Pattern Matching [72] or DPI [66], on alleviating the problem of divergence due to different packet sizes on GPU [41], or on algorithms for scheduling multiple NFV chains on a GPU [8]. GPUnet [67] proposes orthogonal work to ease the GPU abstraction for network programming, connecting the GPU through RDMA but without focusing on NFV workloads. These works are beyond the scope of this paper, which focuses mainly on communication patterns and not the acceleration of the GPU itself.

G-Opt [29] focuses on accelerating CPU-based packet processing inspired by GPU parallelism. Similar recent advances in high-speed NFV CPU processing [15, 18, 46] are also beyond the scope of communication patterns.

NBA [34] studies the load-balancing of packets between the CPU and the GPU for packet processing. It does not attempt to process batches larger than 64 packets, even when they are sent to the GPU for processing. 64 packets is not enough to achieve high-speed, as will be shown in Section 4.4. FlowShader [76] evaluates the balance of flows across CPU and GPU according to flow size, while G-NET [77] focuses on virtualizing the GPU and scheduling functions between multiple tenants. These works are orthogonal, as we focus on modeling performance for each system.

Romein, J. W. [64] processes a large amount of data from telescopes using a high-end NVIDIA integrated platform (NVIDIA Grace Hopper). In this work, we stick to commodity platforms and propose a systematic review of multiple use cases, across multiple approaches beyond the design of a single use-case. They also use a feature of modern NICs to split packets, sending a certain number of bytes to the CPU, and the rest to the GPU. We do not evaluate this possibility in this work, but it enables use cases where zero-copy can be used for most of the payload while the CPU handles the header of the packet.

2.3 DPU

Other works have explored the use of DPUs to free CPU cores from packet processing. DPUs can intercept network traffic and either process it independently,

fully offloading tasks from the CPU, or preprocess packet traffic before forwarding it to the host. As GPUs, they can therefore be used as candidates for our study on flexible (co-)processors for NFV processing.

DPUs are heavily heterogeneous computing platforms. They embed accelerators tailored for network processing, like crypto engines or RegEx acceleration [52]. Lognic [20] accounts for this heterogeneity by modeling the performance of an offloaded program on the SmartNIC as a graph of computing units interconnected by a fabric. By specifying the performance of the accelerators and the fabric, as well as a traffic profile, Lognic can predict the latency and throughput of an application.

DPUs also feature rich programmable pipelines capable of high-rate packet processing (e.g., DPDK's `rte_flow` [13]). However, these pipelines are typically limited in flexibility [30]. In addition, as both DPUs and hosts share access to this common pipeline, we disregard it as a distinguishing factor. Instead, our analysis focuses on the performance of the general-purpose CPU cores within the DPU.

As opposed to packet processing on GPUs, we note that the body of work on packet processing on DPUs reflects the heterogeneity of the DPUs themselves. The techniques used vary from DPU to DPU, making it impractical to rerun all the ideas from the literature. Nonetheless, in this work, we propose a performance comparison of 3 different types of CPU cores present in 2 generations of NVIDIA BlueField cards: the BlueField-2 [52] and the BlueField-3 [53].

In addition to general-purpose CPU cores running a traditional operating system, the BlueField-3 [53] is equipped with a Data-Path Accelerator (DPA) [56]. It consists of 16 RISC-V cores, allowing up to 256 hardware threads. Those cores run a custom operating system that does not support DPDK. However, the DPA includes several architectural adaptations tailored for packet processing. Notably, its proximity to the ASIC can reduce latency. Chen *et al.* [9] benchmarked the BlueField-3 DPA and showed that its cores are considerably less powerful than the BlueField ARM cores and host CPU cores. In this work, in addition to the performance aspect addressed by their work, we benchmark those processors for energy efficiency.

2.4 Energy Efficiency

Previous work has looked at techniques to reduce the energy footprint of network functions on the CPU. They explore different power reduction mechanisms, such as Dynamic Voltage Frequency Scaling (DVFS) [39], sleeping time [14], core count reduction [60] and interrupts [48]. Because of the NFVs heavy reliance on packet polling, any effort for frequency autoscaling such as using *performance governors* [25] is annihilated by polling instructions filling CPUs to their maximal capacity.

Metronome [14] successfully reduced power consumption by implementing a *sleep&wake* method to reduce the impact of NIC polling. *ixcp* [60] progressively increases the core count then the frequency to scale the system's capacity to the

workload. TUPE [39] reduces core frequency and inserts *napping* times following heuristics on CPU utilization. Natori et al. [48] takes advantage of a new CPU governor to control the C-states of a CPU core, allowing to go into deeper and energy-efficient C-states while ensuring short wake up delays upon packets arrival. Their work also takes advantage of hardware interrupts to lengthen the duration of stay in such deeper states. Additionally, other works [64] use DVFS directly on a GPU combined with frequency and core count reduction on the CPU to reduce power consumption. However, taking advantage of applying DVFS to the GPU requires a significant workload to fill GPUs' high capacity. Even at 100 Gbps, a GPU can remain at its idle-level energy consumption.

These solutions all induce an increase in latency, as packets are either not processed immediately when they arrive or because the CPU is slowed down. Sloth [11] allows reducing this latency increase by relying on previous observations of the workload. From these observations, Sloth identifies the least energy consuming cores-frequency combination which can sustain any specific latency.

Reducing power consumption is also possible by tuning the *uncore frequency*. Such terms, coined by Intel, designate the frequency at which every controller outside cores (e.g., integrated memory controllers, on-ship peripheral controllers, etc.) are operating. While the uncore frequency is scaled automatically by the hardware, it tends to be aggressive [10] and leads to unneeded overconsumption. Furthermore, uncore frequency scaling remains a frequent blind spot within the literature.

While prior research has explored the energy efficiency of individual architectures, like DPUs [44, 73], CPUs [11, 14, 39, 48, 60], and GPUs [64], to the best of our knowledge, we present the first systematic and comprehensive comparison of these xPUs, evaluating their trade-offs across both performance and energy efficiency under a unified framework. Particularly, we expose the impact of the uncore even when the CPU is bypassed.

3 Implementation

In this section, we present the design and implementation of xPUBench and the processing models it supports. Since most of the state-of-the-art methods are unavailable or unmaintained, as shown in Table 1, we cannot easily rerun them. Therefore, we implement these ideas from scratch to review them on a similar baseline.

We implement state-of-the-art communications patterns and our proposals for CPU-based methods (including CPU-GPU co-processing) in FastClick [5], an extended version of the Click Modular Router [35], when applicable. We considered other NFV-oriented processing frameworks such as BESS [21] or VPP [16], but FastClick provides most of the intrinsic features for high-speed, including kernel bypassing with DPDK [70], I/O batching and ARM support, comes with a richer set of features and has been the subject of recent academic works to improve its performance [15, 18, 49]. As the Click framework is not suitable for CPU-less platforms, we later detail the GPU-only approach, based on an

NVIDIA sample, as well as the RISC-V implementation for the BlueField DPA, built on top of the samples provided by Chen *et al.* [9].

3.1 Automated Workload Generation and Measurements

xPUBench automates measurements using NPF [3]. NPF is responsible for orchestrating experiments, including defining the experimental design, deploying it, and collecting the data.

We integrate all the aforementioned processing models, as well as workload generation and energy measurements, into xPUBench. We contributed around 1600 lines of code (LoC) for the scripts used by NPF.

xPUBench and all implementations are available at `https://github.com/UCLouvain-ENSG/xPUBench`.

3.2 CPU-GPU FastClick Pipelines

In our proposals, we stick to FastClick's best practices for multithreading [5] and follow a run-to-completion multithreaded processing path. In such a model, each packet is processed by the CPU core that received it, from reception to transmission. We added or modified 3662 LoC in FastClick. We built new FastClick template elements to implement a full-copy, ROI or zero-copy approach. These templates also hide the communication with the GPU to ease the development. Internally, we base the communication on the DPDK [70] GPU library. We also enable packet payloads to be received directly on GPU memory or on CPU memory mapped to the GPU. As in previous work, GPU elements themselves are rewritten from scratch in CUDA. Automatically compiling an existing CPU code to GPU, for instance, P4 to GPU [37], is orthogonal to our study.

The GPU workload is performed in parallel on all packets of a batch. One packet is processed with one GPU thread. When the processing of all packets from the batch is done, a shared flag is set. During GPU processing, the CPU alternates between processing packets from the NIC and polling the GPU completion flag, allowing packets to be enqueued while batches are being processed. When the GPU has processed the batch, the CPU retrieves it and continues processing in the pipeline. As in Batchy [36], we can configure both a minimal batch size and a maximum waiting delay for packets in the pipeline, but we prefer to show the tradeoff in this work and do not set any waiting time limit in our study.

3.3 Decoupling CPU and GPU Scalability

With the combination of processing models, memory models, and kernel models discussed in Section 2.1, we can effectively compare all previous work shown in Table 1. We further notice that if the application is computationally or memory intensive, new packets can arrive and be enqueued to the GPU in less time than it takes to process a single batch on the GPU.

We propose a new, more scalable and resource-efficient CPU-GPU model that extends the parallel workers model. In this approach, GPU utilization can be tuned in two ways, as shown in Figure 5: either by adjusting the number of CPU cores used to receive packets, or by increasing the number of queues a CPU core uses to communicate with the GPU. The latter increases GPU utilization while keeping the same CPU utilization, decoupling I/O scaling from computation scaling.

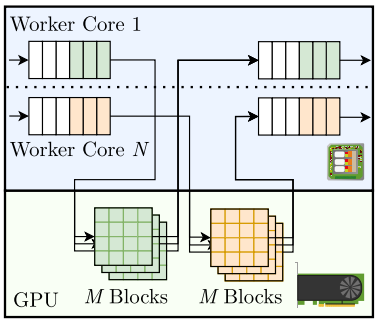


Fig. 5: Decoupled Parallel, our proposed processing model.

Batches from a queue are processed sequentially, but multiple queues are processed simultaneously, thanks to GPU *multi-stream* capabilities [50]. A stream defines an in-order sequence of operations. While all operations associated with a stream are executed in sequence, multiple streams can run on the same device simultaneously, allowing for parallel GPU kernel execution. The CPU fills the queues in a round-robin fashion. The GPU, however, processes the current batch of packets in all queues in parallel, using one stream per shared queue.

3.4 GPU-Only

The DOCA framework from NVIDIA provides GPUNetIO [54], a CPU-bypass implementation similar to GPU-Ether [28] working with NVIDIA NICs [52, 53, 55]. As the GPU-only version is, of course, only running on GPU, it is a different code than FastClick based on an NVIDIA sample [54]. We contributed around 1364 LoC to implement the various applications.

The application is launched from the host CPU. Once it is started, all packet payloads and descriptors are received in GPU memory. Some GPU threads perform the packets' I/O and some others perform workloads. No CPU core is thus used when the traffic is being handled. This reduces latency, as there is no need to issue transactions to pass packets' info from the CPU to the GPU. Receive and send queues residing in GPU memory are used to transfer packets from the NIC to the GPU and the other way around. Each queue has a dedicated GPU block to process its packets. Packets are received per batch and packets of a batch are processed in parallel by up to 1024 packets [50]. Increasing the number of queues thus both spreads the load between multiple queues and increases GPU utilization. This work is the first to evaluate DOCA GPUNetIO performance at high speed.

As the NIC has to directly access GPU memory, the PCIe BAR of the GPU must be resizable, which limits the platforms that can run this implementation, as it is a BIOS-specific option.

3.5 DPU-Only

ARM cores. We slightly modified FastClick to enable compilation on both BlueField-2 (BF2) [52] and BlueField-3 (BF3) [53] ARM cores. This allows to compare the performance of the CPU with that of the onboard ARM processor because they both run the same FastClick pipeline. The BlueField-2 is equipped with up to 8 ARMv8 A72 cores and 16 GB of DDR4 RAM. The BlueField-3 is equipped with up to 16 ARMv8.2+ A78 Hercules cores (64-bit) and 32 GB of DDR5 RAM.

DPA cores. We build our applications on top of the datapath proposed by Chen *et al.* [9]. As recommended in their work, we use buffers allocated in ARM memory as opposed to DPA memory to receive packets, which enables us to sustain the line rate for an Ethernet mirroring application. The DPA can then access the packets using MMIO. Compared to DPDK which is used on the ARM cores, this datapath is particularly efficient. It leverages a shared RX-TX ring, eliminating the need for pointer copies between separate rings. In addition, its memory management is simpler: assuming run-to-completion execution, it can safely reuse the same buffers without relying on a memory pool. Following the approach of Chen *et al.* [9], we balance traffic across DPA threads using the destination MAC address. This approach is necessary due to the lack of documentation on using RSS with the DPA. Out of the 256 available threads, we were able to utilize 254. The DPA implementation adds around 300 LoC to the Chen *et al.* sample.

3.6 Limitations

In this work, we focus on simple stateless functions, an approach that facilitates similar implementations across different device architectures. Exploring stateful application performance would require complex, device-specific data structures and is left for future work. Particularly, DPU cores often exhibit architectural differences from traditional cores, such as smaller and slower caches in the case of the DPA [9]. We expect these differences to result in greater performance gaps in more complex, stateful workloads. Finally, we exclude all specialized accelerators present on the DPU, such as pattern-matching or compression engine [52], which are heavily task-specific.

4 Evaluation

In this section, we use xPUBench to evaluate the performance of all implementations. We first independently evaluate multiple VNFs processed on the CPU using standard FastClick, on the GPU using its various processing models, and on the DPUs. We also evaluate how important different factors are for each platform. We finally explore the energy efficiency of the various approaches.

4.1 Testbed

The server under test has an AMD EPYC 9124 16-core Processor at 3 GHz and 128 GB of RAM. An NVIDIA L4 GPU and an NVIDIA Mellanox ConnectX-6 100GbE NIC are connected on the same NUMA node, both on a PCIe 4.0 x16 slot. The second CPU on another NUMA node is only used for control. Simultaneous multithreading is disabled. The server runs the Linux kernel 5.15.0-113-generic. DPDK 23.11.0 is used in conjunction with CUDA 12.4, NVIDIA driver version 550.54.15 and DOCA 2.8.0082.

The BlueField-2 and BlueField-3 are running in standalone, externally powered. We stress the fact that for this study they can run in standalone, as packets do not ever reach the host CPU.

Packets are generated from another server, using Pktgen-DPDK [75] 23.06.1 with DPDK 23.03.0. The generator server is equipped with an Intel Xeon E-2378 octa-core Processor at 2.60 GHz, with 32 GB of RAM, and a single NUMA node. The NIC is an Intel Ethernet Controller E810-CAM1/2 100GbE, connected to the machine on a PCIe 4.0 x16 slot. The server runs the Linux kernel 5.15.0-113-generic.

Every run is repeated 3 times, and the standard deviation is always shown on the graph. When invisible, it should be assumed to be very low. When showing latency versus throughput, each point is only run once, as the variability is visible through the many measurements.

4.2 Workloads

We evaluate three workloads that stress different parts of the systems under review.

Ethernet Mirroring (labeled MAC) swaps the frame's source and destination Media Access Control (MAC) addresses. It is a baseline example application with a light workload, as only 12 B are read and modified for each received packet, regardless of its size. Furthermore, no memory lookup is required apart from accessing the packet.

Cyclic Redundancy Check computation (labeled CRC) computes a checksum on the full packet. It is an interesting proxy use case because it processes the whole packet. Therefore, the amount of data processed depends on the size of the frame.

IP Lookup (labeled IP1K or IP10K) looks up the destination IP address of the packet in a routing table. We used a linear search for its simplicity and ease of implementation. We perform IPv4 lookup exclusively. In this work, we do not look at faster implementation for CPU [65] or GPU [40] but simply consider it as a workload highly dependent on the routing table size. The table size therefore acts as a factor of complexity, similar in all implementations. Only 4 B of the IPv4 address is needed for each packet. The routing table used comes from the RIPE RIS database [62]. We obtained the data from collector 01 on 8 April 2024. We then take the top-N entries to form a routing table of size N, with $N = 1000$ in the IP1K cases and $N = 10000$ in the IP10K cases.

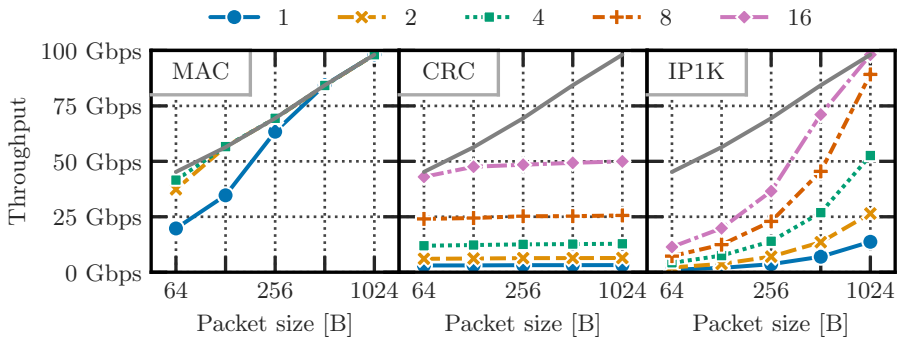


Fig. 6: Performance of the CPU implementation acc. to the number of CPU cores. Gray lines indicate the generator rate.

4.3 CPU Scalability

We first evaluate the scalability of the CPU implementation, where increasing the number of cores enables to run the VNF in parallel. We use RSS [27] to spread the load. Techniques such as RSS++ [4] could be used to ensure an even more balanced load between cores, but our synthetic workload ensures enough entropy for RSS to work efficiently.

The throughput of the CPU implementation as the number of CPU cores increases is shown in Figure 6. Four CPU cores are needed to handle Ethernet Mirroring on all packet sizes, but with large packets only one core is enough, as there are fewer packets received per second. For CRC computation, whose complexity is a function of the size of the packets, 16 CPU cores saturate the generator rate only for packets of 64 B. Recall that CPU cores handle both receiving and sending packets, as well as performing the workload: if it is heavy, there are fewer CPU cycles available to receive packets, leading to greater losses that are mitigated by spreading the load across more CPU cores. For IP lookup, independently of the packet size, increasing the number of cores by a factor of N leads to an increase of $\sim N$ times in the throughput, as our VNF under test does not share state between flows and can almost linearly scale.

⊙ **Takeaway 1.** CPUs struggle for even a relatively simple task such as CRC computation.

4.4 CPU-GPU Scalability

We then evaluate processing models in a CPU-GPU hybrid approach.

CPU Cores. We evaluate how performance scales in regard to the number of CPU cores used. The performance of the MW processing model, as well as the ROI, Z-C on CPU and Z-C on GPU memory models for the parallel processing

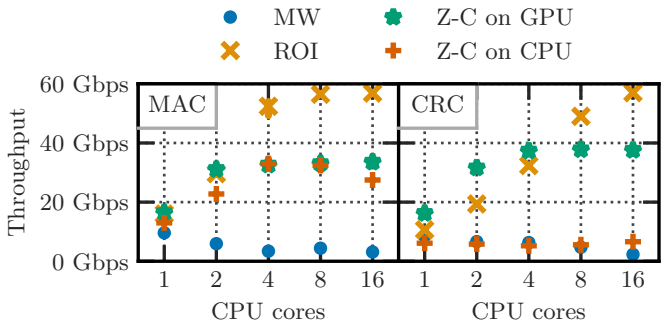


Fig. 7: Performance based on processing and memory model, with one CPU-GPU communication queue per core. Packet size is 128 B.

model, are shown in Figure 7. We do not evaluate separately full packet copy, as the ROI for the CRC actually showcases a complete packet copy.

We notice the extremely poor performance of the MW processing model. Indeed, it saturates around 10 Gbps, and does not scale with the number of cores receiving packets. This threading strategy [19, 22] is no longer viable at today's network speeds, as only one CPU core handles all communication with the GPU. Even with larger packets, the model fails to scale.

⊙ **Takeaway 2.** The Master-Workers processing model (PacketShader, APUNet) does not scale to modern speeds.

On the other hand, both the ROI and Zero-Copy using parallel workers scale with the number of cores, as they increase GPU utilization. We see that the Z-C on GPU plateaus after 2 cores for the Ethernet Mirroring and 4 cores for the CRC. We believe the bottleneck to be the number of PCIe transactions for the NIC-GPU interface. The ROI implementation continues to scale as it sends the bytes with a single DMA transaction per batch. Z-C on CPU matches Z-C on GPU for Ethernet mirroring, where only 16 bytes need to be written from and to CPU memory by the GPU. For the CRC workload where the whole packet needs to be fetched from the CPU memory by the GPU, Z-C on CPU offers poor performance. We measure the PCIe bandwidth with AMD uPerf and observe the Z-C on CPU leads to a 4× PCIe bandwidth increase. We believe it is because the GPU accesses the CPU transparently, always in cacheline-sized bursts. From now on, we only consider Z-C on GPU as it always provides better or equal performance.

⊙ **Takeaway 3.** ROI and full packet copy can still reach 100 Gbps, even with small packets.

⊙ **Takeaway 4.** While Z-C on GPU decreases the CPU PCIe utilization and slightly improves the per-core performance with regard to ROI when the full packet is needed, sending packet payloads from NIC to GPU leads to many

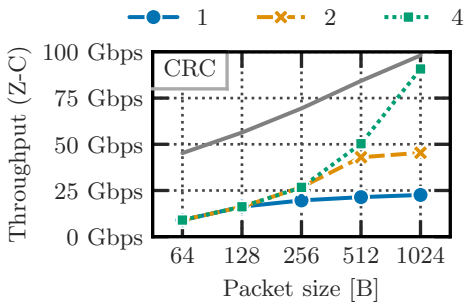


Fig. 8: Performance of the Z-C on GPU memory model on the CRC workload with a single CPU core, acc. to the number of CPU-GPU shared queues.

PCIe transactions, leading to a plateau for small packets.

⊙ **Takeaway 5.** Accessing transparently the CPU memory from the GPU (as GASPP for large packets) is much less efficient than directly sending packets payloads to GPU memory.

GPU Decoupled Parallelism. We then evaluate the performance of our new decoupled parallel model presented in Section 3.3. By balancing the load across CPU-GPU queues, we reduce the number of CPU cores needed. In the ROI model, increasing the number of queues does not lead to any improvement. As the implementation uses explicit copies, using multiple queues will lead to interleaved copies that cannot overlap, as the GPU only has a limited number of copy engines (2 in the L4 GPU we use) [50]. For cases that are more computationally intensive, more queues enable an increase of parallelism on the GPU, and thus a better throughput at constant CPU cores, as can be seen in Figure 8 for the CRC use case.

⊙ **Takeaway 6.** Our new decoupled scaling strategy enables better use of GPU parallelism.

GPU Batching Size. We finally evaluate the performance based on the GPU batching size. We see in Figure 9 that increasing the batching size always induces greater latency, as the Round-Trip Time (RTT) of the first packet in the batch includes the time taken to receive as many packets as there are in the batch, and leads to an equivalent or better throughput, as packets in a batch are processed in parallel.

⊙ **Takeaway 7.** In all implementations, using small batches results in poor throughput. Dynamic batch size adjustment is needed to avoid unnecessarily high latency.

⊙ **Takeaway 8.** While Takeaway 3 found parallel ROI (used by Snap) and full copies (both Snap and NBA) are efficient in terms of throughput, they

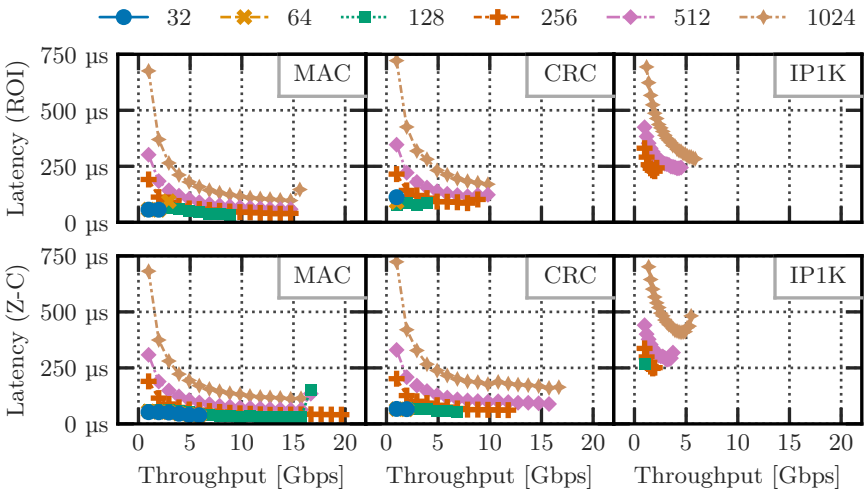


Fig. 9: Performance acc. to the GPU batching size. Packet size is 128 B, one CPU core retrieves packets and communicates with the GPU using a single queue.

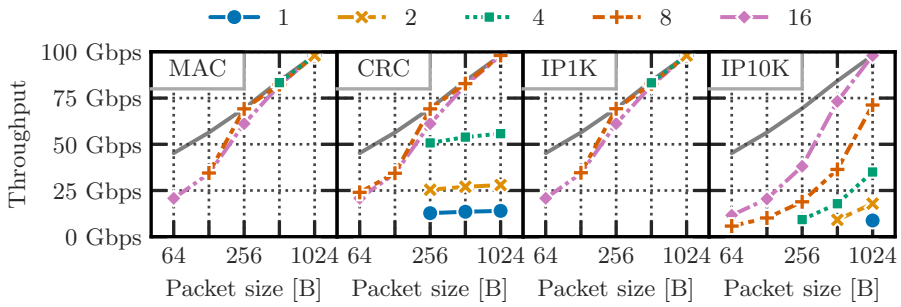
induce a high latency due to the high need for batching to realize this throughput.

4.5 GPU-Only Scalability

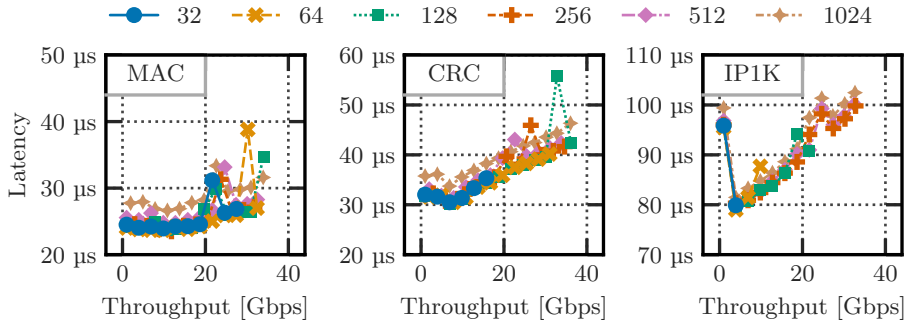
We now evaluate the GPU-only implementation, analyzing both the impact of the number of queues used to spread the load and the batching size. We see in Figure 10a that a high number of queues is needed to achieve a high throughput for all workloads. Using a low number of queues can make the implementation unable to handle a high volume of packets per queue, depending on the complexity of the use case. The problem arises when GPU processing is faster than the NIC and overwrites the NIC descriptors. This is a feature of the DOCA GPUNetIO library, stating that, for performance reasons, it is the developer's responsibility to ensure the NIC can flush the TX rings quickly enough before the GPU enqueues new packets. A heavier workload makes it more difficult to overwrite NIC descriptors because it processes more packets.

With small packets, the implementation is unable to saturate the generator rate, as already observed in Section 4.4 with the Z-C on GPU model. We also see that the throughput scales perfectly with the number of queues, as increasing it also extends the GPU utilization: one persistent kernel is running per queue. Furthermore, we see in Figure 10b that the batching size does not influence much the latency, but has a steady impact on throughput for heavy workloads.

We finally measured the same benchmark on another testbed in Appendix A. With newer generations of NICs and GPU, the limit in messages per second is alleviated.



(a) Throughput acc. to the number of GPU queues, using a batching size of 1024.



(b) Performance acc. to the GPU batching size. Packet size is 128 B. Y-axis is truncated to highlight the results.

Fig. 10: Performance of the GPU-only implementation.

⊙ **Takeaway 9.** Processing packets entirely on GPU is extremely efficient for large packets.

⊙ **Takeaway 10.** However, previous work at lower speeds failed to demonstrate the NIC has a modest packet rate when sending small packets directly to the GPU (both in Z-C or in GPU-Only).

⊙ **Takeaway 11.** To take advantage of GPU parallelism, a high number of queues is needed. GPU-Ether speed did not bring up such a need.

4.6 DPU Scalability

We evaluate the performance of CPU-based DPUs for executing NFV pipelines. We evaluate the ARM cores on both the BlueField-2 and BlueField-3, as well as the RISC-V cores (DPA) available on the BlueField-3.

ARM Scalability. We see in Figure 11 that performance differs drastically between the two DPUs. The BF2 is not able to saturate the NIC for small

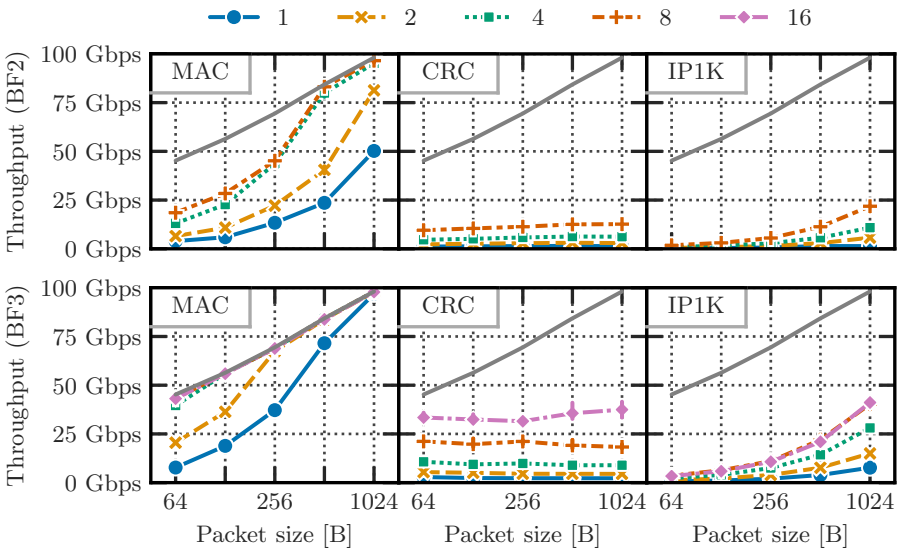


Fig. 11: Performance of the DPU implementations acc. to the number of ARM cores, running on the BlueField-2 (BF2) and BlueField-3 (BF3).

packets using its 8 ARM cores, even with a processing as lightweight as Ethernet Mirroring. With heavier workloads, the BF2's ARM cores performance is even more degraded.

The BF3, on the other hand, can reach the generator rate, doing Ethernet Mirroring for all packet sizes with only 8 of its embedded ARM cores. Using all 16 ARM cores provides a throughput close to the CPU implementation for the CRC computation and IP lookup. The performance of this DPU enables offloading even intensive tasks directly, thereby avoiding the host CPU entirely.

⊙ **Takeaway 12.** The BlueField-2 ARM cores are unable to reach the generator rate for small packets.

⊙ **Takeaway 13.** The BlueField-3 ARM cores provide similar throughput to a (x86) CPU-only implementation.

DPA Scalability. As shown in Figure 12, DPA performance scales linearly with the number of threads. For both CRC and IP Lookup operations, increasing the number of threads continuously improves performance, even beyond the 190-thread limit used in previous work [9]. We observe that the DPA on the BF3 is on par with the ARM cores on the BF2, the previous generation of BlueField cards.

⊙ **Takeaway 14.** The BlueField-3 DPA performance is similar to that of the BlueField-2 weaker ARM cores.

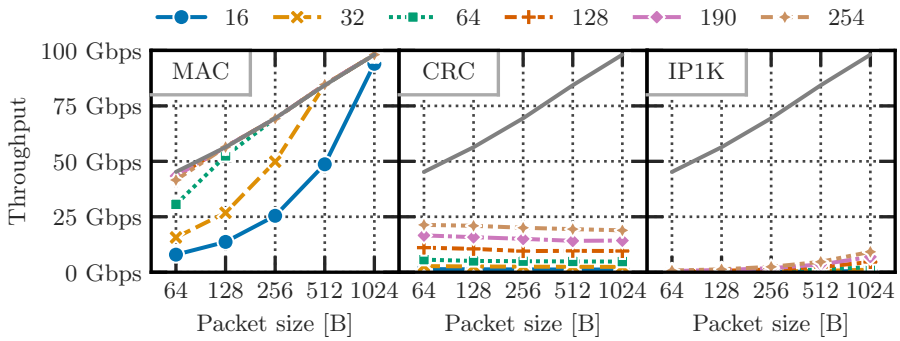


Fig. 12: Performance of the DPA implementations on the BlueField-3 with a variable number of threads.

4.7 Comparison of Processing Models and Devices

We compare in Figure 13 the performance of the CPU-only, DPU-only (both BlueField-2 ARM cores and BlueField-3 ARM and DPA cores), two scalable CPU-GPU techniques (ROI and Z-C on GPU), and GPU-only (GPU Direct) approaches. Each implementation is tuned to its best-performing point, based on all previous evaluations of parameters. We see that, in the Ethernet Mirroring use case, the generator rate is easily reached by the CPU, ROI, BF3 and DPA implementations. As explained before, the Z-C and GPU-only implementations cannot sustain the rate offered with small packets.

Regarding CRC computation, only a GPU can sustain the computation load, except for small packets where the CPU and BF3 ARM cores can saturate the generator.

On a 1K IP lookup, as the amount of data needed on the GPU is small, the ROI is advantageous. GPU Direct can sustain the computation but is still limited with small packets. It is, however, the only one able to handle the 10K lookup. By removing the CPU bottleneck, performing the packet I/O on GPU turns out to be highly beneficial for such high-load use cases.

Regarding latency, the CPU and DPU implementations generally perform the best; however, they are unable to deliver high throughput. The CPU-GPU implementations are the most latency-prone, suffering from the additional communication between the two devices and the need to batch many packets to realize this throughput.

Regarding the BF3 DPA, we see that for intensive workloads its performance is comparable to the BF2 ARM cores. From this observation, we conclude that the DPA on the BF3 can be used to offload tasks from the BF3's ARM cores, similar to how DPUs can offload tasks from the host.

⊙ **Takeaway 15.** DPU and DPA implementations provide low latencies thanks to their on-NIC positions, but struggle to sustain high throughput.

⊙ **Takeaway 16.** The CPU-GPU hybrid models' latency is high because of

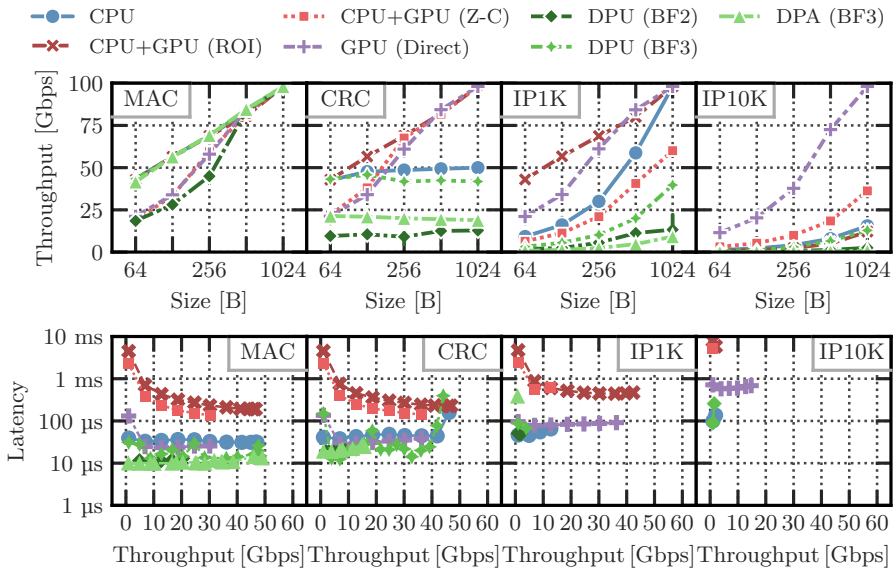


Fig. 13: Performance of each implementation on different applications. Latency is measured with packets of 128 B, and its scale is logarithmic.

the added communication between the two devices.

⊙ **Takeaway 17.** GPU Direct performs best for large packets and intensive workloads while maintaining a small latency.

4.8 Energy Efficiency

This section addresses the energy efficiency of the different implementations. First, we focus solely on the impact of various factors of the CPU power consumption. Then, we consider the total (CPU + GPU) energy consumption with regard to the application workload. We report the CPU power consumption using RAPL [32] and the GPU power consumption using the NVIDIA System Management Interface [51] tool. We observed a delta of approximately 100 W between the sum of our counters and the actual power drawn by the server reported by IPMI. This difference can be explained by other components, such as the cooling system, the motherboard and disks. For the DPUs, we use a Benchlab and its PCIe adapter [6] that cuts down the PCIe power from the motherboard and replaces it with a monitored power supply connected through USB.

CPU Down-Scaling. When the host CPU is processing packets, as DPDK is polling, the CPU is always busy at 100 % and consumes a significant amount of power. In addition to the approach of Sloth [11] that uses core count and

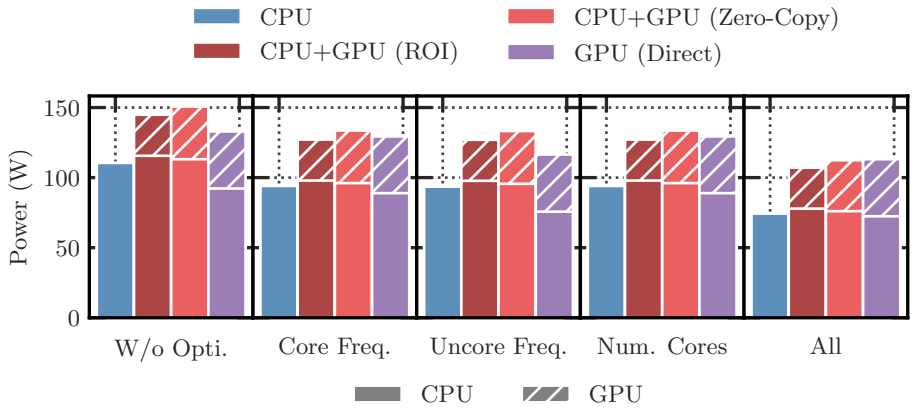


Fig. 14: Impact of core and uncore frequency scaling and core count on the power consumption of the CPU and GPU running Ethernet mirroring. The rate is fixed at 100 Gbps. DPA and DPU implementations are not included due to their insensitivity to the host CPU.

core frequency scaling as levers to reduce energy consumption, we introduce the study of uncore frequency scaling.

Section 2.4 explained how DVFS can be used to slow down the cores of a CPU, or the uncore (memory controller, PCIe root complex, and so on). While the core frequency can be adjusted through built-in tools within the Linux kernel, the uncore frequency relies on vendor-dependent tools to interact with hardware components. On AMD processors, such changes require communicating with the hardware through the HSMP mailbox system to adjust *fabric* and *memory* clocks [1].

Figure 14 presents the impact of core count and core/uncore frequency scaling on power consumption. Except from GPU Direct, each implementation benefits from all these optimizations and further power reduction can be achieved through a combination of these techniques. Following this approach, the power consumption of the CPU implementation is reduced by more than 30 W.

Although the GPU Direct implementation bypasses the CPU, therefore canceling any effort to reduce energy consumption by reducing core count or core frequency, adjusting the uncore frequency leads to a $\sim$ 15 W reduction as packets still flow through the PCIe root complex. Even with the uncore at its minimal frequency, the CPU draws more than $\sim$ 70 W, although it is not processing a single packet.

⊙ **Takeaway 18.** CPU core and uncore frequency scaling, and/or core count scaling, benefits to all implementations' power consumption.

⊙ **Takeaway 19.** The CPU power consumption is still high even when it is completely bypassed.

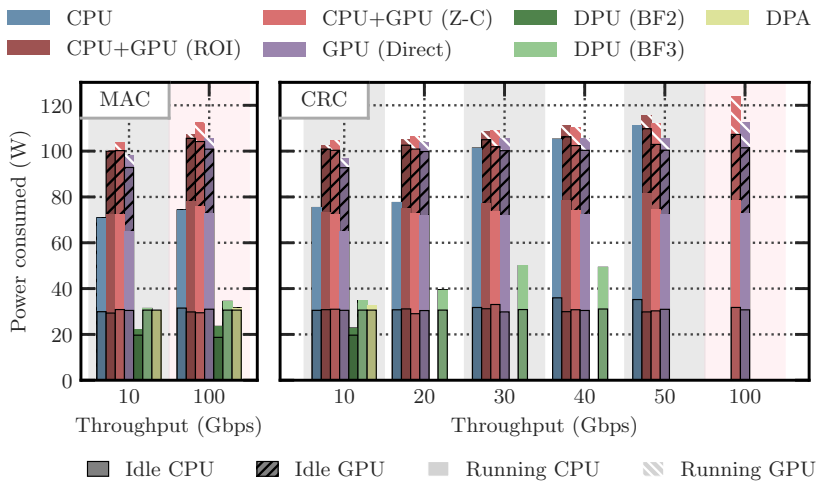


Fig. 15: Power consumed by the different processing models at various speeds, on 1024 B packets. The missing bars at different rates indicate that the system cannot sustain the generator rate, so there are no valid data points.

⊙ **Takeaway 20.** Adjusting the uncore frequency can further reduce the power consumption, although it remained essentially unexplored in previous work.

Energy Tradeoff. Figure 15 shows the consumption of the system under test at various input rates, with 1024 B packets. Bars are only displayed for systems that can keep up with the input rate. We run the Ethernet Mirroring use case, where all proposed implementations can forward traffic at 100 Gbps, and the CRC use case.

The CPU+GPU approaches are less energy-efficient, as they pay the price of being a hybrid system with two relatively high idle consumption. The CPU draws up to 35 W when idle while the GPU consumes around 25 W idling.

The DPU approaches are by far the most energy-efficient ones. However, they can only sustain a relatively low rate. For the CRC use case, the BF2 is limited to 11 Gbps, and the BF3 to 43 Gbps, at which point they respectively draw 24 W and 50 W. Moreover, the power consumption of the DPU includes the whole board capable of running in standalone mode, whereas for other methods, we only report the power of the CPU and/or the GPU. The CPU approach follows in terms of efficiency, but fails to reach 100 Gbps.

⊙ **Takeaway 21.** DPU ARM and DPA cores are the most power efficient, but are limited in terms of performance.

⊙ **Takeaway 22.** The DPU power consumption is dominated by idle

consumption. Therefore, for the BF3, using the more efficient DPA instead of the ARM cores does not significantly reduce the consumption.

⊙ **Takeaway 23.** In terms of power consumption, GPU Direct is best when heavy computations are needed.

5 Conclusion

We propose xPUBench, a benchmarking environment that allows us to review the current body of research on packet processing using GPUs and DPUs. Our analysis shows that there is no one-size-fits-all solution. Each approach has its own strengths and drawbacks. Some existing models rely on a single master core to interface with the GPU. This approach struggles to scale with the speeds of modern NICs. Parallel models make each worker communicate directly with the GPU, removing the single core bottleneck. We evaluate the use of new capabilities of recent NICs such as Zero-Copy on GPU memory, and introduce a new model of scalability by allowing each worker to communicate with the GPU through multiple queues. Our approach achieves over a 2× improvement in throughput compared to a 16-cores CPU implementation on heavy computations, reaching 100 Gbps with a single CPU core.

Our evaluation demonstrates that the use of a GPU to run the entire pipeline, without CPU involvement, provides higher throughput than hybrid solutions that rely on the CPU for I/O operations, while offering a highly reduced latency. However, current hardware fails to handle small packets at high rate. Even when avoiding CPU processing entirely, GPU solutions rely on CPU resources during PCIe transfers, which prevents the CPU from entering low-power states. While this limitation can be partially mitigated by slowing down the CPU uncore frequency, it remains a significant burden on the system's power consumption.

Finally, we showed that DPUs like the BlueField-2 and 3 can be used as coprocessors for NFV workloads. They offer the lowest latencies, as the ARM CPUs and DPA reside directly on the NIC. While the BF2 ARM cores and the DPA are very limited in terms of compute capacity, we observe that the more powerful BF3 ARM cores put the whole card on par with the host CPU in terms of performance, using less than half the power consumed by the host CPU alone.

A Hardware Impact on GPU-only

We also evaluate the performance of the GPU-only implementation on another testbed. A HPE server is equipped with an NVIDIA L40S GPU and a BlueField-3 in NIC mode, connected via a PCIe 5.0 x16 system bus. The two are not connected through a dedicated PCIe switch. Results are shown in Figure 16. In this situation, performance results are different with respect to the main testbed. There are no more points where the implementation is unable to answer, as the NIC is faster to flush the descriptors in its rings. Overall, benchmarks show better numbers in terms of throughput, as implied by the better GPU used.

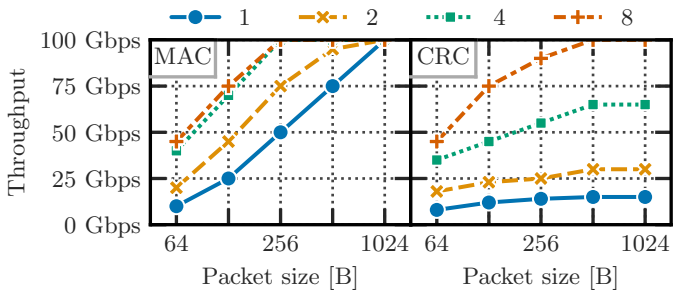


Fig. 16: Throughput of the GPU-only implementation using an NVIDIA L40S GPU and a BlueField-3 acc. to the number of queues.

Acknowledgments. The authors would like to acknowledge the anonymous reviewers for their valuable feedback. This work was supported by the Walloon region's CyberExcellence program under Grant № 2110186, and by the UCLouvain FSR. It was also supported by the Fonds de la Recherche Scientifique – FNRS MIS Grant № 40020886. Maxime Vanliefde is a Research Fellow of the FNRS. Clément Delzotti is a FRIA grantee of the FNRS. We also thank NVIDIA for granting a BlueField-2 used in prototyping.

Ethical Considerations. This work does not raise any ethical concerns.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Advanced Micro Devices, Inc.: AMD HSMP, `https://github.com/amd/amd_hsmp`
2. Ahmed, M.R., Shatabda, S., Islam, A.M., et al.: Intrusion Detection System in Software-Defined Networks Using Machine Learning and Deep Learning Techniques–A Comprehensive Survey. Authorea Preprints (2023)
3. Barbette, T.: Network performance framework, `https://github.com/tbarbette/npf`
4. Barbette, T., Katsikas, G.P., Maguire, G.Q., et al.: Rss++: load and state-aware receive side scaling. (CoNEXT '19), ACM, New York, NY, USA (2019). https://doi.org/10.1145/3359989.3365412
5. Barbette, T., Soldani, C., Mathy, L.: Fast Userspace Packet Processing. In: ANCS '15. p. 5–16. IEEE Computer Society, USA
6. BENCHLAB: Measuring pcie slot power consumption (5 2024), `https://benchlab.io/blogs/technical/measuring-pcie-slot-power-consumption`
7. Bonfim, M.S., Dias, K.L., Fernandes, S.F.L.: Integrated nfv/sdn architectures: A systematic literature review. ACM Comput. Surv. **51**(6) (Feb 2019). https://doi.org/10.1145/3172866

8. Chaurasia, A.K., Garg, A., Raman, B., et al.: Simmer: Rate proportional scheduling to reduce packet drops in vgpu based nf chains. (ICPP '22), ACM, New York, NY, USA (2023). https://doi.org/10.1145/3545008.3545068
9. Chen, X., Zhang, J., Fu, T., et al.: Demystifying datapath accelerator enhanced off-path smartnic. In: 2024 IEEE 32nd International Conference on Network Protocols (ICNP '24). pp. 1–12. IEEE (2024)
10. Corbalan, J., Vidal, O., Alonso, L., et al.: Explicit uncore frequency scaling for energy optimisation policies with EAR in Intel architectures. In: 2021 IEEE International Conference on Cluster Computing (CLUSTER '21). pp. 572–581 (Sep 2021). https://doi.org/10.1109/Cluster48925.2021.00089, iSSN: 2168-9253
11. Delzotti, C., Maistriaux, P., Barbette, T.: Sloth: A kernel-bypass scheduler maximizing energy efficiency under latency constraints. In: IFIP 2025 Slices Workshop-International Federation for Information Processing (IFIP '25) Networking 2025 Conference (2025)
12. Di Girolamo, S., Kurth, A., Calotoiu, A., et al.: A risc-v in-network accelerator for flexible high-performance low-power packet processing. In: (ISCA '21). pp. 958–971. IEEE (2021)
13. DPDK: Generic flow api (rte_flow) (12 2026), `https://doc.dpdk.org/guides-24.07/prog_guide/rte_flow.html`
14. Faltelli, M., Belocchi, G., Quaglia, F., et al.: Metronome: adaptive and precise intermittent packet retrieval in DPDK. pp. 406–420. (CoNEXT '20), ACM, New York, NY, USA (Nov 2020). https://doi.org/10.1145/3386367.3432730
15. Farshin, A., Barbette, T., Roozbeh, A., et al.: Packetmill: toward per-core 100-gbps networking. (ASPLOS '21), ACM, New York, NY, USA (2021). https://doi.org/10.1145/3445814.3446724
16. fd.io: Vpp, `https://wiki.fd.io/view/VPP/`
17. Feng, Y., Panda, S., Kulkarni, S.G., et al.: A smartnic-accelerated monitoring platform for in-band network telemetry. In: 2020 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN. pp. 1–6. IEEE (2020)
18. Ghasemirahni, H., Farshin, A., Scazzariello, M., et al.: Fajita: Stateful packet processing at 100 million pps (CoNEXT '24)
19. Go, Y., Jamshed, M., Moon, Y., et al.: APUNet: revitalizing GPU as packet processing accelerator. In: Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation. p. 83–96. (NSDI'17), USENIX Association, USA (2017)
20. Guo, Z., Lin, J., Bai, Y., et al.: Lognic: A high-level performance model for smartnics. In: Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture. pp. 916–929 (2023)
21. Han, S., Jang, K., Panda, A., et al.: Softnic: A software nic to augment hardware. EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2015-155 (2015)
22. Han, S., Jang, K., Park, K., et al.: PacketShader: A GPU-Accelerated Software Router. ACM SIGCOMM Computer Communication Review **40**(4), 195–206 (2010)
23. Harris, S.L., Chaver, D., Piñuel, L., et al.: Rvfpga: Using a risc-v core targeted to an fpga in computer architecture education. In: 2021 31st International Conference on Field-Programmable Logic and Applications (FPL). pp. 145–150. IEEE (2021)
24. Hasanth, K.M., Basu, S., Nadig, D.: Data processing unit (dpu) based network process offloading for efficient service meshes. In: 2024 IEEE 10th International Conference on Network Softwarization (NetSoft). pp. 319–321 (June 2024). https://doi.org/10.1109/NetSoft60951.2024.10588912

25. Huang, D., Costero, L., Atienza, D.: Is the powersave governor really saving power? In: 2024 IEEE 24th International Symposium on Cluster, Cloud and Internet Computing (CCGrid). pp. 273–283 (May 2024). https://doi.org/10.1109/CCGrid59990.2024.00039, iSSN: 2993-2114
26. Humphries, J.T., Natu, N., Kaffes, K., et al.: Wave: Offloading resource management to smartnic cores. In: Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3. pp. 264–281 (2025)
27. Intel Corporation: Improving network performance in multi-core systems (2007), `https://www.intel.com/content/dam/support/us/en/documents/network/sb/318483001us2.pdf`, White Paper
28. Jung, C., Kim, S., Yeom, I., et al.: GPU-Ether: GPU-native Packet I/O for GPU Applications on Commodity Ethernet. In: IEEE INFOCOM 2021 - IEEE Conference on Computer Communications. pp. 1–10 (2021). https://doi.org/10.1109/INFOCOM42981.2021.9488699
29. Kalia, A., Zhou, D., Kaminsky, M., et al.: Raising the bar for using gpus in software packet processing. (NSDI '15), USENIX Association, USA
30. Katsikas, G.P., Barbette, T., Chiesa, M., et al.: What you need to know about (smart) network interface cards. In: International Conference on Passive and Active Network Measurement. pp. 319–336. Springer (2021)
31. Kfoury, E.F., Choueiri, S., Mazloum, A., et al.: A comprehensive survey on smartnics: Architectures, development models, applications, and research directions. IEEE Access (2024)
32. Khan, K.N., Hirki, M., Niemi, T., et al.: Rapl in action: Experiences in using rapl for power measurements. ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS) **3**(2), 1–26 (2018)
33. Khazraee, M., Forencich, A., Papen, G.C., et al.: Rosebud: Making fpga-accelerated middlebox development more pleasant. In: Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3. pp. 586–605 (2023)
34. Kim, J., Jang, K., Lee, K., et al.: Nba (network balancing act): a high-performance packet processing framework for heterogeneous processors. In: Proceedings of the Tenth European Conference on Computer Systems. EuroSys '15, ACM, New York, NY, USA (2015). https://doi.org/10.1145/2741948.2741969
35. Kohler, E., Morris, R., Benjie, C., et al.: The Click Modular Router. ACM Transactions on Computer Systems **18**(3), 263–297 (2000)
36. Lévai, T., Németh, F., Raghavan, B., et al.: Batchy: batch-scheduling data flow graphs with service-level objectives. In: (NSDI '20)
37. Li, P., Luo, Y.: P4gpu: Accelerate packet processing of a p4 program with a cpu-gpu heterogeneous architecture. In: Proceedings of the 2016 Symposium on Architectures for Networking and Communications Systems. pp. 125–126 (2016)
38. Li, T.H., Chu, H.M., Wang, P.C.: Ip address lookup using gpu. In: 2013 IEEE 14th International Conference on High Performance Switching and Routing (HPSR). pp. 177–184. IEEE (2013)
39. Li, X., Cheng, W., Zhang, T., et al.: Power Efficient High Performance Packet I/O. ICPP '18, ACM, New York, NY, USA. https://doi.org/10.1145/3225058.3225129
40. Li, Y., Zhang, D., Liu, A.X., et al.: Gamt: a fast and scalable ip lookup engine for gpu-based software routers. In: Architectures for Networking and Communications Systems. pp. 1–12. IEEE (2013)
41. Lin, H., Wang, C.L.: Efficient low-latency packet processing using on-gpu thread-data remapping. Journal of parallel and distributed computing **133**, 51–62 (2019)

42. Lin, J., Guo, Z., Shah, M., et al.: Enabling portable and high-performance smartnic programs with alkali. In: (NSDI '25)
43. Liu, M., Cui, T., Schuh, H., et al.: Offloading distributed applications onto smartnics using ipipe. In: Proceedings of the ACM Special Interest Group on Data Communication, pp. 318–333 (2019)
44. Mafioletti, D.R., Dominicini, C.K., Martinello, M., et al.: PIaFFE: A Place-as-you-go In-network Framework for Flexible Embedding of VNFs. In: ICC 2020 - 2020 IEEE International Conference on Communications (ICC). pp. 1–6 (2020). https://doi.org/10.1109/ICC40277.2020.9149240
45. Miano, S., Doriguzzi-Corin, R., Risso, F., et al.: Introducing smartnics in server-based data plane processing: The ddos mitigation use case. IEEE Access **7**, 107161–107170 (2019)
46. Miano, S., Sanaee, A., Risso, F., et al.: Domain specific run time optimization for software data planes. In: Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '22). pp. 1148–1164 (2022)
47. Mijumbi, R., Serrat, J., Gorricho, J.L., et al.: Network function virtualization: State-of-the-art and research challenges. IEEE Communications surveys & tutorials **18**(1), 236–262 (2015)
48. Natori, K., Fujimoto, K., Shiraga, A.: Sleep control of packet receiving thread toward power saving. Annals of Telecommunications (Apr 2025). https://doi.org/10.1007/s12243-025-01084-2
49. Navarre, L., Michel, F., Barbette, T.: A high-speed robust tunnel using forward erasure correction in segment routing. In: (ICNP '24)
50. NVIDIA Corporation: Cuda c++ programming guide, `https://docs.nvidia.com/cuda/cuda-c-programming-guide/`
51. NVIDIA Corporation: System management interface smi, `https://developer.nvidia.com/system-management-interface`
52. NVIDIA Corporation: Nvidia bluefield-2 dpu (2021), `https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/documents/datasheet-nvidia-bluefield-2-dpu.pdf`
53. NVIDIA Corporation: Nvidia bluefield-3 dpu (2021), `https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/documents/datasheet-nvidia-bluefield-3-dpu.pdf`
54. NVIDIA Corporation: Doca gpunetio (2024), `https://docs.nvidia.com/doca/sdk/doca+gpunetio/index.html`
55. NVIDIA Corporation: Nvidia connectx-7 400g adapters (2024), `https://www.nvidia.com/content/dam/en-zz/Solutions/networking/infiniband/connectx-7-datasheet.pdf`
56. NVIDIA Corporation: Dpa subsystem (6 2025), `https://docs.nvidia.com/doca/sdk/dpa+subsystem/index.html`
57. PCI-SIG: PCI Express Base Specification Revision 5.0 Version 1.0. PCI-SIG, available at: `https://picture.iczhiku.com/resource/eetop/SYkDTqhOLhpUTnMx.pdf`
58. Poli, L., Saha, S., Zhai, X., et al.: Design and implementation of a risc v processor on fpga. In: 2021 17th international conference on mobility, sensing and networking (MSN). pp. 161–166. IEEE (2021)
59. Poutievski, L., Mashayekhi, O., Ong, J., et al.: Jupiter evolving: transforming google's datacenter network via optical circuit switches and software-defined networking. In: Proceedings of the ACM SIGCOMM 2022 Conference. pp. 66–85 (2022)

60. Prekas, G., Primorac, M., Belay, A., et al.: Energy proportionality and workload consolidation for latency-critical applications. In: Proceedings of the Sixth ACM Symposium on Cloud Computing. pp. 342–355. SoCC '15, ACM, New York, NY, USA (Aug 2015). https://doi.org/10.1145/2806777.2806848
61. Ring, W.: 100 gbit interconnects and above: The need for speed (2007)
62. RIPE NCC: Routing information service (ris) (2024), https://www.ripe.net/analyse/internet-measurements/routing-information-service-ris/
63. Rizzo, L.: netmap: a novel framework for fast packet i/o. In: 21st USENIX Security Symposium (USENIX Security 12). pp. 101–112 (2012)
64. Romein, J.W.: Breaking the i/o barrier: 1.2 tb/s ethernet packet processing on a gpu. In: European Conference on Parallel Processing. pp. 225–238. Springer (2025)
65. Sahni, S., Kim, K.S.: Efficient construction of multibit tries for ip lookup. IEEE/ACM Transactions on Networking **11**(4), 650–662 (2003)
66. Shoaib, N., Shamsi, J., Mustafa, T., et al.: Gdpi: Signature based deep packet inspection using gpus. International Journal of Advanced Computer Science and Applications **8**(11) (2017)
67. Silberstein, M., Kim, S., Huh, S., et al.: Gpunet: Networking abstractions for gpu programs. ACM Transactions on Computer Systems (TOCS '16) **34**(3), 1–31 (2016)
68. Sonai, V., Bharathi, I., Noor Mahammad, S.: A perspective of ip lookup approach using graphical processing unit (gpu). In: International Conference on Distributed Computing and Intelligent Technology. pp. 98–103. Springer (2023)
69. Sun, W., Ricci, R.: Fast and Flexible: Parallel Packet Processing with GPUs and Click. In: Architectures for Networking and Communications Systems. pp. 25–35. IEEE (2013)
70. The Linux Foundation Projects: Dpdk, https://www.dpdk.org/
71. Vasiliadis, G., Koromilas, L., Polychronakis, M., et al.: GASPP: A GPU-Accelerated stateful packet processing framework. In: 2014 USENIX Annual Technical Conference (ATC '14). pp. 321–332 (2014)
72. Vasiliadis, G., Polychronakis, M., Ioannidis, S.: Parallelization and characterization of pattern matching using gpus. In: 2011 IEEE International Symposium on Workload Characterization (IISWC). pp. 216–225. IEEE (2011)
73. Vogt, F.G., Rodriguez, F., Luizelli, M.C., et al.: Poster: Towards in-network resource scaling of vnfs. In: CoNEXT '24. pp. 29–30
74. Wang, Y., Zu, Y., Zhang, T., et al.: Wire speed name lookup: A gpu-based approach. In: (NSDI '13)
75. Wiles, K.: Pktgen-dpdk, https://github.com/pktgen/Pktgen-DPDK
76. Yi, X., Wang, J., Duan, J., et al.: Flowshader: A generalized framework for gpu-accelerated vnf flow processing. In: (ICNP '19)
77. Zhang, K., He, B., Hu, J., et al.: G-net: Effective gpu sharing in nfv systems. In: (NSDI '18)