BenchCouncil Transactions on Benchmarks, Standards and Evaluations 3 (2023) 100120

Contents lists available at ScienceDirect

BenchCouncil Transactions on Benchmarks, Standards and Evaluations

journal homepage: www.keaipublishing.com/en/journals/benchcouncil-transactions-on-benchmarks-standards-and-evaluations/

Full length article

DPUBench: An application-driven scalable benchmark suite for comprehensive DPU evaluation

Zheng Wang [a,b], Chenxi Wang [a,b,*], Lei Wang [a,b]

[a] Institute of Computing Technology, Chinese Academy of Sciences, China
[b] University of Chinese Academy of Sciences, China

ARTICLE INFO

Keywords:
DPU
Evaluation
Benchmark
Application-driven

ABSTRACT

With the development of data centers, network bandwidth has rapidly increased, reaching hundreds of Gbps. However, the network I/O processing performance of CPU improvement has not kept pace with this growth in recent years, which leads to the CPU being increasingly burdened by network applications in data centers. To address this issue, Data Processing Unit (DPU) has emerged as a hardware accelerator designed to offload network applications from the CPU. As a new hardware device, the DPU architecture design is still in the exploration stage. Previous DPU benchmarks are not neutral and comprehensive, making them unsuitable as general benchmarks. To showcase the advantages of their specific architectural features, DPU vendors tend to provide some particular architecture-dependent evaluation programs. Moreover, they fail to provide comprehensive coverage and cannot adequately represent the full range of network applications. To address this gap, we propose an **application-driven** scalable benchmark suite called **DPUBench**. DPUBench classifies DPU applications into three typical scenarios — network, storage, and security, and includes a scalable benchmark framework that contains essential Operator Set in these scenarios and End-to-end Evaluation Programs in real data center scenarios. DPUBench can easily incorporate new operators and end-to-end evaluation programs as DPU evolves. We present the results of evaluating the NVIDIA BlueField-2 using DPUBench and provide optimization recommendations. DPUBench are publicly available from https://www.benchcouncil.org/DPUBench.

1. Introduction

In the past decade, the growth rate of CPU performance has been relatively slow due to the physical limitations it faces [2]. As the size of transistor circuits approaches the scale of atoms, increasing challenges caused by physical limitations, such as leakage, have led to the failure of Dennard Scaling Law [3]. In contrast, many emerging computing fields, such as artificial intelligence (AI), big data, and the Internet of Things, are thriving as computing resources reach a threshold scale. The demand for computing resources in these fields is rapidly growing, resulting in CPU becoming increasingly incapable of meeting it in data centers. As a result, deploying specialized chips, such as GPU, TPU [4], and DPU, in data centers has become a new trend for both academia and industry.

DPU is a hardware accelerator designed to offload network applications from CPU in data centers. With the increase in network bandwidth from 10 Gbps to 25 Gbps, 40 Gbps, 100 Gbps, 200 Gbps, and even 400 Gbps, CPU has become increasingly burdened by network applications, and its computing resources are heavily consumed before processing computing applications. To ensure that CPU's computing resources are focused on CPU-bound applications, DPU has emerged.

DPU typically consists of multiple hardware accelerators for network applications, a multi-core CPU for scheduling and programming, and high-bandwidth network IO interfaces [5]. As an emerging hardware accelerator, the DPU architecture has not yet been standardized and is decided by DPU manufacturers. Typical DPU architectures include those that can fully offload infrastructural network applications in data centers, such as NVIDIA Bluefield [6]; those that offload specific network application scenarios in data centers, such as YUSUR KPU [7–10]; and programmable architectures developed based on FPGA, such as Intel Mount Evans [11].

Benchmarking is a widely-used research method in computer science for evaluating the performance of systems. Benchmarking evaluations can provide insights into the actual performance of the evaluated object and can guide future co-design and optimization of software and hardware. With the development of DPU and data centers, a DPU benchmark suite is necessary. However, to the best of our knowledge, there is currently no benchmark suite available for comprehensive

* Corresponding author.
E-mail addresses: wz917942636@gmail.com (Z. Wang), wangchenxi21s@ict.ac.cn (C. Wang), wanglei_2011@ict.ac.cn (L. Wang).

https://doi.org/10.1016/j.tbench.2023.100120
Received 2 June 2023; Received in revised form 4 July 2023; Accepted 5 July 2023
Available online 7 July 2023

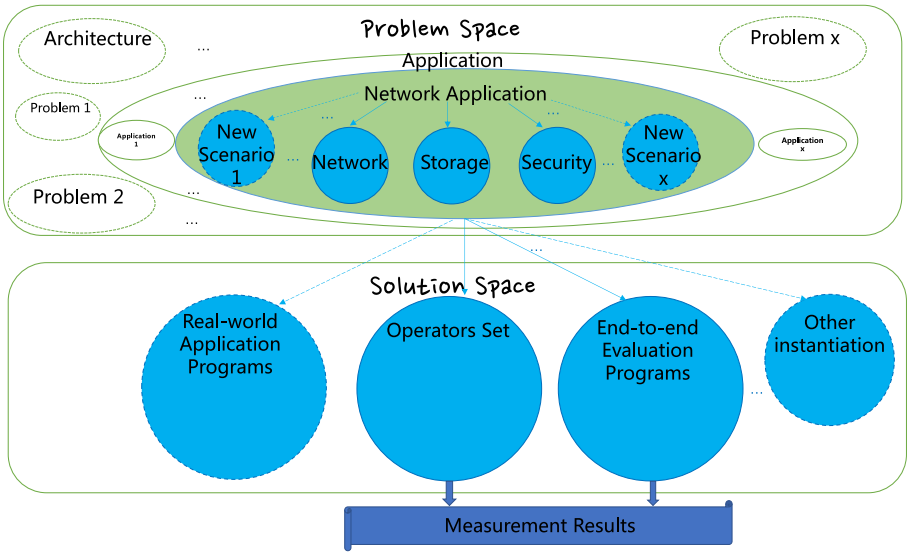

Fig. 1. The overview of DPUBench. Inspired by Zhan's [1] benchmarking methodology, DPUBench comprises problem definition, instantiation, and measurement. The solid lines in the figure indicate the current implementations of DPUBench, while the dashed lines indicate future implementations that can be added or other benchmark implementations. The elliptical box represents the problem definition, the thin arrow, and circle denote the specific instantiation, and the thick arrow at the bottom represents the measurement results output of DPUBench.

DPU evaluation. Existing DPU evaluation programs are either provided by DPU manufacturers [7–10,12,13], which are based on their specific DPU architecture design, or they are selected and rewritten from some open source benchmark programs based on the architecture characteristics of the evaluated DPU in previous research [14–19]. These previous DPU evaluation programs are architecture-dependent, meaning that they are designed based on a specific architecture and not suitable to evaluate DPU with different architectures. To perform a comprehensive DPU evaluation, the architecture-dependent DPU benchmark programs are not feasible at this stage because the DPU architecture has not yet been standardized and is undergoing rapid evolution. Even evolving DPU architectures from the same manufacturer may have significant differences.

Zhan [1] proposed a benchmarking methodology from the problem definition, instantiation, and measurement, making benchmark design and research more standardized and theoretical. We utilized Zhan's methodology to develop our benchmark suite, DPUBench and adopted an application-driven approach at the problem definition stage. For problem instantiation, we selected network, storage, and security as the typical DPU application scenarios. At the solution instantiation stage, we constructed operators in these scenarios to evaluate early DPU designs and developed end-to-end evaluation programs to obtain results in a real data center environment. DPUBench is an application-driven scalable benchmark framework that can easily incorporate new operators and end-to-end evaluation programs as DPU evolves. As an application-driven benchmark, DPUBench can add new DPU application scenarios and corresponding operators and end-to-end evaluation programs, regardless of any changes to the DPU architecture. A DPUBench overview is presented in Fig. 1.

Our contributions are as follows.

(1) We present DPUBench, an application-driven scalable benchmark suite for comprehensive DPU evaluation. DPUBench is scalable and standardized, which can accommodate new operators and end-to-end evaluation programs as DPU architecture evolves, making it a comprehensive and fair benchmark suite for DPU evaluation.

(2) We select network, storage, and security as typical DPU application scenarios and extract 16 representative operators from real-world applications. Our experiments demonstrate that these operators have good representativeness, diversity, and coverage, making them suitable for low-cost evaluation of early-stage DPU designs.

(3) We develop two end-to-end DPU workloads for typical DPU applications and measure their throughput, packet loss ratio, Server CPU utilization ratio and latency in a real data center machine. Our experiments demonstrate the effectiveness of end-to-end evaluation programs in assessing the performance of DPUs in real-world network applications.

(4) We evaluate NVIDIA BlueField-2 [6] using DPUBench and provide optimization recommendations. Our experiments reveal that NVIDIA BlueField-2 can efficiently offload network applications from the CPU, particularly network storage protocol and DPI applications. In the end-to-end evaluation, we demonstrate that NVIDIA BlueField-2 can effectively reduce the server CPU utilization ratio in network applications and allocate more CPU computing resources for computing applications.

The rest of this paper is structured as follows: Section 2 provides the background and motivation. Section 3 introduces the methodology of DPUBench. Section 4 presents the Operator Set in DPUBench and the corresponding experimental results. Section 5 discusses the End-to-end Evaluation Programs in DPUBench along with their respective experiment results. Section 6 concludes with a discussion of related work, while Section 7 outlines the conclusions and plans for further work.

2. Background and motivation

In this section, we will first introduce the background of DPUBench, including existing DPU benchmarks, DPU evaluation programs, and DPU characterization studies. Next, we will have a brief introduction to the NVIDIA BlueField-2 DPU. Based on the above discussion, we will provide the motivation for DPUBench.

2.1. Background of DPUBench

DPU is a new hardware accelerator in data centers designed to offload network applications from the CPU. However, due to the lack of standardized DPU architectures, DPU evaluation is typically conducted by DPU manufacturers who provide evaluation programs that are

Table 1
The overview of representative existing DPU evaluation studies.

Benchmark/Programs	Provider	Workload	Metric	Evaluation object
RXPBench [12]	NVIDIA	Regular Expression Matching	Time	NVIDIA BlueField DPU
Evaluation Programs [13]	Liguori	Hypervisor	Performance Metric	Amazon Nitro DPU
Evaluation Programs [7–10]	YUSUR	SQL Program	Latency	YUSUR DPU
Evaluation Programs [14]	Wei	RDMA Read/Write & Send/Recv Request	End-to-end Latency Throughput Bottleneck	NVIDIA BlueField-2
Evaluation Programs [15]	Ibanez	RPC Program	Wire-to-wire Lattency Throughput	RPC SmartNIC
Evaluation Programs [16]	Ma	Matrix Multiplication	Time Training Time	AI SmartNIC
Evaluation Programs [17]	Mandal	RDMA Read/Write	Throughput	Storage SmartNIC
Evaluation Programs [18]	Sabin	RDMA Read/Write	Throughput	Security SmartNIC
Evaluation Programs [19]	Bosshart	Network Transport Protocol	Latency	SDN SmartNIC

tailored to their own products or by researchers who select and rewrite existing benchmark programs based on the architectural characteristics of a specific DPU product. Currently, there is no benchmark suite available for comprehensive DPU evaluation. Table 1 has summarized some representative DPU evaluation studies from previous work.

From Table 1, we observe that all of the DPU evaluation programs [7–10,12–19] listed are designed for DPUs with similar architecture or for SmartNICs with specific acceleration units. Moreover, eight out of nine of these programs are [7–10,12,13,15–19] designed for one single specific application scenario, which results in inadequate coverage for comprehensive DPU evaluation. The lack of evaluation programs in a real network environment also limits the efficacy and reliability of the results. Three out of nine of these programs [12,13,16] are not conducted in a real network environment in data centers, further limiting their relevance to real-world network applications evaluation. Additionally, three out of nine of these programs [12,13,16] only measure performance metrics commonly used for CPU evaluation, which are insufficient for DPU evaluation as they do not take into account network-related metrics such as network throughput and latency.

2.2. NVIDIA BlueField-2 DPU

NVIDIA BlueField-2 is a typical DPU that is designed to fully offload infrastructural network applications in data centers. Its architecture, as shown in Fig. 2, integrates a variety of hardware acceleration units for network applications, high bandwidth network IO interfaces, a multi-core ARM AArch64 processor, and optional on-board DRAM of either 16 GB or 32 GB [6]. The hardware acceleration units of NVIDIA BlueField-2 can help offload infrastructural network application operators, such as the regular expression matching acceleration unit (Reg-Ex) for regular expression matching operator and the public key encryption and decryption acceleration unit (Public-Key Crypto) for public key encryption and decryption operator. The high bandwidth network IO interfaces include the ConnectX interface with two 100 Gbps Remote Direct Memory Access (RDMA) [20] ports or one single 200 Gbps Ethernet/InfiniBand [21] port, which are used in production environments. The ARM AArch64 processor contains 8 Cortex-A72 cores with a 2.75 GHz frequency, sharing a 4 MB L2 Cache between cores and an 8MB L3 Cache among the units of NVIDIA BlueField-2. As for memory units, NVIDIA BluField2 equips with DDR4-1600 DRAM and eMMC flash memory, which are used for storage that will not be lost after a power failure.

2.3. Motivation of DPUBench

Section 2.1 has provided a brief introduction of representative existing DPU benchmarks or evaluation programs, all of which are

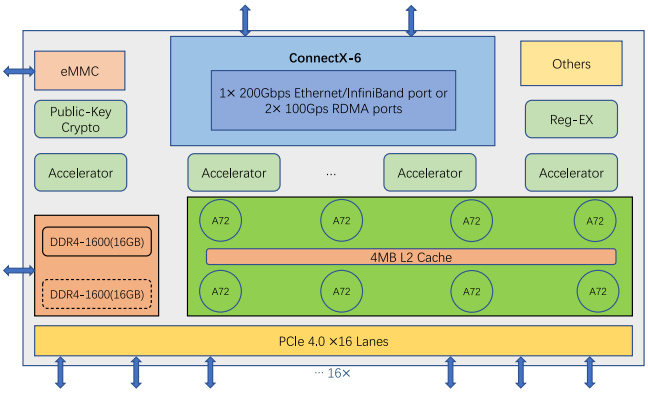

Fig. 2. The architecture of NVIDIA BlueField-2.

used for one specific DPU or DPU with one specific architecture. However, there is currently no DPU benchmark that can effectively evaluate DPUs with different architectures, which is a significant gap in the field. Furthermore, DPU architecture is rapidly evolving due to the increasing CPU computing resources that need to be offloaded in data centers, resulting in significant differences in DPU architectures between different DPU manufacturers or even the adjacent generations of the same manufacturer. Therefore, a scalable DPU benchmark that can evaluate DPUs of different architectures is needed, and it should be able to support the addition of new evaluation programs and metrics to accommodate the rapid development of DPUs.

Another motivation behind DPUBench is to ensure the representativeness and coverage of the benchmark suite, as well as the effectiveness and reliability of the evaluation results. In terms of coverage, the benchmark programs should not only be at a certain scale to handle basic network application scenarios but also not impose excessive evaluation costs in terms of time and resource utilization. Additionally, to ensure the reliability of the evaluation results, network-related metrics should be carefully selected, and DPUs should be evaluated in a real network environment within data centers.

3. Methodology

A study is typically aimed at solving a specific problem or class of problems with corresponding solutions. To construct DPUBench, we divide the process into problem space and solution space and implement it step by step in these two spaces. Our methodology for DPUBench is illustrated in Fig. 1, which includes problem definition, problem instantiation, solution instantiation, and measurement results. This methodology is inspired by Zhan's benchmark science methodology [1]

of problem definition, instantiation, and measurement. By providing a clear and detailed description of each step in the construction of DPUBench, our methodology makes it easy to develop, maintain, and update. In this section, we will provide a detailed introduction to the various steps involved in constructing DPUBench.

3.1. Problem definition

The first step in constructing DPUBench is problem definition, which involves clarifying the research object and establishing a clear research direction. The problem definition is critical in the problem space as there are numerous problems, and failure to define the problem may lead to deviations or even irrelevance in the final research results. For example, DPU is used to offload network applications from CPU in data centers, so the evaluation of DPU should focus on network application issues. Without proper problem definition, the evaluation metrics may be directly determined as performance metrics, such as time, resulting in evaluation results that do not accurately reflect the DPU's capabilities.

The problem definition of DPUBench aims to determine the construction of the benchmark suite from an application perspective, specifically for network applications. In this regard, network-related metrics such as latency and throughput are selected as the evaluation metrics of DPUBench, and the throughput acceleration ratio is chosen as the performance metric, along with the CPU utilization ratio. Other approaches for problem definition in the problem space for conducting a DPU benchmark include determining the construction of the benchmark suite from an architecture perspective, a simulation perspective, and a real object perspective, among others.

However, due to the rapidly evolving nature of DPU architecture and products, adopting an application-driven benchmark construction as the problem definition of DPUBench methodology is more comprehensive, clear, and easy to expand and update while maintaining the authenticity and effectiveness of DPU evaluation. Since DPU is used to offload network applications from the CPU in data centers, developing a benchmark suite with a focus on network applications provides a stable platform for DPU evaluation, as network application development is in a relatively stable stage compared to the rapidly iterating DPU architecture.

3.2. Problem instantiation

After the problem definition, the next step in constructing DPUBench is problem instantiation. This involves concretizing the defined problem within a certain scope, thereby transforming research from abstract theory into concrete practice. Different researchers may approach the same defined problem from different perspectives and research different aspects of it. Even the same research team may have different understandings of the problem at different stages of research, resulting in differences in the research focus. Problem instantiation serves to unify the specific boundaries of the research problem after the problem definition and before the solution instantiation. This makes subsequent research solutions more standardized and unified, with a clear methodology roadmap.

The problem instantiation of DPUBench involves selecting network, storage, and security as typical network application scenarios and implementing DPUBench based on these three scenarios. These scenarios are chosen based on previous work [7–10,12–15,17–19], which identifies them as common representative scenarios for DPU at the current stage of offloading network applications from CPU in data centers.

By selecting these three scenarios, DPUBench covers different aspects of network applications. The network scenario covers various network transmission protocols, the storage scenario includes compression and decompression algorithms as well as storage protocols, and the security scenario encompasses various encryption and decryption algorithms. As a result, DPU evaluation with DPUBench is more comprehensive.

To maintain focus on the network applications, we do not select AI or other computation-intensive scenarios as representative scenarios, as only a few DPUs [16,22] can assist with those scenarios at the current stage. However, as DPUs and data centers continue to develop, these scenarios may become representative scenarios for network applications in data centers, and we will add them in future versions of DPUBench.

3.3. Solution instantiation

We then do the solution instantiation and implement DPUBench. Solution instantiation is to solve the instantiated problems and provide the research outcomes. It is a critical step in scientific research as it enables the provision of tangible research outcomes, such as tools, products, and research papers. And in DPUBench, solution instantiation is one step of the methodology.

The solution instantiation of DPUBench involves the extraction and implementation of basic operators from network, storage, and security scenarios, which compose the DPUBench's Operator Set for DPU evaluation. We also implement end-to-end evaluation programs to conduct DPU evaluation in a real network environment. Operators represent the most common algorithms in these three scenarios, and their combination can construct typical programs in each scenario. The end-to-end evaluation programs simulate the business of a real data center machine and evaluate the performance of DPU in a real network environment through communication between the Client and Server. We do not include a separate application set in DPUBench because the main execution part of application programs can be implemented with operators combination, and their evaluation cost is higher compared to operators, as well as their evaluation results are less reliable and effective compared to end-to-end evaluation programs.

Table 2 provides a brief summary of the methodology used in DPUBench. And the overview of DPUBench's methodology is shown in Fig. 1, which consists of problem definition, problem instantiation, and solution instantiation.

4. Operator set of DPUBench

Operator Set is a component of the solution instantiation in DPUBench, as mentioned in Section 3. In this section, we will outline the process of extracting the fundamental operators from network, storage, and security scenarios for DPUBench. We will then present the experimental results of the Operator Set, which include validating its representativeness and coverage, as well as evaluating the NVIDIA BlueField-2 using the micro-benchmarks of Operator Set. Based on these evaluation results, we will provide optimization recommendations for utilizing DPU effectively.

4.1. The extraction of operator set of DPUBench

We initially establish two rules for extracting the operators in DPUBench, and then conclude the typical programs and protocols in network, storage, and security scenarios based on previous work [7–10,12–19] in Table 3 to comply with **Rule1**. Additionally, upon breaking down these programs, we observe that certain processes, such as the three-way handshake in TCP/IP protocol [23] and the establishment of a secure initial key in an IPSec session [24], are executed only once during program initialization and have a relatively small proportion of execution time. Therefore, we do not extract operators from these processes to ensure representativeness. Instead, we decompose the most time-consuming and frequently executed processes within these typical programs to derive the operators for DPUBench based on **Rule2**. The two rules for extracting DPUBench operators are defined as follows.

Rule1. Operators should be integral components of typical programs on the network applications DPU has offloaded.

Rule2. The combinations of operators should constitute the primary execution portion of the typical programs on the network applications DPU has offloaded.

Table 2
The brief summary of DPUBench's methodology. We construct DPUBench from the perspectives of the problem definition, the problem instantiation, the solution instantiation.

Benchmark suite	Problem definition	Problem instantiation	Solution instantiation	
DPUBench	Network Applications	Network Scenario Storage Scenario Security Scenario	Operator Set	End-to-end Evaluation Programs

Table 3
The typical programs and protocols in network, storage, and security scenarios.

Network	TCP/IP [23], RDMA [25], OVS [26]
Storage	VirtIO-Blk [27], NVMe-Of [28]
Security	OpenSSL [29], IPSec [24], IDS

Table 4
The Operator Set of DPUBench.

Network	LPM, TCPSeg, IPSeg, CheckSum, CRC, Toeplitz
Storage	LZ77, Huffman, Snappy, CheckSum
Security	RSA, AES, DSA, ECDSA, MD5, SHA256, LPM, RXPMatch

4.1.1. Operators extraction in network scenario

From Table 3, we start with decomposing the data packet processing of the TCP/IP protocol [23] in network scenario, as it serves as the fundamental protocol used in networking. As illustrated in Fig. 3, the data packet processing of the TCP protocol is decomposed into three parts: TCP fragmentation, TCP checksum, and data copying. Similarly, the data packet processing of the IP protocol is decomposed into four parts: IP fragmentation, IP route lookup, IP checksum, and data copying.

TCP/IP protocol defines a maximum length for data packets to ensure efficient transmission in a network [23]. Therefore, the first step in transmitting a data packet is to perform packet fragmentation, which divides the packet into smaller fragments that do not exceed the maximum length specified in the protocol. From this process, we extract the TCPSeg and IPSeg operators. And in the IP protocol, when processing a data packet, it needs to perform a route lookup in the route table to determine the destination IP address and update the route table accordingly. For this operation, we extract the Longest Prefix Match (LPM) operator. To ensure the integrity of transmitted data packets, TCP/IP protocol uses the Checksum algorithm for verification when the receiving node in the data center receives the data packet. From this process, we extract the CheckSum operator. Since TCP/IP protocol programs run in the kernel space, data packets that need to be transferred typically undergo at least one data copying process. From this operation, we extract the MemCpy operator. However, please note that the MemCpy operator is currently under development and some bugs still need to be fixed.

In the Ethernet protocol, we focus on the data packet reception processing and decompose it into several key operations. As shown in Fig. 4, we extract LPM, CRC and Toeplitz operators from those operations. The Longest Prefix Match (LPM) operator is used for ARP MAC address resolution, which involves looking up the MAC address in the ARP table based on the destination IP address. The Cyclic Redundancy Check(CRC) operator is used for error detection and verification of the received data packet, as well as the Toeplitz operator used for performing a hash map for Receive Side Scaling (RSS) core selection in a multi-core processor.

The operators in the RDMA protocol [25] and Open vSwitch (OVS) protocol [26] are encompassed by the extracted operators in the previous network scenario. All the extracted operators in the network scenario of DPUBench are summarized in Table 4.

4.1.2. Operators extraction in storage scenario

In storage scenario, we focus on the data packet processing in storage protocols such as VirtIO-Blk [27] and NVMe-OF [28]). As shown in Fig. 5, we extract LZ77, Huffman, and Snappy operators. The Lempel–Ziv-77(LZ77) operator is based on a lossless data compression algorithm that achieves compression by replacing repeated occurrences of data with references to a dictionary [30]. The Huffman operator is based on the Huffman coding algorithm [31], which is a variable-length prefix coding technique used for lossless data compression. And the

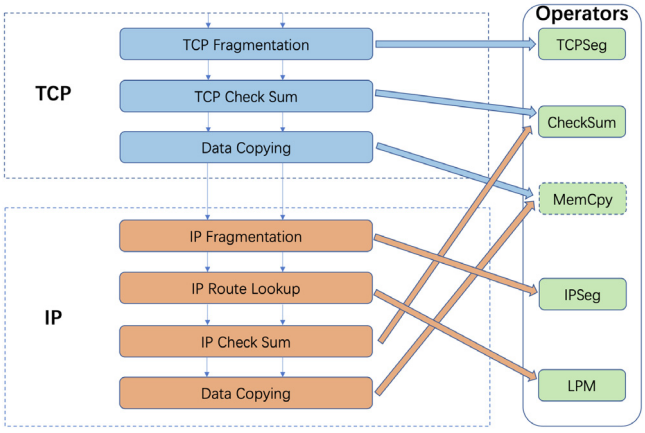

Fig. 3. The operators extracted in TCP/IP protocol.

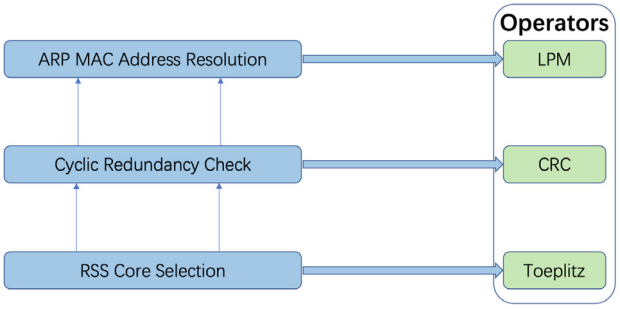

Fig. 4. The operators extracted in Ethernet protocol.

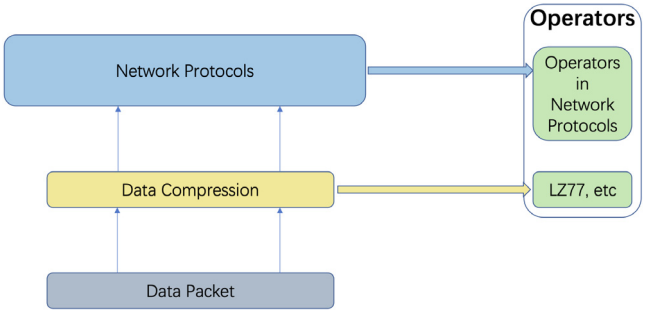

Fig. 5. The operators extracted in Storage protocols.

Table 5
The validation of Rule1 and Rule2 for operator set in DPUBench.

Programs	Scenarios	Operators
L3fwd	Network	LPM, CheckSum, CRC, IPSeg, Toeplitz
IPSec	Network & Security	LPM, RSA, CheckSum, CRC
File-Compress	Storage	Compress
File-Integrity	Network & Security & Storage	SHA256, SHA1, MD5
IPS	Security	RXPMatch, TCPSeg
Url-Filter	Security	RXPMatch

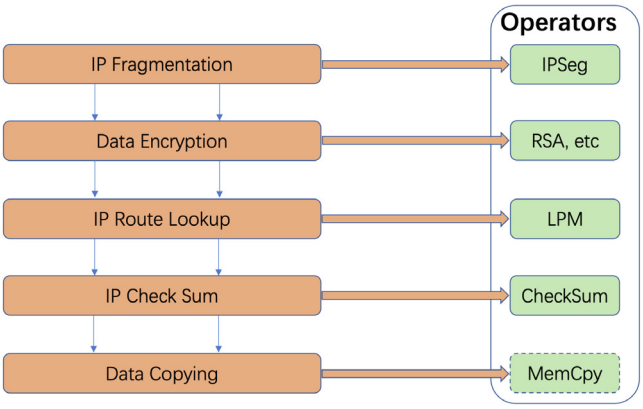

Fig. 6. The operators extracted in IPSec protocol.

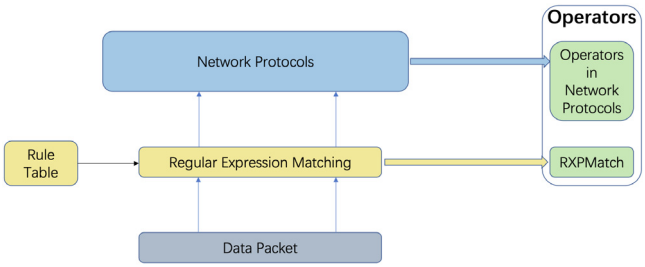

Fig. 7. The operators extracted in DPI.

Snappy operator is based on a fast, block-based compression algorithm that aims to provide high compression and decompression speeds with reasonable compression ratios.

These compression operators are commonly used in storage scenarios to compress data packets before transmission to effectively utilize network bandwidth. The detailed network protocols and their operators are discussed in Section 4.1.1, while in the storage scenario, we primarily focus on extracting compression operators. All the operators of DPUBench in the storage scenario are summarized in Table 4.

4.1.3. Operators extraction in security scenario

In a security scenario, programs can be primarily classified into network protocols (such as OpenSSL [29] and IPSec [24]) for ensuring data security during transmission, as well as application programs (e.g., Firewall) based on Deep Packet Inspection (DPI). We decompose the IPSec protocol [24] in Fig. 6 and extract four additional encryption operators, in addition to the network operators extracted in Section 4.1.1. We implement RAS [32], AES [33], DSA, and ECDSA [34] operators for data compression. The operators in OpenSSL [29] are already included in the operators extracted from IPSec [24].

The decomposition of DPI is illustrated in Fig. 7. In the network context, data packets undergo regular expression matching before transmission. The outcome of the regular expression matching determines whether the data packet is transmitted over the network. In addition to the operators extracted in the network scenario, we include the RXPMatch operator for performing regular expression matching. Table 4 presents all the operators implemented in DPUBench.

4.2. The experiments of operator set of DPUBench

4.2.1. Experimental configurations

The experiments are conducted on two platforms: the Intel Xeon E5-2620 v3 CPU (with 2 processors), which has 10 GB of memory and runs Ubuntu 18.04 OS, and the NVIDIA BlueField-2 DPU, which has 16 GB of memory and runs Ubuntu 20.04 OS. The development and profiling tools used in the experiments are DPDK (version 20.11.3.1.18) and DOCA (version 1.2.1). Each experiment is repeated more than three times, and the average values are reported for analysis.

4.2.2. Validate the representativeness, diversity and coverage of the operator set of DPUBench

To evaluate the representativeness, diversity, and coverage of the workload characteristics of Operator Set in DPUBench, we have selected 6 real workloads from the network, storage, and security scenarios for comparison. These selected workloads are representative application programs that are primarily offloaded by the DPU or essential components in real network applications. Each workload can be implemented using the combination of operators in DPUBench. The detailed information on these workloads, along with their corresponding operators, is presented in Table 5. This validation process also confirms the effectiveness of the two rules (Rule1 and Rule2) for extracting the representative operators, as discussed in Section 4.1.

The radar charts presented in Fig. 8 illustrate seven workload characteristics of the operators in DPUBench: IPC, iTLB-Miss-Ratio, dTLB-Miss-Ratio, L1D-Cache-Miss-Ratio, Integer instruction ratio, Branch instruction ratio, and Load&Store instruction ratio. The shapes of these radar charts demonstrate the diversity of workload characteristics covered by the Operator Set in DPUBench.

In Fig. 9, we compare the coverage of workload characteristics between the set of real workloads and the Operator Set in DPUBench. The radar charts representing the real workloads show that most of their workload characteristics can be effectively covered by the composed operators in DPUBench. This comparison validates the capability of the Operator Set in capturing the workload characteristics of real-world applications.

In addition to the radar charts, we have employed Principal Component Analysis (PCA) [35] to compare the diversity and coverage of workload characteristics between the Operator Set in DPUBench and the set of real workloads. The results are presented in Fig. 10. For visualization purposes, we have selected the top four principal components that contribute the most to the variance, accounting for 84.8% of the total variance contribution.

The area covered by the principal components of the Operator Set in DPUBench is capable of encompassing the area covered by the principal

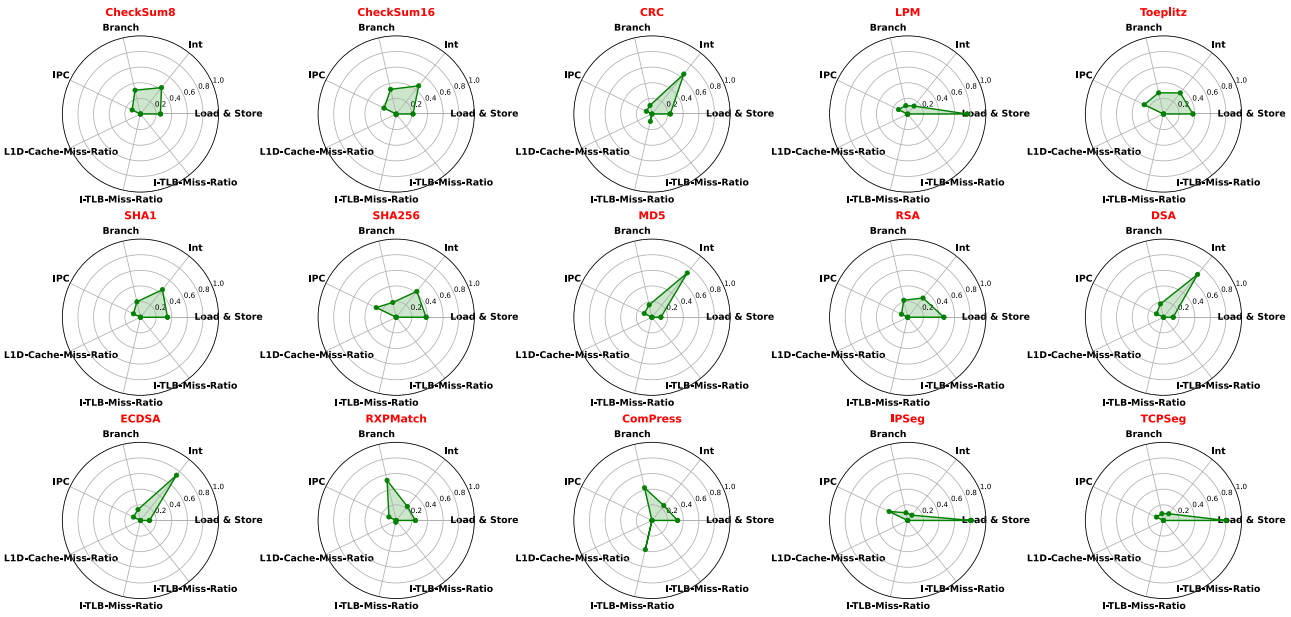

Fig. 8. The coverage experiment results of Operator Set under single thread 64B packet size configuration.

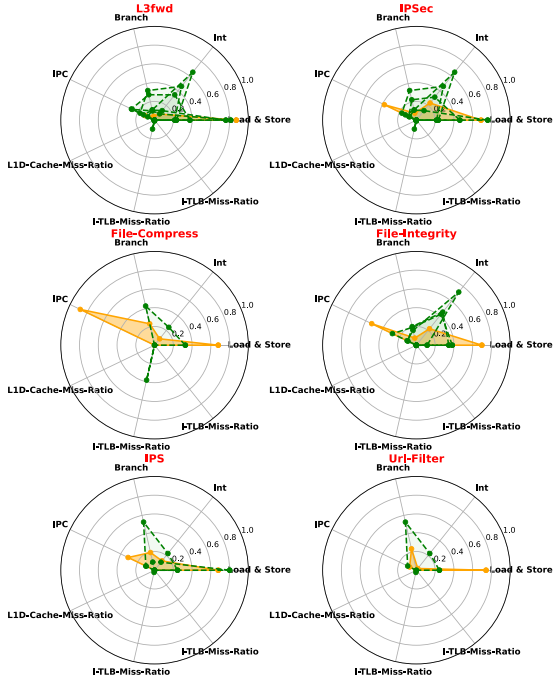

Fig. 9. Compare the coverage of operators with real application programs for validation. The composed operators for each real application program are shown in Table 5.

components of the real workloads, and it covers a significant portion of that area as well.

In conclusion, all of the experimental results consistently demonstrate that the Operator Set in DPUBench provides better coverage of workload characteristics compared to real workloads. Furthermore, the workload characteristics covered by the Operator Set in DPUBench exhibit a diverse range in terms of IPC, iTLB-Miss-Ratio, dTLB-Miss-Ratio, L1D-Cache-Miss-Ratio, Integer instruction ratio, Branch instruction ratio, and Load&Store instruction ratio.

4.2.3. Evaluate the NVIDIA BlueField-2 using operator set of DPUBench

We have conducted a performance evaluation on the NVIDIA BlueField-2 DPU, specifically the BlueField-2 model with encryption disabled and 25GbE capability. To ensure the micro-benchmarks, which are based on the operators in DPUBench, are representative, we have implemented them using general optimizations commonly used in real DPU applications. These optimizations include multi-threading, resource pooling, and cache-line alignment.

The micro-benchmarks can be configured with different input data sizes and number of threads. In our experiments, we have used input data sizes of 64B, 128B, 256B, 512B, and 1024B, which facilitates cache-line alignment. The number of threads can be configured as 1, 2, 4, 6, 8, 12, or 16. It is worth noting that the number of threads set to 16 exceeds the number of physical cores on the CPU (12 cores) and the number of ARM cores on the NVIDIA BlueField-2 (8 cores). The results of the performance evaluation are presented in Fig. 11. Each sub-figure correspond to the experiment result for one operator in the Operator Set, and each line in the figure shows micro-benchmark throughput for the different number of threads. Different lines in the same sub-graph correspond to different input data sizes.

The throughput of 10 operators, such as CRC, Checksum, toeplitz, RSA, DSA, ECDSA, AES, SHA256, SHA1, and MD5, exhibits linear scalability with the number of threads and is constrained by the number of physical cores. In these cases, the throughput increases proportionally with the number of threads, and the limiting factor is the number of available cores. Additionally, the throughput of these operators shows a fluctuation of approximately 10% to 20% when varying the input data size. This level of fluctuation is within the normal range, considering that the experiments are conducted under the same thread number and input data configuration, and the observed throughput variations fall within a consistent range.

The throughput of operators LPM, TCPSeg, and IPSeg demonstrates linear scaling with the input data size. This behavior can be attributed to the fact that these operators process a fixed amount of data within the input data. Consequently, in real-world applications, we can reduce the workload by encapsulating the data into fewer packets.

The Compress and RXPMatch operators are implemented using DOCA and deployed on the dedicated hardware accelerator of the BlueField-2. Their performance is not constrained by the number of physical cores available on the BlueField-2. Hence, when deploying these operators on the BlueField-2, it is possible to utilize more threads

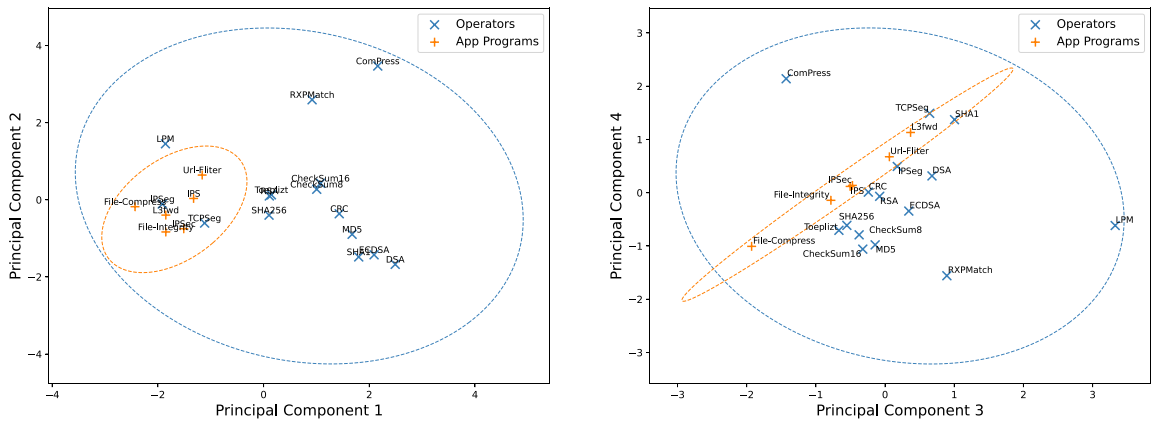

Fig. 10. PCA visualization results of Operator Set in DPUBench and typical Application Programs.

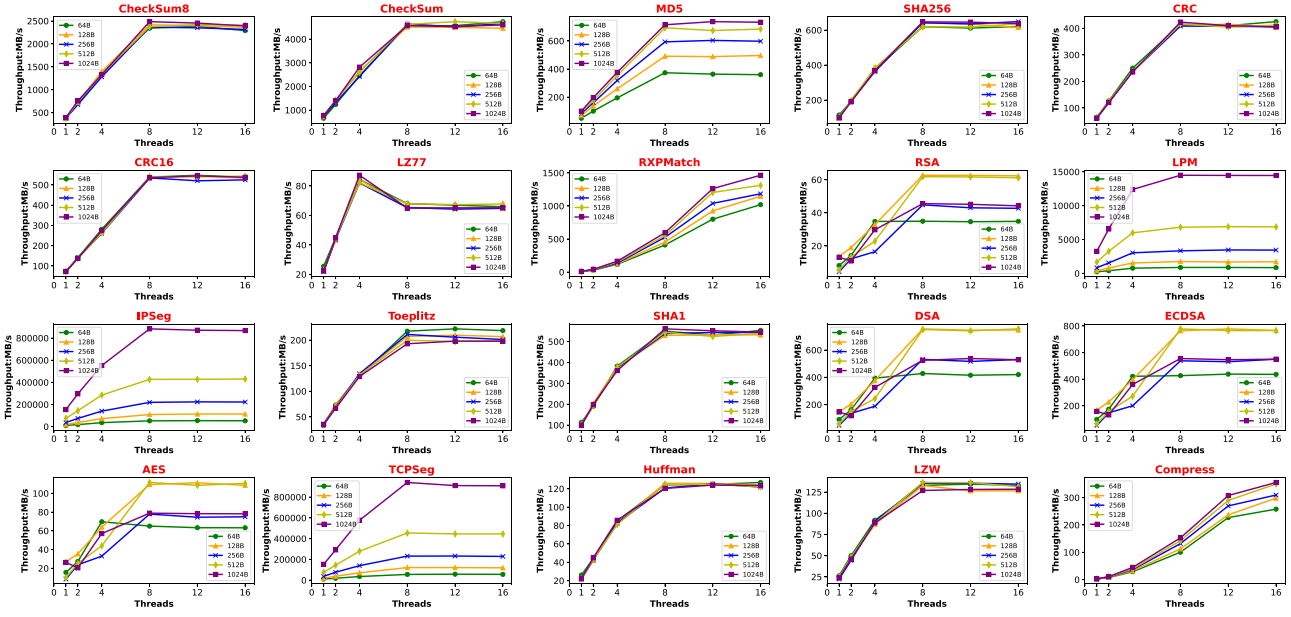

Fig. 11. The throughput of NVIDIA BlueField-2 using Operator Set in DPUBench under different data packet size and threads.

than the number of physical cores to further enhance their performance.

The comparison of throughput ratios between deploying micro-benchmarks on the NVIDIA BlueField-2 and Intel CPU is depicted in Fig. 12. For the majority of operators, the throughput is lower when deployed on the BlueField-2 compared to the Intel CPU. However, two exceptions are observed for the RXPMatch and Compress operators. These operators exhibit peak performance that can be 1.5 to 2 times higher when deployed on the BlueField-2 than on the Intel CPU. Consequently, applications such as IPS and Url-Filter (which utilize the RXPMatch operator) and NVMe-oF and File-Compress (which utilize the Compress operator) are more suitable for deployment on the BlueField-2 rather than the Intel CPU based on our experimental findings.

5. End-to-end evaluation programs of DPUBench

End-to-end Evaluation Programs are the other component of the solution instantiation in DPUBench, as mentioned in Section 3. In this section, we will outline the framework of End-to-end Evaluation Programs in DPUBench, which consists of both Client and Server. We will then provide the workloads of End-to-end Evaluation Programs in DPUBench and use them to do an evaluation of NVIDIA BlueField-2. Finally, We will present the experimental results of the End-to-end Evaluation Programs, which show the advantages of DPU in data centers.

5.1. The framework of end-to-end evaluation programs in DPUBench

Fig. 13 shows the framework of End-to-end Evaluation Programs in DPUBench, consisting of a Client with a dataset and traffic generator and a Server with applications that DPU can offload and cannot offload. Client device only contains CPU and is used for generating and sending data to the network. The server device can be a node that only contains a CPU or a node contains both CPU and DPU in data centers. The server receives and processes data packets sent by the Client, and transmits the results to the Client through the network.

The dataset in Client is used to generate the data and the traffic generator is used to send data packets with specified traffic according to network protocols. The network applications in Server are the application programs that DPU can offload and implement by DOCA SDK and DPDK SDK for DPU and programs for CPU. The applications in

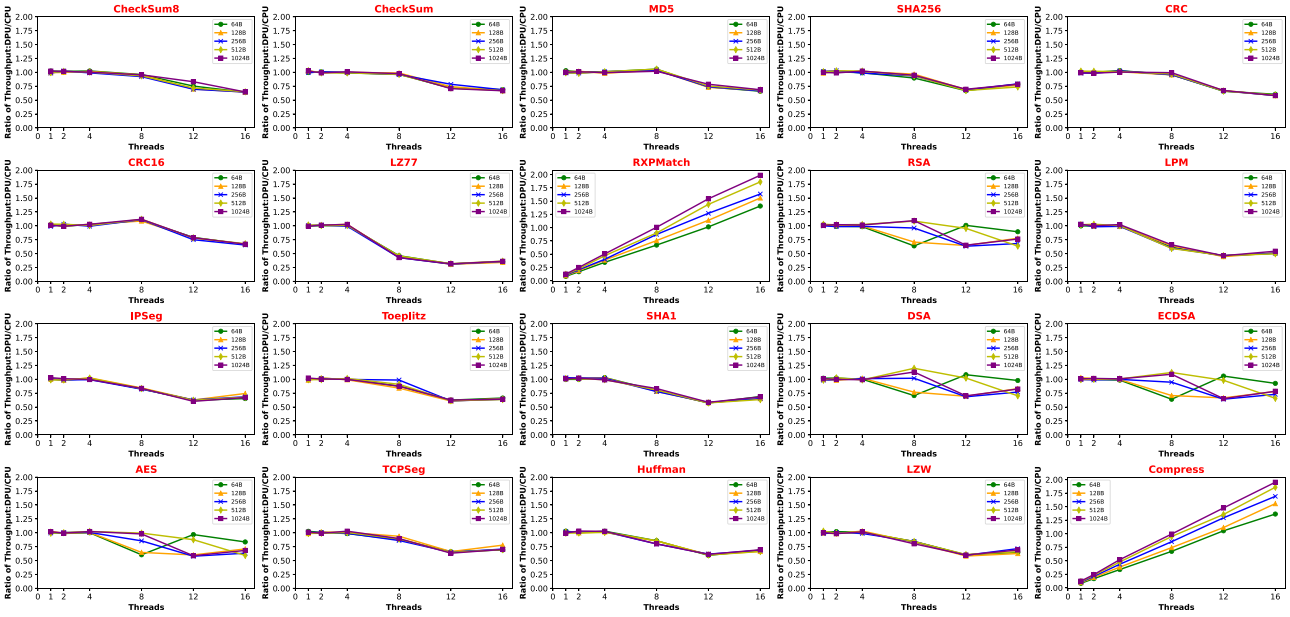

Fig. 12. The ratio of throughput of NVIDIA BlueField-2 and Intel CPU using Operator Set in DPUBench under different data packet sizes and threads.

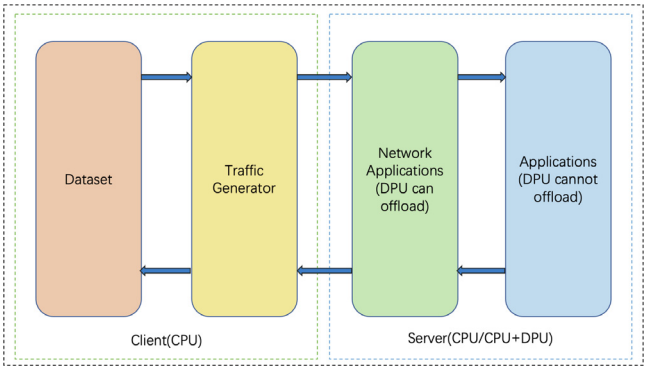

Fig. 13. The framework of end-to-end evaluation programs in DPUBench.

the Server are application programs that DPU cannot offload and are executed by the CPU.

5.1.1. The workloads

We have implemented two end-to-end DPU workloads in DPUBench. The first workload focuses on evaluating the performance of offloading the flow table to the DPU. This process is crucial as it enables the implementation of various network applications such as Packet Filters, Quality of Service, and Load Balancing. However, it should be noted that the first workload does not encompass the complete packet processing procedure found in real-world applications.

To address this limitation, we have developed a second end-to-end DPU workload that closely resembles a real application structure. By evaluating this workload, we gain deeper insights and uncover additional information that may remain hidden when conducting experiments solely on the first workload.

5.2. The experiments of end-to-end evaluation programs of DPUBench

The structure of the two end-to-end DPU workloads is based on Fig. 13. In the first workload, we utilize pktgen, which is available in the released version of the Linux kernel, as the traffic generator. And the dataset consists of randomly generated data by pktgen. The logic of the network application is straightforward: when packets arrive at the Server node, the Server searches the entries in the flow table based on the 5-tuple information (source IP, destination IP, destination port IP, source MAC, destination MAC) extracted from the packet header. If the tuple matches an existing entry, the count value of that entry is incremented. Otherwise, a new entry is added to the flow table. It is important to note that in this workload, the flow table is typically populated solely from the application layer, and therefore, the workload does not encompass the application part illustrated in Fig. 13.

The second workload encompasses both the network application and application parts depicted in Fig. 13. The network application in this workload focuses on application recognition, which involves inspecting the payload of received packets to determine if they contain specific character strings based on regular expression matches. Depending on the recognition results, the application performs different tasks. These tasks include calculating the hash value of the entire packet, compressing the payload section of the packet, or directly sending the packet back to the Client. By incorporating both the network application and application parts, this workload can provide a complete packet processing procedure compared to the first workload.

5.2.1. Experimental configurations

The Server is the same as the configurations mentioned in Section 4.2.1 and we just introduce the Client. We deploy the experiments on the Intel Xeon E5-2620 v3 CPU (4 processors) equipped with 10 GB of memory for the Client. The OS is Ubuntu 20.04 and the profiling tool is DPDK (version 20.11.3.1.18). We also repeat each experiment more than three times and report the average values.

5.2.2. Evaluate the NVIDIA BlueField-2 using end-to-end evaluation programs of DPUBench

In Fig. 14, the throughput and packet loss ratio are depicted for different time intervals between packets sent by the Client. The network application is accomplished in two ways: hardware and software. The hardware version implementation uses BlueField's flow table offloading capacity and all of the flow table operations are offloaded to BlueField, while in the software implemented version, the flow table and

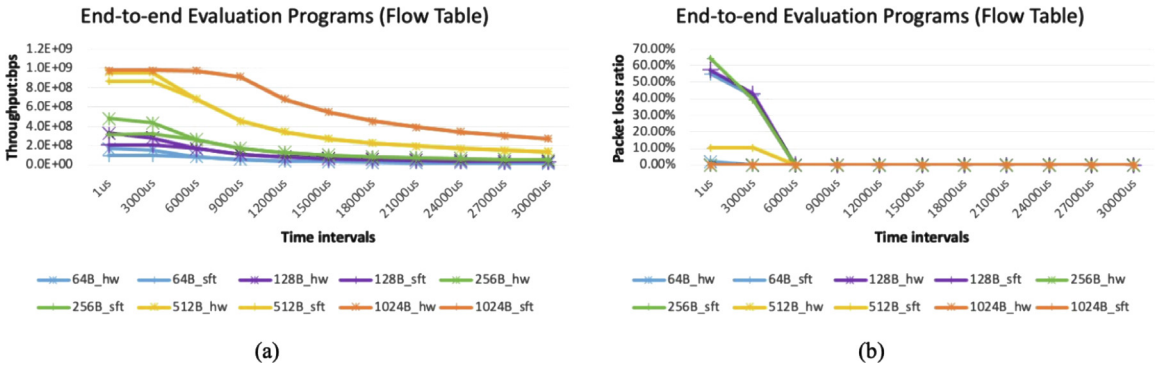

Fig. 14. The throughput and packet loss ratio of NVIDIA BlueField-2 using the first end-to-end DPU workload (flow-table) in DPUBench under different data packet sizes and time intervals. Fig. 14(a) corresponds to experiment results for throughput, Fig. 14(b) shows experimental results for packet loss rate. 64B_sft means software-implemented flow table under 64 Byte packet size, and 64B_hw means hardware-implemented flow table under 64B packet size.

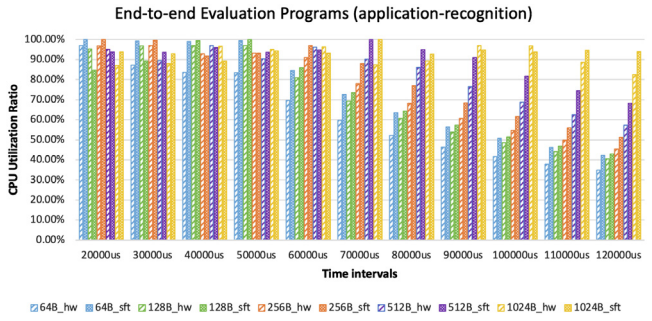

Fig. 15. The Server CPU utilization ratio for the second end-to-end DPU workload (application-recognition) in DPUBench under different data packet size and time intervals.

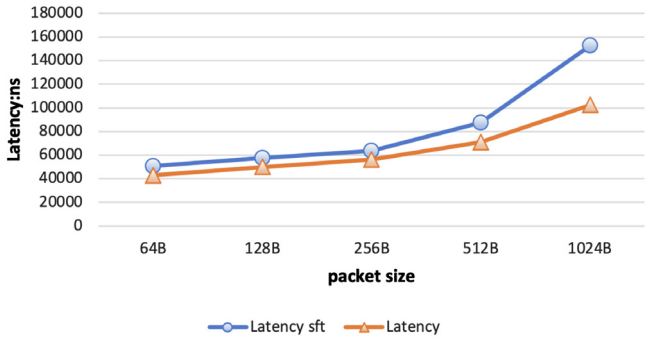

Fig. 16. The latency of the second end-to-end DPU workload (application recognition) in DPUBench under different data packet size.

all of the flow table operations are implemented by the C++ codes. For ease of comparison, the results for both software and hardware implementations are presented in the same sub-figure. Additionally, the experiments are conducted with different packet sizes sent by the Client, and a different color in the sub-figure represents each packet size.

The experiment results demonstrate that offloading the flow table to the DPU can yield greater throughput improvement when the packet size sent by the Client is small and when the Client sends packets at a fast speed (with low time intervals). This can be attributed to the fact that with larger packets, the Client's ability to send packets at maximum speed is limited by the network port's bandwidth. In such cases, the data processing speed does not become a bottleneck, and therefore, the DPU's ability to improve the system's throughput is limited. Conversely, when the Client sends packets at high speed that exceeds the Server's processing capacity, packet loss occurs. Offloading the flow table to the DPU can provide greater throughput improvement and reduce the packet loss ratio in such situations since the DPU can process packets at a faster speed.

Fig. 15 illustrates the Server CPU utilization ratio when the Client sends packets of different packet sizes at varying time intervals. For ease of comparison, the experiment results are presented in two ways: offloading the application recognition to the DPU (NVIDIA BlueField-2) and deploying the application on the Server CPU, depicted in the same sub-figure. The results indicate that when the Client sends packets at a slow speed, although offloading the application to the DPU does not improve the system's throughput, it can reduce the Server CPU utilization ratio by 10% to 20%.

Fig. 16 presents the results of experiments conducted to measure the reduction in process delay achieved by deploying the app-recognition workload on NVIDIA BlueField-2. The results indicate that offloading the application recognition to the DPU can result in a decrease of 10% to 15% in process delay. However, this reduction ratio is lower compared to the results obtained for the RXPMatch operator. This is because the packet processing procedure in the app-recognition workload involves both processing on the Server CPU and on the DPU (NVIDIA BlueField-2), and the DPU can only accelerate the latter part of the processing procedure.

6. Related work

Modern computer chips can be broadly categorized into two types: general-purpose processors (CPU) and specialized processors designed for specific acceleration tasks, including GPU, TPU [4], and DPU. As the performance gains predicted by Moore's Law [36] and Dennard's Scaling Law [3] have been slowing down, CPU performance improvement is also slowing down. This poses a challenge in meeting the increasing demand for computing resources in emerging fields like artificial intelligence, big data, and the Internet of Things. To address this challenge, specialized processors optimized for specific acceleration tasks are being developed. For instance, as the field of artificial intelligence has grown, AI models have significantly increased in size and complexity, with parameters ranging from 62 million in AlexNet [37] to 175 billion in GPT-3 [38] and beyond, with even larger models on the horizon.

As chip development progresses, research on evaluating various types of chips is also ongoing, with benchmarks being a key focus. CPU evaluation is supported by representative benchmarks such as SPEC CPU [39] provided by the Standard Performance Evaluation

Corporation (SPEC), which assesses single-core performance, and PARSEC [40] provided by Princeton University, which evaluates multi-core performance. The latest version of SPEC CPU is SPEC CPU2017 [41], which is the sixth iteration of the benchmark. Additionally, a new version of SPEC CPU, temporarily referred to as SPEC CPU v8 [42], is currently in development.

For evaluating AI chips, there are representative benchmarks such as MLPerf [43] and AIBench [44]. MLPerf focuses on selecting models from various AI tasks, including image classification, object detection, and machine translation, and constructs workloads for both training and inference evaluations [43]. AIBench, on the other hand, extracts 13 operators from typical AI scenarios and constructs micro-benchmarks using these operators [44]. These benchmarks provide standardized and comprehensive evaluation metrics for assessing the performance of AI chips in different AI tasks.

In the field of DPU evaluation, there are several existing studies and benchmarks. NVIDIA has developed RXPBench [12] using the DOCA SDK for their BlueField DPU, which currently focuses on regular expression matching as the evaluation program and measures the execution time on the chip. Amazon has conducted performance improvement tests in virtual machines by deploying the hypervisor on Nitro chips [13]. YUSUR has developed evaluation programs for their four DPU products [7–10] in different scenarios, such as using SQL queries for financial scenarios and measuring query latency [8]. Wei et al. [14] and Sun et al. [45] have evaluated the latency and throughput of NVIDIA BlueField-2 and provided optimization recommendations for DPUs with specific characteristics. These studies contribute to assessing and optimizing DPUs in specific architectures and scenarios.

As a new generation of programmable SmartNIC, the evaluation studies on SmartNIC can provide valuable insights for designing DPU benchmarks. Ibanez et al. [15] introduced the wire-to-wire latency metric to measure the time taken from receiving RPC requests to sending them over the network, using a SmartNIC placed in the network. Ma et al. [16] evaluated the performance of matrix multiplication and other operators in AI applications on a SmartNIC for distributed AI training, as well as the impact on AI model training time after deploying the SmartNIC in a distributed training framework. Mandal et al. [17] evaluated a SmartNIC for storage applications, focusing on the throughput of processing storage system read and write requests after connecting the SmartNIC to the network. Sabin et al. [18] evaluated a SmartNIC for security applications, measuring the throughput of processing encrypted communication requests in the corresponding scenarios. Bosshart et al. [19] offloaded a network layer protocol using a SmartNIC for SDN and evaluated the latency of communication with a data center node where the SmartNIC was deployed. These studies provide valuable performance metrics and insights that can inform the design of DPUBench.

7. Conclusion and plan

In conclusion, we have proposed DPUBench, an application-driven scalable benchmark suite for comprehensive DPU evaluation. DPUBench follows a methodology comprising problem definition, problem instantiation, and solution instantiation. We focus on network applications and select network, storage, and security as typical application scenarios. We extract essential operators from these scenarios and develop end-to-end evaluation programs, forming the Operator Set and Workload Programs of DPUBench. We present evaluation results of the NVIDIA BlueField-2 using DPUBench and provide optimization recommendations. DPUBench will be continuously maintained and updated to keep pace with DPU's development, and we will evaluate other DPUs in our future version of DPUBench. We will also investigate the IO virtualization application scenario in the future, as it plays a vital role in modern data centers.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] J. Zhan, A BenchCouncil view on benchmarking emerging and future computing, BenchCouncil Trans. Benchmarks, Stand. Eval. (2022) 100064.
[2] J. Shalf, The future of computing beyond Moore's law, Phil. Trans. R. Soc. A 378 (2166) (2020) 20190061.
[3] R.H. Dennard, F.H. Gaensslen, H.-N. Yu, V.L. Rideout, E. Bassous, A.R. LeBlanc, Design of ion-implanted MOSFET's with very small physical dimensions, IEEE J. Solid-State Circuits 9 (5) (1974) 256–268.
[4] N.P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, et al., In-datacenter performance analysis of a tensor processing unit, in: Proceedings of the 44th Annual International Symposium on Computer Architecture, 2017, pp. 1–12.
[5] The NVIDIA's definiton of DPU, https://resources.nvidia.com/en-us-accelerated-networking-resource-library/whats-a-dpu-data-product?lx=LbHvpR&topic=networking-cloud.
[6] NVIDIA BlueField-2, https://resources.nvidia.com/en-us-accelerated-networking-resource-library/bluefield-2-dpu-datasheet?lx=LbHvpR&topic=networking-cloud.
[7] YUSUR's DPU evaluation programs for cloud data center, https://www.yusur.tech/solution/cloudDataCenter.
[8] YUSUR's DPU evaluation programs for financial data calculation acceleration, http://www.yusur.tech/solution/financialDataCalculationAcceleration.
[9] YUSUR's DPU evaluation programs for high performance computing, https://www.yusur.tech/solution/highPerformenceComputing.
[10] YUSUR's DPU evaluation programs for industrial Internet, https://www.yusur.tech/solution/industrialInternet.
[11] The information of Intel Mount Evans, https://www.intel.com/content/www/us/en/newsroom/resources/press-kit-architecture-day-2021.html#gs.xbri9l.
[12] Doca document v1.5.1 :nvidia doca rxpbench user guide, https://docs.nvidia.com/doca/sdk/rxpbench/index.html.
[13] A. Liguori, The nitro project–next generation AWS infrastructure, in: Hot Chips: A Symposium on High Performance Chips, 2018.
[14] X. Wei, R. Chen, Y. Yang, R. Chen, H. Chen, A comprehensive study on off-path SmartNIC, 2022, arXiv preprint arXiv:2212.07868.
[15] S. Ibanez, A. Mallery, S. Arslan, T. Jepsen, M. Shahbaz, N. McKeown, C. Kim, The nanoPU: Redesigning the CPU-network interface to minimize RPC tail latency, 2020, arXiv preprint arXiv:2010.12114.
[16] R. Ma, E. Georganas, A. Heinecke, S. Gribok, A. Boutros, E. Nurvitadhi, FPGA-based AI smart NICs for scalable distributed AI training systems, IEEE Comput. Archit. Lett. 21 (2) (2022) 49–52.
[17] P.C. Mandal, N. Mariyappa, S. Das, A. Venkataraman, Storage Offload on SmartNICs.
[18] G. Sabin, M. Rashti, Security offload using the SmartNIC, A programmable 10 Gbps ethernet NIC, in: 2015 National Aerospace and Electronics Conference, NAECON, IEEE, 2015, pp. 273–276.
[19] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, M. Horowitz, Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN, ACM SIGCOMM Comput. Commun. Rev. 43 (4) (2013) 99–110.
[20] R. Recio, B. Metzler, P. Culley, J. Hilland, D. Garcia, A remote direct memory access protocol specification, Technical Report RFC 5040, October, 2007.
[21] G.F. Pfister, An introduction to the infiniband architecture, in: High Performance Mass Storage and Parallel I/O, Vol. 42, (617–632) 2001, p. 102.
[22] NVIDIA BlueField-3, https://resources.nvidia.com/en-us-accelerated-networking-resource-library/datasheet-nvidia-bluefield?lx=LbHvpR&topic=networking-cloud.
[23] G.R. Wright, W.R. Stevens, TCP/IP Illustrated, Volume 2 (Paperback): The Implementation, Addison-Wesley Professional, 1995.
[24] N. Doraswamy, D. Harkins, IPSec: The New Security Standard for the Internet, Intranets, and Virtual Private Networks, Prentice Hall Professional, 2003.
[25] C. Guo, H. Wu, Z. Deng, G. Soni, J. Ye, J. Padhye, M. Lipshteyn, RDMA over commodity ethernet at scale, in: Proceedings of the 2016 ACM SIGCOMM Conference, 2016, pp. 202–215.
[26] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, et al., The design and implementation of open vswitch, in: 12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15), 2015, pp. 117–130.
[27] R. Russell, Virtio: towards a de-facto standard for virtual I/O devices, Oper. Syst. Rev. 42 (5) (2008) 95–103.
[28] D. Minturn, Nvm express over fabrics, in: 11th Annual OpenFabrics International OFS Developers' Workshop, 2015.
[29] OpenSSL, https://www.openssl.org.

[30] J. Ziv, A. Lempel, A universal algorithm for sequential data compression, IEEE Trans. Inform. Theory 23 (3) (1977) 337–343.
[31] D.A. Huffman, A method for the construction of minimum-redundancy codes, Proc. IRE 40 (9) (1952) 1098–1101.
[32] W. Diffie, M.E. Hellman, New directions in cryptography, in: Democratizing Cryptography: The Work of Whitfield Diffie and Martin Hellman, 2022, pp. 365–390.
[33] D. Joan, R. Vincent, The design of Rijndael: AES-the advanced encryption standard, Inf. Secur. Cryptogr. (2002).
[34] D.B. Johnson, A.J. Menezes, Elliptic curve DSA (ECDSA): an enhanced DSA, in: Proceedings of the 7th Conference on USENIX Security Symposium, Vol. 7, 1998, pp. 13–23.
[35] I.T. Jolliffe, Principal Component Analysis for Special Types of Data, Springer, 2002.
[36] G.E. Moore, et al., Cramming More Components onto Integrated Circuits, McGraw-Hill New York, 1965.
[37] A. Krizhevsky, I. Sutskever, G. Hinton, Imagenet classification with deep convolutional networks, in: Proceedings of the 26th Annual Conference on Neural Information Processing Systems, NIPS, pp. 1106–1114.
[38] T. Brown, B. Mann, N. Ryder, M. Subbiah, J.D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al., Language models are few-shot learners, Adv. Neural Inf. Process. Syst. 33 (2020) 1877–1901.
[39] SPEC CPU, https://www.spec.org./benchmarks.html#cpu.
[40] PARSEC, https://parsec.cs.princeton.edu/index.htm.
[41] J. Bucek, K.-D. Lange, J. v. Kistowski, SPEC CPU2017: Next-generation compute benchmark, in: Companion of the 2018 ACM/SPEC International Conference on Performance Engineering, 2018, pp. 41–42.
[42] SPEC CPU v8, https://www.spec.org/cpuv8.
[43] V.J. Reddi, C. Cheng, D. Kanter, P. Mattson, G. Schmuelling, C.-J. Wu, B. Anderson, M. Breughe, M. Charlebois, W. Chou, et al., Mlperf inference benchmark, in: 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture, ISCA, IEEE, 2020, pp. 446–459.
[44] W. Gao, F. Tang, J. Zhan, X. Wen, L. Wang, Z. Cao, C. Lan, C. Luo, X. Liu, Z. Jiang, Aibench scenario: Scenario-distilling ai benchmarking, in: 2021 30th International Conference on Parallel Architectures and Compilation Techniques, PACT, IEEE, 2021, pp. 142–158.
[45] S. Sun, C. Huang, R. Zhang, L. Chen, Y. Huang, M. Yan, J. Wu, A comprehensive study on optimizing systems with data processing units, 2023, arXiv preprint arXiv:2301.06070.