Demystifying Datapath Accelerator Enhanced Off-path SmartNIC

Xuzheng Chen[†,1], Jie Zhang[†,1], Ting Fu[2], Yifan Shen[2], Shu Ma[2], Kun Qian[2], Lingjun Zhu[2], Chao Shi[2], Yin Zhang[*,1], Ming Liu[3], and Zeke Wang[*,1]
[1] Zhejiang University [2] Alibaba Cloud [3] University of Wisconsin–Madison

***Abstract*—Network speeds grow quickly in the modern cloud, so SmartNICs are introduced to offload network processing tasks, even application logic. However, typical multicore SmartNICs such as BlueFiled-2 are only capable of processing control-plane tasks with their embedded processors that have limited memory bandwidth and computing power. On the other hand, cloud applications evolve rapidly, such that a limited number of fixed hardware engines in a SmartNIC cannot satisfy the requirements of cloud applications. Therefore, SmartNIC programmers call for a programmable datapath accelerator (DPA) to process network traffic at line rate. However, no existing work has unveiled the performance characteristics of the existing DPA.**

To this end, we present the first architectural characterization of the latest DPA-enhanced BlueFiled-3 (BF3) SmartNIC. Our evaluation results indicate that BF3's DPA is significantly wimpier than the off-path Arm processor and the host CPU. However, we still identify that DPA has three unique architectural characteristics that unleash the performance potential of DPA. Specifically, we demonstrate how to take advantage of DPA's three architectural characteristics regarding computing, networking, and memory subsystems. Then we propose three important guidelines for programmers to fully unleash the potential of DPA. To demonstrate the effectiveness of our approach, we conduct detailed case studies regarding each guideline. Our case study on key-value aggregation achieves up to 4.3× higher throughput by using our guidelines to optimize memory combinations.

I. Introduction

In the modern cloud, network speeds grow quickly, with 100/200 Gigabit Ethernet (GbE) network interface controllers (NICs) widely deployed [1]–[3] and 400/800 GbE expected in the near future [4], [5]. At the same time, Moore's law is slowing down, and the gap between network and CPU speeds is rapidly increasing. More and more works intend to equip NICs with more computing power to alleviate the computing pressure of host servers.

These NICs with computing power are usually called SmartNICs, e.g., multicore system-on-chip (SoC) SmartNICs. SoC SmartNICs provide significantly more generality and have been widely used in various application domains [6]. Prior work [7] categorizes SoC SmartNIC into two types: on-path and off-path, according to how SmartNIC processors interact with network traffic. Processors of on-path SmartNICs sit on the packet path and can directly manipulate each incoming/outgoing packet. For off-path SmartNICs, incoming packets from the network are delivered to either host CPUs or off-path processors based on forwarding rules installed on the NIC switch, while both host CPUs and off-path processors can send out packets through the NIC TX port.

The processors in the SoC SmartNICs are usually Arm [6], [8], [9] or cnMIPS [10], and have fewer threads compared with the host CPU. This prevents the SmartNIC processors from processing network traffic at line rate. Current off-the-shelf SmartNICs solve this problem by equipping NICs with a few powerful hardware domain-specific engines, such as erasure coding, de/compression, and encryption engines [6], [8]–[11]. However, different cloud applications have different offloaded computation kernels, and fifty hot cloud applications occupy around 60% host CPU cycles [12]–[14], each hot application could need a customized engine to accelerate [15], [16], and thus Google even proposes the concept of "a sea of accelerators" for cloud applications. Even worse, hot cloud applications change over time [12], [17], [18]. Therefore, the current SoC SmartNICs that feature a few fixed hardware engines fail to meet the requirements of cloud applications.

This trend calls for a programmable datapath accelerator that can provide line-rate data processing ability while serving a broad range of applications. SmartNIC vendors are trying to integrate a programmable datapath accelerator in the SmartNIC datapath. Nvidia's latest BlueField-3 (BF3) [8] adds a many-core RISC-V processor in the datapath (called datapath accelerator, DPA). Even though plenty of recent works [7], [19]–[25] have characterized the traditional SmartNICs, there is no comprehensive study on DPA-enhanced SmartNICs. Therefore, it's still unclear for programmers to fully understand the performance characteristics of DPA, which could heavily interact with off-path processors and host CPUs.

To this end, this paper conducts the first systematic benchmark on characterizing the performance of DPA-enhanced SmartNIC, specifically the BlueField-3. We thoroughly evaluate the resources in a BF3-attached server including general-purpose computing power. To tap into the potential of DPA, we assess it from an architectural perspective instead of reporting performance numbers. Our evaluation results indicate that the current DPA is markedly underperforming than the host/Arm. However, we still identify that DPA has three unique architectural characteristics regarding computing, networking, and memory subsystems, that expose the potential to benefit certain types of applications. We demonstrate how to take advantage of DPA's three architectural characteristics to fully unleash the potential of the underperforming DPA by using three

[†]Equal contribution
[*]Corresponding Authors: {zhangyin98, wangzeke}@zju.edu.cn

979-8-3503-5171-2/24/$31.00 ©2024 IEEE

corresponding case studies. The experimental results show that the achievable throughput of the key-value aggregation service can be increased by up to 4.3× and the time uncertainty bound of the clock synchronization service can be decreased by up to 2.3×. Then we conclude three important DPA-related guidelines for future SmartNIC programmers regarding these architectural characteristics:

- **1. Offloading latency-sensitive and simple workloads.** Compared with the host/Arm, DPA is much closer to the network, since DPA and NIC are on the same chip. As such, DPA enjoys the lowest network latency. Latency-sensitive network applications can exploit this characteristic to improve end-to-end performance. Our case study on clock synchronization service achieves up to 2.3× lower time uncertainty bound. However, we cannot offload compute- and/or memory-bandwidth-bound logic to DPA due to its limited computing ability and memory bandwidth of a single DPA thread.
- **2. Offloading easy-to-parallelize workloads with small working set sizes.** Our benchmarking results reveal that offloading stateless network functions to DPA cores can achieve line rate as host/ARM, even though each DPA thread is significantly wimpier, because DPA has more cores than host/Arm. However, when the application's working set size exceeds DPA's cache size, significant performance downgradation would occur. Programmers can offload easy-to-parallelize workloads to exploit DPA's many-core parallelism. Also, the memory working set size should better fit in DPA's cache size to avoid significant degradation.
- **3. Carefully select memory buffers when running network workloads in DPA cores.** DPA can access not only its own memory but also the off-path Arm's memory and the host CPU's memory. Using different memories for the DPA cores can result in several times performance differences. Programmers must carefully choose different memories according to the specific usage. Our case study on key-value aggregation service achieves up to 4.3× higher throughput by optimizing memory combinations.

We acknowledge that our findings may not fully apply to the next generation of BlueField SmartNICs or products from other vendors. However, we believe that our methodology (i.e., studying the characteristics of each component and offloading workloads accordingly) can be applied to other SmartNICs. Our benchmark code and tools are available at https://github.com/RC4ML/BenchBF3.

II. Background

A. On-path/Off-path SmartNIC

SoC SmartNICs comprise a multicore processor (i.e., MIPS/ARM) and the processor is usually underperforming due to the cost, form factor, and power [26]. SoC SmartNIC has its own onboard SRAM/DRAM and usually has a DMA engine to access host server memory. A different SoC SmartNIC has a different set of domain-specific accelerators (e.g., encryption, regular expression matching, and erasure coding).

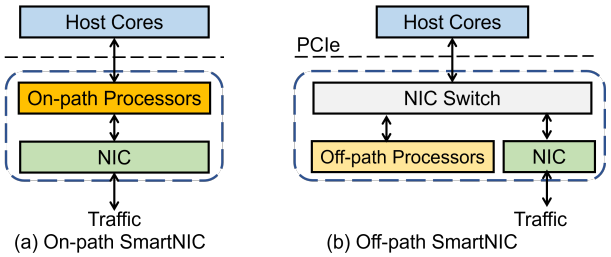


Figure 1: On-path and Off-path SmartNICs.

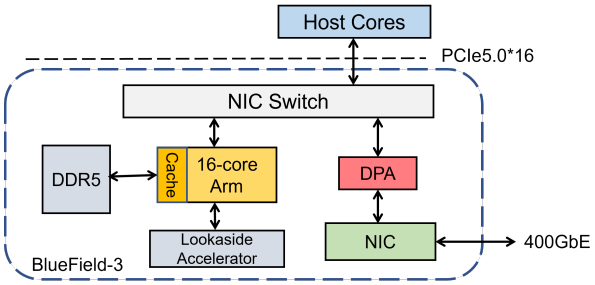


Figure 2: BlueField-3 SmartNIC architecture.

Depending on where the processor sits on the network path, SmartNICs can be categorized into two categories: On-path and Off-path. As shown in Figure 1a processors in on-path SmartNICs provide inline processing power for each incoming/outgoing packet. Figure 1b shows the schematic diagram of off-path SmartNICs, and each incoming packet can be forwarded to SmartNICs processor or host CPU. This kind of pattern is also called lookaside acceleration [27]. Processors of on-path SmartNICs are usually thought to be closer to the network than off-path SmartNICs due to the higher wire latency introduced by the NIC switch (∼500 ns in BF3).

B. DPA-enhanced BlueField-3 SmartNIC

Figure 2 demonstrates the overall architecture of BF3 SmartNIC. BF3 mainly consists of an off-path Arm processor, a many-core datapath RISC-V processor, an off-chip DDR5 memory, a PCIe switch, several domain-specific accelerators, and a NIC module.

Unlike the off-path Arm processor and domain-specific accelerators that provide look-aside computing power, the many-core DPA sits in the network critical path, providing inline processing ability. Unlike a naive on-path processor, DPA is designed to be able to tightly interact with the Arm-related resources and the host-related resources. Despite DPA's three-level cache and memory, DPA can access the host's last-level cache (LLC), host memory, Arm LLC, and Arm memory. The complicated cache/memory hierarchy makes it challenging for programmers to fully unlock the potential of the datapath accelerator-enhanced off-path SmartNIC.

C. Experiment Setup

We build a testbed and conduct various experiments on it to fully understand the performance characteristics of the abundant resources of DPA-enhanced BF3. The testbed comprises two servers running Ubuntu 22.04 (Linux kernel-5.15.102), each equipped with a BF3 SmartNIC. The two BF3

Table I: Hardware description of experiment setup

Component	Hardware description
Server	SuperServer SYS-421GE-TNRT
Host CPU	Intel Xeon Gold 6426Y
Host Memory	256 GB DDR5-4800 (8 out 8 channels)
NIC	1× BlueField-3 B3220(2 × 200Gbps)
NIC PCIe	PCIe 5.0 ×16

Table II: Hardware Specifications of compute resources

Resources	Host (X86)	Arm	DPA (RISC-V)
Processor	Intel Xeon	Cortex-A78AE	RV64IMAC
Cores	16	16	16
Threads	32	16	256
L1D Cache	48K×16	64K×16	1K×256
L1I Cache	32K×16	64K×16	1K×16
L2 Cache	1M×16	0.5M×16	1.5M×1
L3 Cache	37.5M×1	16M×1	3M×1
Frequency	2.5GHz	2.133GHz	1.8GHz

SmartNICs are connected back-to-back using two 200GbE QSFP56 cables. The detailed configuration of the servers is shown in Table I. In all network-related experiments, we use link aggregation [28] to combine two network interfaces into a single interface and use hash mode to distribute packets. DPA software uses NVIDIA DOCA framework [29] (v2.5.0) to process network packets. Currently, only 190 out of 256 DPA threads can be used concurrently due to the limitation of the DOCA driver. As such, we use at most 190 DPA threads in related experiments. The host/Arm software leverages DPDK [30] (v22.11) to process network packets and uses RSS [31] to distribute packets among different queues (one queue per core) unless stated otherwise.

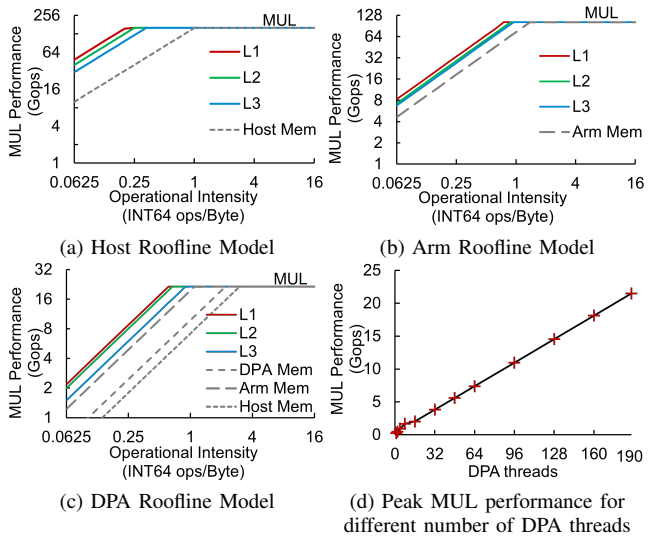
The detailed comparison of three kinds of compute resources is shown in Table II. The noticeable difference is that DPA has markedly more threads than the host/Arm, so it becomes necessary to benchmark the DPA-enhanced SmartNIC such that programmers can easily optimize their applications on top of our benchmarking hints.

Benchmarking Methodology:

- First, we benchmark the general-purpose computing power in a BF3-attached server regarding memory subsystem and computing ability (§ III).
- Second, we benchmark the networking ability of the three processors (§ IV).
- Third, we present three case studies to demonstrate our hints related to how to fully exploit the potentials of DPA-enhanced SmartNIC (§ V).

III. Benchmarking General-purpose Computing Power

In this section, we characterize the general-purpose computing power in a BF3-attached server, i.e., the three processors. Prior works [7], [19]–[22], [24] evaluate the SmartNIC processors from a high-level application perspective. To thoroughly understand the characteristics of these processors, we assess them from two architectural perspectives: 1) computing and 2) memory subsystem.



(a) Host Roofline Model (b) Arm Roofline Model

(c) DPA Roofline Model (d) Peak MUL performance for different number of DPA threads

Figure 3: Cache-aware Roofline Model for different general-purpose computing power.

A. Benchmarking Three Computing processors

In this section, we benchmark the three processors regarding their computing ability. To experimentally assess their efficiency, we conducted a series of tests following the methodology outlined in [32]. The bandwidth is analyzed by varying the number of memory operations to access different levels of the memory hierarchy, specifically by using contiguous memory and incrementally increasing the working set sizes to fit the capacities of L1, L2, L3 caches, and main memory. Figure 3a/ 3b/ 3c shows three processors' Cache-aware Roofline Model [32] for the "INT64 Multiplication" operations. We find that Arm can provide similar operations per second (Gops) comparable to that of the host CPU under the same core counts (16) and without hyper-threading. Although DPA has 256 threads (16 cores), its achievable Gops is 7.5× lower than the host CPU and 4.7× lower than the Arm. DPA's single-thread computing power is much lower than the host/Arm (up to 26× lower). Figure 3d uses the same testing method as previous experiments. It shows DPA's "INT64 Multiplication" performance under different numbers of DPA threads, demonstrating linear scalability.

Takeaways (Computing): Arm cores can serve as a powerful supplement to the host's computing capabilities. DPA's single-thread computing power is very low, and serial compute-intensive workloads should not be offloaded to DPA cores.

DPA's Unique Computing Characteristic: Unlike the host/Arm, DPA has a thread count (256) that is an order of magnitude higher. To fully leverage DPA's underperforming computing ability, the workloads should be very easy to parallelize. We further use a case study to explore this DPA's unique computing characteristics in § V-B.

B. Memory Subsystem

In this section, we evaluate the memory subsystem of the BF3-attached server. All three processors in the BF3-attached

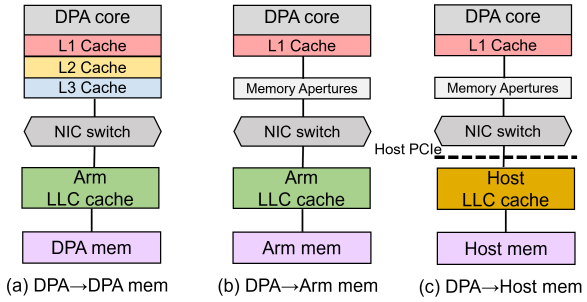


Figure 4: DPA accesses three memory types.

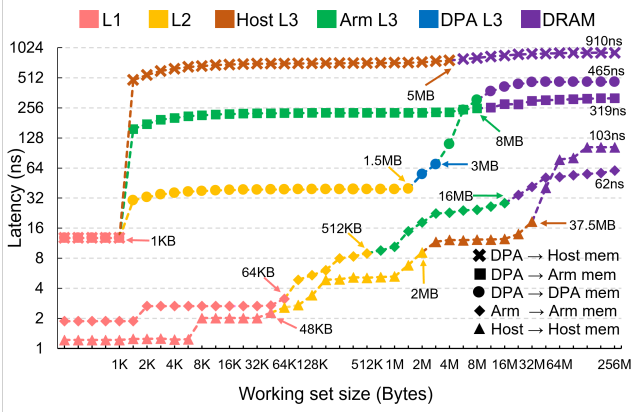


Figure 5: Cache latency for all computer resources.

server have their own three-level cache. Both X86 and Arm core have directly connected DDR memory.

DPA does not have directly connected DDR memory, instead, a 1-GB region in the Arm DDR is allocated exclusively for DPA access. DPA has to access this memory region (DPA memory) through the NIC switch which incurs high latency. DPA's access to the DPA memory would be cached with Arm's L3 cache and DPA's L1/L2/L3 cache. In addition to DPA memory, DPA can access the host memory and Arm memory through a memory aperture module [33] near the DPA core. This module converts a memory request into a PCIe transaction, thus allowing DPA cores to directly access Arm memory and host memory using a load/store instruction. It's worth noting that such accesses will only go through DPA's L1 cache and will not go through DPA's L2/L3 caches. Also, DPA's access to the host/Arm memory would go through the host L3 cache or Arm L3 cache. Figure 4 shows the physical paths of the DPA's access to these three kinds of memories.

1) Cache/Memory Latency: We first measure the memory/cache read latency. We use a pointer-chasing manner [34] and carefully vary the access stride and the working set size. Figure 5 shows the latencies of the supported five kinds of memory accesses. "X→Y mem" means X processor accesses Y memory. We have three observations.

First, The L1 cache latency of DPA is 10.5× of the host L1 latency. DPA's L2/L3 cache latency is also significantly higher than that of the host/Arm. The high cache latency of DPA indicates that carefully using DPA cores is extremely important for application offloading.

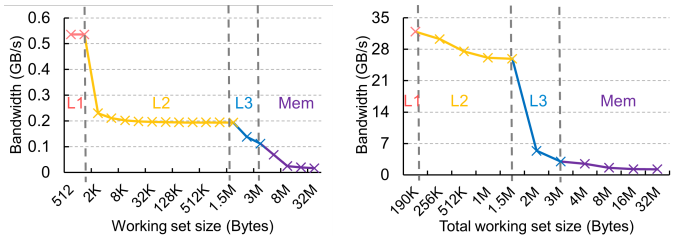


Figure 6: DPA random access DPA memory throughput.

Second, DPA has noticeably higher memory reading latency than Arm and the host. DPA's memory read latency is at least several times higher than the host/Arm. This can primarily be attributed to the high latency of the NIC switch and PCIe interconnect. Compared to the Arm core and host core's direct memory access, DPA's access to the DPA memory or Arm memory would involve an additional NIC switch. In addition to the NIC switch, DPA's access to host memory also involves another PCIe interconnect.

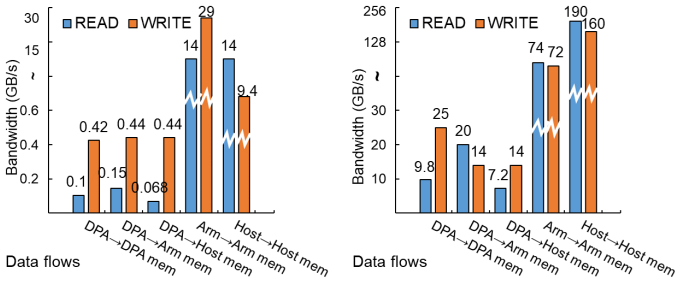
Third, although Arm memory and DPA memory are physically the same DDRs connected to the Arm core, the latency of DPA's access to the DPA memory is noticeably higher than DPA's access to the Arm memory. We suspect this is mainly because DPA's access to the host memory and Arm memory only go through DPA's L1 cache [33], while DPA's access to the DPA memory would additionally go through DPA's poor L2/L3 cache. The latency of "DPA→Host" is still higher than DPA accessing DPA memory although it bypasses the L2/L3 cache, due to the additional step of the host PCIe interconnect.

2) Cache Bandwidth: In this section, we measure DPA's random read bandwidth from a DPA memory buffer. We vary the memory buffer size (i.e., the working set size) and use different numbers of threads.

Figure 6a shows the read bandwidth of using a single DPA thread and Figure 6b shows the bandwidth of using all 190 DPA threads. We observe that the read bandwidth significantly drops when the working set exceeds DPA's L2 cache size (1.5 MB), the bandwidth loss can be up to 25×. This indicates that memory-intensive DPA applications must carefully manage their working set size.

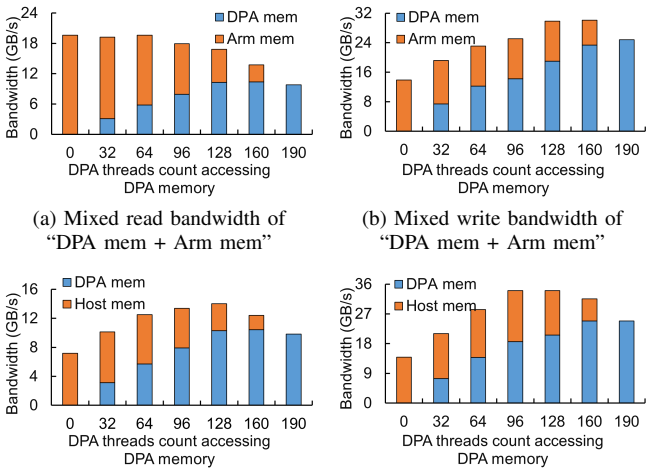
3) Memory Bandwidth: Figure 7 shows the per-thread and all-thread sequential read bandwidth of the mentioned five kinds of accesses. We let the working set be large enough that all reading requests are served by the memory instead of the cache. "X→Y mem" means X cores access Y memory. The all-threads bandwidth is measured by using all threads concurrently. We have two observations.

First, no matter access which memory, DPA has up to 205× lower per-thread read/write bandwidth than the host and Arm. Although DPA has 256 threads, the all-threads bandwidth is still up to 7.6× lower than Arm and the host. The all-thread bandwidth is also much lower than the network line rate (400 Gbps full-duplex). This implies that DPA can perform poorly in memory-intensive applications.



(a) Per-thread (b) All-threads

Figure 7: Achievable memory throughput.



(a) Mixed read bandwidth of "DPA mem + Arm mem"

(b) Mixed write bandwidth of "DPA mem + Arm mem"

(c) Mixed read bandwidth of "DPA mem + Host mem"

(d) Mixed write bandwidth of "DPA mem + Host mem"

Figure 8: Mixed memory throughput under 190 available DPA threads. We let different numbers of DPA threads access DPA memory and let all remaining DPA threads access the Host/Arm memory.

Second, Arm cores in BF3 have a comparable per-thread memory bandwidth to host cores. This suggests offloading memory-intensive workloads to BF3's Arm cores. The all-threads memory bandwidth of the host is 2.7× higher than the Arm because BF3's Arm only has two memory channels while the host features eight memory channels.

Mixed memory bandwidth of DPA cores. From Figure 7's DPA-related results, we can observe that the all-threads memory bandwidth for each memory is far lower than the product of per-thread bandwidth and the number of DPA threads. This indicates that the DPA's memory bandwidth is not bounded by the DPA thread count. This inspires us to explore whether we could use the spare DPA threads to access other different memories to further improve DPA's all-threads memory bandwidth.

Accordingly, we use a different number of DPA threads to access DPA memory and the remaining threads to access the host memory or Arm memory[1]. Figure 8 shows the mixed read/write bandwidth for two combinations: "DPA mem + Arm mem" and "DPA mem + Host mem". We observe that the write bandwidth of "DPA mem + Arm mem" and the read/write bandwidth of "DPA mem + Host mem" are all higher than only using a single type of memory. The maximum bandwidth improvement can be up to 2.4×. This indicates that memory-intensive workloads running in DPA cores should consider using multiple kinds of memories concurrently to maximize the achievable memory bandwidth.

Takeaways (Memory subsystem). DPA's cache/memory latency is much higher than the host/Arm. DPA's all-threads memory bandwidth is also significantly lower than the host/Arm. Unlike the host/Arm, DPA does not have a directly connected memory. DPA has to access DPA memory or Arm memory through the NIC switch, which can greatly limit the performance on memory-intensive workloads. As such, DPA applications' working set sizes should better fit in the DPA cache size. DPA can use multiple kinds of memories concurrently to improve achievable memory bandwidth. We further use a case study to explore the influence of DPA's unique memory characteristic in § V-C.

IV. Benchmakring Networking

In this section, we evaluate the networking ability of a BF3-attached server. The datapath accelerator enhanced BF3 offers various strategies to leverage its powerful networking ability. The host, DPA, and Arm cores can send/receive network packets. In this section, we use two back-to-back connected servers as mentioned in § II-C to evaluate the networking abilities of the BF3-attached server.

A. Effect of Network Buffer Location

We first examine the effects of network buffer location (either host/Arm memory or DPA memory) when using DPA core to process network traffic[2].

Host/Arm Memory. When DPA cores use the host/Arm memory as the network buffer, NIC can directly put the newly arrived packets into the host/Arm L3 cache. Regarding processing latency, letting DPA cores use the host/Arm memory as the network buffer is not a good choice, because NIC writing packets into the host/Arm L3 cache would travel through the high-latency NIC switch (and an additional PCIe interconnect if using host memory). We quantitatively analyze this impact on the network latency in § IV-B.

DPA Memory. When DPA cores use DPA memory as the network buffer, NIC can directly put the newly arrived packets into the DPA L2 or L3 cache.

To validate the effect of network buffer location, we conduct the following experiment. We first sequentially send a bunch of packets (e.g., 512 packets of 1KB each) to a DPA core that uses DPA memory as the network receives buffer. After a long enough period to make sure all packets have arrived, the DPA core reads the first packet of this bunch of packets and measures the read latency. Then we sequentially send another

[1]We do not let DPA access the host memory and Arm memory concurrently because the current DOCA framework does not allow this kind operation.

[2]We do not explore the host CPU/Arm's mechanism since there are plenty of direct cache access related researches [35]–[37] about X86/Arm.

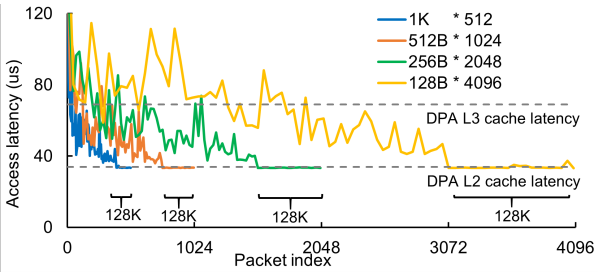


Figure 9: Average packet access latency under different combinations with a total of 512KB size. A higher index corresponds to a more recently received packet.

bunch of packets, and the DPA core measures the latency of the second packet. We repeat this process until we get the latency of the last packet. Note that we only measure one packet latency to prevent the effects of cache prefetching, in addition, we would pollute the DPA cache before the sending side sends a bunch of packets.

Figure 9 shows the access latency of each packet. We have two observations. First, the latest received 128 KB packets are guaranteed to be put in the DPA L2 cache. No matter the packet size, the access latency of the latest 128 KB packets is always DPA L2 latency, indicating that these packets are directly put into the L2 cache by the NIC.

Second, the newer received packet has a higher possibility of being put into the places that are closer to the DPA core, DPA L2/DPA L3/Arm L3/DPA memory from the closest to the farthest, respectively.

Takeaways (Working set size). When DPA is managing network traffic, the NIC can directly access the host L3 cache (host memory as network buffer), Arm L3 cache (Arm memory or DPA memory as network buffer), and DPA L2/L3 cache (DPA memory as network buffer). Therefore, programmers need to be aware of the working set size such that the working set stays at the expected cache and thus the memory traffic can be minimized.

B. Network Latency

In this section, we measure the network latency regarding different computing cores via an L2-reflector application, as shown in Figure 10. The client sends a packet to the server. The server swaps the source and destination MAC addresses of each packet and returns the packet to the client. The packet size is fixed to 1 KB. We deploy the client/server in different processors and measure the average round-trip time, and DPA uses different types of memories. We have two observations.

First, regarding different processor implementations, the DPA-based client and server offer the lowest latency while the host-based client and server offer the highest latency. This also aligns with our expectations that DPA enjoys the shortest distance from the network because DPA and NIC are in the same NIC chip. Arm experiences moderate network latency due to its additional NIC switch hop, and the host CPU suffers from the longest network latency due to its additional NIC switch hop and its host PCIe interconnect.

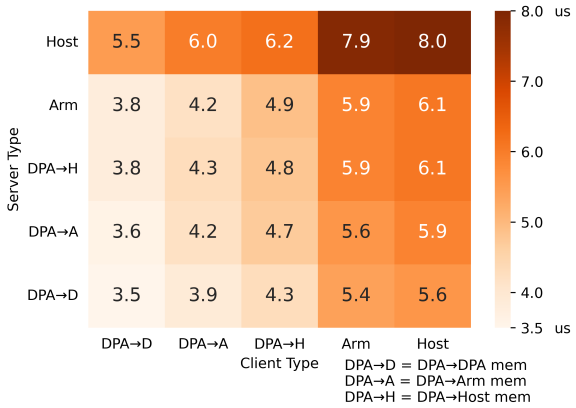


Figure 10: L2 reflector latency for different units.

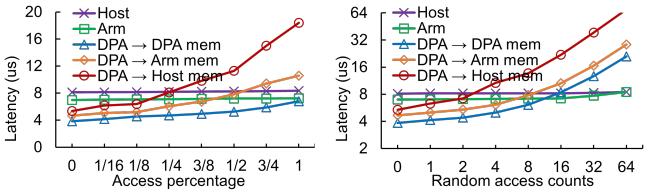
Second, the DPA implementation on DPA memory offers the lowest latency compared to that on host/Arm memory. This is because received network packets can be directly put into DPA's L2/L3 memory when using DPA memory without crossing the high-latency NIC switch or PCIe interconnect. When using the Arm memory, the received packets can only be put into the Arm L3 cache through the NIC switch. When using the host memory, the received packets can only be put into the host L3 cache through the NIC switch and PCIe interconnect.

Latency vs. Application complexity. Although DPA provides lower network latency, its memory subsystem and computing power are far inferior to the host/Arm. When a networking application involves heavy processing, the advantage of DPA being closer to the network is overshadowed by its poor memory and computing performance.

We conduct two experiments to demonstrate this issue. In the first experiment, the L2 reflector server additionally reads different percentages of the packet and sums them up as a one-by-one 8-byte integer, as shown in Figure 11a. In the second experiment, we approximate a memory-intensive network function by configuring the L2 reflector to perform different numbers of random memory reads per packet from an 8 MB buffer, as shown in Figure 11b. In both experiments, the packet size is fixed to 1 KB, and we measure the end-to-end L2-reflector latency of the host, Arm, and DPA using three different memories.

We observe that with the increase in the reading ratio or the number of per-packet random accesses, DPA's latency significantly increases while the host/Arm latency slightly increases. This is because DPA's memory subsystem and single-thread computing power are much wimpier than the host/Arm. These two experiments demonstrate the fragility of DPA's latency advantages, which can easily be surpassed in complex processing due to the host/Arm's stronger computing power and memory subsystem.

Takeaways (Networking Latency). DPA is closer to the network compared with the host/Arm and thus has the lowest network latency. However, this latency advantage is fragile and will be nullified when handling complex operations.



(a) Access sub-packet (b) Random access in 8MB

Figure 11: L2 reflector latency on different memory.

C. Network Throughput

In this subsection, we measure the achievable network throughput regarding different patterns (send/receive). For the DPA core, we use different memories as packet buffers. Figure 12 shows the achievable network throughput for different patterns under different packet sizes (64 Bytes and 1 KB). We have three observations.

First, when the packet size is small (64 Bytes), none of the three processors can achieve the network line rate. DPA cores require more threads to achieve comparable throughput than the host CPU and Arm.

Second, when the packet size is large (1 KB), all three processors can achieve the network line rate. But DPA still requires more threads than the host CPU and Arm, indicating that the DPA thread is much wimpier than the host/Arm thread. Luckily, the number of DPA threads (256) is noticeably more than the Arm (16) and the host (32). So DPA cores can still achieve comparable network throughput using more threads.

Third, when the packet size is large (1 KB), DPA must use Arm memory or host memory to achieve the network line rate. Using DPA memory only achieves around 100 Gbps for sending packets and around 50 Gbps for receiving packets. We speculate that this is due to the internal limitation of the DPA's L2/L3 cache since using host/Arm memory would bypass DPA's L2/L3 cache.

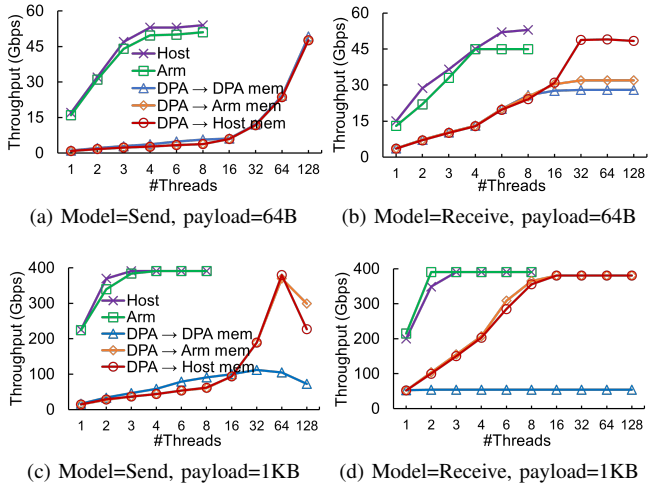
Takeaways (Networking Throughput). DPA thread is significantly wimpier than host/Arm threads regarding sending/receiving network packets, but DPA has more threads and can achieve comparable network throughput as host/Arm.

DPA's unique networking characteristic: DPA has the advantage of being closer to the network and thus enjoys lower network latency. Programmers can consider offloading latency-sensitive network applications to the DPA. However, DPA's single-thread performance is too wimpy, so the offloaded workloads can not involve heavy computation or many memory operations. Otherwise, the latency advantages would quickly be overshadowed. We further use a case study to measure the impact of DPA's unique network characteristic on the end-to-end applications in § V-A.

V. Case Studies

In this section, guided by the above benchmarking results, we further use three case studies to explore the three unique characteristics of the DPA cores. In each case study, we have five kinds of implementations unless stated otherwise.

- "Host": all functions are deployed in the host CPU.



(a) Model=Send, payload=64B (b) Model=Receive, payload=64B

(c) Model=Send, payload=1KB (d) Model=Receive, payload=1KB

Figure 12: Throughput comparison under different threads.

- "Arm": all functions are deployed in Arm.
- "DPA→Y mem": all functions are deployed in DPA. "Y mem" indicates the network buffer memory type (host/Arm/DPA memory) used.

A. Clock Synchronization Service

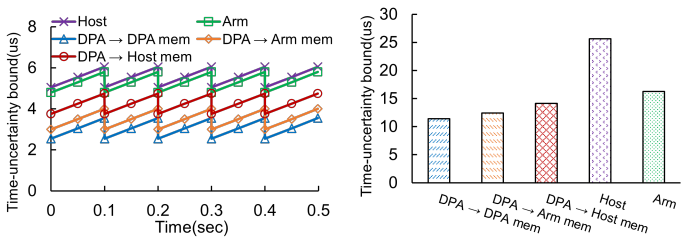
Compared with the host/Arm, DPA is the closest to the network. Latency-sensitive network workloads can benefit from DPA's characteristic that is closer to the network.

To explore the potential impact, we conduct a case study on latency-sensitive clock synchronization service. Clock synchronization is critical for many datacenter applications [38] such as distributed transactional databases [39], [40], consistent snapshots [41], [42], and network telemetry [43], [44]. The key metric for clock synchronization is the *time uncertainty bound* for each node [38], [39], [45], donated as ϵ. Lower time uncertainty bound (ϵ) can improve the performance of distributed applications and enable more accurate one-way delay measurement.

Implementations. We use two servers mentioned in § II-C, one as a synchronization client and the other as a synchronization master node. The synchronization duration is set to be 0.1 seconds and the clock drift is set to be 10 microseconds per second [46]. We have deployed five kinds of clock synchronization services as mentioned above.

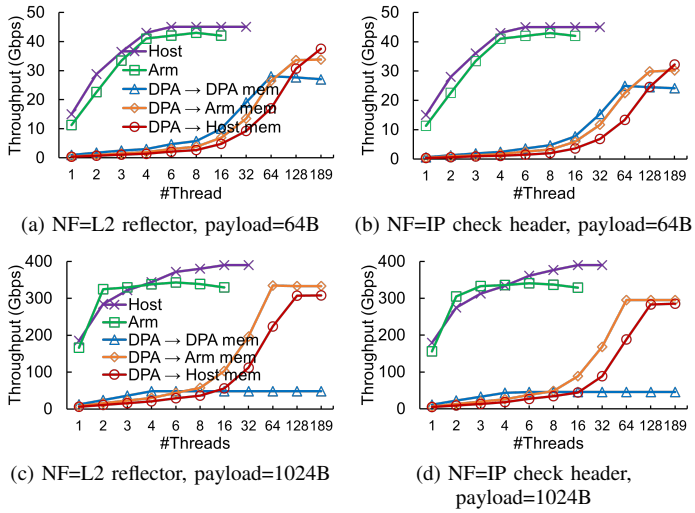
Figure 13a shows the time uncertainty bound (ϵ) for different implementations when the network is under-loaded, i.e., there is only a clock synchronization service in each server. Figure 13b shows the 999th percentile time uncertainty bound when the network is heavily used. To make the network heavily used, we let an extra network-intensive L2-reflector application run in the host cores at the same time and it consumes up to 400 Gbps bi-direction network throughput. We have two observations.

First, all three DPA implementations offer much lower time uncertainty bound than the host/Arm, up to 2.0× in average latency and 2.3× in 999th percentile latency. This



(a) Without network pressure (b) P999 under 400Gbps network

Figure 13: Time-uncertainty bound under different patterns.



(a) NF=L2 reflector, payload=64B (b) NF=IP check header, payload=64B

(c) NF=L2 reflector, payload=1024B (d) NF=IP check header, payload=1024B

Figure 14: L2 reflector and IP check header network functions under different network patterns.

indicates that DPA's characteristics can greatly improve the performance of latency-sensitive network applications due to its location closer to the network. Second, among three DPA implementations, "DPA→DPA mem" offers the lowest time uncertainty bound. This is because the network packets can directly be put into or be fetched from DPA's L2/L3 cache when DPA cores are using DPA memory. This indicates that latency-sensitive network applications running in the DPA cores should use DPA memory as the network packet buffer.

Guideline 1: Offloading latency-sensitive logic to DPA to improve performance. DPA is closer to the network compared with the host/Arm. An application that is sensitive to network latency and fits within DPA cache working set size can leverage this characteristic to improve end-to-end performance. Also, the programmer should choose DPA memory as the network buffer to promote incoming network packets to DPA caches and thus further decrease the processing latency.

B. Network Function Virtualization

Compared with the host/Arm, DPA has a thread number that is an order of magnitude higher. To fully enjoy the vast number of threads, the offloaded workloads should be very easy to parallelize, otherwise, the application performance can significantly downgrade due to DPA's poor single-thread computing ability.

In this case study, we leverage stateless network function virtualization (NFV) [47]–[49] to explore DPA's many-core parallelism. Stateless network functions are usually easy to parallelize and can scale well with the number of threads.

Implementations. We focus on the throughput metric instead of the latency metric. We choose two stateless network functions: L2-reflector and CheckIPHeader. These two network functions are not memory-intensive such that we can minimize the impact of DPA's memory subsystem characteristics. We have five kinds of implementations as mentioned above.

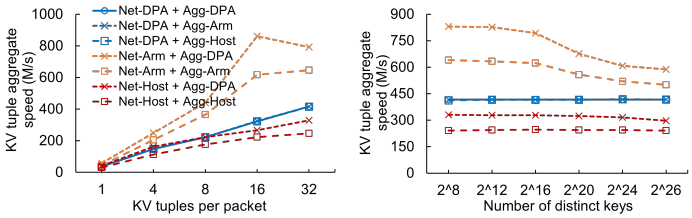
Figure 14 shows the result when packet size is 64B and 1KB. We have two observations. First, DPA's single thread throughput is substantially lower than that of the host/Arm, which is within our expectations. However, with its high thread count, DPA can still achieve comparable throughput as the host/Arm. This indicates that programmers should consider offloading easy-to-parallelize workloads to DPA cores, otherwise, the system throughput may drop significantly (up to several magnitudes) due to DPA's wimpy single-thread performance. Second, when the packet size is large (1KB), the throughput of "DPA→DPA mem" can not scale along the number of threads. This is consistent with our results from previous network throughput measurements § IV-C, i.e., the maximum send and receive throughput of "DPA→DPA mem" is around 100 Gbps and 50 Gbps, respectively. This indicates that throughput-intensive applications running in the DPA should avoid using DPA memory as the network buffer.

Guideline 2: Offload easy-to-parallelize applications to DPA. Although DPA's single thread is very wimpy, in terms of computing power and memory bandwidth, its thread count (256) is far more than the host/Arm, and can provide line-rate processing ability while processing stateless network function workloads. We suggest programmers can consider using easy-to-parallelize workloads to leverage DPA's many-core characteristics. In addition, using DPA memory for memory-bandwidth-intensive applications is not the best solution due to internal DPA cache limitation.

C. Key-value Aggregation

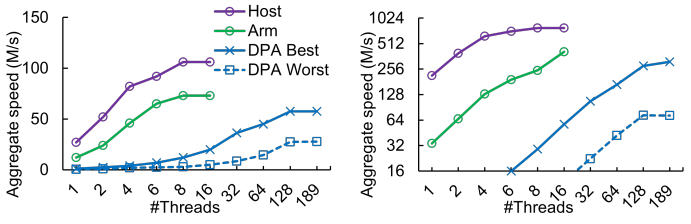
To explore the potential impact of this characteristic, we conduct a case study on memory-intensive key-value stream aggregation [50]. Key-value stream aggregation is a memory-intensive operation widely existing in various distributed systems, e.g., $reduce()$ in big data processing [51], [52], $AllReduce()$ in distributed training [53]–[56], $MPI\_Reduce()$ in high-performance computing [57], etc. Key-value aggregation is considered to be memory-intensive because the application needs a large memory to hold the intermediate aggregation results and needs to frequently update the intermediate results according to the received network packets. We explore DPA's memory characterization by carefully optimizing the DPA implementation.

Implementations. When running key-value stream aggregation, we mainly have two memory regions: network buffer (NetBuf) and aggregation buffer (AggBuf). The network buffer



(a) 2^{16} distinct keys (b) 32 KV tuples per packet

Figure 15: Effect of memory types on DPA's throughput.



(a) 1 KV tuple per packet (b) 8 KV tuples per packet

Figure 16: Throughput comparison between three processors.

is used to send/receive network packets (also known as network data ring buffer) while the aggregation buffer is used to hold the intermediate aggregation results for each key. We have seven implementations with supplementary explanations.

- "Net-X+Agg-Y": the aggregation service is deployed in the DPA cores. "X" indicates the used memory type for NetBuf while "Y" indicates the used memory type for AggBuf.

Effect of Memory Type. We first evaluate the effect of memory types on the DPA's end-to-end key-value aggregation throughput. We generate an artificial uniform distribution trace to understand the effectiveness of hot-key. The key-value tuple size is 16 bytes and each packet has a 64 bytes header.

Figure 15a shows the achieved aggregation throughput under different key-value tuples per packet. We observe that "Net-Arm+Agg-DPA" always achieves the highest throughput. Regarding NetBuf, we should use the host memory or Arm memory, as using DPA memory to handle network packets has up to 7.5× lower throughput than the host/Arm (§ IV-C). Regarding AggBuf, we should consider using the DPA memory or Arm memory for their higher memory bandwidth than the host (§ III-B3). Among these possible combinations, "Net-Arm+Agg-DPA" provides the highest throughput.

Figure 15b shows the aggregation throughput under different numbers of distinct keys. The per-packet KV tuples are fixed to 32. We have three observations. First, when the number of distinct keys increases, the throughput for "Net-Arm+Agg-DPA" and "Net-Arm+Agg-Arm" noticeably decreases because the number of distinct keys exceeds the DPA's L2/L3 cache size and Arm's L3 cache size.

Second, all implementations that use DPA memory as NetBuf (all blue lines) suffer from low throughput since they are bounded by DPA memory's limited ability to receive packets.

Third, two implementations that use the host memory as NetBuf (two red lines) show the lowest throughput because the Key-value aggregation service requires reading the received packets after receiving the packets into the memory, while the throughput of the DPA core accessing the host memory is very low (7.2 GB/s read bandwidth and 14 GB/s write bandwidth).

Comparisons with the host/Arm. We then compare the aggregation throughput of the host/Arm/DPA. We use "yelp" trace from production [58]. We select the best("Net-Arm+Agg-DPA") and worst("Net-Host+Agg-Host") results from all combinations for DPA (named "DPA-Best" and "DPA-Worst"). Figure 16 shows the achieved aggregation throughput under different thread numbers. We have two observations.

First, DPA's best-case aggregation throughput is lower but comparable to the host (2.5×)/Arm (1.3×). Although lower than the host/Arm, DPA's end-to-end aggregation throughput still surprises us, considering that DPA's memory latency is an order of magnitude higher than the host/Arm and the memory throughput is significantly lower than the host/Arm.

Second, DPA's best-case aggregation throughput is markedly up to 4.3× higher than that of the worst-case. This indicates the importance of selecting appropriate memories for different usages when running applications in DPA cores.

Guideline 3: DPA programmers should carefully choose memories from the host/DPA/Arm memory. Using different memories for DPA can result in significantly different performance. The performance gap between the best (Net-Arm + Agg-DPA) and worst (Net-Host + Agg-Host) combinations can be as high as several times. Programmers must choose different memories according to the specific usage, to guarantee high performance when implementing on DPA.

VI. Discussion

Conclusion of guidelines. In this section, we conclude three important DPA-related guidelines for future datapath accelerator-enhanced SmartNIC programmers, followed by our suggestions for DPA-enhanced SmartNIC vendors.

- **1. Offloading latency-sensitive and simple workloads to DPA cores.** DPA is much closer to the network and enjoys lower network latency. Latency-sensitive applications can exploit this characteristic to improve end-to-end performance. Meanwhile, DPA's single-thread performance is minimal and the offloaded applications cannot involve too heavy computing, otherwise, the low-latency benefit can be quickly overshadowed.
- **2. Offloading easy-to-parallelize workloads with small working set sizes.** Our benchmarking results reveal that DPA has more cores than the host/Arm, while each thread is much wimpier. Programmers can offload easy-to-parallelize workloads to exploit DPA's many-core parallelism. However, when the application's working set size exceeds DPA's cache size, DPA has to frequently access memory and thus suffers from severe performance downgradation. Therefore, the memory working set size should better fit in DPA's cache size to avoid significant performance downgradation.
- **3. Carefully select memory buffers for DPA applications.** DPA can not only access its own memory/cache but also access the off-path Arm's memory and the host CPU's

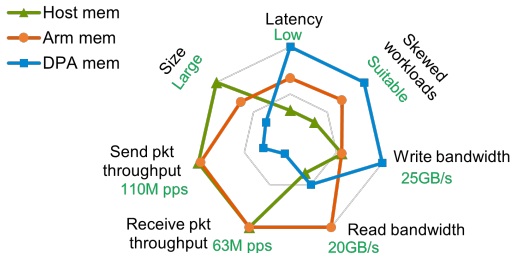


Figure 17: Radar chart for DPA under three memory types. The more prominent a line appears on a radar chart, the more indicative it is of a better performance.

memory. Placing application buffers in any of three kinds of memory leads to significantly different performance. Therefore, we list seven main indicators that programmers may be concerned about when choosing memories, as shown in Figure 17. where a larger value in each axis is better. We highlight three noticeable hints. First, use the host memory or the Arm memory as the network buffer for bandwidth-intensive network applications, since DPA memory performs poorly regarding the throughput of sending/receiving packets. Second, using host memory is more suitable for DPA applications with large memory footprints because its available size is substantially larger than DPA/Arm memory, given that the host X86 has up to eight per socket memory channels while BF3 only features two DDR5 channels. Third, DPA memory has the potential to perform better in skewed workloads since it features an additional DPA L2/L3 cache that is closer to the DPA core.

Suggestions for SmartNIC vendors. Our characterization has uncovered several important factors that bound the performance of the datapath accelerator. And we have some suggestions for SmartNIC vendors.

- **1. Directly attach a memory to the DPA.** Currently, DPA has no directly attached memory, all memory accesses have to go through a high-latency NIC switch. This incurs at least 5× higher memory access latency than that of Arm. This can be harmful for applications whose working sets can not be fully cached by DPA's cache. As such, we suggest vendors consider directly attaching a memory to the DPA.
- **2. Equip DPA with a more powerful cache.** Current DPA cache suffers from very high latency (up to 10.5× higher L1 latency than the host CPU cache). Besides, the per-thread L1 cache bandwidth is only 0.53 GB/s (up to 92× lower than the host L1 cache). Although the frequency of DPA (1.8 GHz) is not significantly lower than that of the host (2.5 GHz), DPA's uncore frequency is 4.4× lower than the host, which may explain why the DPA cache performance is so poor. As such, we suggest vendors consider equipping DPA with a more powerful cache.

VII. Related Works

To our knowledge, this paper is the first to benchmark DPA-enhanced SmartNIC.

Characterizing SmartNICs. Prior works [7], [19], [21]–[24] characterize SmartNICs from different perspectives. iPipe [7] characterizes SoC SmartNICs regarding their computing capacity, memory, traffic control, and host communication. In contrast, we characterize the different processors in a BF3-attached server from architectural perspectives and explore the possible influence of the unique architectural characteristics of the DPA. Wei et al. [23] characterize the off-path BlueField-2 SmartNIC from a communication-path perspective and concurrently use multiple communication paths to improve throughputs. In contrast, we focus on the architectural differences of processors within a BF3-attached server and optimize offloading performance according to the architectural characteristics. Sun et al. [22] evaluate off-path BlueField-2 using high-level applications and suggest using the SmartNIC as a new endpoint to provide horizontal scaling. Liu et al. [24] characterize the networking and computing of the BlueField-2's off-path Arm. Michalowicz et al. [19] compare BlueField-2 and BlueField-3 mainly regarding the off-path Arm's computing ability. In contrast, we focus on characterizing BlueField-3's DPA and Arm from an architectural perspective.

SmartNIC offloading. Offloading host workloads to SmartNICs has recently attracted significant attention in both academia and industry. Many prior works [14], [55], [59]–[89] offload host tasks to FPGA-based or SoC-based SmartNICs. PANIC [63] addresses the performance isolation and fairness problems under the multi-tenant environment. FlowBlaze [64] enables stateful network packet processing on FPGAs. Xenic [90] offloads distributed transactions to SmartNIC. These works leverage SmartNIC to alleviate host CPU pressure but do not provide a comprehensive study on DPA-enhanced SmartNIC.

VIII. Conclusion

Typical multicore SmartNICs such as BlueFiled-2 are only capable of processing control-plane tasks with their embedded processors, which cannot directly process network traffic from cloud applications that evolve rapidly over time. Therefore, SmartNIC-related research calls for a programmable datapath accelerator (DPA) to process network traffic at line rate. However, no existing work has unveiled the performance characteristics of DPA. We present the first architectural characterization of the latest DPA-enhanced BlueField-3 SmartNIC. We identify that DPA has three unique architectural characteristics that unleash the potential of DPA. We propose three pioneering guidelines for programmers to fully unleash the potential of DPA. To demonstrate the effectiveness of our approach, we conduct detailed case studies regarding each guideline.

Acknowledgement. We thank the anonymous ICNP reviewers and our shepherd Sebastiano Miano for their detailed feedback. The work is supported by the following grants: the National Science and Technology Major Project (NO.2022ZD0119301), the National Natural Science Foundation of China (Grant No. 62472384), a research grant from Alibaba Group through Alibaba Innovative Research (AIR) Program, Starry Night Science Fund of Zhejiang University Shanghai Institute for Advanced Study (SN-ZJU-SIAS-0010). Yin Zhang and Zeke Wang are the corresponding authors.

REFERENCES

[1] Broadcom, "BCM957504-N1100G," https://docs.broadcom.com/doc/957504-N1100G-DS, 2020.

[2] Mellanox, "ConnectX®-5 En Card Product Brief," https://www.mellanox.com/sites/default/files/relateddocs/prod_adapter_cards/PB_ConnectX-5_EN_Card.pdf, 2017.

[3] ——, "ConnectX®-6 En Card Product Brief," https://www.mellanox.com/sites/default/files/relateddocs/prod_adapter_cards/PB_ConnectX-6_EN_Card.pdf, 2017.

[4] Broadcom, "BCM957508-P2200G," https://docs.broadcom.com/doc/957508-P2200G-DS, 2019.

[5] Ehernet Technology Consortium, "800G specification," https://ethernettechnologyconsortium.org/wpcontent/uploads/2020/03/800G-Specification_r1.0.pdf, 2020.

[6] Nvidia, "NVIDIA BLUEFIELD-2 DPU," https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/documents/datasheet-nvidia-bluefield-2-dpu.pdf, 2021.

[7] M. Liu, T. Cui, H. Schuh, A. Krishnamurthy, S. Peter, and K. Gupta, "Offloading distributed applications onto smartnics using ipipe," in *SIGCOMM*, 2019.

[8] Nvidia, "NVIDIA BLUEFIELD-3 DPU," https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/documents/datasheet-nvidia-bluefield-3-dpu.pdf, 2022.

[9] Broadcom, "Stingray™ PS250," https://docs.broadcom.com/doc/PS250-PB, 2024.

[10] Marvell, "Marvell LiquidIO II SmartNICs," https://www.marvell.com/products/infrastructure-processors/liquidio-smart-nics/liquidio-ii-smart-nics.html, 2024.

[11] Intel, "Intel® Infrastructure Processing Unit," https://www.intel.com/content/www/us/en/products/details/network-io/ipu.html, 2024.

[12] S. Kanev, J. P. Darago, K. Hazelwood, P. Ranganathan, T. Moseley, G.-Y. Wei, and D. Brooks, "Profiling a warehouse-scale computer," *ISCA*, 2015.

[13] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: research problems in data center networks," 2008.

[14] M. Handley, C. Raiciu, A. Agache, A. Voinescu, A. W. Moore, G. Antichi, and M. Wojcik, "Re-architecting datacenter networks and stacks for low latency and high performance," in *SIGCOMM*, 2017.

[15] A. Gonzalez, A. Kolli, S. Khan, S. Liu, V. Dadu, S. Karandikar, J. Chang, K. Asanovic, and P. Ranganathan, "Profiling hyperscale big data processing," in *ISCA*, 2023.

[16] C. Zeng, L. Luo, T. Zhang, Z. Wang, L. Li, W. Han, N. Chen, L. Wan, L. Liu, Z. Ding *et al.*, "Tiara: A scalable and efficient hardware acceleration architecture for stateful layer-4 load balancing," in *NSDI*, 2022.

[17] A. Caulfield, P. Costa, and M. Ghobadi, "Beyond smartnics: Towards a fully programmable cloud," in *HPSR*, 2018.

[18] M. T. Arashloo, A. Lavrov, M. Ghobadi, J. Rexford, D. Walker, and D. Wentzlaff, "Enabling programmable transport protocols in high-speed nics," in *NSDI*, 2020.

[19] B. Michalowicz, K. K. Suresh, H. Subramoni, D. K. D. Panda, and S. Poole, "Battle of the bluefields: An in-depth comparison of the bluefield-2 and bluefield-3 smartnics," in *HOTI*, 2023.

[20] Y. Li, A. Kashyap, Y. Guo, and X. Lu, "Characterizing lossy and lossless compression on emerging bluefield dpu architectures," in *HOTI*, 2023.

[21] Z. Wang, C. Wang, and L. Wang, "Dpubench: An application-driven scalable benchmark suite for comprehensive dpu evaluation," *TBench*, 2023.

[22] S. Sun, C. Huang, R. Zhang, L. Chen, Y. Huang, M. Yan, and J. Wu, "A comprehensive study on optimizing systems with data processing units," *arXiv*, 2023.

[23] X. Wei, R. Cheng, Y. Yang, R. Chen, and H. Chen, "Characterizing off-path smartnic for accelerating distributed systems," in *OSDI*, 2023.

[24] J. Liu, C. Maltzahn, C. Ulmer, and M. L. Curry, "Performance characteristics of the bluefield-2 smartnic," *arXiv*, 2021.

[25] Z. Yu, B. Su, W. Bai, S. Raindel, V. Braverman, and X. Jin, "Understanding the micro-behaviors of hardware offloaded network stacks with lumina," in *SIGCOMM*, 2023, pp. 1074–1087.

[26] Y. Yuan, J. Huang, Y. Sun, T. Wang, J. Nelson, D. R. Ports, Y. Wang, R. Wang, C. Tai, and N. S. Kim, "Rambda: Rdma-driven acceleration framework for memory-intensive μs-scale datacenter applications," in *HPCA*, 2023.

[27] A. Bolat, F. Siddiqui, S. Sezer, K. Tasdemir, and R. Khan, "Investigation of communication overhead of soc lookaside accelerators," in *SOCC*, 2023.

[28] Nvidia, "Link Aggregation," https://docs.nvidia.com/networking/display/bluefielddpuosv450/link+aggregation, 2024.

[29] ——, "DOCA," https://developer.nvidia.com/networking/doca, 2024.

[30] DPDK, "Data Plane Development Kit," https://www.dpdk.org/, 2024.

[31] Ted Hudek, "Receive Side Scaling," https://learn.microsoft.com/en-us/windows-hardware/drivers/network/introduction-to-receive-side-scaling, 2017.

[32] A. Ilic, F. Pratas, and L. Sousa, "Cache-aware roofline model: Upgrading the loft," *IEEE Computer Architecture Letters*, 2013.

[33] Nvidia, "DOCA DPA Subsystem Programming Guide," https://docs.nvidia.com/doca/sdk/dpa-subsystem-programming-guide/index.html, 2024.

[34] G. Weisz, J. Melber, Y. Wang, K. Fleming, E. Nurvitadhi, and J. C. Hoe, "A study of pointer-chasing performance on shared-memory processor-fpga systems," in *FPGA*, 2016.

[35] A. Farshin, A. Roozbeh, G. Q. Maguire Jr, and D. Kostić, "Reexamining direct cache access to optimize {I/O} intensive applications for multi-hundred-gigabit networks," in *ATC*, 2020.

[36] M. Alian, Y. Yuan, J. Zhang, R. Wang, M. Jung, and N. S. Kim, "Data direct i/o characterization for future i/o system exploration," in *ISPASS*. IEEE, 2020.

[37] M. Wang, M. Xu, and J. Wu, "Understanding i/o direct cache access performance for end host networking," *POMACS*, 2022.

[38] Y. Li, G. Kumar, H. Hariharan, H. Wassel, P. Hochschild, D. Platt, S. Sabato, M. Yu, N. Dukkipati, P. Chandra *et al.*, "Sundial: Fault-tolerant clock synchronization for datacenters," in *OSDI*, 2020.

[39] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild *et al.*, "Spanner: Google's globally distributed database," *TOCS*, 2013.

[40] A. Shamis, M. Renzelmann, S. Novakovic, G. Chatzopoulos, A. Dragojević, D. Narayanan, and M. Castro, "Fast general distributed transactions with opacity," in *SIGMOD*, 2019.

[41] K. M. Chandy and L. Lamport, "Distributed snapshots: Determining global states of distributed systems," *TOCS*, 1985.

[42] R. Koo and S. Toueg, "Checkpointing and rollback-recovery for distributed systems," *TSE*, 1987.

[43] Y. Li, R. Miao, C. Kim, and M. Yu, "Flowradar: A better netflow for data centers," in *NSDI*, 2016.

[44] ——, "Lossradar: Fast detection of lost packets in data center networks," in *CoNEXT*, 2016.

[45] Y. Deshpande, P. Diederich, and W. Kellerer, "Towards a network aware model of the time uncertainty bound in precision time protocol," in *INFOCOM WKSHPS*, 2023.

[46] Y. Geng, S. Liu, Z. Yin, A. Naik, B. Prabhakar, M. Rosenblum, and A. Vahdat, "Exploiting a natural network effect for scalable, fine-grained clock synchronization," in *NSDI*, 2018.

[47] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," *IEEE Communications surveys & tutorials*, 2015.

[48] G. Liu, Y. Ren, M. Yurchenko, K. Ramakrishnan, and T. Wood, "Microboxes: High performance nfv with customizable, asynchronous tcp stacks and dynamic subscriptions," in *SIGCOMM*, 2018.

[49] K. Yasukata, F. Huici, V. Maffione, G. Lettieri, and M. Honda, "Hypernf: Building a high performance, high utilization and fair nfv platform," in *SOCC*, 2017.

[50] Y. He, W. Wu, Y. Le, M. Liu, and C. Lao, "A generic service to provide in-network aggregation for key-value streams," in *ASPLOS*, 2023.

[51] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, 2008.

[52] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauly, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *NSDI*, 2012.

[53] N. Gebara, M. Ghobadi, and P. Costa, "In-network aggregation for shared machine learning clusters," *MLSys*, 2021.

[54] C. Lao, Y. Le, K. Mahajan, Y. Chen, W. Wu, A. Akella, and M. Swift, "Atp: In-network aggregation for multi-tenant learning," in *NSDI*, 2021.

[55] A. Sapio, M. Canini, C.-Y. Ho, J. Nelson, P. Kalnis, C. Kim, A. Krishnamurthy, M. Moshref, D. Ports, and P. Richtárik, "Scaling distributed machine learning with in-network aggregation," in *NSDI*, 2021.

[56] A. Sapio, M. Canini, C.-Y. Ho, J. Nelson, P. Kalnis, C. Kim, A. Krishnamurthy, M. Moshref, D. Ports, and P. Richtarik, "Scaling distributed machine learning with {In-Network} aggregation," in *NSDI*, 2021.

[57] I. Laguna, R. Marshall, K. Mohror, M. Ruefenacht, A. Skjellum, and N. Sultana, "A large-scale study of mpi usage in open-source hpc applications," in *SC*, 2019.

[58] Yelp, "Yelp Open Dataset," https://www.yelp.com/dataset, 2022.

[59] B. Li, K. Tan, L. Luo, Y. Peng, R. Luo, N. Xu, Y. Xiong, P. Cheng, and E. Chen, "Clicknp: Highly flexible and high performance network processing with reconfigurable hardware," in *SIGCOMM*, 2016.

[60] J. Zhang, H. Huang, L. Zhu, S. Ma, D. Rong, Y. Hou, M. Sun, C. Gu, P. Cheng, C. Shi *et al.*, "Smartds: Middle-tier-centric smartnic enabling application-aware message split for disaggregated block storage," in *ISCA*, 2023.

[61] D. Firestone, A. Putnam, S. Mundkur, D. Chiou, A. Dabagh, M. Andrewartha, H. Angepat, V. Bhanu, A. Caulfield, E. Chung, H. K. Chandrappa, S. Chaturmohta, M. Humphrey, J. Lavier, N. Lam, F. Liu, K. Ovtcharov, J. Padhye, G. Popuri, S. Raindel, T. Sapre, M. Shaw, G. Silva, M. Sivakumar, N. Srivastava, A. Verma, Q. Zuhair, D. Bansal, D. Burger, K. Vaid, D. A. Maltz, and A. Greenberg, "Azure accelerated networking:smartnics in the public cloud," in *NSDI*, 2018.

[62] M. T. Arashloo, A. Lavrov, M. Ghobadi, J. Rexford, D. Walker, and D. Wentzlaff, "Enabling programmable transport protocols in high-speed nics," in *NSDI*, 2020.

[63] J. Lin, K. Patel, B. E. Stephens, A. Sivaraman, and A. Akella, "Panic: A high-performance programmable nic for multi-tenant networks," in *OSDI*, 2020.

[64] S. Pontarelli, R. Bifulco, M. Bonola, C. Cascone, M. Spaziani, V. Bruschi, D. Sanvito, G. Siracusano, A. Capone, M. Honda, F. Huici, and G. Bianchi, "Flowblaze: Stateful packet processing in hardware," in *NSDI*, 2019.

[65] P. M. Phothilimthana, M. Liu, A. Kaufmann, S. Peter, R. Bodik, and T. Anderson, "Floem: A programming system for nic-accelerated network applications," in *OSDI*, 2018.

[66] Z. Wang, K. Kara, H. Zhang, G. Alonso, O. Mutlu, and C. Zhang, "Accelerating generalized linear models with mlweaving: a one-size-fits-all system for any-precision learning," *VLDB*, 2019.

[67] M. Liu, S. Peter, A. Krishnamurthy, and P. M. Phothilimthana, "E3:energy-efficient microservices on smartnic-accelerated servers," in *ATC*, 2019.

[68] S. Choi, M. Shahbaz, B. Prabhakar, and M. Rosenblum, "λ-nic: Interactive serverless compute on programmable smartnics," in *ICDCS*, 2020.

[69] S. Grant, A. Yelam, M. Bland, and A. C. Snoeren, "Smartnic performance isolation with fairnic: Programmable networking for the cloud," in *SIGCOMM*, 2020.

[70] Wang, Zeke and Huang, Hongjing and Zhang, Jie and Wu, Fei and Alonso, Gustavo, "FpgaNIC: An FPGA-based Versatile 100Gb SmartNIC for GPUs," in *ATC*, 2022.

[71] Z. Wang, H. Huang, J. Zhang, and G. Alonso, "Shuhai: Benchmarking high bandwidth memory on fpgas," in *FCCM*, 2020.

[72] M. Khazraee, A. Forencich, G. C. Papen, A. C. Snoeren, and A. Schulman, "Rosebud: Making FPGA-Accelerated Middlebox Development More Pleasant," in *ASPLOS*, 2023.

[73] G. Alonso, "Technical perspective: Dfi: The data flow interface for high-speed networks," *SIGMOD Rec.*, 2022.

[74] J. Li, Y. Lu, Q. Wang, J. Lin, Z. Yang, and J. Shu, "AlNiCo: SmartNIC-accelerated contention-aware request scheduling for transaction processing," in *ATC*, 2022.

[75] Z. István, D. Sidler, G. Alonso, and M. Vukolic, "Consensus in a Box: Inexpensive Coordination in Hardware," in *NSDI*, 2016.

[76] J. Wirth, J. A. Hofmann, L. Thostrup, C. Binnig, and A. Koch, "Scalable and Flexible High-Performance In-Network Processing of Hash Joins in Distributed Databases," in *FPT*, 2021.

[77] T. Kim, D. M. Ng, J. Gong, Y. Kwon, M. Yu, and K. Park, "Rearchitecting the tcp stack for i/o-offloaded content delivery," in *NSDI*, 2023.

[78] H. Sadok, N. Atre, Z. Zhao, D. S. Berger, J. C. Hoe, A. Panda, J. Sherry, and R. Wang, "Enso: A streaming interface for nic-application communication," in *OSDI*, 2023.

[79] J. Huang, J. Lou, S. Vanavasam, X. Kong, H. Ji, I. Jeong, D. Zhuo, E. K. Lee, and N. S. Kim, "Hal: Hardware-assisted load balancing for energy-efficient snic-host cooperative computing," in *ISCA*, 2024.

[80] Y. Liao, J. Wu, W. Lu, X. Li, and G. Yan, "Dpu-direct: Unleashing remote accelerators via enhanced rdma for disaggregated datacenters," *TC*, 2024.

[81] J. Kim, I. Jang, W. Reda, J. Im, M. Canini, D. Kosti, Y. Kwon, S. Peter, and E. Witchel, "Linefs: Efficient smartnic offload of a distributed file system with pipeline parallelism," in *SOSP*, 2021.

[82] H. Huang, Y. Li, J. Sun, X. Zhu, J. Zhang, L. Luo, J. Li, and Z. Wang, "P4sgd: Programmable switch enhanced model-parallel training on generalized linear models on distributed fpgas," *TPDS*, 2023.

[83] A. Lerner, C. Binnig, P. Cudré-Mauroux, R. Hussein, M. Jasny, T. Jepsen, D. R. K. Ports, L. Thostrup, and T. Ziegler, "Databases on modern networks: A decade of research that now comes into practice," *VLDB*, 2023.

[84] J. Kim, I. Jang, W. Reda, J. Im, M. Canini, D. Kostić, Y. Kwon, S. Peter, and E. Witchel, "Linefs: Efficient smartnic offload of a distributed file system with pipeline parallelism," in *SOSP*, 2021.

[85] S. Ibanez, A. Mallery, S. Arslan, T. Jepsen, M. Shahbaz, C. Kim, and N. McKeown, "The nanopu: A nanosecond network stack for datacenters," in *OSDI*, 2021.

[86] J. Huang, J. Lou, Y. Sun, T. Wang, E. K. Lee, and N. Sung Kim, "Making sense of using a smartnic to reduce datacenter tax from slo and tco perspectives," in *IISWC*, 2023.

[87] N. Lazarev, S. Xiang, N. Adit, Z. Zhang, and C. Delimitrou, "Dagger: efficient and fast rpcs in cloud microservices with near-memory reconfigurable nics," in *ASPLOS*, 2021.

[88] Q. Zhang, P. Bernstein, B. Chandramouli, J. Hu, and Y. Zheng, "Dds: Dpu-optimized disaggregated storage," *VLDB*, 2024.

[89] Z. Guo, J. Lin, Y. Bai, D. Kim, M. Swift, A. Akella, and M. Liu, "Lognic: A high-level performance model for smartnics," in *MICRO*, 2023.

[90] H. N. Schuh, W. Liang, M. Liu, J. Nelson, and A. Krishnamurthy, "Xenic: Smartnic-accelerated distributed transactions," in *ASPLOS*, 2021.