Title
Use It or Lose It: Cheap Compute Everywhere

ISBN
9783030964979

Authors
Groves, Taylor
Hazen, Damian
Lockwood, Glenn
et al.

Publication Date
2022

DOI
10.1007/978-3-030-96498-6_16

Peer reviewed

Use It or Lose It: Cheap Compute Everywhere*

Taylor Groves[1], Damian Hazen[1], Glenn Lockwood[1], and Nicholas J. Wright[1]

NERSC, Lawrence Berkeley National Laboratory
{tgroves, dhazen, glock, njwright}@lbl.gov

Abstract. Moore's Law is tapering off, but FLOPS per dollar continues to grow. Inexpensive CPUs are emerging everywhere from network to storage as an effective way of managing and deploying hardware and firmware as well as providing services close to the data path. Examples of this include ARM cores within Mellanox Bluefield, Broadcom Stingray DPUs, switches, and compute in storage. This additional processing power can be useful for (1) enabling higher throughput, (2) decreasing or hiding latency, (3) increasing power/cost efficiency, (4) alleviating contention for oversubscribed resources. In order to make these additional resources available to a wide range of services and applications we must first develop: (1) an understanding of the strengths and weaknesses of the hardware, (2) an understanding of how portions of a workload might be decomposed into tasks for offload, (3) abstractions to allow code portability on the heterogeneous components. We take a look at existing hardware trends through a survey of existing and original work to examine how new compute-in-network show promise, where they fall short and how HPC might evolve to take advantage of them.

Keywords: Data Processing Unit · Smart NIC · Infrastructure Processing Unit · Compute-in-Network.

1 Introduction

Facilities deploy high-end systems for a variety of reasons, including (1) faster time to solution, (2) higher-precision support (3) better power efficiency (4) difficulty of problem decomposition (as seen in the ML landscape) and (5) achieving unprecedented simulation scale. Given the coming ubiquity of cheap computational resources, the question naturally arises - how can ancillary services and infrastructure be profitably offloaded to these compute resources?

Over the last 10 years compute power has continued to become more affordable. The cost of a transistor and FLOPS per dollar have continued to decrease in recent

years. However, at any point in time there is a range of performance per dollar that can be obtained. At one end, high-end (premium) hardware offers greater density, additional I/O connectivity and more sophisticated error correction, albeit at a higher price. On the other end of the spectrum, lower-end (freemium) components provide moderate computation at a value price point. And while they may not have all of the features of the high-end parts (e.g. increased failure rates, decreased memory capacity, or reduced precision) we are seeing a proliferation of these compute resources scattered throughout HPC systems, including in networking and storage.

Traditionally, everything was run on a homogeneous CPU system. As Moore's Law slowed down, and Dennard scaling ended, this was followed by systems that offloaded portions of codes to accelerators like GPUs. The next logical step is to push the code and software that can effectively use them onto these lower performance components. This will leave the premium hardware to the applications and software that can leverage the most benefit.

In this paper we begin by reviewing the current landscape providing background and further comparing the qualities of premium vs. freemium components. We begin with a focus on Data Processing Units (DPUs)[1] and how cheap compute is creating new opportunities. We discuss hardware design points, uses of the hardware, the perceived strengths and weaknesses as well as open challenges. We follow this with a brief discussion of the memory market and potential solutions for addressing the growing cost of system memory. Last, we summarize how cheap compute everywhere may alter the HPC landscape as we must orchestrate distributed heterogeneous resources for complex workflows.

2 Motivation

2.1 Premium and Freemium

In today's hardware landscape there is often a breadth of choice when it comes to balancing power, density, performance and cost. More capable processor SKUs command a premium, but typically don't provide the most efficient value in $/FLOP. In some cases the lower-end (or freemium) components may have started as premium components that encountered a defect in manufacturing. The manufacturers then disable the defective portions of the unit and sell the remainder at a discount. Customers that purchase these models are, in effect, riding on the coat tails of the customers purchasing premium SKUs. In other examples we see how economies of scale (such as the cell phone and gaming market) have led to massive increases in performance per dollar. We see these price-to-performance gaps across both GPU and CPU architectures.

The price premium for top-tier performance provides motivation for offloading ancillary software and infrastructure. If we can offload these workloads, the remaining code running on the premium hardware benefits by reducing contention for shared resources. An ancillary offload makes sense, when the workload is either (1) not in the critical performance path or (2) does not make use of premium hardware capabilities (e.g. 64 bit precision, or vector and tensor units). This ancillary offload may be on

[1] DPU, Smart NIC, and IPU are used interchangeably.

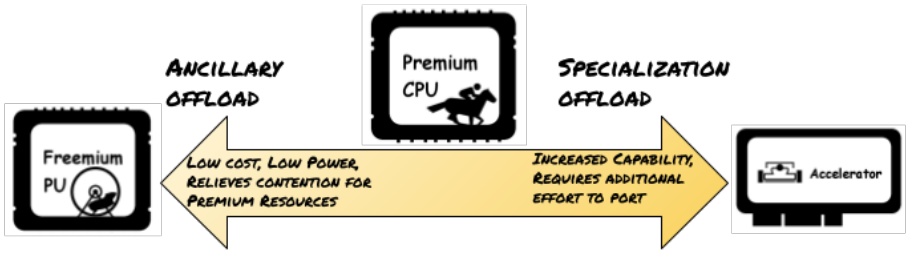

Fig. 1. In the spectrum of offload, HPC has pivoted towards specialization offload such as GPUs to provide additional capability to a subset of the workloads. Moving in the opposite direction is the possibility of ancillary offload which isolates performance critical workloads from those that could run on less capable hardware.

the same CPU as the Premium CPU (e.g. big.LITTLE) or distributed more broadly, such as on a NIC. The benefit of having the ancillary services on the same CPU is tighter coupling. The downside is that this takes up chip area on the high density premium CPU and there are challenges of having multiple clock and power domains within an integrated circuit.

2.2 Data movement

Another reason to distribute cheap compute throughout the system is to reduce the cost of data movement, which can improve both throughput and latency [56]. Placing computation throughout the data path reduces the need for additional transfers and hops across the network and through complex memory hierarchies, resulting in improved latency. By placing the compute next to the data we may be able to take advantage of higher throughput links while reducing the load on shared network resources.

Though, just because the compute is closer physically to the data does not guarantee it will achieve improved performance. Even if positioned closer to the data, low power, cheap cores running a full OS may not be able to provide enough processing power to facilitate increased throughput and lower latency. From a throughput perspective, a single core does not have enough cycles to keep up with 100, 200, 400 Gbps or greater network speeds we are seeing in HPC [19, 38]. Because of the deep memory hierarchy on the host CPU, it would take many cores to saturate the NIC peak message rate (IB HDR is 5 ns per message). Figure 2 illustrates how a single Broadwell core is unable to come close to saturating the Mellanox EDR 100GBps NIC. Furthermore the lower power and cheaper A72 incurs an additional delay of around 250 ns per message. The penalty of using lightweight cores may not provide a data locality benefit since link latency is approximately 5ns per meter of fiber plus an additional 100-200 ns per switch.For this reason, many approaches targeting throughput and latency improvements are built around lightweight and specialized line-rate offload [27].

However, offload hardware need not be devoted entirely to line rate processing. Cheap cores may be part of a data pipeline in DPUs that allows for variable performance offload as seen in work by Lin et al [35]. In their design, separate paths

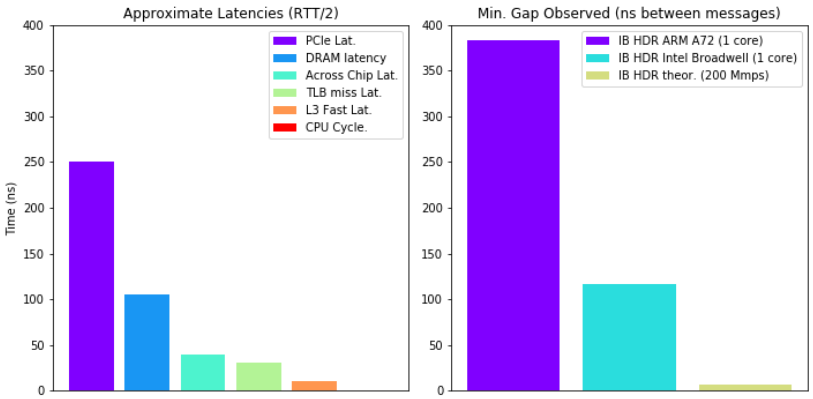

Fig. 2. Approximate latencies (half round trip) for different interconnects and memory alongside the time (gap) between sending consecutive messages over an Infiniband network with different CPUs. Latency values are approximates from [45, 8]. Best case gap was recorded for single core ib_write_bw tests using Arm A72 and Intel Broadwell E5-2697A CPUs connected to an IB HDR NIC.

in the switching fabric of the NIC allow for line rate traffic to proceed through a lightweight reconfigurable match table. Depending on the match/action, traffic may (1) forwarded directly out the NIC or (2) proceed to a buffer and scheduling unit where more complex and potentially non-line rate offload chains may be composed on the NIC. Additionally, if we look at current trends for hardware such as the Mellanox Bluefield [46], Xilinx Alveo [60] and Broadcom Stingray [14] series DPUs, we see that there is a set of inexpensive Arm cores on the NIC (ancillary offload) as well as a set of accelerator units (e.g. compression, encryption, regular expression engines or FPGAs) which provide specialization offload.

In summary, the benefits of reduced data movement can provide a benefit as both (1) a direct performance benefit by reducing latencies and providing greater throughput and (2) indirect performance benefit by reducing contention for shared network resources. In some cases it may create opportunities for performance gains, but the other benefit is from freeing premium resources.

3 Hardware Design Points

NICs began as a card to provide a connection between the computer and an external network. Historically, a substantial amount of network processing was done on the host CPU, but over time more functionality shifted onto the NIC, such as protocol offload [51], Remote Direct Memory Access (RDMA) and programmable match and action engines [28]. Smart NICs have been a part of the HPC vocabulary for over twenty years [51, 18], but recently there has been a revival of interest in the topic with Smart NIC, or in vogue data processing units (DPU) and Infrastructure Processing Unit (IPU), being announced by a variety of vendors and hyperscalers. While many

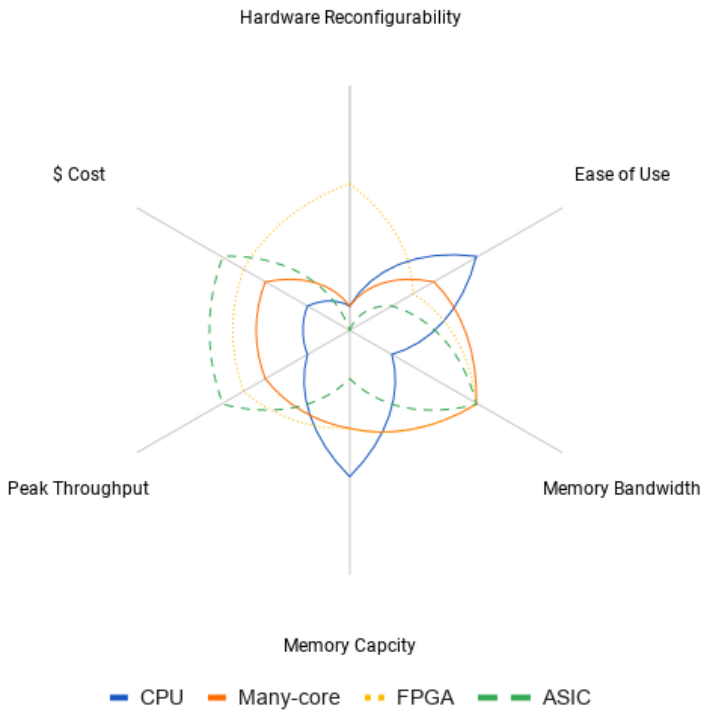

Fig. 3. Summary of different strengths and weaknesses of varied DPU architectures.

of these cards include a mixture of CPU, many-core, Field Programmable Gate Array (FPGA) or Application Specific Integrated Circuit (ASIC) components we must understand the strengths and weaknesses of each approach..

In Figure 3 we map the attributes of FPGA, CPU, many-core and ASIC hardware used in DPUs to their strengths and weaknesses. We provide rough guidelines regarding Peak Throughput, Memory Bandwidth, Ease of Use, Cost (in terms of hardware and design time), and Hardware Reconfigurability.

3.1 FPGA

A large proportion of DPUs rely on line-rate packet processing with fast turn around times using FPGAs [59, 15, 60, 17, 30, 43, 19]. This class of hardware expanded on traditional Smart NIC use cases offloading hypervisor functionality, multi-tenant Software Defined Networking (SDN), and application-specific functionality such as portions of a web search engine [53]. The strengths of these architectures are focused around high peak throughput, hardware reconfigurability and memory bandwidth. The downsides of this these cards are the absolute cost (though they may provide good performance per dollar value), the DRAM memory capacity and ease of use.

FPGAs are a common architectural choice for processing packets at line rate. This is accomplished by allowing developers to construct application-specific assembly lines that can fully leverage pipelining and parallelism inherent to a particular workload. This contrasts with a CPU where data moves between deep memory hierarchies and stalls incurred are hidden by pipelining and scheduling. FPGAs have a large amount of extremely fast SRAM. As a reference point, the Xilinx Alveo U250 targets memory bandwidth-bound code and features 38TB/s of internal SRAM bandwidth (54 MB capacity).

Regarding ease of use, these cards require a significant investment in software and programming, but have a faster development cycle than an ASIC. These approaches target specialization offload but have an emphasis on use cases that require iterative hardware design and deployment.

3.2 Many-core

Other approaches utilize a mixture of many packet and flow processing cores combined with specialized hardware blocks to perform processing for a variety of different network and data-centric flows [44, 23]. By leveraging many cores these platforms provide good peak throughput, but the hardware is not reconfigurable. These approaches still target line-rate processing, but may focus on a subset of targeted use cases (e.g. disaggregated storage platform [24]). As a reference point, the Fungible F1 DPU [23] is divided into multiple data cluster of 200 threads with full cache coherency. All of this is connected to a scheduling block, network units and host units by an on chip network. Again, there is a higher absolute cost than cheap CPUs, though the performance per dollar can be great if the application can leverage the parallelism of the architecture. Memory bandwidth is often favored in lieu of memory capacity. For this reason, these approaches typically target specialization offload.

One of the challenges of building out a many-core approach is the on-chip network that ties it all together. As the number of cores and endpoints increases, it becomes increasingly difficult to scale a non-blocking full bisection network that can facilitate line rate processing on the chip [35, 19]. Another challenge of many-core architectures is extracting sufficient parallelism from the workload to fully leverage collections of threads in a SIMD architecture.

3.3 Cheap CPUs

A third class of DPUs [14, 46, 50, 39] combine general purpose, but relatively weak, inexpensive and low power Arm cores with network and storage centric acceleration blocks. These often run a Linux operating system. In the Bluefield [46] architecture these CPUs are connected to the NIC over a PCIe switch that treats the CPU the same as the host CPU, while in other cases the CPU is more tightly integrated [50]. The benefits of these CPUs are that they are extremely cheap and easy to integrate. This means that they are commonly added to other varieties of FPGA DPUs [60] to handle exceptions, errors and non-conventional control plane functions. However, this style of CPU provides lower throughput than alternatives [36] and they may need to provide greater memory capacity for data buffering and storage with relatively

modest memory bandwidth compared to many-core and FGPA architectures. **This third class of NIC is the optimal choice for ancillary offload, since they provide a minimal of specialized and costly resources.**

3.4 ASICs and ASIC-hybrid

Throughout all of these design points are many heterogenous ASIC components such as encryption, compression and regular expression engines. These are specialized hardware units that provide excellent performance and value but only for a subset of fixed functionality. The fixed function blocks are designed around static workloads that don't change frequently. To create a good value proposition the speedups of an ASIC must offset one-time design and verification costs (both time and money). It is likely that many DPU architectures will take a hybrid approach combining ASICs for well understood, slow-to-change acceleration and flexible, lightweight cores or FPGAs for rapidly evolving use cases.

4 Target market and use cases

4.1 Opportunities better suited for specialization offload

Protocol Offload, load balancing, security and encryption: Things that used to be considered *smart* such as TCP/IP acceleration and virtualization offload (e.g. SR-IOV) are now commonplace in NICs. Newer Smart NICs have expanded the role of what is offloaded and added additional functionality for networks and packet processing [9, 31, 32, 51]. This may include things such as RDMA protocol acceleration, traffic monitoring, programmable match+action engines and HPC targeted protocol offload such as MPI tag-matching.

Virtualization offload and multi-tenancy: Hyperscalers have embraced DPU solutions to free up premium CPU cores on the host for applications and customers. Publicly announced platforms include Microsoft Catapult [19], Amazon Web Services Nitro [10], and Alibaba X-Dragon [16]. On these platforms there is an emphasis on providing as close to bare metal performance, while enabling the efficiency, security and flexibility that virtualization offers. Offload challenges here include scheduling [57], multi-tenant QoS [54], and security[10]. All of these have an emphasis on line rate processing and a majority of them rely on FPGAs for implementation/prototyping, this is because the lightweight hypervisor being offloaded is in the critical path for accessing resources and application performance.

The initial work using FPGAs at Microsoft for the Catapult Project was motivated primarily by finding alternative pathways to performance in light of the slowdown of Moore's Law. In the Catapult work [53], the authors chose FPGAs to provide a layer of computation to accelerate a search ranking engine between the host CPU and the network. FPGAs were selected because they could provide price/performance benefits in latency and throughput at line rate, while being reconfigurable as the datacenter workloads evolved. In later work, FPGAs were used to both offload and accelerate the

software defined networking (SDN) that ran on the host hypervisor[19]. For Microsoft the deployment of FPGAs makes sense, given the fleet of more than a million nodes, the development cost is easily amortized. However the HPC environment has different economies. Top HPC centers have significantly lower node counts and rather than focusing on a small number of workloads (such as search) our centers contain thousands of unique workloads with application developers running on bare-metal systems.

Virtualization and resource isolation has not been a focus in the HPC space because workloads are designed to run across multiple nodes and utilize the full node for each node allocated. However machine learning is pushing node sizes bigger, with more GPUs and resources tied up in a single node. The economics of this have bled into many recent HPC designs which now feature nodes with fatter architectures. As future systems grow even larger and more heterogeneous, we may need to leverage multi-tenancy techniques similar to cloud providers, but it is unlikely to be a job suited for cheap cores and ancillary offload.

4.2 Opportunities for ancillary offload

Compared to cases leveraging specialization offload, relatively little work has been done to motivate ancillary offload on the DPU. However there are several areas where DPUs have found success by being able to alleviate pressure for host resources.

Key value stores: In work by Liu et al. [38] a range of microservices were evaluated comparing, the Marvell CN78XX DPU with 12 MIPS cores, against more powerful host based CPUs. The work showed that complicated workloads performed poorly on the DPU, but was unique in that it highlighted the potential energy savings cheap DPU cores could provide. One of the microservices the DPU provided greater energy efficiency for was key value stores. Key value stores are fundamentally (1) a hash evaluation which provides a memory location and (2) a memory access to read/write or perform atomics on the data, both of which can be adequately handled by cheap cores. In work by [36] using the *stress-ng* benchmark, memory focused benchmarks such as mcontend performed better on Bluefield Arm cores than compared to a range of host CPUs. In work by Phothilimthana et al. [52], a DPU was used as a write-back cache for a host-based key-value store (KVS) and observed a 28-60% performance improvement. By offloading to portions of a KVS, host applications may experience less pollution of the cache due to network-induced memory contention [25]. Despite the benefits, a challenge of utilizing the DPU as a cache is maintaining a consistent state between the data that resides on the DPU and host memory.

Simple analytics: A number of simple analytics functions can be implemented on a DPU where a powerful host CPU would otherwise be overkill. These include things like K-nearest neighbor, and Spike [38], or portions of the Apache Storm [21] like multiplexing and demultiplexing operations. In the case of the latter, a 76% improvement was observed when multiplexing and demultiplexing was offloaded to the DPU compared to running the workload entirely on the host CPU [52]. In cases where analytics are sufficiently independent of host workloads, such that they can tolerate the extra hop over PCIe or network interfaces, cheap compute on DPUs can be leveraged successfully.

If the DPU processor and host memory could be more tightly coupled, additional avenues for analytics would be possible as often the penalty for moving memory across PCIe is several times greater than local memory accesses (as shown in Figure 2).

Communication and computation overlap: A third avenue for ancillary offload is providing communication and computation overlap by leveraging cheap cores on DPUs to manage complex communications on the host's behalf. In work by Bayatpour et al. [11], non-blocking all-to-all collectives are delegated to Arm cores on the Bluefield 2 DPU. In this model the host simply provides metadata information about the collective to threads on the DPU. These threads then read and write host memory using Remote Direct Memory Addressing (RDMA) to complete the iterations of the all-to-all algorithm. Because of the extra latency incurred transferring data to and from host memory to DPU cores this approach focused on message sizes greater than 16KiB. In this range, four A72 Arm cores were able to deliver performance similar to that of the host CPU. However, the real benefit of this approach is that the all-to-all operation is offloaded and the host is left able to focus on other more demanding work.

Node-local IO virtualization The proliferation of AI and other non-traditional I/O workloads on HPC systems has resulted in a tension in storage systems design between traditional parallel file systems and node-local storage. Node-local NVMe (Non-volatile Memory Express) has been shown to dramatically accelerate some workloads by localizing metadata-intensive and high-frequency accesses to the PCIe data paths within nodes, sparing the global network and parallel file system from disruption that would otherwise adversely affect other users of those shared network and file system resources. Not every HPC workload can use node-local storage because they collectively lack the coherence required by shared-file I/O. As a result, the decision of whether to provision hinges on balancing the cost of adding new components to every compute node against the fact that those components may be poorly utilized.

NVMe over Fabrics (NVMeoF) allows compute nodes to mount NVMe devices over the network, offering the benefits of localizing many expensive metadata and high-frequency I/O operations to a compute node by fully delegating namespace coherence and I/O buffering to each compute node. Using NVMeoF, it is possible to provision only a subset of nodes with node-local NVMe, and letting any node "borrow" an unused NVMe drive from a neighboring compute node on-demand. As such, systems designers can provision only a subset of compute nodes with NVMe drives to keep costs commensurate with utilization but still achieve the effect of any compute node having a node-local NVMe when needed.

The biggest challenge with this disaggregated NVMeoF model is that the node "loaning" its NVMe to another node still has to service interrupts triggered by NVMeoF activity, and its memory bandwidth must be shared between its local computational workload and the I/O workload targeting its loaned NVMe drive. SNAP [42] holds promise to break this tradeoff by allowing the entire NVMeoF stack to run entirely on the loaner node NIC, fully shielding the host from interrupts and memory contention associated with serving up NVMeoF. A borrower node uses the NVMe drive loaned by a DPU, and the "loaner" node can be completely unaffected by I/O targeting its physically local NVMe and compute its local workload without interruption.

Machine learning inference: By sitting on the network data path, DPUs provide an opportunity for deploying ML inference at the edge. A typical inference workload will load relevant features of the data, apply the weights of the model and then perform multiply-accumuate operations on the result. Many neural networks may only need 8 bit precision compared to 64 bit precision of many scientific workloads. This means that the host CPU is mismatched with the workload. A standalone GPU may provide much greater efficiency than the host CPU, but may be overkill for modest inference workloads. The standalone GPU often contains a large volume of costly on-package memory that is critical for training but less for inference. In this scenario having a less powerful GPU on the DPU may provide an opportunity for ancillary offload and leave premium GPU resources for more intensive workloads. Nvidia has recently announced the Bluefield-X [6] line of DPUs which include an A100 GPU integrated into the DPU. While this is a premium GPU it may make sense to explore future offerings with less power-hungry GPUs – similar to the approach taken with Nvidia Jetson [7]. Other companies may begin to incorporate functionality of the ARM Ethos NPUs [1] into their DPU. This is an exciting area to look forward to in the future of DPUs.

Specialization offload often receives most of the attention when researchers evaluate the benefits of DPUs. However, given the premium paid for HPC systems and the proliferation of cheap CPUs appearing on NICs we believe ancillary offload deserves greater consideration. However, in order to accelerate this development there are open questions that must be addressed.

5 Open questions

5.1 Balancing power, performance and cost

One such question is, how powerful a CPU should reside on the DPU? Current studies show that the Arm A72 on the Bluefield 2 DPU and Marvell CN913X are incapable of driving line rate speeds of 100 Gbps. This means that products relying on CPUs-on-NIC for pushing the network will need to shift towards more cores. Additionally, newer architectures may increase the number of memory channels and bandwidth. This has the downside of adding increased power and cost to the DPU. For example, Bluefield 3 is anticipated to have a 5X increase in SPECint performance over the previous-generation Bluefield 2 [47]. Similar upgrades to the Marvell line of DPUs required moving from four to thirty-two cores with the transition to the latest generation of processors. This requires increasing the TDP by 70-110 watts [41]. How much processing power to shift over to the DPU from the host is an open question that requires the community to engage in successive evaluations to determine whether future DPUs are best suited for ancillary or specialization offload. Determining exactly where the delineation will fall between host CPU, NIC, DPU and GPU is difficult to predict.

5.2 The right level of abstraction

One of the biggest hurdles of developing code for a DPU is the interfaces for programming them. One of the reasons for this is the breadth of architectural design points

that DPUs span and the range of use cases. Many interfaces are low-level, focused on processing packets, designed to be stateless and perform at line rate, such as P4 [12]. A number of efforts have attempted to ease the development burden by providing abstraction layers on top [34, 52]. Interfaces such as the Data Plane Development Kit (DPDK) [4] abstract common packet processing functions in a run-to-completion model. Other efforts introduce abstract machine models to allow for portability across varying architectures. In sPIN [27] different header, payload and completion handlers are associated with different connections to facilitate flexible offload, but has the constraint that the workload must not obstruct line rate processing. The INCA model [55] supports additional functionality by removing the deadline constraints of line-rate processing, but requires the code running on the DPU be preemptable. In an environment where cheap cores are everywhere, the INCA model facilitates ancillary offload and allows for more varied types of computation than prior approaches.

There is a huge gap between these abstractions and a fully featured Linux environment on a Bluefield DPU, where process scheduling is provided by the operating system and codes are written at a higher level. In order to encompass the total range of functionality possible, we are seeing the continued evolution and layering of abstraction on top of performance-focused, lower-level interfaces. This will allow systems and communication library designers to develop powerful tools that can be composed and leveraged by higher level user applications. Work by Liu et al [37] developed the iPipe framework to provide an actor-based model that supports multi-tenancy, scheduling and hardware heterogeneity. Cloud-focused DPU company, Pensando has proposed portable APIs that target popular cloud services, but has noted that there is a lack of standardization for controller interfaces, operating systems, and data paths [49]. Within the Unified Communication Framework there is an effort to create a Smart NIC API (OpenSNAPI) [22] which will allow developers to leverage compute cores in the network. Nvidia is developing the DOCA SDK [48]. With the DOCA SDK Nvidia's goal is to create a single portal for harnessing the DPU in a manner akin to CUDA for GPUs. We see similar approaches from Broadcom [13] and Marvell [40] where each of these SDKs consolidates lower level functionality. **Two of the questions around this are how much will these SDKs differ as products try to differentiate themselves and what will the hurdles be for program portability?**

Other companies take more of an appliance-based approach, where services are accessed via REST APIs [24] or blocks of functionality can be downloaded from product specific app-stores [61]. While this approach may provide great performance it isn't necessarily a good fit for ancillary offload in an HPC environment where the emphasis is on ease of programming and portability rather than raw performance.

When considering which approaches are likely to gain traction with ancillary offload, DPUs that contain cheap CPUs are attractive because they provide one of the simplest programming environments. Since they run a full Linux OS, it is possible to log onto and compile code much like any other node in the system. The downside of running a full system software stack means that cheap cores incur greater overheads on performance. However, this is not as much of a concern for ancillary offload.

Finally, DPUs create an additional domain for computation, which creates additional challenges for memory interfaces and migration of data in much the same way

GPUs and CPUs experience today with unified virtual memory. Adding in a third processing unit to the mix will require additional coordination between the CPU, GPU and DPU.

5.3 The memory problem

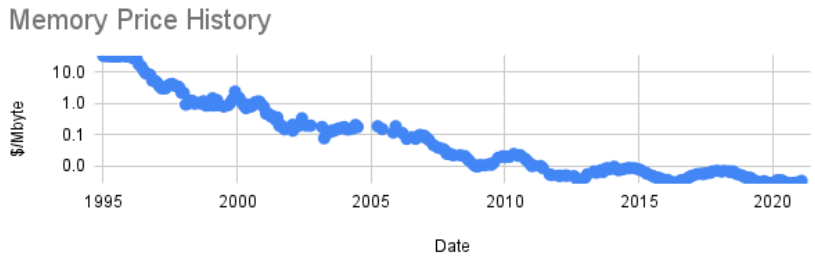

Fig. 4. Historic memory prices per byte. Memory prices per byte have not continued to drop as the same rate of computation, resulting in memory taking up larger percentages of system budgets. Data was originally from John C. McCallum of http://jcmit.net.

While FLOPS per dollar has continued to become more affordable over time, memory prices per byte have been relatively flat or occasionally spiked depending on demand as shown in Figure 4. 3D technologies such as HBM have raised the cost of memory even further for the workloads that require greater bandwidth than traditional DDR can provide.

Exacerbating this problem is the fact that each time an accelerator such as a GPU is added into a system we are paying for extra memory in addition to the host memory. This is because host DRAM often serves as a launching point for GPU kernels before data is transferred to HBM, but then may go unused during the execution of the GPU kernel. DPUs threaten to create another instance where additional memory is required. Futhermore, having multiple memory spaces complicates programming and necessitates additional data movement.

One idea would be to create a unified memory space between the DPU, CPU and GPU with coherency supported in hardware. This would allow for a reduction of device attached memory, but the interfaces to support this efficiently are (1) still under development and (2) the market has not yet chosen a standard. PCIe is insufficient for these goals since it suffers substantial latency overheads [29] compared to a local memory access and the protocol does not provide the abstractions necessary for complex interactions across multiple memory domains. In work by Li et al [33], PCIe presented a number of challenges to implementing a key-value cache on DPUs such as a limited number of requests due to credit-based flow control, and a limited number of PCIe tags to differentiate between DMA read response that may arrive out of order.

The need for a coherent interface between processors and accelerators has been identified in the past. Coherent Accelerator Processor Interface (CAPI) [58] is the

protocol supported by IBM. CAPI provides coherency and virtual addressing capabilities that more tightly couple the accelerator to IBM Power Processors. In joint development between Nvidia and IBM, ORNL's Summit [26] system leveraged a tighter coupling between its Power 9 CPUs and Nvidia GPUs by having multiple NVLink [20] connections going directly on-die. Nvidia has bypassed the limitations of PCIe by utilizing NVLink for its DGX products. NVLink provides high bandwidth and low latency between Nvidia devices with hardware coherency and load/store semantics. NVLink is a central component of enabling machine learning training by composing multiple GPUs and creating large memory spaces for problems that are difficult to decompose. As companies expand their offerings to cover greater breadth we would like to see tighter integration between CPU, GPU and future DPUs.

Two other technologies, Compute Express Link (CXL) [3] and Cache Coherent Interconnect for Accelerators (CCIX) [2], both provide opportunities to address problems with managing memory by building on PCIe. CXL has three different levels of capabilities that hardware may support: CXL.io, CXL.cache and CXL.mem. CXL.io is similar to PCIe with a few extensions for initialization, device discovery and memory mapped register access. CXL.cache targets applications that require basic coherency protocols between a CXL compliant accelerator's cache and host-attached memory. This is beneficial for applications that want to leverage CXL supported atomic operations, and allow for flexible ordering models. CXL.mem provides additional functionality to CXL.cache by allowing the host to access device memory without incurring overheads that offset the benefit of having an attached device or DPU. This relies on host CPU to provide management of device memory and manage coherency via a *Home Agent* and *Coherency Bridge*. Ownership of memory can favor the host or device depending on different *bias* assigned. The downside to this approach is that it is host-centric and doesn't readily facilitate device to device memory transactions. CCIX is a competing standard that differs by allowing the Home Agent to reside on the accelerator rather than limiting that location to solely the host. In either case having the reduced latency and coherency that CXL and CCIX support greatly increases the variety of work that could be offloaded to a DPU and reduces the number of required transactions over PCIe.

One of the biggest hurdles is that there are many competing standards today which limit portability between future CPUs and devices which must provide mutual support for a standard.

5.4 Tying it all together

In Figure 5, we provide an example of the different places that compute-in-network may be applied throughout a HPC center such as NERSC . Green stars indicate where offload may be placed and text pop-outs exemplify types of offload that could be enabled. Determining how these resources are accessed and shared across hundreds of workloads creates new challenges for HPC workflows. In contrast to typical HPC applications running on a single system such as Perlmutter [5], these workflows may span (1) a larger variety of heterogeneous hardware, (2) multiple networks, and (3) may sit beyond the reach of traditional resource management and scheduling systems. One solution is to implement these offload units as appliances with user interaction limited to specific purposes similar to data-transfer nodes on today's

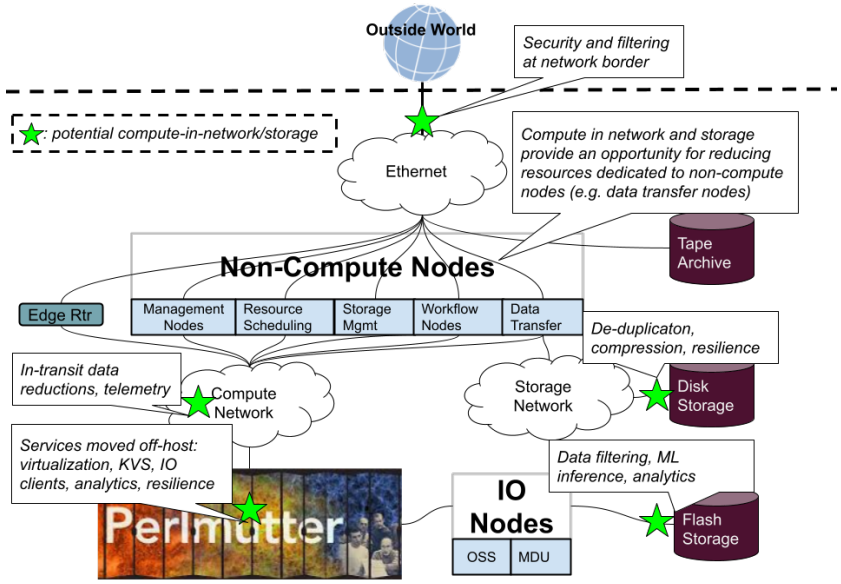

Fig. 5. Opportunities for compute-in-network/storage to offload tasks (both ancillary and accelerated) in a typical HPC datacenter. Green stars represent different points where compute-in-NIC, fabric or storage could be applied with example uses called out. Potential benefits include a (1) reduced data movement, (2) reduction of required non-compute nodes, IO nodes and (3) reduced contention for premium resources on the compute cluster.

systems. But the more interesting discussion is focused on how we can make these programmable, flexible and accessible to a wide ranger of users and workflows, bridging the HPC-to-workflow gap. New methods for orchestrating and facilitating data-flow and communication may be necessary and we may need to leverage best practices from cloud environments for this. For example we may need a stable definition for Quality of Service (QoS) throughout the data center, so that an urgent HPC workload may run at a higher priority as it traverses the many locations of compute-in-network. This will require efforts in scheduling, program portability, and multi-tenancy. If we do this successfully we reduce the number of non-compute and service nodes, while opening up new possibilities for scientific workflows and machine learning.

6 Conclusions

Heterogeneity and specialized units are now part of the HPC landscape. In the future we want to reserve premium resources for high priority workloads, while taking advantage of the cheap compute that is being distributed throughout the network. For these types of resources we propose ancillary offload, and an effort should be made in the HPC community to identify the opportunities in hardware and software to leverage this strategy. Ancillary workloads are the types of workloads that can tolerate

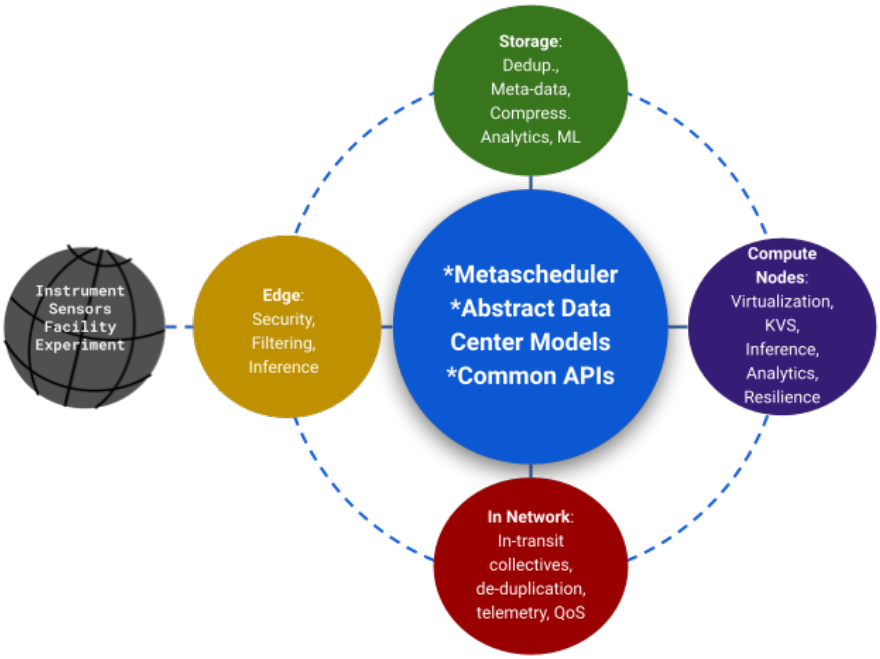

Fig. 6. Abstract view of cheap compute in the data center (edge, storage, traditional compute nodes and in-network) and the functionality needed (center) to tie it all together.

the performance penalty that may occur because of (1) lower frequency or fewer cores, (2) reduced memory capacity/bandwidth and (3) additional latency across an on-node interconnect. Already identified opportunities include: key-value stores, node-local storage virtualization, analytics and inference, but many more exist. We show in Fig. 6, that to fully leverage the hardware we must expand beyond abstract machine models and develop abstract data center models that enable center-wide scheduling and runtimes. Abstract data center models must efficiently represent heterogenous hardware characteristics and match them to appropriate portions of a workflow with an understanding of the quality of service required by the workload. In order to make this vision a reality we must address future challenges of developing programming abstractions that support a multitude of architectures and use cases, multi-tenancy, and the memory problem. If we do this successfully, we can reduce the number of non-compute and service nodes, alleviate the noise on premium compute units and enable new possibilities for scientific workflows and machine learning.

References

1. Arm Ethos, https://developer.arm.com/ip-products/processors/machine-learning/arm-ethos-n
2. Cache coherent accelerator interface, https://www.ccixconsortium.com/

3. Compute express link, https://www.computeexpresslink.org/
4. Data plane development kit, https://www.dpdk.org/
5. NERSC perlmutter, https://www.nersc.gov/systems/perlmutter/
6. Nvidia DPU, https://www.nvidia.com/en-us/networking/products/data-processing-unit/
7. Nvidia embedded systems, https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/
8. AnandTech: AMD 3rd generation Milan review, https://www.anandtech.com/show/16529/amd-epyc-milan-review/4
9. Arashloo, M.T., Lavrov, A., Ghobadi, M., Rexford, J., Walker, D., Wentzlaff, D.: Enabling programmable transport protocols in high-speed NICs. In: 17th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 20). pp. 93–109 (2020)
10. AWS: AWS Nitro, https://aws.amazon.com/ec2/nitro/
11. Bayatpour, M., Sarkauskas, N., Subramoni, H., J. Hashmi, D.P.: BluesMPI: Efficient mpi non-blocking alltoall offloading designs on modern BlueField smart NICs (June 2021)
12. Bosshart, P., Daly, D., Gibb, G., Izzard, M., McKeown, N., Rexford, J., Schlesinger, C., Talayco, D., Vahdat, A., Varghese, G., et al.: P4: Programming protocol-independent packet processors. ACM SIGCOMM Computer Communication Review **44**(3), 87–95 (2014)
13. Broadcom: Broadcom SmartNIC SDK, https://docs.broadcom.com/doc/5880X-UG30X
14. Broadcom: Stingray PS225, https://docs.broadcom.com/doc/PS225-PB
15. Cisco: Cisco Nexus smartnic+ V9P data sheet, https://www.cisco.com/c/en/us/products/collateral/interfaces-modules/nexus-smartnic/datasheet-c78-743830.html
16. Cloud, A.: X-dragon, https://www.alibabacloud.com/blog/introducing-the-sixth-generation-of-alibaba-clouds-elastic-compute-service_595716
17. Data, A.: ADM-PCIE-9V3, https://www.alpha-data.com/pdfs/adm-pcie-9v3.pdf
18. Dufey, J.P., Jost, B., Neufeld, N., Zuin, M.: Event building in an intelligent network interface card for the lhcb readout network. In: 2000 IEEE Nuclear Science Symposium. Conference Record (Cat. No. 00CH37149). vol. 3, pp. 26–50. IEEE (2000)
19. Firestone, D., Putnam, A., Mundkur, S., Chiou, D., Dabagh, A., Andrewartha, M., Angepat, H., Bhanu, V., Caulfield, A., Chung, E., et al.: Azure accelerated networking: Smartnics in the public cloud. In: 15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18). pp. 51–66 (2018)
20. Foley, D., Danskin, J.: Ultra-performance pascal gpu and nvlink interconnect. IEEE Micro **37**(2), 7–17 (2017)
21. Foundation, A.S.: Apache Storm, https://storm.apache.org/
22. Framework, U.C.: OpenSNAPI, https://www.ucfconsortium.org/projects/opensnapi/
23. Fungible: Fungible F1 data processing unit, https://www.fungible.com/wp-content/uploads/2020/08/PB0028.01.02020820-Fungible-F1-Data-Processing-Unit.pdf
24. Fungible: Fungible storage cluster, https://www.fungible.com/wp-content/uploads/2020/12/PB0020.04.02021214-Fungible-Storage-Cluster-FSC-Disaggregated-Storage-Platform.pdf
25. Groves, T.L., Grant, R.E., Gonzales, A., Arnold, D.: Unraveling network-induced memory contention: Deeper insights with machine learning. IEEE Transactions on Parallel and Distributed Systems **29**(8), 1907–1922 (2017)
26. Hanson, W.A.: The coral supercomputer systems. IBM Journal of Research and Development **64**(3/4), 1–1 (2019)

27. Hoefler, T., Di Girolamo, S., Taranov, K., Grant, R.E., Brightwell, R.: spin: High-performance streaming processing in the network. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. pp. 1–16 (2017)
28. Ibanez, S., Brebner, G., McKeown, N., Zilberman, N.: The p4-¿ netfpga workflow for line-rate packet processing. In: Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. pp. 1–9 (2019)
29. Ibanez, S., Shahbaz, M., McKeown, N.: The case for a network fast path to the cpu. In: Proceedings of the 18th ACM Workshop on Hot Topics in Networks. pp. 52–59 (2019)
30. Intel: Intel FPGA programmable acceleration card N3000, https://www.intel.com/content/www/us/en/programmable/products/boards_and_kits/dev-kits/altera/intel-fpga-pac-n3000/overview.html
31. Kaufmann, A., Peter, S., Sharma, N.K., Anderson, T., Krishnamurthy, A.: High performance packet processing with flexnic. In: Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems. pp. 67–81 (2016)
32. Le, Y., Chang, H., Mukherjee, S., Wang, L., Akella, A., Swift, M.M., Lakshman, T.: Uno: Uniflying host and smart nic offload for flexible packet processing. In: Proceedings of the 2017 Symposium on Cloud Computing. pp. 506–519 (2017)
33. Li, B., Ruan, Z., Xiao, W., Lu, Y., Xiong, Y., Putnam, A., Chen, E., Zhang, L.: Kv-direct: High-performance in-memory key-value store with programmable nic. In: Proceedings of the 26th Symposium on Operating Systems Principles. pp. 137–152 (2017)
34. Li, B., Tan, K., Luo, L., Peng, Y., Luo, R., Xu, N., Xiong, Y., Cheng, P., Chen, E.: Clicknp: Highly flexible and high performance network processing with reconfigurable hardware. In: Proceedings of the 2016 ACM SIGCOMM Conference. pp. 1–14 (2016)
35. Lin, J., Patel, K., Stephens, B.E., Sivaraman, A., Akella, A.: {PANIC}: A high-performance programmable {NIC} for multi-tenant networks. In: 14th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 20). pp. 243–259 (2020)
36. Liu, J., Maltzahn, C., Ulmer, C., Curry, M.L.: Performance characteristics of the bluefield-2 smartnic (2021)
37. Liu, M., Cui, T., Schuh, H., Krishnamurthy, A., Peter, S., Gupta, K.: Offloading distributed applications onto smartnics using ipipe. In: Proceedings of the ACM Special Interest Group on Data Communication, pp. 318–333 (2019)
38. Liu, M., Peter, S., Krishnamurthy, A., Phothilimthana, P.M.: E3: Energy-efficient microservices on smartnic-accelerated servers. In: 2019 {USENIX} Annual Technical Conference ({USENIX}{ATC} 19). pp. 363–378 (2019)
39. Marvell: Marvell CN913X, https://www.marvell.com/content/dam/marvell/en/public-collateral/embedded-processors/marvell-infrastructure-processors-octeon-tx2-cn913x-product-brief-2020-02.pdf
40. Marvell: Marvell octeon SDK, https://www.marvell.com/content/dam/marvell/en/public-collateral/embedded-processors/marvell-octeon-tx2-sdk-solutions-brief.pdf
41. Marvell: Marvell Octeon TX2 Press Deck, https://www.marvell.com/content/dam/marvell/en/company/media-kit/infrastructure-processors/marvell-octeon-tx2-press-deck.pdf
42. Mellanox: NVMe SNAP, https://www.mellanox.com/files/doc-2020/sb-mellanox-nvme-snap.pdf
43. NetFPGA: Netfpga sume, https://netfpga.org/site/#/systems/1netfpga-sume/details/
44. Netronome: Netronome NFP-4000 flow processor, https://www.netronome.com/media/documents/PB_NFP-4000-7-20.pdf

45. Neugebauer, R., Antichi, G., Zazo, J.F., Audzevich, Y., López-Buedo, S., Moore, A.W.: Understanding pcie performance for end host networking. In: Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication. pp. 327–341 (2018)
46. Nvidia: Nvidia Bluefield-2 DPU, https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/documents/datasheet-nvidia-bluefield-2-dpu.pdf
47. Nvidia: Nvidia Bluefield-3 DPU, https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/documents/datasheet-nvidia-bluefield-3-dpu.pdf
48. Nvidia: Nvidia DOCA SDK, https://developer.nvidia.com/networking/doca
49. Pensando: The need for standardization, https://pensando.io/the-need-for-standardization-in-a-thriving-network-ecosystem/
50. Pensando: Pensando DSC-100, https://pensando.io/wp-content/uploads/2020/03/Pensando-DSC-100-Product-Brief.pdf
51. Petrini, F., chun Feng, W., Hoisie, A., Coll, S., Frachtenberg, E.: The quadrics network (qsnet): high-performance clustering technology. In: HOT 9 Interconnects. Symposium on High Performance Interconnects. pp. 125–130 (2001). https://doi.org/10.1109/HIS.2001.946704
52. Phothilimthana, P.M., Liu, M., Kaufmann, A., Peter, S., Bodik, R., Anderson, T.: Floem: A programming system for nic-accelerated network applications. In: 13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18). pp. 663–679 (2018)
53. Putnam, A., Caulfield, A.M., Chung, E.S., Chiou, D., Constantinides, K., Demme, J., Esmaeilzadeh, H., Fowers, J., Gopal, G.P., Gray, J., et al.: A reconfigurable fabric for accelerating large-scale datacenter services. In: 2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA). pp. 13–24. IEEE (2014)
54. Radhakrishnan, S., Geng, Y., Jeyakumar, V., Kabbani, A., Porter, G., Vahdat, A.: {SENIC}: Scalable {NIC} for end-host rate limiting. In: 11th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 14). pp. 475–488 (2014)
55. Schonbein, W., Grant, R.E., Dosanjh, M.G., Arnold, D.: Inca: in-network compute assistance. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. pp. 1–13 (2019)
56. Shi, H., Lu, X.: Triec: Tripartite graph based erasure coding nic offload. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. SC '19, Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3295500.3356178, https://doi.org/10.1145/3295500.3356178
57. Stephens, B., Akella, A., Swift, M.: Loom: Flexible and efficient {NIC} packet scheduling. In: 16th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 19). pp. 33–46 (2019)
58. Stuecheli, J., Blaner, B., Johns, C., Siegel, M.: Capi: A coherent accelerator processor interface. IBM Journal of Research and Development **59**(1), 7–1 (2015)
59. Technology, A.: ANIC host cpu offload features overview, https://accoladetechnology.com/whitepapers/ANIC-Features-Overview.pdf
60. Xilinx: Alveo SN1000 data sheet, https://www.xilinx.com/support/documentation/data_sheets/ds989-sn1000.pdf
61. Xilinx: Xilinx app-store, https://www.xilinx.com/products/app-store/alveo.htm