Seminar Report

In-Network Computing: State-of-the-art

Tim Dettmar

MatrNr: 26327113

Supervisor: Jack Ogaja

Georg-August-Universität Göttingen
Institute of Computer Science

March 30, 2023

Abstract

The demand for high-performance networks has significantly increased over time, especially with the increasing sizes, compute demands, and popularity of big data analytics. To that end, traditional network architectures and protocols have resulted in limitations in throughput and efficiency. In-network computing claims to address many of the issues related to data movement in high-performance computer networks in an efficient manner, without the need to significantly change workflows for system engineers and users alike. Existing in-network computing solutions on the market are compared and contrasted with incumbent options in order to evaluate their merits in high-performance computing tasks, as well as possible drawbacks.

Contents

List of Figures

List of Abbreviations

HPC	High-Performance Computing
RDMA	Remote Direct Memory Access
DPU	Data Processing Unit
IPU	Infrastructure Processing Unit
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
IP	Internet Protocol
ASIC	Application-Specific Integrated Circuit
FPGA	Field-Programmable Gate Array
SHARP	Scalable Hierarchical Aggregation and Reduction Protocol
ML	Machine Learning
AI	Artificial Intelligence
TOE	TCP Offload Engine
API	Application Programming Interface
NIC	Network Interface Card
MPI	Message Passing Interface
CPU	Central Processing Unit
SoC	System-on-Chip
AN	Aggregation Node
EN	End Node
NVMe	Non-Volatile Memory Express
NVMeoF	NVMe over Fabrics
OvS	Open Virtual Switch
AHCI	Advanced Host Controller Interface
IDE	Integrated Drive Electronics
SCSI	Small Computer Systems Interface
IOPS	Input/Output Operations per Second
HTTP	Hypertext Transfer Protocol

1 Introduction

Traditionally, network hardware has served a simple purpose- to move packets from one location to another [KR22]. To an end-user, network hardware consists of a series of black boxes connecting machines to each other and to the wider Internet. However, the combination of ever-increasing dataset sizes, faster networking, and the end of Moore's law have given rise to network hardware with sophisticated capabilities far beyond simple packet forwarding. This report delves into the history of in-network computing, its uses in high-performance data analytics, and the current state-of-the-art.

1.1 Performance Issues in HPC

New technologies that improve communication and computation throughput and reduce latency are often first adopted in HPC. For instance, while many consumer-grade devices rely on Wi-Fi or gigabit networking [KR22], HPC-focused networking hardware can be as fast as 800Gbps at the time of writing [NVIg]. Tasks such as weather forecasting require significant resources which far outstrip those available on a single node. As such, clusters of computing nodes are used to increase overall compute power. However, because the datasets involved in HPC can be extremely large, one needs fast networking in order to reduce message bottlenecks and effectively take advantage of the available processing power in the cluster. Amdahl's law [Amd67] states that the maximum speedup of a workload is limited by what cannot be parallelized- and slow networks can be a significant part of this equation. In a talk by John D. McCalpin, it is demonstrated that networking performance has fallen far behind raw compute performance [McC]. In addition, project budgets, energy, and physical space are not infinite. Getting the most out of the available hardware is thus beneficial in order to decrease the consumption of these limited resources, or to increase utilization and the return on investment.

The slowing of processor advancements due to the increased difficulty of shrinking lithography techniques over time have also forced the industry to look beyond faster processors alone for performance enhancements [TW17]. Instead, optimization techniques in networking hardware were explored in an attempt to both increase overall throughput, but also increase energy and space efficiency. Research and development in this field have brought technologies such as SHARP, RDMA, and DPUs - which will be further defined and explored in this report.

1.2 Optimization Targets

According to a profiling study of users of Message Passing Interface (MPI), middleware libraries designed for communication in High-Performance Computing (HPC) performed by Rolf Rabenseifner [Rab], collective communication routines were invoked in over 90% of jobs and consumed a significant majority of CPU time, mostly as a result of waiting for synchronization to complete. By definition, increasing throughput reduces the time for data to be transferred, and reducing communication latency also reduces the time for synchronization to complete, especially for small messages where raw throughput performance is less important (e.g., a barrier operation).

2 Fixed-Function Network Offload

Early network cards can be considered little more than simple signal converters. The host CPU would construct packets to be sent over the network, and the network adapter handled the encoding and decoding of data at the physical, i.e. electrical signal, layer [Ros13]. As Ethernet-compatible cards grew faster and more popular over time, the overhead of the host CPU performing the construction, check-summing, and payload handling of every packet became more apparent, and would reduce application and network performance under heavy load as resource contention occurred. Network adapters thus became more advanced to address the demand for higher and more efficient network performance, both for consumer and professional use cases.

Modern enterprise NICs are typically capable of performing stateless checksum offload, though many support full-stack protocol offloads.

2.1 Full-Stack Offload

Hgh-level protocols such as Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) are designed to abstract much of the lower-level functionality of the network away from the programmer. The primary advantages of this method are that the Application Programming Interface (API) provided to the programmer is relatively simple; after connection establishment, the sending or receiving of data can be done in a single system call. While this has proven to be robust enough to be used on the Internet in the form of Hypertext Transfer Protocol (HTTP), for instance [NFB96], the demands of the HPC space have exceeded the limits of what these protocols are able to provide. Fundamentally, as the user is not required to manage their own buffers and message queues, this task is left to the kernel and results in unnecessary memory copies, and thus an increase in consumed memory bandwidth and latency [Cla+89].

Two main approaches appeared in an attempt to address this issue. The most obvious solution would be to offload the entire TCP/UDP stack to the network card, while retaining the same socket interface to the programmer. This approach came to be known as TCP Offload Engine (TOE), which has, for all intents and purposes, failed. Linux has never integrated TOE code into the upstream kernel, and Microsoft has deprecated their support for TOE [Wil18]. The specific reasons as to why are numerous; critically, TOE implementations failed to deliver on performance and scalability promises [Mog03].

The solution now widespread across the HPC space came in the design of new protocols specifically designed to address the limitations of existing networking APIs, moving more of the computing associated with data transfer onto network adapters. These protocols are categorized under the general term Remote Direct Memory Access (RDMA). Compared to TOE and basic networking offloads (e.g., checksum offload), these APIs are significantly lower-level than incumbents TCP and UDP. Rather than handing off control to the kernel with every data transfer, RDMA-enabled programs communicate directly with the Network Interface Card (NIC). This allows for the direct movement of data to and from user-defined buffers without Central Processing Unit (CPU) involvement, reducing memory bandwidth, CPU utilization, and latency [Ros13]. RDMA-capable technologies such as InfiniBand and Omni-Path make up the majority of high-performance interconnects in the TOP500 supercomputer rankings [TOP].

An additional benefit of being able to offload these operations is that, even if the CPU has an abundance of free cycles, switching between kernel and user space to perform data transfer tasks results in context switching. According to Jeffrey C. Moful and Anita Borg, context switches have a cost of thousands of CPU cycles, and can result in increased latency due to increased cache miss rates [MB91]. It thus follows that especially for programs which require high message rates, the number of context switches results in a significant number of wasted cycles neither performing useful work nor transferring data in the kernel.

The integration of these offloads into high-level APIs such as MPI [Ope] have allowed for easy adoption of these technologies despite the difficulty in programming directly to them. One could consider RDMA to MPI as machine code is to C; few tend to program directly for the former. In addition, domain-specific libraries such as GROMACS may abstract communication operations even further on top of MPI. Thus, for end users that program for these applications, there are little to no changes required to benefit from RDMA technologies, unless one needs the fine-grained control provided by the underlying RDMA library (for example, if extreme optimization measures are needed).

2.1.1 Performance

Due to the low overhead and direct communication with the NIC, RDMA excels both in terms of latency and maximum throughput. On the tested GWDG cluster nodes equipped with Omni-Path networking, an uplift of up to 10x in terms of throughput, and a reduction of up to 10x in latency was observed. Crucially, only the use of RDMA offload was able to bring performance closer to the actual link speed of 100 Gbit/s. TCP achieved a peak bandwidth of 16.7 Gbps while PSM (Omni-Path) achieved a peak of 93.4 Gbps.

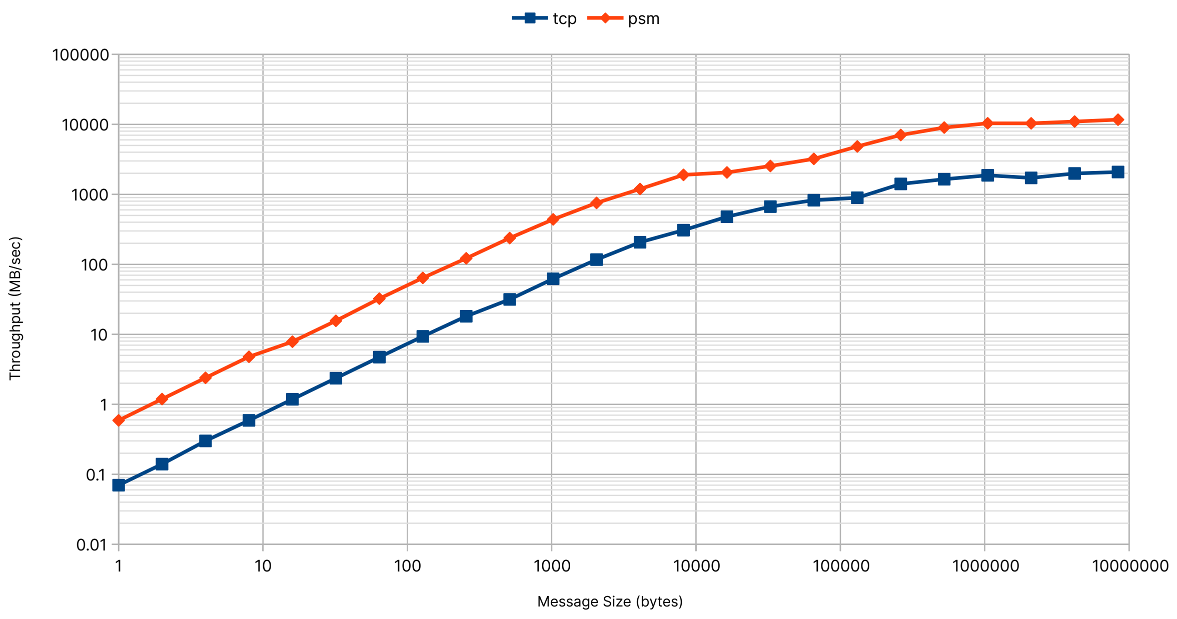

Figure 1: TCP vs PSM - Bandwidth

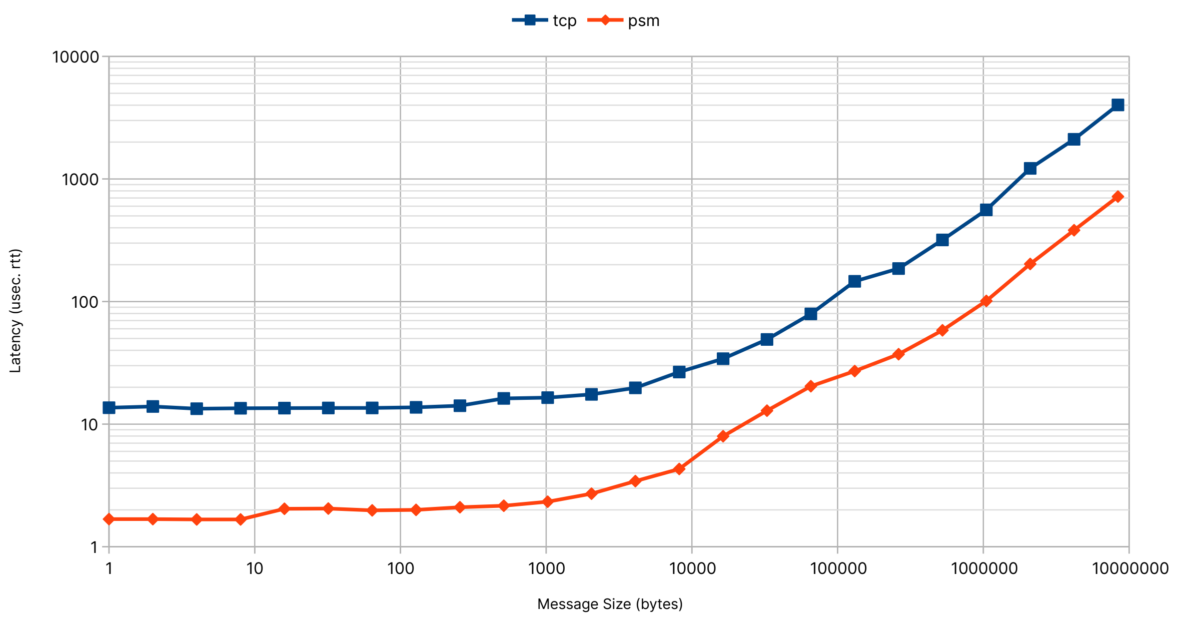

Figure 2: TCP vs PSM - Latency

In addition, the proportion of time spent in kernel space is significantly lower than that of TCP. A ratio of 0.5, for example, indicates 50% of CPU time is spent not in user code, but in the kernel - in this case, the only significant task the kernel performs for the benchmark is data transfer. This leaves the CPU available for more data processing throughput.

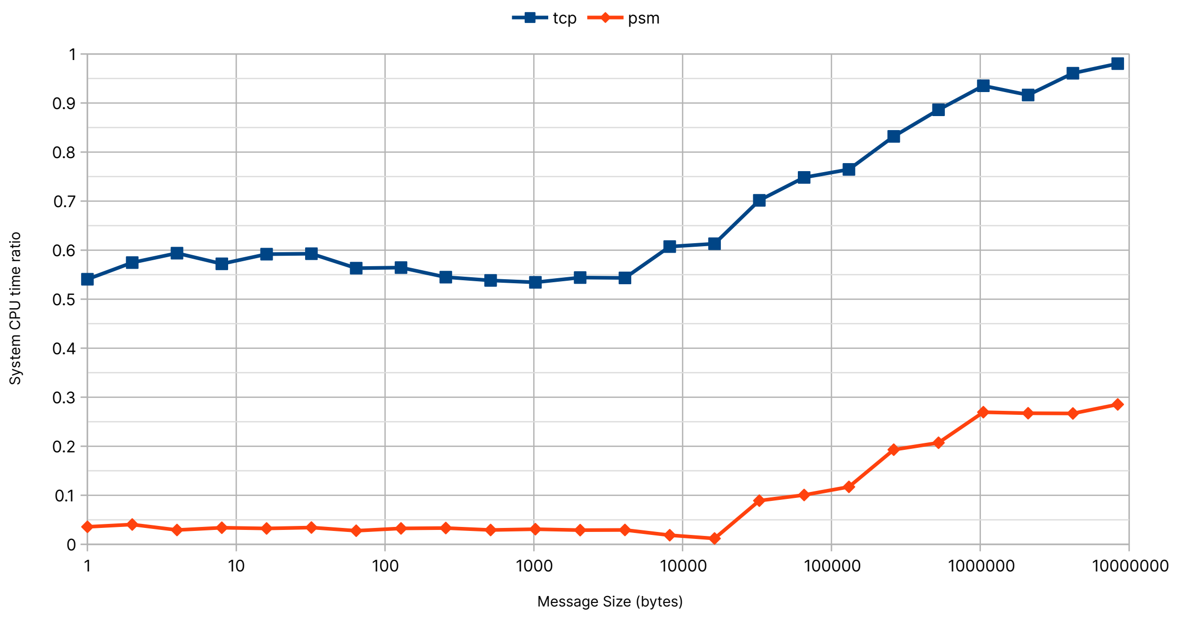

Figure 3: TCP vs PSM - Kernel Time

3 Programmable & Smart Switches

Connecting thousands of nodes directly to each others' network ports, while the highest bandwidth option, becomes impractical after a handful of compute nodes. Thus the primary purpose of network switches is moving packets between different devices on the network quickly and in a scalable fashion. The exact interconnect type, bandwidth, and speed varies from cluster to cluster.

Network switches are designed to switch traffic as fast as possible between ports. Most modern switches, especially those designed for the datacenter, feature custom Application-Specific Integrated Circuit (ASIC) designs for simultaneous, non-blocking, bidirectional bandwidth between all ports [NVIf]. Adding processing features on the switch allows for the switch to perform advanced data manipulation operations with full view of all connected ports, and potentially at up to the combined bandwidth of all ports.

There are two main ways this is realised in practice. Fixed-function acceleration, such as Mellanox Scalable Hierarchical Aggregation and Reduction Protocol (SHARP), or a programmable Field-Programmable Gate Array (FPGA) connected directly to the high-speed switch fabric. SHARP was specifically designed to solve the needs of HPC customers, while FPGA-based solutions are designed to be adaptable towards a wide variety of other high-performance networking tasks beyond HPC, such as line rate deep packet inspection. The latter will not be covered in this report.

3.1 SHARP

SHARP is a technical specification designed to allow for basic in-network computing functionality. It was first introduced in the Mellanox SwitchIB-2 series of InfiniBand network switches, and allows for certain operations to be performed on-switch rather than by compute nodes [Gra+16].

In a traditional data-center network without SHARP support, collective operations are highly bandwidth- and compute-intensive [Rab]. Performing operations such as finding a global sum require all participating nodes to send their data to a single root node for final computation. This leads to a potential bandwidth bottleneck as data converges towards a single network link. In addition to bandwidth issues, the nodes' processors are also (at least partially) occupied performing the necessary computation, slowing down other computations and thus limiting the amount of possible communication and computation overlap. Techniques such as recursive doubling, or selecting a node in a switch group to perform intermediate computations depending on the network topology could alleviate this problem to some extent by reducing the consumption of slow inter-switch bandwidth and increasing the amount of parallel I/O occurring in the MPI job [Mat+06; Ara+15; PY07]. However, the traffic pattern inevitably remains as many-to-one, nodes also receive disproportionate amounts of traffic, and processors are still occupied performing the actual reduction tasks.

SHARP's claimed benefit is to address these bottlenecks by offloading these collective computation and synchronization operations onto switches. On Mellanox switches, this is achieved by adding an ALU into the switch. The capabilities are as follows [Gra+16]:

- 32-bit and 64-bit integer and floating point support
- Sum, Min, Max, MinLoc, MaxLoc, Bitwise OR/AND/XOR

Two types of nodes are defined: Aggregation Node (AN) and End Node (EN). The AN becomes the target of aggregation operations, and can be either a switch or end-node itself (i.e., a selected compute node within a switch group). ENs transmit the data to be aggregated to ANs, which perform any necessary operations before moving to the root node. The nodes are organized into a tree (exactly how is not defined), which determines the direction of data flow [Gra+16].

In the following example, a global sum is to be calculated. The thickness of the link indicates the relative amount of data traffic consumed over the link. **N** indicates a regular compute node, while **R** indicates the root node (i.e., MPI Rank 0).

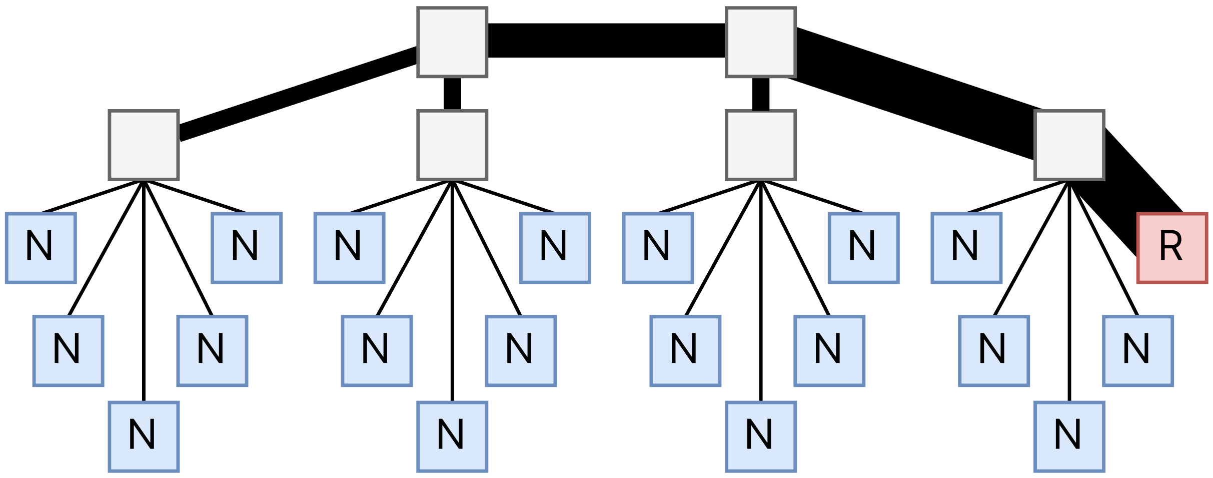

Figure 4: Global Sum, No SHARP

Depending on the MPI implementation used and the network configuration, some nodes are used as intermediate compute nodes in order to reduce overall bottlenecks [NVIa]. **I** indicates an intermediate node, **black** lines indicate traffic flow from compute to intermediate nodes, and **red** lines indicate traffic flow towards root nodes.

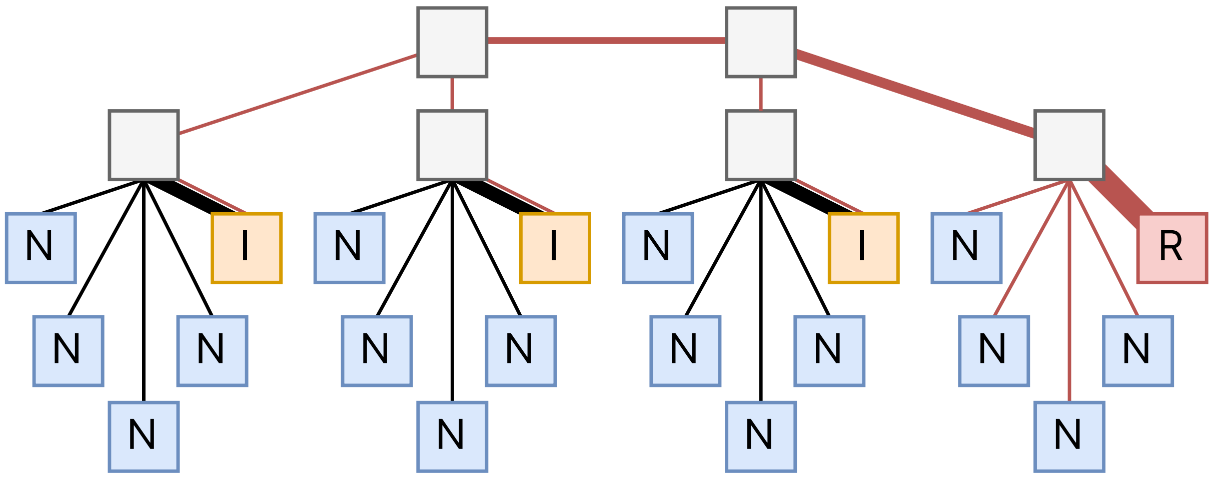

Figure 5: Global Sum, No SHARP, Optimized

In a SHARP network, switches that support the protocol take the place of intermediate and root nodes for the purposes of collective communication. The following network topology is used where certain switches **S** do not support SHARP offload, while switches labeled **S+** do.

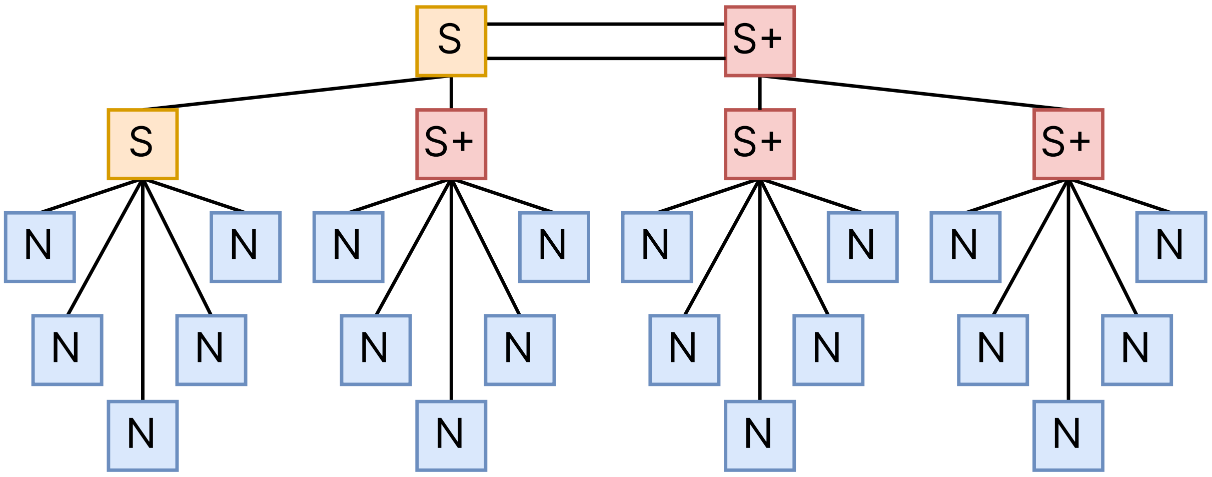

Figure 6: Network with partial SHARP switch support

Combining the previously discussed techniques, both traditional intermediate nodes and SHARP nodes can be used where possible. Aggregation steps in the global sum reduces the total inter-switch bandwidth to a single unit of data when SHARP is supported along the entire data path. Additionally, no node is required to consume CPU resources to perform summing operations, unless SHARP is unsupported.

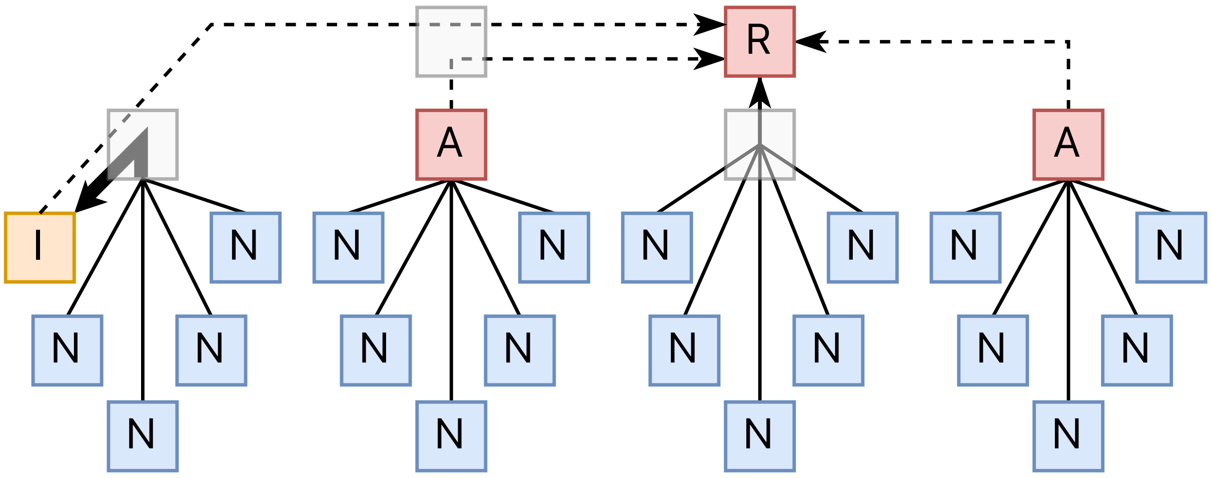

Figure 7: Global Sum, SHARP

SHARP has been shown to perform especially well in large-scale clusters, where the amount of traffic flow for collective operations can be especially high. According to results published by the MVAPICH2 developers, the latency of `MPI_Barrier`, a synchronization primitive, can be up to 9 times lower than in conventional networks (tested on a 7,861 node cluster) [Sub21].

4 DPUs

The Data Processing Unit (DPU) is currently the most programmable form of network adapter. Existing in some form as early as 2006 with the Cavium Octeon series of NICs [Cav06], the fundamental building blocks of a DPU are a base NIC combined with a custom, typically ARM-based System-on-Chip (SoC) placed in the path of data traffic [NVId]. Modern DPUs such as the BlueField NIC, run near-standard ARM Linux distributions [NVIb] which are familiar to system administrators and programmers, especially those in the HPC space where Linux makes up 100% of the TOP500 list [TOP]. In effect, a modern DPU is a mostly independent, fully-featured computer in a NIC form factor, connected to the same high-speed, low-latency network as the host machine.

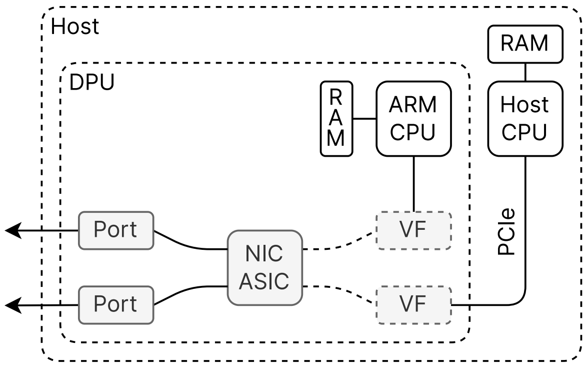

Figure 8: DPU Block Diagram (BlueField-2)

In practice according to NVIDIA, both in HPC and data centers more broadly, DPUs are used in order to offload ancillary tasks off of the main system processors (i.e., CPUs/GPUs). As the name suggests, this usually involves the processing of data on-device in ways which might have previously been performed by the main processors. Theoretically, the offloading of these tasks to the DPU makes more CPU resources available for performing other tasks [NVIi].

4.1 Vendor-Supplied Offloads

Vendors usually supply premade offloads for their DPU products for common use cases, as well as SDKs to simplify programming for them. NVIDIA, for instance, supplies an Open Virtual Switch (OvS) [NVIh] and NVMe over Fabrics (NVMeoF) [NVIe] offload.

4.1.1 OvS Offload

OvS is, essentially, a virtual network switch. It is typically used for firewalling, rate limiting, and switching of traffic between virtual machines and/or containers, making it easier and more secure to share a single compute node's network resources among multiple users [OVS]. It has been shown that moving these tasks onto the DPU reduces the overhead of performing these operations and thus leaves more CPU time available for more application density or lower power consumption. According to benchmarks performed by Red Hat on a Kubernetes cluster, the BlueField-2 DPU reduced CPU utilization by 70% at 25Gbps of traffic flow (line-rate). In absolute terms, this resulted in approximately 3 cores worth of computing power being made available on the host [GKF]. Up to a 29% energy savings can be achieved at full server load, according to NVIDIA [NVIc], with 18 cores worth of computing power being saved on the host moving 49Gbps of traffic. However, the exact hardware details were not stated.

4.1.2 NVMeoF Offload

As data sets grow larger, the demand placed on storage systems and protocols has significantly increased. In contrast to its predecessors Advanced Host Controller Interface (AHCI), Small Computer Systems Interface (SCSI), and Integrated Drive Electronics (IDE) which were designed in the era of magnetic hard drive storage, Non-Volatile Memory Express (NVMe) is a modern storage protocol designed specifically for flash storage [NVMa]. This results in comparatively low latency and high performance relative to its predecessors. In order to extend these benefits to network storage, NVMeoF encapsulates NVMe commands over the network, rather than only the local machine [NVMb]. So important was this use case for datacenters and HPC, that the NVMeoF protocol was integrated into the core NVMe 2.0 specification in 2021 [NVMa].

NVMeoF has been implemented over both traditional TCP and RDMA-enabled transports. When entirely offloaded to the DPU, performance remains high while consuming little to no system resources. Two benchmarks were evaluated, both using SPDK, a library designed for high-performance storage applications: one from Intel, and the other from NVIDIA using the BlueField-2 DPU. Performance was evaluated in these benchmarks by reading 4KB of random data and measuring the total number of Input/Output Operations per Second (IOPS).

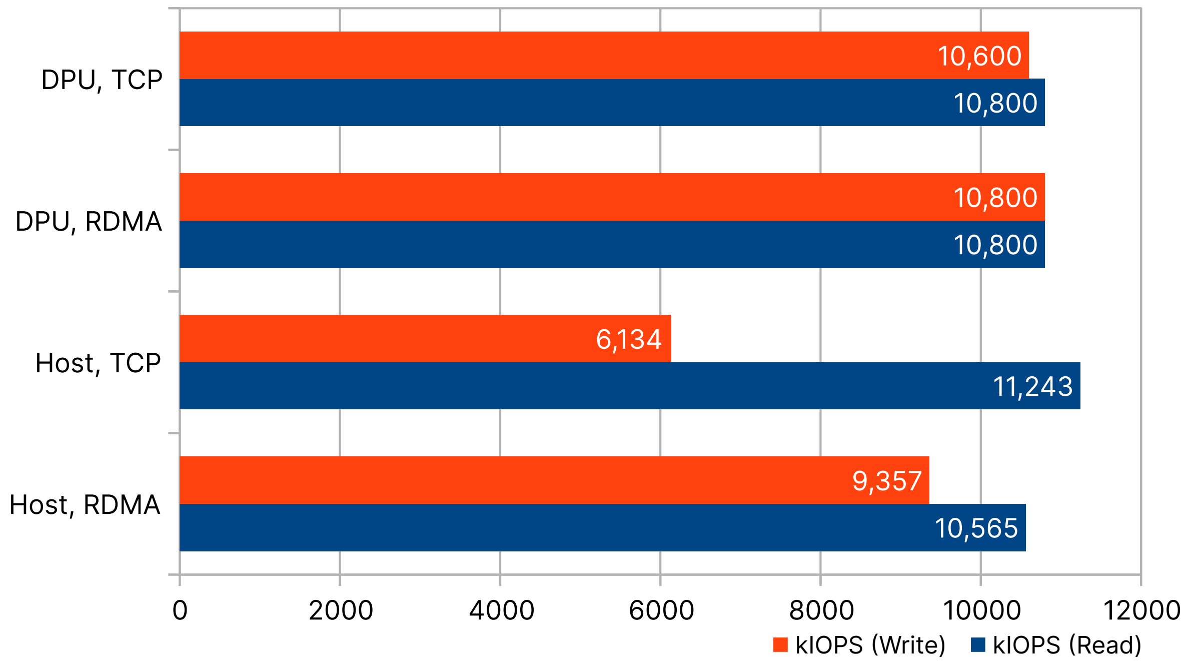

Figure 9: DPU vs Host I/O Performance

It should be noted that, while host-based TCP slightly outperforms the DPUs in one specific test, it does so by consuming all the CPU resources on the system (48 cores). All benchmarks are close to the theoretical maximum for the networks used in the test (400 Gbit/s = 12.5M 4KB IOPS, less with overhead), however, the DPU solution consumes no host system resources in doing so.

4.2 Use in Data Analytics

Machine Learning (ML) workloads on HPC can be roughly broken down into a few basic steps: data must be loaded and pre-processed, the model trained on the pre-processed slices of data, and results validated after a training epoch is complete [Jai+22]. This is repeated until a certain threshold of performance is reached. For cutting-edge models, these steps can take years worth of computing hours [Bro+20], and as such techniques to reduce overall training time are of great interest to those working with such models.

The use of DPUs to offload these tasks is a relatively new area of research. However, initial results from DPU-accelerated deep learning have shown up to a 17.5% improvement in training speed. Three offloading designs were proposed and evaluated in a paper by Arpan Jain et al. [Jai+22].

4.2.1 Data Preprocessing

By the paper's definition, data preprocessing offload involves the use of the DPU in the loading of data from storage (usually a centralized storage server on the network) and performing user-defined transformations to data (such as normalization) before they are fed into the model. The data are split into sub-batches for each incoming batch of data by the DPU, allowing one sub-batch to be processed by the host while the DPU queues up data for the next batch asynchronously. Theoretically, the increased parallelism allows for reduced time waiting on new data to arrive. The following diagram shows the parallelism achieved by this offload: while buffer A is being loaded, buffer B is being processed by the host, and vice versa.

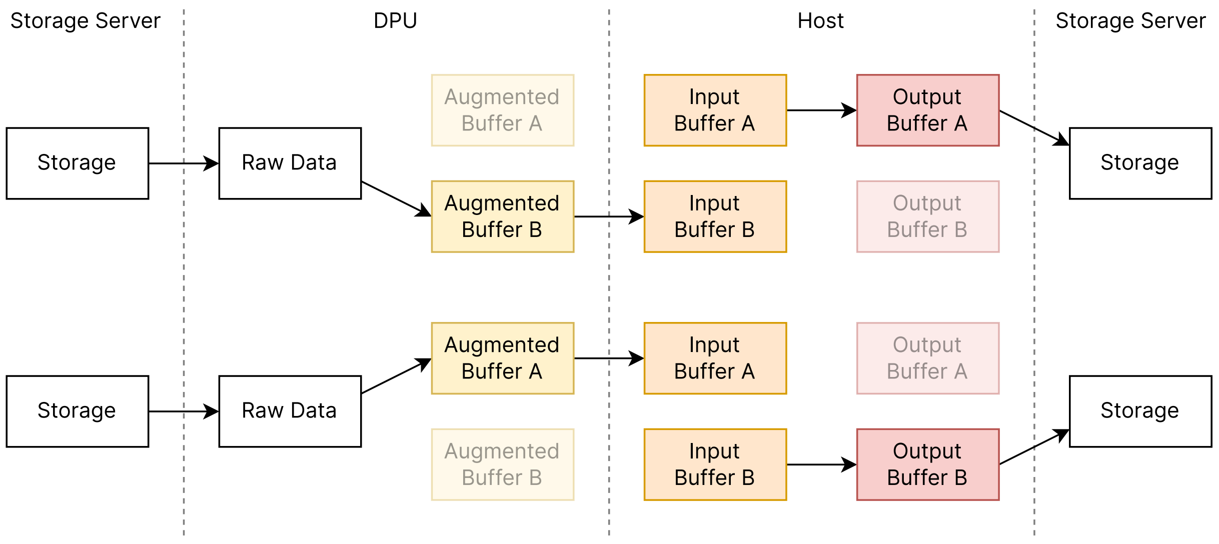

Figure 10: Data Preprocessing Offload

4.2.2 Model Validation

Alternatively, the model can be validated asynchronously by transferring model parameters to the DPU after a completed training step, which allows training to continue immediately after without requiring the use of host resources for validation. The host, however, performs the aforementioned fetching of data from the server.

4.2.3 Combined Offload

The combined (or hybrid) approach described combines both data preprocessing and validation offloads simultaneously.

4.2.4 Results

According to the researchers' data, preprocessing offload performs well with smaller models (in this case: ResNet-20). With larger models, the hybrid approach performs best. The below graph from the research paper shows the strongest scaling results of the presented benchmarks.

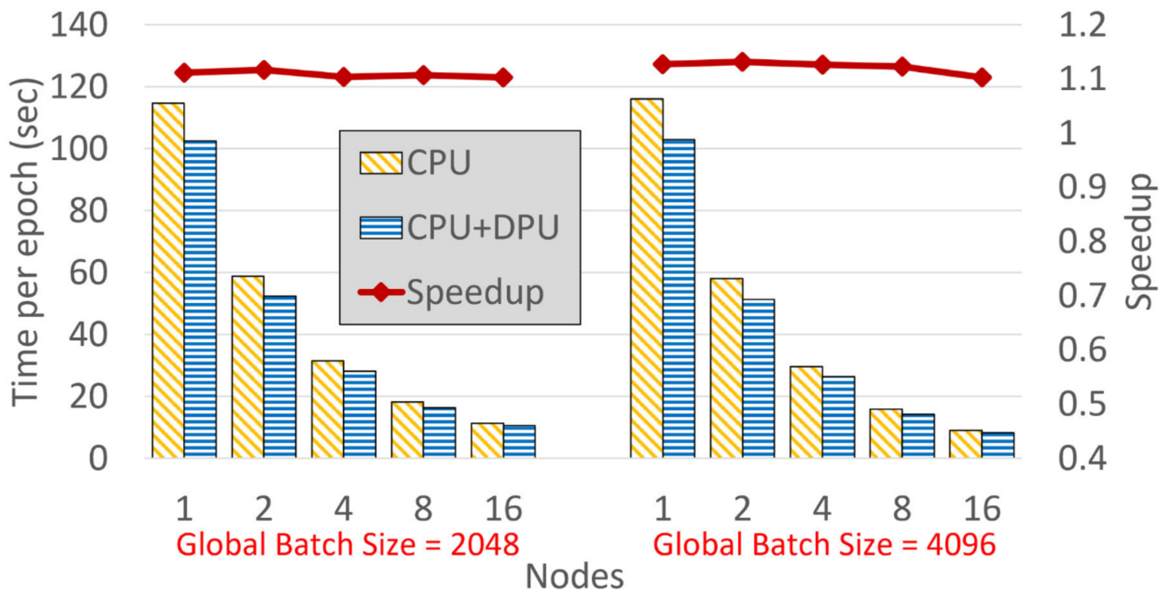

Figure 11: DPU Acceleration Performance (ResNet-20, Preprocessing Offload)

Part of the benefit of data preprocessing offload demonstrated was the masking of slower performance of networked filesystems. As the DPU is able to perform data loading in parallel with host computation, this effectively provides more available time for the system to load in new data before the next batch is processed. A speedup of up to 17% was demonstrated when data were loaded from the slower (`/scratch`) filesystem rather than host memory (`/tmp`).

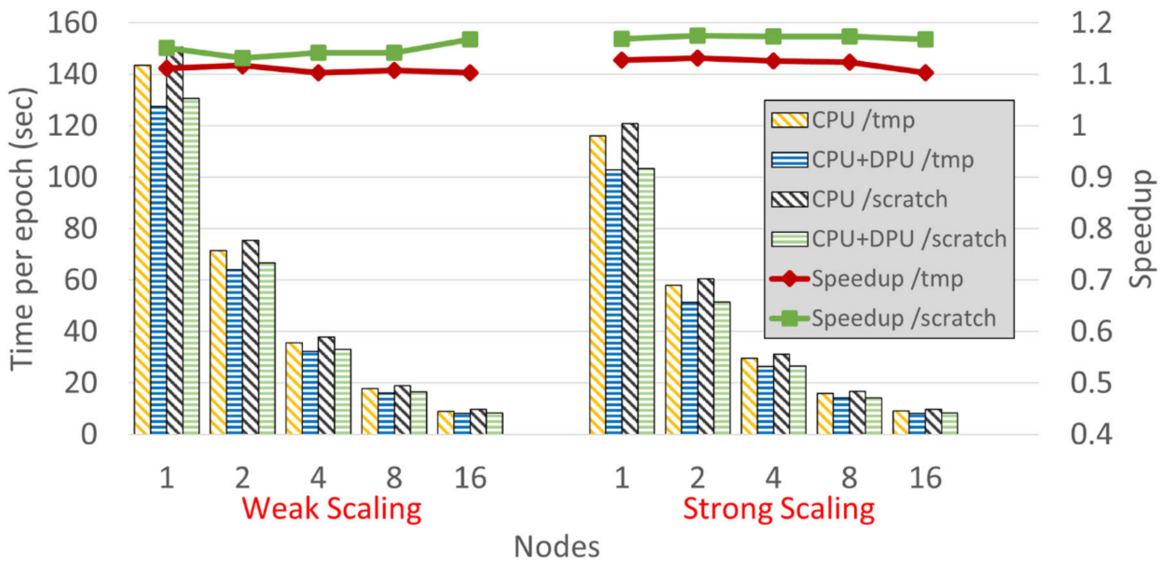

Figure 12: Preprocessing Offload on Various Filesystems

4.3 MPI offload

MPI provides a collection of high-level functions for data manipulation and transfer between two nodes (point-to-point) or involving all nodes (collective). For instance, a 100-length integer array may be sliced and distributed evenly across all nodes for processing with an `MPI_Scatter` operation. The MPI implementation then performs the lower-level communication and shuffling internally, greatly simplifying parallel programming for the user.

As an alternative or supplement to SHARP support, DPUs can be programmed to support the acceleration of MPI collective functions as well. However, existing production implementations are neither publicly available or open-source, as opposed to the many competing MPI implementations, such as OpenMPI, MPICH, Intel MPI, and more. One implementation, MVAPICH2-DPU, is based on the MVAPICH library with added DPU-enabled collective communication offload [X-S22]. According to the developers, up to a 22% reduction in total compute time for collective operations was achieved on a 32-node cluster, relative to the non-DPU-accelerated version of MVAPICH2. Additionally, full simultaneous communication and computation was made possible through offloading the communication routines fully to the DPU.

5 Conclusion

In-network computing has been demonstrated to bring significant potential advantages to the HPC space. In particular, the offloading capabilities of DPUs and network switches have been particularly useful in bandwidth-heavy or latency-sensitive workloads, where they have reduced compute time and total system power consumption at scale in a cost-effective manner. RDMA has been proven in HPC for years, with fast HPC networks reliant on its often order-of-magnitude performance improvements over traditional network transports to achieve their large scales. However, much of the existing literature on newer technologies, especially with regard to DPUs are relatively recent. Thus, the adoption of these technologies in HPC are still in their early stages. Given the potential demonstrated by existing research, it is likely that future advances in DPUs and other types of speciality network adapters will provide researchers with new possibilities in research, as well as system administrators with increased cluster utilization, space efficiency, and reduced power consumption. In addition, there is currently limited native integration for DPU offload into incumbent libraries. What makes RDMA and SHARP successful, as discussed, was the integration into commonly used libraries making taking advantage of these offloads often as simple as enabling a toggle on supported hardware. Since many DPU offloads currently only exist in research or in proprietary applications, applying them to existing programs is therefore a much more involved and manual process- one which researchers that are not specialized in HPC programming are unlikely to take.

References

[Amd67] Gene M. Amdahl. “Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities”. In: *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference.* AFIPS ’67 (Spring). Atlantic City, New Jersey: Association for Computing Machinery, 1967, pp. 483–485. ISBN: 9781450378956. DOI: 10.1145/1465482.1465560. URL: https://doi.org/10.1145/1465482.1465560.

[Ara+15] Omer Arap et al. “Adaptive Recursive Doubling Algorithm for Collective Communication”. In: *2015 IEEE International Parallel and Distributed Processing Symposium Workshop.* 2015, pp. 121–128. DOI: 10.1109/IPDPSW.2015.82.

[Bro+20] Tom B. Brown et al. *Language Models are Few-Shot Learners.* 2020. arXiv: 2005.14165 [cs.CL].

[Cav06] Cavium Inc. *OCTEON™XL NIC Accelerator Board Family - Product Brief.* https://web.archive.org/web/20061021072446/http://www.cavium.com/pdfFiles/OCTEON_NIC_productBrief.pdf. Internet Archive. 2006.

[Cla+89] D.D. Clark et al. “An analysis of TCP processing overhead”. In: *IEEE Communications Magazine* 27.6 (1989), pp. 23–29. DOI: 10.1109/35.29545.

[GKF] Eric Garver, Rashid Khan, and Yan Fisher. *Optimizing server utilization in datacenters by offloading network functions to NVIDIA BlueField-2 DPUs.* https://www.redhat.com/en/blog/optimizing-server-utilization-datacenters-offloading-network-functions-nvidia-bluefield-2-dpus.

[Gra+16] Richard L. Graham et al. “Scalable Hierarchical Aggregation Protocol (SHArP): A Hardware Architecture for Efficient Data Reduction”. In: *2016 First International Workshop on Communication Optimizations in HPC (COMHPC).* 2016, pp. 1–10. DOI: 10.1109/COMHPC.2016.006.

[Jai+22] Arpan Jain et al. “Optimizing Distributed DNN Training Using CPUs and BlueField-2 DPUs”. In: *IEEE Micro* 42.2 (2022), pp. 53–60. DOI: 10.1109/MM.2021.3139027.

[KR22] James F. Kurose and Keith W. Ross. *Computer Networking: A Top-Down Approach. Eighth Edition.* Pearson, 2022.

[Mat+06] Motohiko Matsuda et al. “Efficient MPI Collective Operations for Clusters in Long-and-Fast Networks”. In: *2006 IEEE International Conference on Cluster Computing.* 2006, pp. 1–9. DOI: 10.1109/CLUSTR.2006.311848.

[MB91] Jeffrey C. Mogul and Anita Borg. “The Effect of Context Switches on Cache Performance”. In: *SIGOPS Oper. Syst. Rev.* 25.Special Issue (Apr. 1991), pp. 75–84. ISSN: 0163-5980. DOI: 10.1145/106974.106982. URL: https://doi.org/10.1145/106974.106982.

[McC] John D. McCalpin. *Memory Bandwidth and System Balance in HPC Systems.* https://repositories.lib.utexas.edu/bitstream/handle/2152/86843/MemoryBW_SystemBalance_2016-11-16.pdf.

[Mog03] Jeffrey C. Mogul. “TCP Offload is a Dumb Idea Whose Time Has Come”. In: *Proceedings of the 9th Conference on Hot Topics in Operating Systems - Volume 9*. HOTOS’03. Lihue, Hawaii: USENIX Association, 2003, p. 5.

[NFB96] Henrik Nielsen, Roy T. Fielding, and Tim Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.0*. RFC 1945. May 1996. DOI: `10.17487/RFC1945`. URL: `https://www.rfc-editor.org/info/rfc1945`.

[NVIa] NVIDIA Corporation. `https://docs.nvidia.com/networking/display/HPCXv27/HCOLL`.

[NVIb] NVIDIA Corporation. *BlueField Software Overview*. `https://docs.nvidia.com/networking/display/BlueFieldDPUOSLatest/BlueField+Software+Overview`.

[NVIc] NVIDIA Corporation. *DPU Power Efficiency White Paper*. `https://resources.nvidia.com/en-us-accelerated-networking-resource-library/nvidia-dpu-power-eff`.

[NVId] NVIDIA Corporation. *Functional Diagram*. `https://docs.nvidia.com/networking/display/BlueFieldDPUOSLatest/Functional+Diagram`.

[NVIe] NVIDIA Corporation. *NVIDIA BlueField-2 DPU*. `https://resources.nvidia.com/en-us-accelerated-networking-resource-library/bluefield-2-dpu-datasheet?lx=LbHvpR&topic=networking-cloud`.

[NVIf] NVIDIA Corporation. *NVIDIA Spectrum Ethernet Switch Family*. `https://nvdam.widen.net/s/zcs9mvtnmn/ethernet-switches-switch-datasheet-spectrum-2347441`.

[NVIg] NVIDIA Corporation. *NVIDIA Spectrum-4*. `https://nvdam.widen.net/s/v6skdhzdrh/ethernet-switches-product-brief-gtc22-spring-spectrum-4-2169045-r8`.

[NVIh] NVIDIA Corporation. *Virtual Switch on DPU*. `https://docs.nvidia.com/networking/display/BlueFieldDPUOSLatest/Virtual+Switch+on+DPU`.

[NVIi] NVIDIA Corporation. *What is a DPU?* `https://blogs.nvidia.com/blog/2020/05/20/whats-a-dpu-data-processing-unit/`.

[NVMa] NVM Express. *NVMe Base Specification*. `https://nvmexpress.org/developers/nvme-specification/`.

[NVMb] NVM Express. *NVMe-oF Specification*. `https://nvmexpress.org/developers/nvme-of-specification/`.

[Ope] OpenMPI Developers. *Tuning the run-time characteristics of MPI InfiniBand, RoCE, and iWARP communications*. `https://www.open-mpi.org/faq/?category=openfabrics`.

[OVS] OVS Developers. *Open vSwitch*. `https://www.openvswitch.org/`.

[PY07] Pitch Patarasuk and Xin Yuan. “Bandwidth Efficient All-reduce Operation on Tree Topologies”. In: *2007 IEEE International Parallel and Distributed Processing Symposium*. 2007, pp. 1–8. DOI: `10.1109/IPDPS.2007.370405`.

[Rab] Rolf Rabenseifner. *Automatic MPI Counter Profiling*. `https://cug.org/5-publications/proceedings_attendee_lists/2000CD/S00_Proceedings/pages/Authors/Rabenseifner/RABENSEI.PDF`.

[Ros13] Rami Rosen. *Linux Kernel Networking: Implementation and Theory*. Apress, 2013.

[Sub21] Hari Subramoni. *The MVAPICH2 Project Latest Status and Future Plans*. `https://www.mpich.org/static/docs/slides/2021-sc-bof/MVAPICH2.pdf`. 2021.

[TOP] TOP500.org. *List Statistics*. `https://www.top500.org/statistics/list/`.

[TW17] Thomas N. Theis and H.-S. Philip Wong. "The End of Moore's Law: A New Beginning for Information Technology". In: *Computing in Science & Engineering* 19.2 (2017), pp. 41–50. DOI: `10.1109/MCSE.2017.29`.

[Wil18] Brandon Wilson. *Why Are We Deprecating Network Performance Features (KB4014193)?* `https://techcommunity.microsoft.com/t5/core-infrastructure-and-security/why-are-we-deprecating-network-performance-features-kb4014193/ba-p/259053`. 2018.

[X-S22] X-Scale Solutions. *MVAPICH2-DPU*. `https://x-scalesolutions.com/mvapich2-dpu/`. 2022.