# ACiS: Complex Processing in the Switch Fabric

Pouya Haghi* Anqi Guo+ Tong Geng* Anthony Skjellum# Martin Herbordt+
*University of Rochester +Boston University #Tennessee Tech

***Abstract*—For the last three decades a core use of FPGAs has been for processing communication: FPGA-based SmartNICs are in widespread use from the datacenter to IoT. Augmenting *switches* with FPGAs, however, has been less studied, but has numerous advantages built around the processing being moved from the edge of the network to the center. Communication switches have previously been augmented to process collectives, e.g., IBM BlueGene and Mellanox SHArP, but the support has been limited to a small set of predefined scalar operations and datatypes.**

**In this extended abstract we present ACiS, a framework and taxonomy for *Advanced Computing in the Switch* that unifies and expands our previous work in this area. In addition to fixed scalar collectives (Type 1), we propose three more types of in-switch application processing: (Type 2) *User-defined* operations and types, including data structures; (Type 3) *Look-aside* operations that have state within the operation and can have loops; and (Type 4) *Fused* collectives built by fusing multiple existing collectives or collectives with map computations. ACiS is supported in hardware with modular switch extensions including a CGRA architecture. Software support for ACiS includes evaluation and translation of relevant parts of user programs, compilation of user specifications into control flow graphs, and mapping the graphs into switch hardware. The overall goal is the transparent acceleration of HPC applications encapsulated within an MPI implementation.**

## I. Introduction

High performance computing (HPC) systems are facing a crisis in performance-portability and scalability. Problems include increasing communication latency, networks that are both overloaded and underutilized (depending on the application mix), load balancing, and process skew. In this work we propose ACiS, a framework and taxonomy for *Advanced Computing in the Switch*. ACiS adds application-level computation to communication switches; we find that extremely high-value computing can be enabled with minimal redesign of either network, NIC, or processing node, and that ACiS hardware can be added to the switch without changing standard dataplane architecture or loss of non-ACiS performance.

Thirty years of codesign have made FPGAs ideal communication processors. There are at least three aspects to this: (i) compute capability that is high performance and per-application configurable; (ii) communication capability that is vast and flexible through scores of multigigabit transceivers (MGTs); and (iii) tight coupling of computation with communication that enables application-level transfers in just a few cycles. Use of FPGAs in ACiS is therefore a plausible approach for addressing HPC scalability problems.

Offload of collective processing into SmartNICs [9], [61] is well established and has a number of benefits: first, it enables the bypassing of layers in the communication software stack; second, the hardware implementations are substantially faster than the software; third, it frees up the host processor for other tasks and, potentially, enables better communication-computation overlap; and fourth, some network-host communication is removed as the NIC handles additional send/receive operations. While SmartNICs are valuable, this scheme still forces processing into the endpoints.

Another approach is to offload collective processing into the switches [16], [19]. This has two additional benefits: first, latency is improved as computation is distributed rather than performed in a single source (broadcast) or endpoint (reduction); and, second, communication volume may be drastically reduced as messages are quickly merged (reduction) or slowly replicated (broadcast). Communication switches have previously been augmented to process collectives, e.g., the IBM BlueGene project and the Mellanox SHArP switch, but the support has been limited to a small set of predefined scalar operations and datatypes (e.g., [17], [19]). Moreover, beyond collectives there are additional acceleration opportunities.

With ACiS we unify and expand our previous work [24], [28], [29], [31] to provide a framework that further augments switches to accelerate additional and more complex functions that integrate communication with computation. In addition to simple collectives (Type 1), we propose three more types of in-switch application processing: Type 2 – *User-defined* operations and types, including simple data structures; Type 3 – *Look-aside* operations that have state within the operation and can have loops; and Type 4 – *Fused* collectives built by fusing multiple existing collectives or collectives with computations.

Implementations with reconfigurable logic have several inherent advantages over fixed logic implementations.[1] First, they are not limited to a small, fixed set of operations; second, hardware resources can be configured to match application requirements; third, support can be extended beyond simple datatypes to higher order structures such as matrices, tensors, and user defined datatypes; fourth, since reconfiguration time is similar to program load time, only resources that will actually be used need to be configured.

Our hardware approach begins with a seamless integration into a general router, added hardware support for aggregating (reordering) packets for reduce-type (gather-type) operations, and the flexibility to add more computational resources as the switch bandwidth increases. Complexity is supported with a CGRA-like architecture. Software support for ACiS includes

[1] We note that software implementations are out of the question since they could never keep up with current line rates.

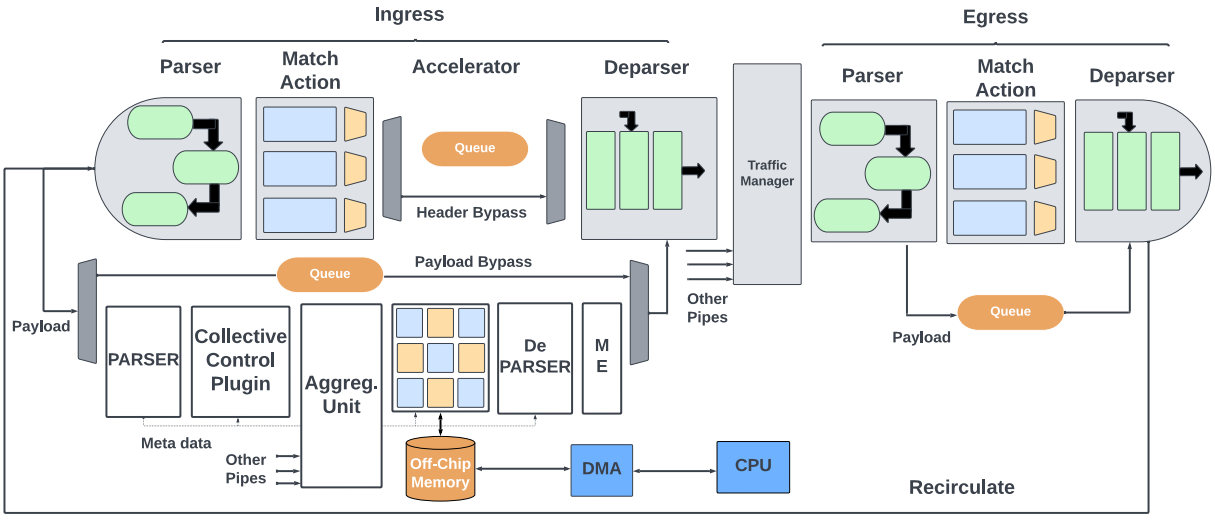


Fig. 1. The well-known Protocol Independent Switch Architecture (PISA) enhanced with ACiS accelerator, a composable plugin to a packet processing pipeline (one pipe shown). White blocks are proposed and grey blocks are in existing switches.

TABLE I
ACIS TYPES REFERENCE

| | |
|---|---|
| Type 0 | Processing single data streams |
| Type 1 | Collectives on primitive types |
| Type 2 | User defined operations and types |
| Type 3 | Look-aside processing - loops and memory |
| Type 4 | Fused collectives and map functions |

evaluation and translation of relevant parts of user programs, compilation of user specifications into control flow graphs, and mapping the graph into switch hardware. Another core part of ACis is encapsulation within an MPI implementation for transparent acceleration of HPC applications. Efficacy of ACiS has been demonstrated [24], [28], [29], [31] on a number of HPC applications including the NAS parallel benchmarks, two MiniApps from Mantevo (miniFE and HPCCG), graph convolutional networks (GCNs), PGEMM, and AMG.

For the rest of this report we first expand on the ACiS Types, then give an overview of the hardware aspects of ACiS, followed by some experimental results, an overview of software support, and related work.

## II. ACIS TYPES

We classify ACiS capabilities into progressively complex types of in-switch computing (Table I):

**Type 0:** Consists of transformations on **streams** such as changes in data types or appending a CRC code. This is a well-understood subset of work proposed here and only mentioned for completeness.

**Type 1:** Adds to Type 0 by supporting **collectives**, but on a limited number of primitive data types (e.g., int) and operations (e.g., add, max). Datatypes, operations, and communication contexts (MPI communicator) are fixed.

**Type 2 [28]:** Adds to Type 1 by supporting collectives of **user-defined** operations (e.g., dot product), datatypes (e.g., matrices, sparse datatypes), and communication context with MPI communicators scalable to a large number of nodes.

**Type 3 [29]:** Adds to Type 2 by supporting **look-aside** capability: functions requiring loops and off-chip memory. The data plane has direct access to off-chip memory for storing/retrieving packet data. Examples include accelerating compression algorithms and communication-intensive parts of machine learning inference for large-scale datasets.

**Type 4: [31]** Adds to Type 3 by supporting **fused collectives** and the combining of collectives with map operations. An example is fusing AllReduce with AlltoAll found in the NAS sort benchmark. Another example is accelerating MapReduce type of operations (used in DNN inference).

## III. RELATED WORK, ACIS DIFFERENTIATION

Although *the network is the computer* is an old trope [51], in practice it has most often meant use of the network as a passive, transparent conduit. Recent years have seen the flipping of the data center model from compute-centric to data-centric [48], [65], in part, through the emergence of SmartNICs [5]–[8], [10], [37], [46], [49], [52], [56] and similar network-facing devices [12], and their use in offloaded application [13], [21]–[23], [64] and system [18], [41] processing. Simple application-stream processing for collectives has also been implemented in switches by IBM and Mellanox [4], [20], [52], but these operations have had limited scope, e.g., reductions only, and on a small set of operations and simple data types (part of ACiS Type 1). Also, in general, they have offered only modest benefit in typical environments [27], [35]. Our recent work has demonstrated the viability and benefit of extending in-switch processing in two ways: expanding in-switch processing support along several dimensions (Types 2-4 [28], [29], [31]) and network-centric acceleration of specific applications [25], [26]. The work described here proposes extensions and generalizations to these approaches, namely,

to advance intelligent communication by augmenting switches to enable complex processing augmentations (to Types 2-4) as described in Section II.

Of special mention is P4, the "domain-specific programming language for network devices, specifying how data plane devices (e.g, switches, routers, NICs, filters, etc.) process packets" [50]. P4 has aided in increasing switch flexibility [11] including accelerating applications that are beyond the basic switching function. Examples are consensus algorithms [15], database transaction processing [38], caching [44], and key-value store [39]. While there has also been some exploration of application-level processing [53] in P4 switches, including support of collectives in ML training [54], these capabilities are currently limited [60] by the supported set of operations (e.g., no multiply), data types (e.g., no sparse data types), and memory footprint. Perhaps most significantly, packets can only access each memory location once within a traversal [16]; while it is possible to recirculate packets, this reduces throughput. To address these limitations, ACiS is designed to handle advanced calculations, such as fused multiply-accumulate (MAC) and sparse accumulation, off-chip memory access with cacheable buffers, data reuse, and control.

In-switch computing is an active area of research with Types 1-3 (defined in Section II) being partially addressed. The work in [42] provides an in-switch computing paradigm, implemented on NetFPGA, to accelerate aggregation of gradients used in the training phase of reinforcement learning (Type 1). The work in [16] designs a flexible programmable switch on top of PsPIN building blocks to accelerate Allreduce with custom operators and data types; that is, sparse data (Type 2 on a single collective type). The work [45] presents an RDMA-compatible in-network reduction architecture to accelerate distributed DNN training in which the FPGAs are connected to the switch and the switch is configured to route the packets that need to be aggregated to the FPGA (Type 2). The work in [60] adds custom hardware based on a MapReduce pattern/abstraction (built upon a CGRA) to P4 devices to enable per-packet inference of machine learning (Type 3 - but with no fusion of collectives). To summarize, we are not aware of previous work that fully supports user-defined or complex collectives (Types 2 and 3) or in any way addresses look-aside capability (Type 4).

## IV. Hardware Design

An important switch design consideration is modularity [47], which ACiS follows by introducing *composable plugins* to successively add capabilities. These plugins can be (nearly) seamlessly added to existing switches and are network friendly. Capabilities for Types 2-4 include flexible aggregation and communication management, schedulability, and stateful processing.

Given that ACiS types are built in successive layers, the architecture is composed of successive plugins. Figure 1 shows PISA, a state-of-the-art protocol-independent switch model, enhanced with an ACiS accelerator for Types 1-4. Only one pipe is shown in this figure; a switch is comprised of replicated pipes and each pipe can include a number of physical ports [62]. We note that packet processing in this model is done at the header level. However, MPI-specific fields (e.g., source/destination ranks, tag) are embedded in the payload. Thus, we modify the architecture by introducing a separate pipeline for payload-level packet processing (bottom part of Figure 1). We summarize the key ideas for each plugin.

**Type 2** supports a collective processing unit including flexible management of MPI communicator context and programmable aggregation units with the support for different operations and data types. Two plugins are (1) a collective control unit, and (2) a programmable aggregation unit. We implement the collective control plugin with lookup tables. Current protocol-independent switches, however, do not offer flexible wide-access to these lookup tables; that restriction is addressed with this plugin. To route the output of the aggregation unit to any pipe (1-to-N converter) there is (3) a multicast engine at the end of the pipeline (inspired from the packet replication engine in P4 switches [55]). Finally, in order to be compliant with any MPI implementation, we also need two other plugins: (4) a parser and (5) a deparser, but in the payload pipeline (see Figure 1). These are again inspired from P4-based switches and make it feasible to support different MPI flavors.

**Type 3,** which supports state and loops, uses a plug-in that has an instruction-based reconfigurable compute unit and a recirculate interface (if the switch architecture does not already have one [36]). The key plugin is the interface for off-chip memory, which facilitates stateful processing of HPC applications. To minimize the effort for the design of this type we note that it is equivalent to supporting load/store instructions in the CGRA (see Type 4 and Figure 2). However, a good memory model is needed to fully exploit the capabilities of the hardware without hampering the overall application throughput. We take advantage of multi-banks provided by recent off-chip memory technologies and specify separate data and instruction memory for each SIMD Processing Unit (SPU).

**Type 4,** Since the map operation (computation sandwiched between collectives) can be any user-provided function, the plugin must be programmable. We posit that a coarse grained reconfigurable array (CGRA) is a suitable candidate for the first plugin since both software-like programmability and near-ASIC performance is achieved. Using an instruction-capable CGRA makes the plugin user-friendly and brings the control plane closer to the switch fabric. We divide the CGRA's processing elements into SPUs and then stack these SPUs in a deep pipeline with a simple yet high speed interconnect. We observe that many HPC applications operate on vectors of data and these vectors are typically large. Consequently, the SPUs have wide vector instruction support. For the second plugin, a recirculate interface is used to process a chain of collectives.

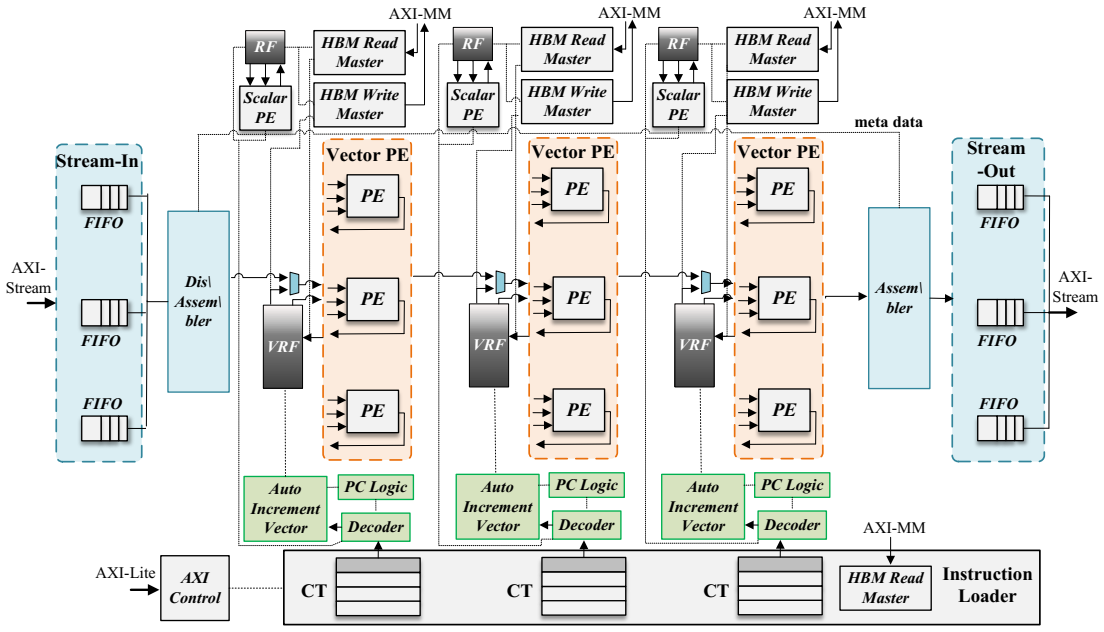


Fig. 2. The proposed CGRA architecture used in Figure 1 with three SIMD processing units (SPUs) in a deep pipeline. It is packaged with AXI interface to facilitate the integration with switch pipelines.

## V. ACiS Experimental Evaluation

In this section, we present the experimental setup and the performance/scalability study of ACiS for collectives, HPC/AI kernels, and applications for different types.

### A. Experimental Setup

For proof-of-concept, we have implemented and tested ACiS on an FPGA-based system in both direct and indirect network settings using the Xilinx *Vitis* unified software platform. We compare ACiS with a high-end CPU cluster.

**Indirect Network:** The testbed is a two-node system on CloudLab [32], [33] with a Xilinx Alveo U280 FPGA attached to a Dell Z9100-ON switch (total of three nodes including host). A 100 Gbps switch interconnects all three nodes. Each process in the leaf nodes, two in this case, in addition to the FPGA itself, is assigned an IP address and a port number. This information is stored in the networking kernel of the FPGA to forward the messages to the correct destination according to the collective type and algorithm. Messages are sent from the leaf nodes to the FPGA through the switch, processed in the FPGA user kernel, and sent back to the corresponding leaf node(s). We also provide a runtime that automates and manages the execution of processes in basic/fused collectives. This includes connecting to leaf nodes from the master process (through SSH), creating processes there, assigning new port numbers for each process, and waiting for the completion. We use the Xilinx Vitis 2021.2 unified software platform to program the FPGA. Our accelerator is coupled with a modified version of [63] to send/receive packets from two leaf nodes. The operating frequency is 250 MHz.

**Direct Network:** In the direct network testbed, two Alveo U280 boards are directly connected using QSFP28 network interfaces (capable of 100 *Gb/s*). Each board is connected to an Intel Xeon E5-2620V2 server. To simulate a larger number of nodes, we conduct an experiment to obtain parameters used for the emulation of a larger-scale proxy system. In this experiment, a sender process sends 1408 bytes worth of data to a receiver process using TCP/IP network logic [34] handled by FPGAs. We used the ExaMPI implementation [57] as the middleware for this experiment. The parameters used in the emulation (derived from our system setup) are shown in Table II. MPI overhead is the average overhead of MPI_send and MPI_Recv in ExaMPI.

For the rest of this section, we evaluate the performance of ACiS on the direct network based on the emulator. The emulation has the same requirements as [43]. That is, the emulation should possess: (1) the same volume of traffic in the network links, (2) an identical number of network hops, and (3) an accurate overhead of the accelerator. For the ACiS accelerator overhead, we use cycle-accurate RTL simulation through testbenches using the Xilinx Vivado Tool. We emulate an FPGA cluster with up to 128 nodes in a 3D-torus.

**CPU Implementation:** For the CPU reference, we use the TACC Stampede2 [59] Skylake (SKX) compute cluster with 48-cores per node (2 sockets) 2.1 $GHz$ Intel Xeon Platinum 8160 CPUs, and a 100 Gb/s Intel Omni-Path (OPA) network. We used Intel MPI 18.0.2 as an Intel-compatible MPI is recommended for this cluster; we found it usually gives better performance than other MPI implementations.

### B. Performance and Scalability

**Type 2:** Fig. 3 shows the simulation results of ACiS and baseline MPI collectives for small (4 Bytes to 4 KB) and medium-to-large message sizes (4 KB to 4 MB) for 32, 64,

TABLE II
PARAMETERS USED IN THE EMULATION OF THE DIRECT NETWORK.
* DENOTES THE AURORA IP LATENCY.

| | |
|---|---|
| MPI Overhead | 14.8 usec |
| Maximum Network Bandwidth (BW) | 95.9 Gbps |
| PCIe Latency | 0.9 usec |
| FPGA-to-FPGA Latency* | 0.44 usec |
| Minimum Port-to-Port Latency | 52 nsec |

and 128 nodes (direct network) using the OSU benchmarks (v5.6.2) [3]. The reported average latency is the average time it takes for the processes to finish the operation. Processor-FPGA communication latency is included in the time. To isolate the impact of the design under study, e.g., from contention at the PCIe interface, we focused simulations with one process per node. As the results suggest, ACiS demonstrates greater performance compared to the CPU cluster baseline.

**Type 3:** We evaluate Type 3 for a Graph Convolution Network (GCN) application on the direct network using four datasets: PPI, Citeseer, Pubmed, ogbn-mag, and ogbn-products. Figure 4 shows the performance and scalability of GCN with and without ACiS acceleration across all datasets. It demonstrates superior scalability of ACiS. On average, ACiS improves application performance compared to a baseline SKX cluster by a factor of 2.2×, 2×, 1.1×, 1.4×, and 10.1× for PPI, Citeseer, Pubmed, ogbn-mag, and ogbn-products on 24 nodes with an average of 3.4× across all datasets.

**Type 4: Indirect Network:** To evaluate the efficacy in indirect network settings we use CloudLab [32] with three nodes (two leaf nodes and one node to host the FPGA) capable of 100 Gbps. Since the runtime and MPI support are based on Python, we compare this approach with a Python-based MPI, MPI4py [14]. We demonstrate its performance compared to traditional MPI for an instance of fused collectives used in FEM applications. Figure 5 shows the latency comparison of Allgather_op_allgather in both MPI4py and ACiS for different message sizes. *Op* here is a prefix sum. The results are from taking the average of five runs. It clearly shows that ACiS provided superior performance, especially for larger message sizes with, on average, a 1.98× improvement. The performance benefit comes from the fact that intermediate communications are bypassed and computations–sandwiched between collectives–are directly processed in the accelerator.

**Direct Network:** We evaluate ACiS against the baseline CPU cluster for NAS parallel benchmark (NPB) [2] and miniFE [1]. NPB includes the following: IS (Integer Sort), LU (Lower-Upper Gauss-Seidel solver), MG (Multi-Grid on a sequence of meshes), and SP (Scalar Penta-diagonal solver). And miniFE is an unstructured implicit finite element code. Figure 6 shows the performance benefit of ACiS over the original MPI implementation on the SKX cluster (64 and 128 nodes) for the NPB and MINIFE proxy applications. SKX time and error bars represent the time it takes on SKX cluster and *std* for five runs. BENCHMARK-X-TY represents the benchmark with *X* nodes and *Y* OpenMP threads. Among NPB applications, the performance benefits for MG and IS are higher than for the others. For IS, one reason is that the message size of collectives is relatively high, and ACiS can take advantage of communication-computation overlap and in-network data reduction. For MINIFE, the performance improvement percentage is typically higher than that of NPB.

## VI. SOFTWARE SUPPORT

Architectural enhancements should ideally be transparent to all other aspects of the system. This inevitably requires software support. Two aspects of this support are described, followed by a database that allows programmers to automatically find where ACiS collectives could collectives.

### A. MPI Transparency

To support MPI transparently, and thus a large fraction of HPC applications, a new transport layer needs to be created using device-specific APIs to communicate with the switch accelerator device. Each supported MPI routine can then be modified to use the new transport (see, e.g., [28]. We have already developed an FPGA transport to communicate between MPI processes through a secondary FPGA-directed network for an ExaMPI implementation [58]. ExaMPI is a light-weight MPI implementation, being developed by one of the PIs, which focuses on key blocks of functionality and new MPI concepts.

### B. Configuring ACiS in the Switch

To facilitate programming and interaction with a switch accelerator directly from an MPI application support for a source-to-source (S2S) translator should be added. The S2S translator includes (1) a parser, (2) a compiler, (3) an assembler, and (4) a wrapper. The input is the source code for an MPI application and the output is new MPI code enhanced with fused collective routines. New routines are added for each fused collective; each is modified with a loader (in addition to Types 1-2 software support) to load the CGRA binary (discussed below) into the switch accelerator.

We now briefly describe its features [29]. It is based on LLVM [40] with the back-end target instruction set being RISC-V (used by the CGRAs). LLVM emits an intermediate representation (IR). In the back-end, target-dependent parameters (e.g., CGRA dimension) are used to apply a number of optimizations. The steps in the back-end include parsing the IR, generating the DFG, code generation, scheduling, and register allocation. Next, an assembler takes the generated instructions and outputs CGRA binary. Finally, the wrapper replaces parts of the application containing the fused collectives with the corresponding fused collective routine. The generated CGRA binary is carried as an argument to the fused collective routine.

### C. Database

In [30] we characterize a large number of MPI applications to determine *overall* applicability of ACiS-supported operations, in both breadth and type, and so provide insight for hardware designers and MPI developers about future offload possibilities. Besides increasing the scope of prior surveys to include finding (potential) new MPI constructs, we also tap into new methods to extend the survey process. Prior surveys

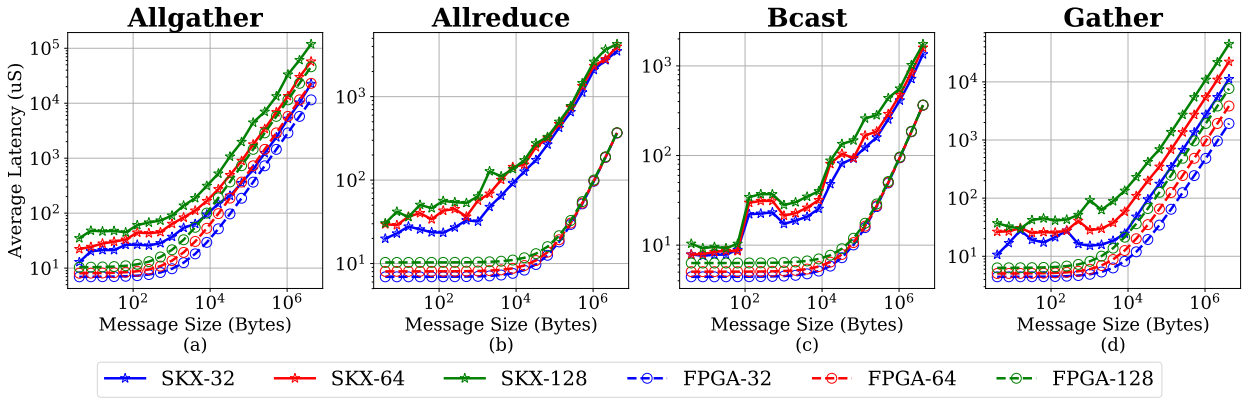


Fig. 3. ACiS vs MPI CPU cluster (SKX) execution times for 32, 64, and 128 nodes: (a) osu_allgather, (b) osu_allreduce, (c) osu_bcast, and (d) osu_gather.

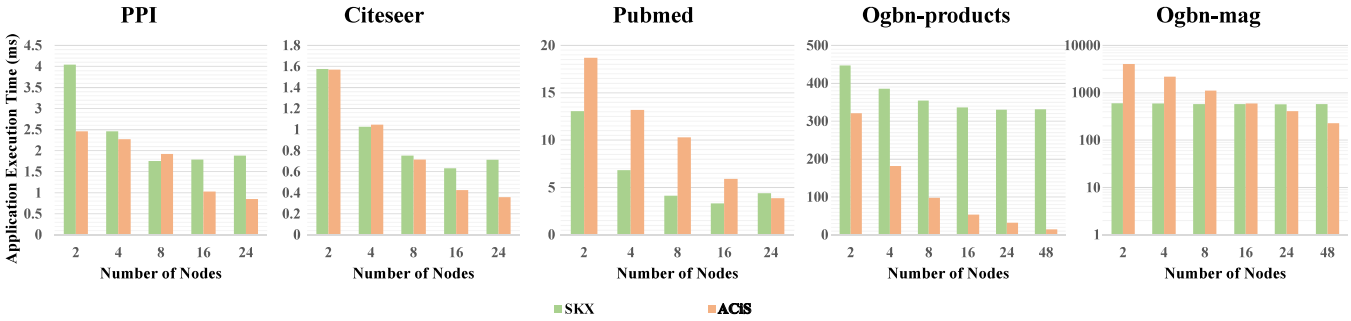


Fig. 4. Application performance and scalability comparison of GCN on a baseline CPU cluster (SKX) vs. ACiS.

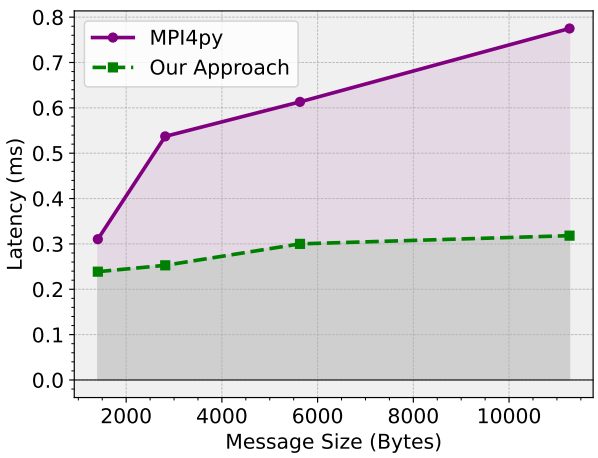


Fig. 5. Latency comparison of Allgather_op_Allgatherv in MPI4py and ACiS. *Op* is prefix sum. The X-axis shows the message size in bytes used for Allgathers and the Y-axis shows the latency in milliseconds.

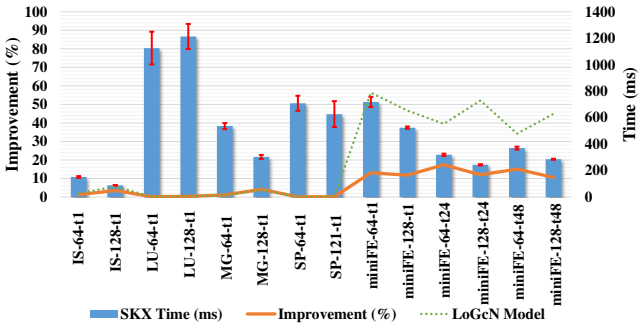


Fig. 6. Performance improvement of ACiS over original MPI implementation on SKX cluster (64 and 128 nodes) for the NAS parallel benchmarks and MINIFE. SKX time and error bars represent the time it takes on SKX cluster and *std* for five runs, respectively.

on MPI usage considered lists of applications constructed based on application developers' knowledge. The we take, however, is based on an automated *mining* of a large collection of code sources. More specifically, the mining is accomplished by GitHub REST APIs. We use a database management system to store the results and to answer queries. Another advantage is that this approach provides support for a more complex analysis of MPI usage, which is accomplished by user queries.

## VII. Conclusion

In this work, we propose a general-purpose, transparent, framework for in-switch computing in reconfigurable devices to provide application-level acceleration. We concentrate on communication collectives and their combinations with each other and with mapping functions. We find that the same basic hardware substrate can be used for multiple extensions to basic collectives: user definition, look-aside (context), and fusion (including some level of control). We describe some of the software support that enables transparency.

## ACKNOWLEDGEMENTS

This work was supported, in part, by the National Science Foundation through awards CCF-1919130, CCF-2151021, and CCF-2326494; and by AMD and Intel both through donated FPGAs, tools, and IP.

## REFERENCES

[1] "ECP Proxy Applications: miniFE Catalog," https://proxyapps.exascaleproject.org/app/minife/.

[2] "NAS Parallel Benchmarks," https://www.nas.nasa.gov/software/npb.html.

[3] "OSU Micro-benchmarks." [Online]. Available: http://mvapich.cse.ohio-state.edu/benchmarks/

[4] G. Almàsi, P. Heidelberger, C. J. Archer, X. Martorell, C. C. Erway, J. E. Moreira, B. Steinmacher-Burow, and Y. Zheng, "Optimization of MPI collective communication on BlueGene/L systems," in *Proceedings of the 19th annual international conference on Supercomputing (ICS'05)*, 2005, pp. 253–262.

[5] Arap, O. and Swany, M., "Offloading Collective Operations to Programmable Logic on a Zynq Cluster," in *High-Performance Interconnects (HOTI), 2016 IEEE 24th Annual Symp. on*, 2016, pp. 76–83.

[6] Intel, "FPGA Programmable Acceleration Card D5005," https://www.intel.com/content/www/us/en/programmable/products/boards_and_kits/dev-kits/altera/intel-fpga-pac-d5005/overview.html [Last accessed: April 29, 2021].

[7] Xilinx, "Alveo SmartNIC Accelerator Card," https://www.xilinx.com/products/boards-and-kits/alveo.html [Last accessed: April 29, 2021].

[8] Xilinx, "Alveo SN1000 Accelerator Card," https://www.xilinx.com/applications/data-center/network-acceleration/alveo-sn1000.html [Last accessed: April 29, 2021].

[9] M. Bayatpour, N. Sarkauskas, H. Subramoni, J. M. Hashmi, and D. K. Panda, "BluesMPI: Efficient MPI Non-blocking Alltoall Offloading Designs on Modern BlueField Smart NICs," in *High Performance Computing*. Springer International Publishing, 2021, pp. 18–37.

[10] ——, "BluesMPI: Efficient MPI Non-blocking Alltoall Offloading Designs on Modern BlueField Smart NICs," in *High Performance Computing*. Springer International Publishing, 2021, pp. 18–37.

[11] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming Protocol-Independent Packet Processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, p. 87–95, 2014.

[12] A. Caulfield, E. Chung, A. Putnam, H. Angepat, J. Fowers, M. Haselman, S. Heil, M. Humphrey, P. Kaur, J.-Y. Kim, D. Lo, T. Massengill, K. Ovtcharov, M. Papamichael, L. Woods, S. Lanka, D. Chiou, and D. Burger, "A cloud-scale acceleration architecture," in *49th IEEE/ACM Int. Symp. Microarchitecture*, 2016, pp. 1–13.

[13] E. Chung, J. Fowers, K. Ovtcharov, M. Papamichael, A. Caulfield, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, M. Abeydeera, L. Adams, H. Angepat, C. Boehn, D. Chiou, O. Firestein, A. Forin, K. S. Gatlin, M. Ghandi, S. Heil, K. Holohan, A. El Husseini, T. Juhasz, K. Kagi, R. K. Kovvuri, S. Lanka, F. van Megen, D. Mukhortov, P. Patel, B. Perez, A. Rapsang, S. Reinhardt, B. Rouhani, A. Sapek, R. Seera, S. Shekar, B. Sridharan, G. Weisz, L. Woods, P. Yi Xiao, D. Zhang, R. Zhao, and D. Burger, "Serving dnns in real time at datacenter scale with project brainwave," *IEEE Micro*, vol. 38, no. 2, pp. 8–20, 2018.

[14] L. Dalcin and Y.-L. L. Fang, "mpi4py: Status update after 12 years of development," *Computing in Science and Engineering*, vol. 23, no. 4, pp. 47–54, 2021.

[15] H. T. Dang, M. Canini, F. Pedone, and R. Soulé, "Paxos made switch-y," *SIGCOMM Comput. Commun. Rev.*, vol. 46, no. 2, p. 18–24, may 2016. [Online]. Available: https://doi.org/10.1145/2935634.2935638

[16] D. DeSensi, S. D. Girolamo, S. Ashkboos, S. Li, and T. Hoefler, "Flare: Flexible in-network allreduce," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '21, 2021.

[17] A. Faraj, S. Kumar, B. Smith, A. Mamidala, and J. Gunnels, "MPI collective communications on the blue gene/P supercomputer: Algorithms and optimizations," *Proceedings - Symposium on the High Performance Interconnects, Hot Interconnects*, pp. 63–72, 2009.

[18] 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 *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. Renton, WA: USENIX Association, Apr. 2018, pp. 51–66. [Online]. Available: https://www.usenix.org/conference/nsdi18/presentation/firestone

[19] R. L. Graham and et al., "Scalable Hierarchical Aggregation Protocol (SHArP): A Hardware Architecture for Efficient Data Reduction," in *2016 Workshop on Communication Optimizations in HPC*, 2016.

[20] R. L. Graham, L. Levi, D. Burredy, G. Bloch, G. Shainer, D. Cho, G. Elias, D. Klein, J. Ladd, O. Maor, A. Marelli, V. Petrov, E. Romlet, Y. Qin, and I. Zemah, "Scalable hierarchical aggregation and reduction protocol (sharp)tm streaming-aggregation hardware design and evaluation," in *High Performance Computing*, P. Sadayappan, B. L. Chamberlain, G. Juckeland, and H. Ltaief, Eds. Cham: Springer International Publishing, 2020, pp. 41–59.

[21] A. Guo, T. Geng, Y. Zhang, P. Haghi, C. Wu, C. Tan, Y. Lin, A. Li, and M. Herbordt, "A Framework for Neural Network Inference on FPGA-Centric SmartNICs," in *International Conference on Field-Programmable Logic and Applications*, 2022, dOI: 10.1109/FPL57034.2022.00071.

[22] ——, "FCsN: A FPGA-Centric SmartNIC Framework for Neural Networks," in *30th IEEE International Symposium on Field-Programmable Custom Computing Machines*, 2022, dOI: 10.1109/FCCM53951.2022.9786193.

[23] A. Guo, Y. Hao, C. Wu, P. Haghi, Z. Pan, M. Si, D. Tao, A. Li, M. Herbordt, and T. Geng, "Software-hardware co-design of heterogeneous smartnic system for recommendation models inference and training," in *ICS 2023: International Conference on Supercomputing*, 2023, dOI = 10.1145/3577193.3593724.

[24] P. Haghi, "ACIS: smart switches with application-level acceleration," Ph.D. dissertation, Department of Electrical and Computer Engineering, Boston University, 2023.

[25] P. Haghi, T. Geng, A. Guo, T. Wang, and M. Herbordt, "FP-AMG: FPGA-Based Acceleration Framework for Algebraic Multigrid Solvers," in *28th IEEE International Symposium on Field-Programmable Custom Computing Machines*, 2020, dOI: 10.1109/ FCCM48280.2020.00028.

[26] ——, "Reconfigurable Compute-in-the-Network FPGA Assistant for High-Level Collective Support with Distributed Matrix Multiply Case Study," in *IEEE Conference on Field Programmable Technology*, 2020.

[27] P. Haghi, A. Guo, T. Geng, A. Skjellum, and M. Herbordt, "Workload Imbalance in HPC Applications: Effect on Performance of In-Network Processing," in *IEEE High Performance Extreme Computing Conference*, 2021, doi: 10.1109/HPEC49654.2021.9622847.

[28] P. Haghi, A. Guo, Q. Xiong, C. Yang, T. Geng, J. Broaddus, R. Marshall, D. Schafer, A. Skjellum, and M. Herbordt, "Reconfigurable switches for high performance and flexible MPI collectives," *Concurrency and Computation: Practice and Experience*, vol. 34, no. 2, 2022, doi: 10.1002/cpe.6769.

[29] P. Haghi, W. Krska, C. Tan, T. Geng, P. Chen, C. Greenwood, A. Guo, T. Hines, C. Wu, A. Li, A. Skjellum, and M. Herbordt, "FLASH: FPGA-Accelerated Smart Switches with GCN Case Study," in *37th ACM International Conference on Supercomputing (ICS)*, 2023, dOI = 10.1145/3577193.3593739.

[30] P. Haghi, R. Marshall, A. Skjellum, and M. Herbordt, "A Survey of Potential MPI Complex Collectives: Large-Scale Mining and Analysis of HPC Applications," 2023.

[31] P. Haghi, C. Tan, A. Guo, C. Wu, D. Liu, A. Li, A. Skjellum, T. Geng, and M. Herbordt, "Smartfuse: Reconfigurable smart switches to accelerate fused collectives in hpc applications," in *38th ACM International Conference on Supercomputing (ICS)*, 2024, dOI: 10.1145/3650200.3656616.

[32] S. Handagala, M. Herbordt, and M. Leeser, "OCT: The Open Cloud FPGA Testbed," in *31st International Conference on Field Programmable Logic and Applications (FPL)*, 2021, doi: TBD.

[33] S. Handagala, M. Leeser, K. Patle, and M. Zink, "Network Attached FPGAs in the Open Cloud Testbed (OCT)," in *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2022, pp. 1–6.

[34] Z. He, D. Korolija, and G. Alonso, "Easynet: 100 gbps network for hls," in *2021 31st International Conference on Field-Programmable Logic and Applications (FPL)*. Los Alamitos, CA, USA: IEEE Computer Society, sep 2021, pp. 197–203. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/FPL53798.2021.00040

[35] T. Hoefler, T. Schneider, and A. Lumsdaine, "The effect of network noise on large-scale collective communications." *Parallel Processing Letters*, vol. 19, pp. 573–593, 12 2009.

[36] S. Ibanez, G. Antichi, G. Brebner, and N. NcKeown, "Event-Driven Packet Processing," in *18th ACM Workshop on Hot Topics in Networks*, 2019.

[37] Inventec, "FPGA SmartNIC C5020X," https://ebg.inventec.com/en/product/Accessories/Smart\%20NIC\%20Card/Inventec\%20FPGA\%20SmartNIC\%20C5020X [Last accessed: April 29, 2021].

[38] M. Jasny, L. Thostrup, T. Ziegler, and C. Binnig, "P4db - the case for in-network oltp," in *Proceedings of the 2022 International Conference on Management of Data*, ser. SIGMOD '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 1375–1389. [Online]. Available: https://doi.org/10.1145/3514221.3517825

[39] X. Jin, X. Li, H. Zhang, R. Soulé, J. Lee, N. Foster, C. Kim, and I. Stoica, "Netcache: Balancing key-value stores with fast in-network caching," in *Proceedings of the 26th Symposium on Operating Systems Principles*, ser. SOSP '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 121–136. [Online]. Available: https://doi.org/10.1145/3132747.3132764

[40] C. Lattner and V. Adve, "LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation," in *Proceedings of the International Symposium on Code Generation and Optimization: Feedback-Directed and Runtime Optimization*, ser. CGO '04. USA: IEEE Computer Society, 2004, p. 75.

[41] 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 *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, 2016.

[42] Y. Li, I.-J. Liu, Y. Yuan, D. Chen, A. Schwing, and J. Huang, "Accelerating distributed reinforcement learning with in-switch computing," in *Proceedings of the 46th International Symposium on Computer Architecture*, 2019, p. 279–291.

[43] ——, "Accelerating Distributed Reinforcement learning with In-Switch Computing," in *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*, 2019, pp. 279–291.

[44] M. Liu, L. Luo, J. Nelson, L. Ceze, A. Krishnamurthy, and K. Atreya, "Incbricks: Toward in-network computation with an in-network cache," *SIGARCH Comput. Archit. News*, vol. 45, no. 1, p. 795–809, apr 2017. [Online]. Available: https://doi.org/10.1145/3093337.3037731

[45] S. Liu, Q. Wang, J. Zhang, Q. Lin, Y. Liu, M. Xu, R. C. C. Chueng, and J. He, "Netreduce: Rdma-compatible in-network reduction for distributed DNN training acceleration," *CoRR*, vol. abs/2009.09736, 2020. [Online]. Available: https://arxiv.org/abs/2009.09736

[46] Mellanox, "Innova-2 Flex Open Programmable SmartNIC," https://www.mellanox.com/files/doc-2020/pb-innova-2-flex.pdf [Last accessed: April 29, 2021].

[47] O. Michel, R. Bifulco, G. Rétvári, and S. Schmid, "The programmable data plane: Abstractions, architectures, algorithms, and applications," *ACM Comput. Surv.*, vol. 54, no. 4, 2021.

[48] T. Morgan, "Intel's best dpu will be commercially available – someday," *The Next Platform*, vol. August 31, 2021, 2021.

[49] Napatech, "FPGA acceleration cards," https://www.napatech.com/products/ [Last accessed: April 29, 2021].

[50] P4 Team, "Web page for P4 Opensource Programming Language," https://opennetworking.org/p4/, accessed 10/30/2022.

[51] T. Perry, "Does the Repurposing of Sun Microsystems' Slogan Honor History, or Step on It?" *IEEE Spectrum)*, no. 30 July, 2019.

[52] R.L. Graham, et al., "Scalable Hierarchical Aggregation Protocol (SHArP): A Hardware Architecture for Efficient Data Reduction," in *First International Workshop on Communication Optimizations in HPC (COMHPC)*, 2016.

[53] G. Sankaran, J. Chung, and R. Kettimuthu, "Leveraging In-Network Computing and Programmable Switches for Streaming Analysis of Scientific Data," in *IEEE NetSoft*, 2021, pp. 293–297.

[54] 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 *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, 2021, pp. 785–808.

[55] A. Shukla, K. N. Hudemann, A. Hecker, and S. Schmid, "Runtime verification of p4 switches with reinforcement learning," in *Workshop on Network Meets AI & ML*, 2019.

[56] Silicom, "FPGA SmartNIC N5010 Series," https://www.silicom-usa.com/pr/fpga-based-cards/fpga-intel-based/fpga-intel-stratix-based/silicom-fpga-smartnic-n5010_series/ [Last accessed: April 29, 2021].

[57] A. Skjellum and et al, "ExaMPI: A Modern Design and Implementation to Accelerate Message Passing Interface Innovation," *Communications in Computer and Information Science*, vol. 1087 CCIS, pp. 153–169, 2020.

[58] A. Skjellum, M. Rüfenacht, N. Sultana, D. Schafer, I. Laguna, and K. Mohror, "Exampi: A modern design and implementation to accelerate message passing interface innovation," in *High Performance Computing*, J. L. Crespo-Mariño and E. Meneses-Rojas, Eds. Cham: Springer International Publishing, 2020, pp. 153–169.

[59] D. Stanzione, B. Barth, N. Gaffney, K. Gaither, C. Hempel, T. Minyard, S. Mehringer, E. Wernert, H. Tufo, D. Panda, and P. Teller, "Stampede 2: The Evolution of an XSEDE Supercomputer," in *Practice and Experience in Advanced Research Computing on Sustainability, Success and Impact*, 2017.

[60] T. Swamy, A. Rucker, M. Shahbaz, I. Gaur, and K. Olukotun, "Taurus: A data plane architecture for per-packet ml," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS 2022, 2022, p. 1099–1114.

[61] K. D. Underwood, J. Coffman, R. Larsen, K. S. Hemmert, B. W. Barrett, R. Brightwell, and M. Levenhagen, "Enabling flexible collective communication offload with triggered operations," in *IEEE 19th Symposium on High Performance Interconnects*, 2011, pp. 35–42.

[62] F. L. Verdi and M. Chiesa, "Heavy hitter detection on multi-pipeline switches," in *Symp. on Architectures for Networking and Communications Systems*, 2022.

[63] Xilinx, "XUP Vitis Network Example (VNx)," https://github.com/Xilinx/xup_vitis_network_example, 2023.

[64] Q. Xiong, C. Yang, R. Patel, T. Geng, A. Skjellum, and M. Herbordt, "GhostSZ: A Transparent SZ Lossy Compression Framework with FPGAs," in *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2019, pp. 258–266, doi: 10.1109/FCCM.2019.00042.

[65] H. Yoshida, "How is Data Ops Related to Data Centric Computing?" Hitachi Blog, https:// community.hitachivantara.com/ blogs/hubert-yoshida/2020/10/14/ how-is-data-ops-related-to-data-centric-computing, 2020.