FPGA-based AI Smart NICs for Scalable Distributed AI Training Systems

Rui Ma, *Member, IEEE,* Evangelos Georganas, Alexander Heinecke, Sergey Gribok, *Member, IEEE,* Andrew Boutros, *Member, IEEE,* Eriko Nurvitadhi, *Member, IEEE,*

Abstract—Training state-of-the-art artificial intelligence (AI) models requires scaling to many compute nodes and relies heavily on collective communication operations, such as all-reduce, to exchange the weight gradients between nodes. The overhead of these operations can bottleneck training performance as the number of nodes increases. In this paper, we first characterize the all-reduce operation overhead. Then, we propose a new smart network interface card (NIC) for distributed AI training using field-programmable gate arrays (FPGAs) to accelerate all-reduce operations and optimize bandwidth utilization via data compression. The AI smart NIC frees up the system's compute resources to perform the more compute-intensive tensor operations and increases the overall node-to-node communication efficiency. We build a prototype 6-node AI training system and show that our proposed FPGA-based AI smart NIC enhances overall training performance by 1.6×, with an estimated 2.5× performance improvement at 32 nodes.

Index Terms—AI training, all-reduce, smart NIC, FPGA

1 Introduction

Advances in artificial intelligence (AI) have led to unprecedented quality of results in a myriad of domains. However, to achieve this level of performance on practical tasks, the compute complexity and memory footprint of deep neural networks (DNNs) are rapidly increasing. For example, state-of-the-art natural language processing models have grown in size by more than 5000× in less than five years [1]. Therefore, the training of such huge DNNs on massive amounts of data is only feasible on large clusters of many compute nodes (e.g. CPUs, GPUs or custom chips). Data parallel training is a commonly used scheme for distributed learning, in which each node in the system trains the model on a different mini-batch of the training data. The calculated weight gradients are then periodically aggregated and used by all nodes to update the model weights.

Efficient scaling of AI training to large distributed systems is challenging. Ideally, we aim to maximize the utilization of the compute nodes performing the key compute-intensive tensor operations in the forward and backward propagations. However, collective communication operations, such as all-reduce, are required to aggregate the calculated weight gradients from different nodes and update the model weights before starting the next training iteration. These all-reduce operations consume a portion of the compute node resources, reducing scalability and degrading the overall system performance as less compute resources are available for executing the core tensor operations.

This work proposes offloading the all-reduce operations from the compute nodes of a distributed training system to an AI-targeted smart network interface card (NIC) implemented on a field-programmable gate array (FPGA). This allows the compute nodes to focus primarily on the core tensor operations they are optimized for, while inter-node collective communication is performed simultaneously by the FPGAs. Moreover, the reconfigurability of FPGAs enables implementing custom application-specific optimizations; our AI smart NIC also performs line rate compression of weight gradients into custom narrow block floating-point (BFP) formats as in Microsoft's cloud AI accelerators [2]. Our additional AI-specific smart NIC functionality consumes only a small portion of the FPGA resources, making it suitable for adoption in existing FPGA smart NICs [3] or infrastructure processing units (IPUs) targeted for cloud deployments [4]. Our contributions in this paper are:

- quantifying all-reduce communication overhead in conventional AI distributed training systems,
- implementing an FPGA-based AI smart NIC to offload all-reduce communication freeing up workers (e.g. CPUs, GPUs) for compute-intensive tensor operations, and perform gradient compression for efficient network bandwidth utilization[1],
- demonstrating that our AI smart NIC enhances training performance by up to 1.6× in a 6-node prototype system,
- formulating a detailed analytical performance model showing performance gains up to 2.5× for larger 32-node systems.

2 Background

DNN Training: Supervised training of DNNs uses a labeled application-specific training dataset, typically split into smaller groups (i.e. *mini-batches*). Training is comprised of three steps: (1) In a *forward pass*, the DNN computes a prediction for each input in the mini-batch and the prediction error compared to ground truth label is calculated. (2) Then, the error is propagated through all layers in a *backward pass* to calculate weight gradients. (3) Finally, weights are updated using calculated gradients and an optimizer update rule to minimize the prediction error. This process is repeated for all mini-batches (an *epoch*), and then multiple epochs are performed until accuracy converges.

Distributed Training Systems: For large state-of-the-art DNNs, the training procedure is typically scaled out on a distributed system with many *workers*. These workers can be CPUs, GPUs, custom accelerators, or a mix of them. In this work, we focus on the most commonly used approach for distributed training which is *data parallelism*. In this approach, different workers train the model using different mini-batches of training data. Then, they exchange their learning (i.e. weight gradients) after every N training steps to synchronize weight values among themselves using an *all-reduce* operation. The all-reduce operation aggregates data from different workers using an associative operation (e.g. summation) and then distributes the result back to all workers. It can be implemented in many different approaches such as tree, round-robin, butterfly, and ring topologies. In this work, we implement a pipelined ring

1. Our FPGA-based AI smart NIC implementation is open-sourced on Github at `https://github.com/libxsmm/fpga_ai_nic`

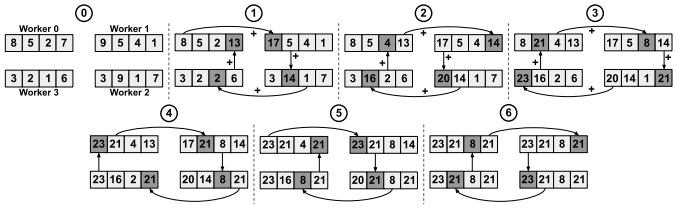

Fig. 1: Pipelined ring all-reduce with four workers ($w = 4$).

all-reduce which is contention-free and bandwidth optimal at the cost of linear latency scaling with the number of workers [5]. As illustrated in Fig. 1, each of the w workers sends/receives data to/from one of its neighbors over $2(w-1)$ time steps. In the first $(w-1)$ steps, each worker aggregates the data received from the previous node with its corresponding locally buffered data. In the remaining steps, workers replace their buffered data with the received final results.

3 Related Work

FPGA-based smart NICs have been commercially deployed to accelerate network functionality on a large scale in the Microsoft Azure cloud [3]. The reconfigurability of FPGAs combines software-like programmability with hardware-like efficiency unlike custom application-specific chips that lack the flexibility, and general-purpose processors (CPUs/GPUs) that trade efficiency for generality [6]. Intel IPUs are another example for FPGA network acceleration; they combine a Xeon CPU with an FPGA to accelerate cloud infrastructure, freeing up the server CPU resources for key datacenter workloads [4]. These solutions accelerate application-agnostic network functionality and, unlike our work, do not target AI training. FPGAs have also been used to implement in-network processing specifically for distributed AI training. Prior work proposed the implementation of FPGA-based switches that accelerates weight gradient aggregation on-the-fly as data is transferred between system workers [7], [8]. Unlike these works, our study uses FPGAs as smart NICs in distributed training. Tanaka et al. [9] propose the use of FPGA-based smart NICs to accelerate all-reduce operations in distributed training. However, unlike our work, they do not use the reconfigurable FPGA fabric for weight gradient compression to increase node-to-node communication efficiency. In addition, besides measurements on our prototype system, we also formulate an accurate performance model to evaluate the scalability of our solution for larger systems.

4 Profiling Distributed AI Training

To quantify the overhead of all-reduce operations in distributed training, we experiment with training feedforward multi-layer perceptrons (MLPs), which represent a major component of many contemporary AI workloads (e.g. recommender systems [10]). We choose MLPs as a representative example however, the same training framework is applicable to essentially all other deep learning models. Our experimental setup is a cluster of Intel Xeon Platinum 8280 28-core CPUs, connected via a 100 Gbps network switch. We first deploy a naive solution in which all 28 cores per CPU are performing backward pass tensor operations and when required, one thread triggers an asynchronous all-reduce operation and all the threads (if needed) wait for it to be finished. This approach exposes the all-reduce latency on the critical path of the training procedure. Then, we implement a more optimized solution where a number of threads on each CPU are dedicated to handle the all-reduce operation and weight updates in parallel to the remaining threads performing backward pass tensor operations. This overlap hides most of all-reduce latency behind tensor operations of the backward pass and significantly improves the overall system performance. Although our setup is based on CPU workers as a proof-of-concept, the same approach and

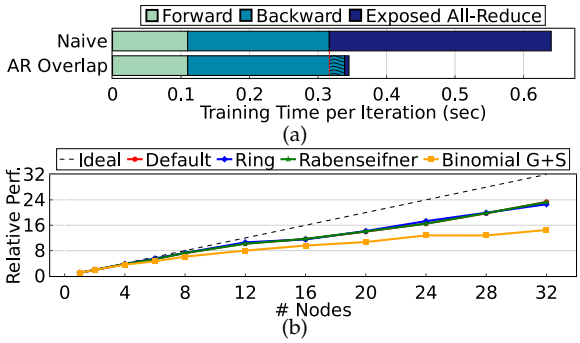

Fig. 2: (a) Breakdown of training iteration time for 20-layer MLP on a 6-node system with/without overlapping all-reduce (AR) with backward pass compute. The increase in backward pass time (shaded in black) in the overlapped implementation is due to dedicating a portion of the node resources for managing network communication and executing all-reduce. (b) Scaling of overlapped implementation for different all-reduce schemes.

trends apply for network-attached GPU workers (e.g. Nvidia GPUs using GPUDirect RDMA [11] in DGX systems).

Fig. 2a compares the training iteration time of both approaches for a 20-layer MLP with matrices of size 2048×2048 and mini-batch size of 1792 when trained on 6 workers in our CPU cluster. The exposed all-reduce latency constitutes 51% of the total training time of the naive implementation. For the overlapped implementation, the exposed all-reduce time is 50× less than the naive approach, resulting in a 1.85× reduction in the overall training time. However, the backward pass execution time increases by 11% (shaded portion in Fig. 2a) since fewer resources are dedicated for the compute-intensive tensor operations. In this specific experiment, we found that dedicating 2 cores for communication (all-reduce) and 26 cores for compute (backward pass) yields the best performance. However, this balance between compute/communication cores is workload dependent and needs to be tuned based on number and sizes of layers, mini-batch size and number of workers. In some cases, the backward pass slowdown can be larger, reducing the overall gains of overlapping communication and compute.

Fig. 2b shows the results of scaling the overlapped implementation to more workers in our CPU cluster compared to the ideal scaling shown by the dashed line. We experiment with different MPI all-reduce schemes such as the ring, Rabenseifner, and binomial gather/scatter implementations as well as the default setting which employs heuristics to use different algorithms based on message sizes and number of workers [12]. The results show that the default, ring and Rabenseifner schemes achieve similar results and are consistently better than the binomial gather/scatter approach. They also scale well for up to 12 workers, but the gap to ideal scaling gradually increases with the number of workers. For the rest of our experiments, we use the ring all-reduce scheme which matches what we implement in our system with AI smart NICs.

5 FPGA-Based AI Smart NIC

In this work, we enhance a conventional distributed training system with FPGA-based AI smart NICs to which worker nodes (CPUs or GPUs) can offload all-reduce communication and free up resources for the more compute-intensive tensor operations increasing the efficiency of the overall system. We also leverage the FPGA programmability to implement on-the-fly gradient compression for efficient network bandwidth utilization.

5.1 System Overview and AI Smart NIC Architecture

Fig. 3a shows an overview of our system enhanced with AI smart NICs. An FPGA is attached to each worker via PCIe, and

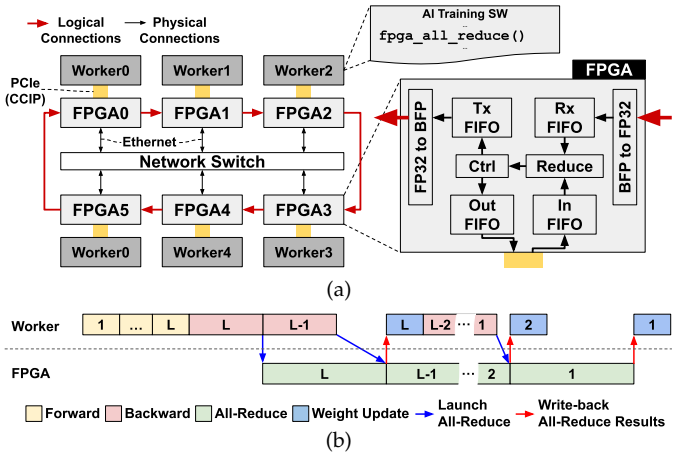

Fig. 3: (a) System overview and AI smart NIC architecture, and (b) Example execution trace for L-layer MLP training.

all the FPGAs are connected via a network switch. A ring topology is implemented between the FPGAs on top of the Ethernet layer as shown by the red logical connections in the figure. Fig. 3b depicts an example execution trace for training an L-layer MLP on our enhanced system. After each worker executes the forward pass and the backward propagation of the last layer, a thread launches a non-blocking all-reduce request specifying the start memory address and number of the weight gradients to be reduced. Meanwhile, the workers progress to execute the backward propagation of the next layer, then execution is blocked until the FPGAs finish the all-reduce. After that, the workers perform weight update and backward propagation of the next layer while the FPGAs are busy with the next all-reduce, which repeats until the complete backward pass is finished.

A simplified diagram of our AI smart NIC architecture is also shown in Fig. 3a. We adopt the same pipelined ring all-reduce algorithm described previously in Section . The FPGA first reads weight gradients from its local worker and buffers them in the input FIFO, while the second set of operands of the reduction operation arrives from the previous node over Ethernet and is buffered in the receive (Rx) FIFO. Once both FIFOs are ready, their contents are dequeued and then reduced using a set of FP32 adders. Finally, a control FSM (Ctrl) steers results to be sent to the next node in the ring via the transmit (Tx) FIFO and/or write it back to the local worker memory as the final all-reduce result via the output FIFO.

5.2 Block Floating Point Compression

BFP format can reduce the computational complexity and memory footprint of DNNs, with minimal impact on accuracy [2], [13]. In this format, floating point weights or activations are split into small blocks sharing the same exponent value among different mantissas. The block size and bitwidth of the mantissas and shared exponent are all parameters of the BFP format and represent a tradeoff between efficiency and model accuracy. Our AI smart NIC can exploit the reconfigurability of FPGAs to implement other application-specific optimizations such as BFP compression. As shown in Fig. 3a, weight gradients are transferred over the network between FPGAs in BFP format; when recieved by the AI smart NIC, they are decompressed from BFP into FP32 to perform reduction and/or be written back to the worker's memory. Then, they are compressed again to BFP on the way out when sent over the network to the next node. In our experiments, we use the BFP16 format from [2] with 8-bit shared exponents, 7-bit mantissas and a block size of 16 elements, which achieves 3.8× compression ratio. However, these parameters can be flexibly adjusted to better suit different workloads making use of the FPGA's flexibility.

5.3 Performance Model

We develop an analytical model to estimate the performance of our system enhanced with AI smart NICs for MLP training. Using this model, we can study the scalability of our AI smart NIC solution and also perform system architecture exploration by conducting what-if analyses for different system parameters (e.g. worker compute capabilities, network bandwidth). Assuming that the MLP has L layers each of which has a symmetric weight matrix of dimensions $M_l \times M_l$ for $1 \leq l \leq L$, training mini-batch size is B, and the worker's compute throughput is P_{Worker} in FLOPS. Then the forward (T_F) and backward (T_B) propagation runtimes for layer l can be calculated as:

$$T_{F_l} = \frac{2 \times M_l^2 \times B}{P_{Worker}} \qquad T_{B_l} = \frac{4 \times M_l^2 \times B}{P_{Worker}}$$

For a pipelined ring all-reduce for a system with N nodes, the weight gradients are padded (if needed) and evenly split into N chunks. Therefore, if b is the reduction bitwidth, then the total number of bits all-reduced per node for layer l is:

$$R_l = b \times N \times \left\lceil \frac{M_l^2}{N} \right\rceil$$

We assume that the AI smart NIC can utilize a portion α of the Ethernet bandwidth BW_{eth} and we can achieve a compression ratio β using the BFP format as discussed in Sec. 5.2. Then, the all-reduce runtime is bottlenecked by one of 3 factors: (1) time for transferring data through the ring T_{ring}, (2) time for addition T_{add}, or (3) time for receiving and writing back data to the worker memory T_{mem}. If PCIe bandwidth is BW_{pcie} and the FPGA addition throughput is P_{FPGA} in FLOPS, we can calculate these factors and the total all-reduce time for layer l as:

$$T_{ring_l} = \frac{R_l \times 2(N-1)}{N \times \alpha BW_{eth} \times \beta} \qquad T_{add_l} = \frac{R_l \times 2(N-1)}{N \times P_{FPGA} \times b}$$

$$T_{mem_l} = \frac{2 \times Rl}{BW_{pcie}} \qquad T_{AR_l} = \max(T_{ring_l}, T_{add_l}, T_{mem_l})$$

Finally, we measure the weight update time executed by the worker (T_U), which depends mainly on the layer size and memory bandwidth, and linearly scale it for any given layer size. To calculate the total training iteration time, we use the components $T_{F_l}, T_{B_l}, T_{AR_l}$ and T_{U_l} to sum up the components of the training trace (similar to that in Fig. 3b) as follows. Our results show that our analytical model can estimate system performance within 3% of the real measurements on our prototype system.

$$T_{total} = \left[\sum_{l=1}^{L} T_{F_l}\right] + T_{B_L} + \max(T_{B_{L-1}}, T_{AR_L}) +$$

$$\left[\sum_{l=2}^{L-1} \max(T_{U_{l+1}} + T_{B_{l-1}}, T_{AR_l})\right] + \max(T_{U_2}, T_{AR_1}) + T_{U_1}$$

6 Evaluation

6.1 Implementation Details

We implement a prototype 6-node system to evaluate the performance gains of our proposed AI smart NIC. Each node consists of an Intel Xeon Platinum 8280 28-core CPU attached to an Intel Arria 10 1150 FPGA via PCIe Gen3x8. The FPGAs are connected to a Dell EMC S6100-ON switch via 40 Gbps Ethernet links, on top of which a ring topology is implemented as shown in Fig. 3a. We use Intel's IKL for direct inter-FPGA network communication [14] and Intel's OPAE stack for CPU-FPGA communication over PCIe [15]. Although our prototype system uses CPU workers, the AI smart NIC enhancement is applicable to systems with GPU (or custom accelerator) workers. We implement the FPGA-based AI smart NIC in parameterizable Verilog RTL and use Quartus Prime Pro 17.1 to synthesize, place and route the AI smart NIC logic along with the OPAE and IKL shim.

TABLE 1: FPGA resource breakdown (ALMs: logic blocks, M20Ks: 20Kb RAMs, DSPs: digital signal processing blocks).

	ALMs	M20Ks	DSPs
OPAE + IKL Shim	64,480 (15.1%)	368 (13.6%)	0 (0%)
All-Reduce	2,233 (0.5%)	46 (1.7%)	8 (0.5%)
BFP Compression	2,857 (0.7%)	120 (4.4%)	0 (0%)
Total	**69,570 (16.3%)**	**534 (19.7%)**	**8 (0.5%)**

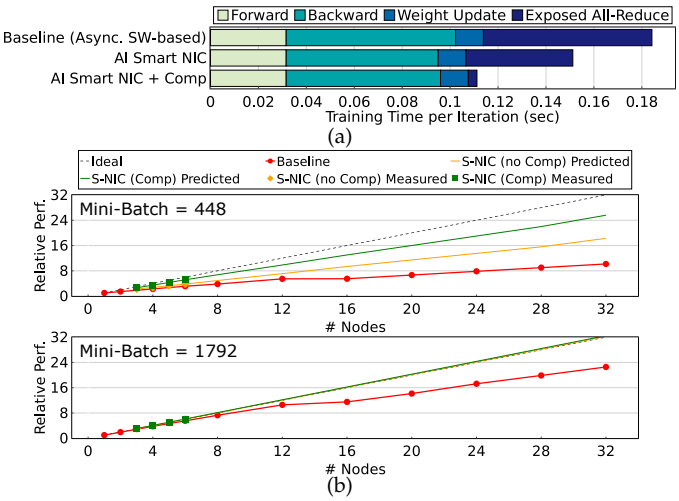

Fig. 4: (a) Training time per iteration breakdown of 20-layer MLP and mini-batch of size 448 on a 6-node system for baseline system using optimized asynchronous software-based all-reduce and our AI smart-NIC-enhanced system with and without BFP compression. (b) Performance scaling for systems with more nodes compared to ideal scaling (black dashed line).

Table 1 lists the FPGA resource utilization breakdown for our AI smart NIC, showing that the additional AI-targeted functionality (all-reduce and BFP compression) are very lightweight; they constitute only 1.2%, 6.1%, and 0.5% of the FPGA's logic, RAM, and DSP resources and thus can be added to existing FPGA smart NICs deployed in production as in the Microsoft Azure cloud [3] with minimal resource overhead. We also customize our AI smart NIC implementation to match higher speed 100 Gbps and 400 Gbps network interfaces. Even with 400 Gbps interfaces, our additional AI-specific functionality consumes less than 2%, 9%, and 5% of the FPGA logic, RAMs, and DSPs.

6.2 Performance Results

Fig. 4a compares the training iteration runtime on a 6-node system when using conventional NICs (baseline) and when using our FPGA-based AI smart NIC with and without BFP compression. For this experiment, we train a 20-layer MLP with 2048×2048 layers and mini-batch size 448. For the baseline system, we use the optimized implementation overlapping backward propagation compute and all-reduce. The results show that the FPGA AI smart NIC, even without using the BFP compression, can reduce the exposed all-reduce time by 37% and free up the workers' compute resources to focus on the tensor operations of the backward pass reducing its execution time by 10%. Overall, this results in an 18% reduction in the total training time. These gains are further increased when deploying the BFP compression on the AI smart NIC; the exposed all-reduce time and total training time are reduced by 95% and 40% respectively compared to the baseline system due to the more efficient utilization of inter-node communication bandwidth.

Fig. 4b shows how the training throughput (normalized to performance of a system with only 1 worker) scales with increasing the number of nodes for mini-batch sizes of 448 (top) and 1792 (bottom). The ideal scaling of system performance is showed as a black dashed line for reference. For the system enhanced with our AI smart NIC, we measure performance for 3, 4, 5 and 6 nodes on our prototype system, then use our analytical model from Sec. 5.3 to predict performance for more nodes. The figures show that measured performance on our prototype system is within 3% error from that predicted by our analytical model. For mini-batch size 448, the plot shows that system performance shifts more towards ideal scaling performance (dashed line) when using the FPGA AI smart NIC and when using BFP compression, with performance gains up to 2.5× and 1.8× with and without BFP compression respectively.

For mini-batches of size 1792, the system with AI smart NIC achieves ideal scaling as shown in the bottom plot of Fig. 4b. In this case, BFP compression provides no additional benefits since for higher batch sizes, the performance is already bottlenecked by the backward propagation compute when using the FPGA AI smart NIC. For this batch size, the AI smart NIC increases the overall system performance by 1.1× and 1.4× compared to the baseline for systems with 6 and 32 nodes, respectively. These performance gains are achieved by the AI smart NIC despite using a much lower inter-node network bandwidth of 40 Gbps, compared to the 100 Gbps network used in the baseline system.

7 Conclusion

The all-reduce collective communication in distributed AI training can throttle the overall system performance since a portion of the compute resources of each worker is dedicated for performing reduction operations and managing network communication. To alleviate this bottleneck, we propose an FPGA-based AI smart NIC to accelerate all-reduce operations and free up the worker resources for more compute-intensive tensor operations. We also exploit the FPGA flexibility by implementing custom BFP weight gradient compression for efficient inter-node communication. Our 6-node prototype system demonstrates that the AI smart NICs can reduce the overall training time by 40%, while adding minimal resources to existing FPGA smart NICs for the AI-specific functionalities. We also formulate a detailed model to study system scalability, showing that our AI smart NIC can increase performance by up to 2.5× in 32-node systems.

References

[1] A. Chowdhery *et al.*, "PaLM: Scaling Language Modeling with Pathways," *arXiv:2204.02311*, 2022.

[2] B. Darvish Rouhani *et al.*, "Pushing the Limits of Narrow Precision Inferencing at Cloud Scale with Microsoft Floating Point," 2020.

[3] D. Firestone *et al.*, "Azure Accelerated Networking: SmartNICs in the Public Cloud," in *NSDI*, 2018.

[4] B. Burres *et al.*, "Intel's Hyperscale-Ready Infrastructure Processing Unit (IPU)," in *Hot Chips Symposium (HCS)*, 2021.

[5] P. Patarasuk and X. Yuan, "Bandwidth Optimal All-Reduce Algorithms for Clusters of Workstations," *Journal of Parallel and Distributed Computing*, vol. 69, no. 2, pp. 117–124, 2009.

[6] A. Boutros and V. Betz, "FPGA Architecture: Principles and Progression," *IEEE Circuits and Systems Magazine*, vol. 21, no. 2, 2021.

[7] T. Itsubo *et al.*, "Accelerating Deep Learning Using Multiple GPUs and FPGA-based 10GbE Switch," in *PDP*, 2020.

[8] Y. Li *et al.*, "Accelerating Distributed Reinforcement Learning with In-switch Computing," in *ISCA*, 2019.

[9] K. Tanaka *et al.*, "Distributed Deep Learning With GPU-FPGA Heterogeneous Computing," *Micro*, vol. 41, no. 1, pp. 15–22, 2020.

[10] D. Kalamkar *et al.*, "Optimizing Deep Learning Recommender Systems Training on CPU Cluster Architectures," in *SC*, 2020.

[11] Nvidia Corp., "NVidia GPUDirect RDMA," *https://docs.nvidia.com/cuda/gpudirect-rdma/index.html*.

[12] R. Thakur *et al.*, "Optimization of Collective Communication Operations in MPICH," *The International Journal of High Performance Computing Applications*, vol. 19, no. 1, pp. 49–66, 2005.

[13] M. Drumond *et al.*, "Training DNNs with Hybrid Block Floating Point," *NeurIPS*, 2018.

[14] S. M. Balle *et al.*, "Inter-Kernel Links for Direct Inter-FPGA Communication," *Intel White Paper (WP-01305-1.0)*, 2020.

[15] Intel Corp., "Intel Open Programmable Acceleration Engine (OPAE)," *https://opae.github.io/*.