SmartReduce: Efficient Sparse AllReduce via Low-Latency SmartNIC-Switch Coordination

Yingtao Li*, Hongjing Huang‡, Changyue Liao*, Zihan Yang*, Mo Sun*,
Xueying Zhu*, Jie Zhang*, Bo Zhao†, Zeke Wang*
*Zhejiang University, Hangzhou, China
{Li_Yingtao, changyueliao, zihanyang, sunmo, zhuxueying, carlzhang4, wangzeke}@zju.edu.cn
†Aalto University, Espoo, Finland
bo.zhao@aalto.fi
‡Minjiang University, Fuzhou, China
huanghj@mju.edu.cn

***Abstract*—Sparse AllReduce is a promising primitive for accelerating distributed machine learning training with sparse gradients, but existing designs fail to deliver high algorithm bandwidth. The root cause is their reliance on CPU-initiated coordination, which imposes high packet-processing overhead during aggregation and creates severe load imbalance across CPU cores, limiting effective exploitation of sparsity. We observe that sustaining high-throughput sparse AllReduce requires removing the CPU from both the coordination path and the fine-grained sparsification path. Guided by this insight, we present SMARTREDUCE, a SmartSwitch-aware, SmartNIC-centric sparse AllReduce. SMARTREDUCE combines two key mechanisms: (i) a switch-aware, NIC-initiated aggregation mechanism that performs in-network coordination between SmartNICs and the SmartSwitch with stable sub-RTT overhead, and (ii) a NIC-centric fine-grained sparsification mechanism that processes fine-grained blocks in high parallelism to exploit intra-block sparsity. Experimental results show that SMARTREDUCE at 99% sparsity achieves up to $6.4\times$ higher algorithm bandwidth than the state-of-the-art aggregator-server-based sparse AllReduce method.**

***Index Terms*—high performance computing, distributed computing, hardware acceleration, network interfaces**

I. INTRODUCTION

AllReduce is a fundamental collective primitive in machine learning (ML) systems [1]–[26]. It aggregates data across distributed workers and disseminates the result to all participants. When aggregated values are sparse, sparse AllReduce can substantially improve aggregation throughput by communicating only non-zero entries [27]–[29]. Such sparsity is common in modern ML workloads. Prior studies report that up to 99.9% of aggregated gradients can be zero in distributed pre-training [30]–[42], while fine-tuning is even sparser because only a small subset of parameters is updated [43], [44]. For instance, parameter-efficient fine-tuning methods such as LoRA typically update less than 0.1% of model parameters [45]–[47]. Existing techniques such as gradient clipping [48]–[50], gradient pruning [51]–[53], and error compensation [54]–[56] further exploit this sparsity to reduce communication volume without compromising model accuracy.

To quantify the end-to-end efficiency of sparse AllReduce, we use *algorithm bandwidth* (algbw), defined as the logical input data size reduced by an AllReduce operation divided by its end-to-end completion time. Existing sparse AllReduce systems can be grouped into three categories.

AllGather-based sparse (AG-Sparse) AllReduce is the conventional design point [57]–[60]. It aggregates sparse data by gathering non-zero elements from all nodes and then reducing them locally. However, AG-Sparse fails to achieve high algorithm bandwidth due to two severe issues. First, as the number of nodes increases, the communication overhead of both AllGather and local Reduce increases, resulting in low algbw (**P1**). Second, sparse local Reduce requires irregular and memory-intensive accesses, which cause severe cache thrashing and high latency (**P2**).

Aggregator-Server-based sparse (AS-Sparse) AllReduce is proposed to address the scalability and streaming limitations of AG-Sparse [27], [29]. Its key operation is *coordination*. Unlike dense aggregation, where all participants contribute uniformly, sparse AllReduce must repeatedly determine which non-zero blocks to send, where to send them, and when workers can proceed under data-dependent sparsity patterns. This introduces a non-trivial coordination problem on the critical path. We refer to the latency of each such coordination round as *coordination cost*. As a result, even though sparse AllReduce reduces communication volume, its end-to-end performance is often bounded by coordination cost rather than raw network bandwidth. The following two issues lead to high coordination cost in AS-Sparse AllReduce.

- **P3. Network Amplification from Using Large Blocks.** AS-Sparse AllReduce relies on a CPU-controlled coarse-grained sparsification method. As a result, redundant zeros sandwiched between non-zero elements within the same large block are still transmitted over the network, preventing full exploitation of intra-block sparsity.
- **P4. Requirement of Extra Aggregators to Avoid Incast.** To prevent incast at the aggregators, AS-Sparse AllReduce requires the number of aggregator servers to be no fewer than the number of worker nodes. Otherwise, aggregation throughput drops substantially.

SmartNetwork-assisted sparse (SNA-Sparse) AllReduce is proposed to address the aggregator requirement of AS-Sparse by introducing SmartSwitch-based in-network aggre-

SC26, November 15-20, 2026, Chicago, Illinois, USA
979-8-3195-4789-7/26/$31.00 ©2026 IEEE

gation [27], [28]. However, this design introduces a new constraint: the maximum block size is limited by SmartSwitch resources (*e.g.*, stateful ALUs [61], [62]). In practice, such restriction forces SNA-Sparse to use small blocks, exposing the fundamental limitations of CPU-initiated coordination.

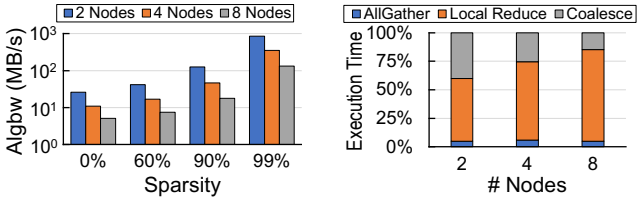
- **P5. Increased Coordination Cost with More Concurrent Blocks.** To sustain high throughput with small blocks, SNA-Sparse AllReduce uses multiple CPU cores to issue concurrent block aggregations. However, increasing parallelism causes coordination cost to grow superlinearly because of the inherent architectural limitations of CPU-side coordination, offsetting the benefit of additional concurrency.
- **P6. Severe Load Imbalance between Cores.** The distribution of sparse blocks across CPU cores is often imbalanced, and end-to-end latency is determined by the slowest core, i.e., the one that processes the largest number of coordination rounds. This imbalance leads to high tail latency and poor resource utilization.

Fundamentally, *CPU-initiated coordination* is the bottleneck in existing sparse AllReduce designs based on block-wise streaming aggregation algorithm. By placing fine-grained sparse coordination on the CPU, existing systems pay high coordination cost exactly where sparse aggregation is most latency-sensitive, which in turn limits throughput and weakens the benefit of sparsity. This bottleneck motivates offloading sparse collectives to dedicated network accelerators. Modern SmartNICs and SmartSwitches have become increasingly capable on-path compute resources and attractive complements to host CPUs for high-performance collective communication [63]–[66]. Yet existing designs typically employ only one type of accelerator in isolation, rather than co-designing heterogeneous network devices to jointly execute sparse aggregation. Consequently, the full performance potential of accelerator-driven sparse AllReduce remains unrealized.

We present **SmartReduce**, a SmartSwitch-aware, SmartNIC-centric sparse AllReduce that removes CPU-initiated coordination from the critical path. The key idea is to co-design the SmartNIC and the SmartSwitch so that sparse aggregation is initiated directly by the NIC, while the switch provides low-latency in-network aggregation. By moving both coordination and data movement onto a hardware pipeline close to the network path, SmartReduce achieves stable sub-round-trip-time (sub-RTT) coordination cost and supports highly concurrent sparse aggregation.

SmartReduce consists of two key mechanisms.

I1. Switch-Aware NIC-Initiated Aggregation Mechanism. To minimize the coordination cost of sparse block aggregation, we propose the Switch-Aware NIC-Initiated Aggregation Mechanism, implemented by On-NIC Coordination (ONC) engines in the SmartNIC and matching aggregators in the SmartSwitch. The ONC engines directly issue sparse aggregation requests from the SmartNIC and coordinate with the switch aggregators. This design shortens the critical path of sparse AllReduce to an on-path hardware path between NIC and switch. Compared to prior SmartNetwork-assisted designs, SmartReduce sustains high concurrency without increasing coordination cost.



(a) Algbw of AG-Sparse. (b) Breakdown under 99% sparsity.

Fig. 1: Two issues of AG-Sparse AllReduce.

I2. NIC-Centric Fine-Grained Sparsification Mechanism. To enable high parallelism while better exploiting intra-block sparsity, we propose the NIC-Centric Fine-Grained Sparsification Mechanism, implemented by a NIC-centric Sparsification Orchestration (NSO) engine in the SmartNIC. The NSO engine reads large batches of blocks from GPU memory and separates them into multiple aggregation streams for concurrent coordination. It connects to the ONC engines through low-latency on-chip queues, allowing blocks to be transmitted directly from the queues once coordination starts, without waiting for PCIe transfers. Compared to CPU-controlled sparsification, SmartReduce therefore supports low-latency, fine-grained sparse transmission while sustaining line-rate block throughput even with small block sizes.

We implement SmartReduce on programmable FPGAs and a programmable switch, and integrate it into PyTorch. Experiments on a real testbed show that SmartReduce at 99% sparsity achieves up to $6.4\times$ higher algorithm bandwidth than the state-of-the-art aggregator-server-based sparse AllReduce.

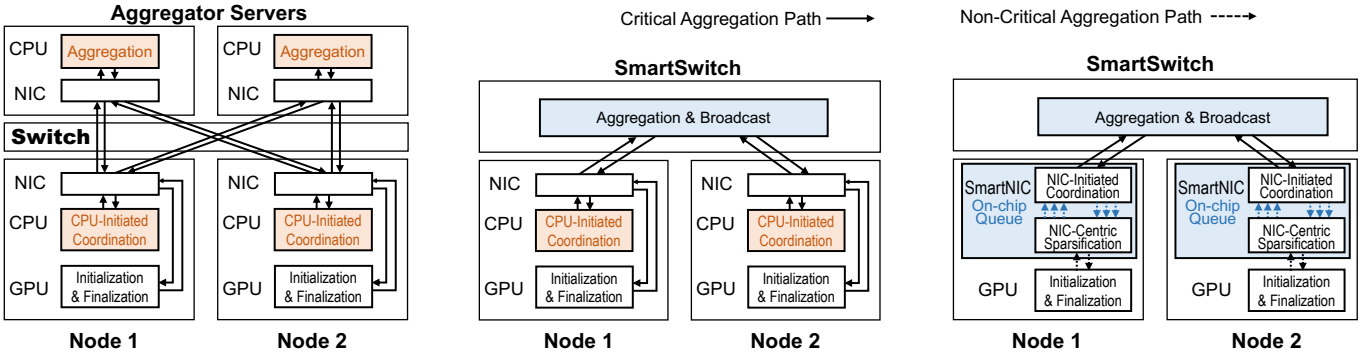
II. Background and Motivation

In this section, we introduce three kinds of sparse AllReduce: AllGather-based sparse (AG-Sparse, §II-A) AllReduce, Aggregator Server-based sparse (AS-Sparse, §II-B) AllReduce, and SmartNetwork-assisted sparse (SNA-Sparse, §II-C) AllReduce along with their issues, respectively.

A. AllGather-Based Sparse AllReduce

AllGather-based sparse (AG-Sparse) AllReduce [57]–[59] is the algorithm that aggregates sparse data through AllGather operations, which consists of three steps: (1) all nodes share their sparse indices and data using two AllGather operations, (2) each node performs a local Reduce operation on the gathered data, and (3) perform a coalescing to merge duplicate indices in a sparse tensor. However, the AG-Sparse AllReduce fails to achieve a high algbw due to two severe issues. To demonstrate this, we evaluate the sparse AllReduce implemented by Gloo [60] as an example.

P1. Poor Scalability. The algbw of AG-Sparse AllReduce drops as the number of nodes increases. Fig. 1a shows the algbw of Gloo's sparse AllReduce with a different number of nodes, where the input is 128 MB of data with sparsity range from 0% to 99%. The nodes are connected via 100 Gbps Ethernet. The algbw of Gloo's sparse AllReduce achieves at most 855 MB/s at the 2-node setting. As the number of



(a) AS-Sparse AllReduce. (b) SNA-Sparse AllReduce. (c) SMARTREDUCE.

Fig. 2: Comparison of different sparse AllReduce. The key idea is to remove the CPU from both the coordination path and the fine-grained sparsification path, thus (1) achieving stable sub-RTT coordination cost, (2) enabling massive in-flight coordinations, and (3) utilizing intra-block sparsity, to enable interference-free sparse AllReduce.

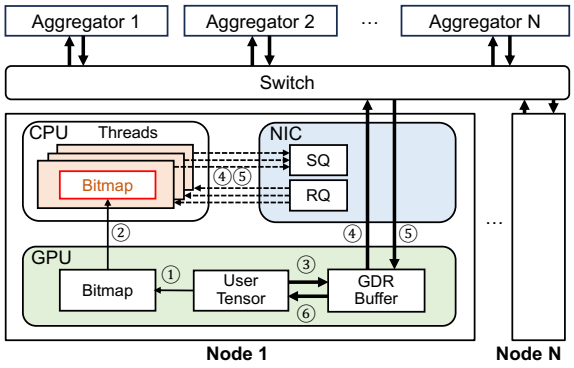
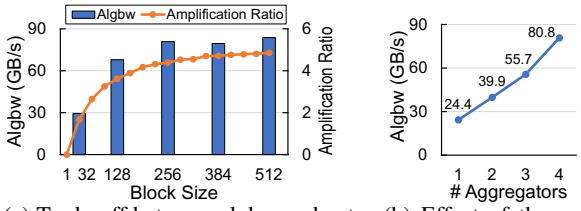


Fig. 3: Workflow of CPU-initiated coordination in OmniReduce.



(a) Trade-off between algbw and network amplification. (b) Effect of the number of aggregators.

Fig. 4: Two issues of AS-Sparse AllReduce.

nodes increases, algbw drops by 84% to 133 MB/s at the 8-node setting. This is because the communication volume of AllGather and the computation volume of local Reduce also scale as the number of nodes increases.

P2. Heavy Memory Pressure. The intensive memory access pressure limits the performance of AG-Sparse AllReduce. Fig. 1b shows the time breakdown of AG-Sparse AllReduce. In an 8-node configuration, the primary bottleneck is the local Reduce, which accounts for 80% of the duration of AG-Sparse AllReduce. The reason is that local Reduce traverses all local indices to find the corresponding position of the matching global index and accumulates blocks with the same index together, leading to irregular and intensive memory accesses that cause massive cache misses, and incurring long latency [67]. Finally, serial AllGather and local Reduce waste the network bandwidth and increase aggregation latency.

B. Aggregator Server-Based Sparse AllReduce

To solve the issues of AG-Sparse AllReduce, existing works propose aggregator server-based sparse (AS-Sparse) AllReduce [27], [29]. Fig. 2a shows that this method utilizes external servers as aggregators to achieve scalability (P1) and facilitate streaming aggregation (P2).

1) Workflow of AS-Sparse AllReduce: The major improvement of AS-Sparse AllReduce is the block-wise stream aggregation on sparse data. Fig. 3 illustrates the workflow of the state-of-the-art AS-Sparse AllReduce method OmniReduce [27] that consists of three stages:

- **Initialization Stage:** It first computes a bitmap indicating the positions of non-zero blocks that contain at least one non-zero element (①), then transfers the corresponding aggregation metadata to CPU memory (②), and finally copies the data to be aggregated into the GDR buffer (③).
- **Aggregation Stage:** The streaming aggregation is completed through multiple rounds of *coordination*. In each coordination round, first, all nodes vote to select a block to be aggregated by sending the indices of their local non-zero blocks to the corresponding aggregator (④); second, they send the voted block to an aggregator for aggregation and receive the aggregated block (⑤). We define *coordination cost* as the time taken for each coordination round to quantify the efficiency of aggregation. It is also worth mentioning that OmniReduce utilizes a parallelism method that allows each node to run multiple coordinations simultaneously through multiple independent threads.
- **Finalization Stage:** It writes the aggregated data back to the user tensor (⑥).

2) Issues of AS-Sparse AllReduce: We observe that OmniReduce can only achieve about 92.7 GB/s algbw when the sparsity is 99%. However, its actual goodput is only 1.4 GB/s, representing merely 10% of the line rate (100 Gbps network), due to two severe issues.

P3. Network Amplification from Using Large Blocks. To reduce the overhead of the host network stack in coordination, OmniReduce aggregates a large block in one coordination round. However, transmitting large blocks introduces signifi-

cant network traffic amplification, as redundant zero elements (padding zeros) must be included to maintain block alignment. Although block-wise indexing and transmission reduce the overhead of locating non-zero elements, they introduce residual zeros that cannot be entirely eliminated, limiting the exploitation of intra-block sparsity. To prove this, we count the number of padding zeros and non-zero elements that are transmitted by OmniReduce when training a DeepLight [68] model. Fig. 4a shows the algbw of AS-Sparse AllReduce and the ratio of the number of additional zero elements to non-zero elements (labeled as "Amplification Ratio") across various block sizes (i.e., 32, 128, 256, 384, and 512). We observe that OmniReduce reaches its peak algbw when the block size is 256. However, the number of padding zeros is $4.4\times$ of the number of non-zero elements, indicating that it transmits more than 80% of unnecessary data in the network, and thus fails to fully exploit data sparsity.

Using small blocks is an intuitive strategy to minimize unnecessary padding zeros. However, this approach introduces new challenges. Specifically, the transmission of small blocks increases the number of coordinations, thereby increasing the host network stack overhead. Furthermore, using small blocks introduces massive DMA reads and small-sized PCIe transactions that saturate PCIe transaction throughput. To address PCIe bandwidth limitations, a straightforward solution involves transferring data in small blocks using batched operations. For example, a batching approach can read the blocks needed by all threads at once and then segment them into individual packets to send within the NIC, which is similar to the TCP segmentation offload (TSO). However, the challenge is that the blocks to be accessed cannot be located in a contiguous memory area. This uncertainty comes from the varying speeds at which each thread consumes independent blocks, making it impossible to predict the next block.

P4. Requirement of Extra Aggregators to Avoid Incast. OmniReduce requires N aggregators to handle aggregation requests from N nodes, under the assumption that all servers, including workers and aggregators, have identical network bandwidth. A smaller number of aggregator servers leads to a degradation of algbw due to the incast issue. Fig. 4b illustrates the algbw of OmniReduce with 4 worker nodes, when the number of aggregators ranges from 1 to 4. We observe that when the number of aggregators is 1, algbw of each node is only 26.8% compared to the case with 4 aggregators.

C. SmartNetwork-Assisted Sparse AllReduce

To eliminate extra aggregators (P4), SmartNetwork-assisted sparse (SNA-Sparse) AllReduce [27], [28] is introduced to utilize a SmartSwitch, instead of aggregator servers, to perform in-network aggregation, as shown in Fig. 2b.

However, it also brings the challenge that only a small maximum block size (i.e., 64) can be used for block-wise aggregation, thus restricting the algbw. The underlying reason is that the number of stateful ALUs responsible for element aggregation in SmartSwitch limits the number of elements that can be processed at the same time. A maximum of 32 stateful ALUs (aggregating 64 elements) [61], [62] can be used for data block aggregation in addition to the number of stateful ALUs required to ensure the normal forwarding function and the reliability of sparse AllReduce.

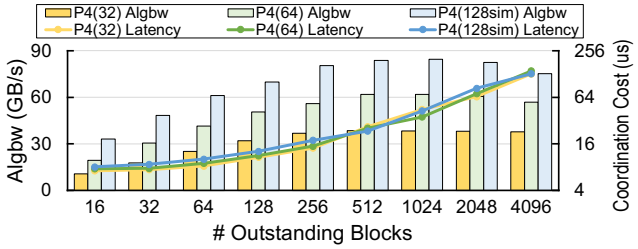


Fig. 5: Algbw of OmniReduce-P4 with different numbers of outstanding blocks. P4(32) and P4(64) are measured on Tofino-1, whereas P4(128sim) is obtained from simulation because one switch pipe supports at most 64 elements.

1) Issues of SNA-Sparse AllReduce: We evaluate the state-of-the-art SNA-Sparse AllReduce method OmniReduce-P4 [27] in a setup comprising four worker nodes and one switch aggregator. However, SNA-Sparse AllReduce only achieves 61.8 GB/s algbw when the block size is set to 64, due to high coordination cost.

P5. Increased Coordination Cost with More Concurrent Blocks. We measure algbw with block sizes of 32 and 64 on the Tofino-1 prototype and obtain the block-size-128 results from simulation because the hardware implementation supports at most 64 elements per block on one switch pipe. We vary the number of outstanding blocks from 16 to 4096. The total number of outstanding blocks is calculated by "the number of working threads $\times$ the number of outstanding blocks of each thread". In practice, we utilize 16 threads on each node and each thread allows a number of outstanding blocks from 1 to 256. We perform 100 iterations of AllReduce after 10 warm-up iterations, and we report the median algbw along with the CPU-initiated coordination cost. We also measure the average coordination cost, which is the time elapsed between two coordinations in each thread. Fig. 5 illustrates the algbw and the coordination cost under different configurations respectively. We have three major observations.

First, with a small number of outstanding blocks, an increasing number of outstanding blocks increases the achievable algbw. When the number of outstanding blocks increases from 16 to 512, OmniReduce-P4 significantly increases the achievable algbw, while the average CPU-initiated coordination cost only increases slightly within a range of 10-20 us. Second, the increased coordination cost restricts the peak algbw. When the number of outstanding blocks increases from 512 to 1024, OmniReduce-P4 achieves the peak algbw of 38.3 GB/s and 61.8 GB/s with measured block sizes of 32 and 64, respectively, while the simulated block-size-128 configuration reaches 84.4 GB/s. Meanwhile, in the case of 1024 outstanding blocks, the CPU-initiated coordination cost increases to 43 us, nearly doubling when compared to the setup with 512 outstanding blocks. This is because such a large number of outstanding blocks allowed by AS-Sparse

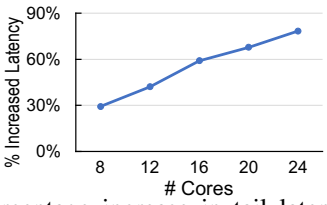


Fig. 6: The percentage increase in tail latency when aggregating imbalanced distributed blocks compared to balanced distributed blocks under different numbers of cores.

AllReduce occupies a huge working set size and thus leads to severe cache misses, and TLB refresh overheads [69], [70]. As a result, resource competition between threads makes it longer for a CPU thread to process each coordination, and increasing the number of outstanding blocks no longer increases the achievable algbw. Third, the congestion of a large number of outstanding blocks leads to the degraded algbw. When the number of outstanding blocks increases further to 4096, algbw drops by up to 14% in the 128 block size setup, and the CPU-initiated coordination cost goes up to 130 us.

P6. Severe Load Imbalance between CPU Cores. Each CPU core is responsible for coordinating a certain number of blocks with a fixed mapping. The CPU core that processes the most number of coordination rounds determines the communication latency. However, the imbalanced number of coordinations between CPU cores leads to severe tail latency. To illustrate this, we generate both randomly and evenly distributed data as input and measure the time each CPU core takes to aggregate all its blocks. Fig. 6 shows the percentage increase in tail latency when the sparsity is 99%. Aggregation on randomly generated data leads to a 79% increase in the completion time of the last core, relative to evenly distributed data, highlighting a significant load imbalance. This imbalance results in idle cores and under-utilization of network bandwidth, as faster cores must wait for the slowest one to complete. So, it is difficult to increase algbw merely by increasing the number of outstanding blocks, even with a high-performance CPU that has more cores or a GPU to execute sparse AllReduce due to the load imbalance issue.

III. Design of SmartReduce

Given the limitations of existing approaches, we keep two goals in mind when designing SmartReduce to achieve promising algbw of sparse AllReduce:

G1. Minimizing the Coordination Cost under Massive In-Flight Coordinations. SmartReduce must achieve as low coordination cost as possible to achieve line-rate aggregation throughput with massive outstanding blocks to solve **P1**, **P2**, **P4**, **P5**, and **P6**.

G2. Reducing the Amplification Ratio to Better Leverage Data Sparsity. SmartReduce must maintain high throughput when transmitting small blocks to achieve a better trade-off between host network stack overhead and network traffic amplification to address **P3**.

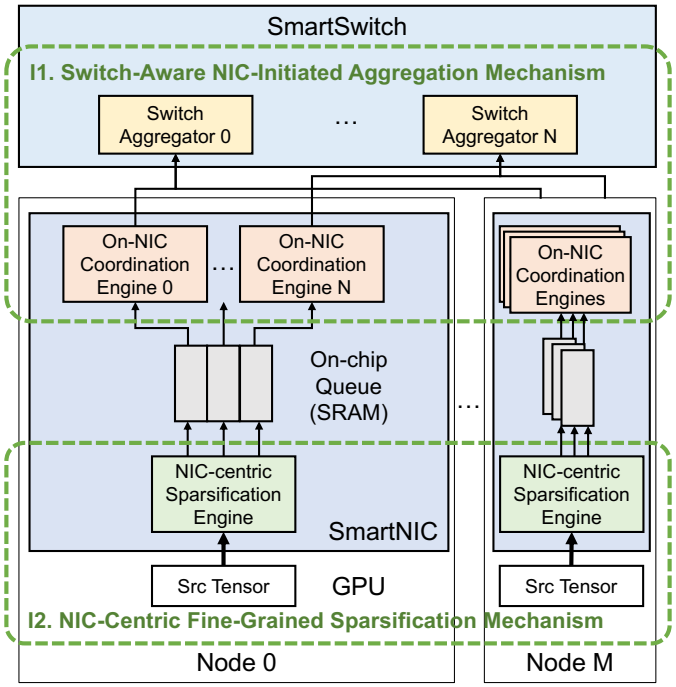


Fig. 7: Overview of SmartReduce.

A. Naive SmartNIC-offloaded Approach

Naively offloading the sparse AllReduce control logic to the SmartNIC seems to be a viable approach to reduce coordination cost by avoiding CPU overhead. Existing works accelerate applications by SmartNIC [71]–[81] which allows SmartNIC to start a block transmission from GPU just like a CPU-initiated RDMA operation. However, this naive method suffers from a high coordination cost. Specifically, the coordination cost of about 4∼6 us in the best case [82] even only considering the critical path. Because the naive offloading method inherits a long critical path from SNA-Sparse AllReduce, which involves initiating coordination from the SmartNIC, transferring data from the GPU memory to the SmartNIC, packaging and sending within the SmartNIC, and aggregating in the aggregators. Specifically, coordinating a 128-byte block takes at least 2 microseconds on a PCIe transaction [69] that includes worker-side and aggregator-side, accounting for 33% of the coordination cost. Even worse, the latency is vulnerable to increase due to PCIe bandwidth contention from massive data transfer between GPU and NIC, and GPU memory bandwidth contention from matrix multiplication of model training. Therefore, such a naive approach could still suffer from increased coordination cost for such a latency-sensitive sparse AllReduce.

B. Overview of SmartReduce

To achieve the above goals, we propose SmartReduce, a SmartSwitch-aware, SmartNIC-centric sparse AllReduce, as illustrated in Fig. 2c. The key idea is to sustain high-throughput sparse AllReduce by removing the CPU from both the coordination path and the fine-grained sparsification path. To achieve our design goals, we have two key mechanisms, as shown in Fig. 7.

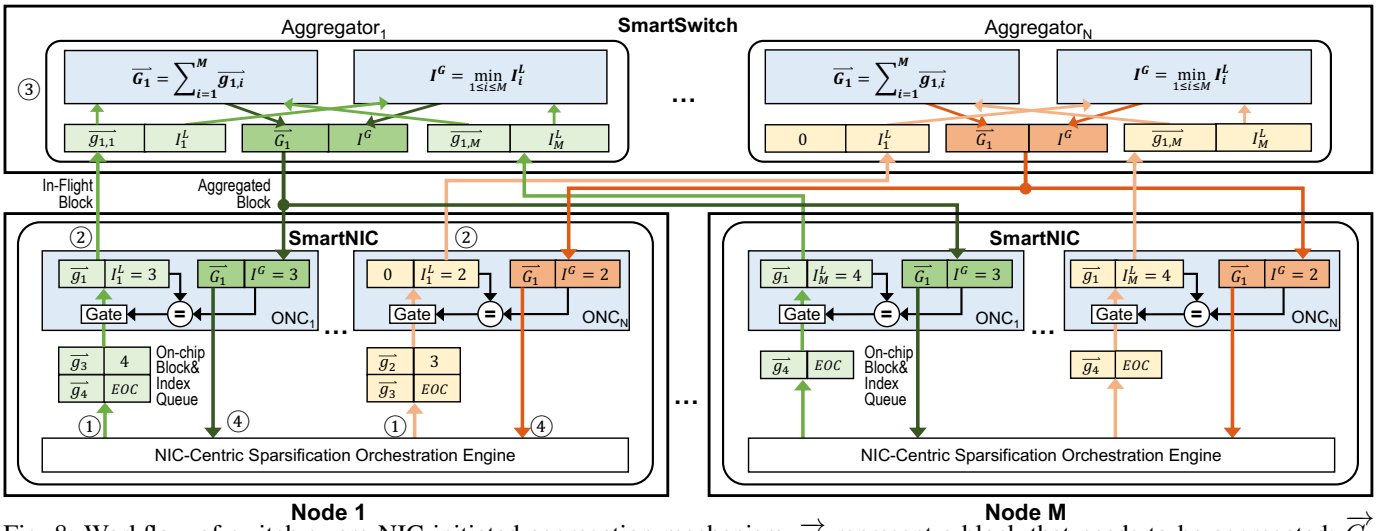


Fig. 8: Workflow of switch-sware NIC-initiated aggregation mechanism. $\overrightarrow{g_i}$ represent a block that needs to be aggregated; $\overrightarrow{G_i}$ represent a aggregated block; I^L indicates the index of local next block; I^G indicates the index of global next block.

I1. Switch-Aware NIC-Initiated Aggregation Mechanism. To achieve **G1**, we realize the Switch-Aware NIC-Initiated Aggregation Mechanism with On-NIC Coordination (ONC) engines in the SmartNIC and matching block aggregators in the SmartSwitch. The ONC engines consume sparse block streams from low-latency on-chip queues and initiate coordination directly with their matching switch aggregators that enables the shortest critical path of coordination. With the collaboration of SmartNIC and SmartSwitch, SmartReduce can perform massive concurrent coordinations with stable sub-RTT coordination cost to achieve line-rate sparse AllReduce while being highly resistant to load imbalance.

I2. NIC-Centric Fine-Grained Sparsification Mechanism. To achieve **G2**, we realize the NIC-Centric Fine-Grained Sparsification Mechanism with a NIC-centric Sparsification Orchestration (NSO) engine, which organizes sparse data into concurrent aggregation streams and places fine-grained blocks into low-latency on-NIC queues. Specifically, during the sparse AllReduce process, the NSO engine continuously monitors the progress of all aggregation streams and proactively prefetches upcoming data blocks from the GPU for aggregation. Therefore, SmartReduce can achieve line-rate sparse aggregation throughput while exploiting intra-block sparsity.

In the following, we introduce the detailed designs of the two key mechanisms.

C. Switch-Aware NIC-Initiated Aggregation Mechanism

SmartReduce realizes block-wise streaming aggregation through a switch-aware, NIC-initiated coordination path. In each round, parallel ONCs send local blocks and their next-block indices directly from the SmartNIC to matching switch aggregators. Each aggregator computes the global minimum index and aggregated block, then returns them to the ONCs to drive the next round. Because the ONCs consume prefetched blocks and indices from on-chip queues and receive the switch results directly, each coordination round bypasses CPU packet processing and avoids waiting for PCIe data reads, thereby removing the CPU from the coordination critical path. The Switch-Aware NIC-Initiated Aggregation Mechanism comprises ONC engines in the SmartNIC and matching block aggregators in the SmartSwitch, as illustrated in Fig. 8.

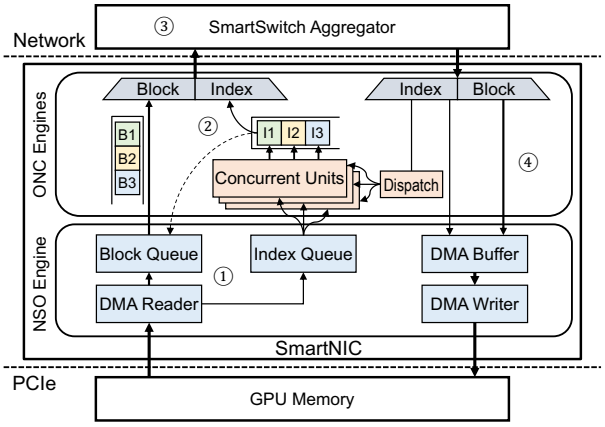
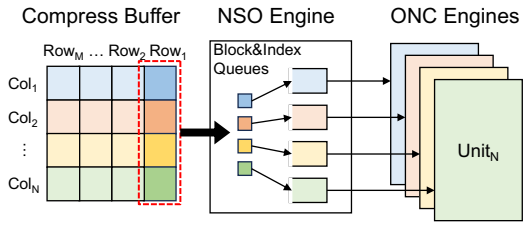


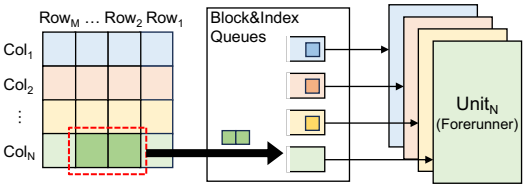
Fig. 9: Hardware Engines in SmartNIC.

1) On-NIC Coordination Engine: Fig. 9 shows the detailed design of SmartNIC that comprises several modules. When aggregation operations commence, the SmartNIC initially employs DMA by NSO engine to prefetch blocks and indices from GPU memory, which are stored in dedicated on-chip queue (①). Then the "Unit" initiates coordination according to the index in the queue (②). Indices from multiple cores are managed through a send FIFO. The "Block Queue" can read the FIFO signal and pop the corresponding blocks. The block and the index are transmitted to the switch aggregator in a packet (③). After aggregation, SmartNIC receives the packet containing the global index and the aggregated block. The global index is dispatched to the appropriate unit to control the next coordination. Both index and block are stored in the "DMA Buffer" and wait for writing to the GPU memory (④).

The SmartNIC can fully utilize network bandwidth through the parallelism of concurrent units in the ONC engine, which addresses load imbalance issue due to its hardware context



(a) Phase 1. Read a row of blocks to allow all units in the ONC engine to perform coordinations simultaneously.



(b) Phase 2. The NSO engine polls the number of blocks in each queue and replenishes blocks to the queues with insufficient blocks. Forerunner refers to the engine that consumes blocks at a higher speed.

Fig. 10: Two-phase prefetching of NIC-centric fine-grained sparsification mechanism.

switching ability. When only one unit is deployed, the long aggregation step and the dependency between packets sent by a single engine leave substantial idle time during aggregation. When multiple units are deployed, parallel units can utilize these gaps to ensure stream data processing that maximizes network bandwidth utilization.

2) Switch Aggregator: SmartSwitch is designed to provide low-latency and high-throughput in-network aggregation service for parallel ONC engine. As shown in Fig. 8, the number of independent aggregators matches the number of units of ONC engine in SmartNIC, so each unit can execute block aggregation without interfering with each other. The aggregation request from engines contains the local block of gradients ($\vec{g}$) and the local next block index (I^L). The aggregator adds $\vec{g}$ to the aggregate block $\vec{G}$ and updates the global index I^G to the minimum of I^G and I^L. Each aggregator has a counter that records the number of packets received from distinct workers. Retransmissions and incomplete rounds are handled by the reliability protocol described in Section V. After receiving blocks from all nodes, SmartSwitch broadcasts $\langle \vec{G}, I^G \rangle$ to all nodes.

D. NIC-Centric Fine-Grained Sparsification Mechanism

The NIC-Centric Fine-Grained Sparsification Mechanism decouples PCIe transfer granularity from sparse aggregation granularity. SmartReduce implements this mechanism with the NSO engine, which fetches batches of blocks from GPU memory and stages fine-grained blocks and indices in per-ONC on-chip queues. Consequently, ONCs transmit fine-grained blocks directly from on-chip queues instead of waiting for per-block PCIe reads, keeping data preparation off the transmission critical path while reducing padding zeros.

Batch prefetching must balance pipeline startup latency and steady-state transfer throughput, motivating two complementary design choices.

- *Latency-centric batching* approach that reads the data block needed by all control engines at once and then distributes it to each engine, which helps eliminate idle time at the head and tail of the pipeline.
- *Throughput-centric batching* approach that reads the blocks needed by one control engine at once, with different engines taking turns to read via polling, which can achieve high throughput without deadlock between ONC engines.

To achieve both advantages of two batching approaches, our approach adopts a two-phase block batching mechanism consisting of fast-startup and steady-polling, as shown in Fig. 10.

During the fast-startup phase, we aim to have each engine be able to initiate its first block coordination quickly, so we apply a latency-centric batching mechanism. First, we transmit all the indices to the on-chip SRAMs, which are the key information to control the aggregation, because the volume of the indices is relatively small compared to the block. Second, we transmit the blocks that each engine will send for the initial few coordinations to the SmartNIC, so that engines can issue the coordination immediately. However, continuing to use the latency-centric approach can lead to deadlock. Because when some forerunner units exhaust their block buffers and require a fresh set of blocks while other straggler units still have full buffers, there is a risk of deadlock between the forerunner engines and the straggler engines. So, in the steady-polling phase, we apply a throughput-centric batching mechanism that aims to prepare blocks to aggregate for engines without any idle time. To achieve this, the block loader scans the buffer of each unit periodically. When the number of blocks in a buffer falls below a certain threshold, a batch of blocks is loaded into the buffer. Thus, we can achieve both low-latency startup and line-rate block prefetching.

IV. Implementation

SmartNIC. Our Alveo U280 prototype implements 128 ONCs and one NSO engine at 250 MHz. Each ONC is an independent state machine that processes one block at a time and maps one-to-one to a switch aggregator. The NSO engine interfaces with the DMA engine and extracts block indices and values through hardware pipelines before placing them in per-ONC on-chip queues. During steady polling, NSO triggers a 4 KB refill when a queue contains less than 8 KB of data. The prototype supports blocks of up to 128 elements, and we select 32 elements for the evaluated configuration. The SmartNIC modules comprise approximately 2,000 lines of Chisel, which are translated to SystemVerilog and synthesized with Vivado into the bitstream deployed on each FPGA.

SmartSwitch. Our Tofino-1 prototype instantiates 128 aggregators on one switch pipe, matching the 128 ONCs on each SmartNIC. Each aggregator maintains independent aggregate, minimum-index, counter, bitmap, and shadow-copy state, and uses integer addition, minimum selection, registers, counters, and multicast in the match-action pipeline. As reported in Table I, the program occupies 83.3% of the stages, 70.8% of the registers, and 14.2% of the SRAM in one pipe. Deploying

the same program on additional pipes incurs similar per-pipe utilization.

Numerical Format. We adopt SwitchML's numerical representation [61], which combines 32-bit fixed-point addition in the switch with adaptive scaling at the workers. Each packet uses a scaling factor derived from the maximum absolute value of its gradient block and the number of workers, reserving sufficient range for the aggregated sum. Workers quantize gradients before transmission and convert the aggregated result back to floating point, while weights and activations remain in floating point. The fine-grained sparsification in SMARTREDUCE only omits zero values. Changing the block size changes the number of transmitted padding zeros but does not alter non-zero values, so quantization is the only additional numerical approximation on the aggregation path.

V. DISCUSSION

In this section, we discuss some design choices outside of our key innovations.

Why Not SoC-Based SmartNIC? Currently, there are two common types of SmartNICs: SoC-based SmartNIC (i.e. BlueField-2/-3 [83]) and FPGA-based SmartNIC. We choose the FPGA-based SmartNICs for the flexible and line-rate data processing ability required by SMARTREDUCE. In contrast, the SoC-based SmartNIC fails to achieve line-rate packet processing due to the low memory bandwidth and high processing latency of the on-chip ARM CPU [66], [84], [85].

Specifically, the NSO engine of SMARTREDUCE loads the data blocks into the memory of SmartNIC and then transmits them to the ONC engine to perform aggregation. However, existing SoC-based SmartNICs need to store packets in off-chip DRAM and perform sparse aggregation logic on the ARM CPU. The limited processing ability and memory bandwidth of ARM CPUs limit the line-rate processing of sparse data blocks. For example, the state-of-the-art SoC-based SmartNIC BlueField-3 equips a 16-core ARM CPU running on 2133 MHz frequency, which makes it hard to support line-rate processing of small packets, as we discuss in Section II-C. Besides, an off-the-shelf BlueField-2 SmartNIC requires 400 Gbps memory bandwidth to serve 100 Gbps packets, while it only provides 204.8 Gbps memory bandwidth. This memory bandwidth constraint still holds for the newer-generation BlueField-3. A BlueField-3 requires 1600 Gbps memory bandwidth to achieve 400 Gbps line-rate, while it only provides 716.8 Gbps memory bandwidth [86]. In contrast, FPGA-based SmartNICs allow processing packets in hardware pipelines and thus allow the two key modules of SmartReduce to interconnect via queues implemented in on-chip SRAMs with extremely low latency, thereby achieving line-rate, high-concurrency aggregation of small data blocks.

Portability. The switch implementation of SMARTREDUCE requires integer addition, minimum selection, counters, registers, multicast, and sufficient stateful pipeline capacity. These primitives enable SMARTREDUCE to be functionally mapped to PISA-class switches, with line-rate aggregation determined by the available pipeline and state capacity.

On the NIC, SMARTREDUCE requires GPU-memory DMA, a programmable packet and data pipeline, on-chip queues, and enough independent state machines to cover the coordination latency. These functions can also run on a SoC-based SmartNIC, but general-purpose cores may not sustain the same small-packet concurrency. With similar coordination logic implemented on a BlueField-3, we measure coordination costs of 13 us with one outstanding block and 53 us with 32 outstanding blocks. From these measurements, we estimate an aggregation algorithm bandwidth of 5.46 GB/s, approximately 2% of the FPGA-based SMARTREDUCE prototype. Thus, the protocol is functionally portable, while matching the evaluated performance requires hardware that can sustain its fine-grained parallel datapath.

Scaling Beyond One Switch. Existing training clusters can scale to thousands of GPUs, interconnected via a 400 Gbps network and NVLink with a PCI×NVLink (PXN) topology [87], [88], where each GPU is equipped with a unique SmartNIC and GPUs with the same index across different nodes are connected to the same SmartSwitch. To accommodate such scenarios, we propose a hierarchical aggregation mechanism. Compared to the single-GPU scenario, the hierarchical SMARTREDUCE has two additional steps, which are intra-node Reduce-Scatter and intra-node AllGather. Specifically, each GPU first performs Reduce-Scatter within a node via NVLink to get a shard of local aggregation data. Then, GPUs with the same index aggregate their data on a single leaf SmartSwitch with SMARTREDUCE. Finally, each GPU gets the global aggregation result via an intra-node AllGather.

As the number of GPUs increases [89]–[91], the network topology has more than one layer. We arrange multiple switches in a tree, and each leaf switch sends the aggregation result to the parent switch after completing local aggregation. We estimate the latency of a two-level tree based on the measured single-switch prototype. Our prototype measures an approximately 1.5 us NIC–switch–NIC round-trip time through one switch level. A two-level tree adds a second switch level to the coordination path, so we estimate that the round-trip time would approximately double to 3 us. Under the prototype's 100 Gbps configuration, the in-flight coordination requirement scales with this latency. We therefore estimate that 256 ONC units, twice the current 128, are needed to cover the 3 us round trip. Scaling to 256 units only replicates the concurrent ONC units shown in Fig. 9. A linear estimate from Table I places their design at approximately 28% of FPGA LUTs. The switch-side aggregators share the same processing pipeline, so increasing their number primarily expands per-aggregator register state rather than adding pipeline stages. Therefore, the current 83.3% stage utilization does not grow proportionally with the aggregator count. These estimates show that the required SmartNIC concurrency fits comfortably within the current FPGA LUT capacity.

Multi-Job State Isolation. We consider a multi-job cluster in which each training job occupies a disjoint set of GPU workers. Workers configure a disjoint aggregator-ID range for each job through prior assignment. The aggregator ID carried

by each packet selects independent aggregate-buffer, counter, bitmap, and shadow-copy state in the switch, preventing state from different jobs from interfering. The switch chip provides sufficient processing throughput to aggregate multiple jobs at line rate simultaneously.

Reliability. We use timeout-based retransmission while preserving the invariant that each worker contributes at most once to each aggregation window. After sending a request, each ONC starts a timer and retransmits upon timeout, up to a configured limit of seven retries. For an incomplete aggregation, the switch indexes an aggregation bitmap by worker ID and accumulates a retransmitted request only if that worker's bit is unset. If aggregation has already completed, the switch instead retransmits the retained result from the aggregator's shadow copy [61]. The next aggregation request piggybacks an acknowledgment, and the switch clears and reuses the counter, bitmap, aggregate, and shadow state only after receiving acknowledgments from all workers. A sliding window prevents an ONC from sending requests beyond the current aggregation window, so delayed or straggling workers cannot advance the associated aggregator state prematurely. Packet loss can therefore increase completion latency but does not duplicate a worker's contribution. Exhausting the retry limit is treated as a collective failure. The current protocol does not recover membership after a permanent worker or job failure.

Supporting More Collectives. SMARTREDUCE can be generalized to support popular collective communication primitives, such as AllGather and ReduceScatter, which are adopted by LLM training. To support AllGather, SMARTREDUCE only needs the SmartSwitch to serve as a Broadcast engine, reducing per-node transmission by enabling a centralized broadcast proxy. To support ReduceScatter, SMARTREDUCE can first perform the in-switch Reduce operation just like AllReduce, and second directly unicast the partitioned results to the corresponding node. Importantly, SMARTREDUCE naturally supports concurrent execution of a AllGather and a ReduceScatter due to its hardware parallelism, compared to serial collective execution from NCCL. We leave it to future work.

VI. EVALUATION

In this section, we present an end-to-end sparse AllReduce performance comparison (§VI-B), followed by effect of two innovations **I1** (§VI-C) & **I2** (§VI-D). Finally, we show performance of SMARTREDUCE in a real training scenario (§VI-E).

A. Experimental Setup

Testbeds. Our testbed consists of four nodes, each with two Intel Xeon Silver 4214 CPUs, two NVIDIA A100 GPUs connected by 300 GB/s NVLink, and one Alveo U280 FPGA-based SmartNIC. Table I reports the resource utilization of our SmartNIC and SmartSwitch implementations. Each SmartNIC connects to a 100 Gbps front-panel port on a Tofino 1 Wedge100BF-32X SmartSwitch, and all ports used in our experiments map to the same Tofino pipe. Because Tofino does not support floating-point arithmetic, we quantize values to integers using SwitchML's method [61]. For the baselines, each node uses a Mellanox ConnectX-5 100 Gbps NIC.

TABLE I: Resource utilization of SmartNIC and SmartSwitch.

SmartNIC				SmartSwitch		
LUT	**FF**	**BRAM**	**URAM**	**Stage**	**Reg**	**SRAM**
13.91%	9.79%	19.87%	8.33%	83.3%	70.8%	14.2%

Baselines. We compare SMARTREDUCE against two dense and three sparse AllReduce baselines under the same input configurations. Two baselines implement dense AllReduce. (1) *NCCL* performs ring AllReduce on the experimental nodes with GPUDirect RDMA enabled. (2) *SwitchML* [61] performs in-network aggregation on the same Tofino-1 switch. The remaining three implement sparse AllReduce. (3) *Gloo* [60] performs sparse AllGather over ConnectX-5 NICs followed by local reduction, as evaluated in Fig. 1a. (4) *OmniReduce* uses four additional CPU servers, each with one Intel Xeon Silver 4214 CPU, as parameter servers. We follow its recommended configuration [27], using NCCL for intra-node reduce and broadcast and OmniReduce for inter-node AllReduce. (5) *OmniReduce-P4* uses a Tofino 1 Wedge100BF-32X SmartSwitch as the aggregator with a block size of 64.

SMARTREDUCE Parameters. We set the block size to 32 and use 128 ONC units. In NSO, the fast-startup phase prefetches one block per unit, totaling 16 KB for 128 units. During the steady-polling phase, NSO scans all 128 queues every 640 ns and issues a 4 KB DMA transaction for a queue requiring refill. We use 4 KB because it is the smallest transaction size that sustains high PCIe throughput on our platform.

Workloads. For the aggregation benchmarks, we use randomly generated sparse inputs following OmniReduce's data-generation method [27]. We vary two parameters: data size, which is 128 MB unless otherwise specified, and data sparsity, which is set to 0%, 60%, 90%, and 99%.

B. AllReduce Performance Comparison

To show the efficiency of SMARTREDUCE, we compare the algbw of SMARTREDUCE and the baselines with different numbers of nodes and GPUs. We adopt four configurations: (2,1), (4,1), (2,2), (4,2), where (x, y) indicates AllReduce among x nodes, each of which has y GPUs. The single-GPU configuration uses a three-stage pipeline comprising initialization, inter-node SMARTREDUCE, and finalization. The multi-GPU configuration adds intra-node reduce and broadcast, forming a five-stage pipeline.

Fig. 11 compares the algbw of SMARTREDUCE and four baselines across device configurations and sparsity levels. Two trends emerge. First, at 60% sparsity and above, SMARTREDUCE consistently achieves the highest algbw among the systems in the figure. At 99% sparsity, it outperforms OmniReduce by 3.8×–6.4×, NCCL by 15.3×–39.7×, and SwitchML by 19.6×–33.4× across the four configurations. Using the Gloo result in Fig. 1a, the two-node comparison at 99% sparsity gives SMARTREDUCE a 360× advantage. These gains arise from different baseline bottlenecks. NCCL and SwitchML

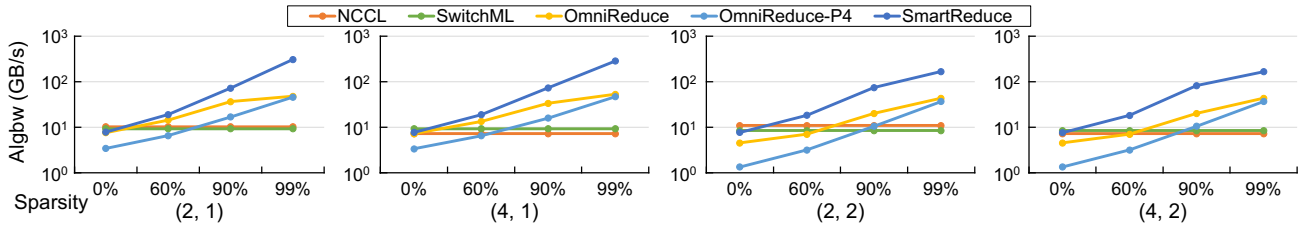
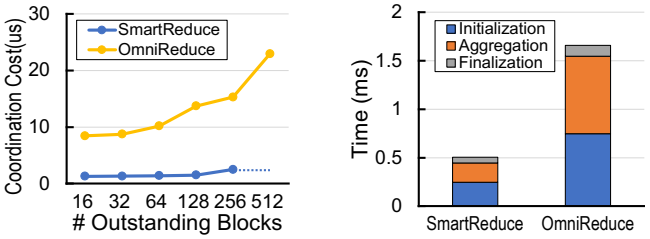


Fig. 11: Comparison of four baselines and SMARTREDUCE. (x, y) means the case with x node(s), each with y GPU(s).



(a) The cost with different # outstanding blocks.

(b) Breakdown of two systems.

Fig. 12: Comparison between SMARTREDUCE and OmniReduce with 99% sparsity under the (4,1) configuration.

process every tensor element, Gloo incurs memory-intensive local reduction after sparse AllGather, and OmniReduce relies on CPU-initiated block coordination. SMARTREDUCE instead uses the Switch-Aware NIC-Initiated Aggregation Mechanism, implemented by ONC engines and matching switch aggregators, to remove CPU coordination from the critical path. It uses the NIC-Centric Fine-Grained Sparsification Mechanism, implemented by the NSO engine, to transmit only non-zero segments. Second, at 0% sparsity, SMARTREDUCE provides no consistent advantage over dense collectives. Across the four configurations, it achieves 70%–107% of NCCL's algbw and 83%–91% of SwitchML's. Without zeros to eliminate, block metadata and initialization and finalization overhead remain, so SMARTREDUCE targets workloads whose sparsity amortizes these costs.

C. Effect of Switch-Aware NIC-Initiated Aggregation

To show the effect of **I1**, we break down the coordination cost of existing systems. Fig. 12a presents the coordination cost of OmniReduce and SMARTREDUCE under the (4,1) configuration with a fixed sparsity of 99%, across varying numbers of outstanding blocks. Due to hardware resource constraints on the FPGA, SMARTREDUCE cannot support 512 outstanding blocks. When the number of outstanding blocks is below 256, SMARTREDUCE consistently incurs lower coordination cost than OmniReduce. Moreover, while OmniReduce exhibits an increasing coordination overhead as the number of outstanding blocks grows, SMARTREDUCE maintains a stable coordination cost around 1.5 us. Only at 256 outstanding blocks does SMARTREDUCE's cost slightly increase to 2.5 us, primarily because 128 outstanding blocks are sufficient to saturate the network bandwidth, and exceeding this limit causes contention among parallel units. These results highlight the efficiency of the Switch-Aware NIC-Initiated Aggregation Mechanism, implemented by the ONC engines and matching switch aggregators, in sustaining stable sub-RTT coordination latency under high parallelism.

TABLE II: Network amplification analysis under peak algbw.

System	% Zeros	% Non-zeros	Amp. Ratio
NCCL	99.7%	0.3%	332
OmniReduce	81.46%	18.54%	4.39
OmniReduce-P4	72.61%	27.39%	2.65
SMARTREDUCE	62.64%	37.36%	1.67

Execution Time Breakdown. Fig. 12b shows the execution time breakdown of SMARTREDUCE and OmniReduce. First, SMARTREDUCE reduces the sparse AllReduce time by 75%, showing that the proposed methods can significantly reduce the aggregation time. Second, since SMARTREDUCE reads the sparse index directly from the GPU through SmartNIC without additional memory copy, the initialization time of SMARTREDUCE is 60% lower than that of OmniReduce.

D. Effect of NIC-Centric Fine-Grained Sparsification

To validate that the NSO (**I2**) can relieve the issue of sending too many zeros (**P3**), we count the number of padding zeros and non-zero elements in gradients when training a DeepLight [68] model. Table II quantifies the percentage of zero and non-zero elements in the transmitted gradients and the corresponding amplification ratio (Amp. Ratio), when each system achieves its peak throughput. A higher amplification ratio indicates that a system transmits a larger proportion of redundant zeros, leading to inefficient bandwidth usage. Since the gradients exhibit over 99% sparsity, NCCL sends a large number of padding zeros. OmniReduce transmits 81.46% zeros, resulting in an amplification ratio of 4.39, while OmniReduce-P4 sends 72.61% zeros, with a reduced but still significant amplification of 2.65. In contrast, SMARTREDUCE achieves a much lower amplification ratio of 1.67, which reduces the transmission volume by 50% compared to OmniReduce. This is because the NSO mechanism ensures that communication reduction does not come at the cost of achievable algbw, maintaining line-rate packet transmission even under aggressive sparsification.

Hardware and Design Benefits. To isolate the benefit of FPGA-based coordination, we simulate *OmniReduce+FPGA* by retaining OmniReduce's block size and resulting network traffic while replacing its CPU coordination latency with the measured FPGA ONC latency. Under the same 128 MB input and 99% sparsity used above, OmniReduce+FPGA reaches 96 GB/s, 2.0× the measured CPU-based OmniReduce. This result isolates the benefit of moving coordination to the FPGA. SMARTREDUCE further reaches 3.2× the algbw

TABLE III: Attributes of testing models.

Model	Model Size	Bounded by	Sparsity
LSTM [92]	1.52 GB	Communication	95%
DeepLight [68]	2.26 GB	Communication	99%
BERT [93]	1.2 GB	Computation	10%

of OmniReduce+FPGA because NSO enables finer-grained blocks that reduce padding and network amplification. The OmniReduce+FPGA result is a simulated estimate rather than a hardware measurement.

E. Sparse AllReduce in Model Training

We train three models with different characteristics using SmartReduce and the baselines. Table III shows the models we use and their attributes, where the sparsity field shows the gradient sparsity measured after 50 epochs of training.

1) Training Performance: To show the benefits of SmartReduce in end-to-end training, we measure the average iteration time of the first 100 iterations on SmartReduce and the baselines when training the three models. Fig. 13 shows the speedup in average iteration time of SmartReduce compared to OmniReduce with the (4,2) configuration. We modified PyTorch's bucket implementation so that SmartReduce can process aggregation requests with a micro-batch size of 128 MB and allow backpropagation computation to overlap with AllReduce. We make three major observations.

First, SmartReduce outperforms OmniReduce by $1.16\times\sim 1.36\times$, primarily due to two key factors: (1) SmartReduce achieves higher algorithm bandwidth for sparse AllReduce due to a much lower coordination cost implemented by SmartNIC and SmartSwitch, while OmniReduce's CPU-centric coordination incurs heavy overhead. (2) SmartReduce overlaps the intra- and inter-node communication, while OmniReduce serially executes the hierarchical aggregation. Second, SmartReduce achieves a more significant speedup when training sparse models (LSTM and DeepLight) than dense models (BERT). This is because SmartReduce leverages the model sparsity by gradient compression to reduce the communication time, making the communication easier to overlap with GPU computation. Third, SmartReduce achieves a higher speedup on LSTM ($1.36\times$) than DeepLight ($1.21\times$), although DeepLight has higher sparsity. This is because the high sparsity gives DeepLight a relatively short communication time to optimize after OmniReduce's compression. In particular, communication costs $21.1\times$ time compared to computation during backpropagation on LSTM, but only $5.9\times$ on DeepLight. Therefore, DeepLight with a higher sparsity rather has a smaller optimization space.

2) Convergence: To evaluate convergence under the numerical representation described in Section IV, we fine-tune three models using SmartReduce and NCCL. We fine-tune LSTM with the Google One Billion Word Dataset [94], DeepLight with Criteo 1TB Click Prediction Dataset[1], and BERT with the Stanford Question Answering Dataset [95]. Fig. 14 shows the training loss trend during fine-tuning. Across the evaluated fine-tuning runs, the two systems follow similar loss trajectories and reach similar final loss values.

[1]https://ailab.criteo.com/download-criteo-1tb-click-logs-dataset/

VII. Related Work

To our knowledge, SmartReduce is the first sparse AllReduce design to realize block-wise streaming aggregation with NIC-initiated coordination on a co-designed SmartNIC–SmartSwitch hardware pipeline.

Topology-Aware Communication. Existing work optimizes collective communication in distributed systems by sensing device topology, specifically targeting heterogeneous network topologies in modern clusters. In the multi-node multi-GPU scenario, the characteristics of device heterogeneity have been widely studied [96]–[100]. For complex device topologies, various communication schemes [101], [102] are proposed. Compared to existing HA methods, SmartReduce focuses on optimizing for a common and specific topology, where GPUs within a server are connected via NVLink and servers are connected via 100 Gbps network.

In-Network Aggregation. Recent works [62], [103]–[109] leverage dedicated hardware such as programmable switches [110] and dedicated on-chip FPUs [111], [112]) to reduce the network data volume during DDP. Dense in-network aggregation systems such as SwitchML [61] and SHARP [111] avoid sparse coordination but aggregate every tensor element, favoring low-sparsity inputs. In contrast, SmartReduce co-designs SmartNICs and SmartSwitches to perform in-network aggregation while exploiting sparsity. OmNICCL [28] places sparse aggregation on SmartNIC SoC cores and relies on DCA-backed LLC buffers, whereas SmartReduce initiates coordination in FPGA SmartNIC pipelines and performs aggregation in SmartSwitch pipelines using explicit on-chip state.

Gradient Compression. Several works try to solve the communication bottleneck in DDP by applying GC in AllReduce and parameter server. Optimization of GC algorithms [31], [113]–[115] improves GC communication in DDP by introducing more effective compression methods to keep accuracy, faster compression algorithms to reduce compression time, and effective sparse formats to reduce communication volume. SmartReduce is orthogonal to these GC algorithm optimizations because our approach is not limited to any specific GC algorithm and can also benefit from these studies. Compression strategies [37], [116], [117] determine the optimal way to apply GC by analyzing workload and resource usage in DDP. SmartReduce focuses on providing efficient AllReduce primitives that can be integrated with existing scheduling methods for further optimization. Considering that the GC algorithm does not integrate well with existing deep learning frameworks, some training frameworks and libraries with sparse communication [59], [118]–[120] are proposed to make it easier for users to employ GC methods. SmartReduce have already compared with the state-of-the-art GC method OmniReduce, our approach offloads GC to SmartNIC and SmartSwitch that can fully utilize data sparsity.

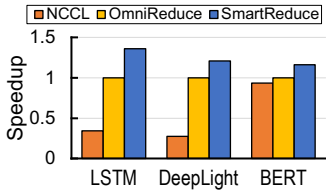
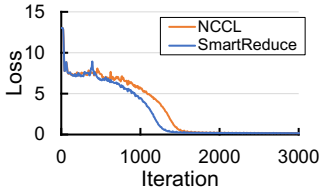
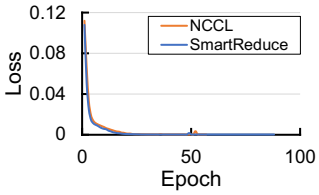


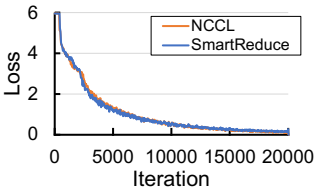
Fig. 13: Training speedup.



(a) LSTM.



(b) DeepLight.



(c) BERT.

Fig. 14: Training loss of three popular models.

Leveraging Sparsity in LLM. To improve training efficiency in LLM, recent work has increasingly leveraged the inherent sparsity of gradients. Gradient-Mask Tuning (GMT) [43] applies hard thresholding to prune low-magnitude gradients, updating only task-relevant parameters. Sparse Gradient Compression (SGC) [121] performs gradient compression by retaining only a sparse subset of gradient entries, significantly reducing communication and memory overhead. TAGC [122] incorporates the model structure to guide sparsification, reducing communication cost without sacrificing convergence. GradDrop [123] uses stochastic thresholding to drop small gradients during backpropagation, serving as both compression and regularization.

VIII. Conclusion

In this paper, we present SmartReduce, a SmartSwitch-aware, SmartNIC-centric sparse AllReduce that removes CPU-initiated coordination from the critical path. By co-designing NIC-initiated coordination, switch-based aggregation, and fine-grained sparsification, SmartReduce achieves low-latency, highly concurrent sparse aggregation. Experiments on a real prototype show that SmartReduce at 99% sparsity achieves up to 6.4× higher algorithm bandwidth than the state-of-the-art aggregator-server-based sparse AllReduce. These results demonstrate the promise of cooperative SmartNIC–SmartSwitch design for future high-performance sparse collectives.

Acknowledgment

The work is supported by the following grants: the Major Project of the Zhejiang Provincial Natural Science Foundation under Grant No. LD26F020002, the National Natural Science Foundation of China under the grant numbers (62472384, 62441236, U24A20326). Zeke Wang is the corresponding author.

References

[1] 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, Feb. 2009, https://doi.org/10.1016/j.jpdc.2008.09.002.

[2] C.-H. Chu, P. Kousha, A. A. Awan, K. S. Khorassani, H. Subramoni, and D. K. D. K. Panda, "NV-group: Link-efficient reduction for distributed deep learning on modern dense GPU systems," in *Proceedings of the 34th ACM International Conference on Supercomputing*. Barcelona Spain: ACM, Jun. 2020, pp. 1–12, https://doi.org/10.1145/3392717.3392771.

[3] S. Shi, Z. Tang, X. Chu, C. Liu, W. Wang, and B. Li, "A Quantitative Survey of Communication Optimizations in Distributed Deep Learning," *IEEE Network*, vol. 35, no. 3, pp. 230–237, May 2021, https://doi.org/10.1109/MNET.011.2000530.

[4] S. Gan, J. Jiang, B. Yuan, C. Zhang, X. Lian, R. Wang, J. Chang, C. Liu, H. Shi, S. Zhang, X. Li, T. Sun, S. Yang, and J. Liu, "Bagua: Scaling up distributed learning with system relaxations," *Proceedings of the VLDB Endowment*, vol. 15, no. 4, pp. 804–813, Dec. 2021, https://doi.org/10.14778/3503585.3503590.

[5] R. Mayer and H.-A. Jacobsen, "Scalable Deep Learning on Distributed Infrastructures: Challenges, Techniques, and Tools," *ACM Computing Surveys*, vol. 53, no. 1, pp. 1–37, Jan. 2021, https://doi.org/10.1145/3363554.

[6] Y. Zou, X. Jin, Y. Li, Z. Guo, E. Wang, and B. Xiao, "Mariana: Tencent deep learning platform and its applications," *Proceedings of the VLDB Endowment*, vol. 7, no. 13, pp. 1772–1777, Aug. 2014, https://doi.org/10.14778/2733004.2733082.

[7] H. Li, A. Kadav, E. Kruus, and C. Ungureanu, "MALT: Distributed data-parallelism for existing ML applications," in *Proceedings of the Tenth European Conference on Computer Systems*. Bordeaux France: ACM, Apr. 2015, pp. 1–16, https://doi.org/10.1145/2741948.2741965.

[8] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. San Francisco California USA: ACM, Aug. 2016, pp. 785–794, https://doi.org/10.1145/2939672.2939785.

[9] J. Jiang, B. Cui, C. Zhang, and F. Fu, "DimBoost: Boosting Gradient Boosting Decision Tree to Higher Dimensions," in *Proceedings of the 2018 International Conference on Management of Data*. Houston TX USA: ACM, May 2018, pp. 1363–1376, https://doi.org/10.1145/3183713.3196892.

[10] Z. Zhang, J. Jiang, W. Wu, C. Zhang, L. Yu, and B. Cui, "MLlib*: Fast Training of GLMs Using Spark MLlib," in *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE Computer Society, Apr. 2019, pp. 1778–1789, https://doi.org/10.1109/ICDE.2019.00194.

[11] R. Y. Aminabadi, S. Rajbhandari, A. A. Awan, C. Li, D. Li, E. Zheng, O. Ruwase, S. Smith, M. Zhang, and J. Rasley, "Deepspeed-inference: Enabling efficient inference of transformer models at unprecedented scale," in *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2022, pp. 1–15, https://doi.org/10.1109/SC41404.2022.00051.

[12] S. Rajbhandari, C. Li, Z. Yao, M. Zhang, R. Y. Aminabadi, A. A. Awan, J. Rasley, and Y. He, "Deepspeed-moe: Advancing mixture-of-experts inference and training to power next-generation ai scale," in *International Conference on Machine Learning*. PMLR, 2022, pp. 18 332–18 346.

[13] N. Luehr, "Fast Multi-GPU collectives with NCCL," https://developer.nvidia.com/blog/fast-multi-gpu-collectives-nccl/, Apr. 2016.

[14] X. Jia, S. Song, W. He, Y. Wang, H. Rong, F. Zhou, L. Xie, Z. Guo, Y. Yang, L. Yu, T. Chen, G. Hu, S. Shi, and X. Chu, "Highly Scalable Deep Learning Training System with Mixed-Precision: Training ImageNet in Four Minutes," Jul. 2018, https://doi.org/10.48550/arXiv.1807.11205.

[15] L. Zhang, S. Shi, X. Chu, W. Wang, B. Li, and C. Liu, "DeAR: Accelerating Distributed Deep Learning with Fine-Grained All-Reduce Pipelining," in *2023 IEEE 43rd International Conference on Distributed Computing Systems (ICDCS)*, Jul. 2023, pp. 142–153, https://doi.org/10.1109/ICDCS57875.2023.00054.

[16] E. Warraich, O. Shabtai, K. Manaa, S. Vargaftik, Y. Piasetzky, M. Kadosh, L. Suresh, and M. Shahbaz, "Ultima: Robust and Tail-Optimal AllReduce for Distributed Deep Learning in the Cloud," Oct. 2023, https://doi.org/arXiv.2310.06993.

[17] A. A. Awan, K. Hamidouche, A. Venkatesh, and D. K. Panda, "Efficient Large Message Broadcast using NCCL and CUDA-Aware MPI for Deep Learning," in *Proceedings of the 23rd European MPI Users'*

Group Meeting. Edinburgh United Kingdom: ACM, Sep. 2016, pp. 15–22, https://doi.org/10.1145/2966884.2966912.

[18] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour," Apr. 2018, https://doi.org/10.48550/arXiv.1706.02677.

[19] J. H. Park, G. Yun, M. Y. Chang, N. T. Nguyen, S. Lee, J. Choi, S. H. Noh, and Y.-r. Choi, "{HetPipe}: Enabling large {DNN} training on (whimpy) heterogeneous {GPU} clusters through integration of pipelined model parallelism and data parallelism," in *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, 2020, pp. 307–321.

[20] H. Fan, H. Zhou, G. Huang, P. Raman, X. Fu, G. Gupta, D. Ram, Y. Wang, and J. Huan, "HLAT: High-quality Large Language Model Pre-trained on AWS Trainium," Nov. 2024, https://doi.org/10.48550/arXiv.2404.10630.

[21] D. Arfeen, Z. Zhang, X. Fu, G. R. Ganger, and Y. Wang, "PipeFill: Using GPUs During Bubbles in Pipeline-parallel LLM Training," Sep. 2024, https://doi.org/10.48550/arXiv.2410.07192.

[22] J. Ren, S. Rajbhandari, R. Y. Aminabadi, O. Ruwase, S. Yang, M. Zhang, D. Li, and Y. He, "{Zero-offload}: Democratizing {billion-Scale} model training," in *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, 2021, pp. 551–564.

[23] J. Liu, B. Nicolae, and D. Li, "Lobster: Load Balance-Aware I/O for Distributed DNN Training," in *Proceedings of the 51st International Conference on Parallel Processing*. Bordeaux France: ACM, Aug. 2022, pp. 1–11, https://doi.org/10.1145/3545008.3545090.

[24] X. He, J. Liu, Z. Xie, H. Chen, G. Chen, W. Zhang, and D. Li, "Enabling energy-efficient DNN training on hybrid GPU-FPGA accelerators," in *Proceedings of the ACM International Conference on Supercomputing*. Virtual Event USA: ACM, Jun. 2021, pp. 227–241, https://doi.org/10.1145/3447818.3460371.

[25] A. Mohan, R. Walkup, B. Karacali, M.-h. Chen, A. Kayi, L. Schour, S. Salaria, S. Wen, I.-h. Chung, A. Alim, C. Evangelinos, L. Luo, M. Dombrowa, L. Schares, A. Sydney, P. Maniotis, S. Koteshwara, B. Tang, J. Belog, R. Odaira, V. Tarasov, E. Gampel, D. Thorstensen, T. Gershon, and S. Seelam, "Vela: A Virtualized LLM Training System with GPU Direct RoCE," in *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. Rotterdam Netherlands: ACM, Mar. 2025, pp. 1348–1364, https://doi.org/10.1145/3676641.3716280.

[26] W. Cai, L. Qin, and J. Huang, "MoC-System: Efficient Fault Tolerance for Sparse Mixture-of-Experts Model Training," in *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. Rotterdam Netherlands: ACM, Mar. 2025, pp. 655–671, https://doi.org/10.1145/3676641.3716006.

[27] J. Fei, C.-Y. Ho, A. N. Sahu, M. Canini, and A. Sapio, "Efficient sparse collective communication and its application to accelerate distributed deep learning," in *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*. Virtual Event USA: ACM, Aug. 2021, pp. 676–691, https://doi.org/10.1145/3452296.3472904.

[28] T. Gu, J. Fei, and M. Canini, "OmNICCL: Zero-cost Sparse AllReduce with Direct Cache Access and SmartNICs," in *Proceedings of the 2024 SIGCOMM Workshop on Networks for AI Computing*. Sydney NSW Australia: ACM, Aug. 2024, pp. 75–83, https://doi.org/10.1145/3672198.3673804.

[29] Z. Wang, Z. Xu, A. Shrivastava, and T. S. E. Ng, "Zen: Near-Optimal Sparse Tensor Synchronization for Distributed DNN Training," Sep. 2023, https://doi.org/arXiv.2309.13254.

[30] H. Pan, P. Cui, Z. li, R. Jia, P. Zhang, L. Zhang, Y. Yang, J. Wu, J. Dong, Z. Cao, Q. Li, H. H. Liu, M. Laurent, and G. Xie, "Enabling Fast and Flexible Distributed Deep Learning with Programmable Switches," Aug. 2022, https://doi.org/arXiv.2205.05243.

[31] Y. Lin, S. Han, H. Mao, Y. Wang, and W. J. Dally, "Deep Gradient Compression: Reducing the Communication Bandwidth for Distributed Training," Jun. 2020, https://doi.org/arXiv.1712.01887.

[32] S. Li and T. Hoefler, "Near-optimal sparse allreduce for distributed deep learning," in *Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. Seoul Republic of Korea: ACM, Apr. 2022, pp. 135–149, https://doi.org/10.1145/3503221.3508399.

[33] C. Chen, M. Li, and C. Yang, "bbTopk: Bandwidth-Aware Sparse Allreduce with Blocked Sparsification for Efficient Distributed Training," in *2023 IEEE 43rd International Conference on Distributed Computing Systems (ICDCS)*, Jul. 2023, pp. 1–11, https://doi.org/10.1109/ICDCS57875.2023.00015.

[34] J. Peng, Z. Li, S. Shi, and B. Li, "Sparse Gradient Communication with AlltoAll for Accelerating Distributed Deep Learning," in *Proceedings of the 53rd International Conference on Parallel Processing*, ser. ICPP '24. New York, NY, USA: Association for Computing Machinery, Aug. 2024, pp. 148–157, https://doi.org/10.1145/3673038.3673140.

[35] H. Xu, K. Kostopoulou, A. Dutta, X. Li, A. Ntoulas, and P. Kalnis, "DeepReduce: A Sparse-tensor Communication Framework for Federated Deep Learning," in *Advances in Neural Information Processing Systems*, vol. 34. Curran Associates, Inc., 2021, pp. 21 150–21 163.

[36] Z. Wang, X. Wu, and T. S. E. Ng, "MergeComp: A Compression Scheduler for Scalable Communication-Efficient Distributed Training," Mar. 2021, https://doi.org/10.48550/arXiv.2103.15195.

[37] Z. Wang, H. Lin, Y. Zhu, and T. S. E. Ng, "Hi-Speed DNN Training with Espresso: Unleashing the Full Potential of Gradient Compression with Near-Optimal Usage Strategies," in *Proceedings of the Eighteenth European Conference on Computer Systems*. Rome Italy: ACM, May 2023, pp. 867–882, https://doi.org/10.1145/3552326.3567505.

[38] Z. Chai, Y. Chen, A. Anwar, L. Zhao, Y. Cheng, and H. Rangwala, "FedAT: A high-performance and communication-efficient federated learning system with asynchronous tiers," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. St. Louis Missouri: ACM, Nov. 2021, pp. 1–16, https://doi.org/10.1145/3458817.3476211.

[39] A. Sarma, S. Singh, H. Jiang, A. Pattnaik, A. K. Mishra, V. Narayanan, M. T. Kandemir, and C. R. Das, "Exploiting Activation based Gradient Output Sparsity to Accelerate Backpropagation in CNNs," Sep. 2021, https://doi.org/10.48550/arXiv.2109.07710.

[40] M. Pellauer, J. Clemons, V. Balaji, N. Crago, A. Jaleel, D. Lee, M. O'Connor, A. Parashar, S. Treichler, P.-A. Tsai, S. W. Keckler, and J. S. Emer, "Symphony: Orchestrating Sparse and Dense Tensors with Hierarchical Heterogeneous Processing," *ACM Transactions on Computer Systems*, vol. 41, no. 1-4, pp. 1–30, Nov. 2023, https://doi.org/10.1145/3630007.

[41] G. Huang, H. Li, L. Qin, J. Huang, Y. Kang, Y. Ding, and Y. Xie, "TRACI: Network Acceleration of Input-Dynamic Communication for Large-Scale Deep Learning Recommendation Model," in *Proceedings of the 52nd Annual International Symposium on Computer Architecture*. Tokyo Japan: ACM, Jun. 2025, pp. 1880–1893, https://doi.org/10.1145/3695053.3731105.

[42] Y. Zhang, P. Ouyang, S. Yin, Y. Zhang, W. Zhao, and S. Wei, "A Computing Efficient Hardware Architecture for Sparse Deep Neural Network Computing," in *2018 14th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT)*, Oct. 2018, pp. 1–3, https://doi.org/10.1109/ICSICT.2018.8565755.

[43] H. Li, X. Zhang, X. Liu, Y. Gong, Y. Wang, Y. Yang, Q. Chen, and P. Cheng, "Gradient-Mask Tuning Elevates the Upper Limits of LLM Performance," Jun. 2024, https://doi.org/10.48550/arXiv.2406.15330.

[44] Y. N. Wu, P.-A. Tsai, S. Muralidharan, A. Parashar, V. Sze, and J. Emer, "HighLight: Efficient and Flexible DNN Acceleration with Hierarchical Structured Sparsity," in *56th Annual IEEE/ACM International Symposium on Microarchitecture*. Toronto ON Canada: ACM, Oct. 2023, pp. 1106–1120, https://doi.org/10.1145/3613424.3623786.

[45] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "LoRA: Low-Rank Adaptation of Large Language Models," Oct. 2021, https://doi.org/10.48550/arXiv.2106.09685.

[46] S. Khaki, X. Li, J. Guo, L. Zhu, C. Xu, K. N. Plataniotis, A. Yazdanbakhsh, K. Keutzer, S. Han, and Z. Liu, "SparseLoRA: Accelerating LLM Fine-Tuning with Contextual Sparsity," Jun. 2025, https://doi.org/10.48550/arXiv.2506.16500.

[47] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "QLoRA: Efficient finetuning of quantized llms," in *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., vol. 36. Curran Associates, Inc., 2023, pp. 10 088–10 115.

[48] A. Koloskova, H. Hendrikx, and S. U. Stich, "Revisiting Gradient Clipping: Stochastic bias and tight convergence guarantees," in *International Conference on Machine Learning*. PMLR, 2023, pp. 17 343–17 363.

[49] J. Qian, Y. Wu, B. Zhuang, S. Wang, and J. Xiao, "Understanding gradient clipping in incremental gradient methods," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2021, pp. 1504–1512.

[50] J. Zhang, T. He, S. Sra, and A. Jadbabaie, "Why gradient clipping accelerates training: A theoretical justification for adaptivity," Feb. 2020, https://doi.org/10.48550/arXiv.1905.11881.

[51] X. Ye, P. Dai, J. Luo, X. Guo, Y. Qi, J. Yang, and Y. Chen, "Accelerating CNN Training by Pruning Activation Gradients," in *Computer Vision – ECCV 2020*, A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, Eds. Cham: Springer International Publishing, 2020, pp. 322–338, https://doi.org/10.1007/978-3-030-58595-2_20.

[52] Y. Wang, Y. Qin, L. Liu, S. Wei, and S. Yin, "HPPU: An Energy-Efficient Sparse DNN Training Processor with Hybrid Weight Pruning," in *2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, Jun. 2021, pp. 1–4, https://doi.org/10.1109/AICAS51828.2021.9458410.

[53] B. McDanel, H. Dinh, and J. Magallanes, "Accelerating DNN Training with Structured Data Gradient Pruning," in *2022 26th International Conference on Pattern Recognition (ICPR)*, Aug. 2022, pp. 2293–2299, https://doi.org/10.1109/ICPR56361.2022.9956574.

[54] H. Tang, C. Yu, X. Lian, T. Zhang, and J. Liu, "Doublesqueeze: Parallel stochastic gradient descent with double-pass error-compensated compression," in *International Conference on Machine Learning*. PMLR, 2019, pp. 6155–6165.

[55] P. Richtárik, I. Sokolov, and I. Fatkhullin, "EF21: A new, simpler, theoretically better, and practically faster error feedback," *Advances in Neural Information Processing Systems*, vol. 34, pp. 4384–4396, 2021.

[56] E. Gorbunov, D. Kovalev, D. Makarenko, and P. Richtárik, "Linearly converging error compensated SGD," *Advances in Neural Information Processing Systems*, vol. 33, pp. 20 889–20 900, 2020.

[57] H. Zhao and J. Canny, "Kylix: A Sparse Allreduce for Commodity Clusters," in *Proceedings of the 2014 Brazilian Conference on Intelligent Systems*, ser. BRACIS '14. USA: IEEE Computer Society, Sep. 2014, pp. 273–282, https://doi.org/10.1109/ICPP.2014.36.

[58] "Support sparse gradients in DistributedDataParallel," https://github.com/pytorch/pytorch/pull/22037.

[59] C. Renggli, S. Ashkboos, M. Aghagolzadeh, D. Alistarh, and T. Hoefler, "SparCML: High-performance sparse communication for machine learning," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. Denver Colorado: ACM, Nov. 2019, pp. 1–15, https://doi.org/10.1145/3295500.3356222.

[60] "Gloo: Collective communications library with various primitives for multi-machine training," Meta Incubator, https://github.com/facebookincubator/gloo.

[61] 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.

[62] C. Lao, Y. Le, K. Mahajan, Y. Chen, W. Wu, A. Akella, and M. Swift, "ATP: In-network Aggregation for Multi-tenant Learning," in *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, 2021, pp. 741–761.

[63] C. Zhao, C. Deng, C. Ruan, D. Dai, H. Gao, J. Li, L. Zhang, P. Huang, S. Zhou, S. Ma, W. Liang, Y. He, Y. Wang, Y. Liu, and Y. Wei, "Insights into DeepSeek-V3: Scaling Challenges and Reflections on Hardware for AI Architectures," in *Proceedings of the 52nd Annual International Symposium on Computer Architecture*. Tokyo Japan: ACM, Jun. 2025, pp. 1731–1745, https://doi.org/10.1145/3695053.3731412.

[64] Y. Li, A. Kashyap, W. Chen, Y. Guo, and X. Lu, "Accelerating Lossy and Lossless Compression on Emerging BlueField DPU Architectures," in *2024 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2024, pp. 373–385, https://doi.org/10.1109/IPDPS57955.2024.00040.

[65] Y. Li, A. Kashyap, Y. Guo, and X. Lu, "Characterizing Lossy and Lossless Compression on Emerging BlueField DPU Architectures," in *2023 IEEE Symposium on High-Performance Interconnects (HOTI)*, Aug. 2023, pp. 33–40, https://doi.org/10.1109/HOTI59126.2023.00019.

[66] ——, "Compression Analysis for BlueField-2/-3 Data Processing Units: Lossy and Lossless Perspectives," *IEEE Micro*, vol. 44, no. 2, pp. 8–19, Mar. 2024, https://doi.org/10.1109/MM.2023.3343636.

[67] B. Asgari, R. Hadidi, J. Cao, D. E. Shim, S.-K. Lim, and H. Kim, "FAFNIR: Accelerating Sparse Gathering by Using Efficient Near-Memory Intelligent Reduction," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, Feb. 2021, pp. 908–920, https://doi.org/10.1109/HPCA51647.2021.00080.

[68] W. Deng, J. Pan, T. Zhou, D. Kong, A. Flores, and G. Lin, "DeepLight: Deep Lightweight Feature Interactions for Accelerating CTR Predictions in Ad Serving," in *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*. Virtual Event Israel: ACM, Mar. 2021, pp. 922–930, https://doi.org/10.1145/3437963.3441727.

[69] R. Neugebauer, G. Antichi, J. F. Zazo, Y. Audzevich, S. López-Buedo, and A. W. Moore, "Understanding PCIe performance for end host networking," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. Budapest Hungary: ACM, Aug. 2018, pp. 327–341, https://doi.org/10.1145/3230543.3230560.

[70] M. Vuppalapati, S. Agarwal, H. Schuh, B. Kasikci, A. Krishnamurthy, and R. Agarwal, "Understanding the Host Network," in *Proceedings of the ACM SIGCOMM 2024 Conference*, ser. ACM SIGCOMM '24. New York, NY, USA: Association for Computing Machinery, Aug. 2024, pp. 581–594, https://doi.org/10.1145/3651890.3672271.

[71] Z. Wang, H. Huang, J. Zhang, F. Wu, and G. Alonso, "{FpgaNIC}: An {FPGA-based} Versatile 100Gb {SmartNIC} for {GPUs}," in *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, 2022, pp. 967–986.

[72] W. Lin, Y. Shan, R. Kosta, A. Krishnamurthy, and Y. Zhang, "SuperNIC: An FPGA-Based, Cloud-Oriented SmartNIC," in *Proceedings of the 2024 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*. Monterey CA USA: ACM, Apr. 2024, pp. 130–141, https://doi.org/10.1145/3626202.3637564.

[73] S. Grant, A. Yelam, M. Bland, and A. C. Snoeren, "SmartNIC Performance Isolation with FairNIC: Programmable Networking for the Cloud," in *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*. Virtual Event USA: ACM, Jul. 2020, pp. 681–693, https://doi.org/10.1145/3387514.3405895.

[74] Z. Wang, J. Zhang, H. Huang, Y. Li, X. Zhu, M. Sun, Z. Yang, D. Ma, H. Tang, G. Pan, F. Wu, B. He, and G. Alonso, "FpgaHub: Fpga-centric Hyper-heterogeneous Computing Platform for Big Data Analytics," Mar. 2025, https://doi.org/10.48550/arXiv.2503.09318.

[75] B. Li, Z. Ruan, W. Xiao, Y. Lu, Y. Xiong, A. Putnam, E. Chen, and L. Zhang, "KV-Direct: High-Performance In-Memory Key-Value Store with Programmable NIC," in *Proceedings of the 26th Symposium on Operating Systems Principles*. Shanghai China: ACM, Oct. 2017, pp. 137–152, https://doi.org/10.1145/3132747.3132756.

[76] 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)*, 2018, pp. 51–66.

[77] J. Lin, K. Patel, B. E. Stephens, A. Sivaraman, and A. Akella, "PANIC: A High-Performance Programmable NIC for Multi-tenant Networks," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, 2020, pp. 243–259.

[78] A. Guo, T. Geng, Y. Zhang, P. Haghi, C. Wu, C. Tan, Y. Lin, A. Li, and M. Herbordt, "FCsN: A FPGA-Centric SmartNIC Framework for Neural Networks," in *2022 IEEE 30th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, May 2022, pp. 1–2, https://doi.org/10.1109/FCCM53951.2022.9786193.

[79] H. Ji, M. Mansi, Y. Sun, Y. Yuan, J. Huang, R. Kuper, M. M. Swift, and N. S. Kim, "STYX: Exploiting SmartNIC Capability to Reduce Datacenter Memory Tax," in *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, 2023, pp. 619–633.

[80] M. Khalilov, S. D. Girolamo, M. Chrapek, R. Nudelman, G. Bloch, and T. Hoefler, "Network-Offloaded Bandwidth-Optimal Broadcast and Allgather for Distributed AI," in *SC24: International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov. 2024, pp. 1–17, https://doi.org/10.1109/SC41406.2024.00109.

[81] A. Kashyap, Y. Li, and X. Lu, "Dpu-kv: On the benefits of dpu offloading for in-memory key-value stores at the edge," in *Proceedings of the 34th International Symposium on High-Performance Parallel and*

Distributed Computing, 2025, pp. 1–14, https://doi.org/10.1145/3731545.3731571.

[82] X. Wei, R. Cheng, Y. Yang, R. Chen, and H. Chen, "Characterizing Off-path {SmartNIC} for Accelerating Distributed Systems," in *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, 2023, pp. 987–1004.

[83] "Nvidia bluefield platform," https://www.nvidia.com/en-us/networking/products/data-processing-unit/.

[84] H. Shi and X. Lu, "TriEC: Tripartite graph based erasure coding NIC offload," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. Denver Colorado: ACM, Nov. 2019, pp. 1–34, https://doi.org/10.1145/3295500.3356178.

[85] A. Kashyap, Y. Li, D. Ng, and X. Lu, "Understanding the idiosyncrasies of emerging bluefield dpus," in *Proceedings of the 39th ACM International Conference on Supercomputing*, 2025, pp. 807–821, https://doi.org/10.1145/3721145.3725780.

[86] C. Xuzheng, Z. Jie, Z. Baolin, Z. Xueying, C. Zhongqing, F. Ting, M. Shu, Z. Lingjun, C. Shi, Z. Yin, S. Yunchao, C. Peng, and W. Zeke, "Smartns: Enabling line-rate and flexible network stack with smartnic," in *European Conference on Computer Systems (EuroSys)*, 2026.

[87] K. Qian, Y. Xi, J. Cao, J. Gao, Y. Xu, Y. Guan, B. Fu, X. Shi, F. Zhu, R. Miao, C. Wang, P. Wang, P. Zhang, X. Zeng, E. Ruan, Z. Yao, E. Zhai, and D. Cai, "Alibaba HPN: A Data Center Network for Large Language Model Training," in *Proceedings of the ACM SIGCOMM 2024 Conference*. Sydney NSW Australia: ACM, Aug. 2024, pp. 691–706, https://doi.org/10.1145/3651890.3672265.

[88] "Doubling all2all performance with nvidia collective communication library 2.12," https://developer.nvidia.com/blog/doubling-all2all-performance-with-nvidia-collective-communication-library-2-12/, 2022.

[89] Z. Jiang, H. Lin, Y. Zhong, Q. Huang, Y. Chen, Z. Zhang, Y. Peng, X. Li, C. Xie, S. Nong, Y. Jia, S. He, H. Chen, Z. Bai, Q. Hou, S. Yan, D. Zhou, Y. Sheng, Z. Jiang, H. Xu, H. Wei, Z. Zhang, P. Nie, L. Zou, S. Zhao, L. Xiang, Z. Liu, Z. Li, X. Jia, J. Ye, X. Jin, and X. Liu, "{MegaScale}: Scaling Large Language Model Training to More Than 10,000 {GPUs}," in *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, 2024, pp. 745–760.

[90] G. Xiong, G. Yan, S. Wang, and J. Li, "Straggler-Resilient Decentralized Learning via Adaptive Asynchronous Updates," Jun. 2023, https://doi.org/10.48550/arXiv.2306.06559.

[91] H. Kim, C. Song, H. Lee, and H. Yu, "Addressing Straggler Problem Through Dynamic Partial All-Reduce for Distributed Deep Learning in Heterogeneous GPU Clusters," in *2023 IEEE International Conference on Consumer Electronics (ICCE)*. Las Vegas, NV, USA: IEEE, Jan. 2023, pp. 1–6, https://doi.org/10.1109/ICCE56470.2023.10043527.

[92] R. Jozefowicz, O. Vinyals, M. Schuster, N. Shazeer, and Y. Wu, "Exploring the Limits of Language Modeling," Feb. 2016, https://doi.org/10.48550/arXiv.1602.02410.

[93] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," May 2019, https://doi.org/10.48550/arXiv.1810.04805.

[94] C. Chelba, T. Mikolov, M. Schuster, Q. Ge, T. Brants, P. Koehn, and T. Robinson, "One Billion Word Benchmark for Measuring Progress in Statistical Language Modeling," Mar. 2014, https://doi.org/10.48550/arXiv.1312.3005.

[95] P. Rajpurkar, R. Jia, and P. Liang, "Know What You Don't Know: Unanswerable Questions for SQuAD," Jun. 2018, https://doi.org/10.48550/arXiv.1806.03822.

[96] M. Cho, U. Finkler, M. Serrano, D. Kung, and H. Hunter, "BlueConnect: Decomposing all-reduce for deep learning on heterogeneous network hierarchy," *IBM Journal of Research and Development*, vol. 63, no. 6, pp. 1:1–1:11, Nov. 2019, https://doi.org/10.1147/JRD.2019.2947013.

[97] A. Li, S. L. Song, J. Chen, J. Li, X. Liu, N. R. Tallent, and K. J. Barker, "Evaluating Modern GPU Interconnect: PCIe, NVLink, NV-SLI, NVSwitch and GPUDirect," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 1, pp. 94–110, Jan. 2020, https://doi.org/10.1109/TPDS.2019.2928289.

[98] Q. Weng, W. Xiao, Y. Yu, W. Wang, C. Wang, J. He, Y. Li, L. Zhang, W. Lin, and Y. Ding, "MLaaS in the Wild: Workload Analysis and Scheduling in Large-Scale Heterogeneous {GPU} Clusters," in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, 2022, pp. 945–960.

[99] G. Wang, A. Phanishayee, S. Venkataraman, and I. Stoicat, "Blink: A fast NVLink-based collective communication library," in *Proc. Conf. Syst. Mach. Learn*, 2018.

[100] L. Zhao, S. Maleki, Z. Yang, H. Pourreza, A. Shah, C. Hwang, and A. Krishnamurthy, "ForestColl: Efficient Collective Communications on Heterogeneous Network Fabrics," Feb. 2024, https://doi.org/10.48550/arXiv.2402.06787.

[101] Z. Cai, Z. Liu, S. Maleki, M. Musuvathi, T. Mytkowicz, J. Nelson, and O. Saarikivi, "Synthesizing optimal collective algorithms," in *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. Virtual Event Republic of Korea: ACM, Feb. 2021, pp. 62–75, https://doi.org/10.1145/3437801.3441620.

[102] A. Shah, V. Chidambaram, M. Cowan, S. Maleki, M. Musuvathi, T. Mytkowicz, J. Nelson, O. Saarikivi, and R. Singh, "TACCL: Guiding Collective Algorithm Synthesis using Communication Sketches," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, 2023, pp. 593–612.

[103] L. Mai, L. Rupprecht, A. Alim, P. Costa, M. Migliavacca, P. Pietzuch, and A. L. Wolf, "NetAgg: Using Middleboxes for Application-specific On-path Aggregation in Data Centres," in *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '14. New York, NY, USA: Association for Computing Machinery, Dec. 2014, pp. 249–262, https://doi.org/10.1145/2674005.2674996.

[104] 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*, ser. ISCA '19. New York, NY, USA: Association for Computing Machinery, Jun. 2019, pp. 279–291, https://doi.org/10.1145/3307650.3322259.

[105] B. Klenk, N. Jiang, G. Thorson, and L. Dennison, "An In-Network Architecture for Accelerating Shared-Memory Multiprocessor Collectives," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, May 2020, pp. 996–1009, https://doi.org/10.1109/ISCA45697.2020.00085.

[106] H. Huang, Y. Li, J. Sun, X. Zhu, J. Zhang, L. Luo, J. Li, and Z. Wang, "P4SGD: Programmable Switch Enhanced Model-Parallel Training on Generalized Linear Models on Distributed FPGAs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 8, pp. 2311–2324, Aug. 2023, https://doi.org/10.1109/TPDS.2023.3279255.

[107] Y. Li, J. Park, M. Alian, Y. Yuan, Z. Qu, P. Pan, R. Wang, A. Schwing, H. Esmaeilzadeh, and N. S. Kim, "A Network-Centric Hardware/Algorithm Co-Design to Accelerate Distributed Training of Deep Neural Networks," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Oct. 2018, pp. 175–188, https://doi.org/10.1109/MICRO.2018.00023.

[108] L. Luo, M. Liu, J. Nelson, L. Ceze, A. Phanishayee, and A. Krishnamurthy, "Motivating in-network aggregation for distributed deep neural network training," in *Workshop on Approximate Computing Across the Stack*, 2017.

[109] Y. He, W. Wu, Y. Le, M. Liu, and C. Lao, "A Generic Service to Provide In-Network Aggregation for Key-Value Streams," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. Vancouver BC Canada: ACM, Jan. 2023, pp. 33–47, https://doi.org/10.1145/3575693.3575708.

[110] A. Sapio, I. Abdelaziz, A. Aldilaijan, M. Canini, and P. Kalnis, "In-Network Computation is a Dumb Idea Whose Time Has Come," in *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*, ser. HotNets '17. New York, NY, USA: Association for Computing Machinery, Nov. 2017, pp. 150–156, https://doi.org/10.1145/3152434.3152461.

[111] R. L. Graham, D. Bureddy, P. Lui, H. Rosenstock, G. Shainer, G. Bloch, D. Goldenerg, M. Dubman, S. Kotchubievsky, V. Koushnir, L. Levi, A. Margolin, T. Ronen, A. Shpiner, O. Wertheim, and E. Zahavi, "Scalable Hierarchical Aggregation Protocol (SHArP): A Hardware Architecture for Efficient Data Reduction," in *2016 First International Workshop on Communication Optimizations in HPC (COMHPC)*, Nov. 2016, pp. 1–10, https://doi.org/10.1109/COMHPC.2016.006.

[112] M. Yang, A. Baban, V. Kugel, J. Libby, S. Mackie, S. S. R. Kananda, C.-H. Wu, and M. Ghobadi, "Using trio: Juniper networks' programmable chipset - for emerging in-network applications," in *Proceedings of the ACM SIGCOMM 2022 Conference*. Amsterdam

Netherlands: ACM, Aug. 2022, pp. 633–648, https://doi.org/10.1145/3544216.3544262.

[113] F. Sattler, S. Wiedemann, K.-R. Muller, and W. Samek, "Sparse Binary Compression: Towards Distributed Deep Learning with minimal Communication," in *2019 International Joint Conference on Neural Networks (IJCNN)*. Budapest, Hungary: IEEE, Jul. 2019, pp. 1–8, https://doi.org/10.1109/IJCNN.2019.8852172.

[114] T. T. Nguyen, M. Wahib, and R. Takano, "Topology-aware Sparse Allreduce for Large-scale Deep Learning," in *2019 IEEE 38th International Performance Computing and Communications Conference (IPCCC)*, Oct. 2019, pp. 1–8, https://doi.org/10.1109/IPCCC47392.2019.8958738.

[115] Z. Wang, Z. Xu, X. Wu, A. Shrivastava, and T. S. E. Ng, "DRAGONN: Distributed Randomized Approximate Gradients of Neural Networks," in *Proceedings of the 39th International Conference on Machine Learning*. PMLR, Jun. 2022, pp. 23 274–23 291.

[116] S. Kim, G.-I. Yu, H. Park, S. Cho, E. Jeong, H. Ha, S. Lee, J. S. Jeong, and B.-G. Chun, "Parallax: Sparsity-aware Data Parallel Training of Deep Neural Networks," in *Proceedings of the Fourteenth EuroSys Conference 2019*. Dresden Germany: ACM, Mar. 2019, pp. 1–15, https://doi.org/10.1145/3302424.3303957.

[117] Y. Bai, C. Li, Q. Zhou, J. Yi, P. Gong, F. Yan, R. Chen, and Y. Xu, "Gradient Compression Supercharged High-Performance Data Parallel DNN Training," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*. Virtual Event Germany: ACM, Oct. 2021, pp. 359–375, https://doi.org/10.1145/3477132.3483553.

[118] H. Xu, C.-Y. Ho, A. M. Abdelmoniem, A. Dutta, E. H. Bergou, K. Karatsenidis, M. Canini, and P. Kalnis, "GRACE: A Compressed Communication Framework for Distributed Machine Learning," in *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*, Jul. 2021, pp. 561–572, https://doi.org/10.1109/ICDCS51616.2021.00060.

[119] J. Huang, S. Di, X. Yu, Y. Zhai, J. Liu, Y. Huang, K. Raffenetti, H. Zhou, K. Zhao, X. Lu *et al.*, "gzccl: Compression-accelerated collective communication framework for gpu clusters," in *Proceedings of the 38th ACM International Conference on Supercomputing*, 2024, https://doi.org/10.1145/3650200.3656636.

[120] J. Huang, S. Di, X. Yu, Y. Zhai, J. Liu, Z. Jian, X. Liang, K. Zhao, X. Lu, Z. Chen *et al.*, "hzccl: Accelerating collective communication with co-designed homomorphic compression," in *SC24: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2024, https://doi.org/10.1109/SC41406.2024.00110.

[121] D. H. Yang, M. M. Amiri, T. Pedapati, S. Chaudhury, and P.-Y. Chen, "Sparse Gradient Compression for Fine-Tuning Large Language Models," Feb. 2025, https://doi.org/10.48550/arXiv.2502.00311.

[122] I. Polyakov, A. Dukhanov, and E. Spirin, "TAGC: Optimizing Gradient Communication in Distributed Transformer Training," in *Proceedings of the 5th Workshop on Machine Learning and Systems (EuroMLSys '25)*, Mar. 2025, pp. 254–260, https://doi.org/10.1145/3721146.3721946.

[123] J. O'Neill and S. Dutta, "Gradient Sparsification For Masked Fine-Tuning of Transformers," in *2023 International Joint Conference on Neural Networks (IJCNN)*, Jun. 2023, pp. 1–8, https://doi.org/10.1109/IJCNN54540.2023.10191206.