Enabling Compute-Communication Overlap in Distributed Deep Learning Training Platforms

Saeed Rashidi[*], Matthew Denton[*], Srinivas Sridharan[†], Sudarshan Srinivasan[‡],
Amoghavarsha Suresh[§], Jade Nie[†], and Tushar Krishna[*]
[*]Georgia Institute of Technology, Atlanta, USA
[†]Facebook, Menlo Park, USA
[‡]Intel, Bangalore, India
[§]Stony Brook University, Stony Brook, USA
saeed.rashidi@gatech.edu, ssrinivas@fb.com, sudarshan.srinivasan@intel.com, tushar@ece.gatech.edu

***Abstract*—Deep Learning (DL) training platforms are built by interconnecting multiple DL accelerators (e.g., GPU/TPU) via fast, customized interconnects with 100s of gigabytes (GBs) of bandwidth. However, as we identify in this work, driving this bandwidth is quite challenging. This is because there is a pernicious balance between using the accelerator's compute and memory for both DL computations and communication.**

This work makes two key contributions. First, via real system measurements and detailed modeling, we provide an understanding of compute and memory bandwidth demands for DL compute and comms. Second, we propose a novel DL collective communication accelerator called *Accelerator Collectives Engine* (ACE) that sits alongside the compute and networking engines at the accelerator endpoint. ACE frees up the endpoint's compute and memory resources for DL compute, which in turn reduces the required memory BW by 3.5× on average to drive the same network BW compared to state-of-the-art baselines. For modern DL workloads and different network sizes, ACE, on average, increases the effective network bandwidth utilization by 1.44× (up to 2.67×), resulting in an average of 1.41× (up to 1.51×), 1.12× (up to 1.17×), and 1.13× (up to 1.19×) speedup in iteration time for ResNet-50, GNMT and DLRM when compared to the best baseline configuration, respectively.

***Index Terms*—communication accelerator, accelerator fabric, deep learning training, collective communication**

I. Introduction

Deep Learning (DL) and Deep Neural network (DNN) models are being deployed pervasively across a wide range of real-world application domains [21], [41], [58]. The size and computational requirements of these DNN models are growing at an unparalleled rate, 2× every 3.4 months [10], to handle the unrelenting growth in data and workload requirements. The advent of energy-efficient accelerators capable of handling these large models and the need for accelerating training time when dealing with 10s to 100s of petabytes of input data is raising the demand for faster and more efficient DL *training* solutions. This can only be achieved through scalable and efficient *distributed* training since a single accelerator cannot satisfy the compute, memory, and I/O requirements of today's state-of-the-art DNNs.

The need for distributed training has led to the emergence of *DL Training Platforms* such as Google Cloud TPU [26], Facebook Zion [11], NVIDIA DGX-1 [39]/DGX-2 [43], and Intel Xe-GPU [8]. A common feature in all these platforms is the use of a customized interconnect connecting the various accelerators together (in addition to the traditional cache-coherent shared-memory CPU network such as QPI and the TCP-IP/RoCE/Infiniband based datacenter network accessible via PCIe/Infiniband NICs), as shown in Fig. 1. In this paper, we use the term *Neural Processing Units* (NPUs) to describe the accelerator (GPUs, TPUs, FPGAs) and the term *Accelerator Fabric* (AF) for the accelerator network[1].

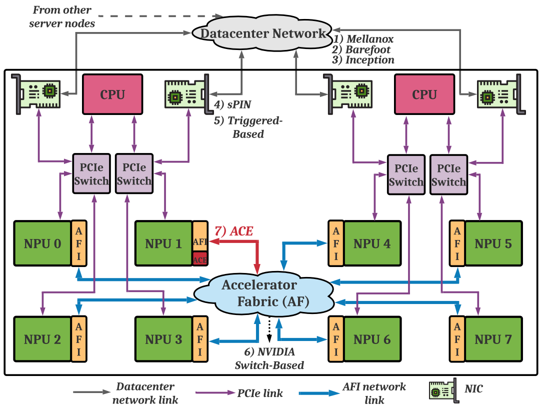
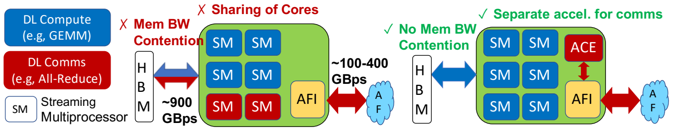


Fig. 1: Architecture of a DL Training Platform (e.g., Google Cloud TPU , NVIDIA DGX-1/DGX-2, Facebook Zion, Intel Xe-GPU [8]). Accelerator Fabrics may be point-to-point (e.g, Hypercube Mesh in Zion [11], [40], Torus in TPU [25], Habana [3]) or switch-based (e.g., NVswitch in DGX-2 [43]). The numbers indicate prior work on offloading collectives and contrasted in Table I



(a) Legend (b) Endpoint in Baseline (c) Endpoint with ACE

Fig. 2: Endpoint NPU Node in baseline systems (today) and with ACE (proposed). In baseline, collective communication tasks contend for NPU cores (e.g., SMs in a GPU) and memory bandwidth.

The AF is different from traditional NIC-based datacenter networks for two reasons: (i) NPUs can directly talk to each other without CPU/NIC intervention, (ii) the AF provides

[1] There is no consistent terminology yet in the community for this fabric. Facebook Zion [11] uses the same terminology as ours, Google Cloud TPU [26] calls it the Inter-core Interconnect (ICI), some papers have called it the device-side interconnect [33] and shared memory fabric [29].

10's of times higher bandwidth than the datacenter network (e.g. 12.5 GB/s vs. 500 GB/s) as it employs a mix of on-package interconnects to bind NPU chiplets together [8], [12] and custom high-bandwidth interconnects like Xe-Link [5] or NVlink [35] to connect these packages together.

While there has been significant work from the architecture community on accelerating the compute portion of DL training via efficient accelerators [25], [30], [31] and memory systems [27], [32], an often ignored part of the story is communication. Distributed DL training fundamentally involves splitting the DNN model, training data, or both across multiple NPUs. These schemes are referred to as model, data, and hybrid parallel, respectively. The parallelization strategy in turn dictates the communication required between NPUs. This communication is *collective* in nature, i.e., all NPUs synchronize input/output/error activations during the forward/backward pass and gradients during the backward pass. Specifically, two collective operations: all-to-all and all-reduce, occur heavily during distributed DL training. These operations are often latency-sensitive (since the next layer of the DNN cannot proceed until gradients have been synchronized[2]) and bandwidth-hungry due to the large sizes of the activations/gradients and limited network bandwidth and hence can easily become bottleneck [36], [51].

One of the primary techniques employed today to minimize the impact of communication is to overlap it behind compute. This is enabled via clever fabric topologies [17], [26] accompanied by topology-aware collective communication algorithms (e.g., Ring and Double-binary tree for All-reduce) implementations in libraries like Intel's MLSL [23]/oneCCL [7] and NVIDIA's NCCL [42]. The Cloud TPU in fact boasts of contention-free O(1) scalability of All-reduce on their Torus [34]. However, in this work, we identify that getting perfect compute-comms overlap on training platforms, despite using optimized topologies and collective algorithm implementations, is prohibitive due to a key architectural bottleneck at the endpoint; both compute and communication contend for the same shared resources: compute units and memory bandwidth.

Fig. 2 shows an NPU and its connected peripherals. We indicate two sources of inefficiency. (i) **Compute:** a portion of the compute resources of the NPU is used for performing collective updates and driving the network, which reduces the compute available to training computations (e.g., GEMMs during the forward and backward passes), (ii) **Memory Bandwidth:** the received gradients need to be written to and read from memory, in turn reducing the bandwidth available for the actual training computations which are known to be highly bandwidth-intensive in modern DNNs such as recommendations [41] and NLP [49]. This problem gets exacerbated as the recent advances in the multi-chip module (MCM) packaging technologies (i.e., silicon interposer) and high-bandwidth inter-package networks provide enormous network BW to the NPUs at the AF level, thus making it more challenging for the NPUs to fully drive the network.

TABLE I: Comparison of previous SmartNIC and switch offload schemes against ACE. Coll. => Collective, DCN => Datacenter Network, AF => Accelerator Fabric

Scheme	App	Offload	Protocol	Topology	Aggr.	Network
Mellanox [6]	HPC, DL	Switch	Infiniband	Switch-based	Coll.	DCN
Barefoot [4]	DL	Switch	Ethernet	Switch-based	Coll.	DCN
sPIN [22]	HPC	NIC	Ethernet	Switch-based	Coll.	DCN
Triggered [56]	HPC	NIC	RDMA-based	Flexible	Coll.	DCN
Inceptionn [37]	DL	NIC	Ethernet	Tree	Param Server	DCN
NVIDIA [29]	DL	Switch	Shared Memory	Switch-based	Coll.**	AF
ACE	DL	Endpoint	RDMA-based	PtToPt, Switch	Coll.	AF

**All-Reduce only

We corroborate the aforementioned issues via real system measurements on a DGX-2 system running both microbenchmarks, real Deep Learning Recommendation Model (DLRM) [41], and Megatron-LM [54] workloads (Section III). The implication of these issues is the under-utilization of the available AF bandwidth as systems scale, which we demonstrate via a detailed simulator. ***To the best of our knowledge, this is the first work to identify these issues.***

To address these inefficiencies, we propose *Accelerator Collectives Engine (ACE)*, which is an accelerator for DL training collectives. ACE sits alongside the *Accelerator Fabric Interface* (AFI) [11] that interfaces NPU to the AF network and handles the communication protocol. ACE houses compute units for running a variety of collective algorithms and scratchpads to cache gradients. This in turn frees up NPU resources and memory bandwidth for the GEMM computations, as shown in Fig. 2. Table I shows the difference between the previous works on communication offload in datacenter networks vs. ACE that is also visualized in Fig. 1. To the best of our knowledge, ***this is the first endpoint-based offload scheme that is tuned for distributed training for the AF***.

With ACE, we demonstrate that the compute speeds up and the available AF bandwidth can be utilized much more efficiently, decreasing overall training time.

This paper makes the following contributions:

- We identify a set of key challenges in the end-points of DL training platforms related to compute and memory bandwidth that can limit the utilization of available AF bandwidth for future training platforms (Section III).
- We propose a novel microarchitecture called ACE designed to handle collective communication and efficiently enhance AF network utilization (Section IV). ACE frees up the endpoint's compute and memory resources for DL compute, which in turn reduces the required memory BW by 3.5× on average to drive the same network BW compared to state-of-the-art baselines. On average, ACE increases the effective network bandwidth utilization by 1.44× (up to 2.67×), resulting in average of 1.41× (up to 1.51×), 1.12×

[2] We assume synchronous updates which provide better guarantees at convergence than asynchronous updates.

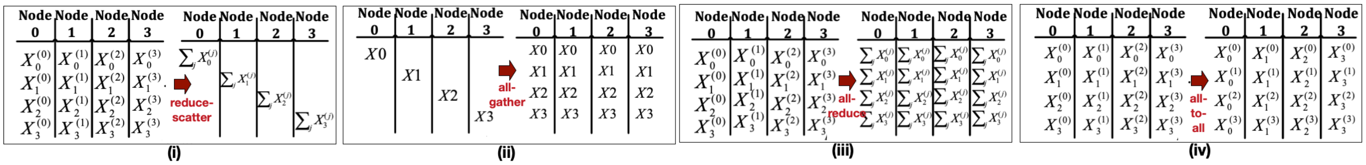


Fig. 3: Overview of collective communication operations used in DNN training networks.

(up to 1.17×), and 1.13× (up to 1.19×) speedup in iteration time for ResNet-50, GNMT and DLRM when compared to the best baseline configuration, respectively.

- Finally, we show that the reduction in the memory BW requirement (enabled by ACE) allows for performing better workload-specific optimizations, with the specific example of optimizing DLRM.

The rest of the paper is organized as follows: Section II presents the necessary background for distributed training systems. Section III establishes the challenges in scaling DL training, specifically focused on critical bottlenecks in the endpoint that inhibit efficient network utilization. Section IV describes the ACE microarchitecture. This is followed by a detailed description of our evaluation and simulation methodology in Section V and the experimental results in Section VI. Next, we compare our work against the related works in Section VII. Finally, we conclude the paper in Section VIII.

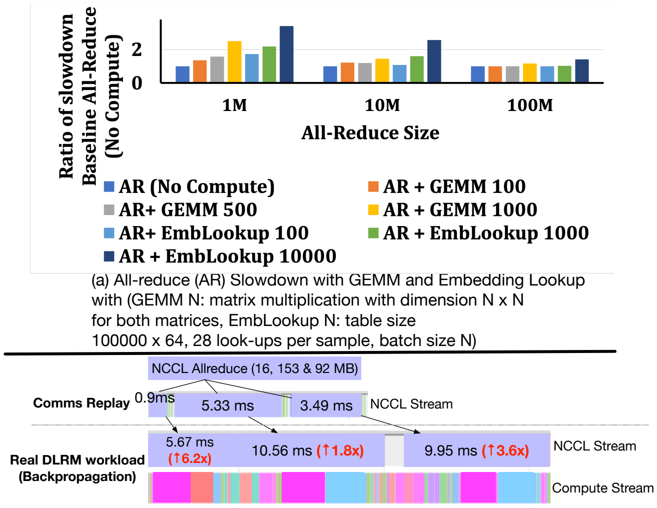
II. Background

Training DNNs involves iteratively refining the parameters (aka weights) of the network by solving a non-convex, non-linear optimization problem to minimize a loss function. Here, we provide background on distributed training [15], [44].

Parallelization. The most common parallelization technique for speeding up DL training is called *data parallelism*. It replicates the entire model on multiple nodes to take advantage of the large number of input samples. Every node computes partially trained weight gradients for its subset of samples of the input data, aka mini-batch. At the end of each iteration, nodes exchange their partially trained weight gradients and perform the SGD operation to update the weights gradients accumulated from all nodes. The updated weights, are then used in the forward pass of the next iteration. In *model parallelism*, all nodes have the same datasets and work on the same mini-batch, but the model is divided among nodes. Each node thus produces a part of the output activations and input gradients during the forward pass and back-propagation, respectively, and these values must be communicated across all nodes to enable forward pass and back-propagation.

Collective Communication Operations. Exchange of input/weight gradients and output activations among the nodes, depending on the parallelism approach, is known as "collective communication". In general, four different collective communication operations are the main contributor in DNN training communication [29], [48], as shown in Fig. 3: (i) reduce-scatter, (ii) all-gather, (iii) all-reduce, and (iv) all-to-all. Reduce-scatter reduces (e.g. sum) all data, initially residing in the nodes, such that at the end each node has a portion globally reduced data. All-gather gathers the data, initially scattered across nodes, such that at the end, all of the nodes have all of the data. All-reduce can be thought of as a reduce-scatter followed by an all-gather. In all-to-all, each node needs to send a different portion of data to other nodes. All-reduce is the dominant communication pattern observed in the DL training for exchanging gradients and activations in various parallelism schemes. However, all-to-all is used in some scenarios such as table embedding exchanges for recommendation models such as Facebook DLRM [41].

Topology-Aware Collective Algorithm. Collectives have efficient implementation algorithms based on the underlying topology. Libraries like Intel oneCCL [7] and NVIDIA's NCCL [42] provide different implementations for collectives, such as ring-based, tree-based, hierarchical, direct all-reduce/all-to-all, to optimize for the available bandwidth in the underlying topology [13], [45], [55]. We will discuss this further in Section V, where we consider topology-aware collectives for evaluating our target systems.



(a) All-reduce (AR) Slowdown with GEMM and Embedding Lookup with (GEMM N: matrix multiplication with dimension N x N for both matrices, EmbLookup N: table size 100000 x 64, 28 look-ups per sample, batch size N)

(b) Impact of compute-comms overlap on a real-world production-class DLRM workload

Fig. 4: Slowdown of the communication when overlapped with computation. The platform is NVIDIA V100 8-GPU system connected with NVSwitch offering 150 GB/s available network BW per GPU.

III. Motivation: Poor AF BW Utilization

In this section, we highlight some of the critical challenges in achieving high network bandwidth utilization for DL training workloads. Fig. 2 qualitatively summarizes our findings.

NPU Compute Availability: On modern accelerator systems, a fraction of NPU cores (e.g. CUDA cores) *concurrently* execute kernels pertaining to DL collective communication while the majority of compute cores execute DL computation

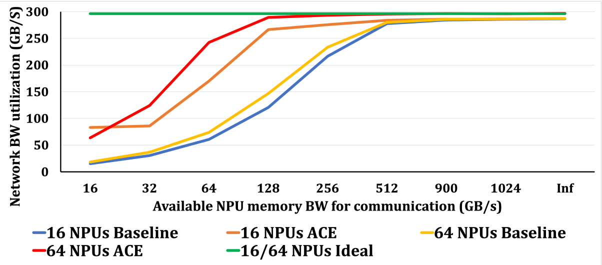


Fig. 5: The sensitivity analysis that shows how the AF network utilization is affected as the amount of NPU memory BW available for DL communication is increased. The communication is a single 64MB all-reduce. Ideal system assumes that received data (during collective communication) is magically processed and ready only after 1 cycle and is used to get and upper bound for network performance. The baseline system assumes all SMs are available for the communication task, while ACE does not consume any SM within the NPU.

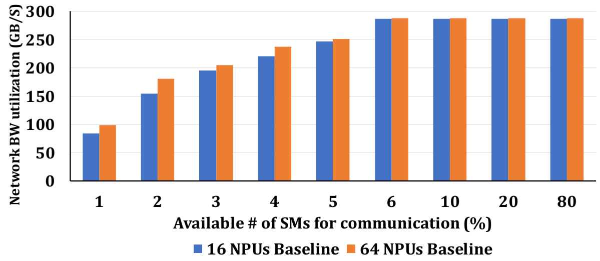


Fig. 6: The sensitivity analysis that shows how the network utilization is affected in the baseline system, as the available # of SMs for DL communication changes. The communication is a single 64MB all-reduce. It is assumed that all NPU memory BW is available for the communication. Note that ACE does not rely on NPU SMs for communication and hence, this experiment is not applicable for it.

(e.g. GEMMs and Convolutions). Collective operations, such as all-reduce, are parallelized across these cores since a single NPU core cannot fully saturate memory bandwidth and network bandwidth, given various bottlenecks in the NPU core - memory - network data-path [23], [42]. Each core iterates over a sequence of send/recv operations to/from peers followed by an optional computation on data that was received with locally available data (e.g. reduction sum). Thus, the optimal number of NPU cores is a function of network bandwidth saturated by a single core, memory bandwidth available to a single core, communication algorithm, message size, and so on.

Memory Bandwidth Availability: Collective communication operations require access to sufficiently high memory bandwidth for performing local reduction sum (streaming operation involving two local memory reads, sum operation, and local write) and remote network write (e.g. RDMA write and synchronization) operations. The memory bandwidth requirements is a function of the actual collective operation and is proportional to network bandwidth.

Real System Measurements and Analysis. We highlight the resource contention issue between communication and computation by creating a microbenchmark workload and also including the results of real training workloads for DLRM model presented in Fig. 4 and Megatron-LM model.

The microbenchmark executes a compute operation, followed by posting a communication operation, then repeating the compute operation, and finally waiting for communication to complete. We consider two fundamental compute operations common in recommendation models [41]: matrix multiplication (GEMM) which consumes GPU compute cores and embedding table lookup (EmbLookup) which mainly consumes GPU memory bandwidth. Fig. 4.a presents the slowdown of NCCL all-reduce collective on an NVIDIA V100 system with 8 GPUs interconnected via NVSwitch, with a total of 150 GB/s available network BW. The slowdown is proportional to the scale of compute operation and all-reduce with a smaller size is more sensitive. Running 100MB all-reduce concurrently with dimension-1000 GEMM (requires 44.8 warps per Stream Multiprocessor (SM)), which is a typical scenario in training recommendation models, will slow down all-reduce by 1.16x. Overlapping 100 MB all-reduce with EmbLookup with batch size 10000 (uses 429.2 GB/s memory bandwidth) slows it down by 1.42x.

To further highlight the performance degradation, we present the slowdown of NCCL all-reduce operations when run concurrently with compute kernels, such as GEMM and EmbLookup, in a real-world PyTorch DLRM workload with batch size 512 [41] compared to the reproduction of the same communication patterns without computation using PARAM replay benchmark [18]. As shown in Fig. 4.b (run on the same system as the above benchmark), we observe the time of 16 MB all-reduce can be increased from 0.9*ms* (without any overlap) to 5.67*ms* (overlapped with GEMM and embedding lookup kernels), up to 6.2x degradation during the backward propagation. Note that such degradation is consistent for all-reduce operations with different sizes as shown in Fig. 4.b and across different epochs.

While PyTorch uses separate CUDA streams for compute and NCCL operations, the actual overlap of these operations changes dynamically based on the CUDA HW scheduler and pending operations. This negative impact of congestion for compute/memory resources by competing kernels results in poor network BW utilization and variability in execution time.

We also evaluated Megatron-LM [54] Workload on our platform (not shown in Fig. 4) and gathered communication time in two scenarios: (i) When communication is overlapped with compute, and (ii) When all communications are explicitly issued at the end of back-propagation when computation is finished. This is the no-overlap scenario where the training algorithm needs to make a blocking wait until communication is done. On average, overlapping communication with computation degrades the communication performance by ≈1.4× compared to the non-overlap scenario, further pointing to the problem of compute-communication resource contention.

Simulation Results. To demonstrate that the network BW drive problem is exacerbated as network BW increases in new systems with higher available network BW (i.e. 500 GB/s), we conducted a simple simulation experiment to show how communication performance is affected as the available memory BW (Fig. 5) or NPU cores (Fig. 6) available for DL communication tasks are varied. We highlight that in this paper we assumed GPU-like NPUs that consist of Streaming

Processor (SM) units as described in Section V.

Fig. 5 shows the results for 16 and 64 NPUs systems running a single 64 MB all-reduce. The other system setup is the same as the systems described in Section V. As can be seen, the ideal system can reach up to 300 GB/s BW utilization, out of available 500 GB/s, since intra-package links (i.e. silicon interposer) become underutilized due to their imbalance speed compared to inter-package links. Fig. 5 also shows that the baseline system needs ≈ 450 GB/s memory BW on average to reach 90% of ideal network BW. The reason is that on average, for each message to send out, there are multiple reads/writes issued to move the data and do the local reduction in the baseline system. However, for ACE, 128GB/s is enough to reach to the 90% of ideal BW utilization, resulting in ≈3.5× reduction in memory BW requirement to reach the same performance. The reason for such improvement is further elaborated in Section VI-A.

Fig. 6 shows how many Streaming multiprocessors (SMs) are required to prevent the GPU-like compute cores from being the bottleneck in driving the networks. SMs are used to read data from the main memory and inject it into the network [42]. For the frequency of 1245 MHz and read/write BW of 64-bytes/cycle, the memory BW is ≈ 80 GB/s per SM (please see Section V for more information). Then, We use Fig. 5 to calculate how much network BW can be driven as we increase the number of SMs for the communication. For our simulated platforms, 6 SMs are enough to reach to the 450 GB/s memory BW. This is in line with the percentage of core usage for libraries such as oneCCL [7] and NCCL [42].

Takeaway. Overall, we argue that using NPU cores (e.g., CUDA cores) for executing collective communication increases contention for precious compute cores and memory bandwidth on the accelerator node. The complex interplay between compute-memory-network not only affects collective performance but also the performance of compute kernels executing concurrently. This in turn increases the effective compute time available for overlapping the communication operations - hiding deep inefficiencies. These challenges are expected to get exacerbated in future training platforms, where better compute accelerators and hierarchical bandwidths (e.g., emerging MCM technologies [8], [14]) will make it harder to hide communication behind compute.

TABLE II: Trade-off between offload at switch vs. endpoint.

Offload Scheme	Increase Available Memory BW & Compute for Training	Endpoint Congestion Reduction	Switch Based Topology Compatibility	Point-to-Point Topology Compatibility	Hybrid Topology Compatibility	Various Collective Algorithm Support
Switch Based	✔	✔	✔		Partially	
Endpoint Based	✔	✔	✔	✔	✔	✔

IV. Accelerator Collectives Engine

Driven by the compute and memory contention issues identified by Section III, this work makes a case for a dedicated accelerator for DL collective communication called ACE, sitting alongside the Accelerator Fabric Interface (AFI) module introduced earlier in Fig. 1. We present design details next.

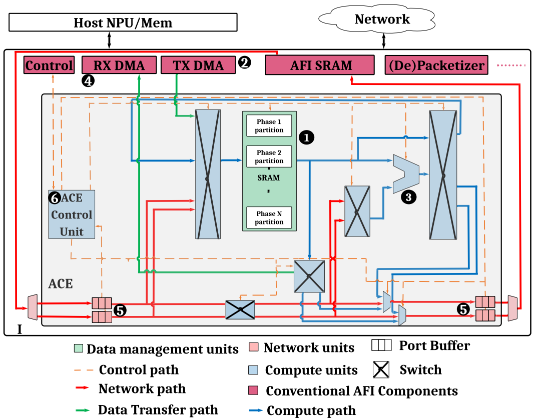


Fig. 7: ACE microarchitecture. #1 is the on-chip SRAM. #2 is the AFI TX DMA for transferring data from main memory to AFI SRAM (normal operation) or ACE SRAM (ACE activated). #3 is the ALU. #4 is the AFI RX DMA for transferring data from AFI SRAM (normal operation mode) or ACE SRAM (ACE activated mode) to main memory. #5 are the input/output port buffers. These buffers are allocated per each physical link and contain packets corresponding for that specific link. #6 is the control unit logic.

TABLE III: Data granularity at different levels of ACE execution.

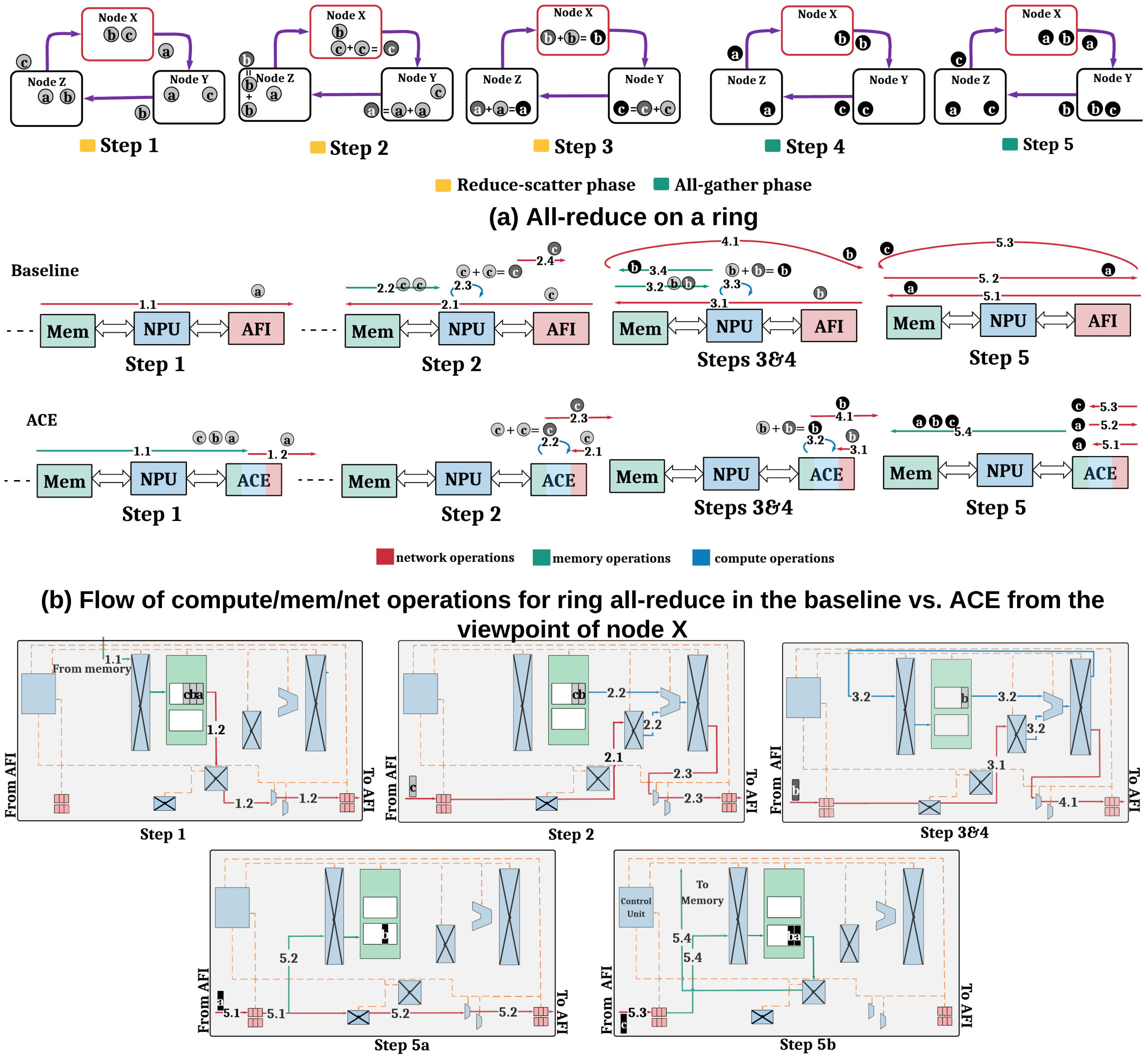
Granularity	Size	Constraint
Payload (variable)	Training Algorithm	Training Algorithm
Chunk (64kB initially)	Parameter for Pipelining	Storage Element Size (Area/Power)
Message (4kB)	Parameter - Multiple of Number of Nodes	Topology
Packet (256B)	Link Technology	Technology
Flit (256B)	Network Buffer Size	Microarchitecture (Area/Power)
Phit (variable)	Link Width	Technology

A. System Overview

Fig. 7 shows the high-level overview of ACE integrated into the AFI module. The key components within the ACE are additional compute units (for all-reduce or all-to-all computations) and scratchpad buffers (to cache gradients). Like other domain-specific accelerators [25], the datapath of ACE is specialized for its target workload - namely running collective communication algorithms.

B. Placement: End-point vs Switch

The communication offload solutions are not new. Fundamentally, there are two options for offload: (i) switch-based, and (ii) endpoint-based offload methods. Table II shows the comparison between these two. Both methods reduce the burden on endpoint memory and compute for handling communication-related tasks. However, endpoint-based methods are more flexible in supporting various network topologies, as opposite to switch-based methods that: (i) require a switch in the network topology, and (ii) point-to-point (PTP) links do not benefit from the offload scheme. Note that in AFs, many platforms are designed solely using point-to-point network topologies (e.g. NVIDIA DGX-1 [39], Google TPU [25], Habana [3]. This is in addition to the intra-package networks that are almost always PTP [59]. Additionally, unlike switch-based methods that only



(a) All-reduce on a ring

(b) Flow of compute/mem/net operations for ring all-reduce in the baseline vs. ACE from the viewpoint of node X

(c) Steps for ring all-reduce within ACE for Node X

Fig. 8: Implementation of ring all-reduce on Baseline vs ACE

support a single reduction/broadcast traffic pattern to/from the switches, endpoint-based methods can support many different algorithms (patterns) (e.g. ring, halve-doubling, double-binary-tree, etc.). The above reasons show why we employ ACE as an endpoint offload, sitting alongside the AFI.

C. Data Granularity

Table III shows the granularity of data at different levels of the ACE execution and their determining factor. It also shows the default value of each level used in ACE. ACE initiates execution by receiving a command from NPU to perform a specific collective on a *payload*. The payload could be activations or gradients depending on the parallelism approach and forward/back pass. The command includes the collective type and the address range for data residing in the main memory. ACE then divides the payload into multiple *chunks* and begins processing and scheduling of each chunk individually and in a pipelined manner. Multiple chunks can be scheduled in parallel.

A chunk itself decomposes into multiple *messages* and the collective algorithm runs at message granularity. The number of messages is a multiple of the number of nodes in the system. For example, if ACE wants to perform an all-reduce in a ring with 4 NPUs, it can divide the chunk into 8 messages, and execute all-reduce serially over two iterations[3].

Each message comprises one or more *packets* when it enters the network layer. The unit of data processing within the ACE is packets. The bus width and SRAM interface might or might not be equal to the size of the packets and data movement/execution is serialized if the size is smaller than packet width.

D. Walk-Through Example for All-Reduce

We describe ACE in action via a detailed walk-through example for running the ring-based all-reduce collective over

[3] Each step leads to processing & performing all-reduce for a group of 4 messages and the algorithm is ring-based. More details on ring-based all-reduce is provided in Section IV-D.

a ring for both baseline and ACE as shown in Fig. 8. The general concepts and the main advantages of ACE compared to the baseline are applicable to any topology/collective, as we describe later in Section IV-H.

Fig. 8.a shows the logical flow of the algorithm across the different nodes. We assume one chunk for simplicity. Since there are three nodes, there are three messages[4]. An all-reduce can be implemented as a reduce-scatter followed by an all-gather, as can be seen from Fig. 3. Steps 1-3 show the reduce-scatter part. Step 1 initiates the reduce-scatter; each node sends one message to its neighbor and waits for receiving another message from its other neighbor. In step 2, each node reduces the received message with its local one and forwards it to the next. Step 3 concludes reduce-scatter by each node reducing the last message it has received. All-gather starts with each node forwarding a copy of its reduced message to its neighbor (step 4) and then each node keeping a copy of its received message and forwarding it (steps 5, 6).

Fig. 8.b shows this flow from node X's view in the case of baseline vs. ACE. It is clear from this figure that in baseline, in all phases, messages need to go all the way from/to main memory to/from AFI to be injected/received into/from the network. The local reduction in baseline begins by loading the local message and the received message into the NPU cache (e.g. sub-step 2.2), performing reduction (e.g. sub-step 2.3), and sending the result out to the network (e.g. sub-step 2.4) or main memory (e.g. sub-step 3.4), depending on the step of the algorithm. This in turn reduces the available memory bandwidth and compute resources for the DL GEMMs. In contrast, ACE restricts the data movement only to the first and last phases (reduced congestion and increased available memory bandwidth) and allows the DL GEMMs to make use of complete NPU compute resources.

Fig. 8.c shows the internal ACE interactions for node X. Here, the ACE SRAM is divided into two partitions - one serves as a source for the (only one) all-reduce phase, and the last one serves as the source to hold the final results to send back to main memory. In step 1, the 3 messages are brought into the first partition of ACE SRAM by the TX DMA (sub-step 1.1). Then, one message is sent out through a series of packets[5] injected into the designated output port buffer to be injected into the network (sub-step 1.2). In step 2, the received message is reduced with the local data (sub-step 2.2) and forwarded to the neighbor (sub-step 2.3). ACE overlaps steps 3 and 4 of the algorithm; after performing a reduction (sub-step 3.2), stores it locally (sub-step 3.2) and forwards it to the next neighbor (sub-step 4.1) at the same time. Step 5 is broken into two figures for more clarity. In step 5a, the received message is stored and forwarded (sub-step 5.2) at the same time, while in step 5b, the final received message is stored and the whole chunk is sent back to memory by RX DMA (sub-step 5.4).

It is clear that in some steps within ACE, multiple resources should be available for some sub-steps to proceed. For example, in sub-step 5.2 in step 5a, both the SRAM input port should be available and the output port should have free space. In such cases, if any of the resources are not free, that step is stalled until all resources are available. Multiple chunks can be executed in parallel to maximize internal hardware resources and link bandwidth, as we discuss next.

E. Parallelism

To achieve high network utilization, we need to apply parallelism at various levels. From the algorithmic perspective, there are several levels where parallelism is possible. More complex hierarchical topologies implement topology-aware collectives over multiple phases [48]. Assuming the collective algorithm has P phases, multiple chunks can run in parallel both within a phase and across different phases. Each chunk will send/receive multiple messages. Hence, multiple in-flight chunks mean multiple in-flight messages (belonging to different chunks) are possible. Parallel chunks mean multiple packets can be processed in parallel. Packets are the unit of data transfer within the network and parallelism below that is the network's job. So the ACE memory management and control units are designed to ensure using all algorithmic parallelism opportunities. The SRAM within ACE is partitioned according to the number of phases of the collective algorithm being run plus one for holding the final results for RX DMA, called the *terminal partition*. For the example in Section IV-D, the ring-based all-reduce has one phase and hence needs two partitions. More details on multi-phase all-reduce are mentioned in Section V.

F. Control Unit

The control unit comprises multiple programmable finite state machines (FSMs). Each FSM can be programmed for a specific phase of a specific collective algorithm and holds a queue of chunks that should process in order. Each entry of this queue holds the *context* of a chunk like its start and end address inside the SRAM and the address range for holding the final result for the next phase. When a chunk is created in ACE, it is also assigned the state machines it should go through for each phase.[6]. When entering each phase, the chunk is inserted into the queue of its assigned state machine for that phase. The state machines then compete with each other to access different resources, resulting in overlapping and out-of-order execution of multiple chunks both within[7] and across phases. This increases resource utilization and network bandwidth. The available parallelism is only bounded by the number of available state machines to manage the dataflow for each phase.

[4] Note that there could be multiple number of 3 messages and as we described in Section IV-C, they should be executed in serial. But here for simplicity we assume only 1 group of 3 messages.

[5] Note that here packets means packet data. The actual packetization of this data is the job of AFI once it wants to send it over the links.

[6] It is possible that, for a given workload and parallelism (e.g., DLRM in our evaluations), different collective operations (e.g. all-reduce and all-to-all) exist for the same phase. In this case, the FSMs allocated for that phase should be programmed to handle all collective operations for that phase.

[7] If the phase is assigned multiple FSMs.

G. Interface with NPU and AFI

ACE extends the existing AFI interface exposed to NPU as shown in Fig. 7. AFI control forwards the ACE-specific commands from NPU/ACE to ACE/NPU. The NPU-AFI command interface is similar to UCX [52] or OFI [20] which are the standard high-level communication interfaces. Once a collective command is received, ACE decides when to load data given the availability of SRAM space. Finally, ACE notifies the completion of chunk collective by raising an interrupt and forwarding it to NPU.

H. Flexibility and Scalability

Supporting Different Collectives. The general principles for running any collective algorithm using ACE remain the same. For a collective with say P phases, the SRAM is divided into P+1 partitions. Each partition is assigned to a phase and one or multiple FSMs, except the last partition (i.e. terminal partition) that is used for storing the final results to be written back to memory. Different collectives can be implemented by programming the dataflow into the FSM.

Supporting Different Topologies. Sine ACE handles the collectives at the endpoints, it is orthogonal to any network topology (e.g., switch-based, point-to-point, hybrid). From the logical view, ACE can perform any collective algorithm (e.g. ring-based all-reduce) on top of any physical topology. It is then the job of network protocol and routing algorithm to deliver the packets accordingly.

TABLE IV: Synthesis Results

Component	**Area** (μm^2)	**Power** (mW)
ALU	16112	7.552
Control unit	159803	128
4×1MB SRAM banks	5113696	4096
Switch & Interconnect	1084	0.329
ACE (Total)	5339031	4255

I. ACE Design-space Exploration and Implementation

Fig. 9.a shows the ACE performance when SRAM size and number of FSMs, the components with the most overheads, are swept. Fig. 9.a shows that increasing beyond 4MB of SRAM and 16 FSMs results in diminishing returns, and so we chose these as our final parameters[8] since the selected parameters are enough to fill most of the network pipeline. Also, 4 wide ALU units, each capable of preforming 16×FP32 or 32×FP16 in parallel were sufficient for ACE. The interconnect between SRAM and functional units are wide 64B buses. We implemented ACE using Verilog and synthesized the design using the Synopsis Design Compiler in 28nm technology node. Table IV shows the area and power estimates for our design, enumerating individual components as well as ACE itself. Compared to the area and power of high-end training accelerators reported in [25], [57], ACE has less than 2% overhead in both area and power.

In addition, we used a simple heuristic for SRAM partitioning that partitions the SRAM based on the (available network bandwidth × initial chunk size) for each phase[9], with the *terminal partition* (i.e. partition P+1) being equal to the the last phase partition (i.e. partition P).

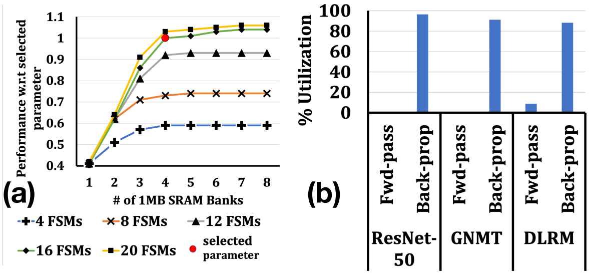


Fig. 9: (a) The performance of ACE with different FSM and SRAM sizes normalized to the performance of selected parameters (i.e. 4MB SRAM, 16 FSMs). The performance is gathered through averaging ACE performance across all target workloads and system sizes described in Section V. (b) The utilization of ACE for the simulations of Fig. 10. ACE is considered utilized when it has assigned at least one chunk for processing.

TABLE V: System parameters

Parameter	**Values**
Compute Accel.	Max of 120 TFLOPs FP16 at GPU-like, with 80 SMs
Bus BW	500 GB/s (NPU-AFI), 900 GB/s (NPU-MEM)
ACE Message size	8KB
Packet size	256 Bytes (Intra & Inter Package)
Per link BW	200 GB/s (Intra), 25 GB/s (Inter)
Link latency	90 cycles (Intra), 500 cycles (Inter)
Total # of links/NPU	2 intra-package links (1 bidirectional ring), 4 inter-package (1 bidirectional horizontal ring, 1 bidirectional vertical ring)
Total BW	400 GB/s (intra-package ring), 50 GB/s (horizontal ring), 50 GB/s (vertical ring)
Link efficiency	94% (Intra & Inter Package)

V. Evaluation Methodology

This section describes our methodology establishing and simulating high-performance training systems and evaluating the benefits of communication acceleration.

Simulator. Table V shows the major system parameters. We used ASTRA-SIM [2], [48], a distributed DNN training simulator and developed ACE on top of that. ASTRA-SIM models the training loop, parallelization, and collective communication scheduling similar to libraries like oneCCL [7]. It interfaces with a compute model [50] for the compute times (i.e. forward-pass, weight gradient, and input gradient) and network simulator [9] for tracking cycle-level network behavior.

Compute Parameters. The compute model (NPU) is a 1245 MHz GPU-like core with 80 Streaming multiprocessors (SMs) that can achieve the max of 120 TFLOPs using FP16 precision. Our compute accelerators is quite similar to the state-of-the-art training accelerators [24].

[8] Only 6% performance improvement is seen for 8MB SRAM and 20 FSMs.

[9] For example, a phase with 2× link bandwidth and 2× initial chunk size has a partition 4× greater than a phase with 1× bandwidth and 1× chunk size. Also note that if for each phase, there are different chunk sizes belonging to different collective operations, we use average of such chunk sizes.

TABLE VI: Target System Configurations

BaselineNoOverlap	State-of-the-art baseline system where there is no overlap between communication and computation. All communications are gathered and explicitly issued at the end of back-propagation phase using a single communication kernel*. This means that all resources are available for compute and communication when they are running. This strategy enhances compute time but puts all communication time on the critical path since training loop can not proceed before the communication is finished.
BaselineCommOpt	State-of-the-art baseline system where enough resources are allocated to make its communication performance reaches 90 % of Ideal case. Using the intuition behind Fig. 5 and Fig. 6, 450 GB/s of memory BW and 6 SMs of the NPU is allocated for the communication task and the rest for the computation.
BaselineCompOpt	State-of-the-art baseline system, but this time training computation performance is optimized. In order to do this and similar to ACE, we assume only 128 GB/s of memory BW is allocated for communication. Since this amount of memory BW is not enough to derive the entrie BW (see figure Fig. 5), then using results of Fig. 6, 2 SMs are enough for the communication task. The rest is for compute.
ACE	Proposed system. In this case, 100% of the NPU resources are dedicated to the training algorithm computation. Moreover, 128 GB/s memory BW is enough to reach to 90% of the ideal (see Fig. 5). The rest of memory BW is allocated for training compute.
Ideal	A system where the endpoint can handle/ process received messages magically within one cycle. This essentially means that there is no associated latency from the endpoint side in the collective communication latency. This gives an upper bound to our design. In this case 100% of compute and memory is allocated for training algorithm only.

*The only exception is DLRM fwd-pass all-to-all where the training loop performs a blocking wait until the communication completes and then starts compute.

AFI and Network Parameters. We extended ASTRA-SIM to model AFI and the interactions (traffic) between AFI, NPU, and Memory. We also modeled the transaction scheduling effects of NPU-AFI and NPU-Mem and queuing delays of subsequent transactions. NPU-Mem bandwidth is chosen based on the state-of-the-art accelerators [24]. NPU-AFI bandwidth is assumed to be the sum of all intra-package/inter-package links as a logical choice to prevent NPU-AFI bandwidth to be the bottleneck in driving the network links. Inter-package link BW is assumed to be the same as NVlink [35], while intra-package bandwidth is selected based on [8], [53] that is an expected bandwidth for high-performance MCM systems.

Target Training Platforms. ACE works for both pt-to-pt and switch-based topologies. In the interest of space, we present the results for a pt-to-pt 3D Torus topology. We model the futuristic platforms with AF comprising of multiple NPUs integrated through multi-chip packaging technology [8], [14] on a package, and multiple packages interconnected via a dedicated fabric. We use the notation LxVxH to describe the system size, where L stands for the number of NPUs within the same package that are connected through an intra-package ring (local dimension). All NPUs with the same offset across different packages then form a separate 2D torus within V rows (vertical dimension) and H columns (horizontal dimension), connected through vertical and horizontal rings, respectively.

Topology-aware Collective Algorithms. We use hierarchical multi-phase collective algorithms for the 3D torus topology, where the ***all-reduce*** occurs in 4 phases: reduce-scatter in local, all-reduce in vertical, all-reduce in horizontal followed by all-gather in local dimension. This implementation uses the higher-bandwidth intra-package local links more than the inter-package links and is provided by the simulator [48]. For the ***all-to-all*** collective, we used the direct all-to-all where each NPU simultaneously sends a distinct portion of data to any other NPU [25], [47]. Since all-to-all is a single-phase, all FSMs are programmed to be able to execute all-to-all, in addition to their assigned all-reduce phase. To perform all-to-all, each FSM places the data to the output link FIFOs based on the destination NPU route. We used XYZ (local dim, vertical dim, horizontal dim) routing for each packet to reach its destination for our torus-based topologies. Note that AF networks like NVLink only support neighbor-to-neighbor communication natively. This means that for the baseline system and for the packets that need to go multiple hops, the communication library (e.g. NCCL) is responsible for writing data to the intermediate hops' memory and again forwarding data to the next hop. This wastes a lot of memory BW on the intermediate hops. But ACE prevents such unnecessary memory overheads since its SRAM absorbs packets and forwards the ones that have different destinations through the FSM responsible for the corresponding chunk.

Target System Configurations. We consider five different system configurations, discussed in Table VI. We investigate three flavors of the baseline systems: two of them with comp/comm overlap either optimized for communication (BaselineCommOpt) or computation (BaselineCommOpt), and one other baseline where there is no overlap between communication and compute (BaselineNoOverlap). In addition to ACE, we also present the Ideal system results to get the upper bound if we had maximum compute/communication performance.

Target Workloads. In order to evaluate our platforms, we consider three different sets of real workloads: (1) ResNet-50 [21] from vision, GNMT [58] from NLP, and DLRM [41] from recommendation DNNs. We use the FP16 precision for activation/gradient computation and communication. We consider data-parallel parallelism (i.e. requiring all-reduce for weight gradients) for ResNet-50 and GNMT and hybrid parallel (data-parallel across MLP layers, model parallel across embedding tables) for DLRM. For DLRM, we use the version described in [47] which is a representative model used in real systems. We simulate two training iterations with Last-In-First-Out (LIFO) collective scheduling policy to give more priority to the collectives of first layers during back-propagation. The mini-batch size is set to be 32, 128, and 512 per NPU for ResNet-50, GNMT, and DLRM, respectively. We present both end-to-end runtime and breakdown of compute-comms overlap. We assume weak scaling as we scale the number of NPUs in our evaluations.

Metric of Evaluation. For real workloads (Section VI-B, Section VI-C, and Section VI-D), our metrics are *total computation* and *exposed communication*. Exposed communication refers to the time where communication cannot be overlapped with computation and hence, the training algorithm (compute) is forced to stop because it is waiting for the communication to be finished. This case for data-parallel is during forward pass, where for each layer we need to make sure the weight gradient communication (i.e. all-reduce) of the previous iteration is completed and weights are updated. For DLRM, we additionally need to wait for the embedding communication operation (i.e. all-to-all) before entering the top MLP layers in forward pass, and after finishing of the back-propagation to update the embedding [41]. The summation of the (total computation + exposed communication) then determines the training iteration time.

VI. Evaluation Results

This section presents the simulation results comparing ACE against *Ideal* and the baseline systems mentioned in Table VI for real workloads. But first, we do an analytical investigation about the memory BW requirement for ACE vs. Baseline, that justifies the simulation results we observed in Fig. 5.

A. Memory BW Requirements for Baseline vs. ACE

According to Fig. 8, in the baseline system, the number of memory reads is 2N bytes for sending every N bytes to the network in the reduce-scatter phase (excluding the initial phase). N memory bytes need to be read to send N bytes to the network for all-gather phase. Since the number of bytes to be sent out in these 2 parts is identical, this means that on average, 1.5N bytes need to be read from memory to send out N bytes. This explains there is a need for ≈450 GB/s memory BW to drive 300GB/s of the network, in the ideal case.

However, ACE caches the data and reuses it multiple times. The amount of reuse depends on the topology and algorithm. Let's consider the 64-NPU (4X4X4) system, similar to Fig. 5. For every N bytes of data that is cached in ACE, $\frac{3}{4}$N is sent out during the first phase of reduce-scatter, $2\times\frac{6}{16}$N is sent out one for vertical and one for horizontal all-reduce phases. Finally, another $\frac{3}{4}$N is sent out for the final all-gather phase, resulting in a total of 2.25N data sent to the network (133 GB/s memory BW to drive 300 GB/s in the ideal case). However, in reality, more memory BW is needed due to hardware delays, queuing delays on the buses, etc., as we showed in Fig. 5. We limit our analytical study to all-reduce collective due to the limitation of space, but a similar analysis can be done for all-to-all as well. However, compared to the all-reduce, all-to-all is used much less frequently and their sizes are usually smaller.

B. Compute-Communication Overlap

Here, we evaluate the four systems with compute-communication overlap mentioned in Table VI for **two** training iterations on ResNet-50, GNMT, and DLRM. The goal is to dissect the compute-communication overlap of different configurations for a single 128 NPUs AF network size. In Section VI-C, we show the general trend for different network sizes and include BaselineNoOveralap configuration as well.

ResNet-50 Analysis. The first row of Fig. 10 shows the NPU compute and network utilization for ResNet-50 training.

BaselineCommOpt. First, we focus on the BaselineCommOpt (Fig. 10.a). Here, we observe that: i) despite being optimized for communication, network links are still not yet completely utilized, and ii) network utilization also fluctuates at different times. This is because of three main reasons. First, in real workloads there are some computation operations between two consecutive communication, allowing for partially/completely finishing current communication before the next communication task arrives. Second, high BW of the intra-package links makes them underutilized in the collective algorithm. Additionally, Resnet-50 issues many small-size collectives that make it not sufficient to completely drive the available network BW[10]. Thus, exposed communication still consists of 31.5% of iteration time.

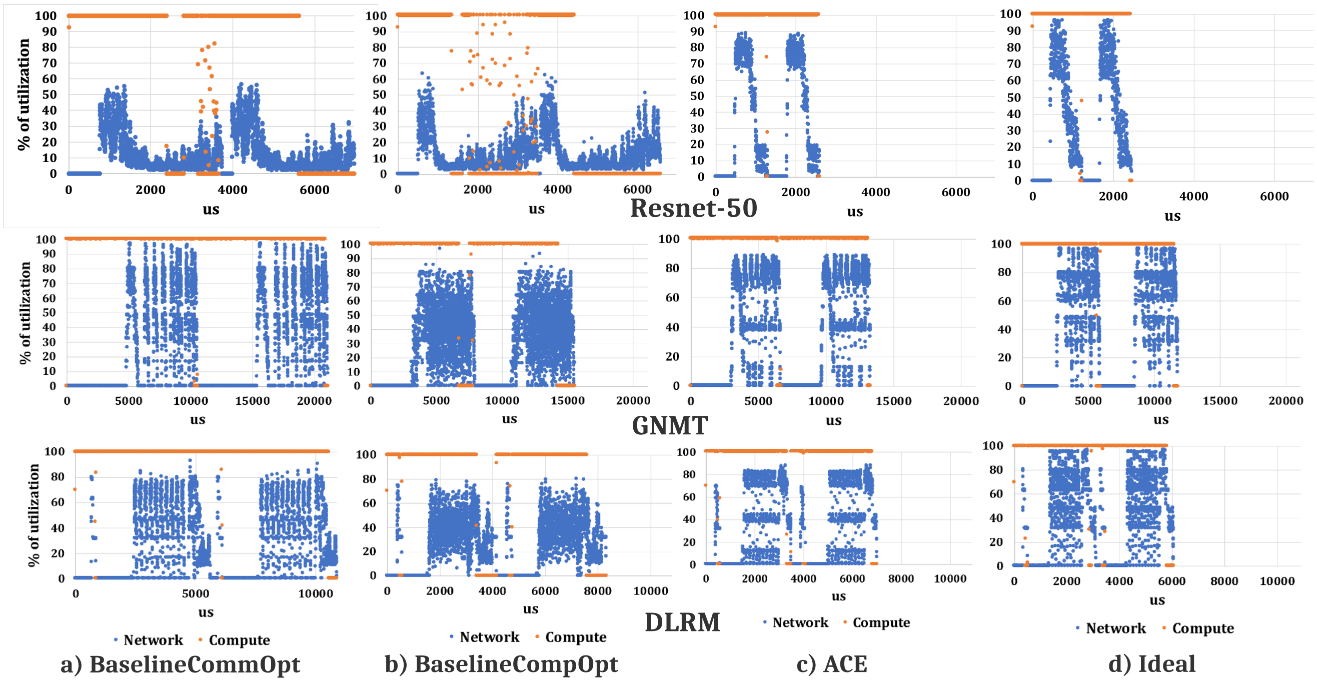
BaselineCompOpt. Next, we dive into the second flavor of the baseline that is optimized for compute. Compared to BaselineCommOpt, BaselineCompOpt reduces the total computation time by 1.75× but it fails to significantly reduce iteration time since exposed communication increases due to the poor utilization of the network. Note that in Fig. 10 the network utilization of BaselineCommOpt and BaselineCompOpt seems similar. This is because due to the much shorter intervals between the communication tasks in BaselineCompOpt, as a result of the reduced computation, that allows it to schedule larger size collectives at a given time. The overall iteration time is improved only by 1.06×, compared to the BaselineCommOpt.

We note that this result does not mean that our baseline system has poor performance. In fact, as Fig. 10.b shows (and we summarize later in Fig. 11) one training iteration duration takes ∼ 3.5 ms for the 128-node baseline system. This means that one epoch of training on the ImageNet dataset [16] takes ∼ 12 seconds that is comparable to the best training performance-per-node times reported [38]. However, we will show that releasing more resources for compute via ACE leads to even better performance.

ACE. ACE can achieve the best of both worlds by releasing most of the resources for compute and still being efficient through caching the data and performing the collective. Compared to the BaselineCompOpt, it can slightly improve compute time by 1.06× because of the releasing all compute cores for compute. However, it still can handle the communication efficiently and exposed communication is only 2% of the iteration time. ACE outperforms BaselineCommOpt and BaselineCommOpt by 2.67× and 2.52× in terms of iteration time, respectively. Overall, ACE can achieve 94.3% of ideal system performance, while this value is 35.3% and 37.3% for BaselineCommOpt and BaselineCompOpt, respectively.

GNMT Analysis. The second row of Fig. 10 shows the compute and communication utilization for GNMT. In GNMT, communication sizes (per layer) are larger allowing BaselineCommOpt to drive the network efficiently. Also, larger compute time means more room to overlap communication with computation. However, BaselineCommOpt still suffers from the

[10] Note that because of these reasons, even in the Ideal case the network utilization fluctuates.



a) BaselineCommOpt b) BaselineCompOpt c) ACE d) Ideal

Fig. 10: Compute-Communication overlap for 2 training iterations of ResNet-50,GNMT and DLRM networks running on a 4x8x4-node 3D Torus. Each point reports the average compute/network utilization over 1K cycles (500 cycles before, and 500 cycles after) period of training. There are 2 bursts of network activity corresponding to 2 iterations. **Note that here, % of network utilization corresponds to the % of links that utilized for scheduling a flit in a cycle, irrespective of their different link BW.**

long compute time especially since GNMT compute is more sensitive to available memory BW. ACE can outperform both BaselineCommOpt and BaselineCompOpt by 1.59× and 1.17×, respectively. Additionally, it achieves 88.7% of the ideal system, compared to the BaselineCommOpt and BaselineCompOpt with 55.9% and 76%.

DLRM Analysis. The third row of Fig. 10 corresponds to the DLRM results. Again, here the communication size is relatively large. ACE iteration time is shorter by 1.55× and 1.19× compared to BaselineCommOpt and BaselineCompOpt, respectively. BaselineCommOpt, BaselineCompOpt, and ACE achieves 55.6%, 72.8%, and 86.5% when compared to the ideal system iteration time, respectively.

ACE Utilization. Fig. 9.b shows the average utilization of ACE. As can be seen in Fig. 9.b, for forward-pass, ResNet-50 and GNMT have zero communication while DLRM has only a single all-to-all communication, resulting in low utilization of ACE. However, for back-propagation, the average utilization for ResNet-50, GNMT, and DLRM is 96.4%, 91.3%, and 88.3%, respectively. The reason for not full utilization is the presence of compute between communication task and high-performance of ACE that results in some idle times before next communication arrives.

C. Scalability and Network Utilization

Fig. 11.a shows the total training loop algorithm latency, decomposed into total compute and exposed communication, for the three different workloads, running on different torus network sizes. As Fig. 11.a shows, the exposed communication delay increases with network size, due to the increased overhead of communication since it requires more steps to be finished. It also shows that among the two baselines with overlap, baselineCompOpt always outperforms BaselineCommOpt The reason is that any saving in compute time directly translates to the reduction in iteration time, while communication may be overlapped with compute and does not impact total training time. However, as we showed in this paper, exposed communication, due to the poor network utilization, can limit the training performance, if left unaddressed. Another interesting point is the BaselineNoOverlap. Compared to the BaselineCommOpt, BaselineNoOverlap is always better thanks to the huge savings in compute time. When comparing to BaselineCompOpt, it works worse for all workloads except for ResNet-50 running on systems larger than 16 NPUs. The reason is combining all small-size communication of ResNet-50 and running them together results in better network performance than running small collectives individually. In all cases, ACE outperforms all baselines. According to this figure and when averaging across different sizes and workloads, BaselineNoOverlap, BaselineCommOpt, BaselineCompOpt and ACE achieves the 68.5%, 49.9%, 75.7%, and 91% of the ideal system performance (in terms of iteration time), respectively.

Fig. 11.b shows the performance improvement of ACE over all baselines as the network size scales. According to Fig. 11.b, in general, the performance gap of ACE over the baselines increases by system size. But this increase is more evident in BaselineNoOverlap and BaselineCompOpt since: i) these configurations are more vulnerable to the increased

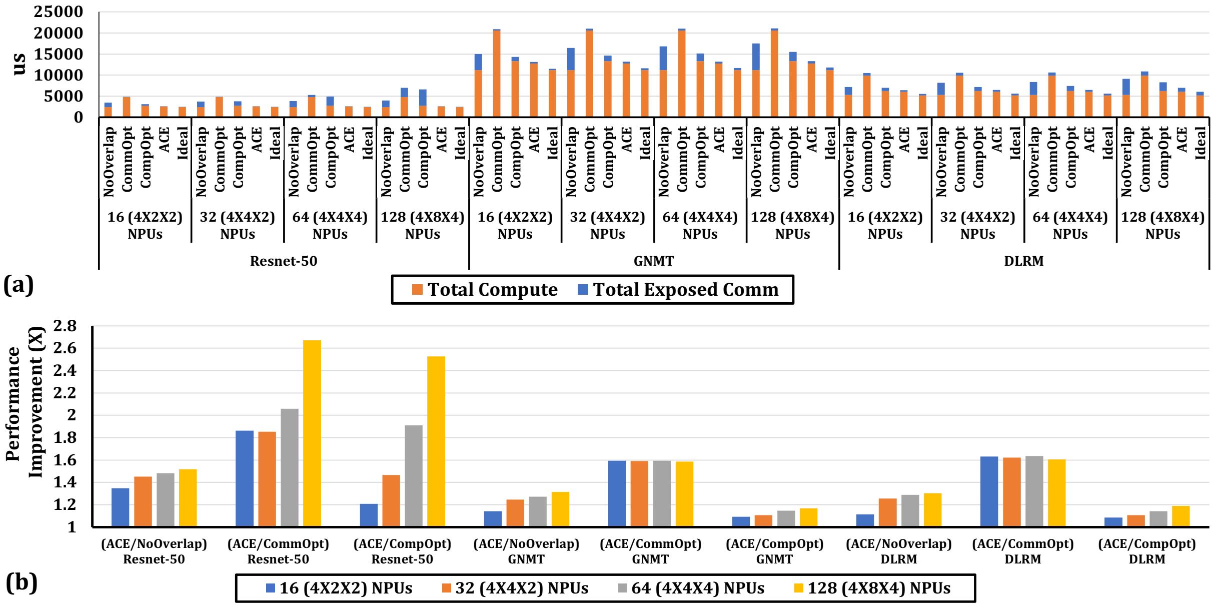


Fig. 11: a) Total computation time vs. total exposed communication (for 2 training iterations) time as the number of NPUs in the AF network increases. b) The corresponding performance of ACE over BaselineNoOverlap, BaselineCommOpt and BaselineCompOpt.

communication overhead, ii) the long compute latency of BaselineCommOpt allows to tolerate more communication overhead for a larger system. Moreover, different workloads get different benefits from ACE as the system scales since the exposed communication performance is also dependent on the interplay between communication size, intervals between communication tasks, and the compute times to overlap communication with. On average and across different network sizes, when compared to the best baseline at any configuration, ACE achieves 1.41× (up to 1.51×), 1.12× (up to 1.17×), and 1.13× (up to 1.19×) speedup in iteration time for ResNet-50, GNMT, and DLRM, respectively. Fig. 11.b also preserves the ratio of the effective network BW utilization (in terms of GB/s) between ACE and different baselines, since for each specific network size/workload, different configurations inject the same amount of traffic to the network. When averaging across all points in Fig. 11.b, ACE can improve the effective network utilization by 1.44× (up to 2.67) over all baselines.

D. DLRM Optimization

The extra available memory bandwidth enabled by ACE offers an opportunity to perform various workload level optimizations to further increase the training performance. Here we pick DLRM as one example for such optimization. It is possible to use extra memory BW to overlap the (memory intensive) embedding lookup/update of the next/previous iteration with the computation of the current iteration. This is because embedding indices are barely reused in the consecutive training iteration [41]. This way, the embedding operation goes outside of the critical path in the training loop. To demonstrate this, we performed a simple experiment where we allocate one SM and 80 GB/s available memory BW for embedding update/lookup of the previous/next iteration. For the next iteration embedding lookup, we immediately issue communication once the lookup is finished and don't wait until the next iteration. Fig. 12 shows the impact of such optimization on the baseline vs. ACE. The total computation time is decreased in both systems since the training loop does not wait for embedding update/lookup at the end/beginning of each iteration. However, BaselineCompOpt benefits little as a result of such optimization due to its poor communication performance. In this case, BaselineCompOpt and ACE achieve 1.05× and 1.2× performance improvement compared to their default training loop, respectively.

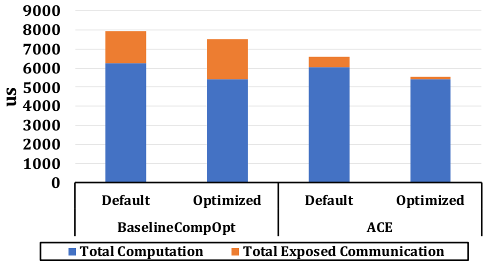


Fig. 12: Impact optimized training loop for the baseline vs. ACE

VII. Related Work

ACE is the first work proposing a dedicated hardware engine at the endpoint for running DL collectives within the Accelerator Fabric. The works closest in flavor to ACE are those on collective offload to datacenter NICs and switches. We contrast them in Table I and discuss some key works here.

Collective Offload for DL. Switch-based offload solutions, such as Intel's Barefoot [4], Mellanox SHARP [6], and NVIDIA's shared memory switch offload [29], have proposed aggregation in switches. Switch offload has also been explored for accelerating Reinforcement learning [36]. Unfortunately, as discussed earlier in Section IV-B, these solutions are restricted to switch-based topologies (limiting their applicability to NVswitch based NVIDIA platforms today). In contrast, ACE can work with both switch-based and point-to-point topologies that are highly popular (e.g., [1], [3], [26]) in most training platforms today.

Collective Offload in HPC Systems. HPC systems with Torus-based topology, such as BlueGene [28], PERCS [46], and Anton2 [19], have supported collective offload on network routers. However, these approaches are designed for CPUs and communicate via message passing rather than accelerators like TPUs/GPUs communicating via shared-memory fabrics.

VIII. Conclusions

In this paper, we identified the issues with compute and memory bandwidth sharing at DL accelerator endpoints in modern distributed training platforms that limit compute-communication overlap. We made a case for optimizing the endpoint with a novel collectives accelerator called ACE. We demonstrated that ACE can efficiently drive the hierarchical AF fabric to close to its peak bandwidth and free up critical compute and memory resources for DL computations, speeding up end-to-end training. On average, ACE frees up the endpoint's required memory BW by 3.5× to drive the same network BW compared to state-of-the-art baselines. For modern DL workloads and different network sizes, ACE, on average, increases the effective network bandwidth utilization by 1.44× (up to 2.67×), resulting in an average of 1.41× (up to 1.51×), 1.12× (up to 1.17×), and 1.13× (up to 1.19×) speedup in iteration time for ResNet-50, GNMT and DLRM when compared to the best baseline configuration, respectively. This work opens up future research in optimizing compute-memory-network interactions during distributed training.

IX. Acknowledgment

This work was supported by awards from Facebook and Intel. We thank Ching-Hsiang Chu from Facebook for his help with collecting real system measurements in Section III. We also thank Anand Samajdar, from Georgia Tech, for his help with modifying the compute-simulator and getting the compute times for the simulations in Section VI.

References

[1] "Analyzing Intel's Discrete Xe-HPC Graphics Disclosure: Ponte Vecchio, Rambo Cache, and Gelato," https://www.anandtech.com/show/15188/analyzing-intels-discrete-xe-hpc-graphics-disclosure-ponte-vecchio/2.

[2] "ASTRA-SIM: Enabling SW/HW Co-Design Exploration for Distributed DL Training Platforms," https://github.com/astra-sim/astra-sim.git.

[3] "Gaudi Training Platfrom White Paper," https://habana.ai/wp-content/uploads/2019/06/Habana-Gaudi-Training-Platform-whitepaper.pdf.

[4] "https://www.barefootnetworks.com/."

[5] "Intel's First 7nm GPU Xe HPC Vecchio, Rambo Cache, and Gelato," https://wccftech.com/intel-7nm-xe-hpc-gpu-diagram-ponte-vecchio/.

[6] "Mellanox scalable hierarchical aggregation and reduction protocol (sharp) https://www.mellanox.com/products/sharp."

[7] "oneapi collective communications library (oneccl)." [Online]. Available: https://github.com/oneapi-src/oneCCL

[8] "Spotted at hot chips: Quad tile intel xe-hp gpu," 2020. [Online]. Available: https://www.anandtech.com/show/15996/spotted-at-hot-chips-quad-tile-intel-xehp-gpu

[9] N. Agarwal, T. Krishna, L. Peh, and N. K. Jha, "Garnet: A detailed on-chip network model inside a full-system simulator," in *2009 IEEE International Symposium on Performance Analysis of Systems and Software*, April 2009, pp. 33–42.

[10] D. Amodei and D. Hernandez, "Ai and compute," 2018. [Online]. Available: https://openai.com/blog/ai-and-compute/

[11] K. L. R. C. Arnold, K. Lee, V. Rao, and W. C. Arnold, "Application-specific hardware accelerators," Mar 2019. [Online]. Available: https://engineering.fb.com/data-center-engineering/accelerating-infrastructure/

[12] A. Arunkumar, E. Bolotin, B. Cho, U. Milic, E. Ebrahimi, O. Villa, A. Jaleel, C.-J. Wu, and D. Nellans, "Mcm-gpu: Multi-chip-module gpus for continued performance scalability," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ser. ISCA '17. New York, NY, USA: ACM, 2017, pp. 320–332. [Online]. Available: http://doi.acm.org/10.1145/3079856.3080231

[13] E. Chan, R. van de Geijn, W. Gropp, and R. Thakur, "Collective communication on architectures that support simultaneous communication over multiple links," in *Proceedings of the Eleventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '06. New York, NY, USA: ACM, 2006, pp. 2–11. [Online]. Available: http://doi.acm.org/10.1145/1122971.1122975

[14] W. J. Dally, C. T. Gray, J. Poulton, B. Khailany, J. Wilson, and L. Dennison, "Hardware-enabled artificial intelligence," in *2018 IEEE Symposium on VLSI Circuits*, June 2018, pp. 3–6.

[15] D. Das, S. Avancha, D. Mudigere, K. Vaidyanathan, S. Sridharan, D. D. Kalamkar, B. Kaul, and P. Dubey, "Distributed deep learning using synchronous stochastic gradient descent," *CoRR*, vol. abs/1602.06709, 2016. [Online]. Available: http://arxiv.org/abs/1602.06709

[16] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

[17] J. Dong, Z. Cao, T. Zhang, J. Ye, S. Wang, F. Feng, L. Zhao, X. Liu, L. Song, L. Peng *et al.*, "Eflops: Algorithm and system co-design for a high performance distributed training platform," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020, pp. 610–622.

[18] Facebook, "Parametrized Recommendation and AI Model benchmarks." [Online]. Available: https://github.com/facebookresearch/param

[19] J. P. Grossman, B. Towles, B. Greskamp, and D. E. Shaw, "Filtering, reductions and synchronization in the anton 2 network," in *2015 IEEE International Parallel and Distributed Processing Symposium*, 2015, pp. 860–870.

[20] P. Grun, S. Hefty, S. Sur, D. Goodell, R. D. Russell, H. Pritchard, and J. M. Squyres, "A brief introduction to the openfabrics interfaces - a new network api for maximizing high performance application efficiency," in *2015 IEEE 23rd Annual Symposium on High-Performance Interconnects*, 2015, pp. 34–39.

[21] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[22] T. Hoefler, S. D. Girolamo, K. Taranov, R. E. Grant, and R. Brightwell, "spin: High-performance streaming processing in the network," *CoRR*, vol. abs/1709.05483, 2017. [Online]. Available: http://arxiv.org/abs/1709.05483

[23] Intel, "Intel machine learning scalability library (mlsl)," 2018. [Online]. Available: https://github.com/intel/MLSL

[24] Z. Jia, M. Maggioni, B. Staiger, and D. P. Scarpazza, "Dissecting the NVIDIA volta GPU architecture via microbenchmarking," *CoRR*, vol. abs/1804.06826, 2018. [Online]. Available: http://arxiv.org/abs/1804.06826

[25] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," *CoRR*, vol. abs/1704.04760, 2017. [Online]. Available: http://arxiv.org/abs/1704.04760

[26] N. P. Jouppi, D. H. Yoon, G. Kurian, S. Li, N. Patil, J. Laudon, C. Young, and D. Patterson, "A domain-specific supercomputer for training deep neural networks," *Communications of the ACM*, vol. 63, no. 7, pp. 67–78, 2020.

[27] L. Ke, U. Gupta, B. Y. Cho, D. Brooks, V. Chandra, U. Diril, A. Firoozshahian, K. Hazelwood, B. Jia, H. S. Lee, M. Li, B. Maher, D. Mudigere, M. Naumov, M. Schatz, M. Smelyanskiy, X. Wang, B. Reagen, C. Wu, M. Hempstead, and X. Zhang, "Recnmp: Accelerating personalized recommendation with near-memory processing," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020, pp. 790–803.

[28] D. J. Kerbyson, K. J. Barker, A. Vishnu, and A. Hoisie, "A performance comparison of current hpc systems: Blue gene/q, cray xe6 and infiniband systems," *Future Generation Computer Systems*, vol. 30, pp. 291 – 304, 2014, special Issue on Extreme Scale Parallel Architectures and Systems, Cryptography in Cloud Computing and Recent Advances in Parallel and Distributed Systems, ICPADS 2012 Selected Papers. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X13001337

[29] B. Klenk *et al.*, "An in-network architecture for accelerating shared-memory multiprocessor collectives," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020. [Online]. Available: https://www.iscaconf.org/isca2020/papers/466100a996.pdf

[30] H. Kwon, P. Chatarasi, M. Pellauer, A. Parashar, V. Sarkar, and T. Krishna, "Understanding reuse, performance, and hardware cost of dnn dataflows: A data-centric approach," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2019, pp. 754–768.

[31] H. Kwon, A. Samajdar, and T. Krishna, "Maeri: Enabling flexible dataflow mapping over dnn accelerators via reconfigurable interconnects," *SIGPLAN Not.*, vol. 53, no. 2, p. 461–475, Mar. 2018. [Online]. Available: https://doi.org/10.1145/3296957.3173176

[32] Y. Kwon, Y. Lee, and M. Rhu, "Tensordimm: A practical near-memory processing architecture for embeddings and tensor operations in deep learning," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '52. New York, NY, USA: Association for Computing Machinery, 2019, p. 740–753. [Online]. Available: https://doi.org/10.1145/3352460.3358284

[33] Y. Kwon and M. Rhu, "Beyond the memory wall: A case for memory-centric hpc system for deep learning," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2018, pp. 148–161.

[34] D. Lepikhin, H. Lee, Y. Xu, D. Chen, O. Firat, Y. Huang, M. Krikun, N. Shazeer, and Z. Chen, "Gshard: Scaling giant models with conditional computation and automatic sharding," *arXiv preprint arXiv:2006.16668*, 2020.

[35] 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," *CoRR*, vol. abs/1903.04611, 2019. [Online]. Available: http://arxiv.org/abs/1903.04611

[36] 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, 2019, p. 279–291. [Online]. Available: https://doi.org/10.1145/3307650.3322259

[37] Y. Li, J. Park, M. Alian, Y. Yuan, Z. Qu, P. Pan, R. Wang, A. Schwing, H. Esmaeilzadeh, and N. Kim, "A network-centric hardware/algorithm co-design to accelerate distributed training of deep neural networks," 10 2018, pp. 175–188.

[38] H. Mikami, H. Suganuma, P. U.-Chupala, Y. Tanaka, and Y. Kageyama, "Imagenet/resnet-50 training in 224 seconds," *CoRR*, vol. abs/1811.05233, 2018. [Online]. Available: http://arxiv.org/abs/1811.05233

[39] S. A. Mojumder, M. S. Louis, Y. Sun, A. K. Ziabari, J. L. Abellàn, J. Kim, D. Kaeli, and A. Joshi, "Profiling dnn workloads on a volta-based dgx-1 system," in *2018 IEEE International Symposium on Workload Characterization (IISWC)*, Sep. 2018, pp. 122–133.

[40] D. Mudigere *et al.*, "High-performance, distributed training of large-scale deep learning recommendation models," 2021.

[41] M. Naumov, D. Mudigere, H. M. Shi, J. Huang, N. Sundaraman, J. Park, X. Wang, U. Gupta, C. Wu, A. G. Azzolini, D. Dzhulgakov, A. Mallevich, I. Cherniavskii, Y. Lu, R. Krishnamoorthi, A. Yu, V. Kondratenko, S. Pereira, X. Chen, W. Chen, V. Rao, B. Jia, L. Xiong, and M. Smelyanskiy, "Deep learning recommendation model for personalization and recommendation systems," *CoRR*, vol. abs/1906.00091, 2019. [Online]. Available: http://arxiv.org/abs/1906.00091

[42] NVIDIA, "Nvidia collective communications library (nccl)," 2018. [Online]. Available: https://developer.nvidia.com/nccl

[43] NVIDIA, "Nvidia dgx-2," 2019. [Online]. Available: https://www.nvidia.com/en-us/data-center/dgx-2/

[44] S. Ouyang, D. Dong, Y. Xu, and L. Xiao, "Communication optimization strategies for distributed deep learning: A survey," 2020.

[45] P. Patarasuk and X. Yuan, "Bandwidth optimal all-reduce algorithms for clusters of workstations," *J. Parallel Distrib. Comput.*, vol. 69, no. 2, pp. 117–124, Feb. 2009. [Online]. Available: http://dx.doi.org/10.1016/j.jpdc.2008.09.002

[46] R. Rajamony, L. B. Arimilli, and K. Gildea, "Percs: The ibm power7-ih high-performance computing system," *IBM J. Res. Dev.*, vol. 55, no. 3, p. 233–244, May 2011. [Online]. Available: https://doi.org/10.1147/JRD.2011.2109230

[47] S. Rashidi, P. Shurpali, S. Sridharan, N. Hassani, D. Mudigere, K. Nair, M. Smelyanski, and T. Krishna, "Scalable distributed training of recommendation models: An astra-sim + ns3 case-study with tcp/ip transport," in *2020 IEEE Symposium on High-Performance Interconnects (HOTI)*, 2020, pp. 33–42.

[48] S. Rashidi, S. Sridharan, S. Srinivasan, and T. Krishna, "ASTRA-SIM: Enabling SW/HW Co-Design Exploration for Distributed DL Training Platforms," in *IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS 2020, Boston, MA, USA, August 22-26, 2020*. IEEE, 2020. [Online]. Available: https://synergy.ece.gatech.edu/wp-content/uploads/sites/332/2020/03/astrasim_ispass2020.pdf

[49] J. Rasley, S. Rajbhandari, O. Ruwase, and Y. He, "Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 3505–3506.

[50] A. Samajdar, J. M. Joseph, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, "A systematic methodology for characterizing scalability of dnn accelerators using scale-sim," in *2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2020, pp. 58–68.

[51] A. Sapio, M. Canini, C. Ho, J. Nelson, P. Kalnis, C. Kim, A. Krishnamurthy, M. Moshref, D. R. K. Ports, and P. Richtárik, "Scaling distributed machine learning with in-network aggregation," *CoRR*, vol. abs/1903.06701, 2019. [Online]. Available: http://arxiv.org/abs/1903.06701

[52] P. Shamis, M. G. Venkata, M. G. Lopez, M. B. Baker, O. Hernandez, Y. Itigin, M. Dubman, G. Shainer, R. L. Graham, L. Liss, Y. Shahar, S. Potluri, D. Rossetti, D. Becker, D. Poole, C. Lamb, S. Kumar, C. Stunkel, G. Bosilca, and A. Bouteiller, "Ucx: An open source framework for hpc network apis and beyond," in *2015 IEEE 23rd Annual Symposium on High-Performance Interconnects*, 2015, pp. 40–43.

[53] Y. S. Shao, J. Clemons, R. Venkatesan, B. Zimmer, M. Fojtik, N. Jiang, B. Keller, A. Klinefelter, N. Pinckney, P. Raina, S. G. Tell, Y. Zhang, W. J. Dally, J. Emer, C. T. Gray, B. Khailany, and S. W. Keckler, "Simba: Scaling deep-learning inference with multi-chip-module-based architecture," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '52. New York, NY, USA: Association for Computing Machinery, 2019, p. 14–27. [Online]. Available: https://doi.org/10.1145/3352460.3358302

[54] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-lm: Training multi-billion parameter language models using model parallelism," *CoRR*, vol. abs/1909.08053, 2019. [Online]. Available: http://arxiv.org/abs/1909.08053

[55] R. Thakur, R. Rabenseifner, and W. Gropp, "Optimization of collective communication operations in mpich," *Int. J. High Perform. Comput. Appl.*, vol. 19, no. 1, pp. 49–66, Feb. 2005. [Online]. Available: http://dx.doi.org/10.1177/1094342005051521

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

[57] Y. Wang, Q. Wang, S. Shi, X. He, Z. Tang, K. Zhao, and X. Chu, "Benchmarking the performance and power of ai accelerators for ai training," 2019.

[58] Y. Wu *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *CoRR*, vol. abs/1609.08144, 2016. [Online]. Available: http://arxiv.org/abs/1609.08144

[59] X. Zhang, J. K. Lin, S. Wickramanayaka, S. Zhang, R. Weerasekera, R. Dutta, K. F. Chang, K. Chui, H. Li, D. Ho, L. Ding, G. Katti, S. Bhattacharya, and D. Kwong, "Heterogeneous 2.5d integration on through silicon interposer," *Applied physics reviews*, vol. 2, p. 021308, 2015.