

# Deep Learning Training in Facebook Data Centers: Design of Scale-up and Scale-out Systems

Maxim Naumov\*, John Kim<sup>†</sup>, Dheevatsa Mudigere<sup>‡</sup>, Srinivas Sridharan, Xiaodong Wang, Whitney Zhao, Serhat Yilmaz, Changkyu Kim, Hector Yuen, Mustafa Ozdal, Krishnakumar Nair, Isabel Gao, Bor-Yiing Su, Jiyan Yang and Mikhail Smelyanskiy  
Facebook, 1 Hacker Way, Menlo Park, CA

**Abstract**—Large-scale training is important to ensure high performance and accuracy of machine-learning models. At Facebook we use many different models, including computer vision, video and language models. However, in this paper we focus on the **deep learning recommendation models (DLRMs), which are responsible for more than 50% of the training demand in our data centers**. Recommendation models present unique challenges in training because **they exercise not only compute but also memory capacity as well as memory and network bandwidth**. **As model size and complexity increase, efficiently scaling training becomes a challenge**. To address it we design Zion Facebooks next-generation large-memory training platform that consists of both CPUs and accelerators. Also, we discuss the design requirements of future scale-out training systems.

## I. INTRODUCTION

Artificial intelligence (AI) applications are rapidly evolving and increasing the demands on hardware and systems. Machine learning (ML), deep learning (DL) in particular, has been one of the driving forces behind the remarkable progress in AI and has become one of the most demanding workloads in terms of compute infrastructure in the data centers [8], [15], [27], [46]. Moreover, the continued growth of DL models in terms of complexity, coupled with significant slowdown in transistor scaling, has necessitated going beyond traditional general-purpose processors and developing specialized hardware with holistic system-level solutions to improve performance, power, and efficiency [11], [33].

Within Facebook, DL is used across many social network services, including computer vision, i.e. image classification, object detection, as well as video understanding. In addition, it is used for natural language processing, i.e. translation and content understanding. However, some of the most important DL models within Facebook are the recommendation models used for ranking and click through rate (CTR) prediction, including News Feed and search services [24].

The use of DL models is often split into inference and training work categories [4], [26]. Details of inference at Facebook has been discussed earlier [23], [40]; in comparison, we address the challenges in training and in particular, the scale-out requirements of the deep learning recommendation models (DLRMs) at Facebook [38].

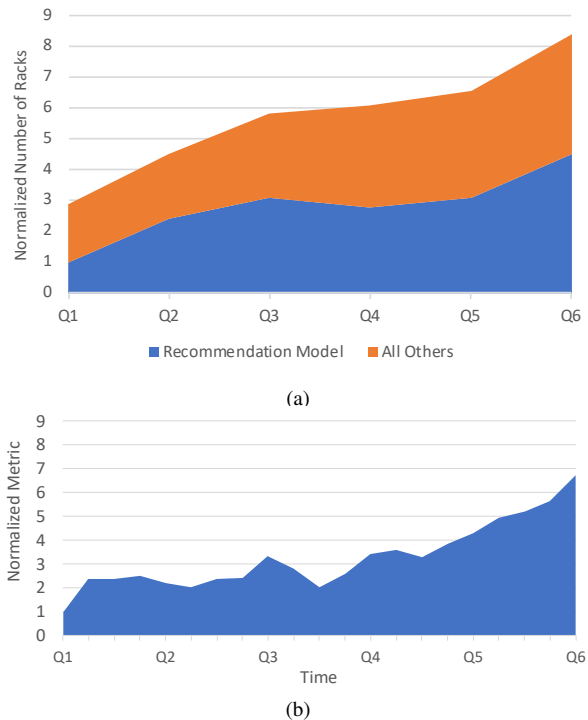


Fig. 1. (a) Server compute demand for training and (b) number of distributed training workflows across Facebook data centers.

The increase in compute in Facebook data centers from training is shown in Figure 1(a). Across a period of 18 months, there has been over  $4\times$  increase in the amount of computer resources utilized by training workloads. In addition, the number of workloads submitted for distributed training, as shown in Figure 1(b), has increased at an even higher rate – resulting in up to  $7\times$  increase in the number of training workflows. Thus, the demand for training deep learning workloads is continuing to increase while the compute necessary to support it is also increasing proportionally.

Prior training platform from Facebook, e.g. Big Basin [31], consisted of NVidia GPUs. However, it did not leverage other accelerators and only had support for a limited number of CPUs. On the other hand, the Zion next-generation training platform incorporates 8 CPU sockets, with a modular design having separate sub-components for CPUs (Angels Landing 8-socket system) and accelerators (Emeralds Pools 8-accelerator system). This provides for sufficient general purpose compute and more importantly, additional memory capacity.

\*mnaumov@fb.com <sup>‡</sup>dheevatsa@fb.com

<sup>†</sup>jjk12@kaist.edu Currently at Korea Advanced Institute of Science and Technology (KAIST). Work done while at Facebook.

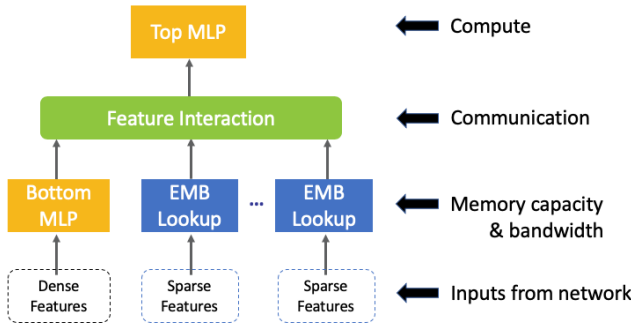


Fig. 2. High-level overview of DLRM.

Zion also introduced the common form factor **OCP Accelerator Module (OAM)** [49]<sup>1</sup>, which has been adopted by leading GPU vendors such as NVidia, AMD, Intel, as well as startups, such as Habana which was recently acquired by Intel. This is important for enabling consumers such as Facebook to build vendor agnostic accelerator based systems.

In this work, we provide an overview of the DLRM workloads, including the description and analysis of

- Training at Facebook
- Zion hardware platform
- Impact on the accelerator fabric design
- Implications for future scale-out systems

## II. BACKGROUND

### A. Recommendation Model

Neural network-based recommendation models, which are used to address personalization for different services, have become an important class of DL algorithms within Facebook. High-level block overview of a typical recommendation model is shown in Figure 2, while its implementation in PyTorch framework has been publicly released in DLRM [38].

The inputs to the recommendation model include both dense and sparse features. The dense or the continuous features are processed with a bottom multilayer perceptron (MLP) while the sparse or the categorical features are processed using embeddings. The second-order interactions of different features are computed explicitly. Finally, the results are processed with a top MLP and fed into a sigmoid function in order to provide a probability of a click.

### B. Increase in Complexity

In general the model complexity in terms of number of parameters increases by more than  $2\times$  over 2 years, as shown on Figure 3. Notice that the increase trend is not monotonic because in order to alleviate pressure on the training, inference and serving resources, we are constantly evaluating novel techniques to improve efficiency, such as quantization and compression. However, over time the newly available complexity budget is often reused to improve and augment the more efficient model, therefore driving its size up again.

<sup>1</sup>Proposed and developed as part of the [Open Compute Project](#) (OCP).

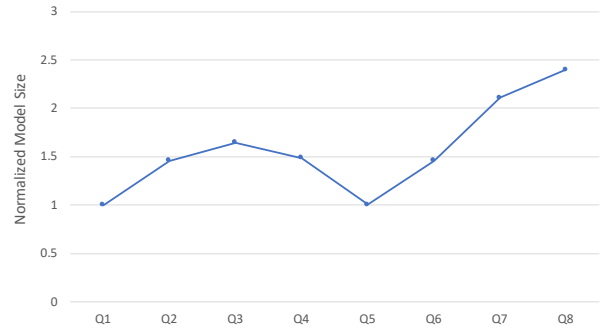


Fig. 3. Increase in model complexity over time.

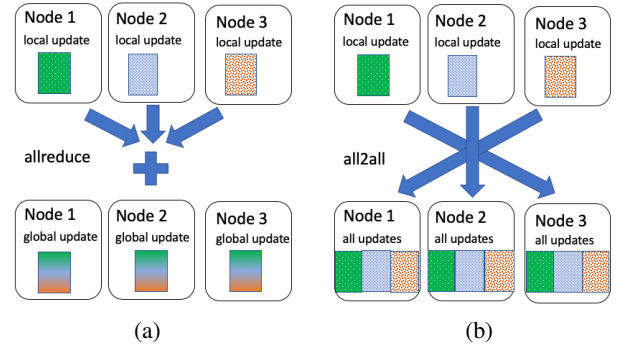


Fig. 4. Communication patterns that are common in (a) data- and (b) model-parallelism. Both communication patterns need to be supported in DLRM.

### C. Distributed Training

Distributed training becomes particularly important in the context of increasing model sizes. It may rely on two types of parallelism: data- and model-parallelism. The former enables faster training as each worker trains on different data while the latter enables bigger model to train on the same data.

**In data-parallelism:** The input data samples are distributed across different nodes. Each node processes the input data independently with replicas of parameters on each node, and aggregating the local parameter updates into a global update on all the nodes. This requires communicating only the updates between the nodes, but communication volume increases due to replication as the number of nodes increases. Therefore, scaling out requires large enough mini-batch size to provide sufficient parallelism and computation to hide the communication overhead.

**In model-parallelism:** The model weights corresponding to neural network layers are distributed across multiple nodes. Each node processes the entire mini-batch of data and communicates the activations forward or error gradients backwards to other nodes. This introduces additional synchronization across all nodes after each distributed layer in the forward and backward pass. However, it allows us to fit the model into the aggregate memory of all distributed nodes.

Note that a single embedding table contains tens of millions of vectors, each with hundreds of elements. It requires significant memory capacity, on the order of GBs. Therefore,

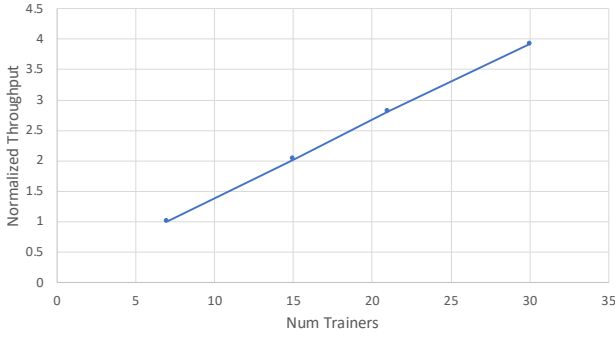


Fig. 5. Performance as the number of trainers are increased.

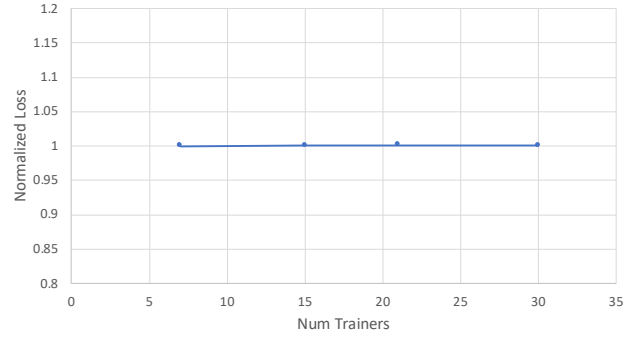


Fig. 6. Quality as the number of trainers are increased.

an embedding table often exists as a single instance and can not be replicated on multiple devices or nodes. In contrast MLPs are relatively small and can be replicated many times. Therefore, we will leverage both types of parallelism while training DLRMs.

### III. TRAINING AT FACEBOOK

#### A. Overview of Training at Facebook

The training of recommendation models often requires the distribution of the model across multiple devices within a single node or multiple nodes. Hence requiring both data- and model-parallelism to scale the performance of training [38]. The distributed training can be performed using a combination of synchronous algorithms that produce results equivalent to a sequential run of the model [18], [39] or asynchronous algorithms that scale to a larger number of nodes [19], [42], [50]. In general, the asynchronous algorithms can perform a single step of forward and backward propagation faster, but may require more steps to achieve convergence or even converge to a sub-optimal minimum [9], [51].

**Synchronous training** relies on collective communication to achieve the model and data parallelism.

The compute intensive MLPs are replicated across devices and work on parts of the mini-batch data samples. Notice that training of MLPs requires an `allreduce` communication primitive to synchronize their weights during backward propagation, as shown in Figure 4(a).

Further, a model may contain tens of embedding tables, which can not be replicated due to memory capacity constraints. These tables are often distributed across devices and each of them processes an entire mini-batch of lookups. Then, an embedding lookup produces several vectors corresponding to the elements in the mini-batch. Let us use the same color to denote vectors resulting from a single embedding table. The need to exchange these vectors in the forward pass and their gradients in the backward pass gives rise to the `alltoall` communication primitive, as shown in Figure 4(b).

In this setting, embedding lookups take advantage of the aggregate memory bandwidth across devices, while the full model exercises the interconnect, because multiple lookup

results need to be communicated through to be exchanged and computed at once.

**Asynchronous training** is well suited for a disaggregated design with use of dedicated parameter servers and trainers, that are relying on point-to-point `send/recv` communication or Remote Procedure Calls (RPC) [10].

The compute intensive MLPs are replicated on different training processes and perform local weights updates based on the data samples they receive individually, only occasionally synchronizing with a master copy of the weights stored on the parameter server.

The embedding tables are not replicated due to memory constraints, but are assigned to different training processes that receive asynchronous updates throughout training. Notice that because we use indices to access only a few of the embedding vectors in the forward pass, the simultaneous updates to an embedding table in the backward pass only collide when the indices used in the sparse lookups overlap between them.

It should be noted that both or a combination of these training algorithms can be used across different platforms such as a single node with multiple devices, multiple nodes or disaggregated setup of multiple trainers and parameter servers.

#### B. Scalability & Challenges

The throughput of model training is important for fast prototyping and iterating on new ideas during model development. A single machine is not able to provide the throughput we need for our large recommendation models, and therefore we are heavily investing in scaling distributed training.

Figure 5 shows the training throughput of one of our recent DLRMs. Here we use asynchronous training to avoid being bottlenecked by slow machines and/or interconnects. Also, we make sure the synchronization of the dense parameters is frequent enough, so that models will not diverge on different machines. As a result, we observe that the training throughput scales almost linearly with the number of hosts we use in one job, while model quality remains in an acceptable range, as is shown in Figure 6.

In our experience, asynchronous training works well when a limited number of trainers is used, but with increasing number of trainers, we must incorporate synchronous training as an option into our system and hardware platforms.

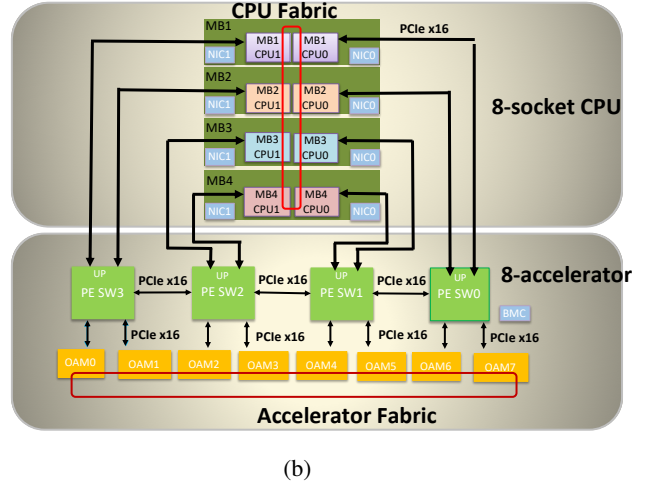
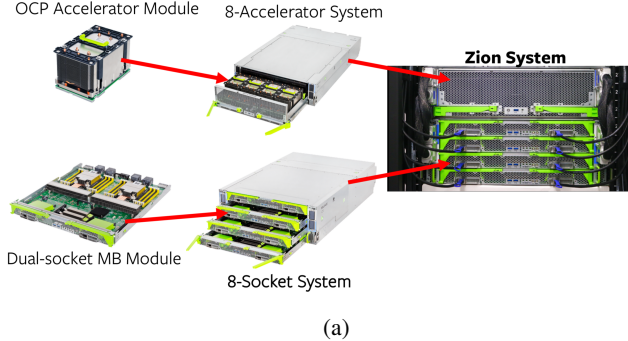


Fig. 7. (a) High-level overview of Zion system integration and (b) detailed block diagram of the Zion platform

### C. Interconnection Networks

Aside from the standard `send/recv` communication primitives, the data- and model-parallelism used in distributed training give rise to two types of communication patterns: (i) `allreduce` operation that is often used to aggregate local parameter updates in the backward pass and (ii) `alltoall` operation that can be used to exchange the activations and error gradients across multiple nodes with different model weights.

The efficient synchronous or asynchronous implementation of these primitives relies on the support from the router or switch microarchitecture and the underlying interconnect network. As technology evolved through the years, the interconnection network fabric has varied. For example, the high performance computing (HPC) interconnect commonly used in the late 80s or 90s were often based on the low-radix topologies, such as 2D or 3D mesh or torus network [6], [16]. As the pin bandwidth has increased it has been shown that the full bandwidth can be effectively utilized by partitioning it across an increasing number of ports, resulting in high-radix topologies [29] that were designed to reduce network diameter and cost [30]. Also, it is important to note that HPC community has often relied on custom fabric and protocols, while in the data centers, in order to drive down the costs, the commodity interconnects are much more prevalent [7].

## IV. ZION SCALE-UP TRAINING

### A. Overview

The building blocks of the Zion system consist of CPUs, accelerators and a flexible fabric that provides high performance while interconnecting these components [44]. Specifically, Zion decouples memory, compute, and network components of the system, allowing each to scale independently as shown in Figure 7. The baseline Zion system provides  $8 \times$  NUMA CPU sockets with a large pool of DDR memory for capacity and  $8 \times$  accelerators to provide the high compute capacity and also high bandwidth memory.

TABLE I  
ZION DEVICE COMPARISONS.

		CPU	Accelerator
# of devices		8	8
Compute in FP32 (TFlops)	aggregate	$\sim 20$	$\sim 100$
Compute in FP16/BF16 (TFlops)	aggregate	$\sim 50$	$\sim 1000$
Memory Capacity (TB)	aggregate	$\sim 2$	$\sim 0.2$
Memory Bandwidth (TB/s)	aggregate	$\sim 1$	$\sim 8$
Power (Watts)	per device	$\sim 100$	$\sim 200$

One of the challenges with leveraging accelerators for a training platform is determining which ones to use, given a large number of them that are becoming available. It is infeasible to develop and enable a unique system for each one of the different accelerators. As a result, Facebook-led Open Accelerator Infrastructure (OAI) initiative [3] proposed to define vendor-agnostic common accelerator infrastructure, including a standard accelerator form factor Open Accelerator Module (OAM) that has been open sourced to the hardware community. The OAM form factor abstracts the various requirements to make it solution-agnostic and defined as a common form factor that can be adopted by different accelerator vendors. The common form factor along with the baseboard enables using multiple accelerator alternatives with the same system design.

The comparison between the characteristics of typical CPUs and accelerators is shown in Table I. Notice that while the number of CPUs and accelerators in the system is the same, their compute and memory capabilities differ significantly. For example, the accelerator provides one or even two orders of magnitude higher compute (i.e., TFlops) as well as almost an order of magnitude higher memory bandwidth, all of which come at the cost of higher power. Also, accelerators rely on high-bandwidth memory (HBM) that does not necessarily provide the same capacity as the DDR memory used in the CPUs – thus, the CPUs provide an order of magnitude higher memory capacity.



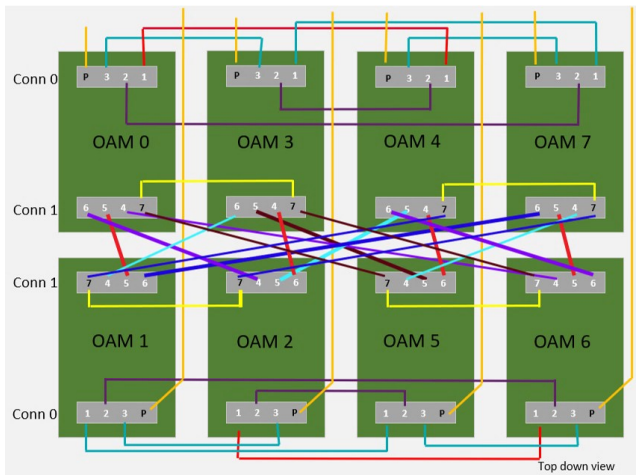


Fig. 8. Accelerator fabric interconnecting layout in Zion.

### B. Accelerator Fabric Topology

Another big challenge in designing a common multi-source accelerator platform is the interconnect, because different vendors have distinct solutions utilizing various topologies, protocols, and number of lanes/links. The Zion platform consists of 3 different types of interconnect fabrics – the CPU fabric, the accelerator fabric and the PCIe interconnect that provides connectivity between CPUs and accelerators, see Figure 7. The CPU fabric options are limited by the vendors, but both PCIe and the accelerator interconnect are flexible, allowing for co-design to meet application needs.

The main components of any interconnection network include the topology, routing, flow control, and router micro-architecture [17]. Table II summarizes how these components differ from conventional high-performance and accelerator fabric interconnection networks. Overall the objectives of the two fabrics are slightly different – HPC systems require low latency (and low network diameter) but also high bisection bandwidth. In comparison, accelerator fabrics are less latency sensitive but often require high node-to-node bandwidth to efficiently support collective communication.

The router micro-architecture impacts the topology and other components of the network. In HPC systems, given a topology, the routing algorithm, especially adaptive routing, is critical to exploit path diversity and maximize performance; however, most accelerator fabrics do not have hardware routers and therefore routing algorithms are not as critical because of the deterministic communication pattern.

In accelerator fabric without hardware support for communication, it often resembles “store & forward” flow control, compared to “cut-through” flow control, since data is copied from one node to its neighboring node before being transmitted downstream. All nodes perform the same flow control at the same time, which results in high utilization across all nodes.

In addition to software, there are many other physical design constraints for vendor agnostic topology design because each offering differs on many aspects and there is no established

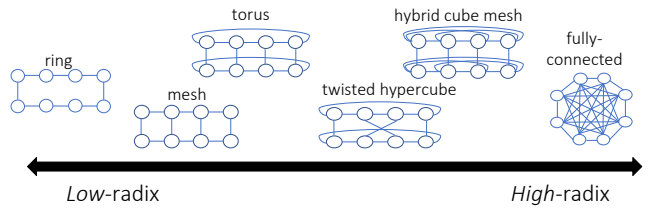


Fig. 9. Different topology design space for an 8-node system.

TABLE II  
HIGH-LEVEL COMPARISON OF INTERCONNECTION NETWORKS.

	High Performance Interconnect	Accelerator Fabric
Topology	low-diameter, high (bisection) bandwidth	high (node) bandwidth
Routing	adaptive routing	deterministic routing
Flow Control	cut-through	store & forward
Fabric Design	router centric	node centric

standard. An example of one such constraint is the accelerator interconnect link mapping, for some offering this may fan out on both sides and in other cases can be only on one side of the accelerator chip. The baseboard design presents routing length and other challenges to deal with the interconnect link insertion loss and also PCB layers/cost considerations. Figure 8 is an illustration of top-down view of the OAM layout in Zion showing the connections between each OAM, with each colored link representing a different routing layer. As per the OAM specification each module has two mezzanine connectors (Conn 0/1) at the south and north of each module and  $1 \times 16$  PCIe link to host (port  $P$ ). Further each module has  $7 \times 16$  interconnect links, shown in Figure 8 as ports number from 1 through 7, among these 1 – 6 are  $\times 16$  links each and 7 is split into  $2 \times 8$  links to accommodate different topologies shown in Figure 9.

### C. Communication Pattern

The Zion platform is designed to support both asynchronous and synchronous training algorithms, including support for model- and data-parallelism. To support both types of parallelism we rely on the optimized `allreduce` and `alltoall` communication primitives implemented over the accelerator fabric topology.

To understand the impact of accelerator topology across these communication primitives, we provide an analytical model comparing the added latency of going through a node for a low-radix ring topology and a high-radix fully-connected (FC) topology for an 8-node system shown in Figure 10. We plot the performance ratio between the execution time predicted by the model for ring and FC topology, ignoring additional software overheads often found in practice [32]. In our model we fix per node bandwidth  $B$ , and vary the message size  $M$  and per-node latency  $\alpha$ , to simulate the traffic pattern. The execution time for `allreduce` ( $T_{AR}$ ) for the two topologies, using a ring algorithm, can be summarized as

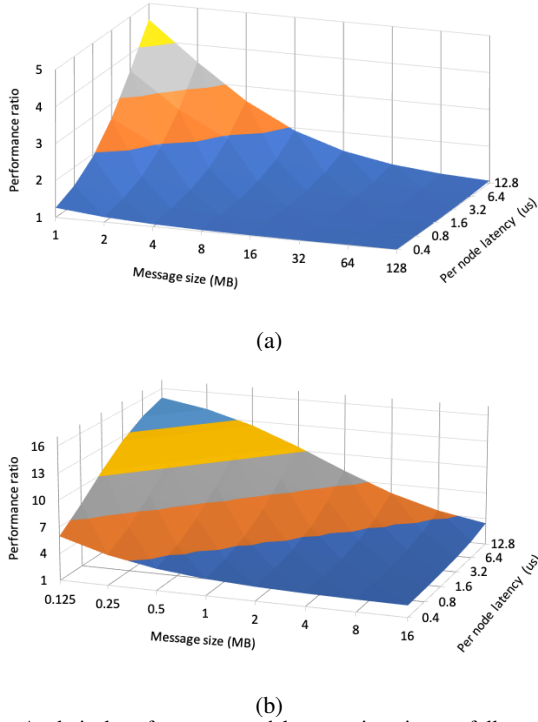


Fig. 10. Analytical performance model comparing ring vs fully connected topology for (a) `allreduce` and (b) `alltoall` communication primitive.

$$T_{AR_{ring}} = 2 \frac{M}{B} \frac{p-1}{p} + 2\alpha(p-1) \quad (1)$$

$$T_{AR_{FC}} = 2 \frac{M}{b} \frac{1}{p} + 2\alpha \quad (2)$$

where  $p$  is the number of nodes and  $b$  is the bandwidth per channel. In the comparison,  $p = 8$  and we assume total bandwidth from each accelerator node is constant – thus, the ring has “fatter” channels while the fully-connected topology has “thinner” channels. Thus, the bandwidth per channel in the FC is effectively  $b = B/(p-1)$ . By plugging this value into  $T_{AR_{FC}}$ , the bandwidth component of performance across the two topologies can be seen to be identical.

Figure 10(a) plots ratio  $T_{AR_{ring}}/T_{AR_{FC}}$ . Note that the ratio higher than 1 represents region where FC outperforms the ring topology. As the message size increases, the bandwidth component dominates and therefore the physical topology has no impact. However, as the message size decreases and per-node latency increases, the latency component dominates and therefore the performance benefit from the FC increases significantly (it minimizes the network hop count).

The execution time for `alltoall` ( $T_{A2A}$ ) for the two topologies, using a ring algorithm, can be summarized as

$$T_{A2A_{ring}} = \gamma \left( \frac{M}{B} \frac{1}{p} + \alpha \right) \quad (3)$$

$$T_{A2A_{FC}} = \frac{M}{B} \frac{p-1}{p} + \alpha \quad (4)$$

where  $\gamma = 2 \sum_{h=1}^q h + r \frac{p}{2}$ , with quotient  $q = (p-1)/2$ , remainder  $r = (p-1)\%2$  and # of hops between nodes  $h$ .

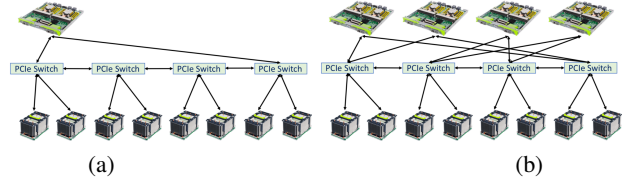


Fig. 11. System flexibility by leveraging only (a) single 2-socket CPU system and (b) four 2-socket CPUs within the Zion platform. Even with a single 2-socket system, the interconnect of the PCIe switches enable full connectivity between the CPUs and the accelerators.

For FC topology, the cost of  $T_{A2A}$  is approximately 1/2 of  $T_{AR}$  because `alltoall` communication only consists of a single phase, while `allreduce` based on the ring algorithm consists of 2 phases. In comparison, for the ring topology, `alltoall` requires each node to send messages to all other nodes and performance is proportional to the sum of the hop count between each pair of nodes  $\gamma$ , while for FC the network diameter is 1. For the ring topology, we assume `alltoall` communication is done in multiple steps where each node sends message to destination that is 1-hop away, 2-hop away, etc and ensure all channels are fully utilized. Thus, for large message sizes, the benefit from FC approaches  $\gamma/(p-1)$  or  $\sim 2.3$  in the 8-node example. For smaller message size, the benefit from FC is much higher because of the latency component as shown in Figure 10(b).

In distributed synchronous training of DLRMs the message size for `allreduce` is often around 10MB – thus, the physical topology does not have significant impact on overall communication cost. However, for `alltoall` communication the message size is much smaller (e.g., on the order of 100KB), therefore FC topology can provide up to  $3\times$  improvement in communication cost compared to the ring topology. This pushes distributed DLRM training characteristics towards HPC workloads that rely on global traffic patterns and where low-diameter topology can provide significant benefits.

#### D. Hardware Design Flexibility

Although Zion hardware is designed mainly for Facebook deep learning recommendation workload, the design is formed by 4 identical dual-socket motherboard modules and is flexible enough to be configured for example to use 2 sockets plus 8 accelerators, as shown on Figure 11 (a). This 2 CPU sockets configuration may be used for other workloads which do not require large memory footprint, such as computer vision and natural language processing. On the other hand, Figure 11 (b) shows the standard 8 socket plus 8 accelerators configuration.

#### V. DISCUSSION ON SCALE-OUT TRAINING

The Zion system described earlier provides significant amount of compute and memory capacity to support large neural network models. However, one limitation of its memory system is that aggregate 1.5TB of DDR memory does not provide high enough memory bandwidth. Moreover, it is anticipated that model complexity and the amount of input data will continue to grow as shown on Figure 3, resulting in the need for increased training throughput.

TABLE III  
DESCRIPTION OF DIFFERENT SCALE-OUT TRAINING SYSTEMS.

Topology Protocol	Flat	Hierarchical
Homogeneous	TPU [48] Habana [36]	Habana HLS-1 [36] Intel SpringCrest [47]
Heterogeneous	N/A	Zion [44] DGX SuperPod [1]

The increasing demand can be best addressed by scaling out training to multiple nodes and increasingly leveraging accelerators. This translates into training system beyond a scale-up solution of single super-node (e.g. Zion) to scale-out systems with multiple such supernodes. However, distributed training introduces additional fabric and connectivity requirements across multiple nodes to efficiently support asynchronous and synchronous training with data- and model-parallelism. In this section, we discuss three vectors, namely network topology, network transport, and implementation of optimized collective primitives, that have significant implications on the design of scale-out systems for synchronous training of DLRMs.

#### A. Network Topology

Different scale-out training systems have been proposed, developed and successfully deployed by the industry as summarized in Table III. These systems can be classified as flat or hierarchical based on the fabric organization – flat topology uses a single global topology (e.g., 2D or 3D torus used in the TPU system [48]) while a hierarchical organization consists of a “supernode” with several accelerators, which is used as the building block to scale-out to larger number of nodes (e.g. DGX SuperPod [1]).

Another difference in scale-out fabric is whether the fabric is accelerator- or CPU-centric. We define an accelerator fabric similar to the definition in the Zion system – if the accelerators are used as the building block to scale-out, the system can be referred to as using accelerator-centric fabric, while if the CPU is used to scale-out, the system is CPU-centric. While the Zion system itself uses an accelerator-centric fabric, the Zion system when scaling-out is heterogeneous and CPU-centric since the 100 GbE network interface of the CPUs are used to scale-out.

The scale-out systems can further be categorized based on the communication protocol – a homogeneous system leverages the same link/interconnect technology for the entire system while a heterogeneous system exploits different interconnect technologies. We note that while different interconnect protocols (e.g., Infiniband [43], RoCE [37], NVlink [21], etc.) are available, heterogeneous systems can be bottlenecked by the channels that provide the smallest amount of bandwidth.

Also, we delineate a distinction between *global* and *local* bandwidth in the scale-out systems. Local bandwidth refers to the bandwidth for communicating between physically neighboring nodes, while global bandwidth is used to provide connectivity across the (global) system. In Figure 12, we compare the ratio of local vs global bandwidth across different scale-out systems. For the NVidia GPU systems, the amount of local

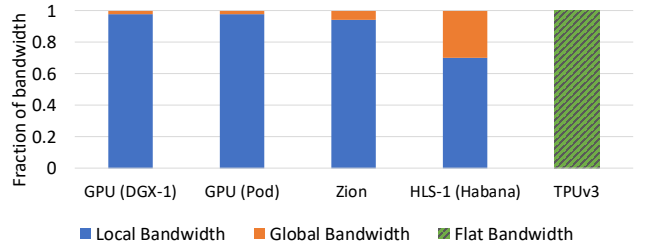


Fig. 12. Comparison of local vs global bandwidth for different scale-out systems. The TPU system does not differentiate between local and global bandwidth since it is flat topology.

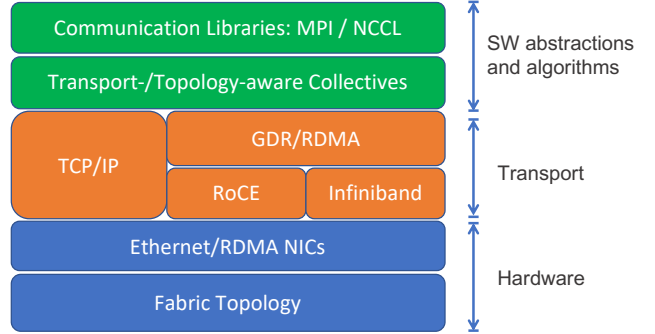


Fig. 13. A view of interconnect software and hardware stack

bandwidth dominates overall system bandwidth as each of the GPU nodes has 300 GB/s of bandwidth (through NVLinks) for local communication while the scale-out bandwidth through the Infiniband is 4x100 Gbps and shared across 8 GPUs. The amount of scale-out bandwidth increases for the GPU(Pod) but it is shared among larger number of GPUs and thus, the ratio is similar. For the Zion system, the fraction of global bandwidth is slightly higher since there are 8 CPU network interfaces to provide the scale-out bandwidth. In comparison, the HLS-1 system from Habana has 10 RoCE channels from the Gaudi accelerator and uses 7 channels for intra-box connectivity while 3 channels are used for scale-out or global bandwidth. In contrast, the flat topology (i.e. TPU) does not differentiate between local and global bandwidth since the fabric is organized as a single, global topology.

#### B. Network Transport

The network transport layer is key in implementing communication primitives for a given fabric topology. Hyper-scale data-centers have typically used kernel-based TCP/IP and Ethernet as the network transport for decades. However, accelerator-centric fabric, efficient scale-out communication entails moving data directly from one accelerator to another without host CPU involvement - requiring a transport that supports Remote Direct Memory Access (RDMA) or similar technology between accelerators.

Infiniband and RDMA over Converged Ethernet (RoCE) are two of the most popular examples of RDMA transport in use today, see Figure 13. These protocols are supported

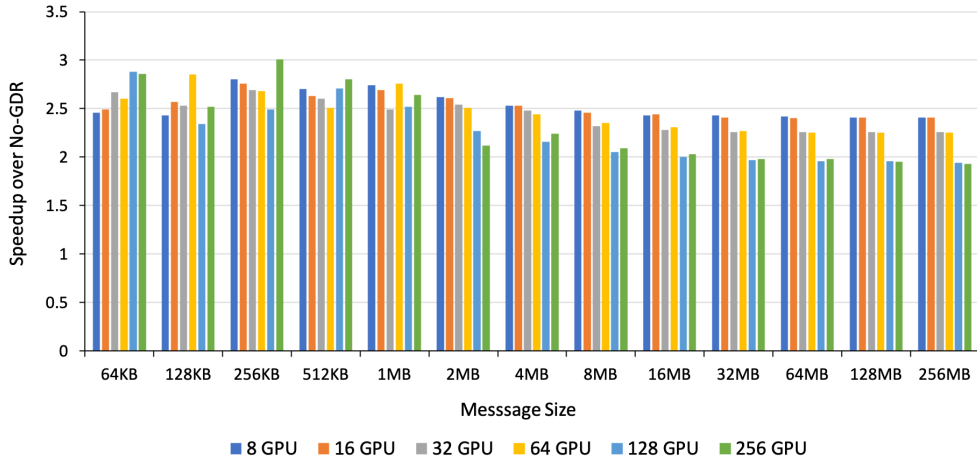


Fig. 14. Speedup of GDR over Non-GDR `allreduce` scale-out communication

by RDMA Network Interface Controllers (RNICs), which are PCIe devices that are comprised of a sophisticated DMA and transport engines, and include various management capabilities. RNICs use Verbs interface, which provides operations for resource management and data transfer, including send/receive and RDMA read/write operations. Today's RNICs typically offer the following capabilities:

- *Zero copy* – direct data exchange between buffers
- *Transport offload* – segment/assembly, retransmission
- *Kernel bypass* – RNICs hardware access from user mode
- *Flow control* – credit-based control, end-to-end reliability

Further, support for RDMA and Verbs by itself is not sufficient for enabling optimized network transports for accelerator-centric fabric. For instance, NVIDIA introduced a series of enhancements to enable RNICs to directly access GPU memory, culminating in GPUDirect RDMA (GDR) [2]. GDR provides direct path for data movement between GPU memory and Infiniband/RoCE NICs. It eliminates significant overheads resulting from PCIe over-subscription<sup>2</sup> and additional copy through the host/system memory.

To illustrate the performance advantages of using GDR, we evaluate GDR on Infiniband fabric against a highly optimized Non-GDR baseline, that requires one extra copy but also uses RDMA transport over lossless Infiniband fabric<sup>3</sup>. We run the benchmarks starting with one DGX-1 node (8 GPUs) all the way up to 32 nodes (256 GPUs). We force all communication via Infiniband to better isolate the benefits of GDR since it primarily affects the scale-out data path. We collect the performance data for 64KB to 256MB message sizes, representing typical message sizes found in machine learning workloads.

In this scenario Figure 14 presents the speedup of GDR over Non-GDR for NCCL `allreduce` collective<sup>4</sup>. Note that

given that NICs and GPUs are on the same PCIe switch with a single link to the CPU, in the non-GDR case send/rcv flows go through the PCIe link to the CPU causing the bandwidth to be halved. GDR avoids this problem attaining  $2\times$  higher bandwidth and corresponding speedup for large messages. On the other hand, for small message sizes, GDR also has an advantage in terms of latency because it avoids intermediate copy to host/system memory, which explains why we see an even greater speedup in this case.

Thus RDMA transports (e.g. GDR) overcome fundamental limitations in host-accelerator connectivity and are important for enabling efficient scaling on accelerator-centric fabric. They are also often used to build higher level abstractions that may be exposed in DL frameworks [5], [12], [25], [41].

### C. Transport- and Topology-aware Collectives

Recall that to support both asynchronous and synchronous training with data- and model-parallelism we rely on the `allreduce` and `alltoall` communication primitives. These primitives have varying characteristics depending on size of data being communicated, the number and topology of nodes involved in the communication.

For both `allreduce` and `alltoall`, the data-sizes of interest vary from a few KBs to several MBs. Typically the smaller data sizes are more sensitive to latency and as the data size increases bandwidth becomes more important. Also, unlike the collective communication requirements from `allreduce`, in the `alltoall` all nodes need to communicate with all other nodes and thus, the average hop count of the topology has a significant impact on overall performance.

In short, collective implementations need to be aware of (a) network transport (GDR/RDMA vs. TCP), and (b) network topology. For instance, Message Passing Interface (MPI) and NVIDIA Collective Communication Library (NCCL), implement topology- and transport-aware collectives today.

Future scale-out systems need to be co-designed taking into account all of these considerations, in particular including fabric topology, transport, and optimized collectives.

<sup>2</sup>On GPU systems, such as NVIDIA's DGX-1, one 1x16 PCIe link to the host is shared by three devices each driving 1x16 PCIe lanes. This PCIe over-subscription on the host PCIe link results in significant overheads when moving data to/from system memory.

<sup>3</sup>We point out that alternate host kernel based transports, such as TCP/IP over Ethernet incur two copies.

<sup>4</sup>NCCL `all2all` collective is not yet available



## VI. RELATED WORK

Given the growing importance of distributed training of deep learning models, there has been a number of papers on workload characterization and scale-out solutions [14], [22], [35], including Alibaba’s Platform of Artificial Intelligence (PAI) [46] and EFLOPS [20], which detail algorithm and system co-design for high performance distributed training platforms. Other works such as DaDianNao [13] and ScaleDeep [45], also showcase similar results. We provide an overview of some of the well established and more recent hardware platforms that have tried to address the corresponding challenges, with emphasis on their scale-out approach and communication.

The NVidia GPUs have been commonly used for deep learning training, and are often organized into the DGX appliance. DGX-1 consists of 8 GPUs that are interconnected with high-bandwidth NVlink. DGX-2 was announced and provides 16 GPUs that are interconnected not only with NVlink for high-bandwidth but also NVSwitch to provide connectivity between all 16-GPUs. Further, Infiniband may be used to scale beyond a single DGX-1 or DGX-2 system. We point out that NVidia DGX Pod is based on DGX-1 system.

The Google TPU Pod [28], [48] (v2 and v3) is an example of a DNN supercomputer that is tightly integrated with custom accelerator (TPU), high-bandwidth memory and a custom interconnect to enable a supercomputing pod. The TPUs are interconnected together with a high-speed link that supports a custom protocol, instead of commonly available communication protocol, to reduce communication overhead and an integrated router is supported in each TPU [11]. While the system is organized hierarchically with a board consisting of 4 TPUs and a rack consisting of multiple boards, the global topology to scale-out is a low-radix 2D toroidal mesh.

The Habana Gaudi training processors [36] provide integrated compute and networking by supporting RoCE within the training processor. Each Gaudi has 10 ports of 100Gb Ethernet and each port can be used for either internal or external (scale-out) connectivity. The HLS-1 system with 8 Gaudi leverages 7 ports for fully connected topology within the server while the remaining 3 ports are used to scale-out.

On the other hand, Cerebras has proposed wafer-scale training system [34] and built one of the largest chips ever created with over 1 trillion transistors. By integrating all the cores in a single wafer, it enables over 100 Pbit/s of fabric bandwidth and 8 PByte/s of memory bandwidth. The cores (or “chips”) within the wafer are interconnected through a 2D mesh topology. While wafer-scale provides significant compute and bandwidth (both memory and interconnect), it presents some unique challenges from wafer-scale computing – including yield, power/thermal challenges, as well as packaging constraints.

Huawei has also recently announced its Ascend 910 [33] training solution. While some of the details are not clear, 8 Ascend 910 chips are integrated into a single “AI server,” similar to a NVidia DGX-1, with a proprietary high-speed link technology (HCCS) providing 720 Gbps of bandwidth

for NUMA connections within the AI server. In addition, 200 Gbps of bandwidth is provided through RoCE interface for scale-out to create an Ascend cluster with 1K to 2K nodes.

Finally, Intel announced Spring Crest Deep Learning Accelerator for training [47] to enable scalable deep learning. Each Spring Crest provides 64 lanes of SerDes for a total aggregate bandwidth of 3.58 Tbps and each chip also has a fully programmable router that enables multiple “glueless” topologies – i.e., a server consisting of 8 Spring Crest can be directly interconnected to another server and can scale up to 1K nodes.

## VII. SUMMARY

In this paper we have discussed different aspects of training of DLRMs. We have touched on **asynchronous and synchronous training, including the mapping of MLPs and embeddings onto data- and model-parallel patterns of computation**. Further, we have outlined how data- and model-parallelism patterns map to `allreduce` and `all2all` communication primitives, respectively. Also, we have shown how the performance of these primitives is influenced by fabric topology and interconnect design.

We have described the design of the Zion scale-up system and how it can be used for training of DLRMs. We have also reviewed and compared the organization of some of the existing scale-out systems. Finally, we have outlined several important considerations for future scale-out systems, including fabric organization, network transport and topology-aware communication primitives.

## REFERENCES

- [1] “NVidia DGX Pod,” 2018. [Online]. Available: <https://www.nvidia.com/en-us/data-center/dgx-pod-reference-architecture/>
- [2] “NVidia GPUDirect RDMA,” 2019. [Online]. Available: <https://docs.nvidia.com/cuda/gpudirect-rdma/index.html>
- [3] “Open Accelerator Infrastructure (OAI),” 2019. [Online]. Available: <https://www.opencompute.org/wiki/Server/OAI>
- [4] “Parallel distributed deep learning: Machine learning framework from industrial practice,” 2019.
- [5] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattemberg, M. Wicke, Y. Yu, and X. Zheng, “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015.
- [6] A. Agarwal, “Limits on interconnection network performance,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 2, no. 4, pp. 398–412, 1991.
- [7] M. Al-Fares, A. Loukissas, and A. Vahdat, “A scalable, commodity data center network architecture,” in *Proc. 32nd Int. Symp. Computer Architecture (ISCA)*, 2005.
- [8] F. Borisjuk, A. Gordo, and V. Sivakumar, “Rosetta: Large scale system for text detection and recognition in images,” in *Proc. 24th Int. Conf. Knowledge Discovery & Data Mining (KDD)*, 2018, pp. 71–79.
- [9] L. Bottou, F. E. Curtis, and J. Nocedal, “Optimization methods for large-scale machine learning,” *CoRR*, vol. 1606.04838, 2016.
- [10] B. Brock, Y. Chen, J. Yan, J. D. Owens, A. Bulu, and K. Yelick, “RDMA vs. RPC for implementing distributed data structures,” *CoRR*, vol. 1910.02158, 2019.
- [11] C. Chao and B. Saeta, “Cloud TPU: Codesigning architecture and infrastructure,” in *Hot Chips 31 Symposium, Palo Alto, CA, USA*, 2019.

- [12] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang, "MXNet: A flexible and efficient machine learning library for heterogeneous distributed systems," *CoRR*, vol. 1512.01274, 2015.
- [13] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam, "DaDianNao: A machine-learning supercomputer," in *Proc. 47th Annual IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, 2014, pp. 609–622.
- [14] T. Chilimbi, Y. Suzue, J. Apacible, and K. Kalyanaraman, "Project Adam: Building an efficient and scalable deep learning training system," in *Proc. 11th Symp. Operating Systems Design and Implementation (OSDI 14)*, 2014, pp. 571–582.
- [15] P. Covington, J. Adams, and E. Sargin, "Deep neural networks for youtube recommendations," in *Proc. 10th ACM Conf. Recommender Systems*, 2016, pp. 191–198.
- [16] W. J. Dally, "Performance analysis of k-ary n-cube interconnection networks," *IEEE Transactions on Computers*, vol. 39, no. 6, pp. 775–785, 1990.
- [17] W. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003.
- [18] D. Das, S. Avancha, D. Mudigere, K. Vaidynathan, S. Sridharan, D. Kalamkar, B. Kaul, and P. Dubey, "Distributed deep learning using synchronous stochastic gradient descent," *CoRR*, vol. 1602.06709, 2016.
- [19] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. A. Ranzato, A. Senior, P. Tucker, K. Yang, Q. V. Le, and A. Y. Ng, "Large scale distributed deep networks," *Advances in Neural Information Processing Systems (NIPS)* 25, pp. 1223–1231, 2012.
- [20] J. Dong, Z. Cao, T. Zhang, J. Ye, S. Wang, F. Feng, L. Zhao, X. Liu, L. Song, L. Peng, Y. Guo, X. Jiang, L. Tang, Y. Du, Y. Zhang, P. Pan, and Y. Xie, "EFLOPS: Algorithm and system co-design for a high performance distributed training platform," *Proc. IEEE Int. Symp. High-Performance Computer Architecture (HPCA)*, 2020.
- [21] D. Foley and J. Danskin, "Ultra-performance Pascal GPU and NVLink interconnect," *Proc. IEEE/ACM Int. Symposium on Microarchitecture (MICRO)*, vol. 37, pp. 7–17, 2017.
- [22] A. Gibiansky, "Bringing HPC techniques to deep learning," *Baidu technical blog*, 2017.
- [23] U. Gupta, X. Wang, M. Naumov, C. Wu, B. Reagen, D. Brooks, B. Cottel, K. M. Hazelwood, B. Jia, H. S. Lee, A. Malevich, D. Mudigere, M. Smelyanskiy, L. Xiong, and X. Zhang, "The architectural implications of facebook's dnn-based personalized recommendation," *CoRR*, vol. 1906.03109, 2019.
- [24] K. Hazelwood, S. Bird, D. Brooks, S. Chintala, U. Diril, D. Dzhulgakov, M. Fawzy, B. Jia, Y. Jia, A. Kalro, J. Law, K. Lee, J. Lu, P. Noordhuis, M. Smelyanskiy, L. Xiong, and X. Wang, "Applied machine learning at Facebook: A datacenter infrastructure perspective," in *Proc. IEEE Int. Symp. High Performance Computer Architecture (HPCA)*, 2018, pp. 620–629.
- [25] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *CoRR*, vol. 1408.5093, 2014.
- [26] B. Jiang, C. Deng, H. Yi, Z. Hu, G. Zhou, Y. Zheng, S. Huang, X. Guo, D. Wang, Y. Song, and et al., "Xdl: An industrial deep learning framework for high-dimensional sparse data," in *Proc. 1st Int. Workshop on Deep Learning Practice for High-Dimensional Sparse Data*, 2019.
- [27] M. Johnson, M. Schuster, Q. V. Le, M. Krikun, Y. Wu, Z. Chen, N. Thorat, F. Viégas, M. Wattenberg, G. Corrado, M. Hughes, and J. Dean, "Google's multilingual neural machine translation system: enabling zero-shot translation," *CoRR*, vol. 1611.04558, 2016.
- [28] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-I. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snellman, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon, "In-datacenter performance analysis of a Tensor Processing Unit," in *Proc. 44th Annual Int. Symp. Computer Architecture (ISCA)*, 2017, pp. 1–12.
- [29] J. Kim, W. J. Dally, S. Scott, and D. Abts, "Technology-driven, highly-scalable dragonfly topology," in *Proc. 35th Annual Int. Symp. Computer Architecture (ISCA)*, 2008, pp. 77–88.
- [30] J. Kim, W. J. Dally, B. Towles, and A. K. Gupta, "Microarchitecture of a high radix router," in *Proc. 32nd Int. Symp. Computer Architecture (ISCA)*, 2005.
- [31] K. Lee, "Introducing Big Basin: Our next-generation AI hardware," 2019. [Online]. Available: <https://engineering.fb.com/data-center-engineering/introducing-big-basin-our-next-generation-ai-hardware>
- [32] A. Li, S. L. Song, J. Chen, J. Li, X. Liu, N. Tallent, and K. Barker, "Evaluating modern GPU interconnect: PCIe, NVLink, NV-SLI, NVSwitch and GPUDirect," *CoRR*, vol. 1903.04611, 2019.
- [33] H. Liao, J. Tu, J. Xia, and X. Zhou, "DaVinci: A scalable architecture for neural network computing," in *Hot Chips 31 Symposium, Palo Alto, CA, USA*, 2019.
- [34] S. Lie, "Wafer scale deep learning," in *Hot Chips 31 Symposium, Palo Alto, CA, USA*, 2019.
- [35] R. Mayer and H.-A. Jacobsen, "Scalable deep learning on distributed infrastructures: Challenges, techniques and tools," *ACM Computing Surveys*, vol. 53, 2019.
- [36] E. Medina, "Habana labs approach to scaling AI training," in *Hot Chips 31 Symposium, Palo Alto, CA, USA*, 2019.
- [37] R. Mittal, A. Shpiner, A. Panda, E. Zahavi, A. Krishnamurthy, S. Ratnasamy, and S. Shenker, "Revisiting network support for RDMA," *CoRR*, vol. 1806.08159, 2018.
- [38] 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. Malevich, 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. 1906.00091, 2019. [Online]. Available: <https://github.com/facebookresearch/dlrm>
- [39] X. Pan, J. Chen, R. Monga, S. Bengio, and R. Jozefowicz, "Revisiting distributed synchronous SGD," 2017.
- [40] J. Park, M. Naumov, P. Basu, S. Deng, A. Kalaiah, D. S. Khudia, J. Law, P. Malani, A. Malevich, N. Satish, J. Pino, M. Schatz, A. Sidorov, V. Sivakumar, A. Tulloch, X. Wang, Y. Wu, H. Yuen, U. Diril, D. Dzhulgakov, K. M. Hazelwood, B. Jia, Y. Jia, L. Qiao, V. Rao, N. Rotem, S. Yoo, and M. Smelyanskiy, "Deep learning inference in facebook data centers: Characterization, performance optimizations and hardware implications," *CoRR*, vol. 1811.09886, 2018.
- [41] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in PyTorch," 2017. [Online]. Available: <https://pytorch.org/>
- [42] A. Sergeev and M. D. Balso, "Horovod: fast and easy distributed deep learning in TensorFlow," *CoRR*, vol. 1802.05799, 2018.
- [43] T. Shanley, *Infiniband Network Architecture*. Addison-Wesley, 2002.
- [44] M. Smelyanskiy, "Zion: Facebook next-generation large-memory unified training platform," in *Hot Chips 31 Symposium, Palo Alto, CA, USA*, 2019.
- [45] S. Venkataramani, A. Ranjan, S. Banerjee, D. Das, S. Avancha, A. Jagannathan, A. Durg, D. Nagaraj, B. Kaul, P. Dubey, and A. Raghunathan, "ScaleDeep: A scalable compute architecture for learning and evaluating deep networks," in *Proc. 44th Annual Int. Symp. Computer Architecture (ISCA)*, 2017, pp. 13–26.
- [46] M. Wang, C. Meng, G. Long, C. Wu, J. Yang, W. Lin, and Y. Jia, "Characterizing deep learning training workloads on Alibaba-PAI," *CoRR*, vol. 1910.05930, 2019.
- [47] A. Yang, N. Garegrat, C. Miao, and K. Vaidyanathan, "Deep learning training at scale: Spring Crest deep learning accelerator," in *Hot Chips 31 Symposium, Palo Alto, CA, USA*, 2019.
- [48] C. Young, "Evaluation of the Tensor Processing Unit: A deep neural network accelerator for the datacenter," in *Hot Chips 29 Symposium, Palo Alto, CA, USA*, 2017.
- [49] W. Zhao, "OCP Accelerator Module (OAM)," 2019. [Online]. Available: <https://engineering.fb.com/data-center-engineering/accelerator-modules/>
- [50] Q. Zheng, B.-Y. Su, J. Yang, A. Azzolini, Q. Wu, O. Jin, S. Karandikar, H. Lupesko, L. Xiong, and E. Zhou, "Shadowsync: Performing synchronization in the background for highly scalable distributed training," *CoRR*, vol. 2003.03477, 2020.
- [51] M. Zinkevich, M. Weimer, L. Li, and A. J. Smola, "Parallelized stochastic gradient descent," in *Advances in Neural Information Processing Systems (NIPS)* 23. Curran Associates, Inc., 2010, pp. 2595–2603.