ASTRA-sim2.0: Modeling Hierarchical Networks and Disaggregated Systems for Large-model Training at Scale

William Won*§, Taekyung Heo*§, Saeed Rashidi*§, Srinivas Sridharan†, Sudarshan Srinivasan‡, Tushar Krishna*

*Georgia Institute of Technology, Atlanta, GA, USA
†Meta, Menlo Park, CA, USA
‡Intel, Bangalore, Karnataka, India
*{william.won, taekyung, saeed.rashidi}@gatech.edu, tushar@ece.gatech.edu
†ssrinivas@fb.com ‡sudarshan.srinivasan@intel.com

Abstract—As deep learning models and input data are scaling at an unprecedented rate, it is inevitable to move towards distributed training platforms to fit the model and increase training throughput. State-of-the-art approaches and techniques, such as wafer-scale nodes, multi-dimensional network topologies, disaggregated memory systems, and parallelization strategies, have been actively adopted by emerging distributed training systems. This results in a complex SW/HW co-design stack of distributed training, necessitating a modeling/simulation infrastructure for design-space exploration. In this paper, we extend the open-source ASTRA-sim infrastructure and endow it with the capabilities to model state-of-the-art and emerging distributed training models and platforms. More specifically, (i) we enable ASTRA-sim to support arbitrary model parallelization strategies via a graph-based training-loop implementation, (ii) we implement a parameterizable multi-dimensional heterogeneous topology generation infrastructure with analytical performance estimates enabling simulating target systems at scale, and (iii) we enhance the memory system modeling to support accurate modeling of in-network collective communication and disaggregated memory systems. With such capabilities, we run comprehensive case studies targeting emerging distributed models and platforms. This infrastructure lets system designers swiftly traverse the complex co-design stack and give meaningful insights when designing and deploying distributed training platforms at scale.

Index Terms—Distributed training, High-performance training, Multi-dimensional network, Disaggregated memory system

I. Introduction

The rapid growth in computation and memory requirement for Deep Neural Network (DNN) models is far greater than the performance and capacity scale of a single Neural Processing Unit (NPU, such as GPU or TPU). As an example, going from BERT [1] model to GPT-3 [2], over the course of two years, requires $1800\times$ more computation to train the model [3]. We have now reached the era of trillion parameter models [4] that require 10's of terabytes of memory and zeta floating point operations to train a model [3], [4]. Despite efforts to reduce the overhead of large model training on the workload side [5], big model training still remains challenging from the

systems perspective [6]. Hence, distributed training is an inevitable option to keep up with the pace of increased resource requirements of DNN and Deep Learning (DL) training.

Designing an efficient distributed training system is challenging as there are many design choices such as parallelization strategies, NPU performance, NPU memory bandwidth, network topology, network bandwidth, and scheduling policies. Moreover, these design choices are interdependent, requiring the co-design of hardware and software for training platforms. ASTRA-sim [7], [8] is an existing open-source infrastructure (originally developed by Georgia Tech, Intel and Meta). ASTRA-sim aims to model the complete SW/HW co-design stack of distributed training systems, shown in Fig. 1(a). It captures different aspects of distributed training platforms via three abstraction layers: (i) workload, (ii) system, and (iii) network. The workload layer implements the training loop (i.e., the DNN model, its parallelization strategy - data-parallel, model-parallel, etc., compute/communication ordering). The system layer provides various collective communication algorithm implementations (e.g., All-Reduce, All-to-All) and also manages pipelining and scheduling of communication operations. Finally, the networking layer models the HW/SW components of the network and simulates the traffic issued by the system layer.

ASTRA-sim is a promising tool for exploring the design space of distributed training systems and has been leveraged by several recent works [9]–[14]. However, in this work, we identify limitations in ASTRA-sim that restrict it from supporting arbitrary parallelism strategies, networks, and memory models. This comes from the rapidly changing SW/HW landscapes for DNN training as we describe next.

On the software end, there has been a growing interest in new parallelism strategies, both hand-designed such as 3D-parallelism [15], [16], FSDP [6], ZeRO [17], expert parallelism [18] and discovered [19], [20]. These strategies enable the training of large models, splitting datasets, parameters and optimizer state, while optimizing for communication [21], [22]. ASTRA-sim did not have a strong motivation to support arbitrary parallelism when it was proposed as there were a

§These authors contributed equally to this work.

handful of parallelism strategies such as data parallelism [23], model parallelism, and hybrid [24].

The hardware landscape for distributed training has been evolving rapidly as well. State-of-the-art systems extensively deploy multi-dimensional network topologies with hierarchical bandwidths to interconnect NPUs [25]–[29]. This is because increasing the aggregated network BW per NPU through a single dimension is fundamentally limited by the link technology the network is leveraging (e.g., current NVLink [30] offers up to 450 GB/s). Naively scaling out through NIC is also not practical due to engineering limitations such as dollar-cost, power, and thermal problems. Meanwhile, wafer-scale systems [31], [32] tackle the communication problem by fabricating NPU chiplets on a large-wafer with low-dimensional, high on-chip networking, then scaling out such wafers using NICs. In order to study these technology-driven network landscapes, there is a need for a mechanism to represent and study arbitrary multi-dimensional topologies at scale, with different shapes and BW configurations. ASTRA-sim natively uses the Garnet simulator [33] from gem5 as its network layer, which has limitations in modeling such platforms.

Memory disaggregation, which allows GPUs to access a larger remote memory pool, is another promising HW solution to overcome the limited GPU memory capacity per node. Although the concept has been studied for several decades [34]–[41], the network and memory did not support memory disaggregation. Motivated by the need for memory disaggregation, the computing industry is now building a framework with a new network technology, compute express link (CXL) [42]. As distributed training systems will benefit from disaggregated memory, there is a strong need for exploring this design space. ASTRA-sim uses a simple BW number to model memory and cannot capture this complex design-space.

In this work, we address the aforementioned limitations of ASTRA-sim and enhance it via three novel features, as shown in Fig. 1(b)-(d): (i) arbitrary parallelism support, (ii) hierarchical network support, and (iii) memory model support. We add arbitrary parallelism support by encoding parallelism strategies as execution traces and developing a parser to translate these into compute and communication tasks with dependencies. For network support, we developed a taxonomy to define hierarchical topologies and created an analytical model to estimate performance when running a topology-aware collective over the physical topology. For the memory models, we augment ASTRA-sim with the ability to model local (e.g., HBM) and networked remote (pooled) memories.

Using these enhancements, we present case studies to deliver key insights about future platforms. We compared conventional multi-dimensional and wafer-scale systems and found that with appropriate collective scheduling and parallelization strategy designs, conventional systems can match wafer-scale systems' performance, whereas wafer-scale shows up to $2.51\times$ better collective time when scaled. We also compared disaggregated memory architectures and found that communication time dominates in training a Mixture-of-Experts (MoE) model, and identify configurations that can hide

communication time to provide 4.6x speedup over a baseline Zero-Infinity [43].

II. BACKGROUND

A. Distributed Training

❶ **Synchronous/Asynchronous Training.** When models/data are distributed across NPUs, it is crucial to decide when and how to synchronize such distributed information across them. The asynchronous training approach, as the name suggests, communicates among NPUs in an asynchronous manner. Therefore, asynchronous training suffers from the convergence problem [10] and is more complex to implement and maintain [44]. Therefore, the most common approach is synchronous distributed training. In this mechanism, all nodes work on their own data and synchronize the distributed information altogether before proceeding to the next iteration, usually in the form of collective communications [7], [22].

❷ **Parallelization Strategy.** Each parameter, including model weights and input data, is distributed across NPUs. How each parameter is sharded and distributed is dictated by its ruling parallelization strategy [7]. The three most pervasive parallelism strategies are: data-parallel (DP), model-parallel (MP), and pipeline-parallel (PP). DP distributes mini-batch across NPUs and synchronizes weight gradients during the backward pass [7], [44]. MP, on the other hand, distributes a model evenly across NPUs and communicates forward activation and input gradients pass [44]. MP and DP are orthogonal patterns, therefore MP and DP can be used simultaneously, called hybrid-parallel scheme [24]. PP distributes model layers across nodes and processes micro-batches in a pipelined manner [45], [46]. Other parallelization strategies are also actively being investigated [17], [43], [47].

❸ **Training Loop.** In addition to parallelization, the order of communication and computation must be clearly defined to execute distributed training. Such computation and communication ordering information is named a training loop [7].

B. Collective Communication

❶ **Collective Communication.** Depending on the parallelization strategy, models and/or input batches are distributed across NPUs. Therefore, it is unavoidable that devices should communicate and synchronize data, such as forward activation or weight/input gradients [9]. This traffic is commonly formulated and processed in the form of collective communications. Some common collective patterns in distributed training are shown in Fig. 2. With a synchronous training approach, the most pervasive collective pattern is All-Reduce [48], which could be logically viewed as Reduce-Scatter followed by an All-Gather.

❷ **Hierarchical Collective Algorithm.** There exist several basic topology-aware collective communication algorithms to execute these communication patterns. A handful of examples of basic topology-aware All-Reduce collective algorithms include Ring-based [49], Tree-based [50], and Halving-Doubling [51]. However, when the underlying network topology is multi-dimensional, such basic algorithms would not
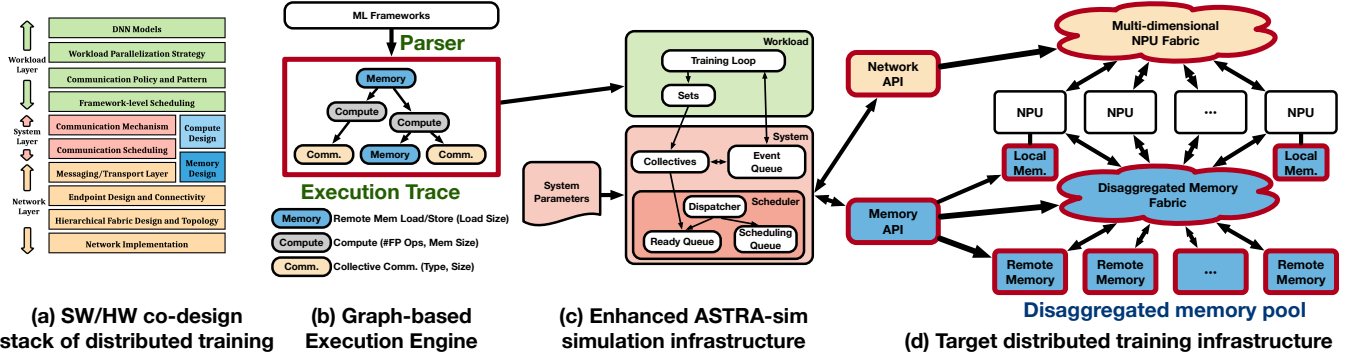
Fig. 1: Overview of Proposed Infrastructure for Modeling Next-generation Training platforms. The components extended in ASTRA-sim 2.0 from the original ASTRA-sim to model emerging platforms are marked in bold.
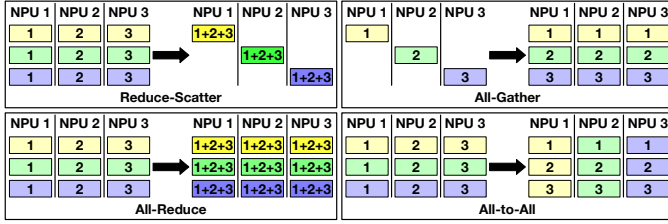

Fig. 2: Definition of Reduce-Scatter, All-Gather, All-Reduce, and All-to-All collective communication patterns.

perform optimally as the logical topology each algorithm assumes mismatches the physical one. In order to mitigate such an effect, multi-rail hierarchical collective algorithms have been proposed [52]. Using this scheme, in order to run an All-Reduce collective on an N-dimensional topology:

- Run Reduce-Scatter in ascending order from Dim 1, then Dim 2, $\cdots$, up to Dim N.
- Run All-Gather in descending order from Dim N, $\cdots$, Dim 2, down to Dim 1.

C. ASTRA-sim

Vast design choices of distributed training shown in Sec. II-A, combined with diverse hardware configurations create an enormous SW/HW design space of distributed training as depicted in Fig. 1(a). Such enormous design space cannot be solely explored by only leveraging physical systems, especially at scale. Therefore, a simulation-based mechanism to quickly model and profile distributed training platforms is necessary for design-space exploration. ASTRA-sim [7] is a distributed training simulation framework to exactly address this demand. ASTRA-sim captures the training configuration explained in Sec. II-A Its high-level components are summarized in Fig. 1(c). It codifies the complex SW/HW search space across three abstraction layers. The workload layer lets the user describe and define target DNN models, target parallelization strategies, and training loops. The system layer implements collective communication algorithms, schedules compute and communication operations, and manages compute-communication overlap. Compute times are fed in via external NPU models [53] or real system measurements. Communication times are computed using a network simula-

tor. The default simulator is Garnet [33] from gem5. It reports detailed system and network-level behaviors as well as end-to-end training throughput.

III. MOTIVATION

Even though ASTRA-sim framework has allowed brisk navigation of distributed training search space [9], [10], the tool as-is does not meet the demand to capture more complex target platforms. In this section, we motivate the need to extend the ASTRA-sim toolchain to enable modeling state-of-the-art and futuristic training systems. Specifically, we identify three emerging requirements for ASTRA-sim as shown below.

- Ability to model arbitrary parallelisms
- Ability to model multi-dimensional hierarchical networks
- Ability to model memory systems

A. Ability to Model Arbitrary Parallelism Strategies

One of the major limitations of ASTRA-sim is the limited parallelism support. The original ASTRA-sim cannot support complex parallelization strategies such as pipeline parallelism [45], [46] and 3D parallelism [16]. There are two reasons for the limitation. First, ASTRA-sim assumes that all NPUs perform the same operation at the same time. While this assumption saves the engineering overhead in implementing data parallelism and model parallelism, it does not allow pipeline parallelism as it requires executing different operations on each NPU at the same time. Second, parallelization strategies are tightly coupled with the frontend implementation of ASTRA-sim and implemented as separate training loops in the workload layer. Parallelization strategies for distributed training are an active area of research [6], [45], [46], [54]–[56], and sometimes several strategies are jointly applied [18]. Therefore, to evaluate arbitrary parallelism strategies, it is critical to decouple parallelization strategies from the ASTRA-sim implementation.

B. Ability to Model Multi-dimensional Networks

From the necessity to distribute and synchronize models and data across devices, large-scale distributed training is usually communication-bound [21], [22], [48]. Therefore, in order to maximize training performance, state-of-the-art systems mix and match a plethora of networking technologies.

This usually ends up in a system having multi-dimensional network topologies with heterogeneous bandwidth configurations [9]. As an instance, NVIDIA DGX-A100 [26] exploits a 2-dimensional network topology whose first dimension is NVIDIA NVLink [30] then scaled-out using InfiniBand [57] or Ethernet [58], [59] technologies. The Google Cloud TPUv4 [27] leverages a 3D Torus where each inter-core interconnect runs at 448 Gb/s [60].

Although ASTRA-sim can, in principle, target multi-dimensional networks, it only supports a limited set of pre-defined network topologies – 2D and 3D torus. In order to study different topologies, one must implement both a new network topology in Garnet and its corresponding topology-aware hierarchical collective algorithm, which significantly drags ASTRA-sim's strength of swift distributed training system modeling and performance analysis.

Therefore, it is necessitated to attach a more powerful network backend to the ASTRA-sim framework for rapid design-space exploration of state-of-the-art and futuristic training platforms. It must define a systematic mechanism to represent arbitrary multi-dimensional network topologies at scale. With such notation, the user cam swiftly represent an arbitrary multi-dimensional networks, instead of manually implementing network topology files and their corresponding collective communication algorithms.

C. Ability to Model Emerging Memory Systems

As DNN model parameters have to be loaded from and stored back to memory, having an efficient memory system is critical in distributed training. To design an efficient memory system, exploring the memory system design space is essential. However, as the original ASTRA-sim does not have detailed memory models, it limits the opportunity to explore the design space. We find that ASTRA-sim should support the following three features. The first feature is the ability to model local HBM memory. ASTRA-sim should have a local memory model that allows how the performance changes as HBM latency and bandwidth vary. This feature allows system and architecture designers to find the optimal local HBM configuration within the same budget. The second feature is the support for memory disaggregation. It is well known that the limited capacity of GPUs is the major bottleneck in large-model training. Model parallelism [44] and memory optimizations [17], [43], [47] have been widely adopted to overcome the limitation. While the proposed solutions have been effective in reducing per-GPU memory footprint, they come with critical limitations such as increased computation and communication time. Memory disaggregation is a fundamental solution to overcome the NPU memory capacity limitation by allowing NPUs to access a larger remote memory pool. Emerging interconnects such as CXL [42] accelerate this trend. ASTRA-sim should be able to answer research questions such as the optimal configurations and design for memory disaggregation. The last feature is in-switch collective communication support. With the introduction of memory disaggregation, network switches are introduced in the memory access

```
eg = None
if args.eg:
    eg_file = f"{out_file_prefix}_eg.json"
    eg = ExecutionGraphObserver()
    eg.register_callback(eg_file)
    eg.start()
...
if eg:
    eg.stop()
    eg.unregister_callback()
```

Snippet 1: Execution trace collection example [65].

path of training systems. Performing collective communication in switches is an attractive option to improve the performance of distributed training by reducing communication time [21], [22], [61]–[64]. To find out the performance benefit and trade-offs of in-switch collective communication, ASTRA-sim should support in-switch collective communication.

IV. EXTENSIONS TO ASTRA-SIM

In this section, we introduce the new features we added to ASTRA-sim and describe how they are implemented. All extensions are released and publicly accessible in the ASTRA-sim repository[1].

A. Graph-based Execution Engine

To support arbitrary parallelization strategies, we replace the frontend of ASTRA-sim with a graph-based execution engine. The graph-based execution engine decouples parallelization strategies from the frontend implementation. As the name implies, the graph-based execution engine works on input graphs. The input graphs encode the execution of ML models and their associated parallelization strategies, which can be generated from ML frameworks such as PyTorch [66], TensorFlow [67], and FlexFlow [19]. Code snippet 1 presents how graphs can be collected with PyTorch. PyTorch offers a seamless option for collecting such graphs, which does not require any modifications to the model. The collected graphs are named execution traces (ETs). ETs are fed into the frontend, and the execution engine is responsible for simulating a distributed training system. ETs encode critical information for simulation such as memory access, computation, and communication. Each operation is modeled as a node, and their dependencies are presented as edges as shown in Fig. 1(b). In ETs, parallelization strategies are encoded with dependencies. As each NPU has an independent graph-based execution engine, each NPU can run different operations. The engine consumes nodes one by one, and the dependent nodes become ready to be issued when all of their parent nodes are completed. Nodes are completed after a specific delay, and the delay is determined by the node type and metadata. The execution engine continues the simulation until it consumes all nodes.

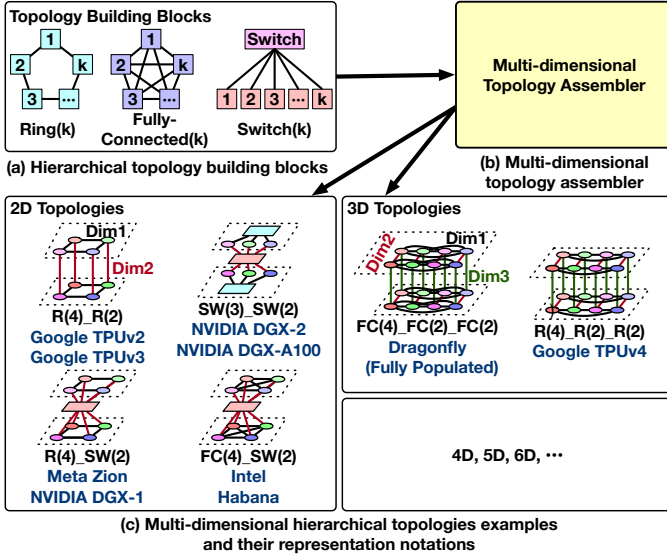[1]https://github.com/astra-sim/astra-sim

Fig. 3: (a) Hierarchical topology building blocks: Ring, Fully-Connected, and Switch (b) Multi-dimensional network topologies are created by stacking up network building blocks (c) Multi-dimensional hierarchical topology examples, their shape notations, and corresponding distributed training framework.

We define a common format for execution traces, called ASTRA-sim ET, to avoid implementing ET parsers for all different ET types in ASTRA-sim. Instead, we provide a converter from any ET (e.g., PyTorch ET) to ASTRA-sim ET[2]. ASTRA-sim ET has three node types: compute, memory, and communication as presented in Fig. 1(b). Each node has metadata that is critical for simulating the operation. Compute nodes have the tensor size and the number of floating point operations to perform computation. ASTRA-sim calculates the number of cycles to perform the operation with an internal roofline model. Memory nodes measure the number of cycles to store or load a tensor. Therefore, the nodes have a tensor size as metadata. Communication nodes encode the communication type (collective communication between NPUs or peer-to-peer communication between a pair of NPUs) and the communication size. This information gets translated into a network delay by the underlying system and network layers of ASTRA-sim.

B. Multi-dimensional Network Representation

In order for users to quickly target arbitrary multi-dimensional network topologies, it is crucial to design a generic notation to represent such multi-dimensional shapes. In this paper, we propose a taxonomy that constructs a multi-dimensional topology by stacking up network building blocks in a hierarchical manner. Fig. 3(a) shows the three network building blocks utilized in this paper: Ring (R), FullyConnected (FC), and Switch (SW). Ring(k) connects k NPUs in a ring shape (i.e., two connections per every NPU). FullyConnected(k), on the other hand, offers all-to-all connectivity among all pairs of NPUs. Finally, Switch(k)

TABLE I: Network building blocks and its corresponding topology-aware collective communication algorithm [68].

Network Building Block	Topology-aware Collective
Ring	Ring [49]
FullyConnected	Direct [51]
Switch	HalvingDoubling [51]

connects all k NPUs using an external switch fabric. We chose these three as the network building blocks as they have corresponding well-known topology-aware collective algorithms as summarized in Table I[3].

Multi-dimensional topologies can be generated by assembling these blocks in an arbitrary hierarchical manner, as glimpsed in Fig. 3(b). A handful of example constructed topologies are shown in Fig. 3(c). Ring(4)_Ring(2) simply denotes a 2D Torus with 8 NPUs in total, where the first dimension is Ring(4) and two such Dim 1 networks are interconnected using Ring(2) topology. Ring(4)_Switch(2), on the other hand, has the same Dim 1 but planes are being scaled out using an external switch instead. An example 3D topology from Fig. 3(c) is FC(4)_FC(2)_FC(2), a fully-populated DragonFly [70] topology with 16 NPUs. Ring(4)_Ring(2)_Ring(2) is also shown, where the NPU placement is equivalent but topologies connecting them are substituted to Ring, thereby resulting in a 3D torus instead. The number of network dimensions or the building blocks' order is not restricted, thus arbitrary 4D, 5D, ⋯, networks can easily be represented using the same notation. Note that each and every example topologies listed in Fig. 3(c) corresponds to some state-of-the-art distributed training platforms, demonstrating the power of our proposed representation in modeling the design space.

With this representation, designing a multi-dimensional topology-aware collective is straightforward and requires minimal modification. As explained in Sec. II-B, multi-rail hierarchical collective algorithms can be run by iteratively running the basic topology-aware collective algorithm on each dimension. Recall that we deliberately chose network building blocks that have known congestion-free collective algorithms. The corresponding topology-aware collective algorithms are listed in Table I. Consequently, collective communications on any arbitrary multi-dimensional network can be run by running these basic algorithms in order and requires no further modification.

C. Analytical Network Backend

Supporting arbitrary multi-dimensional network topologies shown in Sec. IV-B, we implemented a new analytical network backend[4] and ported it to the ASTRA-sim framework. The following points summarize why an analytical equation-based network was sufficient for our purpose:

- There is a need for first-order design-space exploration (topology shape and BW) of the target system at scale.

---

[2]Currently, PyTorch and FlexFlow are supported.

[3]Even if the underlying system uses other topologies, they are logically reduced into one of these building blocks due to the collective communication library [68], [69]. This is a unique feature of DL training platforms.
[4]https://github.com/astra-sim/analytical

```
sim_schedule(delta, callback)
sim_send(msg_size, dest, callback)
sim_recv(msg_size, src, callback)
```

Snippet 2: Abstract view of example ASTRA-sim frontend NetworkAPI methods [12].

- As Garnet is most suitable for modeling network-on-chip targets, it is challenging to easily model arbitrary multi-dimensional network topologies, as discussed in Sec. IV-B.
- Given the scale (1000s of NPUs) of state-of-the-art and futuristic systems and DL models, cycle-level simulation using Garnet is too slow to be practical.
- Multi-dimensional topologies run a topology-aware multi-rail hierarchical collective algorithm, which does not create any network congestion. Thanks to this effect, analytical equation-based modeling shows marginal accuracy change over cycle-accurate simulations, and in fact closely matches real system measurements for a small system, as we show later.

In order to model a communication between two NPUs, ASTRA-sim frontend delegates the network backend to simulate such a communication and requests the backend to invoke a callback function to notify the transmission is completed. This protocol is defined in the form of NetworkAPI (Fig. 1(c)) methods [12]. Several examples of NetworkAPI methods are shown in Snippet 2. Whenever a communication request, such as `sim_send` or `sim_recv` is initiated, the analytical network backend leverages a simple equation to estimate the communication delay instead of simulating actual network behaviors:

$$\text{Time} = (\text{LinkLatency} \times \text{Hops}) + \frac{\text{MessageSize}}{\text{LinkBandwidth}}$$

and simply schedules the callback function to be invoked after this delay, unlike the original Garnet backend which runs packet-level cycle-accurate simulations[5]. Modeling communication with serialization and link delay is suitable when the communication size is relatively large to be bandwidth-bound (e.g., DLRM and Transformer-1T has 100MB–1GB collectives). The analytical equation could be amended to consider other effects, such as wire propagation delay, as desired. For example, complex system and network optimizations (such as remote memory management or in-switch collective communication) can be captured by equations (Sec. IV-D).

Validation. In order to show the accuracy of the analytical network backend, we constructed two real systems and compared various-sized All-Reduce running time. The two real systems leverage NCCL v2.4.6 [68] which consist of 4 and 16 NVIDIA V100 GPUs [71] using a Ring topology with 150 GB/s NVLink [30] among GPUs. Fig. 4 shows the result. We

[5]This approach may have limitations when the network contains non-trivial behaviors, such as network congestion or link oversubscription. Implementing first-order congestion modeling into the analytical backend is our future work.
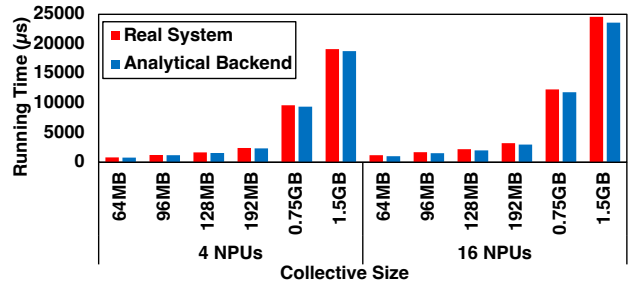

Fig. 4: Analytical network backend validation over real system measurements ranging from 64MB–1.5GB All-Reduce collectives.

run 64 MB – 1.5 GB All-Reduce and the results suggest the mean error of simulation over all configurations is 5%.

Speedup. In order to measure the simulation time improvement, we run a 1MB All-Reduce simulation on a 3D Torus with 64 NPUs (4×4×4). On Garnet-based ASTRA-sim, the simulation took 21.42 minutes to finish. For the same configuration, the analytical backend only spent 1.70 seconds, showing 756× speedup in simulation runtime. Further, the analytical backend supports a 3D Torus with 4K NPUs (16×16×16) in just 3.14 seconds. Nearly three-orders-of-magnitude speedup proves the capabilities of analytical network backend to profile systems of scale at speed.

D. Memory Models

We add a memory API to ASTRA-sim to support various memory models as shown in Fig. 1(d). The goal of memory API is to model various memory systems ranging from local memory to disaggregated memory. Memory API takes tensor location (local or remote), tensor size, memory bandwidth, and memory system design as arguments and returns the number of cycles to load or store a tensor to a memory system. Tensor size and location are encoded in the metadata of ET nodes, and memory bandwidth and system design are given as system configurations. Memory API supports local memory, remote memory, and in-switch collective communication. Memory API determines the model to run based on the tensor location and system parameters.

❶ **Local Memory Model.** This is a simple memory bandwidth model with memory access latency, tensor size, and memory bandwidth as presented in an equation below. Memory access latency and memory bandwidth are given to ASTRA-sim as system parameters, and payload size is encoded in a memory access node of an ET.

$$\begin{aligned}
&(Memory\ Access\ Time) \\
&= (Memory\ \ Access\ Latency) \\
&+ (Tensor\ Size)/(Memory\ Bandwidth)
\end{aligned}$$

❷ **Remote Memory Model.** This model has the ability to calculate the data transfer time with a disaggregated memory system. In addition to the default parameters for the local memory model, this model takes the disaggregated memory design as a parameter. A disaggregated memory can take

(a) Multi-level switches.   (b) Ring.
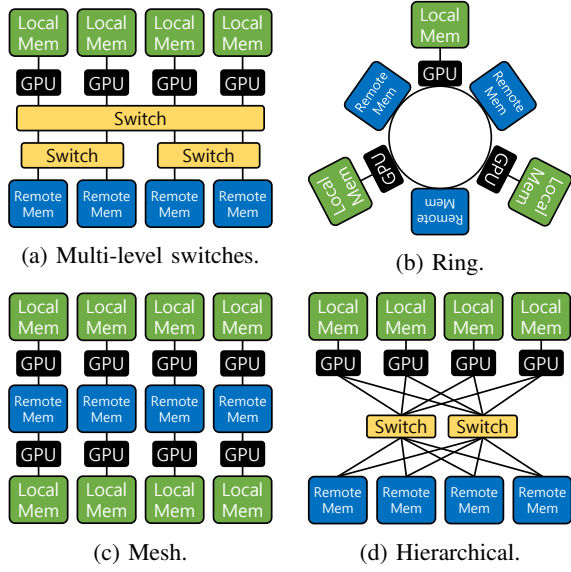
(c) Mesh.   (d) Hierarchical.

Fig. 5: Various memory pool architectures.

any design such as multi-level switches, rings, mesh, and hierarchical as shown in Fig. 5. Different design choices result in different data transfer times because the load on links and the number of network hops change.

The remote memory model calculates the data transfer time for a given disaggregated memory system for given parameters. For ease of explanation, let's assume a system with a hierarchical disaggregated memory. Fig. 6 illustrates how the remote memory model works for the hierarchical disaggregated memory. There are multiple nodes in the system, and each node has multiple pairs of CPU and GPU. CPUs and GPUs are hierarchically connected to out-node switches, and the out-node switches are connected to multiple remote memory groups. Remote memory groups collectively work as a shared memory pool for all CPUs and GPUs. Let's assume that there are 16 nodes with 16 pairs of CPU and GPU in each node. In total, there are 256 CPUs and 256 GPUs. Additionally, we assume that there are four out-node switches and eight remote memory groups.

If each GPU wants to load a tensor of size `W` from the remote memory pool, 256`W` should be loaded from the remote memory pool. As there are eight remote memory modules, each remote memory module will have 32`W`. As there are four out-node switches and each remote memory group is connected to all out-node switches, each link has to transfer 8`W`. The data to transfer on the link between an out-node switch and a node is 4`W` as each node requires 16`W` (the number of GPUs in a node) and four out-node switches will transfer the same amount of data. Once the loads on links are determined, they are transferred in a pipelined manner with the chunk size unit. The chunk size is the basic transfer unit of the network. Fig. 7 demonstrates how tensors are transferred in a pipelined manner. The meaning of notations is described as the following equations. The total data transfer time is the sum of the critical path, and the length of a stage is determined by the max of data transfer time (arrows) in the stage.
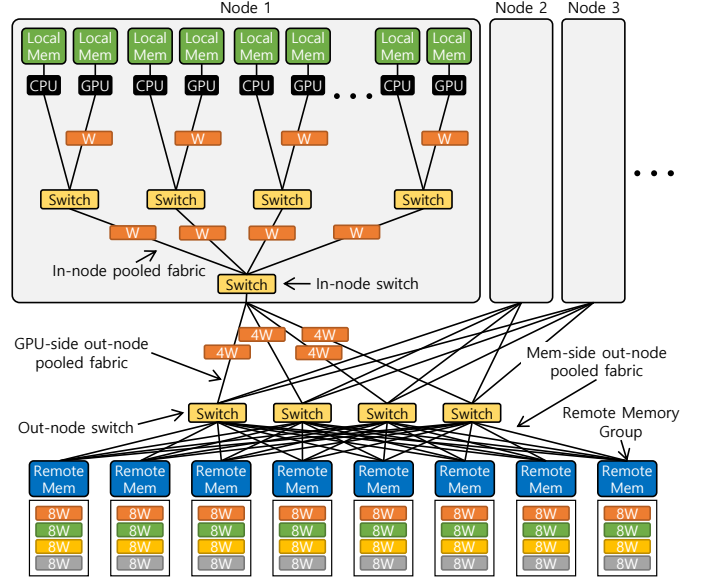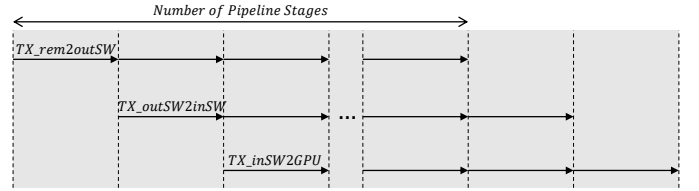

Fig. 6: Remote memory model illustration.

Fig. 7: Pipelined data transfer.

$$(Number\ of\ Pipeline\ Stages)$$
$$= ((Tensor\ Size) \times (Number\ of\ GPUs))$$
$$/(Number\ of\ Remote\ Memory\ Groups)$$
$$/(Number\ of\ Out\text{-}node\ Switches)$$
$$/(Chunk\ Size)$$

$$(TX\_rem2outSW)$$
$$= (Chunk\ Size)/(Mem\text{-}side\ Out\text{-}node\ Fabric\ BW)$$

$$(TX\_outSW2inSW)$$
$$= ((Number\ of\ Remote\ Memory\ Groups) \times (Chunk\ Size))$$
$$/((Number\ of\ Nodes) \times (GPU\text{-}side\ Out\text{-}node\ Fabric\ BW))$$

$$(TX\_inSW2GPU)$$
$$= ((Num\ of\ Rem\ Mem\ Groups) \times (Num\ of\ Out\text{-}node\ SW) \times (Chunk\ Size))$$
$$/((Number\ of\ GPUs) \times (In\text{-}node\ Fabric\ BW))$$

❸ **In-switch Collective Communication.** We support in-switch collective communication with an analytical model. With in-switch collective communication, parameters are gathered while being loaded (All-Gather), and sharded while being stored (Reduce-Scatter). The analytical model for in-switch collective communication is similar to the analytical model for remote memory access. However, the only difference is the data size to transfer for each link as parameters are gathered or scattered. Let's take the same example used for the remote memory model. Fig. 8 illustrates how in-switch collective communication works. In this figure, we assume that each GPU loads a tensor size of `W`. As there are 256 GPUs, the
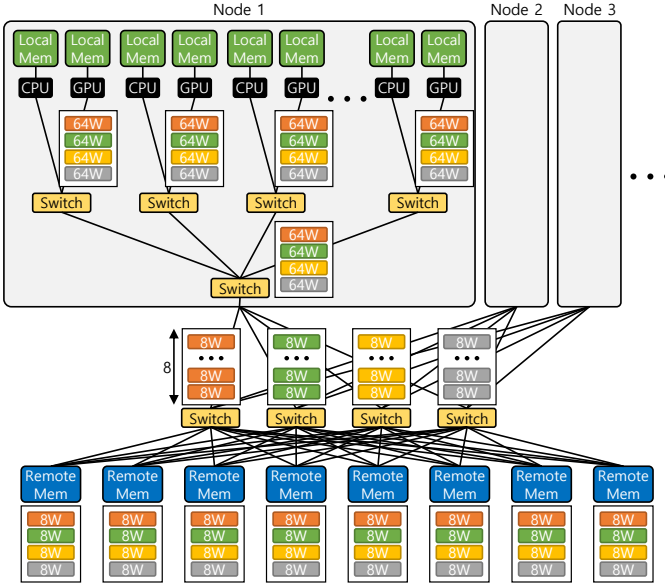
Fig. 8: In-switch collective communication illustration.

total size of tensors to load is 256W. The tensors are sharded into eight remote memory groups, and each remote memory group has 32W. As each remote memory group is connected to four out-node switches, each link transfers 8W. Each out-node switch will have 64W in total because eight remote memory groups transfer 8W for all out-node switches. While receiving the weights, they are gathered. After that, the out-node switches are forwarding 64W to each node. As a result, each in-node switch receives 256W, which is the reconstructed weight. In-node switches are responsible for broadcasting the gathered weights to GPUs. Parameters are transferred in a pipelined manner as shown in the remote memory model. In-switch collective communication changes the equations as below.

$$(TX\_rem2outSW)$$
$$= (Chunk\ Size)/(Mem\text{-}side\ Out\text{-}node\ Fabric\ BW)$$

$$(TX\_outSW2inSW)$$
$$= ((Number\ of\ Remote\ Memory\ Groups) \times (Chunk\ Size))$$
$$/(GPU\text{-}side\ Out\text{-}node\ Fabric\ BW)$$

$$(TX\_inSW2GPU)$$
$$= ((Num\ of\ Rem\ Mem\ Groups) \times (Num\ of\ Out\text{-}node\ SW) \times (Chunk\ Size))$$
$$/(In\text{-}node\ Fabric\ BW)$$

V. CASE STUDIES

In this section, we run comprehensive case studies showcasing the extended capabilities of ASTRA-sim and provide meaningful insights regarding the design space. For all our experiments, we assumed NPU compute power of 234 TFLOPS observed from the measurements of an A100 GPU [26].

A. Conventional System vs Wafer-scale System

Wafer-scale systems feature a number of NPUs (on a wafer) connected using low-dimensional but high-BW on-chip (on-wafer) interconnection networks [31], [32], [72]. Meanwhile,

TABLE II: Target wafer-scale and conventional (multi-dimensional) topologies. Conventional system parameters are borrowed from [25], [26] and wafer-scale params are borrowed from [72], [73].

Topology	Shape	NPU Size	BW (GB/s)
W-1D	Switch	512	350, 500, 600
W-2D	Switch_Switch	32×16	250_250
Conv-3D	Ring_FC_Switch	16×8×4	200_100_50
Conv-4D	Ring_FC_Ring_Switch	2×8×8×4	250_200_100_50

TABLE III: Target training workloads and their characteristics.

Workload	#Params	MP Size	DP Size
DLRM	57M (MLP layers)	1,024	1,024
GPT-3	175B	16	64
Transformer-1T	1T	128	8

conventional systems [26], [28] have multi-dimensional hierarchical topologies with various networking techniques including on-chip, scale-up, and scale-out (NIC). We compare the two distinct approaches by abstracting these systems. Target experimental topologies with 512 NPUs are summarized in Table II. For wafer-scale proxy, we create three 1D topologies with 300, 500, and 600 GB/s on-wafer BW (W-1D), and a 2D topology with 250_250 GB/s BW (W-2D) to model futuristic wafer systems [72], [73]. For conventional systems, we created 3D and 4D topologies (Conv-3D and Conv-4D) using on-chip, scale-up, and scale-out interconnections borrowed from [9], [25], [26]. Target distributed training workloads and their characteristics are also summarized in Table III.

1) Impact of Scheduling: Normalized runtimes of a single 1GB All-Reduce as well as real workloads are shown in Fig. 9. When a topology is multi-dimensional, complex behaviors like pipelining bubbles or unbalanced network BW result in low BW resource utilization and sub-optimal performance [9]. Having only one dimension, *W-1D yielded the overall best performance*. However, if you specifically compare W-1D-350 and Conv-4D (600GBps/NPU), *Conv-4D is driving more BW/NPU, showing better performance* despite being multi-dimensional. Next, we study the impact of scheduling. Themis is a greedy scheduling policy for collectives that aims to balance the load across multiple dimensions to achieve near-optimal BW utilization [9]. W-1D topologies, already being only 1D, show no gain from smart scheduling as shown in Fig. 9(a). However, W-2D, Conv-3D, and Conv-4D, being *multi-dimensional, heavily benefit from Themis scheduler*. It is worth noting that for single All-Reduce and DLRM, *conventional systems with Themis scheduler shows identical results compared to its corresponding wafer-scale systems with equivalent BW/NPU*. Considering the complexity and cost to build a system on a single wafer, such results glimpse the possible advantage of using the conventional hierarchical approach in performance-per-cost aspects. Meanwhile, for GPT-3 and Transformer-1T, wafer-scale systems still maintained better training time. For hybrid parallelism on conventional systems, MP and DP spans over some (and not every) dimensions and utilize only those BW, whereas for wafer-scale every communication runs on full on-wafer BW. This emphasizes the importance of *appropriate parallelization*
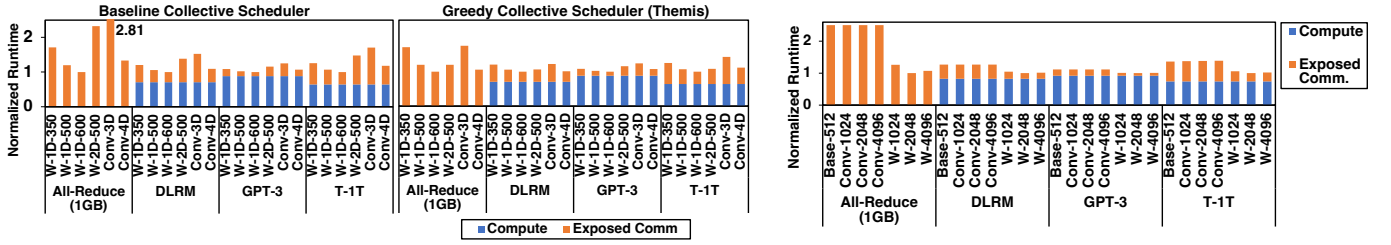
(a) Wafer-scale vs. Conventional (multi-dimensional) systems profiling with 512 NPUs. W-1/2D-x denotes a wafer-scale system that leverages 1D/2D topology with x GB/s bandwidth. Conv-3D/4D means multi-dimensional conventional systems. Normalized training time breakdown with baseline hierarchical scheduling are shown on the left, whereas the results with greedy collective scheduler (Themis) is shown on the right.

(b) Wafer-scale vs. Conventional system scalability comparison. Base-512 denotes baseline 2_8_8_4 (512 NPU) system. Conv-k means conventional (NIC scale-out) framework with k NPUs, whereas W-k denotes wafer-scale (on-chip; Dim 1) scaling with k NPUs.

Fig. 9: (a) Conventional (multi-dimensional) vs. wafer-scale training time breakdown, with and without greedy collective scheduling (Themis) policy. (b) Conventional (scale-out) vs. wafer-scale (on-chip) scalability analysis. Exposed Comm refers to the communication time that is not hidden behind compute time.

TABLE IV: Message size (MB) per each dimension and collective time when running an 1GB All-Gather collective.

System Size	NPUs	Dim 1	Dim 2	Dim 3	Dim 4	Collective Time (µs)
2_8_8_**4**	512	1024	896	112	12	4392.85
2_8_8_**8**	1,024	1024	896	112	14	4392.85
2_8_8_**16**	2,048	1024	896	112	15	4392.85
2_8_8_**32**	4,096	1024	896	112	15.5	4392.85
4_8_8_4	1,024	1536	448	56	6	2212.60
8_8_8_4	2,048	1792	224	28	3	1753.48
16_8_8_4	4,096	1920	112	14	1.5	1879.17

strategies and the need to co-design them with underlying topologies for conventional hierarchical systems.

2) Impact of Scaling using Wafer-scale Systems: Traditional systems scale the infrastructure by scale-out approach, i.e., attach more nodes to the last-dim NICs. On the contrary, wafer-scale technologies let the framework scale up the system, i.e., increasing the number of NPUs on-chip (Dim 1) while maintaining the number of scale-out nodes equally. Measuring the impact, we take the Conv-4D topology from Table II, set the on-chip (i.e., Dim 1) BW to 1,000GB/s to model wafer-scale systems [72], [73], and set it as a baseline. Then, we scale the platform up to 4K nodes and measured the 1GB All-Reduce time. The results are shown in Table IV. *Conventional scale-out increases the Dim 4 (NIC) message size, but the impact was marginal*, thereby showing identical collective time. *Scaling over the wafer, however, significantly increased on-wafer (Dim 1) communication size while dramatically cutting down other dimensions' load*. As far as the system has enough on-wafer BW, collective time decreases due to such an effect, showing an up to 2.51× speedup over the corresponding scale-out mechanism. *Once the on-wafer dimension becomes the bottleneck, the collective time starts to bounce and increase again* as can be seen from the 16_8_8_4 system. End-to-end training time breakdown of GPT-3 and Transformer-1T is also shown in Fig. 9(b), showing the equivalent trend in the end-to-end regime.

B. Comparing Disaggregated Memory Systems

In this case study, we compare the performance of two disaggregated memory systems: ZeRO-Infinity [43] and the hierarchical memory pool (HierMem) presented in Sec. IV-D. We compare the performance of the disaggregated memory
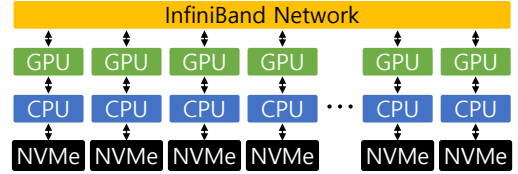

Fig. 10: ZeRO-Infinity system architecture.

TABLE V: Disaggregated memory system configurations

	ZeRO-Infinity	HierMem (Baseline)	HierMem (Opt)
GPU Peak Perf (TFLOPS)	2048	2048	2048
GPU Local HBM BW (GB/sec)	4096	4096	4096
In-node Pooled Fabric BW (GB/sec)	-	256	**512**
Num of Out-node Switches	-	16	16
Num of Remote Memory Groups	256	256	256
Remote Mem Group BW (GB/sec)	100	100	**500**

systems because the latest model sizes already exceed the memory capacity of GPUs available in the market. ZeRO-Infinity is chosen as a baseline system as it is proposed as an effective solution to overcome the limited memory capacity. ZeRO-Infinity is a nascent form of memory disaggregation where each GPU can utilize CPU memory and NVMe in addition to its local HBM memory. Fig. 10 presents the system architecture of ZeRO-Infinity. While ZeRO-Infinity has an advantage in terms of its availability in commodity servers, it does not allow having an arbitrary number of remote memory groups. In other words, it cannot enjoy the major benefit of memory disaggregation, which is cost reduction by eliminating memory underutilization. On the other hand, HierMem can have an arbitrary number of remote memory groups. System parameters for the baseline HierMem configuration are presented in Table V. The values for the baseline configuration are determined based on the latest GPU performance and network bandwidth of commodity servers.

To compare the performance of the systems, we run a training task for a mixture-of-experts (MoE) model with 1 trillion parameters [18]. Fig. 11 presents the execution time breakdown. The execution time of a training task can be broken down into five components: compute time, exposed local memory access time, exposed remote memory access time, exposed communication time, and exposed idle time. The compute time is the total compute time to train a model, and other operations can be hidden behind each other. Non-hidden
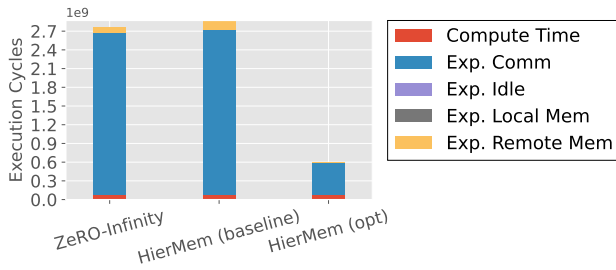
Fig. 11: Runtime breakdown of disaggregated architectures.

time of an operation is defined as exposed time. Overall, ZeRO-Infinity performs 0.1% better than `HierMem`. Both memory systems present similar performance because they have almost equivalent resources. The small performance drop in `HierMem` originates from the additional data transfer stages with multi-level switches.

To find a better-performing configuration of `HierMem`, we explore the design space of `HierMem` while varying in-node pooled fabric bandwidth and the remote memory group bandwidth. We only sweep these parameters as the exposed communication turns out to be a bottleneck. In-node pooled fabric bandwidth is varied between 256GB/s and 2048GB/s with the unit of 256GB/s, and remote memory group bandwidth is varied between 100GB/s and 500GB/s with the unit of 100GB/s. The found best performance with the least resource provision is shown as `HierMem(opt)` in Table V and Fig. 11. It performs 4.6 times better than the baseline configuration.

VI. RELATED WORK

Several simulators exist in our community for modeling distributed systems running general-purpose workloads [74]–[76], with the classic trade-off between simulation accuracy, simulation speed and engineering effort. Moreover, several models/simulators have been proposed to optimize communication performance in HPC platforms, such as LogGOP-Sim [77] and SMPI [78]. This work builds upon the observation of recent works [7], [9], [79], [80] that the compute-memory-communication characteristics of distributed training is possible to abstract and capture via a mix of analytical and simulation methods, without requiring a general-purpose simulator. This is the first simulator, to the best of our knowledge, to enable running arbitrary DNN training execution traces over next-generation platforms with multi-dimensional (scale-up + scale-out) topologies and disaggregated memory systems.

VII. CONCLUSION

In this paper, we motivate the need to swiftly model and profile state-of-the-art and emerging training platforms running large DL models. We enhance the capabilities of ASTRA-sim to enable capturing arbitrary parallelization strategies and training loops, supporting multi-dimensional network topologies, and representing complex memory systems. Using the framework, we run a comprehensive end-to-end, full-stack co-design space exploration of distributed training. With the ability to quickly navigate the complex design space of distributed training, this can give meaningful first-order insights to system designers and assist them in building futuristic training platforms at scale.

REFERENCES

[1] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, vol. 1, Jun. 2019, pp. 4171–4186.

[2] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, 2020, pp. 1877–1901.

[3] S. Lie, "Thinking Outside the Die: Architecting the ML Accelerator of the Future," 2021. [Online]. Available: https://www.microarch.org/micro54/media/lie-keynote.pdf

[4] A. Alford, "Google Open-Sources Trillion-Parameter AI Language Model Switch Transformer," 2021. [Online]. Available: https://www.infoq.com/news/2021/02/google-trillion-parameter-ai

[5] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean, "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer," in *International Conference on Learning Representations (ICLR)*, 2017.

[6] M. Ott, S. Shleifer, M. Xu, P. Goyal, Q. Duval, and V. Caggiano, "Fully Sharded Data Parallel: faster AI training with fewer GPUs," 2021. [Online]. Available: https://engineering.fb.com/2021/07/15/open-source/fsdp

[7] S. Rashidi, S. Sridharan, S. Srinivasan, and T. Krishna, "ASTRA-SIM: Enabling sw/hw co-design exploration for distributed dl training platforms," in *2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2020, pp. 81–92.

[8] "ASTRA-SIM: Enabling SW/HW Co-Design Exploration for Distributed DL Training Platforms," 2020. [Online]. Available: https://github.com/astra-sim/astra-sim.git

[9] S. Rashidi, W. Won, S. Srinivasan, S. Sridharan, and T. Krishna, "Themis: a network bandwidth-aware collective scheduling policy for distributed training of dl models," in *Proceedings of the 49th International Symposium on Computer Architecture (ISCA)*, 2022, pp. 581–596.

[10] S. Rashidi, M. Denton, S. Sridharan, S. Srinivasan, A. Suresh, J. Nie, and T. Krishna, "Enabling Compute-Communication Overlap in Distributed Deep Learning Training Platforms," in *Procidings of the 48th International Symposium on Computer Architecture (ISCA)*, 2021, pp. 540–553.

[11] D. K. Kadiyala, S. Rashidi, T. Heo, A. R. Bambhaniya, T. Krishna, and A. Daglis, "COMET: A comprehensive cluster design methodology for distributed deep learning training," *arXiv:2211.16648 [cs.DC]*, 2022.

[12] 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.

[13] T. Khan, S. Rashidi, S. Sridharan, P. Shurpali, A. Akella, and T. Krishna, "Impact of roce congestion control policies on distributed training of dnns," in *2022 IEEE Symposium on High-Performance Interconnects (HOTI)*, 2022, pp. 39–48.

[14] X. Hou, R. Xu, S. Ma, Q. Wang, W. Jiang, and H. Lu, "Co-designing the topology/algorithm to accelerate distributed training," in *2021 IEEE Intl. Conf. on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*, 2021, pp. 1010–1018.

[15] D. Narayanan, M. Shoeybi, J. Casper, P. LeGresley, M. Patwary, V. Korthikanti, D. Vainbrand, P. Kashinkunti, J. Bernauer, B. Catanzaro *et al.*, "Efficient large-scale language model training on gpu clusters using megatron-lm," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2021, pp. 1–15.

[16] R. Majumder and J. Wang, "DeepSpeed: Extreme-scale model training for everyone," 2020. [Online]. Available: https://www.microsoft.com/en-us/research/blog/deepspeed-extreme-scale-model-training-for-everyone

[17] S. Rajbhandari, J. Rasley, O. Ruwase, and Y. He, "ZeRO: Memory optimizations toward training trillion parameter models," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2020.

[18] S. Rajbhandari, C. Li, Z. Yao, M. Zhang, R. Y. Aminabadi, A. A. Awan, J. Rasley, and Y. He, "DeepSpeed-MoE: Advancing mixture-of-experts inference and training to power next-generation ai scale," *arXiv:2201.05596 [cs.LG]*, 2022.

[19] Z. Jia, M. Zaharia, and A. Aiken, "Beyond data and model parallelism for deep neural networks," *Proceedings of the 2019 Conference on Systems and Machine Learning (SysML)*, 2019.

[20] C. Unger, Z. Jia, W. Wu, S. Lin, M. Baines, C. E. Q. Narvaez, V. Ramakrishnaiah, N. Prajapati, P. McCormick, J. Mohd-Yusof *et al.*, "Unity: Accelerating dnn training through joint optimization of algebraic transformations and parallelization," in *Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2022, pp. 267–284.

[21] A. Sapio, M. Canini, C.-Y. Ho, J. Nelson, P. Kalnis, C. Kim, A. Krishnamurthy, M. Moshref, D. Ports, and P. Richtárik, "Scaling distributed machine learning with in-network aggregation," in *Proceedings of the 18th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2021.

[22] 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 (ISCA)*, 2019.

[23] S. Li, Y. Zhao, R. Varma, O. Salpekar, P. Noordhuis, T. Li, A. Paszke, J. Smith, B. Vaughan, P. Damania *et al.*, "Pytorch distributed: Experiences on accelerating data parallel training," *arXiv:2006.15704 [cs.DC]*, 2020.

[24] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-LM: Training multi-billion parameter language models using model parallelism," *arXiv:1909.08053 [cs.CL]*, 2019.

[25] "NVIDIA DGX Systems." [Online]. Available: https://www.nvidia.com/en-us/data-center/dgx-systems

[26] NVIDIA, "NVIDIA DGX A100: The Universal System for AI Infrastructure," 2021. [Online]. Available: https://www.nvidia.com/en-us/data-center/dgx-a100/

[27] G. Cloud, "System Architecture — Cloud TPU," 2022. [Online]. Available: https://cloud.google.com/tpu/docs/system-architecture-tpu-vm

[28] 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.

[29] ServeTheHome, "Intel Architecture Day 2021 Xe HPC Ponte Vecchio Xe Link," 2021. [Online]. Available: https://www.servethehome.com/intel-ponte-vecchio-is-a-spaceship-of-a-gpu/intel-architecture-day-2021-xe-hpc-ponte-vecchio-xe-link

[30] NVIDIA, "NVIDIA NVLink High-Speed GPU Interconnect," 2022. [Online]. Available: https://www.nvidia.com/en-us/design-visualization/nvlink-bridges

[31] Cerebras, "Cerebras Systems: Achieving Industry Best AI Performance Through A Systems Approach," 2021. [Online]. Available: https://cerebras.net/wp-content/uploads/2021/04/Cerebras-CS-2-Whitepaper.pdf

[32] Tesla, "Tesla Dojo Technology: A Guide to Tesla's Configurable Floating Point Formats & Arithmetic," 2022. [Online]. Available: https://tesla-cdn.thron.com/static/MXMU3S_tesla-dojo-technology_1WDVZN.pdf

[33] N. Agarwal, T. Krishna, L.-S. 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 (ISPASS)*, 2009, pp. 33–42.

[34] D. Comer and J. Griffioen, "A new design for distributed systems: The remote memory model," in *Proceedings of the Usenix Summer 1990 Technical Conference*, 1990, pp. 127–136.

[35] L. Iftode, K. Li, and K. Petersen, "Memory servers for multicomputers," in *Digest of Papers. Compcon Spring*, 1993, pp. 538–547.

[36] K. Lim, J. Chang, T. Mudge, P. Ranganathan, S. K. Reinhardt, and T. F. Wenisch, "Disaggregated memory for expansion and sharing in blade servers," in *Proceedings of the 36th International Symposium on Computer Architecture (ISCA)*, 2009.

[37] J. Gu, Y. Lee, Y. Zhang, M. Chowdhury, and K. G. Shin, "Efficient memory disaggregation with infiniswap," in *Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2017.

[38] M. K. Aguilera, N. Amit, I. Calciu, X. Deguillard, J. Gandhi, P. Subrahmanyam, L. Suresh, K. Tati, R. Venkatasubramanian, and M. Wei, "Remote memory in the age of fast networks," in *Proceedings of the 2017 Symposium on Cloud Computing (SOCC)*, 2017.

[39] Y. Shan, Y. Huang, Y. Chen, and Y. Zhang, "Legoos: A disseminated, distributed os for hardware resource disaggregation," in *Proceedings of the 13th Symposium on Operating Systems Design and Implementation (OSDI)*, 2018, pp. 69–87.

[40] Z. Ruan, M. Schwarzkopf, M. K. Aguilera, and A. Belay, "AIFM: High-performance, application-integrated far memory," in *Proceedings of the 14th Symposium on Operating Systems Design and Implementation (OSDI)*, 2020.

[41] Z. Guo, Y. Shan, X. Luo, Y. Huang, and Y. Zhang, "Clio: A hardware-software co-designed disaggregated memory system," in *Proceedings of the 27th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2022, pp. 417–433.

[42] CXL Consortium, "Compute Express Link (CXL)." [Online]. Available: https://www.computeexpresslink.org

[43] S. Rajbhandari, O. Ruwase, J. Rasley, S. Smith, and Y. He, "ZeRO-Infinity: Breaking the gpu memory wall for extreme scale deep learning," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2021.

[44] J. Verbraeken, M. Wolting, J. Katzy, J. Kloppenburg, T. Verbelen, and J. S. Rellermeyer, "A Survey on Distributed Machine Learning," *ACM Computing Surveys*, vol. 53, no. 2, pp. 1–33, 2020.

[45] Y. Huang, Y. Cheng, A. Bapna, O. Firat, D. Chen, M. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu *et al.*, "GPipe: Efficient training of giant neural networks using pipeline parallelism," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[46] A. Harlap, D. Narayanan, A. Phanishayee, V. Seshadri, N. Devanur, G. Ganger, and P. Gibbons, "PipeDream: Fast and efficient pipeline parallel dnn training," *arXiv:1806.03377 [cs.DC]*, 2018.

[47] J. Ren, S. Rajbhandari, R. Y. Aminabadi, O. Ruwase, S. Yang, M. Zhang, D. Li, and Y. He, "ZeRO-Offload: Democratizing billion-scale model training," in *2021 USENIX Annual Technical Conference (ATC)*, 2021.

[48] B. Klenk, N. Jiang, G. Thorson, and L. Dennison, "An in-network architecture for accelerating shared-memory multiprocessor collectives," in *Proceedings of the 47th International Symposium on Computer Architecture (ISCA)*, 2020, pp. 996–1009.

[49] 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 11th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*, 2006, pp. 2–11.

[50] P. Patarasuk and X. Yuan, "Bandwidth optimal all-reduce algorithms for clusters of workstations," *Journal of Parallel and Distributed Computing*, vol. 69, no. 2, pp. 117–124, 2009.

[51] R. Thakur, R. Rabenseifner, and W. Gropp, "Optimization of collective communication operations in mpich," *The International Journal of High Performance Computing Applications*, vol. 19, no. 1, pp. 49–66, 2005.

[52] M. Cho, U. Finkler, D. Kung, and H. Hunter, "BlueConnect: Decomposing all-reduce for deep learning on heterogeneous network hierarchy," *Proceedings of the 2019 Conference on Systems and Machine Learning (SysML)*, pp. 241–251, 2019.

[53] A. Samajdar, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, "SCALE-Sim: Systolic cnn accelerator simulator," *arXiv:1811.02883 [cs.DC]*, 2018.

[54] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang *et al.*, "Large scale distributed deep networks," *Proceedings of the 26th Annual Conference on Neural Information Processing Systems (NIPS)*, 2012.

[55] A. Sergeev and M. Del Balso, "Horovod: fast and easy distributed deep learning in tensorflow," *arXiv:1802.05799 [cs.LG]*, 2018.

[56] 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," in *Proceedings of the 9th International Conference on Learning Representations (ICLR)*, 2021.

[57] NVIDIA, "NVIDIA Mellanox Scalable Hierarchical Aggregation and Reduction Protocol (SHARP)," 2020. [Online]. Available: https://docs.mellanox.com/display/sharpv214

[58] ——, "ConnectX SmartNICs," 2021. [Online]. Available: https://www.nvidia.com/en-in/networking/ethernet-adapters

[59] Ethernet Technology Consortium, "800G Specification," 2020. [Online]. Available: https://ethernettechnologyconsortium.org/wp-content/uploads/2020/03/800G-Specification_r1.0.pdf

[60] T. P. Morgan, "Deep Dive on Google's Exascale TPUv4 AI Systems," 2022. [Online]. Available: https://www.nextplatform.com/2022/10/11/deep-dive-on-googles-exascale-tpuv4-ai-systems

[61] N. Gebara, M. Ghobadi, and P. Costa, "In-network aggregation for shared machine learning clusters," *Proceedings of the 2021 Machine Learning and Systems (MLSys)*, 2021.

[62] C. Lao, Y. Le, K. Mahajan, Y. Chen, W. Wu, A. Akella, and M. Swift, "ATP: In-network aggregation for multi-tenant learning," in *Proceedings of the 18th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2021.

[63] D. De Sensi, S. Di Girolamo, S. Ashkboos, S. Li, and T. Hoefler, "Flare: Flexible in-network allreduce," in *Proceedings of the 2021 International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2021, pp. 1–16.

[64] H. Pan, P. Cui, R. Jia, P. Zhang, L. Zhang, Y. Yang, J. Wu, J. Dong, Z. Cao, Q. Li, H. H. Liu, M. Laurent, and G. Xie, "Libra: In-network gradient aggregation for speeding up distributed sparse deep training," *arXiv:2205.05243 [cs.NI]*, 2022.

[65] Meta, "PARAM," 2023. [Online]. Available: https://github.com/facebookresearch/param

[66] L. Feng, "[PyTorch] Integrate Execution Graph Observer into PyTorch Profiler," 2022. [Online]. Available: https://github.com/pytorch/pytorch/pull/75358

[67] 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. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," *arXiv:1603.04467 [cs.DC]*, 2016.

[68] NVIDIA, "NVIDIA Collective Communication Library (NCCL)," 2017. [Online]. Available: https://developer.nvidia.com/nccl

[69] Intel, "Intel oneCCL 2021.1 documentation," 2021. [Online]. Available: https://docs.oneapi.io/versions/latest/oneccl/env-variables.html

[70] J. Kim, W. J. Dally, S. Scott, and D. Abts, "Technology-driven, highly-scalable dragonfly topology," in *Procedings of the 35th International Symposium on Computer Architecture (ISCA)*, 2008, pp. 77–88.

[71] NVIDIA, "NVIDIA V100 Tensor Core GPU," 2017. [Online]. Available: https://www.nvidia.com/en-us/data-center/v100

[72] S. Pal, "Scale-Out Packageless Processing," 2021. [Online]. Available: https://nanocad.ee.ucla.edu/wp-content/papercite-data/pdf/phdth11.pdf

[73] S. Pal, D. Petrisko, M. Tomei, P. Gupta, S. S. Iyer, and R. Kumar, "Architecting waferscale processors - a gpu case study," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2019, pp. 250–263.

[74] A. Mohammad, U. Darbaz, G. Dozsa, S. Diestelhorst, D. Kim, and N. S. Kim, "dist-gem5: Distributed simulation of computer clusters," in *2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2017, pp. 153–162.

[75] A. F. Rodrigues, K. S. Hemmert, B. W. Barrett, C. Kersey, R. Oldfield, M. Weston, R. Risen, J. Cook, P. Rosenfeld, E. Cooper-Balis *et al.*, "The structural simulation toolkit," *ACM SIGMETRICS Performance Evaluation Review*, vol. 38, no. 4, pp. 37–42, 2011.

[76] D. Sanchez and C. Kozyrakis, "ZSim: fast and accurate microarchitectural simulation of thousand-core systems," *ACM SIGARCH Computer Architecture News*, vol. 41, no. 3, pp. 475–486, 2013.

[77] T. Hoefler, T. Schneider, and A. Lumsdaine, "LogGOPSim: Simulating large-scale applications in the loggops model," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing (HPDC)*, 2010, pp. 597–604.

[78] A. Degomme, A. Legrand, G. S. Markomanolis, M. Quinson, M. Stillwell, and F. Suter, "Simulating mpi applications: the smpi approach," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 8, pp. 2387–2400, 2017.

[79] W. J. Robinson, F. Esposito, and M. A. Zuluaga, "DTS: A simulator to estimate the training time of distributed deep neural networks," in *The 30th International Symposium on the Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2022.

[80] N. Ardalani, S. Pal, and P. Gupta, "DeepFlow: A cross-stack pathfinding framework for distributed ai systems," *arXiv:2211.03309 [cs.AR]*, 2022.