Accelerating Design Space Exploration for LLM Training Systems with Multi-experiment Parallel Simulation

Fei Gui, *Tsinghua University, BNRist, and Tsinghua Shenzhen International Graduate School;* Kaihui Gao and Li Chen, *Zhongguancun Laboratory;* Dan Li, *Tsinghua University;* Vincent Liu, *University of Pennsylvania;* Ran Zhang and Hongbing Yang, *Zhongguancun Laboratory;* Dian Xiong, *Tsinghua University*

https://www.usenix.org/conference/nsdi25/presentation/gui

This paper is included in the Proceedings of the 22nd USENIX Symposium on Networked Systems Design and Implementation.

April 28–30, 2025 • Philadelphia, PA, USA

978-1-939133-46-5

Open access to the Proceedings of the 22nd USENIX Symposium on Networked Systems Design and Implementation is sponsored by

Accelerating Design Space Exploration for LLM Training Systems with Multi-experiment Parallel Simulation

Fei Gui[⋆§‡], Kaihui Gao[†], Li Chen[†], Dan Li[⋆], Vincent Liu[¶], Ran Zhang[†], Hongbing Yang[†], Dian Xiong[⋆]

[⋆]*Tsinghua University* [†]*Zhongguancun Laboratory* [¶]*University of Pennsylvania*
[§]*BNRist* [‡]*Tsinghua Shenzhen International Graduate School*

Abstract

The rapid expansion of large language models (LLMs) requires the development of extensive GPU clusters, with companies deploying clusters with tens to hundreds of thousands of GPUs. This growth significantly expands the design space for LLM training systems, requiring thorough exploration of different parallelization strategies, communication parameters, congestion control, fabric topology, *etc.* Current methods require up to 10k simulation experiments to identify optimal configurations, with inadequate exploration leading to significant degradation of training performance.

In this paper, we tackle the overlooked problem of efficiently conducting parallel simulation experiments for design space exploration. Our analysis and experiments show that Single-process Multi-experiment (SPME) achieves superior performance by reducing scheduling overhead and optimizing resource utilization, yet remains insufficient for current AI cluster scales. To enhance SPME's efficacy, we introduce Multiverse, a novel GPU-based AI training simulator. Multiverse leverages the computing throughput of GPUs efficiently with optimizations such as a pull-based synchronization, high-fidelity intra-server communication, and a kernel-fusion technique. Extensive experiments validate the accuracy and efficiency of Multiverse, demonstrating less than 3.0% discrepancy with real-world LLM training on clusters of up to 54,000 GPUs, achieving $43.1-73.2\times$ speedup over state-of-the-art CPU-based simulators in various use cases.

1 Introduction

The growth of the scale of LLM models requires large training systems [4, 27]. Companies like Tesla and Meta are building clusters with more than 24 thousand GPUs [16, 30] with O(100k) GPU clusters on the horizon. As a consequence, the design space of the LLM training becomes wider and deeper [15, 28, 54], encompassing the parallelization strategy, parameters of collective communication primitives, congestion control algorithms and parameters, fabric topology

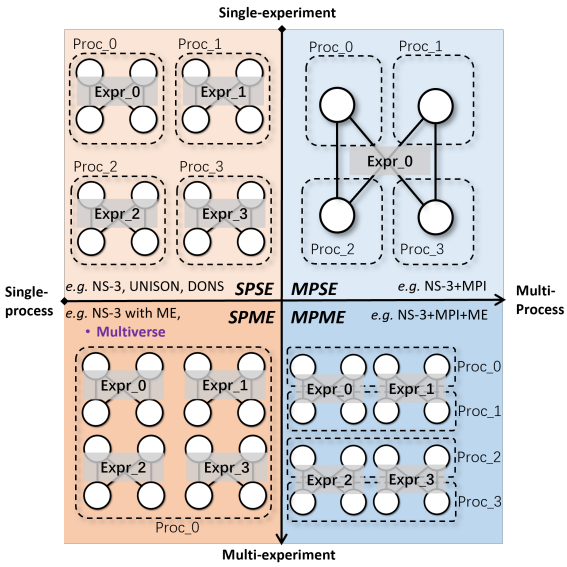


Figure 1: Four types of multi-experiment parallel strategies.

design, and others (see §2.1). As all of these options can interact in complex and unpredictable ways, discovering the optimal design point for a training system requires a massive number of simulation experiments with different option combinations, to fully search the design space. For example, exploring the optimal parallel group size requires ∼100 experiments, while identifying the optimal topology for connecting large-scale GPUs requires over 10k experiments. Insufficient exploration of the design space can lead to suboptimal operational regimes, for instance, employing a suboptimal topology could result in a $3.4\times$ longer training iteration time [58, 59].

Fortunately, we observe that the experimental runs for different points in the design space are generally embarrassingly parallel, which we can leverage to reduce the end-to-end exploration time. How do we best parallelize these experiments? The obvious answer — running n experiments on n CPU cores — falls short of current performance needs, especially when considering that competition for shared resources like memory capacity and low-level cache of CPUs mean that scaling is

sublinear. We believe that, by considering all the experiments at the same time, we can do better.

Our intuition is that batch processing a significant number of experiments aligns well with GPU architecture, which relies on a Single Instruction, Multiple Data (SIMD) execution model. We note that recent work has taken a similar approach to speed up single-experiment network simulation, using a technique called Data-Oriented Design (DOD) [18]. In a DOD-based simulator like DONS [21], users pack data, *e.g.*, packets, in a way where the simulation framework can process steps of the simulation procedure in unison (*e.g.*, running all of ingress pipelines of all switches at the same time) for parallelism and cache efficiency benefits.

We propose to extend the SIMD abstraction to multi-experiment simulation using a strategy in which users run multiple experiments *in a single simulation process*, a strategy we call SPME. Akin to the advantages of Deep Learning and DONS, the advantage of this approach is a substantial reduction in process-inherent overhead and duplicate cache/memory usage. A major additional benefit is that SPME is more amenable to deployment on actual GPUs, whose vastly superior core count and specialization for SIMD provide sizable performance benefits for this type of application.

In this paper, we present Multiverse, the first GPU-based AI training simulator. Multiverse is a DOD-based simulator, but the first to characterize an Entity-Component-System (ECS) model for AI training systems, the first to apply a SPME execution strategy, and the first to adapt all of the above to the constraints of GPU execution. For the latter in particular, Multiverse incorporates three novel techniques for GPU-based AI training simulation: a pull-based synchronization method for lock-free data transfer, a high-fidelity intra-server communication model, and a kernel-fusion technique to greatly improve GPU efficiency.

We perform extensive experiments to confirm the speed and accuracy of SPME and Multiverse. For SPME, our approach occupies a new point in the design space shown in Figure 1. On one axis, we have an existing, well-known criterion: whether to use a single process (SP)[1] or try to accelerate it with multiple processes (MP). On the other axis: running a single experiment in a single process (SE) or running multiple experiments in a single process (ME). We implement all four types of parallel strategies using state-of-the-art simulators (§2.2) and find that on both GPUs and CPUs, SPME achieves the best performance among the four (MP∗E suffers from synchronization overhead, while ∗PSE is inefficient in resource utilization).

For Multiverse as a whole, results show that it can achieve 43.1−73.2× speedup over the state-of-the-art across various use cases (from 50 to 10k simulation experiments) and cluster scales (from 128 to 54k GPUs). Further, the simulation results, such as iteration time, align closely with actual LLM training in real clusters with 1,024 H100 GPUs, exhibiting a difference of less than 3.0%.

We summarize the contributions of this paper as follows:

- We identify the need for parallel experimentation in the design space exploration of large LLM training systems. We then conduct an in-depth analysis and testing of various strategies for parallelizing multiple experiments. The results indicate that the SPME approach outperforms others. We believe that the superior performance of SPME can be attributed to its reduction in process scheduling overhead and its cache-friendly nature.
- We further discover that the DOD principle and the GPU can maximize the potential of the SPME approach. The operational mode of DOD enables parallel execution of identical simulation logic across experiments and devices, such as the forwarding in switches and flow control at the hosts, closely resembling the Single Instruction, Multiple Data (SIMD) tasks that GPUs excel at. We design Multiverse, an AI training simulator, and propose three optimization techniques.
- We evaluate Multiverse on a real testbed, and Multiverse can achieve up to 73.2× speedup over other methods, across various use cases and cluster scales. For simulation accuracy, the training performance metrics output by our simulator closely match the real-world results on a cluster with 1,024 H100 GPUs, showing a discrepancy of <3.0%. Multiverse is open sourced at https://github.com/NASP-THU/multiverse.

2 Background and Motivation

2.1 Exploring the LLM Training Design Space

LLMs require specialized AI training infrastructure, leveraging dozens to 10s/100s of thousands of GPUs that work in concert to complete pre-training or fine-tuning tasks [15, 16, 30, 54]. Constructing large-scale infrastructure to support efficient training involves navigating a deep and broad design space, as illustrated in Figure 2. Mainstream training frameworks such as Megatron [54] and DeepSpeed [48] offer efficient implementations of various parallelization strategies, Data Parallelism (DP), Pipeline Parallelism (PP), and Tensor Parallelism (TP), to utilize large amounts of GPUs efficiently. Collective communication plays a pivotal role in facilitating result interactions among GPUs, utilizing operations like allreduce for gradient synchronization and allgather for full parameter access. The transport protocols and physical network topology underpin the operation of the aforementioned software layers, exerting a fundamental influence on the training performance of LLMs.

Navigating such a vast design space requires extensive exploratory experiments based on simulators, which serve as a crucial tool to validate system designs due to their lower

[1] SP can utilize multi-cores on a single server with multi-threading [8, 21].

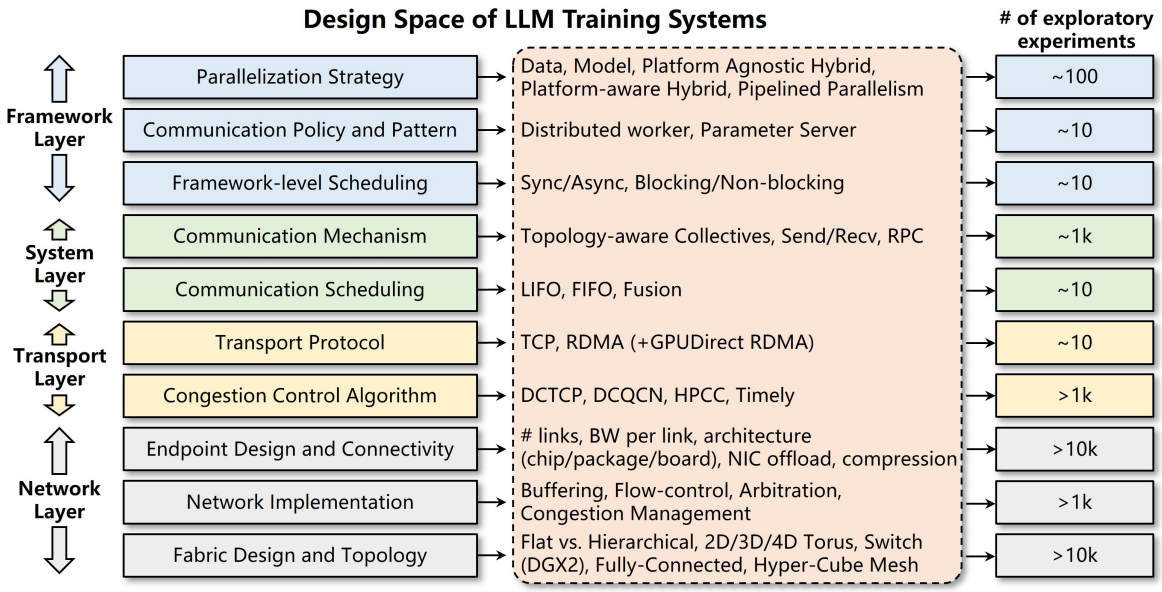


Figure 2: Design space and the number of explorations of LLM training systems.

cost compared to physical testbeds and their higher accuracy relative to analytical models [21, 34, 62]. We list two practical cases to demonstrate how simulation can be utilized to explore optimal design choices for LLM training.

Topology search: There are tons of ways to connect a large number of GPU servers, with classic topologies such as Rail-Optimized Fattree (ROFT) [43], Torus [29], BCube [22], Fat-tree [5], and Dragonfly [31]. However, these well-known topologies may fall short when faced with the evolution of model architectures and training techniques. For example, although ROFT can provide strong all-reduce performance, its all-to-all performance remains less than optimal. Researchers usually try to change parameters such as layers, port numbers, connection relationships, and more in the classic topologies, conduct >10k simulation experiments to compare them, and finally find the most suitable topology for LLM training. For instance, deploying a superior topology can yield a 3.4× increase in training speed [58, 59].

Parallel group size optimization: Allocating GPUs into TP/PP/DP group is also a crucial design decision [26, 42], because TP demands high bandwidth regions while PP needs the avoidance of bubbles. The comparison of group sizes typically requires almost one hundred simulation experiments, depending on the model size and the hardware topology [36].

Although the optimization process can be divided into two distinct phases—a long-term phase for GPU topology search and a short-term phase for other optimizations—to reduce the number of required experiments, a significant number of experiments are still needed in the short-term phase when considering parallel strategy search and collective communication algorithm search. Thus, enhancing simulation speed continues to be an essential objective.

2.2 Multi-experiment Parallelization

We taxonomize the parallelization strategy of multiple independent simulation experiments into four types along two axes: (1) using a single-process or multi-processes to execute one simulation program, and (2) running a single experiment or multiple experiments within one simulation program.

1. Single-Process Single-Experiment (SPSE): This is the most commonly used method during design space exploration, where multiple processes are initiated, each executing a single experiment. For a given experiment, NS-3 [49] and OMNeT++ [57] default to using a single process with a single thread, whereas DONS [21] and UNISON [8] utilize multi-threading (multi-core) to accelerate a single experiment.

2. Multi-Process Single-Experiment (MPSE): This approach uses multiple processes to run a single simulation experiment, like NS-3 or OMNeT++ with MPI [20].

3. Single-Process Multi-Experiment (SPME): Within one program, this strategy extends the single experiment in SPSE to multiple experiments to economize on the inherent overhead of processes.

4. Multi-Process Multi-Experiment (MPME): Similarly, this approach employs multiple processes to execute one program in SPME. For instance, NS-3 utilizes various processes to accelerate a single program, which encompasses several independent topologies running different experiments.

We implement these four parallel strategies based on state-of-the-art simulation technology, and conduct extensive ex-

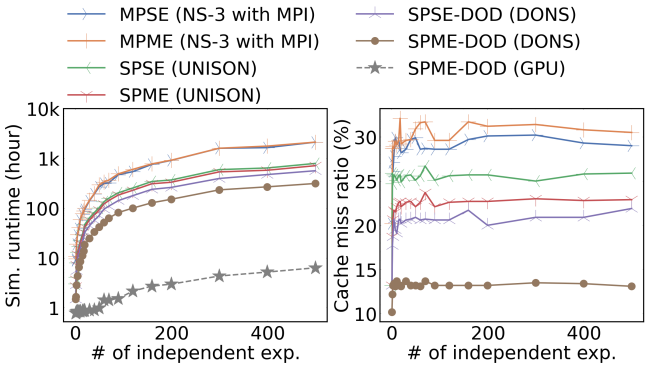


Figure 3: The performance comparison of four multi-experiment parallel strategies.

periments to verify their respective performance. The experimental scenario involves searching for the optimal collective communication algorithm for a cluster with 128 GPU servers, aimed at training a GPT-3 13B model. The simulated topology is Fattree with k=8 (Fattree8), which has 80 switches and 384 links with 100 Gbps. We test these strategies by measuring the time simulators required to complete 1 to 500 independent experiments on a machine, equipped with 80 CPU cores and one H100 GPU. The GPU cache configuration includes 1.3MB L1, 40MB L2, and 55MB L3. Experiments were implemented using C++. The experimental results are depicted in Figure 3, leading us to the following four conclusions.

1. High synchronization overhead in MP∗E: Both MPSE and MPME (the number of processes is set to the optimal value that maximizes CPU utilization) exhibited the poorest performance, necessitating 2k hours to complete 500 experiments. Process-based parallelization cannot share data without incurring context-switching overhead. Experimental results show that high synchronization overhead among multiple processes degrades simulation performance, this problem has been demonstrated in previous literature [21, 62].

2. High cache miss rate in SPSE: We use UNISON and DONS to assess the performance of SPSE. UNISON [8] introduces an efficient multi-threading parallelism technique for NS-3 and claims to achieve super-linear speedup as the number of CPU cores increases, making it the strongest variant to date. DONS [21], leveraging the Data-Oriented Design (DOD) concept, restructures the architecture of the network simulator, achieving improved performance on multi-core CPUs. In both methods, we set the optimal number of cores for each experiment (∼24 cores for Fattree8) to achieve the highest super-linear speedup ratio (∼30×). Consequently, both exhibit superior performance compared to the multi-process versions. However, under multiple experiments, this approach that naively parallelizes by running copies of the simulators will prolong the time of each experiment spent (*i.e.*, the scaling of this dimension is sublinear), due to frequent CPU context switches and severe cache misses.

3. SPME has better performance: We realize the SPME concept by configuring multiple topologies (∼16) within UNISON and DONS. The results show that the performance of both UNISON and DONS improves, benefiting from lower cache misses and process scheduling overhead across experiments. However, DONS with SPME still require significant time (∼370 hours) to complete this case; when used as an exploratory tool, this slow speed is not simply acceptable.

4. Unlocking the potential of SPME requires DOD and GPU: SPME offers a significant performance enhancement for DOD-based DONS. DOD disentangles logic from data, allowing logic to concurrently manipulate the same type of data across all entities. This enables SPME+DOD to identify cross-experiment batching and parallelization opportunities that existing methods ignore. Moreover, this pattern aligns seamlessly with the Single Instruction Multiple Data (SIMD) characteristics of GPUs. The extensive cores within GPUs can further accelerate parallelism across multiple experiments. As a comparison and a preview, Figure 3 illustrates the performance of SPME+DOD+GPU, which surpasses the best available solutions on CPUs by more than 70 times.

2.3 Realizing and Accelerating SPME+DOD for AI Training Systems

With the above observations, we believe that SPME+DOD represents a viable path towards realizing high-performance exploration tools. DOD can improve the execution efficiency of the SPME parallel strategy by separating simulation logic from data as mentioned earlier. Thus, this paper introduces Multiverse, a simulator that realizes SPME+DOD for simulating AI training systems, centered around three key ideas.

1. ECS modeling to enable SPME+DOD for AI training systems. Multiverse uses the Entity-Component-System (ECS) architecture [3, 17] to realize the DOD principle. ECS decomposes objects into separate logic (named *system*) and data (named *component*), and it is currently the default way to realize DOD. We choose ECS because it is easy to develop and prior work, namely DONS [21], has proven its performance benefits. We describe the ECS-based modeling for AI training systems in §3.2.

2. Realizing the SPME+DOD simulation in GPU. SPME+DOD allows multi-experiment simulators to exploit parallelism and coherence (both instruction and data) across experiments to achieve high efficiency on a high-throughput parallel processor like a GPU. To make Multiverse a practical tool for multi-experiment exploration, we delegate all aspects of the simulation to GPUs (§3.3). We centrally manage storage within the GPUs and compile all ECS logic into the GPU, which will adaptively utilize all cores to execute the same simulation logic across all experiments.

3. Optimizing the GPU-based LLM training simulator with three techniques. In the process of developing Multiverse, we see plenty of room to improve performance. Firstly,

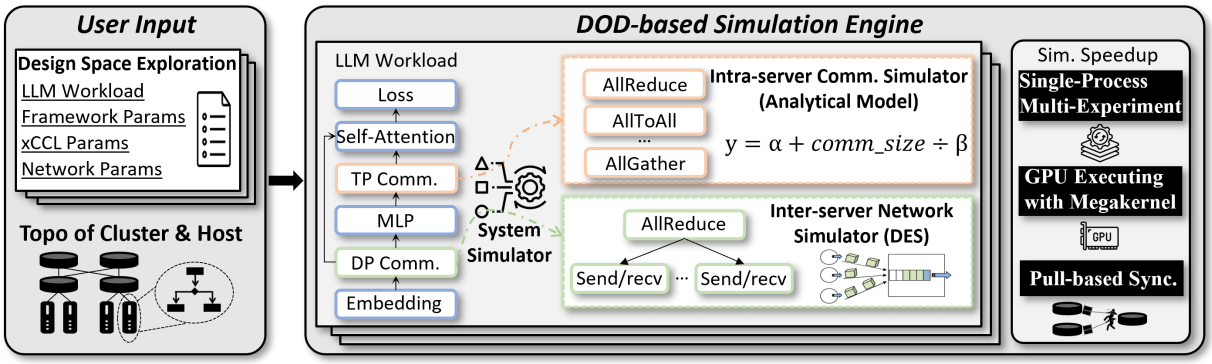


Figure 4: Multiverse's architecture.

executing network simulations with multithreading frequently led to many-to-one write conflicts. To reduce the overhead associated with locks, we implement a pull-based synchronization method. For example, one egress port actively pulls packets from the corresponding ingress ports. Secondly, the communication protocols between NVIDIA GPUs within a single server are not open source. Through extensive profiling experiments, we build an analytical model to estimate the time that GPUs take to execute various collective communication operations. Lastly, we propose a kernel-fusion technique to significantly improve GPU efficiency, compiling the simulation functions, along with wrapper code for component management, into a unified GPU megakernel [45].

3 Multiverse Design

We first overview the architecture of Multiverse and then describe its ECS-based modeling for the AI training system. Finally, we show how to map the ECS models to the GPU. We present the optimization techniques for Multiverse in §4.

3.1 Architecture

Figure 4 shows the key components in Multiverse. Users only need to write the application code, set up an AI training system, including workload, CCL parameters, topology, *etc.* , and specify multiple experiments to run; then Multiverse automatically perform multi-experiment simulations on GPUs.

System Simulator. The system simulator controls and schedules the process of AI training simulation. Its input is the model workload [2], similar to the input in ASTRA-sim [47], which characterizes the computation graph per GPU. Its node is computation operations (such as `Embedding`, `Attetion_linear`, and `MLP_linear`) or collective communication operations (such as `Allreduce`, `Allgather`). We assume that the computation operations are already annotated with the computation times, as the workload produced by Chakra [2]. Multiverse supports typical parallel strategies (*e.g.*, TP, PP, and DP).

For inter-server collective communication operations, the system simulator generates a series of point-to-point send and receive flows by implementing the collective communication algorithms (CCAs) of NCCL [23]. However, within a given CCA, the inherent overhead of the NCCL software stack influences the start time of each flow. To improve the simulation accuracy, we hijack the NCCL APIs to profile the start and end time of each communication in a CCA operation. Consequently, we calibrate the simulation of inter-server collective communications by introducing the measured overhead.

Intra-server Communication Simulator. For an intra-server collective communication operation among GPUs (*e.g.*, TP communications), Multiverse simulates it directly based on the proposed analytical model, which has different empirical parameters according to the operator type and GPU type. This model provides both fast and accurate intra-server communication simulation.

Inter-server Network Simulator. The system simulator registers many point-to-point cross-network communication demands in this network simulator, which would conduct a discrete-event simulation (DES) to rigorously enforce packet-level events and guarantee correctness, like NS-3 [49]. The system simulator is notified when the flow is completed.

GPU Memory Simulator. Since some experimental setups (*e.g.*, the parallel group sizes) may exceed the GPU memory space, Multiverse supports the simulation of GPU memory usage. Before the actual simulation, Multiverse checks whether the parallel strategy and other settings respect the memory size limits. If not, Multiverse generates an *OOM* error.

GPU Runtime. The runtime utilizes ECS abstractions to efficiently allocate multi-experiment simulation tasks to the GPU. Multiverse firstly compiles the ECS system functions, along with wrapper code for component management, into a unified GPU megakernel. Then, the execution graph of ECS systems is transferred to the GPU and processed by the megakernel in every simulation step. The system execution graph informs the runtime that identical systems are executed in each simulation step, and then the GPU adaptively uses all cores to process this system for the corresponding entities across all experiments.

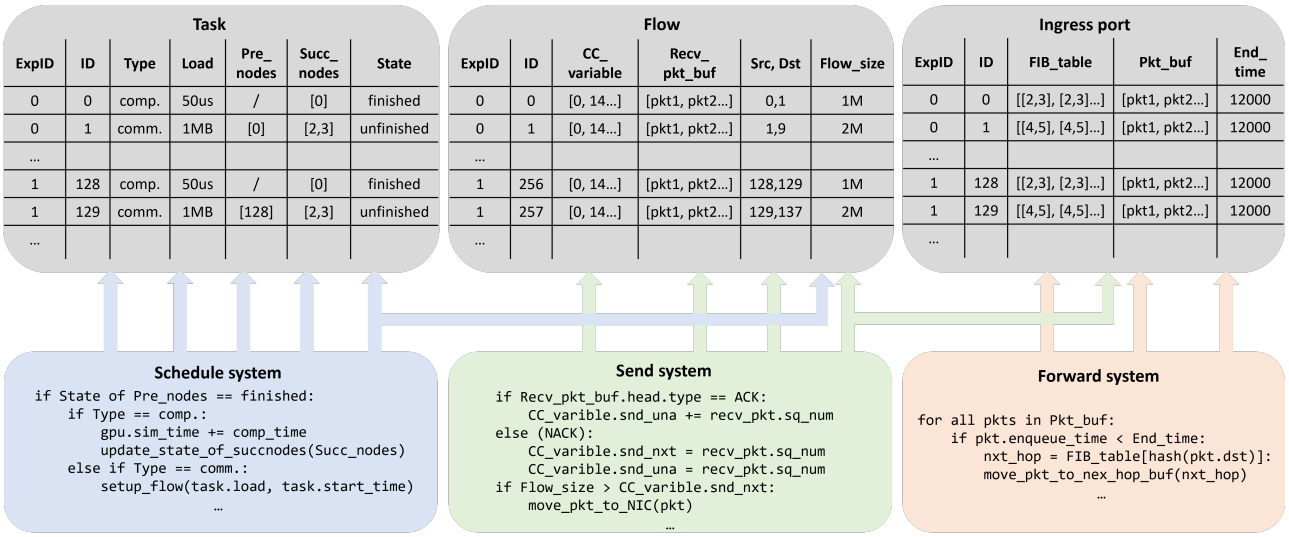


Figure 5: Archetypes and Systems (simplified) in LLM training systems.

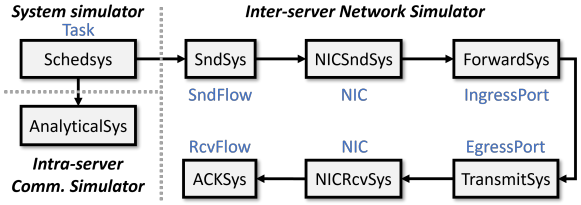


Figure 6: The system execution graph of Multiverse. The terms marked in blue refer to entities.

3.2 ECS Modeling for an AI Training System

Multiverse models the AI training process using Entity-Component-System (ECS) abstractions to realize a high-performance multi-experiment simulator.

Entities and Components. In the context of AI training, we retain the network entities in DONS, such as *Sender*, *IngressPort*, *EgressPort*, and *Receiver*, while introducing pivotal new entities like training *Task*, and *Flow*. The state of an entity is characterized by the values of its components. For instance, as shown in Figure 5, the components defining a *Task* entity encompass type (computation or communication), load (computation time or communication volume), predecessor nodes, and successor nodes. The *Flow* entity is distinguished by components that include its state, congestion control variables, and sending buffer. Entities of the same kind that have the same components are said to share an archetype. Figure 5 illustrates three archetypes of AI training system, as well as example component values for 4 entities of each archetype. Other entities and components are consistent with DONS.

Systems and Queries. Systems represent the data-parallel computations (*i.e.*, simulation logics) executed on collections of entities. A system is delineated by a query that specifies the component data included in the input and a function that is executed on these data. Queries are designed to select entities that possess a predetermined set of components.

For instance, the `Schedule` system in Figure 5 is executed per *Task*. At each step, it checks the completion status of the predecessor nodes, and activates the successor task nodes. For computation tasks, it increases the simulation clock to advance the simulation process; For inter-server communication tasks, it inputs new flows in the packet-level *Network Simulator*; For intra-server communication tasks, the `AnalyticalSys` system executes the analytical model to estimate the communication time.

System execution graph. This graph defines the entire set of ECS systems needed to execute in a simulation step, as illustrated in Figure 6. Nodes in the graph represent systems, and edges represent the execution order.

In a simulation step, Multiverse executes eight systems: `Schedule`, `AnalyticalSys`, `SendSys`, `NICSndSys`, `ForwardSys`, `TransmitSys`, `NICRcvSys`, and `ACKSys`. After Task entities execute the `Schedule` system, the intra-server communication is simulated by `AnalyticalSys`, then new flows (new *SndFlow* entities are created) are injected into the network simulator, where *SndFlow* entities execute the `Send` system to send packets to the corresponding destinations. Then packets traverse *NIC* and a forwarding path comprised of consecutive *IngressPort* entities and *EgressPort* entities. Finally, the packets arrive at the *RcvFlow* entity, which sends back an acknowledgment (ACK) if needed. After the round, Multiverse starts the next simulation step until the simulation ends. As proved in DONS, this mode of execution ensures the simulation correctness if the length of one simulation step is set reasonably (§5).

3.3 Multi-experiment Simulation in GPUs

The system execution graph enables the runtime to recognize that all experiments will utilize the same systems during sim-

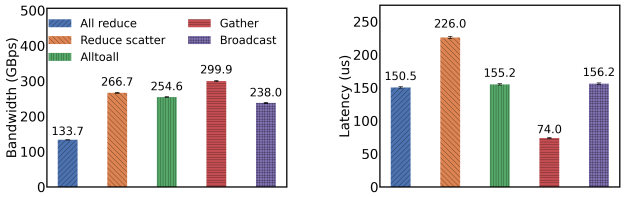
ulations. This understanding allows for safe and efficient parallel execution. By having clear knowledge of execution and data types, Multiverse gains significant control over how simulation logic is assigned to GPUs, leading to high-performance simulations. Multiverse compiles all ECS systems along with the necessary wrapper code for component access into a single GPU megakernel [45]. Then, this megakernel executes the system execution graph for each step of the simulation. Notably, archetypes and their components are set up as column stores in GPU memory.

Management of Component Storage across Experiments. In multi-experiment simulations, Multiverse centrally oversees the storage of all component data. It constructs a single in-memory table that holds component data for entities sharing the same archetype across all experiments, as illustrated in Figure 5. To facilitate efficient access to consecutive components, these tables are organized as column stores, with component data stored in contiguous memory, adhering to standard ECS practices [3]. To handle the per-experiment state when processing entities, Multiverse includes an implicit `ExpID` component in each table, which allows for quick retrieval of experiment-specific data for each entity.

This cross-experiment storage approach improves the cache miss rate of Multiverse. During execution, neighboring GPU threads that process consecutive entities in a table can access component data coherently, even if those entities belong to different experiments. If separate tables were maintained for each experiment, data access would become incoherent when only a few entities per experiment matched an ECS query, as neighboring threads would access data from different tables.

Executing ECS Systems in GPUs. At a higher level of abstraction, Multiverse uses the NVIDIA CUDA C++ compiler to compile these systems for execution on the GPU. Each system invocation is mapped directly to a single GPU thread, utilizing the single instruction multiple threads (SIMT) programming model. To facilitate the parallelization of entity updates and data flow across the execution graph, ECS systems are structured as functions that take the components of individual entities as inputs. When an ECS system is integrated into the execution graph, Multiverse uses the provided ECS query to identify all matching archetypes and their corresponding column indices for each accessed component. This information enables Multiverse to generate the necessary code for managing component access and to invoke the ECS system function across multiple GPU threads for each entity that matches the criteria.

For example, in Figure 5, the `Send` system requires data from columns 3 and 4 in the `Flow` table. Using this information, Multiverse creates a system entry function that links GPU threads to table row indices and supplies the retrieved component data to the ECS system function. Given a table with N rows, Multiverse would call the system entry function N times, with each call resulting in a single GPU thread executing the ECS system function. In the case of the `Send` system, if the query identifies N rows in the `Flow` table, Multiverse would invoke the entry function N times, providing the first N invocations with column pointers for the `Flow` table.



(a) Bandwidth (β). (b) Latency (α).

Figure 7: Calibrated bandwidth and latency parameters in the analytical model.

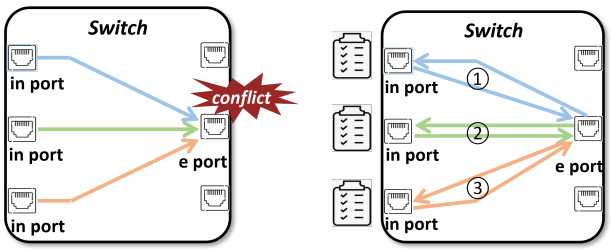
4 Multiverse Optimization

4.1 Intra-server Communication Simulation

In LLM training systems, TP traffic among intra-server GPUs constitutes a significant proportion of the aggregate network load, rendering the precise simulation of TP traffic necessary to accurately simulate LLM training. TP traffic predominantly transfers through high-bandwidth, intra-server conduits such as NVLink and PCIe. Existing simulation methodologies, however, grapple with accurately modeling TP traffic. Dedicated packet-level simulators (*e.g.*, NS-3 [49], DONS [21]) for inter-server networks face challenges in accurately simulating intra-server communication due to the fundamentally different communication protocols involved. Existing AI training simulators employ analytical models (*e.g.*, ASTRA-sim [47]) to estimate this communication: $y{=}\alpha{+}comm\_size/\beta$, where α and β represent the communication latency and bandwidth, respectively. And *comm_size* denotes the total communication volume for a GPU during the collective communication operation. The advantage is the speed of simulation. However, a significant drawback is the inaccuracy of the model parameters, as they fail to capture the overhead introduced by the NCCL software stack.

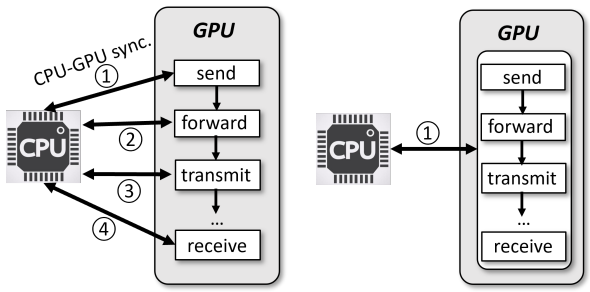
Through extensive empirical testing, we discover that the parameters within analytical models cannot remain static. Deploying a server equipped with 8 A100 GPUs interconnected via 300GB/s NVLink, we use the analytical model from ASTRA-sim to estimate the completion time of TP yields an error margin ranging from 20% to 72%. We identify that the issue does not lie with the linear model itself, but rather with the need for its parameters to vary according to different scenarios. To encapsulate this pattern, we conduct tests to examine the relationship between the communication size and the completion time of various collective communication operators. By fitting a linear model to the results, we are able to derive revised parameters.

As shown in Figure 7, the analytical model we proposed



(a) Multiple-to-one write conflict. (b) Pull-based sync.

Figure 8: Resolute the multiple-to-one write conflict with pull-based synchronization.



(a) Kernel-based. (b) Megakernel-based.

Figure 9: The execution of the ECS systems in GPU.

meticulously adjusts specific parameters (α and β) for each collective communication operation, contingent upon the number and type of GPU (*e.g.*, A100/H100) within a server. The bandwidth values deviated from officially documented metrics, and the latency can be 70*us* to 200*us*. This calibrated approach facilitates the accurate simulation of intra-server communication; evaluation results reveal that the error is merely $0.7\%-1\%$. Crucially, this modeling approach enables a significant enhancement in speed without compromising accuracy. This attribute is beneficial for augmenting the compute throughput of ECS architectures within GPUs, affording a substantial uplift in the simulation performance.

Simulating the overlap of communication and computation. Furthermore, the computation time can be significantly influenced by the degree of overlap with communication due to resource contention, such as streaming multiprocessors (SM) and high-bandwidth memory (HBM) bandwidth. To address that, Multiverse adjusts the computation time based on the overlap between computation and communication. Initially, Multiverse specifies the duration of each computation operator without overlap in the input file. During the simulation, this duration is modified according to the overlap ratio between computation and communication operators and the extent to which the computation duration is extended due to this overlap. To determine the overlap ratio, the duration of the communication operator is obtained using either a scale-out package-level network simulator or a scale-up analytical model. By comparing the durations of the communication and computation operators, the overlap ratio is calculated. Additionally, an empirical modeling approach is used to evaluate the extent to which computation time is extended when the computation operator overlaps with the communication operator. This model considers various factors, including the model type, the computational operator, and the GPU type.

4.2 Pull-based Data Synchronization

In Multiverse, the `nic_forward` and `forward` systems extensively perform multiple-to-one write operations, a scenario where current ECS frameworks (like Unity ECS [56]) exhibit significant limitations due to the lack of atomic operations. As a workaround, these frameworks are compelled to accumulate all send events in a global buffer, subsequently processing them sequentially in the main thread. This methodology severely compromises parallel efficiency. The key challenge is managing concurrent writes without resorting to the conventional lock-based synchronization techniques.

We adopt a pull-based data synchronization strategy to enhance parallel efficiency. This approach bifurcates the multi-to-one writing process within the system into two distinct phases: `set_write_plan` and `write`. In the `set_write_plan` phase, packets slated for dispatch are preliminarily listed in a to-do queue. Then, in the `write` phase, each destination actively retrieves the packets intended for it from the source. For example, we divide the `forward` system into two subsystems: `set_forward_plan`, and the `forward`. In the former, a to-do list (realized as a bitmap) is maintained for each ingress port within the current switch, earmarking impending forward events. Subsequently, in the `forward`, each egress port independently pulls the packet from the ingress ports based on its to-do list one by one, as illustrated in Figure 8b. This method allows all ports to execute the `forward` system in a lock-free manner, improving parallel efficiency.

4.3 Execution with Megakernel

A crucial aspect of Multiverse setup is the execution of the entire system execution graph. The straightforward method would be to run each system as an individual CUDA kernel. However, this strategy has a significant problem. In GPU, initiating a kernel is a time-consuming process because it needs to be initialized by the CPU. As shown in Figure 9a, Multiverse consists of numerous ECS systems, leading to a high amount of CPU-GPU synchronization overhead, which notably decreases GPU efficiency.

As shown in Figure 9b, Multiverse uses a GPU-driven method with a large-scale kernel design [45]. Multiverse compiles all systems in the execution graph into a single CUDA kernel, *i.e.*, megakernel. This kernel is initiated by the CPU once per batch simulation and completes the entire execution graph before returning to the CPU. To manage both the execution of systems defined by the application and necessary engine-level operations, such as evaluating ECS queries and sorting tables, Multiverse creates a task graph. This graph

outlines the full execution of each batch simulation step. Multiverse uses a simple task graph scheduling policy in which all GPU threads work on the same node together before moving to the next node in the task graph. While this approach could lead to low GPU utilization if there is not enough work per node, large-batch simulations of multi-experiment typically provide a significant amount of work per node. So we can easily fully utilize all the GPU cores with a megakernel.

5 Implementation

We base Multiverse on the Madrona framework [52]. Multiverse 's current core code base has $\sim 13k$ lines of C++ code, which we release for the networking community. Currently, Multiverse supports both TP (Tensor Parallelism) and DP (Data Parallelism) parallelization strategies, along with collective communication algorithms including Ring Allreduce, Allgather, and Reducescatter. Additionally, Multiverse integrates multiple congestion control algorithms such as DCQCN, HPCC, and DCTCP, as well as load balancing algorithms like ECMP and packet spraying. Here we briefly describe the length of a step in Multiverse, and how it use multiple GPUs.

The length of one simulation step. Multiverse traverses the execution graph in one simulation step, advancing the simulation clock incrementally. Given that the execution of distinct systems within the execution graph is sequential, prolonged step length can lead to temporal errors. This step length effectively corresponds to the notion of *lookahead* concept in parallel DES [20]. Taking inspiration from DONS [21], we set $min\{link\_delay\}$ as the lookahead. Because event propagation between servers and switches inherently involves at least one link delay.

Using multiple GPUs. Given the finite resources within a single GPU, evaluation reveals that the number of experiments that one GPU can simultaneously run is limited, when each experiment is of substantial scale. Multiverse can leverage multiple GPUs for parallel exploration experiments. Upon user configuration of available GPUs, Multiverse will assess the necessity of reallocating certain independent experiments across different GPUs. If necessary, Multiverse will automatically balance the workload among multiple GPUs to maximize simulation speed. The execution across GPUs are independent.

6 Evaluation

We evaluate Multiverse's simulation speed, accuracy, and scalability. We summarize our results as follows:

Key Results.

- **Simulation speed:** Multiverse can achieve a 57.4-73.2× speedup over the state-of-the-art across various use cases, ranging from 50 to 10,000 simulation experiments, and cluster sizes, from 128 to 8,192 GPUs.
- **Scalability:** Using only one GPU, Multiverse can run one simulation experiment for a cluster of up to 54,000 GPUs and achieve up to 43.1× speedup over other methods. And Multiverse can parallel run 52 simulation experiments for a cluster of 1,024 GPUs and 13 simulation experiments for a cluster of 8,192 GPUs.
- **Fidelity:** Multiverse's simulation results, such as iteration time, align closely with real-world LLM training on testbeds, with a difference of $<3.0\%$, even at 1,024 GPUs.

Alternatives. We select existing state-of-the-art simulators for AI training systems as our comparisons.

- **ASTRA-sim+UNISON:** ASTRA-sim [47, 60] is a comprehensive AI training simulator. To enhance its simulation speed, we improve the network simulator using UNISON [8], a recently proposed multi-threading simulation kernel. This approach can utilize the SPSE and SPME manners, denoted as *ASTRA-sim+UNISON (SPSE)* and *ASTRA-sim+UNISON*, respectively.
- **ASTRA-sim+DONS:** We incorporate ASTRA-sim with another state-of-the-art network simulator, DONS, to implement an accelerated ASTRA-sim. This method can utilize the SPSE and SPME manners, denoted as *ASTRA-sim+DONS (SPSE)* and *ASTRA-sim+DONS*, respectively.
- **Multiverse (SPSE)**: This approach takes Multiverse to follow the SPSE manner. And it employs all the optimization technologies we proposed.

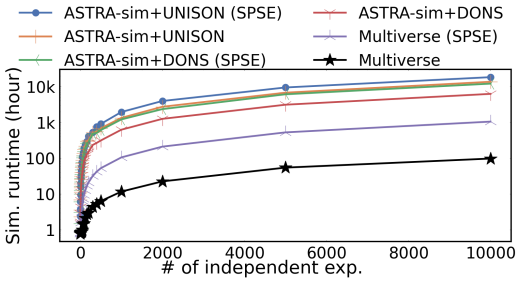
Setup. We conduct experiments with these alternatives on a Linux server configured with one NVIDIA H100 GPU, an 80-core Intel CPU@2.10GHz, and 256GB of memory.

To verify Multiverse's accuracy, we run real LLM training jobs (with LLaMA 65B and GPT-3 175B model) on a cluster, which is based on the Fattree, inter-host RoCEv2 network. The cluster consists of 128 servers, each equipped with eight NVIDIA H100 GPUs and eight Mellanox ConnectX-7 NICs (2×200Gbps). Intra-server GPUs are interconnected via NVLink, providing 900GBps bandwidth. The clusters include two kinds of scale: 128 GPUs and 1,024 GPUs.

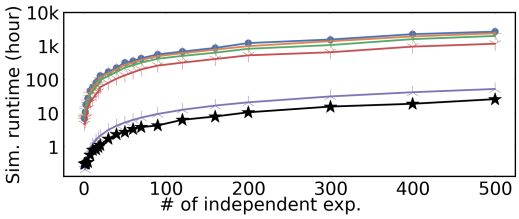
We take four real use cases of design space exploration to test the performance of Multiverse, as shown in Table 1. These use cases span a comprehensive spectrum, including topology search, collective communication enhancement, fine-tuning of parallel group sizes, and the assessment of congestion control algorithms. These cases cover a wide range of cluster scales (ranging from 128 to 54,000 GPUs) and model sizes (from 13 billion to 175 billion parameters). We obtain the approximate number of experiments required for these four explorations from the production datacenters. For instance, topology search needs varying layers, switch radixes, and connectivity configurations to generate a myriad of topologies, requiring 10k simulation experiments.

Use case	GPU scale	Topology	Workload	Explored parameters	# of independent exp.
#1: Topology optimization	128 GPUs	Fattree-like	GPT-3 13B	Set different layers, switch radix, connections, *etc.*	10k
#2: Collective communication optimization	1,024 GPUs	Fattree k=16	LLaMA 65B	Set different flow priority and load balancing strategies.	500
#3: Selection of TP/DP/PP group size	8,192 GPUs	Fattree k=32	GPT-3 175B	Set different TP/DP/PP group size.	100
#4: Comparing congestion control algorithms	54,000 GPUs	Fattree k=60	GPT-dense 175B	Set different CC algorithm, such as DCQCN [63], HPCC [35].	4

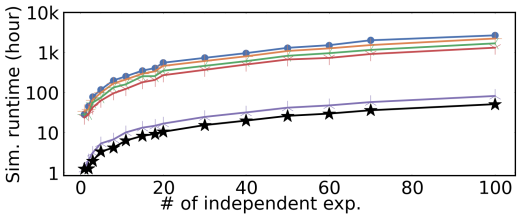
Table 1: The detailed setting for the four different use cases in evaluation.



(a) The total spent time of multiple simulation experiments for a cluster with 128 GPUs (use case #1).



(b) The total spent time of multiple simulation experiments for a cluster with 1,024 GPUs (use case #2).



(c) The total spent time of multiple simulation experiments for a cluster with 8,192 GPUs (use case #3).

Figure 10: The comparison of simulation speed.

6.1 Multiverse is Fast

We first compare the simulation speed of Multiverse, with other techniques across 3 use cases in Table 1. As depicted in Figure 10, Multiverse consistently outperforms all other methods, achieving speed increases of up to 73.2×, 67.6×, and 57.4× compared to other alternatives when simulating a cluster with 128 GPUs, 1,024 GPUs, and 8,192 GPUs, respectively. Furthermore, Multiverse surpasses Multiverse (SPSE), with speed improvements reaching up to 7.3×, 2.4×, and 1.7×, respectively. These speed enhancements can be attributed to the reduced cache miss ratio, as shown in Figure 11, a result of the DOD paradigm and batch and parallel simulation across multiple experiments. Additionally, Multiverse benefits from

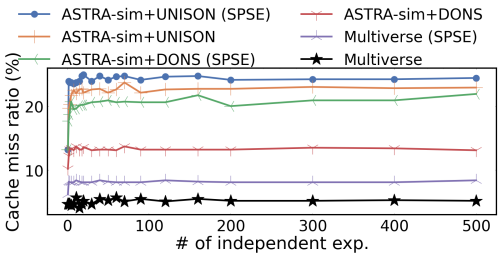


Figure 11: Cache miss ratio in use case #2.

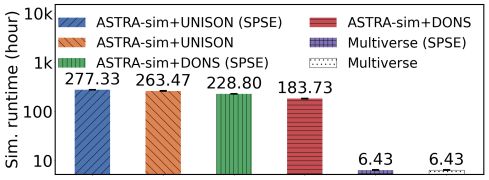


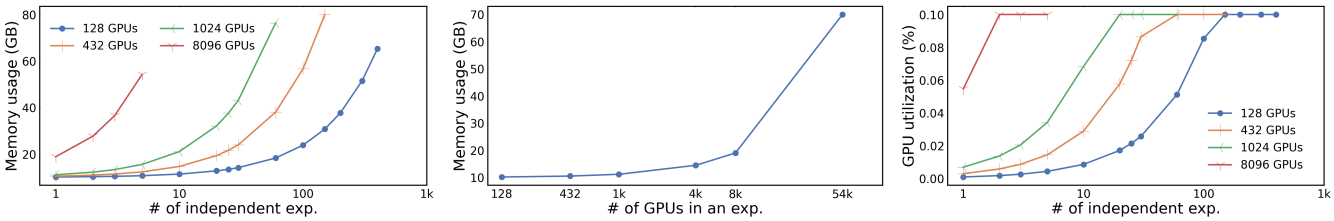
Figure 12: The speed comparison in simulating a large-scale cluster with 54k GPUs (use case #4).

reduced memory overhead as it distributes the inherent memory cost in all experiments. Together, these factors improve the parallel efficiency of Multiverse.

ASTRA-sim+DONS performs better than ASTRA-sim+UNISON, achieving a speedup of approximately 1.9−2.1×, as ECS programming abstractions are well suited for creating novel multi-experiment simulators. These ECS abstractions also provide essential structure over custom logic and state, enabling simulators to efficiently manage memory, distribute work, and identify coherent parallel computations within and across different experiments.

Multiverse also outperforms ASTRA-sim+DONS, with speed improvements of up to 25.1−47.2×. This is because ASTRA-sim+DONS, despite using batch simulation across multiple experiments to achieve good performance against ASTRA-sim+DONS (SPSE), has its parallelism constrained by the number of CPU cores. In contrast, Multiverse can leverage the extensive number of GPU cores to further accelerate the simulation. Moreover, Multiverse uses pull-based synchronization for lock-free data transfer, a fast analytical model, and a megakernel technique to enhance GPU efficiency.

Maximum scale. We test the maximum cluster that Multiverse can simulate using a single GPU. Subject to the constraints imposed by the GPU memory size in our testbed, Multiverse can simulate an LLM training cluster that includes up to 54k GPUs, 4.5k switches, and 162k links. This scale surpasses



(a) GPU memory usage. (b) GPU memory usage running 1 exp. (c) GPU utilization.

Figure 13: The performance of Multiverse.

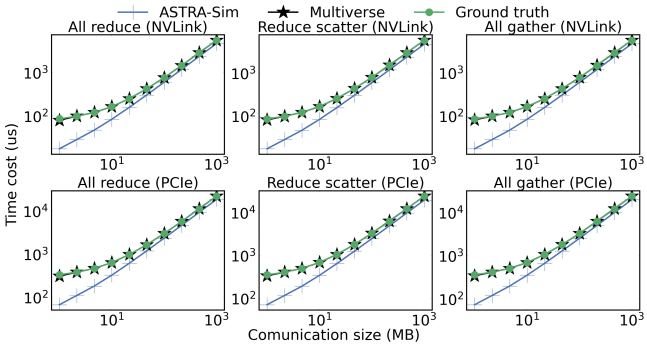
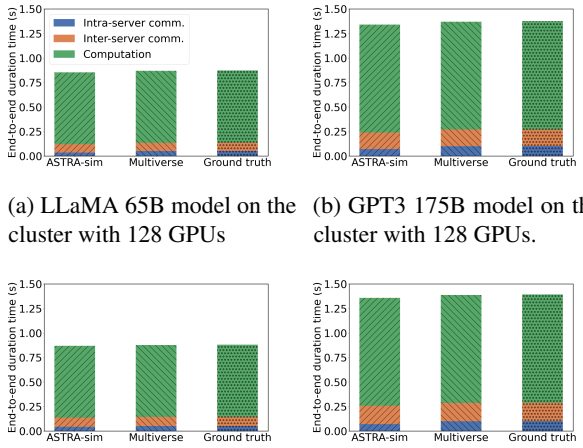


Figure 14: The time cost of intra-server collective communication operations.



(a) LLaMA 65B model on the cluster with 128 GPUs (b) GPT3 175B model on the cluster with 128 GPUs.

(c) LLaMA 65B model on the cluster with 1,024 GPUs (d) GPT3 175B model on the cluster with 1,024 GPUs.

Figure 15: The iteration time of different workloads running on ASTRA-sim, Multiverse, and real clusters.

that of most LLM training systems. Even under these conditions, Multiverse can significantly outperform alternative methods, achieving a speedup factor between 28.6× to 43.1×, as depicted in Figure 12.

The memory consumption and GPU utilization of Multiverse are illustrated in Figure 13. With the use of a single GPU, Multiverse can execute one simulation for a cluster with up to 54k GPUs at a time, this being limited by the GPU memory size, as demonstrated in Figure 13b. Furthermore, Multiverse can concurrently run 520 simulation experiments for a cluster of 128 GPUs, 70 simulation experiments for a cluster of 1,024 GPUs, and 5 simulation experiments for a cluster of 8,192 GPUs.

In terms of GPU utilization, Multiverse can fully engage all GPU cores when running a single simulation experiment for a cluster with more than 8,192 GPUs. Multiverse can also fully utilize all the GPU cores when running more than 30 simulation experiments for a cluster with 128 GPUs, and more than 10 simulation experiments for a cluster with 1,024 GPUs.

6.2 Multiverse is Accurate

The precision of intra-server communication simulation. As depicted in Figure 14, we evaluate the accuracy of Multiverse in simulating different sizes (2-2560MB) of collective communication on a server with 8 A100 GPUs(300GBps NVLink and PCIe5.0). For smaller communication sizes, ASTRA-sim can deviate by as much as 72.1% from the actual values. We test the accuracy of both NVLink and PCIe connected, respectively. Although this error decreases with increasing communication size, it remains above 22.0%. In contrast, Multiverse shows an error of 1.0-1.2% for small communication sizes, but this discrepancy reduces to less than 0.8% as the communication size increases.

From an end-to-end simulation perspective, the accuracy of intra-server communication for both ASTRA-sim and Multiverse improves with larger communication sizes, as seen in Figure 15. This is particularly noticeable with the collective communication operations of GPT-3 175B, which are larger and thus more accurately simulated. However, ASTRA-sim can still exhibit an error greater than 20.0% even when simulating GPT-3 175B, while Multiverse reduces the error to less than 1.0% for both LLaMA 65B and GPT-3 175B.

The precision of inter-server network simulation. As seen in Figure 15, due to the precision of DES, both using Multiverse and ASTRA-sim, the inter-server communication time closely approximates the real-world LLM training.

The precision of end-to-end simulation. We test the accuracy of Multiverse by examining its end-to-end metric across various workload and cluster sizes. Figure 15 demonstrates that the iteration time for all workloads when using Multiverse closely matches that of real LLM training tasks. The difference is less than 3.0%, even at the scale of 1,024 GPUs. Moreover, as the model size increases (for instance, from

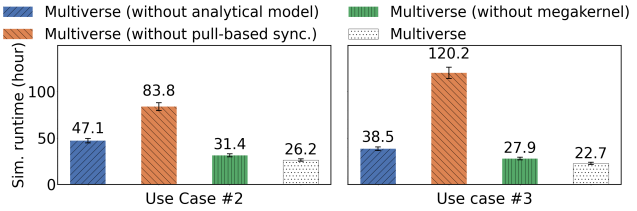


Figure 16: Performance breakdown of Multiverse.

LLaMA 65B to GPT-3 175B), Multiverse's accuracy improves correspondingly, because the message sizes of the collective communication operations in GPT-3 175B are larger.

6.3 Ablation Study of Multiverse

In Figure 16, we perform an ablation study to assess the impact of key features of Multiverse on its overall performance.

The term Multiverse (without analytical model) refers to the method that employs a packet-level DES to simulate intra-server communications, where all intra-server GPUs are interconnected via a switch. Compared to this method, Multiverse can achieve a speedup of 1.7-1.8×, because the calibrated analytical model has lighter computational loads.

Multiverse (without pull-based synchronization) uses the global synchronization method typically utilized in existing ECS frameworks. Compared to this approach, Multiverse can achieve a speedup of 3.2-5.4×. This is because pull-based synchronization allows all ports to execute the forward system in a lock-free parallel manner, significantly enhancing parallelism efficiency.

Lastly, Multiverse (without megakernel) is a method by which each ECS system is initialized as an independent kernel function in the GPU. Compared to this method, Multiverse can reduce the simulation time by 16.6%-18.6%, as shown in Figure 16. This is because using megakernel can substantially decrease the number of times the CPU needs to launch a kernel, resulting in lower CPU-GPU synchronization overhead.

7 Related Work

AI training simulation. ASTRA-sim [47, 60] is a comprehensive AI training simulator, which aims to simulate the software and hardware co-design stack of distributed training systems. Building upon ASTRA-sim, SimAI [7] enhances simulation accuracy, Dally [53] additionally simulates modern networking hardware. Approaches like [1, 9, 19] adopt a more coarse-grained simulation paradigm, such as operator-level or task-level, to achieve faster simulation speeds with the sacrifice of accuracy. SimBricks [34] and SplitSim [33] achieve end-to-end network system simulation by integrating multiple simulators, such as NS-2/3 [24, 49] and gem5 [6, 12, 41]. While they can be extended to simulate AI/LLM training, their performance is constrained. Notably, they fail to fully leverage the fine-grained parallelism opportunities provided by SPME.

Network simulation. Existing network simulators can be divided into three categories: discrete-event simulation (DES) [24, 38, 49, 57], mathematical model estimation [32, 39, 46] and AI-based estimation [50, 61, 62]. DES simulators offer high-fidelity packet-level simulation. Some work leverage parallel and distributed DES (PDES) [20, 25] approaches to accelerate simulation. However, multiple processes often come with poor performance and complex configurations. UNISON [8] introduces fine-grained partitioning and load-adaptive scheduling to enable efficient multi-threading. DONS [21] uses the data-oriented design to reduce cache miss and can be automatically parallelized.

Entity-Component-System (ECS). The ECS architecture has gained recent popularity through implementations in major game engines such as Unity [3] and Unreal [17], as well as a wide array of open-source implementations of the architecture [14, 40, 52]. The ECS architecture has been shown to provide performance benefits due to data access efficiency and parallelism [13, 56].

GPU-based DES. Many efforts employ GPUs to accelerate DES of queuing networks [44, 51, 55, 62], P2P networks [37], and mobile networks [10, 11]. However, they are still rooted in conventional PDES techniques, lacking the utilization of ECS architecture to optimize performance.

8 Conclusion

This paper demonstrate that the parallel strategy of Single-process Multi-experiment (SPME) achieves superior performance by reducing scheduling overhead and optimizing resource utilization, yet remains insufficient for current AI cluster scales. To enhance SPME's efficacy, we introduce Multiverse, a novel GPU-based AI training simulator. Using DOD/ECS modeling, Multiverse leverages the computing throughput of GPUs efficiently with optimizations such as a pull-based synchronization method, high-fidelity intra-server communication model, and a megakernel technique. Extensive experiments validate Multiverse 's accuracy and efficiency, achieving $43.1-73.2\times$ speedup over state-of-the-art CPU-based AI training simulators.

9 Acknowledgement

We thank our shepherd, Dr. Hang Zhu and the anonymous NSDI reviewers for their constructive comments. Kaihui Gao and Li Chen are the corresponding authors. This work was supported by National Key R&D Program of China (Grant No. 2022YFB3105000), the Beijing Outstanding Young Scientist Program (Grant No. JWZQ20240101008), and the National Natural Science Foundation of China(Grant No. U23B2001). It was also supported by Zhongguancun Laboratory.

References

[1] Online; Last accessed Sep. Arcadia: An end-to-end AI system performance simulator. https://engineering.fb.com/2023/09/07/data\-infrastructure/arcadia-end-to-end-ai-system\-performance-simulator/, 2024.

[2] Online; Last accessed Sep. Chakra Working Group. https://mlcommons.org/working-groups/research/chakra/, 2024.

[3] Online; Last accessed Sep. Entity Component System concepts. https://docs.unity3d.com/Packages/com.unity.entities@1.3/manual/concepts-intro.html, 2024.

[4] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[5] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. *ACM SIGCOMM CCR*, 2008.

[6] Mohammad Alian, Daehoon Kim, and Nam Sung Kim. pd-gem5: Simulation infrastructure for parallel/distributed computer systems. *IEEE Computer Architecture Letters*, 15(1):41–44, 2015.

[7] Anonymous authors. Simai: Unifying architecture design and performance tunning for large-scale large language model training with scalability and precision. In *USENIX NSDI*, 2025.

[8] Songyuan Bai, Hao Zheng, Chen Tian, Xiaoliang Wang, Chang Liu, Xin Jin, Fu Xiao, Qiao Xiang, Wanchun Dou, and Guihai Chen. Unison: A parallel-efficient and user-transparent network simulation kernel. In *Proceedings of the Nineteenth European Conference on Computer Systems*, pages 115–131, 2024.

[9] Jehyeon Bang, Yujeong Choi, Myeongwoo Kim, Yongdeok Kim, and Minsoo Rhu. vtrain: A simulation framework for evaluating cost-effective and compute-optimal large language model training. In *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 153–167. IEEE, 2024.

[10] Ben Romdhanne Bilel and Nikaein Navid. Cunetsim: A gpu based simulation testbed for large scale mobile networks. In *2012 International Conference on Communications and Information Technology (ICCIT)*, pages 374–378. IEEE, 2012.

[11] Ben Romdhanne Bilel, Nikaein Navid, and Mohamed Said Mosli Bouksiaa. Hybrid cpu-gpu distributed framework for large scale mobile networks simulation. In *2012 IEEE/ACM 16th International Symposium on Distributed Simulation and Real Time Applications*, pages 44–53. IEEE, 2012.

[12] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, Somayeh Sardashti, et al. The gem5 simulator. *ACM SIGARCH computer architecture news*, 39(2):1–7, 2011.

[13] Blizzard. Overwatch2. https://overwatch.blizzard.com/, 2023.

[14] Michele Caini. Ecs back and forth (part 1). https://skypjack.github.io/2019-02-14-ecs-baf-part-1/.

[15] DeepSeek-AI and et al. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model, 2024.

[16] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[17] Unreal Engine. Mass entity. https://docs.unrealengine.com/5.0/en-US/mass-entity-in-unreal-engine/, 2023.

[18] Richard Fabian. Data-oriented design. *framework*, 21:1–7, 2018.

[19] Yicheng Feng, Yuetao Chen, Kaiwen Chen, Jingzong Li, Tianyuan Wu, Peng Cheng, Chuan Wu, Wei Wang, Tsung-Yi Ho, and Hong Xu. Echo: Simulating distributed training at scale. *arXiv preprint arXiv:2412.12487*, 2024.

[20] Richard M Fujimoto. *Parallel and distributed simulation systems*, volume 300. Citeseer, 2000.

[21] Kaihui Gao, Li Chen, Dan Li, Vincent Liu, Xizheng Wang, Ran Zhang, and Lu Lu. Dons: Fast and affordable discrete event network simulation with automatic parallelization. In *ACM SIGCOMM*, pages 167–181, 2023.

[22] Chuanxiong Guo, Guohan Lu, Dan Li, Haitao Wu, Xuan Zhang, Yunfeng Shi, Chen Tian, Yongguang Zhang, and Songwu Lu. Bcube: a high performance, server-centric network architecture for modular data centers. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, pages 63–74, 2009.

[23] NVIDIA Inc. Nccl. https://developer.nvidia.com/nccl. (Accessed on 09/14/2024).

[24] Teerawat Issariyakul, Ekram Hossain, Teerawat Issariyakul, and Ekram Hossain. *Introduction to network simulator 2 (NS2)*. Springer, 2009.

[25] Shafagh Jafer, Qi Liu, and Gabriel Wainer. Synchronization methods in parallel and distributed discrete-event simulation. *Simulation Modelling Practice and Theory*, 30:54–73, 2013.

[26] Zhihao Jia, Matei Zaharia, and Alex Aiken. Beyond data and model parallelism for deep neural networks. *Proceedings of Machine Learning and Systems*, 1:1–13, 2019.

[27] Peng Jiang, Christian Sonne, Wangliang Li, Fengqi You, and Siming You. Preventing the immense increase in the life-cycle energy and carbon footprints of llm-powered intelligent chatbots. *Engineering*, 2024.

[28] Ziheng Jiang, Haibin Lin, Yinmin Zhong, Qi Huang, Yangrui Chen, Zhi Zhang, Yanghua Peng, Xiang Li, Cong Xie, Shibiao Nong, et al. {MegaScale}: Scaling large language model training to more than 10,000 {GPUs}. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 745–760, 2024.

[29] Norm Jouppi, George Kurian, Sheng Li, Peter Ma, Rahul Nagarajan, Lifeng Nai, Nishant Patil, Suvinay Subramanian, Andy Swing, Brian Towles, et al. Tpu v4: An optically reconfigurable supercomputer for machine learning with hardware support for embeddings. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*, pages 1–14, 2023.

[30] Tech Journalist. xai colossus supercomputer comes online with 100k nvidia gpus. https://www.techopedia.com/news/elon-musks-xai-colossus-supercomputer-comes\-online-with-100k-nvidia-gpus, 2024.

[31] John Kim, Wiliam J Dally, Steve Scott, and Dennis Abts. Technology-driven, highly-scalable dragonfly topology. *ACM SIGARCH Computer Architecture News*, 36(3):77–88, 2008.

[32] Jean-Yves Le Boudec and Patrick Thiran. *Network calculus: a theory of deterministic queuing systems for the internet*. Springer, 2001.

[33] Hejing Li, Praneeth Balasubramanian, Marvin Meiers, Jialin Li, and Antoine Kaufmann. Splitsim: Large-scale simulations for evaluating network systems research. *arXiv preprint arXiv:2402.05312*, 2024.

[34] Hejing Li, Jialin Li, and Antoine Kaufmann. Simbricks: End-to-end network system evaluation with modular simulation. In *ACM SIGCOMM*, page 380–396, 2022.

[35] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, et al. Hpcc: High precision congestion control. In *Proceedings of the ACM special interest group on data communication*, pages 44–58. 2019.

[36] Zhuohan Li, Lianmin Zheng, Yinmin Zhong, Vincent Liu, Ying Sheng, Xin Jin, Yanping Huang, Zhifeng Chen, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. AlpaServe: Statistical multiplexing with model parallelism for deep learning serving. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, pages 663–679, Boston, MA, July 2023. USENIX Association.

[37] Xinhu Liu and Philipp Andelfinger. Time warp on the gpu: Design and assessment. In *Proceedings of the 2017 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, pages 109–120, 2017.

[38] Zheng Lu and Hongji Yang. *Unlocking the power of OPNET modeler*. Cambridge University Press, 2012.

[39] Marco Ajmone Marsan, Michele Garetto, Paolo Giaccone, Emilio Leonardi, Enrico Schiattarella, and Alessandro Tarello. Using partial differential equations to model tcp mice and elephants in large ip networks. *IEEE/ACM Transactions on Networking*, 13(6):1289–1301, 2005.

[40] Sander Mertens. Flecs 2.0 is out. https://ajmmertens.medium.com/flecs-2-0-is-out-1255a2443fbd.

[41] Alian Mohammad, Umur Darbaz, Gabor Dozsa, Stephan Diestelhorst, Daehoon Kim, and Nam Sung Kim. dist-gem5: Distributed simulation of computer clusters. In *2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 153–162. IEEE, 2017.

[42] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan 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*, pages 1–15, 2021.

[43] NVIDIA. Roft. https://docs.nvidia.com/https:/docs.nvidia.com/

dgx-superpod-reference-architecture-dgx-h100.pdf.

[44] Hyungwook Park and Paul A Fishwick. An analysis of queuing network simulation using gpu-based hardware acceleration. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 21(3):1–22, 2011.

[45] Steven G Parker, James Bigler, Andreas Dietrich, Heiko Friedrich, Jared Hoberock, David Luebke, David McAllister, Morgan McGuire, Keith Morley, Austin Robison, et al. Optix: a general purpose ray tracing engine. *Acm transactions on graphics (tog)*, 29(4):1–13, 2010.

[46] Qiuyu Peng, Anwar Walid, Jaehyun Hwang, and Steven H Low. Multipath tcp: Analysis, design, and implementation. *IEEE/ACM Transactions on networking*, 24(1):596–609, 2014.

[47] Saeed Rashidi, Srinivas Sridharan, Sudarshan Srinivasan, and Tushar 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)*, pages 81–92, 2020.

[48] Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 3505–3506, 2020.

[49] George F Riley and Thomas R Henderson. The ns-3 network simulator. *Modeling and tools for network simulation*, pages 15–34, 2010.

[50] Krzysztof Rusek, José Suárez-Varela, Paul Almasan, Pere Barlet-Ros, and Albert Cabellos-Aparicio. Routenet: Leveraging graph neural networks for network modeling and optimization in sdn. *IEEE Journal on Selected Areas in Communications*, 38(10):2260–2270, 2020.

[51] Janche Sang, Che-Rung Lee, Vernon Rego, and Chung-Ta King. Experiences with implementing parallel discrete-event simulation on gpu. *The Journal of Supercomputing*, 75:4132–4149, 2019.

[52] Brennan Shacklett, Luc Guy Rosenzweig, Zhiqiang Xie, Bidipta Sarkar, Andrew Szot, Erik Wijmans, Vladlen Koltun, Dhruv Batra, and Kayvon Fatahalian. An extensible, data-oriented architecture for high-performance, many-world simulation. *ACM Transactions on Graphics (TOG)*, 42(4):1–13, 2023.

[53] Aakash Sharma, Vivek M Bhasi, Sonali Singh, George Kesidis, Mahmut T Kandemir, and Chita R Das. Gpu cluster scheduling for network-sensitive deep learning. *arXiv preprint arXiv:2401.16492*, 2024.

[54] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

[55] Wenjie Tang and Yiping Yao. A gpu-based discrete event simulation kernel. *Simulation*, 89(11):1335–1354, 2013.

[56] Unity. Data-oriented technology stack (dots). https://unity.com/dots, 2023.

[57] András Varga. A practical introduction to the omnet++ simulation framework. *Recent Advances in Network Simulation: The OMNeT++ Environment and its Ecosystem*, pages 3–51, 2019.

[58] Weiyang Wang, Manya Ghobadi, Kayvon Shakeri, Ying Zhang, and Naader Hasani. Rail-only: A low-cost high-performance network for training llms with trillion parameters. *arXiv preprint arXiv:2307.12169*, 2023.

[59] Weiyang Wang, Moein Khazraee, Zhizhen Zhong, Manya Ghobadi, Zhihao Jia, Dheevatsa Mudigere, Ying Zhang, and Anthony Kewitsch. {TopoOpt}: Co-optimizing network topology and parallelization strategy for distributed training jobs. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 739–767, 2023.

[60] William Won, Taekyung Heo, Saeed Rashidi, Srinivas Sridharan, Sudarshan Srinivasan, and Tushar Krishna. Astra-sim2.0: Modeling hierarchical networks and disaggregated systems for large-model training at scale. In *2023 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 283–294, 2023.

[61] Qingqing Yang, Xi Peng, Li Chen, Libin Liu, Jingze Zhang, Hong Xu, Baochun Li, and Gong Zhang. Deepqueuenet: towards scalable and generalized network performance estimation with packet-level visibility. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pages 441–457, 2022.

[62] Qizhen Zhang, Kelvin KW Ng, Charles Kazer, Shen Yan, João Sedoc, and Vincent Liu. Mimicnet: fast performance estimates for data center networks with machine learning. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, pages 287–304, 2021.

[63] Yibo Zhu, Haggai Eran, Daniel Firestone, Chuanxiong Guo, Marina Lipshteyn, Yehonatan Liron, Jitendra Padhye, Shachar Raindel, Mohamad Haj Yahia, and Ming Zhang. Congestion control for large-scale rdma deployments. *ACM SIGCOMM Computer Communication Review*, 45(4):523–536, 2015.