Towards Compute-Aware In-Switch Computing for LLMs Tensor-Parallelism on Multi-GPU Systems

Chen Zhang[1], Qijun Zhang[1*†], Zhuoshan Zhou[1], Yijia Diao[1], Haibo Wang[2], Zhe Zhou[2], Zhipeng Tu[2], Zhiyao Li[2], Guangyu Sun[3], Zhuoran Song[1], Zhigang Ji[1], Jingwen Leng[1*], Minyi Guo[1]
Shanghai Jiao Tong University[1], Huawei Technologies Co. Ltd.[2], Peking University[3]
{chenzhang.sjtu, zs.zhou, diao_yijia, songzhuoran, zhigangji}@sjtu.edu.cn, qijunzhang2000@gmail.com,
gsun@pku.edu.cn, {wanghaibo33, zhouzhe22, tuzhipeng3, lizhiyao5}@huawei.com, {leng-jw, guo-my}@cs.sjtu.edu.cn

***Abstract*—Tensor parallelism (TP) in large-scale LLM inference and training introduces frequent collective operations that dominate inter-GPU communication. While in-switch computing, exemplified by NVLink SHARP (NVLS), accelerates collective operations by reducing redundant data transfer, its communication-centric design philosophy introduces the mismatch between its communication mode and the memory semantic requirement of LLM's computation kernel. Such a mismatch isolates the compute and communication phases, resulting in underutilized resources and limited overlap in multi-GPU systems.**

To address the limitation, we propose CAIS, the first Compute-Aware In-Switch computing framework that aligns communication modes with computation's memory semantics requirement. CAIS consists of three integral techniques: (1) compute-aware ISA and microarchitecture extension to enable compute-aware in-switch computing. (2) merging-aware TB (Thread Block) coordination to improve the temporal alignment for efficient request merging. (3) graph-level dataflow optimizer to achieve a tight cross-kernel overlap. Evaluations on LLM workloads show that CAIS achieves 1.38× average end-to-end training speedup over the SOTA NVLS-enabled solution, and 1.61× over T3, the SOTA compute-communicate overlap solutions but do not leverage NVLS, demonstrating its effectiveness in accelerating TP on multi-GPU systems.

I. Introduction

The rapid scaling of large language models (LLMs) has pushed distributed training and inference systems to unprecedented scales, where clusters composed of hundreds or even thousands of GPUs must work together seamlessly [6], [7], [33], [50]. To fully utilize such large-scale GPU clusters, hybrid parallelism, combining data parallelism (DP), pipeline parallelism (PP), and tensor parallelism (TP), has become the de facto strategy for scaling. Among them, DP and PP mainly serve to scale out across nodes by distributing data batches and network layers, whereas TP is designed to scale up by partitioning large matrix operations across multiple GPUs [25], [30], [35], [49]. This tensor-level partitioning enables fine-grained parallel execution but also makes TP the most communication-intensive and structurally complex scheme. A recent study [30] reveals that TP contributes to over 99% of total data traffic, while data and pipeline parallelism together account for less than 1%. Furthermore, TP exhibits structurally complex dependencies, as its communication lies on the critical path where computation cannot directly overlap, making optimization inherently difficult [43]. Therefore, as models grow to trillion-parameter scale, TP's communication overhead becomes a first-order bottleneck that fundamentally limits cluster scalability and system efficiency [19], [43].

To overcome this limitation, recent GPU architectures have evolved toward tighter interconnect integration. High-bandwidth NVLink and NVSwitch [41], [42] have become essential enablers for scaling up LLM systems through tensor parallelism, providing terabyte-per-second inter-GPU connectivity. However, as LLMs continue to grow, even these high-speed links face severe pressure from redundant data transfers among GPUs. To further relieve this bottleneck, NVIDIA introduced NVLink SHARP (NVLS) [24], [37], an in-switch computing architecture that performs collective reductions directly within the NVSwitch fabric. By performing reductions in-flight, NVLS reduces redundant GPU-to-GPU transfers and achieves 2–8× speedups for collective primitives compared to GPU-driven implementations [24]. However, NVLS remains fundamentally communication-centric, focusing solely on accelerating collective operations without considering their interplay with computation kernels such as GEMM. This isolation prevents smooth and tight overlap of communication and computation, often leaving GPUs idle during communication phases. This problem is further amplified in large-scale LLM inference and training workloads, where tensor parallelism introduces frequent collective operations that account for up to 40–60% of total latency [19], [43], [52]. While recent work has explored compute-communication overlap through software scheduling [4], [19], [52] and hardware-assisted overlapping [43], none leverage in-switch computing, missing the opportunity to fully exploit NVLS's architectural potential.

The key obstacle lies in the mismatches between the existing NVLS communication primitives and the requirements of TP compute kernels: *the communication mode (i.e., push/pull modes) of NVLS primitives fail to align with the required memory semantics (i.e., read/write) of TP kernels such as GEMM*. For example, an AllGather operation followed by a GEMM often requires on-demand remote *reads*, but NVLS implements AllGather as *push*-based stores with the `multimem.st` instruction, transmitting data eagerly regardless of when the consumer computation is ready. Likewise, a GEMM followed by

* Corresponding author.
† Qijun Zhang led this project during his internship at Shanghai Jiao Tong University.

Reduce-Scatter requires distributed *writes*, but NVLS implements it with a *pull*-based `multimem.ld_reduce` instruction, which forces consumers to fetch data instead of receiving it inline. This misalignment introduces strict global barriers that prevent fine-grained compute-communication overlap.

This limitation motivates a shift toward compute-aware in-switch computing that aligns communication modes with computation's memory semantics requirement. Therefore, we propose CAIS, the first Compute-Aware In-Switch computing framework that aligns the two sides. With CAIS, the computation kernel should directly issue load/reduction instructions for communication following its memory semantic requirement, while the switch automatically performs request merging for these remote accesses.

However, designing such a compute-aware in-switch computing has three challenges: ① Current GPU and NVLS lack necessary ISA and architecture support to express and process compute-aware in-switch computing. ② Even with instruction and architecture supports, independently scheduled TBs across GPUs result in staggered requests, reducing merge efficiency and causing switch buffer contention. ③ Isolated operators make it difficult to exploit the producer-consumer relationships in LLM dataflow graph (DFG), limiting resource utilization. To address these challenges, CAIS co-designs GPU ISA, switch microarchitecture, and graph-level dataflow with three techniques: ❶ CAIS provides compute-aware ISA and microarchitecture extension that enables compute-aware in-switch computing. ❷ CAIS introduces a lightweight compiler-architecture co-design to coordinate TB execution across GPUs, maximizing merge success rate without incurring high synchronization costs. ❸ CAIS integrates a graph-level dataflow optimizer, exploiting fine-grained TB-level overlapped execution to maximize computation and communication resource utilization. To the best of our knowledge, this is the first work to realize fine-grained compute-communication overlap within an in-switch computing paradigm.

We summarize the key contributions as follows:

- We uncover a critical misalignment between the communication semantics of current in-switch primitives and the memory access requirements of LLM compute kernels, resulting in underutilized resources and limited overlap.
- We propose CAIS, the first compute-aware in-switch computing framework that co-designs GPU ISA, switch microarchitecture, and operator dataflows to enable fine-grained, kernel-integrated compute-communicate overlap.
- We implement CAIS in a cycle-accurate simulator and evaluate it on three LLM inference and training workloads, achieving on average 1.38× end-to-end training speedup over NVLS-augmented baselines, and 1.61× over T3 [43], the SOTA compute-communicate overlap solutions but do not leverage NVLS.

The rest of this paper is organized as follows. Section II reviews background and motivates compute-aware in-switch computing. Section III presents the CAIS design, including ISA extensions, micro-architecture design, and the supporting compiler and runtime mechanisms. Section IV describes our experimental methodology. Section V evaluates CAIS on representative LLM training and inference workloads. Section VI discusses related work and Section VII concludes.

II. Background and Motivation

A. LLM and Tensor Parallelism

Large language models (LLMs) such as GPT-4 [2] and LLaMA-3 [12] have surpassed the trillion-parameter scale, driving unprecedented demands for compute and memory bandwidth. These models are dominated by dense linear algebra kernels, particularly general matrix multiplications (GEMMs) and matrix-vector multiplications (GEMVs), which exhibit quadratic or cubic growth in compute and memory complexity as hidden dimensions or attention heads increase. To scale these workloads across multiple GPUs, modern deployments employ hybrid parallelism strategies combining data parallelism (DP) [45], [47], pipeline parallelism (PP) [16], [28], and tensor parallelism (TP) [25], [35], [49].

Among these strategies, TP introduces the heaviest communication overhead, because each FFN or attention layer is split across GPUs and requires frequent inter-GPU collective operations, e.g., AllReduce and AllGather, to aggregate partial results. Unlike DP, which incurs communication only during gradient synchronization, or PP, where communication is confined to activation exchanges between stages, TP creates per-layer synchronization points that scale with model depth and width. Recent studies show that for multi-GPU training with TP, 40–60% of end-to-end latency arises from inter-GPU data transfers [19], [43], [52]. As LLMs scale, communication costs are expected to dominate overall runtime unless new architectural solutions emerge.

Fig. 1(a)(b) illustrates two commonly-used TP strategies. (1) Basic TP partitions the key computational modules of a Transformer, such as the Q, K, and V in Attention, as well as the FFN, across multiple GPUs along the hidden dimension or the head dimension. In Fig. 1(a), f is AllReduce (AR) in the forward pass and no operation in the backward pass. $\overline{f}$ is no operation in the forward pass and AllReduce in the backward pass. (2) TP with Sequence Parallelism (SP), a more recent variant, partitions operations along the sequence length and employs a combination of AllGather (AG) and Reduce-Scatter (RS) operations to eliminate redundant activations. In Fig. 1(b), g is Reduce-Scatter in the forward pass and AllGather in the backward pass. $\overline{g}$ is AllGather in the forward pass and Reduce-Scatter in the backward pass. Although AllReduce in Basic TP is mathematically equivalent to Reduce-Scatter plus AllGather, TP with SP can partition more operations (e.g., LayerNorm) and hence reduces memory consumption for activations across GPUs.

B. NVLink/NVSwitch-based Multi-GPU Systems

Modern AI systems attempt to address communication bottlenecks by coupling dozens or hundreds of GPUs via high-radix NVLink/NVSwitch networks [41], [42]. NVLink has evolved from its first generation [9], delivering 160 GB/s GPU-to-GPU bandwidth on Pascal, to the fifth generation [41]

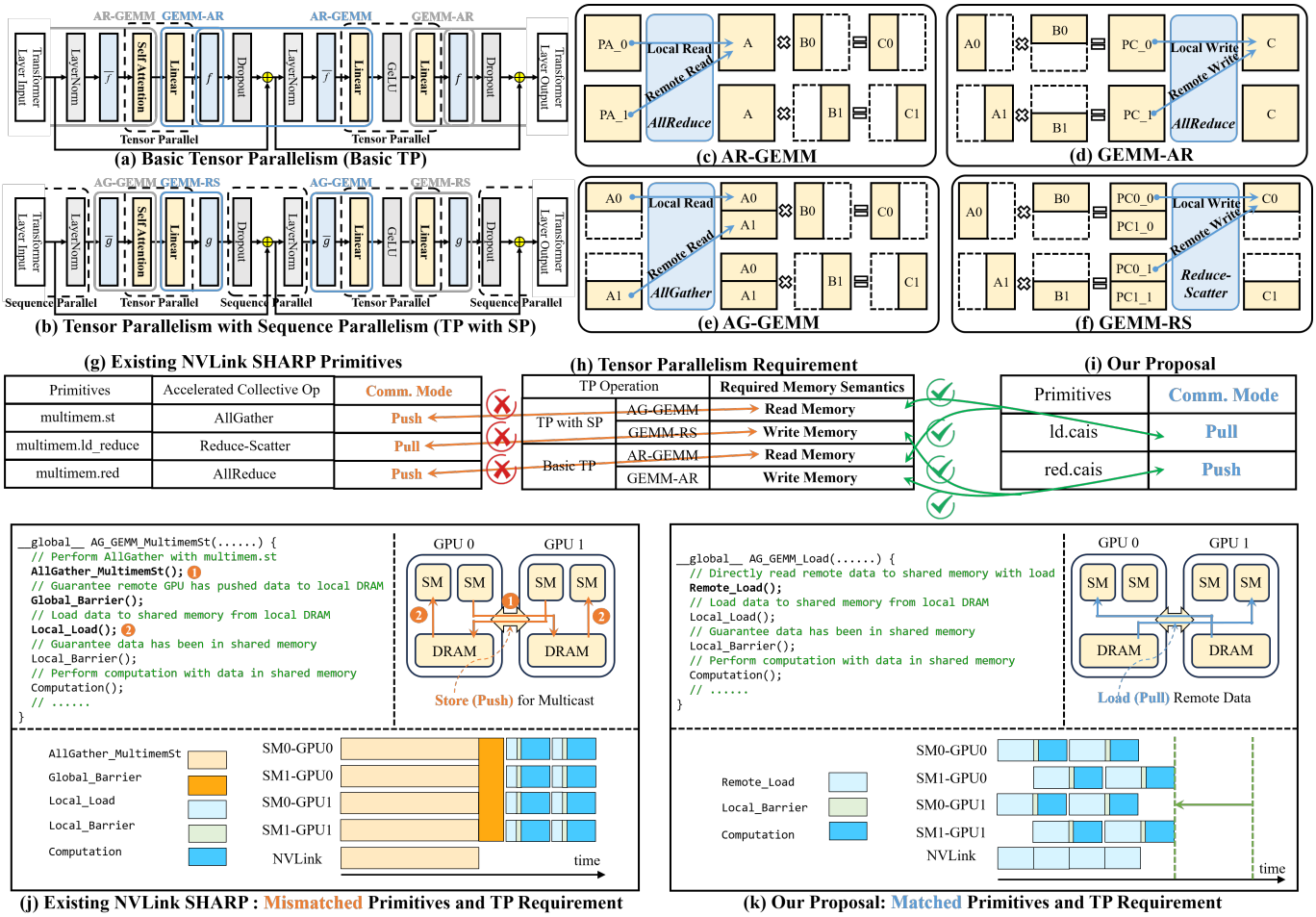

Fig. 1: Motivation for Compute-Aware In-Switch Computing in Tensor Parallelism. (a–b) Tensor Parallelism (TP) in LLM. (c–f) Collective Communication and Computation Kernel Relationship. (g–i) Comparison of Existing NVLS Primitives, Compute-aware TP Requirements and Our Proposal. (j-k) Comparison of Computation Details between Existing NVLS and Our Proposal.

in Blackwell, which provides 1.8 TB/s GPU-to-GPU bandwidth and powers large-scale systems such as NVL72 (72 GPUs) [41], [42]. While these fabrics offer scalable collective operations, their performance remains bounded by link bandwidth. To quantify this limitation, we execute LLaMA-7B on our simulated NVIDIA H100 SuperPOD interconnected via a 900 GB/s NVLink/NVSwitch fabric, varying the number of participating GPUs (see Section IV). As shown in Fig. 2, communication time quickly overtakes computation time once the system scales beyond 4–8 GPUs; In particular, under an 8-GPU configuration, the average communication time is about 1.6× longer than computation across the model. This problem will worsen with future 1T+ parameter models, whose communication volume grows super-linearly due to deeper layers and larger token batches. These observations underscore the urgent need for architectural approaches that reduce or hide communication, rather than merely speeding up links.

In-switch computing has attracted much attention in the computer network community. Many works [5], [8], [10], [11], [13]–[15], [21], [26], [29], [31], [32], [48], [51], [53], [54] have been proposed to accelerate the AllReduce in the distributed system. In recent years, NVIDIA's NVLink SHARP [24], [37] (NVLS) brought in-switch computing into inter-chip network to address these efficiency and scalability bottlenecks of multi-GPU systems. NVLS offloads collective operations (e.g., AllReduce and Reduce-Scatter) to NVSwitch, performing reductions "in-flight" and reducing data movement [24], [37]. NVLS has been supported in modern GPU architecture. With NVIDIA's Hopper GPUs, in-switch operations such as multicast and reduction can be issued via PTX-level `multimem` instructions, including `multimem.st`, `multimem.ld_reduce`, and `multimem.red`, enabling collective operations to be performed inside NVSwitch fabrics. These instructions can, in principle, be embedded in computation kernels such as GEMM to trigger multi-GPU collectives directly, and have become a cornerstone capability in modern systems. The study [24] on NVLS has demonstrated 2×–8× speedups for collective operations compared to GPU-driven communication, thanks to its hardware-accelerated multicast and reduction integrated with NVSwitch.

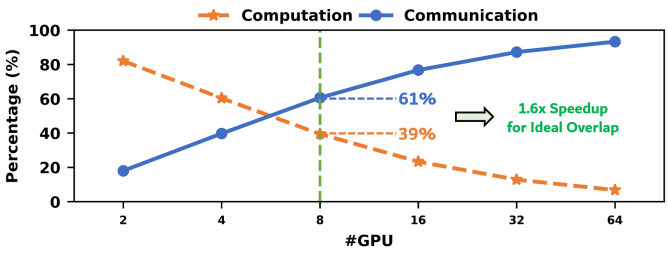

Fig. 2: Computation-Communication Time When Scaling Up.

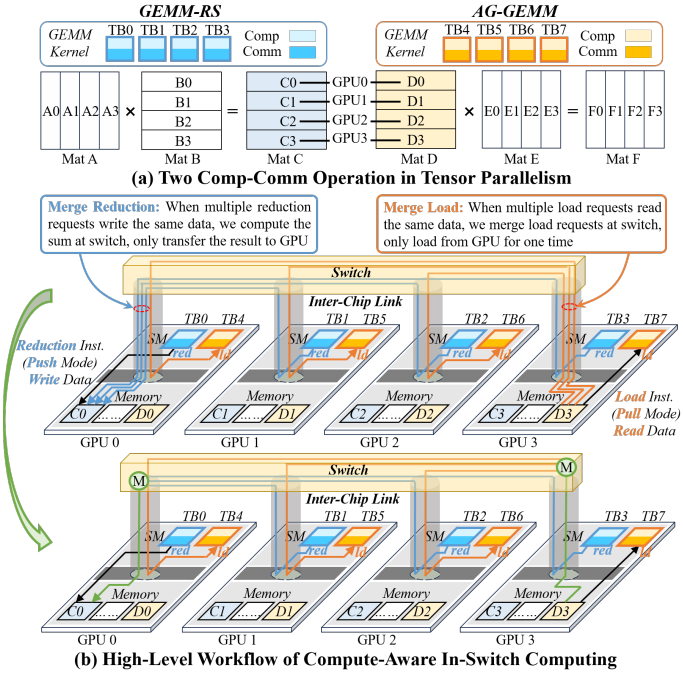

Fig. 3: The System Architecture of CAIS.

C. Limits of Current In-Switch Computing

Despite these advances, current in-switch computing remains fundamentally communication-centric: it is solely designed to accelerate collective operations but remains agnostic to the computation kernels such as GEMM that produce or consume these data streams. For example, AllGather operator collects data chunks from all participating GPUs and redistributes the complete concatenated result back to every GPU. NVLS currently implements this via the `multimem.st` instruction, where each GPU proactively "pushes" its data to all other GPUs as soon as it becomes available. In the context of pure collective communication acceleration for the AllGather operator, push mode provides clear advantages over pull mode. By allowing each GPU to proactively transmit its data to peers as soon as it is ready, push mode forms a continuous, one-way data stream that avoids the roundtrip latency to remote memory inherent in pull-based communication. This also ensures higher bandwidth utilization, as the data pipeline remains fully saturated without idle periods caused by prolonged traffics. Other collective operations follow a similar design paradigm. Fig. 1(g)'s table lists all the NVLS PTX primitives and their corresponding communication modes.

However, this communication-centric design **limits the opportunities for overlapping computation and communication**. For instance, during the forward pass of LLMs, AllGather operation is often immediately followed by a GEMM operator, such as in attention and FFN. As illustrated in Fig. 1(e), the GEMM's computation requires memory **reads** from both local and remote devices. Yet, `multimem.st`, the in-switch computing instruction used for AllGather, operates in **push-mode**. This mismatch between architectural support and workload requirements forces computation and its associated communication to be executed on different GPUs. It introduces global barriers to preserve producer-consumer dependencies. As a result, computation and communication are isolated into separate phases, leaving SMs idle while waiting for synchronization. Fig. 1(j) visualizes this process in detail. Profiling on LLaMA-7B with TP shows that GPU utilization can drop below 60%, even when NVLS is enabled.

To summarize the scope of this mismatch, Fig. 1(c)–(f) illustrate the memory access patterns of the four computation-communication combinations in TP across multiple GPUs, while Fig. 1(g)–(h) summarize the mismatches between the current NVLS design and the requirements of computation kernels. Specifically, AllGather + GEMM (AG-GEMM) requires memory reads, but NVLS only provides `multimem.st` in push mode. GEMM + Reduce-Scatter (GEMM-RS) requires memory writes, but NVLS provides only `multimem.ld_reduce` in pull mode. Similarly, Basic TP with AllReduce + GEMM (AR-GEMM) and GEMM + AllReduce (GEMM-AR) requires both memory reads and writes, while NVLS currently offers only `multimem.red` in push mode.

D. Design Philosophy and Challenges

As previously analyzed, a fundamental mismatch exists between the communication modes supported by current in-switch computing systems and the memory access semantics required by LLM computational kernels, resulting in isolated computation and communication. This limitation motivates a shift toward *compute-aware in-switch computing*. Our philosophy is: *computation kernel should directly issue load/reduction instructions for communication following its memory semantic requirement, while the switch automatically performs request merging for these remote accesses.*

This approach enables semantic alignment between communication mode and computational intent, unlocking more native and tighter communication-computation overlap. As illustrated in Fig. 1(k), taking AG-GEMM as an example, following its memory semantic requirement, computation kernel directly reads remote data via load operations in pull mode. Because both the computation and its remote data access are performed by the same TB, only TB-level local barriers are needed to enforce dependencies between communication and

computation, allowing computation and communication across different SMs to overlap naturally.

The system architecture of CAIS is illustrated in Fig. 3. When a GEMM kernel issues reduction or load instructions (for GEMM-RS and AG-GEMM operations, respectively) to access remote data, the switch dynamically merges these requests to reduce communication overhead: (1) Reduction requests: When multiple reduction requests target the same data, the switch aggregates them by computing the sum and transmitting only the final result to the destination GPU, thus reducing the downstream traffic from switch to GPU. (2) Load requests: When multiple load requests reference the same data, the switch fetches the data only once from the target GPU and replicates it to the requesting GPUs, reducing the upstream traffic from GPU to switch.

However, designing such a compute-aware in-switch computing has three architectural challenges:

(1) Lack of ISA and Microarchitecture Support. Without GPU ISA and switch microarchitecture support, commercial GPU cannot perform compute-aware in-switch computing. To address this limitation, we propose PTX-level instruction extensions for compute-aware in-switch computing and provide microarchitectural support for request merging by integrating a hardware merge unit into the data path at the switch port. These two supports enable the functionality of the compute-aware in-switch computing.

(2) Lack of Temporal Coordination. Even with semantic alignment, temporal misalignment across GPUs [18] remains a critical bottleneck. Thread blocks (TBs) on different GPUs are scheduled independently, resulting in staggered memory requests to the same remote addresses. This misalignment prevents the switch from merging requests effectively, as early-arriving requests must wait for delayed ones, leading to buffer contention and eviction. Our simulation shows that the delay between the earliest and latest requests to the same address averages 35 µs. We address this by introducing a compiler-hardware co-design strategy. The compiler statically groups TBs with shared data dependencies, while the GPU runtime enforces lightweight TB-group synchronization, aligning access timing and improving merge success rates. Our experiments demonstrate that these optimizations reduce the delay to 3 µs, achieving about 10× improvement.

(3) Limited Cross-Kernel Fusion. Prior in-switch computing approaches have struggled to achieve deep cross-kernel fusion due to the aforementioned misalignment, which forces collective kernels operate as isolated phases, making it difficult to exploit the producer-consumer relationships in the LLM dataflow graph (DFG). By resolving it, CAIS enables deeper DFG-level optimizations, allowing TB-level producer-consumer relationships to be established. As soon as one TB of an operator completes, dependent TBs of subsequent (and even further downstream) operators can launch immediately, which greatly improve the computation-communication efficiency. Moreover, we find that this approach also allows CAIS to fuse operators with complementary communication patterns, maximizing overall bandwidth utilization.

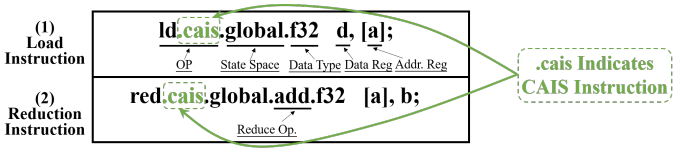

Fig. 4: Extension of the PTX Instructions.

III. CAIS Design

Following the above design philosophy, we introduce CAIS, a compute-aware in-switch computing framework to overcome the limitation of existing communication-centric in-switch computing. The framework consists of three primary components: *1) Compute-Aware ISA and Microarchitecture Extensions*. This is the core design of CAIS, which fundamentally eliminates the global computation–communication barrier by aligning communication modes with the semantic requirements of computation.

Building upon the architectural foundation that removes the global barrier, CAIS further integrates two optimizations: 2) *Multi-GPU TB Coordination* that aligns cross-GPU TB execution using compiler-guided grouping and lightweight in-switch synchronization to maximize temporal locality for request merging. 3) *Graph-Level Dataflow Optimizer* that exploits fine-grained dependency to fuse communication-heavy operator sequences, e.g., GEMM-RS + LN + AG-GEMM, into a single execution pipeline, improving bandwidth utilization and end-to-end performance.

A. Compute-Aware ISA and Microarchitecture Extensions

To support compute-aware in-switch computing, CAIS introduces a co-designed ISA extension and switch microarchitecture that enable dynamic request merging for both load and reduction operations across GPUs. This design transforms the switch from a passive relay into an active compute-aware merging agent, significantly reducing redundant inter-chip traffic and improving execution efficiency for tensor-parallel (TP) workloads.

1) ISA Extention for Mergeable Memory Access: We extend NVIDIA's PTX instruction set with two new instructions: `ld.cais` and `red.cais`, as shown in Fig. 4. These instructions encode a 1-bit CAIS flag in memory access requests, signaling the switch that the request is eligible for in-switch merging. This lightweight annotation allows the system to selectively apply merging to communication-intensive operations such as AllGather loads or ReduceScatter reductions, without modifying existing computation semantics.

2) Switch Micro-architecture for Request Merging: To support CAIS instructions, we enhance NVSwitch datapath with a dedicated merge unit (Fig. 5), mainly consisting of two tables: 1) **CAM Lookup Table** matches incoming requests based on memory address and type (load or reduction). On a match, request is merged into an existing session; otherwise, a new entry is created. 2) **Merging Table** maintains partial results for each session, including cached data for loads or accumulated sums

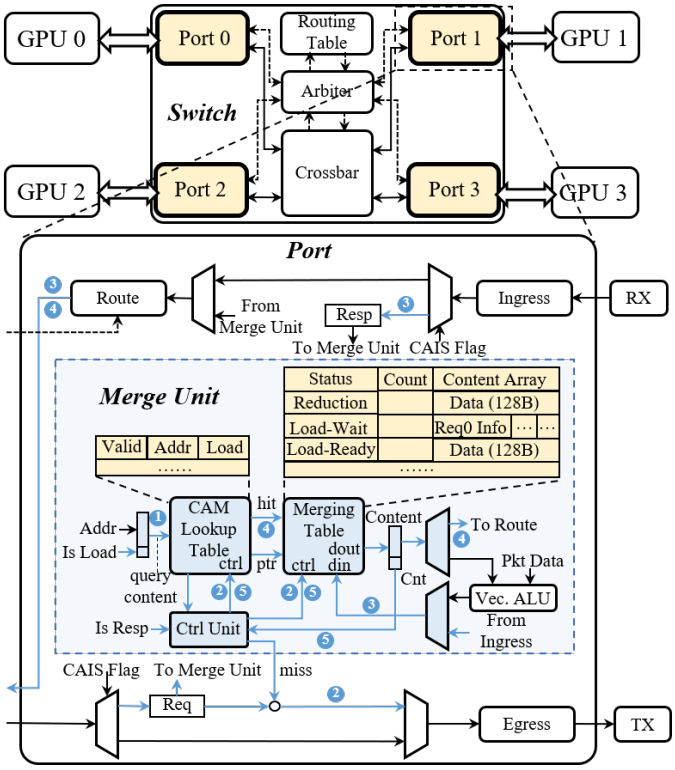

Fig. 5: Switch Micro-architecture for CAIS.

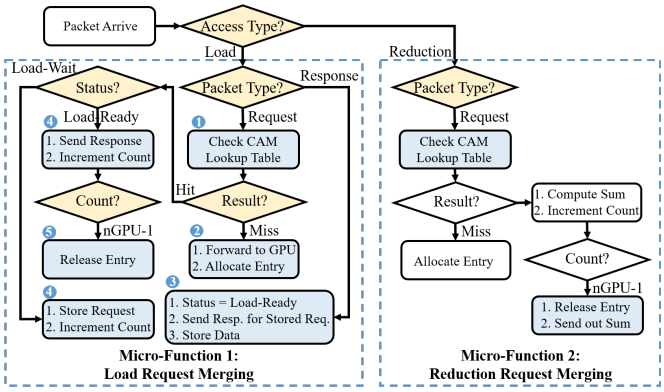

Fig. 6: In-switch Micro-Functions Workflow.

for reductions. Each entry tracks session state (Load-Wait, Load-Ready, or Reduction) and a counter of merged requests.

These tables operate in tandem to perform on-the-fly aggregation of identical accesses across GPUs. When the last contributing request arrives, the merged data is either forwarded to requesters (loads) or written to memory (reductions).

3) In-Switch Micro-Functions for Load and Reduction: With the ISA and switch microarchitecture extensions, CAIS perform request merging with two micro-functions that handle *load* and *reduction* requests inside the NVSwitch. These micro-functions extend the existing NVLS pipeline by performing dynamic request detection, caching, and response generation in-flight, thereby reducing redundant traffic and avoiding unnecessary synchronization. Fig. 6 illustrates the flow of the two micro-functions.

Micro-Function 1: Load Request Merging. Load request merging eliminates redundant load responses. When a `ld.cais` request arrives at the switch, the merge unit first performs an associative search within the CAM Lookup Table to ❶ check for an existing merge entry targeting the same memory address and request type. ❷ If no match is found, a new entry is allocated in both the CAM Lookup Table and the Merging Table. The request is forwarded to the destination GPU through the standard routing path, while the new entry in the Merging Table is initialized with "`Status = Load-Wait, Count = 1`", and the associated request metadata is stored in the `Content Array`. ❸ When the response data from the target GPU returns, the status is updated to `Load-Ready`, and the data is cached in the `Content Array`. The switch also generates responses for requests stored in `Content Array` before caching the arriving data. After that, the switch can serve subsequent requests to the same address directly from this cached data without reissuing memory transactions to the target GPU. ❹ If a later request arrives and hits an active session, the merge unit either appends the request metadata in `Content Array` for deferred response, if the data is still pending, or otherwise immediately generates a response with the cached data in `Content Array`. ❺ The completes and its table entries are released once the `Count` equals the number of participating GPUs minus one, excluding the GPU that holds the local copy.

Micro-Function 2: Reduction Request Merging. Reduction request merging eliminates redundant reduction requests. Similar to load request merging, for `red.cais`, multiple contributions to the same address are accumulated directly within the switch. Once all expected requests are received, the sum is written to the destination memory, avoiding duplicate transmissions. The white blocks in Figure 6 indicate datapaths reused from NVLS.

Through this combination of load and reduction micro-functions, the switch can dynamically merge multiple remote accesses, turning multiple data transmissions into a single consolidated operation.

4) Eviction Mechanism: If a new entry must be allocated but the tables are full, an LRU-based eviction policy is triggered. 1) If the selected entry is for reduction merging, it is directly evicted, and the partial result is sent to the home GPU of its address. 2) If the selected entry is for load merging, entries in the `Load-Ready` state can be safely evicted, whereas those in the `Load-Wait` state are deferred until the response data arrives. In this case, the arriving pending request bypasses the merge unit without triggering further eviction, avoiding thrashing or deadlock.

To handle the remaining requests for the evicted entry, a timeout-based forward-progress mechanism is employed, similar to that in existing NVLS [24]. Each merge entry is equipped with a timer to track the elapsed time since its last access. If this timer exceeds a predefined threshold, the entry is

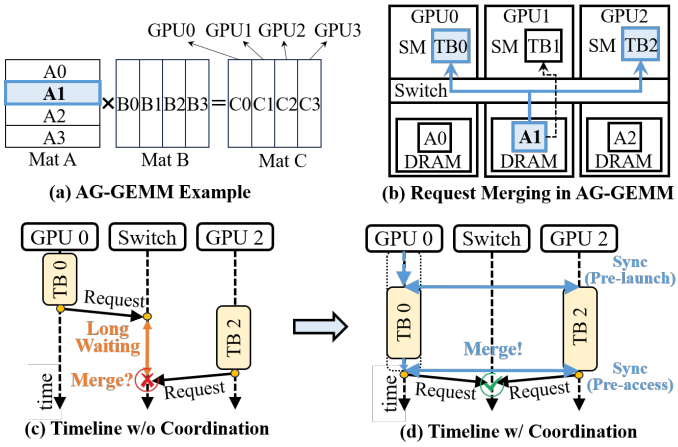

Fig. 7: Merging-aware TB-Group Coordination.

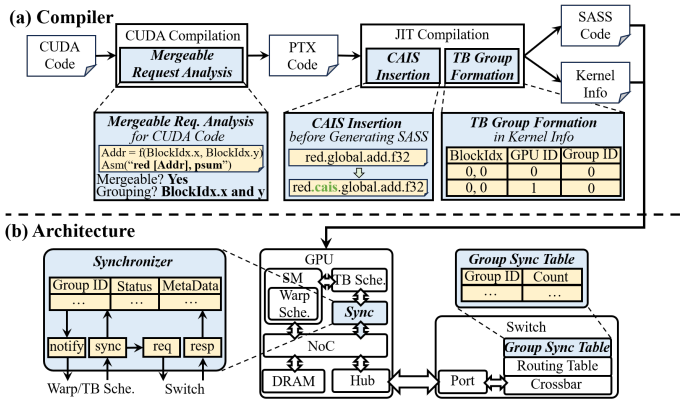

Fig. 8: Compiler and Architecture Support for TB Coordination. The compiler creates TB Groups according to data dependency, and the architecture uses a synchronizer to align request timing across GPUs.

automatically evicted, ensuring that no request remains stalled.

5) Deterministic Routing for Merging Convergence: To ensure all mergeable requests targeting the same address converge at the same switch, CAIS adopts a deterministic routing algorithm similar to that used in existing NVSwitch systems [24]. A lightweight hash function on the request address (or a subset of its bits) maps each request to a fixed path, guaranteeing that matching requests are processed by the same merge unit. Since LLM workloads exhibit regular and predictable access patterns, a simple deterministic routing scheme is sufficient to prevent deadlocks and ensure high link utilization without complex path selection.

B. Cross-GPU TB Coordination

While compute-aware ISA and switch microarchitecture provide the foundation for in-switch request merging, their effectiveness critically depends on the temporal alignment of memory requests across GPUs. In the absence of coordination, mergeable load or reduction requests from different GPUs may arrive at the switch at different times, resulting in missed merging opportunities or buffer pressure due to delayed aggregation. This temporal misalignment is rooted in the fact that TBs are independently scheduled by the GPU runtime, leading to execution drift across devices. Even for the same operator, TBs on different GPUs may issue their memory requests at slightly different times. For in-switch merging to be effective, however, these requests must arrive closely in time, within the lifetime of the Merge Table entry. Otherwise, early-arriving requests may be evicted or bypassed before others arrive, negating the benefits of merging. To address this, CAIS introduces a lightweight coordination mechanism at the granularity of thread blocks, the fundamental parallel execution unit in modern GPU workloads. This coordination enforces temporal locality among semantically equivalent memory requests across GPUs, allowing in-switch merging logic to operate with minimal buffering and maximal aggregation efficiency.

1) Compiler-Guided TB Grouping: **TB-Group Based Coordination.** To ensure temporal locality, CAIS organizes mergeable TBs into logical TB-groups. As is shown in Fig. 7(b), each TB-group contains all TBs across GPUs that access the same data region using CAIS-tagged instructions (e.g., `ld.cais`, `red.cais`). A switch-side merge tracker monitors request arrival patterns for each group and address. Only when requests from all participating GPUs are observed for a given address does the switch perform merging. On the GPU side, each SM uses lightweight hardware counters to track TB-group progress. Once a TB issues a mergeable request, it waits until a local readiness condition is met, either via a credit mechanism or through switch, issued acknowledgment, ensuring alignment with peer TBs on other GPUs.

Compiler Support. CAIS leverages compiler assistance to identify TBs that are likely to issue mergeable memory requests and groups them accordingly. As illustrated in Fig. 8(a), during the CUDA-to-PTX compilation stage, we perform static index analysis on the address expressions of memory access instructions. The analysis detects whether the expression contains the GPU ID. If not, the index is GPU-invariant, indicating that the instruction will access the same memory location when other factors in the expression, such as `blockIdx`, are identical. Consequently, TBs across different GPUs but with the same `blockIdx` are expected to access identical data. The compiler then groups such TBs into logical TB Groups. During JIT compilation, memory access instructions associated with these groups are replaced by their CAIS variants (e.g., `ld.cais`, `red.cais`). Additionally, the compiler attaches TB Group metadata to the kernel launch configuration, which is used by the runtime and switch to guide synchronization and merging behavior.

2) TB Group-Aware Synchronization Mechanisms: To ensure temporal alignment, CAIS introduces two synchronization mechanisms, as shown in Fig. 7(d): (1) **Pre-launch synchronization.** Before a TB is dispatched by the GPU's scheduler, it registers its Group ID with the local synchronizer.

This synchronizer then issues a synchronization request to the switch. The TB remains in a *pending* state until the switch confirms that all GPUs in the group have registered corresponding TBs. Once the switch detects readiness across all participating GPUs, it responds with release signals to each GPU's synchronizer, triggering concurrent TB dispatch. This mechanism ensures aligned launches across devices and prevents early-issued requests from bypassing potential merge opportunities. (2) **Pre-access synchronization.** Even with synchronized launches, compute divergence may cause TBs to reach memory accesses at different times. When a warp encounters its first `*.cais` instruction, it sends a synchronization request tagged with the Group ID. Execution proceeds only after all TBs in the group reach the same point. Meanwhile, the warp scheduler can issue independent instructions to hide synchronization latency.

The synchronization overhead is minimal because it is implemented through the exchange of lightweight empty packets between GPUs and the switch. For each TB, only two empty packets are transmitted between each GPU and the switch. The total latency corresponds merely to the round-trip time between the GPU and the switch, approximately 0.5 µs in our experimental setup, which is negligible compared with the TB execution time. Furthermore, the synchronization scope is strictly confined within each TB group and does not interfere with resource sharing across different TB groups.

TB-Aware Request Throttling. To avoid stalls from outliers, CAIS introduces a TB-aware request throttling strategy. When a GPU detects that it is ahead of its peer TBs in a mergeable group, it temporarily throttles further requests to allow others to catch up. This feedback is driven by the switch's per-address tracking state and exposed to GPUs via a small control interface. Importantly, this throttling is applied only to mergeable TBs, preserving execution parallelism elsewhere.

3) Architecture Support for TB Group Synchronization.: The coordination mechanism is supported by synchronizers on GPUs and a Group Sync Table on the switch, as shown in Fig. 8(b). (1)**On the GPU side,** each device incorporates a synchronizer module that interfaces with the TB and warp schedulers. The synchronizer maintains a small table tracking active TB Groups and handles both pre-launch and pre-access synchronizations by sending TB-group synchronization request to the switch and waiting for a release signal. (2)**On the switch side**, a lightweight Group Sync Table maintains counters for each active TB Group. When synchronization requests from all GPUs are received for a given Group ID, the switch broadcasts a release signal, allowing execution to proceed. This coordination ensures that mergeable requests from different GPUs arrive within a narrow time window, significantly increasing the success rate of request merging.

C. Graph-Level Dataflow Optimizer

CAIS also integrates a graph-level dataflow optimizer to improve the system resource utilization. Graph-level dataflow optimizer supports fine-grained TB-level dependency to unlock tighter kernel fusion opportunities. Built upon the fine-grained TB-level dependency, it introduces Asymmetric Kernel Overlapping to balance complementary traffic between two directions of the inter-chip link, which can significantly improve the overall performance.

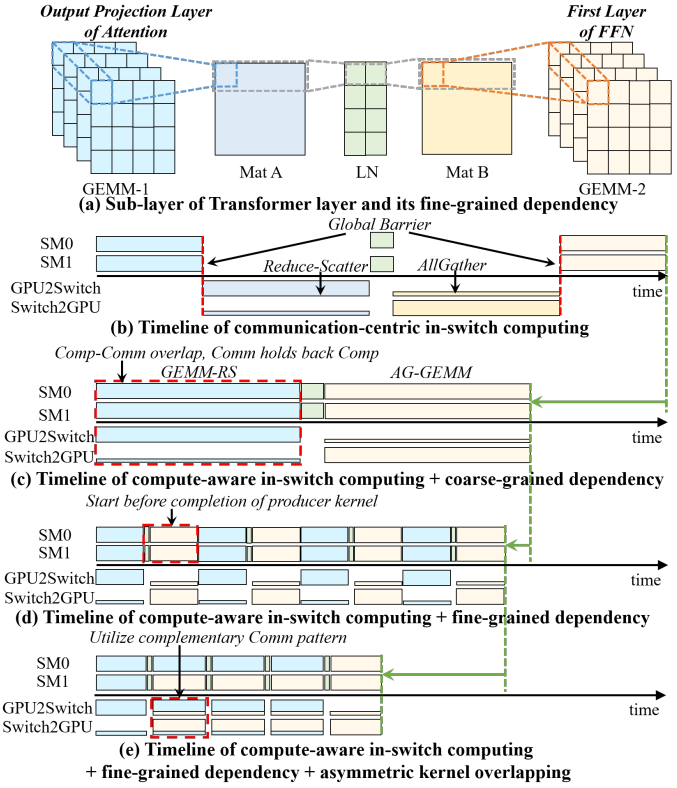

Fig. 9: Graph-Level Dataflow Optimization. Fine-grained TB-level data dependency enables early launch of consumer TBs before producer kernels complete.

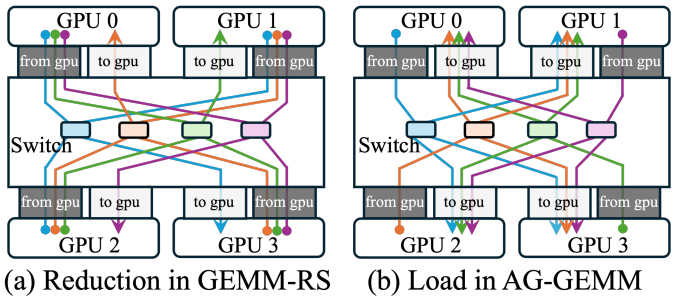

Fig. 10: Illustration of Asymmetric Traffic.

1) Fine-Grained TB Dependency and Deep Kernel Fusion: In contrast to coarse-grained kernel-level dependency that require the full completion of a producer kernel before the consumer kernel can start, fine-grained TB-level dependency [1] allows a TB in the consumer kernel ready to be launched as soon as its input data available, without waiting for the entire producer kernel to finish. This capability enables fused execution of multiple dependent kernels.

Fig. 9(a) illustrates this concept with a portion of a trans-

former layer, where GEMM-1 computes matrix A, followed by a layer normalization (LN) stage producing matrix B, which is then consumed by GEMM-2. In CAIS, TBs in GEMM-1 collaboratively produce tiles of matrix A; each TB in LN operates on a row of A to generate matrix B; GEMM-2 TBs consume tiles of B to compute the final output matrix. Compared to the coarse-grained execution in Fig.9(c), since each TB's input dependencies are localized, execution of GEMM-2 can begin as soon as the corresponding TBs in GEMM-1 and LN complete, unlocking a larger schedule optimization space. As shown in Fig.9(d), this fine-grained chaining enables deep kernel fusion and earlier launch of downstream TBs.

2) Asymmetric Kernel Overlapping: While in-switch merging reduces overall communication volume, it introduces asymmetric bandwidth usage. Operations like GEMM-RS rely on switch-to-GPU reduction traffic, whereas AG-GEMM generates GPU-to-switch load traffic, as is illustrated in Fig. 10. For Fig. 10(a) the reduction operation, operands are read from three GPUs and the result is written back to the destination GPU. This causes the data traffic from GPUs to the switch to be three times higher than that from the switch to the GPUs, creating a bottleneck dominated by the GPU-to-switch path. We refer to this phenomenon as asymmetric traffic. For Fig. 10(b) the load operation, the situation is exactly the opposite: the switch-to-GPU traffic is three times higher than the GPU-to-switch traffic.

CAIS exploit the complementary nature of these two traffic patterns to further optimize kernel fusion and improve overall bandwidth utilization. Using TB-level dependency analysis, CAIS identifies opportunities to pipeline kernels with complementary traffic patterns. For example, when GEMM-RS and AG-GEMM are ready to execute, SMs are partitioned into two groups, each executing one kernel concurrently. This interleaved execution, illustrated in Fig. 9(e), balances bidirectional link usage: as GEMM-RS emits upstream traffic, AG-GEMM consumes downstream data.

Traffic Control. When kernels with asymmetric communication patterns execute concurrently, contention on the G2S link can still arise, particularly when both load and reduction requests compete for bandwidth. CAIS introduces separate virtual channels for load and reduction traffic and uses round-robin arbitration to avoid head-of-line blocking.

Together, deep kernel fusion and asymmetric overlapping maximize the bandwidth utilization and compute resources, delivering significant end-to-end performance improvements.

IV. Experimental Methodology

A. Hardware Configuration

We simulate an 8-GPU system interconnected via four NVSwitch units, replicating the topology of the NVIDIA DGX-H100 [39]. To enable accurate modeling, we extend Accel-Sim [23] with Hopper-specific architectural features and configure the GPU parameters based on the NVIDIA H100 specifications [36]. For multi-GPU communication, we integrate Accel-Sim with a customized BookSim2 [20], enabling concurrent execution across GPUs connected through a switch-based interconnect.

Name	Hidden Size	FFN Hidden Size	Attention Heads	Sequence Length	Batch Size
Mega-GPT-4B	2048	8192	24	1024	16
Mega-GPT-8B	3072	12288	32	1024	12
LLaMA-7B	4096	11264	32	3072	3

TABLE I: LLM Settings Used in Evaluation.

We further modify both Accel-Sim and BookSim2 with custom extensions to support the multimem instructions of NVLS. Specifically, following NVIDIA's NVLS design [24], we augment the "router" in BookSim2 to support in-switch multicast and reduction operations, and extend Accel-Sim to handle the translation from multimem addresses to virtual addresses at the Hub. The quantitative validation of our NVLS simulation is detailed in Section V-E. For fair comparison, we also augment T3 [43] with NVLS support by adopting the DMA-based NVLS design proposed by NVIDIA [24].

The NVLink and NVSwitch are modeled using real device parameters. NVLink is configured with a 16B flit size, a single-flit header, and bidirectional data transfer. NVSwitch employs round-robin arbitration with a 40 KB per-port Merge Table (320 entries) and supports routing to forward requests to their target GPUs. Each input port provides eight 256-depth virtual channels. We implement intra-SM request coalescing, aggregating multiple 32B sector requests into packets of up to 128B to emulate NVLink's burst transfer behavior. Link latency between GPUs and switches (from GPU to switch or from switch to GPU) is configured to 250 ns, resulting in a round-trip latency of approximately 1 µs.

B. Benchmark

We evaluate CAIS using three representative LLMs, summarized in Table I. Both training and inference phases are evaluated, with inference focusing on the communication-heavy prefill stage. The GEMM kernels are implemented using CUTLASS [38]. Due to simulator memory constraints and the long simulation time, simulating full-scale state-of-the-art models is infeasible. To address this limitation, we employ scaled-down LLM variants with key matrix dimensions, including hidden size and FFN hidden size, reduced by 50% compared to state-of-the-art large LLMs. This scaling reduces the computation-to-communication ratio by 50%. To maintain proportionality, we correspondingly reduce the number of SMs by 50%. We validate this scaled-down setup in Section V-E.

C. Baseline

CAIS is evaluated against 9 baselines in four categories, including 6 existing works and 3 NVLS-enhanced baselines.

- *Tensor Parallelism with NVLS* includes **1) Basic TP (TP-NVLS)** [49] that partitions model layers across GPUs and applies AllReduce to merge intermediate results, and **2) TP with Sequence Parallelism (SP-NVLS)** [25] that enhances TP by splitting AllReduce into ReduceScatter and AllGather phases, with layer normalization and

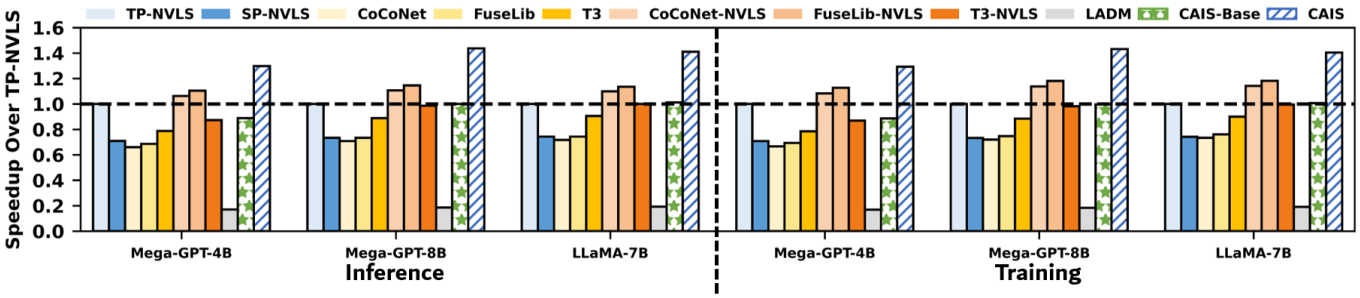

Fig. 11: End-to-End Model Speedup Across Training and Inference.

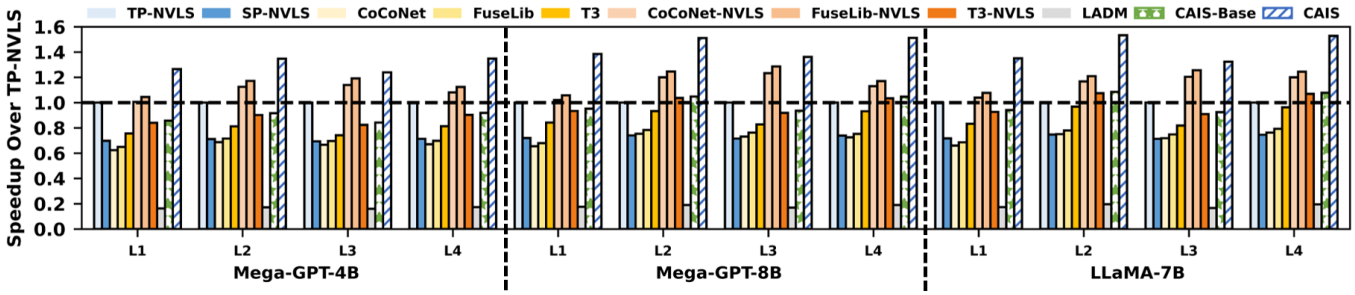

Fig. 12: Sub-layer Performance Speedup.

dropout/Add operations interleaved to reduce memory footprint. NVLS accelerates these collectives.

- *Overlap Solutions* includes **3) CoCoNet** [19] and **4) FuseLib** [44], both of which enable GEMM-AllReduce overlapping through software scheduling techniques, and **5) T3** [43] that introduces hardware-assisted fine-grained overlapping between GEMM and ReduceScatter. We extend T3 to also support AG-GEMM overlap in our evaluation. These solutions do not leverage NVLS.
- *Overlap Solutions with NVLS* includes **6) CoCoNet-NVLS**, **7) FuseLib-NVLS**, and **8) T3-NVLS**, which are enhanced variants of overlap solutions by integrating NVLS support. As introduced in Sec. IV-A, CoCoNet-NVLS and FuseLib-NVLS utilize extended multimem instructions, T3-NVLS adopts a DMA-based NVLS design.
- *Locality-aware TB schedule* places TBs across GPUs/dies for reducing remote access, where we adopt the SOTA, **9) LADM** [22]. LADM cannot utilize NVLS because of its communication-centric design.

V. Experimental Results

A. End-to-End and Sub-Layer Speedup

1) End-to-End Model Speedup: Figure 11 shows the end-to-end speedup of CAIS over nine baseline methods: TP-NVLS, SP-NVLS, CoCoNet, FuseLib, T3, CoCoNet-NVLS, FuseLib-NVLS, T3-NVLS, and LADM. We also include a stripped-down version, CAIS-Base, which disables the proposed merging-aware TB coordination and graph-level dataflow optimizer. For inference, CAIS achieves up to 1.43×, 1.95×, 1.99×, 1.92×, 1.65×, 1.28×, 1.24×, 1.49×, and 7.80× speedup over these nine baselines, with geometric means of 1.38×, 1.89×, 1.98×, 1.90×, 1.61×, 1.25×, 1.21×, 1.45×, and 7.60×, respectively. For training, CAIS achieves up to 1.44×, 1.96×, 2.03×, 1.96×, 1.65×, 1.30×, 1.25×, 1.49×, and 7.75× speedup over these baselines with geometric means of 1.37×, 1.89×, 1.96×, 1.89×, 1.60×, 1.23×, 1.20×, 1.45×, and 7.59×, respectively. These results highlight the significant performance advantages obtained by our proposed compute-aware in-switch computing, as well as leveraging architectural and scheduling co-design optimization to improve the temporal alignment and enhance bandwidth utilization.

2) Sub-Layer Performance: Figure 12 reports the performance of four communication-intensive sub-layers across the model execution flow: [L1] Output projection → LayerNorm → First FFN layer (forward); [L2] Second FFN layer → LayerNorm → Input projection (forward); [L3] First FFN layer → LayerNorm → Output projection (backward); [L4] Input projection → LayerNorm → Second FFN layer (backward).

These sub-layers involve GEMM-RS + LN + AG-GEMM, making them ideal candidates for graph-level optimization in CAIS. CAIS consistently outperforms all baselines across these sub-layers, with up to 1.53×, 2.05×, 2.11×, 2.04×, 1.67×, 1.36×, 1.31×, 1.51×, and 8.08× speedup over TP-NVLS, SP-NVLS, CoCoNet, FuseLib, T3, CoCoNet-NVLS, FuseLib-NVLS, T3-NVLS, and LADM, and corresponding geometric means of 1.39×, 1.91×, 1.99×, 1.91×, 1.64×, 1.24×, 1.20×, 1.47×, and 7.90× speedup.

3) Discussions and Analysis: The improvements over TP-NVLS and SP-NVLS mainly stem from CAIS's native fine-grained computation-communication overlap derived from the

compute-aware in-switch computing. Unlike communication-centric in-switch computing, where the global barrier isolates computation and communication phases, compute-aware in-switch computing only requires the TB-level local barrier, enabling native fine-grained overlapping between computation and communication across different TBs. Besides the computation-communication isolation, SP-NVLS also suffers from the low bandwidth utilization incurred from asymmetric communication patterns when accelerating Reduce-Scatter and AllGather with in-switch computing.

CoCoNet-NVLS, FuseLib-NVLS, and T3-NVLS represent the NVLS-enhanced performance of CoCoNet, FuseLib, and T3. Compared to CoCoNet-NVLS and FuseLib-NVLS, CAIS supports flexible overlapping that also overlaps the communication with the following GEMM. CAIS also eliminates the need for code modification when implementing kernel fusion and mitigates resource contention between compute and communication kernels in CoCoNet-NVLS. Although FuseLib-NVLS executes within a single fused kernel, thereby eliminating kernel-launch overhead and mitigating resource contention, CAIS achieves higher efficiency by enabling more flexible and fine-grained overlap between computation and communication. Compared to T3-NVLS, a hardware-based overlapping solution, CAIS demonstrates notable gains. T3-NVLS still suffers from coarse-grained dependency among ReduceScatter, LN, and AllGather stages, which prevents fine-grained optimization opportunities therefore still cannot tackle the asymmetric communication pattern. In contrast, CAIS's fine-grained TB-level scheduling and asymmetric kernel overlapping unlock additional concurrency and balance inter-GPU bandwidth across two directions. Finally, LADM, though a state-of-the-art locality-aware TB scheduling method for general GPU kernels, focuses on intra-GPU locality rather than inter-GPU communication, and does not leverage NVLS-based acceleration, limiting its applicability to communication-intensive TP workloads.

Comparing CAIS with CAIS-Base reveals the impact of our merging-aware TB coordination and graph-level dataflow optimizer. The maximum and geomean speedups of 1.49× and 1.45× on end-to-end models 1.46× and 1.43× on end-to-end models inference and 1.46× and 1.42× on training, and 1.51× and 1.47× on sub-layers, respectively. These results confirm that merely breaking the global barrier through compute-aware ISA and microarchitectural extensions is insufficient; fully realizing the performance potential requires further optimization within the unlocked scheduling space, leveraging temporal locality and graph-level dataflow integration.

B. Detailed Performance Analysis

This section investigates the effectiveness of key architectural techniques within CAIS, including merging-aware TB coordination and graph-level dataflow optimizer.

1) Impact of Merging-Aware TB Coordination: The merging-aware TB coordination mechanism significantly reduces the waiting time for request merging at the switch by improving the temporal alignment of memory requests

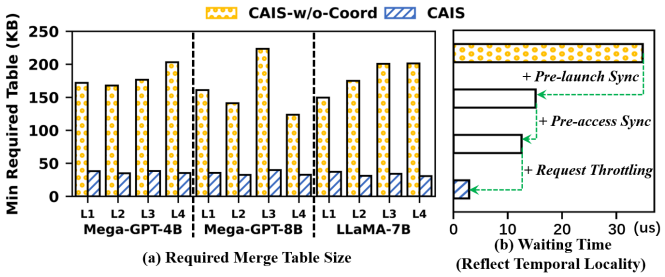

Fig. 13: (a) Required Merge Table Size with and without Merging-Aware TB Coordination. CAIS reduces the minimal table size needed to merge all eligible requests by 87%. (b) Ablation studies for TB coordination.

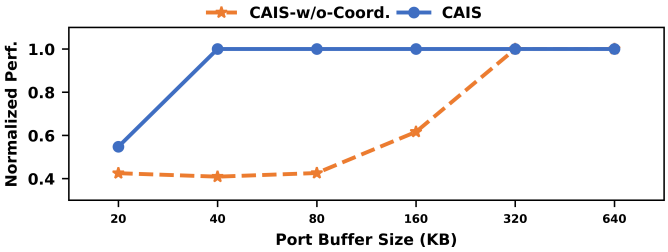

Fig. 14: Performance Sensitivity to Merge Table Size. CAIS maintains high performance with small table sizes, while the uncoordinated version degrades rapidly.

across GPUs. Figure 13 reports the minimal Merge Table size required to merge all mergeable requests for each sub-layer. Without coordination (CAIS-w/o-Coord), the minimal required table size can reach up to 250 KB per port. With coordination enabled, the minimal required table size drops below 40 KB across all ports, which is an 87% reduction in minimal required table size. This result suggests that our coordination strategy can achieve a more effective use of limited switch resources. Figure 13(a) also demonstrates that the minimal required table sizes of CAIS are insensitive to the model sizes and configurations, and are consistently below 40 KB under different model sizes and configurations.

Figure 13(b) further evaluates the effectiveness of each optimization. We measure the improvement using the average waiting time, defined as the delay between the earliest and latest requests targeting the same address. This metric directly reflects the temporal locality optimized by TB coordination. The results show that each optimization step progressively enhances temporal locality, reducing the waiting time from 35 µs to less than 3 µs.

Figure 14 complements this analysis by showing how coordination affects performance under varying Merge Table sizes for the LLaMA-7B model. Merging-aware TB coordination maintains high performance even when the switch buffer is small, while the uncoordinated version degrades rapidly. These comparisons emphasize the importance of merging-aware TB coordination for compute-aware in-switch computing.

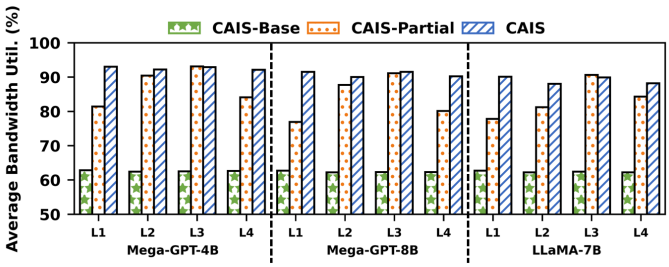

Fig. 15: Average Bandwidth Utilization per Sub-layer.

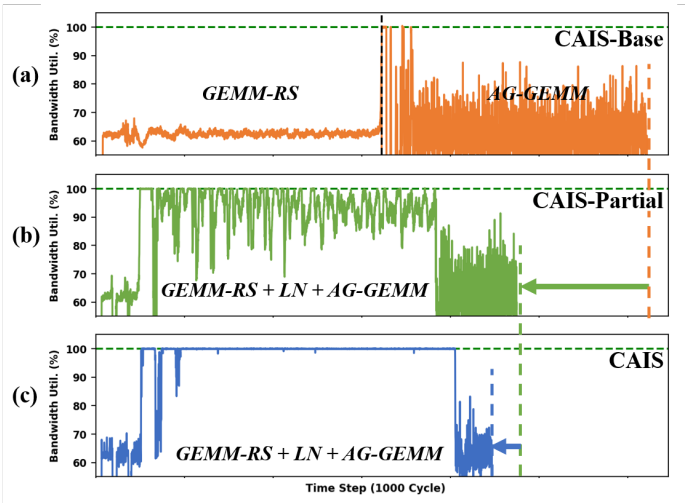

Fig. 16: Bandwidth Utilization over Time for (a) CAIS-Base, (b) CAIS-Partial, and (c) CAIS.

2) Impact of Graph-Level Dataflow Optimizer: Our proposed graph-level dataflow optimizer allows concurrent execution of dependent kernels with complementary asymmetric communication patterns. This optimization improves overall bandwidth utilization by balancing traffic across GPU-to-switch and switch-to-GPU links.

Figure 15 illustrates this effect by comparing the average bandwidth utilization, which is the average across all links and two directions for each link, for all sub-layers of three configurations: (a) CAIS-Base, (b) CAIS with graph-level dataflow optimizer but without traffic control (CAIS-Partial), and (c) full CAIS. Bandwidth utilization improves from 62.4% (CAIS-Base) to 84.7% (CAIS-Partial) and 90.2% (CAIS). The gain from CAIS-Base to CAIS-Partial comes from asymmetric kernel overlapping that tackles the imbalance data movement in both link directions, while the final jump to CAIS reflects the benefit of traffic control.

To further analyze the sustained behavior of these improvements, Figure 16 presents the bandwidth utilization over time for the L2 sub-layer of LLaMA-7B. CAIS maintains near-peak utilization ($\sim$100%) during steady-state operation, while the partial configuration (CAIS-Partial) suffers dips due to contention. The base configuration shows the lowest and most fluctuating utilization. This demonstrates the importance of dataflow optimization and traffic control.

Together, these analyses demonstrate that the graph-level dataflow optimizer is essential to unlocking the full potential of compute-aware in-switch computing.

C. Scalability Analysis

1) Performance Scalability: To demonstrate the performance scalability of CAIS, we evaluate CAIS and CoCoNet-NVLS across different numbers of GPUs based on the LLaMA-7B model. We also scale the model's hidden dimension proportionally to the number of GPUs to avoid under-utilization of computation resources. Figure 17 shows the performance scalability of CAIS and CoCoNet-NVLS, where we measure per-GPU computation throughput normalized to 8-GPU CAIS. It shows that the per-GPU throughput decreases slightly when increasing the number of GPUs. Even with 32 GPUs, the performance drop is still within 5% compared with 8 GPUs. Both CAIS and CoCoNet-NVLS can consistently have superior performance, regardless of the number of GPUs.

2) Hardware Cost Scalability: CAIS hardware exhibit excellent scalability as the number of GPUs increases, owing to a constant and low upper bound on the total required merging table size at the system level. This table size corresponds to the amount of data associated with all outstanding remote requests in the system. CAIS leverages merge-aware coordination to synchronize potentially mergeable requests before they are issued to the switch. This ensures that all GPUs issue outstanding requests for the same set of data, enabling deterministic merging at the switch. As a result, the total required merging table size is bounded by the outstanding remote requests from a single GPU, rather than scaling with the number of GPUs. In addition, CAIS's request throttling mechanism further limits the number of outstanding remote requests per GPU, reducing the actual memory footprint. In our evaluated 8-GPU system, the system-wide upper bound is 1280 KB, corresponding to just 40 KB per switch port. Importantly, this bound remains unchanged even as the number of GPUs increases. Therefore, as the system scales, the relative hardware overhead decreases, since the size of the base switch circuitry grows with GPU count while the enhanced logic remains constant. This bounded memory overhead ensures the scalability and practicality of CAIS's hardware extensions across large-scale multi-GPU deployments.

D. Hardware Overhead

We evaluate our hardware overhead under TSMC's 12nm process technology. Our hardware modification for the switch occupies about $0.50mm^2$, which is less than 1% of NVIDIA's NVSwitch die [17], [24]. On the GPU side, the added logic for TB-group-based synchronization consumes only $0.019mm^2$ per die, less than 0.01% of the H100 GPU area. These results confirm that our proposed architectural support is cost-effective and hardware-feasible.

E. Methodology Validation

This section validates our experimental methodology from two aspects: 1) fidelity of a scaled-down configuration for both

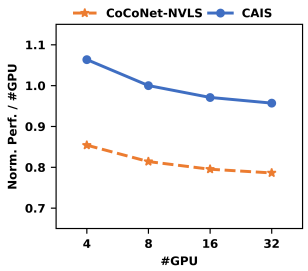

Fig. 17: Scalability with Increasing GPU Count

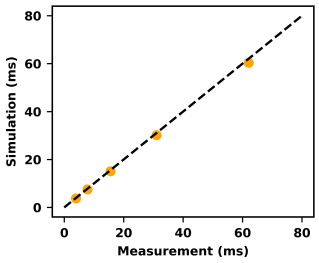

Fig. 18: Validation of Our Simulated NVLS.

Setup	Hidden Size	FFN Hidden Size	Attention Heads	# SM	CAIS Speedup Over TP-NVLS
Full	8192	22528	64	132	**1.43**
Half	4096	11264	32	66	**1.40**

TABLE II: Experimental Validation of Scaling-down Setup.

LLM size and GPU resources, and 2) accuracy of our NVLS-enabled simulator with support for `multimem` instructions.

For scaled-down setup, we compare two systems: a full-scale GPU executing a full-sized LLM that is feasible on Accel-Sim, and a half-scale system with 50% fewer SMs running the same model with matrix dimensions halved. As reported in Table II, the half-scale configuration faithfully reproduces full-scale speedup ordering and magnitudes, preserving system-level behavior and key insights derived.

To validate NVLS support in simulation, we measure AllReduce performance using NCCL [40] on both real hardware and our simulator across message sizes from 1 GB to 16 GB (1, 2, 4, 8, 16 GB). As shown in Fig. 18, simulated results closely match the real-system measurements, yielding high fidelity with an average error of only 3.87%.

VI. Related Work

To mitigate the communication bottleneck, prior works have proposed various optimizations. These can be broadly categorized into two types: 1) communication-computation overlapping [4], [19], [43], [44], [52] and 2) memory efficiency improvement [25], [27]. CoCoNet [19] pioneered the idea of overlapping computation and AllReduce in TP through software pipelining, while [44] further reduces kernel-launch overhead using a similar software-based approach. T3 [43] proposes hardware primitives to enable fine-grained kernel overlap. Centauri [4] proposes overlapping in hybrid parallelism. However, directly applying these techniques to NVLS still fails to achieve effective overlap, since NVLS follows a communication-centric philosophy that lacks awareness of computation semantics, a limitation that CAIS explicitly resolves. Although ACE [46] adopts an in-network computing approach, it primarily reduces DRAM data accesses. Some prior studies have also explored locality-aware TB scheduling [3], [22], [34], but they primarily reduce remote memory access volume across GPUs. CAIS aims to enhance the temporal locality for better request merging.

VII. Conclusion

We present CAIS, a compute-aware in-switch computing framework that provides aligned communication mode with LLM computation kernels. CAIS advances NVLS from a communication-centric primitive accelerator to a compute-integrated, semantics-aligned in-switch computing framework, enabling fine-grained overlap that improves end-to-end LLM system efficiency. CAIS introduces the compute-aware ISA and microarchitecture extension to enable compute-aware in-switch computing, and integrates merge-aware TB coordination and graph-level dataflow optimizer to unlock the full potential of compute-aware in-switch computing. Our evaluation on representative LLMs shows that CAIS delivers significant performance gains over the SOTA solutions.

VIII. Acknowledgment

We sincerely thank the anonymous HPCA'26 reviewers for their valuable suggestions that improved the paper. This work is supported by the Shanghai QiYuan Innovation Foundation and Natural Science Foundation of Shanghai (NSFS) grant 25ZR1402275, National Natural Science Foundation of China (NSFC) grant (62502305 and 62222210), National Key R&D Program of China under Grant 2022YFB4501400.

References

[1] A. Abdolrashidi, H. A. Esfeden, A. Jahanshahi, K. Singh, N. Abu-Ghazaleh, and D. Wong, "Blockmaestro: Enabling programmer-transparent task-based execution in gpu systems," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 333–346.

[2] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[3] A. Arunkumar, E. Bolotin, B. Cho, U. Milic, E. Ebrahimi, O. Villa, A. Jaleel, C.-J. Wu, and D. Nellans, "Mcm-gpu: Multi-chip-module gpus for continued performance scalability," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 2, pp. 320–332, 2017.

[4] C. Chen, X. Li, Q. Zhu, J. Duan, P. Sun, X. Zhang, and C. Yang, "Centauri: Enabling efficient scheduling for communication-computation overlap in large model training via communication partitioning," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, 2024, pp. 178–191.

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

[6] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[7] A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Yang, A. Fan *et al.*, "The llama 3 herd of models," *arXiv e-prints*, pp. arXiv–2407, 2024.

[8] J. Fei, C.-Y. Ho, A. N. Sahu, M. Canini, and A. Sapio, "Efficient sparse collective communication and its application to accelerate distributed deep learning," in *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, 2021, pp. 676–691.

[9] D. Foley and J. Danskin, "Ultra-performance pascal gpu and nvlink interconnect," *IEEE Micro*, vol. 37, no. 2, pp. 7–17, 2017.

[10] N. Gebara, "In-network aggregation for shared machine learning clusters," *Proceedings of Machine Learning and Systems (MLSys)*, 2021.

[11] R. L. Graham, D. Bureddy, P. Lui, H. Rosenstock, G. Shainer, G. Bloch, D. Goldenerg, M. Dubman, S. Kotchubievsky, V. Koushnir *et al.*, "Scalable hierarchical aggregation protocol (sharp): A hardware architecture for efficient data reduction," in *2016 First International Workshop on Communication Optimizations in HPC (COMHPC)*. IEEE, 2016, pp. 1–10.

[12] A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Vaughan *et al.*, "The llama 3 herd of models," *arXiv preprint arXiv:2407.21783*, 2024.

[13] P. Haghi, W. Krska, C. Tan, T. Geng, P. H. Chen, C. Greenwood, A. Guo, T. Hines, C. Wu, A. Li *et al.*, "Flash: Fpga-accelerated smart switches with gcn case study," in *Proceedings of the 37th International Conference on Supercomputing*, 2023, pp. 450–462.

[14] P. Haghi, C. Tan, A. Guo, C. Wu, D. Liu, A. Li, A. Skjellum, T. Geng, and M. Herbordt, "Smartfuse: Reconfigurable smart switches to accelerate fused collectives in hpc applications," in *Proceedings of the 38th ACM International Conference on Supercomputing*, 2024, pp. 413–425.

[15] Y. He, W. Wu, Y. Le, M. Liu, and C. Lao, "A generic service to provide in-network aggregation for key-value streams," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, 2023, pp. 33–47.

[16] 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," *Advances in neural information processing systems*, vol. 32, 2019.

[17] A. Ishii, D. Foley, E. Anderson, B. Dally, G. Dearth, L. Dennison, M. Hummel, and J. Schafer, "Nvswitch and dgx-2," in *Hot Chips*, 2018.

[18] R. Jain, B. Tran, K. Chen, M. D. Sinclair, and S. Venkataraman, "Pal: A variability-aware policy for scheduling ml workloads in gpu clusters," in *SC24: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2024, pp. 1–18.

[19] A. Jangda, J. Huang, G. Liu, A. H. N. Sabet, S. Maleki, Y. Miao, M. Musuvathi, T. Mytkowicz, and O. Saarikivi, "Breaking the computation and communication abstraction barrier in distributed machine learning workloads," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2022, pp. 402–416.

[20] N. Jiang, D. U. Becker, G. Michelogiannakis, J. Balfour, B. Towles, D. E. Shaw, J. Kim, and W. J. Dally, "A detailed and flexible cycle-accurate network-on-chip simulator," in *2013 IEEE international symposium on performance analysis of systems and software (ISPASS)*. IEEE, 2013, pp. 86–96.

[21] X. Jin, X. Li, H. Zhang, R. Soulé, J. Lee, N. Foster, C. Kim, and I. Stoica, "Netcache: Balancing key-value stores with fast in-network caching," in *Proceedings of the 26th Symposium on Operating Systems Principles*, 2017, pp. 121–136.

[22] M. Khairy, V. Nikiforov, D. Nellans, and T. G. Rogers, "Locality-centric data and threadblock management for massive gpus," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 1022–1036.

[23] M. Khairy, Z. Shen, T. M. Aamodt, and T. G. Rogers, "Accel-sim: An extensible simulation framework for validated gpu modeling," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 473–486.

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

[25] V. A. Korthikanti, J. Casper, S. Lym, L. McAfee, M. Andersch, M. Shoeybi, and B. Catanzaro, "Reducing activation recomputation in large transformer models," *Proceedings of Machine Learning and Systems*, vol. 5, pp. 341–353, 2023.

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

[27] S. Li, F. Xue, C. Baranwal, Y. Li, and Y. You, "Sequence parallelism: Long sequence training from system perspective," *arXiv preprint arXiv:2105.13120*, 2021.

[28] S. Li and T. Hoefler, "Chimera: efficiently training large-scale neural networks with bidirectional pipelines," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2021, pp. 1–14.

[29] 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*, 2019, pp. 279–291.

[30] H. Liao, "Ub-mesh: An new interconnection technology for large ai supernode," in *2025 IEEE Hot Chips 37 Symposium (HCS)*. IEEE Computer Society, 2025, pp. 1–13.

[31] M. Liu, L. Luo, J. Nelson, L. Ceze, A. Krishnamurthy, and K. Atreya, "Incbricks: Toward in-network computation with an in-network cache," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, 2017, pp. 795–809.

[32] S. Liu, Q. Wang, J. Zhang, W. Wu, Q. Lin, Y. Liu, M. Xu, M. Canini, R. C. Cheung, and J. He, "In-network aggregation with transport transparency for distributed training," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, 2023, pp. 376–391.

[33] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 10 012–10 022.

[34] U. Milic, O. Villa, E. Bolotin, A. Arunkumar, E. Ebrahimi, A. Jaleel, A. Ramirez, and D. Nellans, "Beyond the socket: Numa-aware gpus," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, 2017, pp. 123–135.

[35] 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*, 2021, pp. 1–15.

[36] NVIDIA, "Nvidia h100 tensor core gpu." *https://www.nvidia.com/en-us/data-center/h100*, 2022.

[37] NVIDIA, "Upgrading multi-gpu interconnectivity with the third-generation nvidia nvswitch." *https://developer.nvidia.com/blog/upgrading-multi-gpu-interconnectivity-with-the-third-generation-nvidia-nvswitch/?ncid=so-nvsh-708451*, 2022.

[38] NVIDIA, "Cuda templates for linear algebra subroutines." *https://github.com/NVIDIA/cutlass*, 2024.

[39] NVIDIA, "Introduction to nvidia dgx h100/h200 systems." *https://docs.nvidia.com/dgx/dgxh100-user-guide/introduction-to-dgxh100.html*, 2024.

[40] NVIDIA, "Nvidia collective communication library (nccl) documentation." *https://docs.nvidia.com/deeplearning/nccl/user-guide/docs/index.html*, 2024.

[41] NVIDIA, "Nvidia gb200 nvl72." *https://www.nvidia.com/en-us/data-center/gb200-nvl72/*, 2024.

[42] NVIDIA, "Cuda: New features and beyond." *https://www.nvidia.com/en-us/on-demand/session/gtc25-s72383/*, 2025.

[43] S. Pati, S. Aga, M. Islam, N. Jayasena, and M. D. Sinclair, "T3: Transparent tracking & triggering for fine-grained overlap of compute & collectives," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, 2024, pp. 1146–1164.

[44] K. Punniyamurthy, K. Hamidouche, and B. M. Beckmann, "Optimizing distributed ml communication with fused computation-collective operations," in *SC24: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2024, pp. 1–17.

[45] S. Rajbhandari, J. Rasley, O. Ruwase, and Y. He, "Zero: Memory optimizations toward training trillion parameter models," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2020, pp. 1–16.

[46] 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 *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 540–553.

[47] J. Rasley, S. Rajbhandari, O. Ruwase, and Y. He, "Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters," in *Proceedings of the 26th ACM SIGKDD international*

conference on knowledge discovery & data mining, 2020, pp. 3505–3506.

[48] 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 *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, 2021, pp. 785–808.

[49] 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 preprint arXiv:1909.08053*, 2019.

[50] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[51] R. Wang, D. Dong, F. Lei, J. Ma, K. Wu, and K. Lu, "Roar: A router microarchitecture for in-network allreduce," in *Proceedings of the 37th International Conference on Supercomputing*, 2023, pp. 423–436.

[52] S. Wang, J. Wei, A. Sabne, A. Davis, B. Ilbeyi, B. Hechtman, D. Chen, K. S. Murthy, M. Maggioni, Q. Zhang *et al.*, "Overlap communication with dependent computation via decomposition in large deep learning models," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, 2022, pp. 93–106.

[53] M. Yang, A. Baban, V. Kugel, J. Libby, S. Mackie, S. S. R. Kananda, C.-H. Wu, and M. Ghobadi, "Using trio: juniper networks' programmable chipset-for emerging in-network applications," in *Proceedings of the ACM SIGCOMM 2022 Conference*, 2022, pp. 633–648.

[54] Y. Yuan, O. Alama, J. Fei, J. Nelson, D. R. Ports, A. Sapio, M. Canini, and N. S. Kim, "Unlocking the power of inline {Floating-Point} operations on programmable switches," in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, 2022, pp. 683–700.