

MKERNEL: Fast Multi-GPU, Multi-Node Fused Kernels

Ziming Mao¹, Yihan Zhang², Shawn Wei Chew³, Shuang Ma², Costin Raiciu⁴, Yang Zhou², Scott Shenker¹, Ion Stoica¹

¹UC Berkeley, ²UC Davis, ³UCLA, ⁴University Politehnica of Bucharest

Communication has become a bottleneck in distributed training and inference of large models. Overlapping communication with computation at the granularity of kernels, on separate streams, reduces only part of this communication cost. Fused kernels often have better performance by transmitting each output tile as soon as it is produced, but existing fused kernels are largely confined to a single NVLink domain. We present MKERNEL, a library of **multi-GPU, multi-node fused kernels that overlap computation, intra-node NVLink communication, and inter-node RDMA at tile granularity**. MKERNEL partitions the streaming multiprocessors (SMs) of a persistent kernel into compute and communication roles, and an on-GPU controller **tunes the SM partition adaptively at run time**, since the best SM partition varies with the kernel and the input shape. It structures data movement hierarchically so that data traversing the inter-node network is minimized. Finally, it drives the network from the GPU through a lightweight command queue and host proxy implemented directly on RDMA verbs, which allows **the same kernels to run on any network backend** (e.g. InfiniBand and on AWS EFA); we observe, surprisingly, that GPUDirect Async (IBGDA) yields little additional benefit over host-assisted GPU-initiated communication. We implement five kernels spanning tensor, sequence, and expert parallelism. On two 16-GPU H200 clusters, MKERNEL achieves speedups of up to $1.72\times$ on GEMM+AllReduce and $1.88\times$ on Ring Attention.

Code: <https://github.com/uccl-project/mKernel>

Blog: <https://uccl-project.github.io/posts/mkernel/>

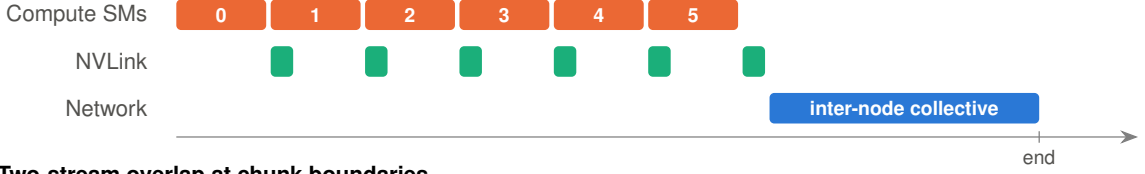
1 Introduction

Training and serving large models requires partitioning either model layers or input sequence across many GPUs, using tensor, sequence, and expert parallelism (Shoeybi et al., 2019; Liu et al., 2024; Lepikhin et al., 2021). In production mixture-of-experts (MoE) training, communication accounts for 43.6% of the forward pass and 32% of end-to-end training time (Jin et al., 2026), and across popular MoE models and frameworks, inter-device communication accounts for up to 47% of execution time (Zhang et al., 2025). The imbalance is widening because accelerator compute throughput is growing faster than network bandwidth: a GB300 NVL72 rack provides 720 PFLOP/s of FP8 compute (NVIDIA, 2025), yet each of its GPUs reaches devices outside the rack through a single NIC of 400–800 Gb/s.

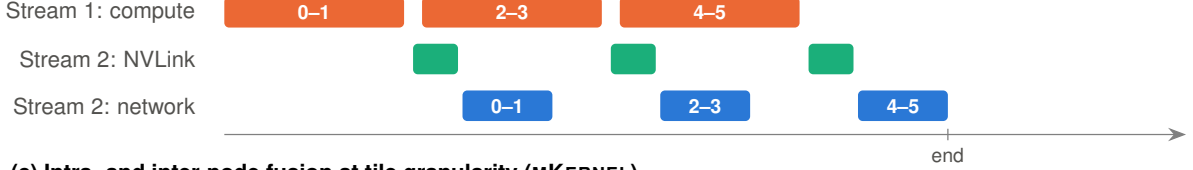
Prior work overlaps computation and communication at different granularities. Intra-node fusion can overlap computation with local transfers while leaving inter-node communication to a separate collective (Figure 1(a)). Two-stream pipelines communicate completed chunks while computing subsequent chunks, but release each chunk only at its kernel boundary (Figure 1(b)). Tile-level fusion enables finer overlap by exposing tile readiness within the kernel (Chang et al., 2024; Zhang et al., 2025; Sul et al., 2026). Most existing works focus on *single-node*; and many systems fuse only selected stages of computation, intra-node communication, and inter-node communication, leaving the remaining stages to execute separately. MKERNEL integrates computation, intra-node NVLink communication, and inter-node RDMA within a single fused kernel, coordinating tile-level progress across all three stages (Figure 1(c)).

Extending fusion across nodes is more difficult than fusing within a node. The inter-node network provides roughly one ninth of the per-GPU bandwidth of NVLink on our clusters, so the two tiers cannot be treated uniformly. The Network Interface Card (NIC) is a separate PCIe device with its own work queues, and GPUDirect Async requires NICs whose work queues the GPU can access directly. Transports also differ in their delivery guarantees: InfiniBand reliable connections deliver RDMA writes in order, whereas the Scalable

(a) Intra-node fusion, followed by an inter-node collective



(b) Two-stream overlap at chunk boundaries



(c) Intra- and inter-node fusion at tile granularity (MKERNEL)



Figure 1 Three schedules for computation and communication. (a) Intra-node fusion followed by a separate inter-node collective. (b) Two streams overlap communication of completed two-tile chunks with computation of subsequent chunks; each producer-kernel boundary releases a chunk. (c) mKERNEL schedules both communication tiers within the kernel, exposing tile-level readiness signal. We note that existing compute kernels already operate on tile granularity, so GEMM efficiency is not affected. Numbers identify output tiles; widths are schematic, not measured.

Reliable Datagram (SRD) transport of AWS EFA (Shalev et al., 2020) does not, so a completion flag written after the data may become visible before it. Both intra-node communication and inter-node communication require SMs, which compete with compute over shared resources.

This paper presents mKERNEL, a library of **inter-node fused kernels** in which computation, intra-node NVLink communication, and inter-node RDMA communication overlap **at tile granularity** (Figure 1(c)). Its design follows four principles. The first is **SM specialization**: each kernel assigns its thread blocks to compute, intra-node communication, inter-node send, and inter-node receive roles. The second is **hierarchical data movement**: data is locally reduced, or broadcasted over NVLink through NVSwitch, so that traffic traversing the inter-node network is minimized across pairs of GPUs across nodes. The third is **portable host-assisted GPU-initiated communication**: kernels enqueue compact transfer commands that a lightweight host proxy submits to the NIC as RDMA writes. This path is implemented directly on libibverbs, without NCCL or NVSHMEM, so the same kernels run on InfiniBand (ConnectX-7) and on AWS EFA. We also implemented GPUDirect Async (IBGDA) for ConnectX-7 and found little performance difference compared with host-assisted GPU-initiated communication. The fourth is **dynamic SM partitioning at runtime**: an on-GPU controller adjusts the compute-communication SM partition using measured progress and remaining work.

We implement five kernels with mKERNEL. For tensor parallelism we provide AllGather+GEMM, GEMM+ReduceScatter, and GEMM+AllReduce; for sequence parallelism, Ring Attention; and for expert parallelism, MoE Dispatch+GEMM. We evaluate them on two clusters of 2 nodes $\times$ 8 H200 GPUs, one cluster is interconnected with ConnectX-7 and the other cluster with EFA. Relative to cuBLAS or FlashAttention followed by NCCL, mKERNEL achieves speedups of up to $1.41\times$ on AllGather+GEMM, $1.72\times$ on GEMM+AllReduce, $1.88\times$ on Ring Attention. It also outperforms Triton-distributed, Flux, Mercury, MagiAttention, and ring-flash-attention in most configurations.

This paper makes three contributions. First, it analyzes the constraints that arise when fused kernels span multiple nodes (Section 2). Second, it presents the design of mKERNEL, including a host-assisted GPU-initiated communication layer that supports both InfiniBand and AWS EFA (Section 3), as well as one with IBGDA that runs on InfiniBand. Third, it describes five inter-node fused kernels (Section 4) and

Table 1 Communication hierarchy of two separate H200 clusters: one uses InfiniBand (IB), and the other uses AWS EFA. Both share the within-GPU and within-node hierarchy shown here. Crossing a node boundary changes which SMs coordinate data movement. NVLink and network bandwidths are per GPU, per direction.

Level	Link	Bandwidth	SM roles	Transfer mechanism
Within a GPU	HBM3e	4.8 TB/s	compute	loads/stores, TMA
Within a node	NVLink + NVSwitch	450 GB/s	local communication	TMA, NVSwitch
Across nodes (IB)	InfiniBand (CX7)	50 GB/s	send and receive	NIC RDMA
Across nodes (EFA)	AWS EFA (SRD)	50 GB/s	send and receive	NIC RDMA

evaluates them against seven baselines on two clusters (Section 5). MKERNEL is open source and available at <https://github.com/uccl-project/mKernel>.

2 Background and Motivation

2.1 Communication hierarchy of GPU clusters

We evaluate two separate clusters: one uses InfiniBand between nodes, and the other uses AWS EFA (Table 1). Both connect eight GPUs per node through NVLink and NVSwitch. The communication bandwidth across nodes is significantly lower within a node. ThunderKittens and ParallelKittens already provide the tile abstractions, asynchronous transfers, and intra-node communication primitives needed within a single node (Spector et al., 2025b; Sul et al., 2026). Across nodes, GPUDirect RDMA lets the NIC transfer payloads directly between GPU memories (NVIDIA, 2024a). The resulting kernel must allocate SMs to computation, local communication, and network coordination despite the different bandwidths of these paths.

2.2 Overlapping communication and computation in fused kernels

Tile-level fusion achieves fine-grained compute-communication overlap inside the kernel: communication can consume each output tile when it is produced, and computation can consume each input tile when it arrives. Flux integrates communication into GEMM epilogues and readiness checks into GEMM prologues, including support for inter-node writes through NVSHMEM (Chang et al., 2024). Comet and MegaScale-MoE apply fine-grained overlap to MoE workloads (Zhang et al., 2025; Jin et al., 2026). TileLink, Triton-distributed, and Mercury expose computation and communication through compiler primitives (Zheng et al., 2025b,a; Guan et al., 2025).

SM allocation in a fused kernel. Consider a GPU with S SMs executing a fused kernel that assigns S_c of them to communication. If compute throughput scales approximately with the number of compute SMs and the three stages overlap in steady state, execution time can be approximated as

$$T_{\text{fused}} \approx \max\left(T_{\text{compute}} \cdot \frac{S}{S - S_c}, T_{\text{NVLink}}, T_{\text{network}}\right) + T_{\text{fill/drain}} + T_{\text{sync}}. \quad (1)$$

Here, T_{compute} is compute time with all S SMs available; transfer times depend on the selected allocation. The final terms account for pipeline fill/drain and synchronization. **The central tradeoff is that additional communication SMs can reduce communication time while slowing computation.** We use this observation to motivate adaptive allocation of SMs and hierarchical communication to minimize T_{network} .

2.3 Challenges of designing efficient multi-GPU, multi-node fused kernels

Five challenges guide design in Section 3.

C1: Intra-node versus inter-node bandwidth asymmetry. The bandwidth gap between the NVLink domain and the inter-node network can make T_{network} the dominant term in equation (1). This motivates performing local aggregation and replication over NVLink, reducing repeated network transfers, and initiating those transfers early. In a rail-optimized topology (Wang et al., 2024) commonly deployed today, it is more

efficient to exchanges inter-node data between GPUs with *the same local index*, while using NVLink for intra-node data movement. We refer to each such remote GPU as a *rail peer*.

C2: Transfer-granularity asymmetry. Intra-node communication exposes peer-memory operations, whereas the network path uses explicit, message-oriented RDMA requests. Although RDMA addresses remote memory, each request specifies a buffer range and completion mechanism. In short, intra-node communication happens over *memory semantics*; inter-node communication happens over *message semantics*. Fused multi-node kernels must bridge these two interfaces: fine-grained local tiles should be grouped into network chunks to amortize per-message costs while exposing ready work early.

C3: Transport ordering semantics vary across platforms. An arrival flag must not become visible before its payload. InfiniBand reliable connections preserve the ordering of writes on a connection, whereas EFA’s Scalable Reliable Datagram transport or the recent OpenAI MRC protocol (Sohan et al., 2026) does not provide the same guarantee (Shalev et al., 2020; Amazon Web Services, 2026). A shared kernel interface must therefore accommodate different completion mechanisms.

C4: Cross-device synchronization. Network completion and GPU memory visibility can also introduce performance overhead (NVIDIA, 2024a). A persistent kernel needs readiness checks and memory-ordering guarantees that let it consume remote data safely without global synchronization after each transfer. The challenge is to provide these guarantees with limited polling and synchronization overhead.

C5: Adapting SM allocation to the workload. The SM resources needed for intra- and inter-node communication vary across input shapes, workload mixes, and kernel types. Static sweeps can identify effective partitions (Zhang et al., 2025; Sul et al., 2026), but repeated profiling and maintenance are needed as these choices change, especially as input shapes can be heterogenous. Our intra-node sweeps span best allocations of 2–64 communication SMs (Figure 3). Additionally, the balance (the best number of SMs for communication versus compute) also changes within a single kernel, such as when there is work imbalance between compute and communication or the work cannot be neatly divided over the available SMs. Adaptive SM partition tuning must respond without excessive overhead.

3 Design

Figure 2 shows how MKERNEL overlaps computation, NVLink transfers, and inter-node RDMA inside persistent kernels. The central design problem is to preserve tile-level progress across both intra- and inter-node communication boundary: a locally ready tile is not necessarily sent out as a complete network message, and a submitted message is not yet safe for remote computation to consume. MKERNEL separates these stages through explicit readiness handoffs. We use ThunderKittens’ compute primitives (Spector et al., 2025b); this section focuses on coordinating them with intra- and inter-node communication.

Thread blocks specialize in computation, intra-node communication, inter-node submission, or receive-side processing. Dedicated thread blocks submit ready work while compute blocks continue producing tiles. A host proxy forwards commands to the NIC; payloads remain in GPU memory. The following subsections cover hierarchical scheduling and transfer granularity (Section 3.1), adaptive SM allocation (Section 3.2), and inter-node batching and completion (Section 3.3).

3.1 Bridging intra- and inter-node communication granularity

Reduce or broadcast locally via NVSwitch before exchanging data across nodes. MKERNEL uses NVSwitch for local reduction and broadcast to minimize redundant inter-node traffic (C1). For example, on GEMM+AllReduce, each GPU in a node is responsible for one eighth of the output tiles. NVSwitch reduces the eight local GPUs’ contributions to each tile. The responsible GPU then exchanges this partial sum with its rail peer (GPUs with the same GPU index across nodes), combines the local and remote sums, and broadcasts the completed tile within its node. These steps proceed independently for each tile. For AllGather+GEMM,

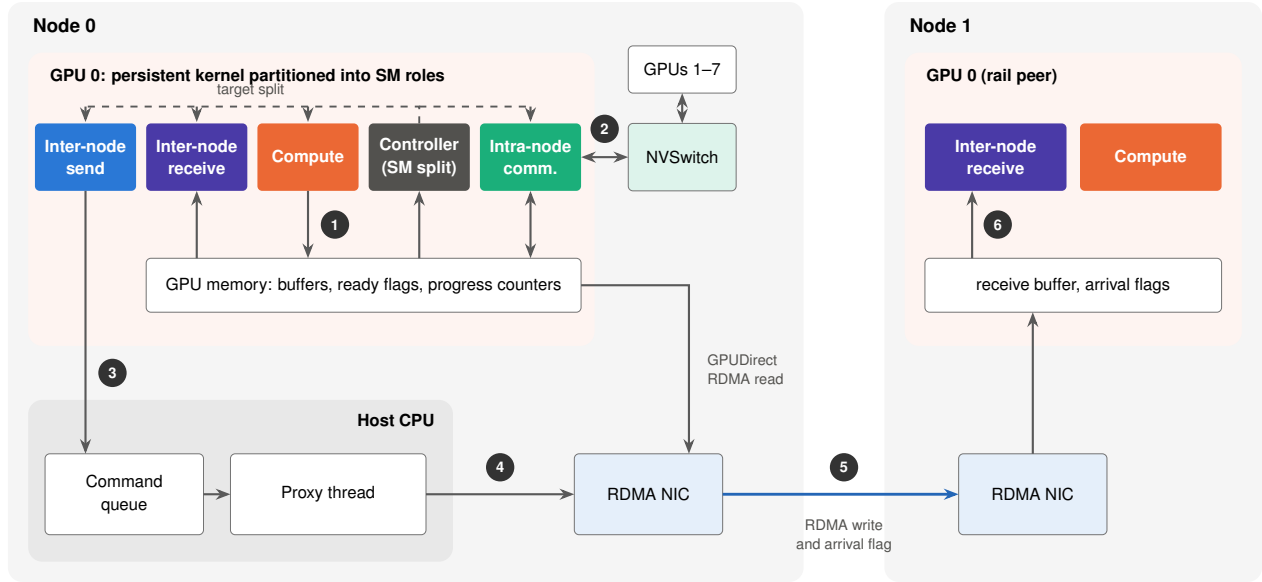


Figure 2 Architecture of mKERNEL (two nodes; one GPU per node shown in detail). Each GPU executes a persistent kernel whose thread blocks are assigned to roles. ① Compute blocks write completed tiles to GPU memory and set readiness flags. ② Intra-node communication blocks exchange tiles with the other seven GPUs over NVLink, using NVSwitch for broadcast and reduction. ③ Inter-node send blocks enqueue transfer commands in host memory. ④ A host proxy thread submits batches of commands to the NIC. ⑤ The NIC transfers data to the rail peer on node 1; the NIC or receiving proxy signals arrival, depending on the transport. ⑥ Receive blocks observe the flag and consume the data. The optional controller (Section 3.2) reads progress counters and publishes a target number of communication blocks (dashed).

each shard is sent once per destination node, where the receiving rail peer broadcasts it to the other local GPUs over NVLink.

Differentiate tile readiness and message readiness. Within a node, communication uses peer-memory operations on tiles; across nodes, RDMA requires explicit requests specifying buffer ranges and completion information. Mapping every local tile to a network request would tie local parallelism to per-message overhead (C2). mKERNEL instead sizes compute tiles, local transfers, and network chunks independently. Larger network chunks amortize submission and notification costs, but wait for more data and can lengthen pipeline fill and drain.

For example, GEMM+AllReduce groups four locally reduced tiles into a 256 KiB network chunk. Local thread blocks process tiles independently; a per-chunk counter lets the last completed tile publish readiness to the sender. A chunk can then be sent out (either with intra-node communication or inter-node communication) without waiting for the remaining GEMM computation to finish. The inverse mapping matters at the receiver. For example, Dispatch+GEMM checks every 512 KiB network chunk intersecting a token’s byte range before loading that token. A token spanning a chunk boundary requires both remote message arrivals. Since there is data dependency between computation and communication, in our case, either communication supplies input to computation, or vice versa.

Takeaway 1. Hierarchical communication reduces traffic on the slower inter-node network, while independently sizing local tiles and network chunks balances early transmission against per-message overhead. mKERNEL uses NVSwitch for local aggregation and replication and sends each RDMA chunk as soon as its constituent tiles are ready.

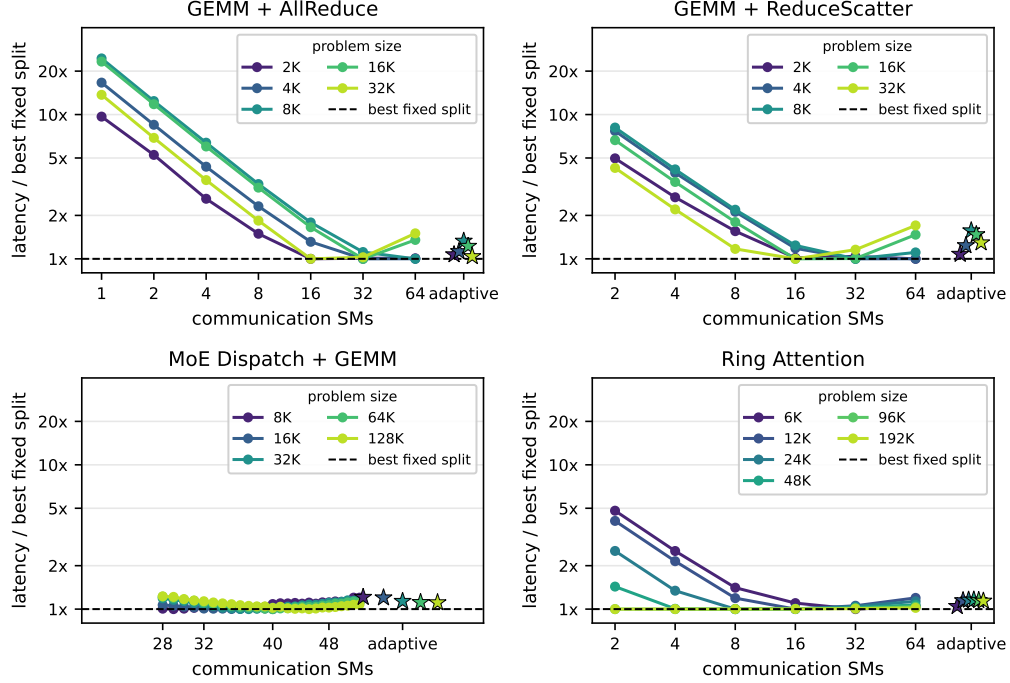


Figure 3 Static sweep of the SM partition (single node, 8 H100 GPUs). Each curve gives the latency of one problem size as a function of the number of communication SMs, normalized to the best fixed SM partition for that size (dashed line). The star markers at the right of each panel give the latency of the adaptive controller for the same sizes.

3.2 SM partitioning and adaptive tuning

The preceding mechanisms determine when work is ready; the SM partitioning determines how quickly each stage can progress. Thread blocks are assigned roles by block index in the fixed configuration and may change roles at task boundaries in adaptive mode. Separate SM roles let mKERNEL change the allocation without changing the compute block’s warp layout, building on the scheduling choices.

Adaptive tuning of the SM partition. Static sweeps identify efficient SM partitions, but require choices to be profiled and maintained for each workload (C5). mKERNEL provides an adaptive mode that updates the allocation within an execution. Each block checks a shared *target* number of communication blocks between compute or communication tasks (*e.g.*, a tile or a message) and changes roles when the active allocation differs from this target. Figure 3 compares this policy with static sweeps for four intra-node kernels.

The controller estimates the per-task cost of each role from cycle counters accumulated by the thread blocks and sets the target to the value that equalizes the predicted completion times of the two roles,

$$n_s^* = B \cdot \frac{R_s C_s}{R_p C_p + R_s C_s}, \quad (2)$$

where R_p and R_s are the remaining compute and communication tasks, and C_p and C_s are their estimated costs.

The data dependency determines the initial allocation and how stalled work is treated. When communication supplies inputs to computation, a compute block waiting for input may temporarily assist communication even if the target has been reached. We evaluate adaptive tuning on intra-node kernels in Section 5.5.

Takeaway 2. SM specialization separates role implementation from resource allocation. mKERNEL can rebalance compute, intra-node communication, and inter-node communication by adjusting their SM budgets without redesigning the compute block’s warp layout.

Table 2 Inter-node notification paths (EFA’s completion-based variant shown). Transport completion and GPU memory visibility are separate requirements.

	ConnectX-7 (InfiniBand)	AWS EFA (SRD)
Transport order	ordered per RC connection	unordered
Notification path	data write, then flag write	write with immediate; proxy sets flag

3.3 Portable inter-node communication in fused kernels

The host-assisted GPU-initiated communication path translates GPU-produced work into explicit RDMA messages. A proxy implemented on `libibverbs` handles submission and transport-specific notification, following the GPU-command/CPU-proxy design in UEP (Mao et al., 2025).

Command publication and backpressure. Send blocks publish 48-byte commands in a ring buffer in pinned host memory. Each command specifies the peer, source and destination offsets, byte count, and chunk identifier. The GPU writes the command body before committing its header; the proxy polls this header before reading the record. Queue credits and a limit on outstanding requests bound the work submitted to the host and NIC. The NIC reads the payload directly from registered GPU memory, including staging buffers when the network layout differs from the compute layout.

Batch commands while preserving per-chunk completion. The proxy groups up to eight ready commands for one connection into a single submission. This is distinct from constructing a larger network chunk: each command retains its own data transfer and arrival notification. A fixed requirement to fill every batch would delay sparse arrivals and the final few chunks. Instead, the proxy submits partial batches, for example, using a bounded polling window to collect additional transfer commands. Submission batching thus follows the rate at which the kernel produces ready work, while chunk size controls when that work first becomes transferable.

Transport delivery semantics vary across platforms. An arrival signal must identify a completed payload, not merely a posted request (C3). InfiniBand and EFA require different notification paths (Table 2):

- *InfiniBand (ConnectX-7).* The proxy posts a data write followed by a small flag write on the same reliable connection (RC). Receive blocks poll the flag in GPU memory, without a receiving host proxy. This protocol requires data-before-flag ordering at the destination.
- *AWS EFA (SRD).* SRD does not order independent writes (Amazon Web Services, 2026). The completion-based path therefore uses RDMA write with immediate data: the immediate value identifies the chunk, and the receiving proxy publishes its arrival flag after the write’s receive completion. This avoids using the arrival order of a separate flag write to infer payload completion.

Host-assisted GPU-initiated communication versus GPUDirect Async communication. We also implement GPUDirect Async on ConnectX-7 using IBGDA (Markthub et al., 2022), exposing the same device interface as host-assisted GPU-initiated communication. Figure 4 compares the two backends. In the scenarios we tested, GPUDirect Async provides little performance gain over host-assisted GPU-initiated communication.

Host-assisted GPU-initiated communication lets the proxy overlap submission with computation and batch multiple commands, while the GPUDirect Async backend needs to issue an *expensive* system-scoped GPU fence and doorbell per chunk. GPUDirect Async (IBGDA) can also be used in small-message, latency-sensitive workloads (Markthub et al., 2022; Zhao et al., 2025b).

Takeaway 3. Local tiles, network chunks, and submission batches need not share one granularity. A batched host proxy connects these stages while keeping payload transfers on the NIC. GPUDirect Async (IBGDA) provides no consistent throughput advantage over host-assisted GPU-initiated communication in our comparison.

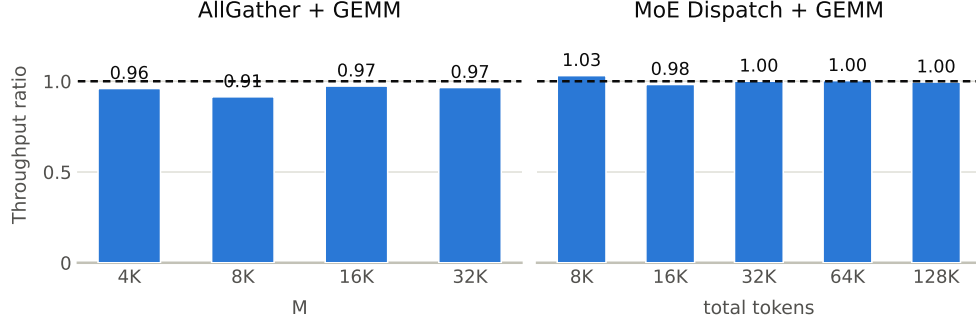


Figure 4 GPUDirect Async (IBGDA) relative to host-assisted GPU-initiated communication on the ConnectX-7 testbed of Section 5.1 (2 nodes $\times$ 8 H200), with identical kernels. Bars give the throughput ratio, with GPUDirect Async in the numerator; the dashed line marks parity.

Table 3 Fused kernels implemented in MKERNEL. TP, SP, and EP denote tensor, sequence, and expert parallelism.

Kernel	Data dependency	Intra-node	Inter-node	Chunk
AllGather + GEMM (TP)	comm. $\rightarrow$ compute	multicast broadcast of each shard	shard sent once to each node; ring forwarding beyond two nodes	128 rows
GEMM + ReduceScatter (TP)	compute $\rightarrow$ comm.	TMA atomic add into the owner’s buffer	exchange of per-node partial sums	2–32 tiles
GEMM + AllReduce (TP)	compute $\rightarrow$ comm.	NVSwitch reduction; multicast broadcast of the result	exchange of 1/8 of the output per GPU	4 tiles (256 KiB)
MoE Dispatch + GEMM (EP)	comm. $\rightarrow$ compute	TMA pull of tokens from peers	token buffer copied to the rail peer	16 tokens; 512 KB
Ring Attention (SP)	independent within a step	TMA store of KV to the next GPU	KV slice sent once to each node	128-row KV tiles

4 Implementation

Table 3 summarizes the five kernels implemented in MKERNEL. They use the mechanisms in Section 3: persistent scheduling with explicit SM roles, hierarchical communication through rail peers, and epoch-stamped readiness flags. MKERNEL makes three scheduling choices: First, inter-node transfers are initiated early: AllGather+GEMM and Ring Attention submit their input shard or KV slice to rail peers at launch. Second, NVSwitch handles local aggregation and replication. For example, GEMM+AllReduce reduces data within NVSwitch before each GPU sends one eighth of the output. Third, compute tiles are ordered according to input availability: AllGather+GEMM processes its local shard, the remaining shards within the node, and then remote shards.

5 Evaluation

Our evaluation addresses four questions:

1. How do the fused kernels perform relative to unfused execution such as using cuBLAS or FlashAttention and NCCL (Sections 5.2 to 5.4)?
2. How does MKERNEL compare with existing fused kernels (Sections 5.2 to 5.4)?
3. How sensitive is performance to the compute–communication SM partition, and how close does the adaptive controller come to the best fixed SM partition (Section 5.5)?

Table 4 Testbeds. Both contain $2 \text{ nodes} \times 8 \text{ H200 GPUs}$ connected by NVLink/NVSwitch within each node, with different inter-node networks.

Testbed	GPUs	Inter-node network	NICs per node
ConnectX-7	$2 \times 8 \text{ H200}$	InfiniBand	$8 \times 400 \text{ Gb/s}$ ConnectX-7
AWS EFA	$2 \times 8 \text{ H200}$	AWS SRD	$16 \times 200 \text{ Gb/s}$ EFA

5.1 Experimental setup

Testbeds. Table 4 describes the two clusters, both of which provide 400 Gb/s of network bandwidth per GPU. MKERNEL is compiled for Hopper (`sm_90a`) with CUDA 12.9, and one process runs per GPU.

Baselines. *unfused* execution: a cuBLAS GEMM or FlashAttention (Dao, 2024) kernel preceded or followed by the corresponding NCCL collective (all-gather, reduce-scatter, all-reduce, or all-to-all). We also compare against the following fused/comm-comp overlapped kernels/systems:

- Triton-distributed (Zheng et al., 2025a) and FLUX (Chang et al., 2024), which fuse GEMMs with communication;
- Mercury (Guan et al., 2025), a multi-GPU kernel compiler;
- DeepEP (Zhao et al., 2025b) combined with DeepGEMM (Zhao et al., 2025a), for MoE dispatch;
- MagiAttention (Tao and Huang, 2025) and ring-flash-attention (Zhu, 2024), for distributed attention.

Each baseline is evaluated on the kernels and testbeds supported by the implementation used in the experiments. The evaluated DeepEP implementation uses GPUDirect Async (IBGDA) on InfiniBand and does not support EFA. The EFA dispatch plot therefore includes its ConnectX-7 measurements as a cross-testbed reference.

Measurements follow a warm-up phase and use the slowest rank’s execution time, which determines when the next layer can begin. Kernels are timed over consecutive launches with one synchronization at the end.

Figures 5 and 6 present the results on EFA and ConnectX-7, respectively.

5.2 Tensor parallelism

AllGather + GEMM. MKERNEL outperforms cuBLAS+NCCL at every evaluated problem size on both testbeds, with speedups of up to $1.41\times$ on EFA and $1.34\times$ on ConnectX-7. The schedule initiates inter-node transfers first and transfers each remote shard once per destination node, allowing communication to overlap with GEMM.

GEMM + AllReduce. GEMM+AllReduce has the largest peak speedup among the evaluated tensor-parallel kernels. MKERNEL achieves up to $1.53\times$ on EFA and $1.72\times$ on ConnectX-7. The gain is due to MKERNEL’s hierarchical schedule: intra-node reductions are performed in NVSwitch; each GPU sends only $1/8$ of the output across the network; all three phases proceed concurrently with the GEMM.

GEMM + ReduceScatter. MKERNEL is faster for large problems ($1.02\text{--}1.27\times$ for $M \geq 16\text{K}$) and outperforms Triton-distributed at most input size.

5.3 Expert parallelism

MoE Dispatch + GEMM. On EFA, MKERNEL achieves speedups of $3.1\text{--}4.5\times$ over an all-to-all followed by a GEMM. This baseline assumes uniform routing and does not group expert GEMMs, so the comparison includes benefits from expert computation organization as well as communication overlap. Our kernel also outperforms DeepEP+DeepGEMM (EFA); where the DeepEP+DeepGEMM bars are measurements on ConnectX-7, due to testbed limitations.

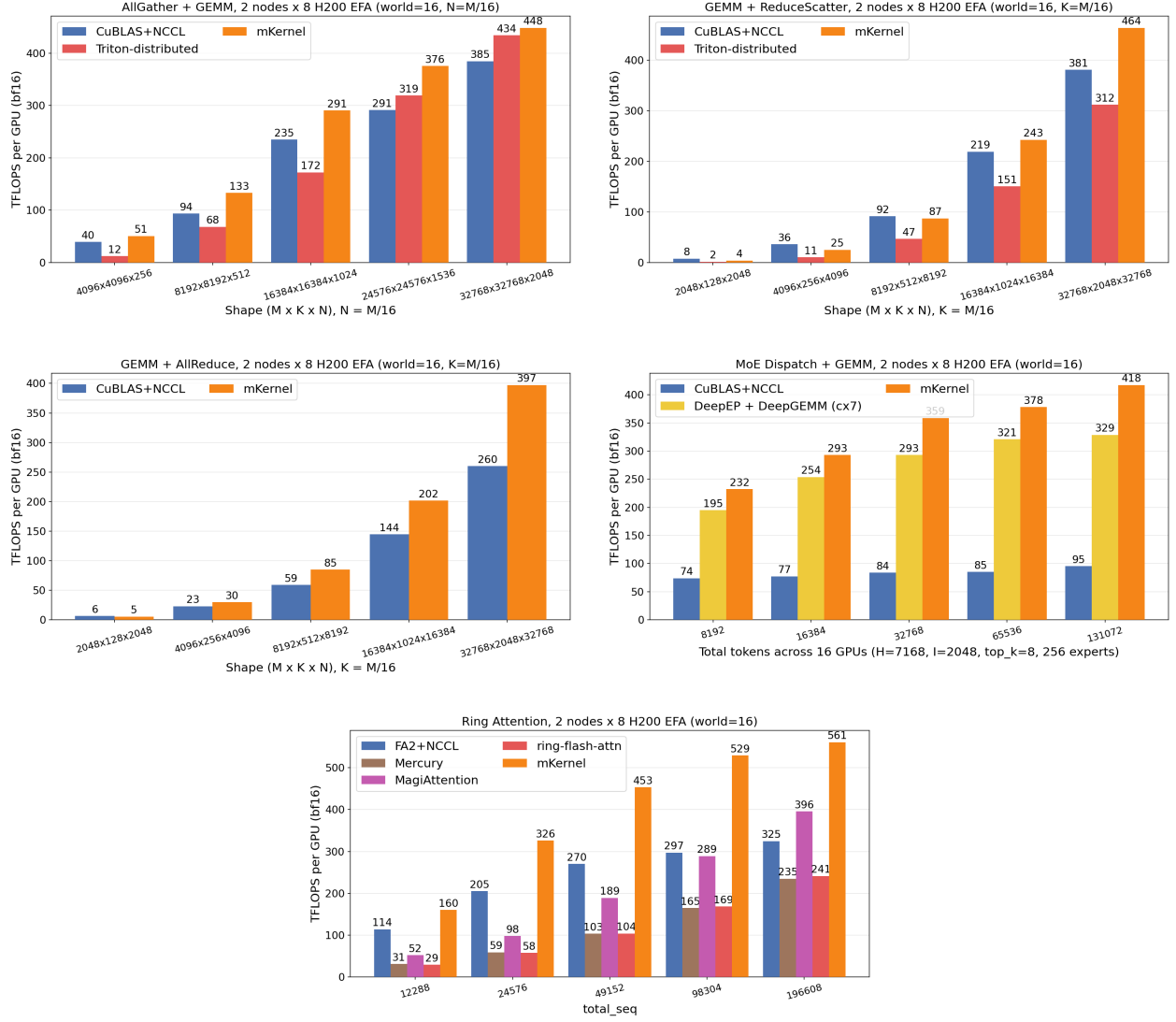


Figure 5 Throughput on the AWS EFA testbed (2 nodes $\times$ 8 H200; TFLOPS per GPU, higher is better). mKERNEL is shown in orange. The evaluated DeepEP+DeepGEMM implementation does not support EFA; its bars show ConnectX-7 measurements for reference.

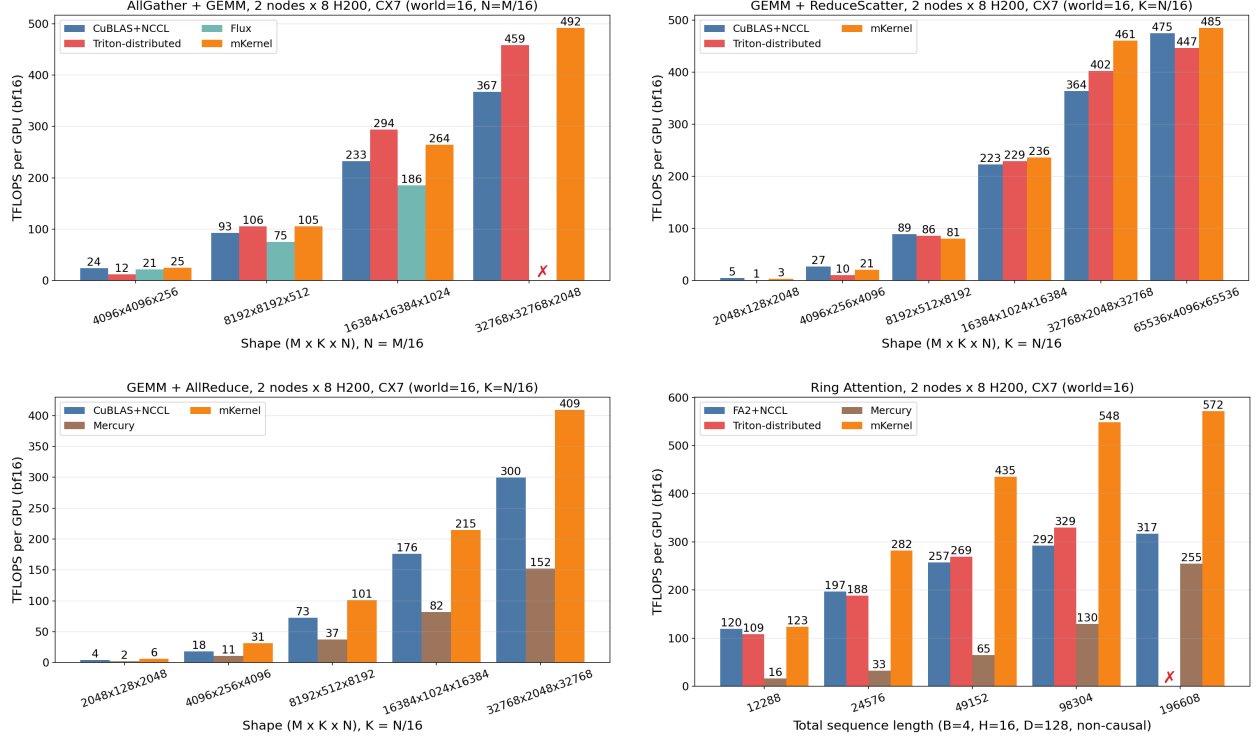


Figure 6 Throughput on the ConnectX-7 InfiniBand testbed (2 nodes $\times$ 8 H200; TFLOPS per GPU). A red $\times$ indicates that a baseline failed at that problem size.

5.4 Sequence parallelism

mKERNEL’s Ring Attention kernel achieves speedups of up to $1.78\times$ on EFA and $1.88\times$ on ConnectX-7 over the unfused baseline. Against the evaluated distributed-attention implementations, mKERNEL achieves the following speedups:

- $1.4\text{--}3.3\times$ over MagiAttention;
- $2.3\text{--}5.6\times$ over ring-flash-attention;
- $2.2\text{--}8.6\times$ over Mercury;
- $1.1\text{--}1.7\times$ over Triton-distributed, whose evaluated implementation fails on the longest sequence.

The advantage is generally larger for shorter sequences, where communication is harder to amortize over attention computation.

5.5 Adaptive SM partitioning

A static sweep measures sensitivity to the SM partition and identifies the best measured configuration for each problem size. We then compare the adaptive controller in [Section 3.2](#) with those configurations. Due to testbed limitations, we are only able to test adaptive SM partitioning on a single node.

Static sweep. We sweep communication allocations from 1 or 2 to 64 SMs in powers of two. [Figure 3](#) normalizes each curve to its minimum measured latency. The preferred allocation ranges from 2 to 64 SMs across workloads. Too few communication SMs can stall computation or leave a large communication backlog: the worst measured AllReduce SM partition is approximately $25\times$ slower than the best. Excess communication allocation reduces compute resources. Static sweeps recover good configurations, but the changing minima show why a single choice does not transfer across kernels and sizes.

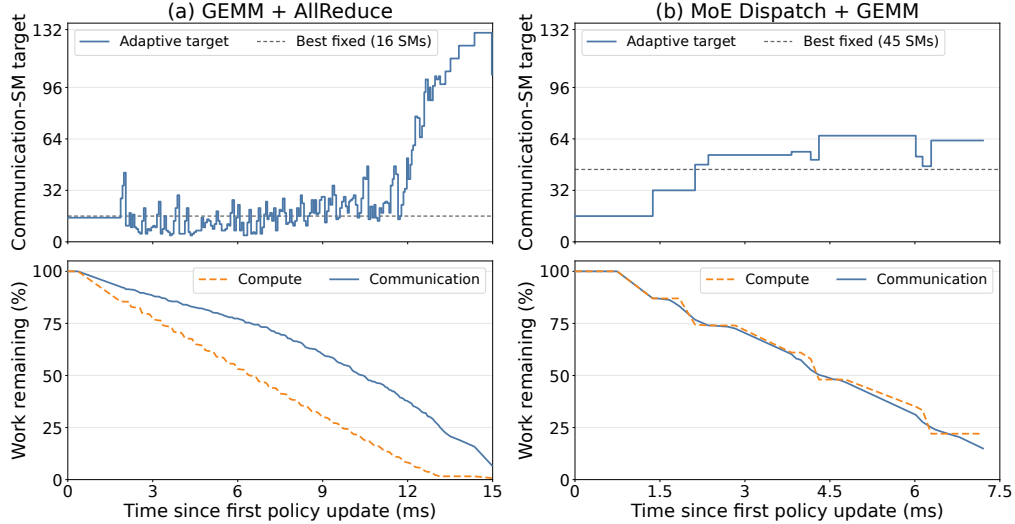


Figure 7 Adaptive SM partition tuning. (a) GEMM+AllReduce ($M = N = 32K$, $K = 4K$) shifts its target toward communication as computation nears completion. (b) MoE Dispatch+GEMM (128K total tokens) increases the dispatch-SM target while computation and communication make similar relative progress. Gray dashed lines mark the best fixed partitions (16 and 45 SMs respectively).

Adaptive controller. The star markers in Figure 3 show the controller’s latency for each size. Without per-shape tuning, it achieves a geometric mean of $1.18\times$ across 21 configurations compared to the static best fixed SM partition.

Intra-kernel SM partition adaptation Figure 7 illustrates how the allocation target responds to remaining work for GEMM+AllReduce and MoE Dispatch+GEMM. For GEMM+AllReduce, since compute tasks finish early, all SMs are dedicated to communication towards the end to finish the remaining work..

6 Related Work

Kernel frameworks and distributed fusion. CUTLASS (Thakkar et al., 2023), Triton (Tillet et al., 2019), and ThunderKittens (Spector et al., 2025b) provide tiled GPU computation abstractions; MKERNEL uses ThunderKittens for its compute blocks. ParallelKittens (Sul et al., 2026) extends these abstractions with intra-node communication and synchronization. FLUX (Chang et al., 2024) fuses communication and readiness checks into GEMM, including inter-node writes through NVSHMEM. TileLink (Zheng et al., 2025b), Triton-distributed (Zheng et al., 2025a), and Mercury (Guan et al., 2025) expose computation and communication to compiler optimization. MKERNEL uses explicit schedules within persistent kernels to coordinate NVLink transfers, RDMA chunks, and SM roles.

Overlap scheduling and SM allocation. Megatron-LM (Narayanan et al., 2021) and Transformer Engine (NVIDIA, 2024c) overlap communication with computation across operations. Work decomposition (Wang et al., 2023) and Centauri (Chen et al., 2024) expose smaller scheduling units for dependent operations. CoCoNet (Jangda et al., 2022) supports compiler transformations for fusion and overlap, while T3 (Pati et al., 2024) uses hardware tracking and triggering. NanoFlow (Zhu et al., 2025) jointly selects batch decomposition and GPU resource allocation; COMET (Zhang et al., 2025) selects profiled compute–communication configurations using workload metadata. MKERNEL specifically targets inter-node fused kernels and updates its target SM partition during kernel execution using measured progress and remaining work (Section 3.2).

Communication interfaces and topology. NVSHMEM (NVIDIA, 2024b) supports device-side one-sided operations through GPUDirect Async (IBGDA) (Markthub et al., 2022) or host-assisted GPU-initiated communication; its libfabric backend supports EFA (NVIDIA, 2026b). MSCCL++ (Hwang et al., 2026)

offers peer-memory access and proxy-mediated networking; NCCL’s device API and GIN (NVIDIA, 2026a; Hamidouche et al., 2025) support device communication with direct and proxy network backends. UCCL-Tran (Zhou et al., 2026) places transport control on CPUs, and UEP (Mao et al., 2025) uses GPU-issued commands and host proxies for portable expert-parallel communication. MKERNEL integrates a compact command queue and direct verbs implementation with fused schedules on InfiniBand and EFA. TACCL (Shah et al., 2023) synthesizes topology-aware collectives; MKERNEL uses hierarchical transfer schedules constrained by tile readiness.

Persistent kernels and megakernels. The Llama megakernel (Spector et al., 2025a) and MPK (Cheng et al., 2026) use persistent execution to schedule model operations, with MPK supporting multi-GPU inference. MKERNEL focuses on computation and communication within each distributed kernel.

7 Conclusion

MKERNEL is a library of **multi-GPU, multi-node fused kernels that overlap computation and communication at tile granularity**. MKERNEL adopts SM specialization, hierarchical data movement, and portable host-assisted GPU-initiated communication, which enables the same kernels to coordinate NVLink and inter-node RDMA on InfiniBand and EFA. We implement five kernels for tensor, sequence, and expert parallelism. On two 16-GPU H200 clusters, MKERNEL achieves speedups of up to $1.72\times$ on GEMM+AllReduce and $1.88\times$ on Ring Attention over the unfused baselines. The intra-node adaptive mode complements this design by dynamically tuning compute and communication SM allocations, reducing overhead of tuning kernels for each kernel type and input shapes.

Acknowledgement

We would also like to thank Chon Lam Lao, Xingyu Xiang for helpful discussions. This work is in part supported by gifts from Accenture, AMD, Anyscale, Broadcom, Cisco, Google, IBM, Intel, Intesa Sanpaolo, Lambda, Lightspeed, Mibura, Microsoft, NVIDIA, Samsung SDS, and SAP. Costin Raiciu was partly funded by HRIA (project no. 351416). Yihan Zhang, Shuang Ma, and Yang Zhou are supported by NSF Grant 2552193.

References

- Amazon Web Services. Elastic Fabric Adapter (EFA) linux driver. <https://github.com/amzn/amzn-drivers/blob/master/kernel/linux/efa/README>, 2026. Documentation accessed September 10, 2026.
- Li-Wen Chang, Wenlei Bao, Qi Hou, Chengquan Jiang, Ningxin Zheng, Yinmin Zhong, Xuanrun Zhang, Zuquan Song, Chengji Yao, Ziheng Jiang, et al. FLUX: Fast software-based communication overlap on GPUs through kernel fusion. *arXiv preprint arXiv:2406.06858*, 2024. <https://arxiv.org/abs/2406.06858>.
- Chang Chen, Xiuhong Li, Qianchao Zhu, Jiangfei Duan, Peng Sun, Xingcheng Zhang, and Chao 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 (ASPLOS)*, pages 178–191, 2024.
- Xinhao Cheng, Zhihao Zhang, Yu Zhou, Jianan Ji, Jinchun Jiang, Zepeng Zhao, Ziruo Xiao, Zihao Ye, Yingyi Huang, Ruihang Lai, et al. MPK: A compiler and runtime for mega-kernelizing tensor programs. In *20th USENIX Symposium on Operating Systems Design and Implementation (OSDI 26)*, 2026. arXiv:2512.22219.
- Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations (ICLR)*, 2024. arXiv:2307.08691.
- Yue Guan, Xinwei Qiang, Zaifeng Pan, Daniels Johnson, Yuanwei Fang, Keren Zhou, Yuke Wang, Wanlu Li, Yufei Ding, and Adnan Aziz. Mercury: Unlocking multi-GPU operator optimization for LLMs via remote memory scheduling. In *Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles (SOSP)*, pages 1046–1061, 2025. doi: 10.1145/3731569.3764798.

- Khaled Hamidouche, John Bachan, Pak Markthub, Peter-Jan Gootzen, Elena Agostini, Sylvain Jeaugey, Aamir Shafi, Georgios Theodorakis, and Manjunath Gorentla Venkata. GPU-initiated networking for NCCL. *arXiv preprint arXiv:2511.15076*, 2025. <https://arxiv.org/abs/2511.15076>.
- Changho Hwang, Peng Cheng, Roshan Dathathri, Abhinav Jangda, Saeed Maleki, Madan Musuvathi, Olli Saarikivi, Aashaka Shah, Ziyue Yang, Binyang Li, Caio Rocha, Qinghua Zhou, Mahdieh Ghazimirsaeed, Sreevatsa Anantharamu, and Jithin Jose. MSCCL++: Rethinking GPU communication abstractions for AI inference. In *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2026. <https://www.microsoft.com/en-us/research/publication/msccl-rethinking-gpu-communication-abstractions-for-ai-inference/>.
- Abhinav Jangda, Jun Huang, Guodong Liu, Amir Hossein Nodehi Sabet, Saeed Maleki, Youshan Miao, Madanlal Musuvathi, Todd Mytkowicz, and Olli 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 (ASPLOS)*, pages 402–416, 2022. <https://arxiv.org/abs/2105.05720>.
- Chao Jin, Ziheng Jiang, Zhihao Bai, Zheng Zhong, Juncai Liu, Xiang Li, Ningxin Zheng, Xi Wang, Cong Xie, Qi Huang, et al. MegaScale-MoE: Large-scale communication-efficient training of mixture-of-experts models in production. In *Proceedings of the 21st European Conference on Computer Systems (EuroSys)*, pages 366–382, 2026.
- Dmitry Lepikhin, HyounJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. GShard: Scaling giant models with conditional computation and automatic sharding. In *International Conference on Learning Representations (ICLR)*, 2021.
- Hao Liu, Matei Zaharia, and Pieter Abbeel. RingAttention with blockwise transformers for near-infinite context. In *International Conference on Learning Representations (ICLR)*, 2024. arXiv:2310.01889.
- Ziming Mao, Yihan Zhang, Chihan Cui, Zhen Huang, Kaichao You, Zhongjie Chen, Zhiying Xu, Zhenyu Gu, Scott Shenker, Costin Raiciu, Yang Zhou, and Ion Stoica. UCCL-EP: Portable expert-parallel communication. *arXiv preprint arXiv:2512.19849*, 2025. <https://arxiv.org/abs/2512.19849>.
- Pak Markthub, Jim Dinan, Sreeram Potluri, and Seth Howell. Improving network performance of HPC systems using NVIDIA Magnum IO NVSHMEM and GPUDirect Async. <https://developer.nvidia.com/blog/improving-network-performance-of-hpc-systems-using-nvidia-magnum-io-nvshmem-and-gpudirect-async/>, 2022. NVIDIA Technical Blog.
- 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 (SC)*, pages 1–15, 2021.
- NVIDIA. GPUDirect RDMA. <https://docs.nvidia.com/cuda/gpudirect-rdma/>, 2024a.
- NVIDIA. NVSHMEM. <https://developer.nvidia.com/nvshmem>, 2024b.
- NVIDIA. Transformer engine. <https://github.com/NVIDIA/TransformerEngine>, 2024c.
- NVIDIA. NVIDIA GB300 NVL72. <https://www.nvidia.com/en-us/data-center/gb300-nvl72/>, 2025.
- NVIDIA. NCCL device-initiated communication. <https://docs.nvidia.com/deeplearning/nccl/user-guide/docs/usage/deviceapi.html>, 2026a. Documentation accessed September 10, 2026.
- NVIDIA. NVSHMEM environment variables: Remote transports and Libfabric providers. <https://docs.nvidia.com/nvshmem/api/latest/gen/env.html>, 2026b. Documentation accessed September 10, 2026.
- Suchita Pati, Shaizeen Aga, Mahzabeen Islam, Nuwan Jayasena, and Matthew 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 (ASPLOS)*, pages 1146–1164, 2024.
- Aashaka Shah, Vijay Chidambaram, Meghan Cowan, Saeed Maleki, Madan Musuvathi, Todd Mytkowicz, Jacob Nelson, Olli Saarikivi, and Rachee Singh. TACCL: Guiding collective algorithm synthesis using communication sketches. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 593–612, 2023. <https://www.usenix.org/conference/nsdi23/presentation/shah>.

- Leah Shalev, Hani Ayoub, Nafea Bshara, and Erez Sabbag. A cloud-optimized transport protocol for elastic and scalable HPC. *IEEE Micro*, 40(6):67–73, 2020.
- 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.
- Rip Sohan, Eric Spada, Eric Davis, et al. The Multipath Reliable Connection (MRC) transport. *arXiv preprint arXiv:2606.18170*, 2026. <https://arxiv.org/abs/2606.18170>.
- Benjamin Spector, Jordan Juravsky, Stuart Sul, Owen Dugan, Dylan Lim, Dan Fu, Simran Arora, and Christopher Ré. Look ma, no bubbles! designing a low-latency megakernel for Llama-1B. <https://hazyresearch.stanford.edu/blog/2025-05-27-no-bubbles>, 2025a.
- Benjamin F. Spector, Simran Arora, Aaryan Singhal, Arjun Parthasarathy, Daniel Y. Fu, and Christopher Ré. ThunderKittens: Simple, fast, and Adorable kernels. In *International Conference on Learning Representations (ICLR)*, 2025b. arXiv:2410.20399.
- Stuart H. Sul, Simran Arora, Benjamin F. Spector, and Christopher Ré. ParallelKittens: Systematic and practical simplification of multi-GPU AI kernels. In *Proceedings of Machine Learning and Systems (MLSys)*, 2026. arXiv:2511.13940.
- Zewei Tao and Yunpeng Huang. MagiAttention: A distributed attention towards linear scalability for ultra-long context, heterogeneous mask training. <https://github.com/SandAI-org/MagiAttention>, 2025.
- Vijay Thakkar, Pradeep Ramani, Cris Cecka, Aniket Shivam, Honghao Lu, Ethan Yan, Jack Kosaian, Mark Hoemmen, Haicheng Wu, Andrew Kerr, et al. CUTLASS. <https://github.com/NVIDIA/cutlass>, 2023.
- Philippe Tillet, Hsiang-Tsung Kung, and David Cox. Triton: An intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pages 10–19, 2019.
- Shibo Wang, Jinliang Wei, Amit Sabne, Andy Davis, Berkin Ilbeyi, Blake Hechtman, Dehao Chen, Karthik Srinivasa Murthy, Marcello Maggioni, Qiao 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 (ASPLOS)*, pages 93–106, 2023.
- 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. In *IEEE Symposium on High-Performance Interconnects (HOTI)*, 2024. https://people.csail.mit.edu/ghobadi/papers/rail_only_hoti_2024.pdf.
- Shulai Zhang, Ningxin Zheng, Haibin Lin, Ziheng Jiang, Wenlei Bao, Chengquan Jiang, Qi Hou, Weihao Cui, Size Zheng, Li-Wen Chang, Quan Chen, and Xin Liu. COMET: Fine-grained computation-communication overlapping for mixture-of-experts. In *Proceedings of Machine Learning and Systems (MLSys)*, 2025. https://proceedings.mlsys.org/paper_files/paper/2025/hash/e27ea0cd50b798ff8942caf9203f0992-Abstract-Conference.html.
- Chenggang Zhao, Zhean Xu, Liang Zhao, Jiashi Li, Chenhao Xu, Anyi Xu, Shengyu Liu, Kexing Zhou, and Kuai Yu. DeepGEMM: Clean and efficient BLAS kernel library on GPU. <https://github.com/deepseek-ai/DeepGEMM>, 2025a.
- Chenggang Zhao, Shangyan Zhou, Liyue Zhang, Chengqi Deng, Zhean Xu, Yuxuan Liu, Kuai Yu, Jiashi Li, and Liang Zhao. DeepEP: An efficient expert-parallel communication library. <https://github.com/deepseek-ai/DeepEP>, 2025b.
- Size Zheng, Wenlei Bao, Qi Hou, Xuegui Zheng, Jin Fang, Chenhui Huang, Tianqi Li, Haojie Duanmu, Renze Chen, Ruifan Xu, et al. Triton-distributed: Programming overlapping kernels on distributed AI systems with the Triton compiler. *arXiv preprint arXiv:2504.19442*, 2025a. <https://arxiv.org/abs/2504.19442>.
- Size Zheng, Jin Fang, Xuegui Zheng, Qi Hou, Wenlei Bao, Ningxin Zheng, Ziheng Jiang, Dongyang Wang, Jianxi Ye, Haibin Lin, Li-Wen Chang, and Xin Liu. TileLink: Generating efficient compute-communication overlapping kernels using tile-centric primitives. In *Proceedings of Machine Learning and Systems (MLSys)*, 2025b. https://proceedings.mlsys.org/paper_files/paper/2025/hash/c6ee784cbe46d854843e4c883a3321ef-Abstract-Conference.html.
- Yang Zhou, Zhongjie Chen, Ziming Mao, ChonLam Lao, Shuo Yang, Pravein Govindan Kannan, Xizhi Zhang, Jiaqi Gao, Yilong Zhao, Yongji Wu, Kaichao You, Fengyuan Ren, Zhiying Xu, Costin Raiciu, and Ion Stoica. UCCL-Tran: An extensible software transport layer for GPU networking. In *20th USENIX Symposium on Operating Systems Design*

and Implementation (OSDI 26), pages 1143–1166, 2026. <https://www.usenix.org/conference/osdi26/presentation/zhou-yang>. arXiv:2504.17307.

Kan Zhu, Yufei Gao, Yilong Zhao, Liangyu Zhao, Gefei Zuo, Yile Gu, Dedong Xie, Tian Tang, Qinyu Xu, Zihao Ye, Keisuke Kamahori, Chien-Yu Lin, Ziren Wang, Stephanie Wang, Arvind Krishnamurthy, and Baris Kasikci. NanoFlow: Towards optimal large language model serving throughput. In *19th USENIX Symposium on Operating Systems Design and Implementation (OSDI 25)*, pages 749–765, 2025. <https://www.usenix.org/conference/osdi25/presentation/zhu-kan>.

Zilin Zhu. ring-flash-attention: Ring attention implementation with flash attention. <https://github.com/zhuzilin/ring-flash-attention>, 2024.