

FiCCO: Design Space Exploration of DMA-based Finer-Grain Compute Communication Overlap

Shagnik Pal

The University of Texas at Austin
shagnik@utexas.edu

Shaizeen Aga

Advanced Micro Devices, Inc.
shaizeen.aga@amd.com

Suchita Pati

Advanced Micro Devices, Inc.
suchita.pati@amd.com

Mahzabeen Islam

Advanced Micro Devices, Inc.
mahzabeen.islam@amd.com

Lizy K. John

The University of Texas at Austin
ljohn@ece.utexas.edu

Abstract—Modern ML workloads demand distributing training and inference across multiple GPUs. However, these parallelization techniques often suffer from exposed critical-path communication, leaving a potential $1.7\times$ speedup on the table through compute-communication overlap. Prior overlapping methods harness the fact that ML model state and inputs are already sharded into the number of GPUs, and overlap the compute and communication at shard granularity. However, such coarse-grained overlap suffers from limited network topology support, and suboptimal dataflows. In this work, we instead make a case for *finer-grain* compute-communication overlap which we term FiCCO. FiCCO operates at a level deeper than traditional sharding, and unlocks overlap for a wider set of network topologies and enables finer-grain dataflow.

We show that FiCCO opens up a wider design space of execution schedules than possible at shard-level alone. To walk the design space of schedules, we study and characterize the performance inefficiencies on doing overlap and overlay the schedules with the associated inefficiency signatures. Our characterization reveals decomposition and contention based slowdowns to be the major performance limiters, and we correlate the slowdown factors with the statically known compute/communication operator sizes. This helps us design heuristics (that frameworks and runtimes can harness) to select bespoke FiCCO schedules based on the nature of underlying ML operations. Finally, to further minimize contention inefficiencies inherent with operation overlap, we offload communication to GPU DMA engines. We evaluate several scenarios from realistic ML deployments and demonstrate that our proposed, heuristics-driven, bespoke schedules deliver up to $1.6\times$ speedup. Further, our heuristics provide accurate guidance to pick the optimal schedule in 81% of unseen scenarios.

Index Terms—Communication, DMA, Finer-grain overlap, GPU, ML

I. INTRODUCTION

The rapid escalation of compute and memory demands in machine learning (ML) workloads [47] has necessitated large-scale distributed execution. For example, training Llama-3 required nearly 16k GPUs [18]. These distributed setups employ various parallelization strategies [31], [39], [41], [68] to shard model states and inputs across devices. Such partitioning necessitates frequent communication collectives, such as all-gather, to synchronize and communicate data (e.g., activations) at periodic intervals.

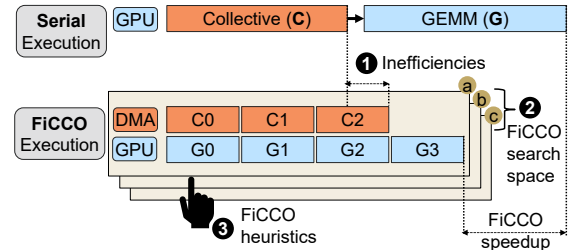


Fig. 1: Speedup via finer-grain decomposition of data-dependent communication-computation (FiCCO). FiCCO heuristics (3) picks the optimal schedule within the search space (2) to minimize overlap-induced inefficiencies (1).

While independent computation-communication can be overlapped to effectively hide communication latency, data-dependent computation-communication often leaves communication exposed on the critical path. An example of the former is fully-sharded data parallelism (FSDP) [68] technique, where weights are partitioned across GPUs and the communication of weights of the next layer can be overlapped with the computation of the current layer. In contrast, examples of data-dependent computation-communication include tensor-sequence parallelism [37] and context-parallelism [41], wherein result of the communication is immediately consumed by the subsequent operation. In such cases, exposed communication provides up to $1.7\times$ speedup opportunity if overlapped with the computation. We target this opportunity in this paper.

Prior works [6], [29] exploit the inherent sharding in the parallelization techniques and overlap computation and communication at the same shard-granularity (e.g., tensor parallelism shards a single layer's weights equally across GPUs). However, such coarse-grain shard-based techniques manifest a severe limitation that they harness peer-to-peer communication operations with only one other GPU at a time. While suitable for switch-based GPU networks (flexible bandwidth allocation), these techniques leave network links idle with direct-connection based GPU networks, delivering considerably lower performance (up to $3.9\times$ lower). Further,

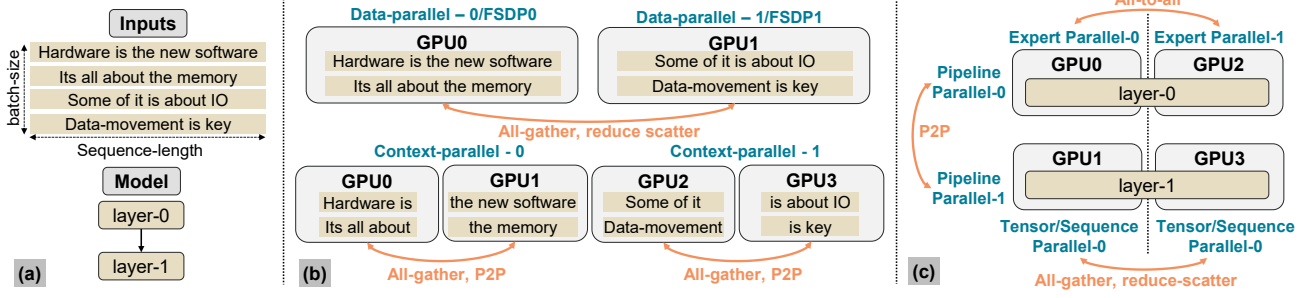


Fig. 2: (a) Sample inputs and ML model. (b) Data-parallelism (Fully-Shared Data-Parallel (FSDP)) and context parallelism shard the input. (c) Expert parallelism, context-parallelism, tensor-sequence parallelism shard model state amongst GPUs.

as they inherently operate at shard-granularity they limit pipelining granularity of subsequent operations.

We propose **FiCCO**, a *finer-grained* compute-communication overlap that decomposes communication one level deeper than traditional sharding (for example, in an 8-GPU system, FiCCO utilizes transfer sizes one-eighth those of shard-based methods). This deeper decomposition enables simultaneous all-to-all communication rather than serial peer-to-peer transfers, thereby reducing latency and maximizing link utilization across diverse topologies. Furthermore, FiCCO unlocks finer execution granularity for all subsequent operations in the dataflow.

As both shard-based and finer-grain techniques execute smaller compute/communication operators, they lead to various forms of execution inefficiencies (Figure 1), which are highly dependent on the nature of operators being overlapped. That is, the aggregate execution time of decomposed operations can be higher than the execution time of the single larger operation. This happens for myriad reasons including poor utilization and/or contention for GPU resources [5]. Since execution inefficiencies can vary based on the operator dimensions, a single execution schedule for computation/communication overlap can fail to realize optimal performance.

To tackle the above challenge, we provide a detailed characterization of inefficiencies incurred as ML operators are decomposed and overlapped, and correlate these inefficiency signatures with the associated operator sizes and shapes (referred to as static sizes henceforth). We next explore the FiCCO design space enabled by finer granularity, which encompasses varying communication shapes, computation uniformity, and computation granularity. The expanded search space of FiCCO schedules necessitates heuristics that can statically select the optimal schedule. We study how different schedules have tradeoffs in the inefficiency signatures, and correlate them with static operator dimensions to provide heuristics for frameworks and runtimes to pick bespoke FiCCO schedules. Finally, we also harness communication offloads to GPU DMA engines to further lower contention based inefficiencies. Our evaluation across several realistic ML deployments demonstrates that our heuristics driven bespoke schedules deliver up to $1.6\times$ speedup and our heuristics provide accurate guidance to pick

the optimal schedule in 81% of unseen scenarios.

In summary, we make the following contributions:

- We propose *finer-grain* compute-communication overlap than shard-based, which we term **FiCCO**, in order to overcome limitations of shard-based overlap.
- We provide detailed characterization of inefficiencies incurred as ML operators are decomposed and executed in an overlapped fashion.
- We provide a design space of FiCCO schedules which allows picking bespoke overlap schedules based on the nature of underlying ML operations. Further, we also offload communication to GPU DMA engines to lower contention-driven inefficiencies.
- We provide heuristics that guide frameworks and runtimes to pick bespoke FiCCO schedule for any given scenario, thereby maximizing performance.
- We evaluate several scenarios from realistic ML deployments and demonstrate that our proposed bespoke schedules deliver up to $1.6\times$ speedup and our heuristics picks the optimal schedule in 81% of studied scenarios.

II. BACKGROUND

A. ML - Increasingly Distributed

Large language models (LLMs) are becoming ubiquitous, from conversational agents to code generation [14], [35], [61]. The LLM architecture consists of a stack of transformer blocks, each transformer block made up of self-attention [21] and feedforward (MLP) layers. As LLM model sizes and training dataset sizes scale [15], [27], the compute and memory throughput of a single GPU is insufficient to meet LLM needs [47]. Consequently, parallelization techniques shard model state and inputs across multiple GPUs.

Figure 2 depicts key ML parallelization techniques and their associated communication collectives. Vanilla data-parallelism [40], [65] shards model inputs across GPUs. Fully-sharded data-parallel (FSDP [68]) additionally shards model weights, requiring all-gather (AG) prior to executing associated computation. In addition, reduce-scatter (RS) of model gradients, in which gradients across GPUs are summed element-wise, is also necessary (Figure 2(b)). Also depicted is context parallelism [66], which shards single input sequence

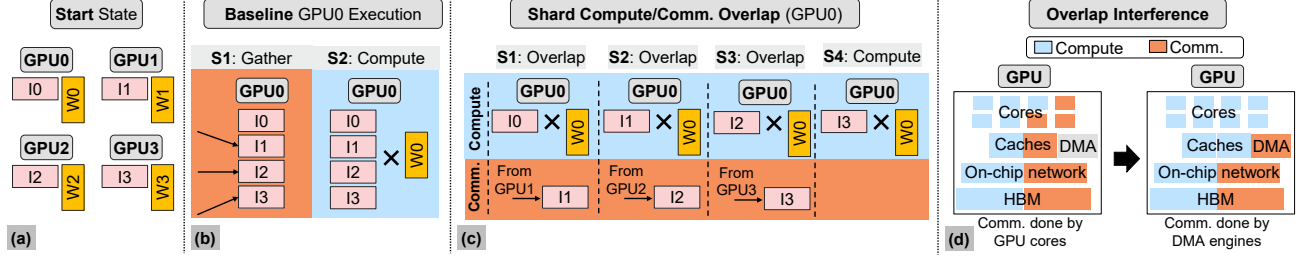


Fig. 3: (a) Start state for weights (W) and inputs (I) on a four GPU system. (b) Steps(S) in baseline serial execution of communication and computation. (c) Shard-based overlap: **P2P** shard communication and shard-level computation overlap. (d) Contention within limited GPU resources with operation overlap, without and with DMA communication offloads. DMA eliminates compute interference and some portion of memory interference within the GPU.

data across GPUs and can incur either ring-based peer-to-peer (**P2P**) [42] or **AG** [18] communication.

Alternative parallelization techniques shard model state across GPUs (Figure 2(c)). In order to scale model parameters without commensurate scaling of compute, mixture-of-expert (MoE) models have gained increasing traction [53], [55]. Such MoE models often deploy expert parallelism, wherein experts in MLP layers are distributed over multiple GPUs incurring all-to-all (**AA**) communication to disperse and collect input tokens across experts. Pipeline parallelism [31], spreads model layers across GPUs in stages, necessitating **P2P** communication of outputs/inputs between said stages. Finally, tensor parallelism [44] shards linear layers in transformer block across GPUs, which is often combined with sequence parallelism [37] to shard sequence-dependent operations as well. This necessitates **AG** and **RS** collectives across participating GPUs.

B. Shard-based Overlap and Interference

With wide-spread deployment of distributed ML, effectively hiding resultant inter-GPU communication in the shadow of computation gets increasingly important. This is straightforward in presence of independent computation, such as in FSDP, wherein all-gather of weights for next layer can be done in shadow of computation of the previous layer. However, for other forms of parallelization highlighted in Figure 2, this is challenging. As an example, Figure 3(a) depicts the start state for tensor-sequence parallel scenarios across four GPUs where each holds a row-sliced input shard (I) and column-sliced weight shard (W). In baseline execution (Figure 3b), the computation—a matrix-matrix multiplication (GEMM)—requires every input shard to interact with the weight shard on each GPU. Consequently, an All-Gather collective must first synchronize all input shards across the GPUs before the GEMM kernel is invoked. Section VI-B shows that this serialized execution can leave up to $1.7\times$ of the ideal operator-level possible speedup unrealized.

To overcome this, prior works [6], [29] harness the fact that model’s inputs/states are already sharded allowing compute-communication overlap as depicted in Figure 3(c). Instead of gathering all input shards upfront, shards are sent peer-to-peer

between GPU pairs one at a time, while overlapping computation on the previously received shard, thereby attaining compute-communication overlap.

Such overlap of computation and communication is realized on modern GPUs via executing two concurrent GPU kernels, one for computation and the other for communication. Such concurrency, as expected, causes interference between the two as depicted in Figure 3(d) across different GPU resources (e.g., cores, caches, HBM). Prior works [5], [48], [69], [70], offload communication to existing DMA engines on GPU to lower a substantial portion of this interference. That is, by not using GPU cores to orchestrate communication, compute interference is completely eliminated and a portion of cache interference is eliminated as well. We also harness such DMA offloads.

III. FiCCO: MOTIVATION

A. Case for Finer-grain Overlap

While shard-based overlap as employed by prior works is indeed promising, we observe that it suffers from key limitations. To this end, we make a case in this work for *finer-grain* compute-communication overlap which we term **FiCCO**, wherein communication is decomposed at one-level deeper granularity (that is, transfer sizes one-eighth that of shard-based overlap in an eight GPU system) which allows overcoming of limitations with shard-based overlap (discussed in more detail below). While shard-based techniques execute compute and communicate at shard granularity, we propose to communicate at one-level deeper granularity (additional sharding by number of GPUs) while keeping the computation granularity configurable and tunable (either match or increase granularity than shard-based techniques). This enables us to communicate at a granularity finer than shard while keeping the computation granularity configurable.

B. FiCCO: Steady State

Higher degree of decomposition with FiCCO unlocks concurrent all-to-all communication pattern in steady state as opposed to serialized peer-to-peer communication pattern in shard-based overlap as depicted at a high-level in Figure 4 ((b) versus (c)). As also depicted, since computation now happens

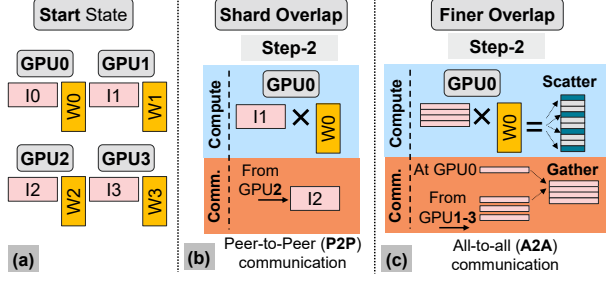


Fig. 4: Shared-based overlap versus FiCCO in action. FiCCO steady state unlocks all-to-all communication, and as a result introduces additional actions such scatter and gather.

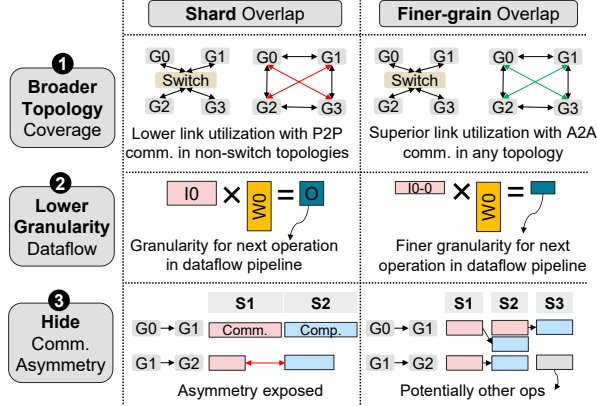


Fig. 5: Benefits of Finer-grain overlap. I_0, W_0 : Input, Weight shard in GPU0 respectively; G: GPU; S: Step.

over finer-grain communication received from multiple peer GPUs, depending on the choice of computation granularity (as will be expanded in Section V), this induces some additional actions in steady state such as gathering of finer-grain communication buffers (**Gather**) and potentially scattering of finer-grain outputs in final output space (**Scatter**). We further discuss in Section V how this unlocks a rich design space of FiCCO schedules.

C. FiCCO: Benefits

We depict in Figure 5 the benefits of FiCCO in comparison to shard-based overlap. First, FiCCO broadens compute and communication overlap for data-dependent scenarios for wider topologies. As discussed above, shard-based overlap deploys peer-to-peer communication which is suitable for switch-based GPU topologies [56] which allow flexible bandwidth allocation between GPUs. That said, state-of-the-art GPUs such as AMD Instinct™ MI300X employ direct connection based topologies. In such cases, peer-to-peer communication can leave network links unused causing performance degradation – up to $3.9\times$ slowdown compared to serial execution, as discussed further in Section VI-B. In contrast, FiCCO, by decomposing communication one-level deeper (additional sharding by number of GPUs) unlocks all-to-all communication

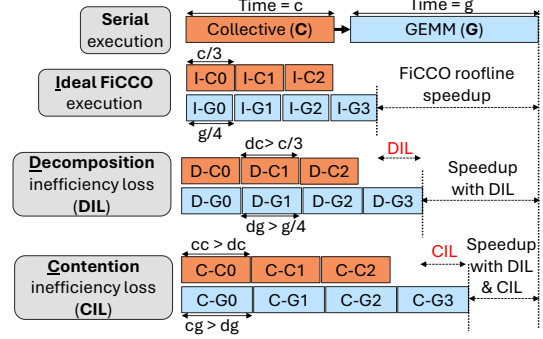


Fig. 6: Inefficiencies with operator decomposition and overlap. $d[c,g]$ and $c[c,g]$ represent [Collective, GEMM] times with decomposition and contention respectively. Roofline speedup refers to the *ideal* speedup in the scenario where sharding or overlapping operators do not cause efficiency losses.

pattern which can keep direct connection based topologies well utilized. Second, finer-grained outputs resulting from FiCCO also enable subsequent operations to execute, and therefore pipeline/overlap, at a finer-granularity. Finally, as also depicted in Figure 5, finer-granularity can also allow better hiding of communication asymmetry that can happen with MoE models wherein communication between pair of GPUs can differ based on tokens to be communicated between them.

IV. DATA-DEPENDENT COMPUTE/COMMUNICATION OVERLAP: CHARACTERIZATION OF INEFFICIENCIES

As both shard-based or finer-grain techniques execute smaller compute/communication operations as compared to baseline serial execution (Figure 3), they lead to various forms of operation execution inefficiencies, which are highly dependent on the nature of operators being overlapped. That is, the aggregated execution time of decomposed operations can be higher than the execution time of the baseline (single large) operation. Careful understanding and characterization of these inefficiencies is necessary in order to effectively manage them, and we provide the same in this section.

A. Overview of Inefficiency Losses

We depict pictorially in Figure 6 the key inefficiency losses that can incur due to operation decomposition and overlap. Note that the inefficiencies we discuss apply to both shard-based overlap and our proposed FiCCO overlap. In the characterization sections (Section IV-C, IV-D) we explicitly separate out the empirical differences between the two.

Ideal Execution: To express the inefficiency losses, we first depict ideal execution in Figure 6. In such an execution, decomposing and overlapping of ML operations scales the execution time commensurate to the decomposition degree. This leads to what we refer in this work as **ideal** speedups for computation/communication overlap.

Decomposition Inefficiency caused Loss (DIL): In practice, decomposing operations into smaller sub-operations (as

done by both shard-based and FiCCO techniques) leads to slower execution than predicted by the ideal baseline (Figure 6). This is because smaller inputs/outputs are less capable of saturating GPU compute units and load-store pipelines, yield poorer cache reuse due to smaller GEMM tile sizes, and suffer from other well-documented inefficiencies [45] [58]. We term this as decomposition inefficiency caused loss (DIL henceforth).

Contention Inefficiency caused Loss (CIL): Additionally, unlike baseline execution where computation and communication run in isolation, both shard-based and FiCCO techniques overlap these operations, introducing contention in GPU compute and memory subsystems. More specifically, as depicted in Figure 3(d), overlapping computation and communication divides GPU compute cores between two concurrent kernels, causing each to receive fewer cores than it would in isolation, an effect we term compute interference. The same phenomenon plays out over caches, network on-chip and HBM. While offloading communication to DMA engines [5], [48] can eliminate compute interference and some portion of cache interference, memory interference still remains. We term this contention related operation execution slowdown as contention inefficiency caused loss (CIL henceforth).

Other Inefficiency Losses: We observe in this work that DIL and CIL are the primary inefficiency losses and focus on them. That said, other inefficiency losses exist and we briefly mention them here. ML models repeatedly execute the same-sized operations over multiple layers across multiple GPUs. We observe variations in execution times of the same operation within the same GPU and across GPUs. However, we find that this variation is within 6%, so we omit it. Finally, CPU launch of GPU kernels, which increase with both shard-based and our proposed FiCCO techniques can add launch overheads. These overheads can matter when operation sizes are particularly small. However, graph-based launch techniques can mitigate these overheads [11].

B. Methodology

1) *System and Execution Setup:* Our setup comprises of AMD MI300X Infinity Platform comprised of $8 \times$ AMD Instinct MI300X GPUs [1] with a fully connected topology using AMD Infinity Fabric™ bi-directional links, each link with a uni-directional bandwidth of 64GB/s. Our setup is based off PyTorch [46] as it is the most common framework for model training and inference. We focus on overlap of key ML operations: GEMM and communication collective kernels. For GEMMs, we harness AMD ROCm™ hipblasLt [2] library of high-performance GEMM kernels. For communication, we either harness AMD ROCm communication collectives library (RCCL) [8], a standard library of collective communication routines for GPUs or launch memory copies (*hipMemcpyDtoDAsync* [7]) to offload communication to GPU DMA engines. We use multiple GPU streams [9] that allow concurrent execution of GPU kernels. We harness symmetric memory [64] for input/output buffers to eliminate intermediate buffer copies and enable direct

TABLE I: GEMMs occurring in real world scenarios.

Name	Parallelism	Model	GEMM (M,N,K)
g1	SP+TP	llama-3-405b	(16384,16384,131072)
g2	SP+TP	llama-3-405b	(131072,16384,16384)
g3	SP+TP	llama-3-405b	(53248,16384,131072)
g4	SP+TP	llama-3-405b	(131072,53248,16384)
g5	SP+TP	llama-2-70b	(8192,8192,262144)
g6	SP+TP	llama-2-70b	(262144,8192,8192)
g7	SP+TP	llama-2-70b	(28672,8192,262144)
g8	SP+TP	llama-2-70b	(262144,28672,8192)
g9	SP+TP	llama-3-405b	(196608,18432,16384)
g10	SP+TP	llama-3-405b	(196608,106496,16384)
g11	SP+TP	llama-2-70b	(1048576,10240,8192)
g12	SP+TP	llama-2-70b	(1048576,57344,8192)
g13	EP	DeepSeek	(1607680,57344,8192)
g14	EP	Mixtral	(147456,28672,4096)
g15	EP	Mixtral	(327680,28672,4096)
g16	EP	Mixtral	(229376,28672,4096)

peer-to-peer GPU memory access. We report the average of five measured runs following ten warm-up iterations.

2) *Data-dependent Overlap Scenarios Under Consideration:* We study data-dependent compute-communication scenarios from real-world ML deployments in this work [18], [23], [43], [59], [60], summarized in Table I. More specifically, we study tensor-sequence parallel scenarios wherein, model weights are sharded and activations are all-gathered [37]. We also study scenarios for expert-parallelism [22], [34], wherein input tokens are communicated in an all-to-all fashion before executing the expert layers.

Finally, as we discussed in Section II-A, there exist other data-dependent compute/communication overlap scenarios. We omit some of them that require communication with associated arithmetic operations as GPU DMA engines lack compute capabilities today (e.g., tensor parallelism with reduce-scatter). If supported in future systems, we believe our conclusions/analysis will apply in those scenarios as well.

C. DIL for Compute and Communication

1) *GEMM DIL:* We characterize the decomposition overhead as the GEMM is decomposed for shard-based and *finer-grain* overlap. To evaluate GEMM DIL, we compare the isolated execution of baseline GEMM to that of eight 8-way sharded and sixty-four 64-way sharded smaller GEMMs. As vanilla GEMM is comprised of **M** rows, **K** inner reduction columns dimension and **N** output columns, sharding of activations/gradients can be done in either the row (**M**) or column (**K**) dimension. We exercise both. Additionally, note that column-sharding necessitates accumulative GEMM kernels (that is, $C += A * B$). Finally, note that, 8-way sharding is employed in both shard-overlap and FiCCO, while 64-way GEMM-sharding can be employed optionally in FiCCO. That is, FiCCO shards communication one-level deeper than shard-based techniques but only optionally shards computation (Section V).

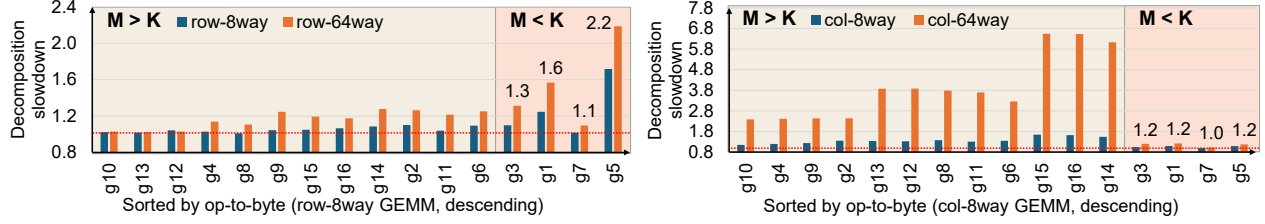


Fig. 7: Decomposition Inefficiency caused Loss (DIL) for GEMM with 8-way and 64-way row (M) or column (K) sharding. Higher degree of sharding leads to higher DIL. DIL is higher when sharding along the smaller of the M and K dimensions. Sensitivity to DIL increases as the static op-to-byte (OTB) ratio for the GEMM decreases.

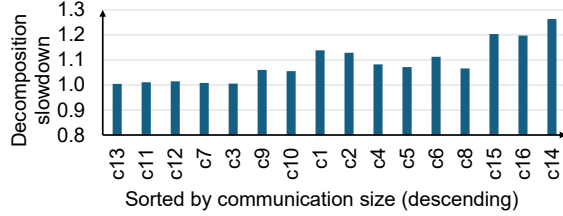


Fig. 8: DIL for collective with DMA based all-gather.

Observations: Figure 7 depicts DIL for GEMMs. First, as expected, 64-way sharding leads to higher DIL (slowdown) as compared to 8-way sharding for both row and column sharding. Second, DIL depends on relative magnitude of GEMM rows (M) and inner-dimension (K), i.e., row-sharding incurs higher overhead when $M < K$, and column-sharding when $M > K$. That is, for $g3, g1, g7, g5$, DIL for row-sharding is higher than column-sharding (as depicted with data labels) and vice-versa for rest of GEMMs. Additionally, DIL generally increases as static op-to-byte (OTB) for GEMM (arithmetic intensity, calculated using resultant GEMM dimensions, derived from MNK for ops and bytes) decreases. That is, lower the OTB ratio, more the sensitivity to DIL. Finally, as static GEMM parameters (OTB) dictate DIL, we note this can aid in design of heuristics for picking optimal schedules (Section V-C).

2) *Communication DIL:* For communication DIL, as individual transfer size (between a pair of GPUs) in shard-overlap is same as baseline serial execution, shard-overlap does not experience DIL. However, as FiCCO communicates at finer-granularity, we scale down collective size 8-way for eight GPU system and study resultant slowdown.

Observations: We depict DIL for communication all-gather in Figure 8. We observe that communication DIL for FiCCO has a geomean of about 10% and positively correlates with communication size as expected. That is, as size increases, there is increased resiliency to DIL as communication gets increasingly bandwidth-bound.

D. CIL for Compute and Communication

We compare overlapped and isolated execution of communication/computation to quantify contention overheads.

1) *GEMM CIL:* To evaluate GEMM CIL, we execute 8-way M-sharded GEMM concurrently with RCCL [8] or DMA-based all-gather and report the slowdown relative to isolated execution. In this case, as DIL is already baked into GEMM execution, slowdown observed is purely attributed to contention between GEMM and communication.

Observations: We depict CIL for GEMM in Figure 9 (left). We first observe that, DMA-based communication causes far lower CIL than traditional GPU core-driven communication (RCCL) in all cases. This is expected as DMAs eliminate compute interference and reduce memory interference [5]. While clean correlations as in DIL are not as evident for CIL, we observe that CIL generally increases as static memory traffic (MT) for GEMM increases (calculated as $MK + NK + MN$ for a MNK GEMM). This is so as memory sub-system interference increases with larger GEMM memory traffic. We also evaluate the CIL for shard-based overlap (omitted for space reasons) and observe a geomean CIL of $1.07\times$ slowdown as opposed to geomean CIL of $1.11\times$ with FiCCO.

2) *Communication CIL:* Using the same setup as above we report slowdown in the communication kernel.

Observations: Figure 9 (right) depicts that communication CIL similarly shows sensitivity to GEMM memory traffic and we observe a geomean CIL of $1.12\times$ slowdown. In contrast, with larger communication sizes, shard-based overlap experiences geomean CIL of $1.03\times$ slowdown.

E. Case for Bespoke FiCCO Schedules

Figure 10 illustrates how DIL and CIL proportionally affect the scenarios under consideration for GEMMs (8-way in (a), 64-way sharding in (b)) and all-gather (c). For GEMMs, as the combination of OTB and MT increases, GEMMs become more susceptible to CIL (left to right in the figure). This is expected based on trends we discussed above that DIL negatively correlates with OTB and CIL positively correlates with MT. This effect however, is less prominent for 64-way sharding as the DIL is particularly more prominent there (Section IV-C1). Communication DIL/CIL proportioning shows similar trend in terms of GEMM MT. Overall, since different scenarios exhibit different inefficiency signatures, using FiCCO schedules cognizant of these losses and guided by heuristics for selecting the optimal schedule, stand to deliver better performance.

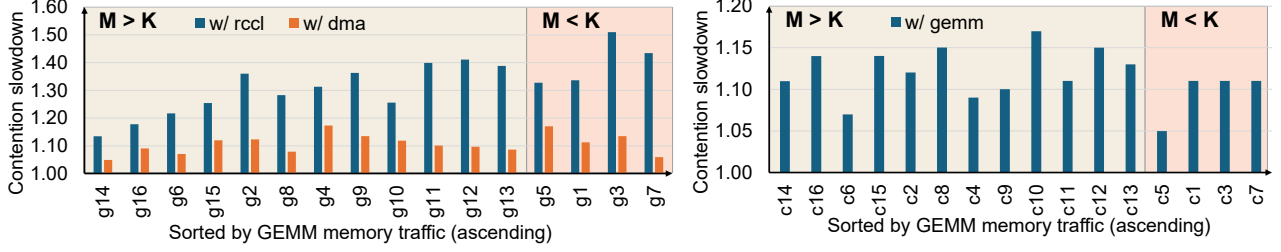


Fig. 9: Contention Inefficiency caused Loss (CIL) for GEMM (left) and all-gather communication (right). We compare CIL for GEMM for both RCCL and DMA-based all-gather. CIL for all-gather is with to 8-way M-sharded GEMM. CIL increases as the static memory traffic (MT) induced by the GEMM ($MK + NK + MN$ for an MNK GEMM) increases.

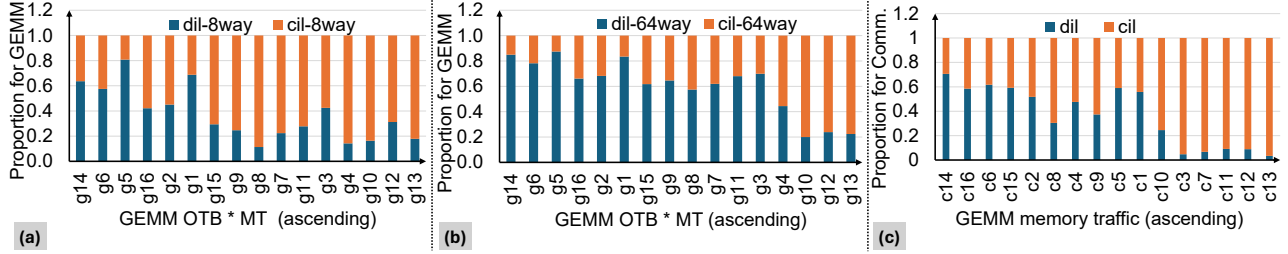


Fig. 10: Proportion of DIL versus CIL for GEMMs (g) and all-gather (c). DIL negatively correlates with OTB, while CIL positively correlates with MT. Since GEMMs or collectives of different sizes exhibit different OTB and MT, each size is affected differently by DIL and CIL under different sharding strategies.

V. FiCCO: DESIGN SPACE & HEURISTICS

We discuss in this section both the design space unlocked via FiCCO and the schedules we study in this work.

A. FiCCO Design Space Dimensions

Recall from Section III that FiCCO employs *finer-grained* compute-communication overlap, decomposing communication one level deeper than shard-based techniques (for example, in an eight GPU system, one-eighth the transfer size of shard-based overlap). Specifically, while shard-based techniques operate at shard granularity for both communication and computation, FiCCO decouples the communication and computation granularity. FiCCO communicates at the finer granularity, while computation granularity is configurable—either matching or finer than shard-based techniques.

Consequently, the design space dimensions possible with additional communication sharding in FiCCO are depicted in Figure 11a. Specifically, for communication, resultant buffers can be 1-dimensional (**1D**) with sharding in row (M) dimension or 2-dimensional (**2D**) with sharding in column (K) dimension allowing two choices in this dimension. Further, for computation, FiCCO unlocks four choices. First, in each overlap step, as GPUs receive buffers from all peer GPUs, either a single/fused GEMM kernel using all the buffers can be executed (**fused**) or individual kernels allowing flexible scheduling (**unfused**) can be executed. Second, at the start of the execution (first step), in order to hide exposed communication, each GPU can choose to start executing on its local

shard without waiting for any data from peer GPUs leading to heterogeneous subsequent steps (**hetero**) or combine remote and local buffers to ensure all steps execute the exact same GEMM (**uniform**). We next discuss concrete FiCCO schedules to study how each choice manifests tradeoffs in terms of inefficiency loss signatures.

B. FiCCO Design Space Schedules

Figure 11b illustrates the four FiCCO schedules studied, depicting their steady-state actions. We classify schedules by computation uniformity (*uniform*, *hetero*), granularity (*fused*, *unfused*), and communication shape (*1D*, *2D*). Also, recall from Section III-B that FiCCO may require gathering of finer-grained communication buffers (**Gather**) and potentially scattering outputs into the final output space (**Scatter**), depending on the compute dimension and communication sharding choices. Finally, we assign each schedule an inefficiency loss signature in terms of DIL and CIL, based on its resultant GEMM dimensions and concurrency degree.

Decoding the provided schedules in a comparative fashion, we observe that all schedules communicate the same effective buffer size and that *uniform-fused-2D* communicates 2D buffers. Next, to ensure uniformity in GEMM sizes, all uniform schedules incur gather of local and remote received buffers. Next, some schedules either invoke single fused GEMM kernel or some invoke multiple GEMM kernels (unfused). Finally, some schedules scatter output in final output space as they compute on non-contiguous rows in input

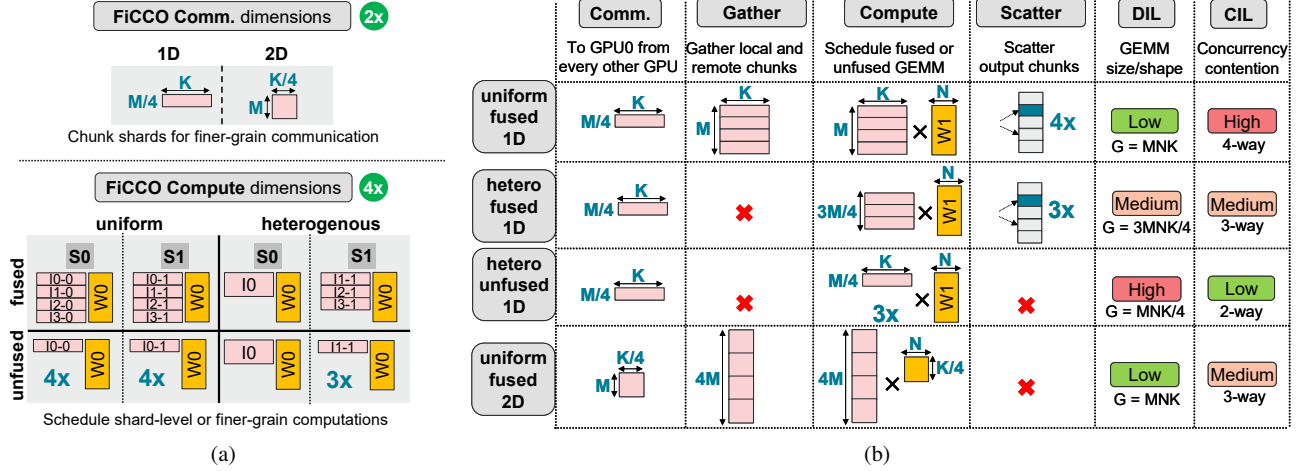


Fig. 11: (a) FiCCO design space consists of three design choices namely communication dimensions, computation uniformity and computation granularity. This results in eight possible schedules. (S: Step) (b) FiCCO schedules under consideration.

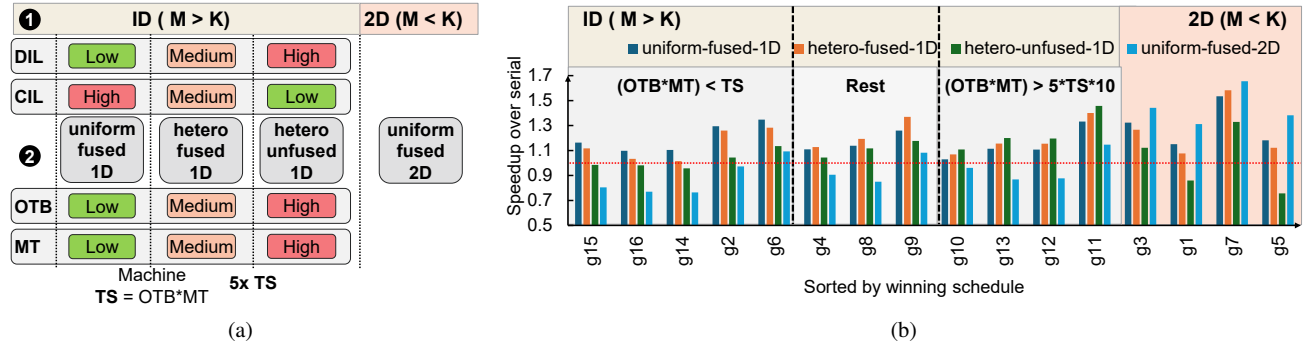


Fig. 12: (a) FiCCO heuristics are governed by the static GEMM op-to-byte/OTB and GEMM memory traffic/MT. (b) The performance of the four FiCCO schedules across our GEMMs.

buffer. We assign DIL degree to schedules based largely on resultant GEMM size for it dictates the GEMM OTB and hence DIL (Section IV-E). Also, we assign CIL degree to a schedule based on the concurrency degree it manifests (e.g., *uniform-fused-1D* can execute communication, gather, compute, and scatter at same time), stressing memory traffic which dictates CIL (Section IV-E).

While with our proposed design space (Section V-A) a total of eight schedules are possible, we study four of these. This is because we observe that the inefficiency loss signatures of these missing schedules are strictly worse than the ones we study. For the three other 2D schedules ($M < K$), row-sharding is suboptimal. Thus, creating *hetero* or *unfused* schedules that shard in row-dimension would further degrade performance. Similarly, *uniform-unfused-1D* further shards *hetero-unfused-1D*, thereby increasing DIL while keeping CIL unchanged since the Scatter/Gather operations remain unchanged.

C. FiCCO Heuristics

Since FiCCO leads to distinct schedules, selecting the optimal one is essential for maximizing performance. While we can benchmark common GEMM dimensions in distributed ML, the vast diversity of batch sizes, sequence lengths, and model scales makes exhaustive offline profiling infeasible. We therefore utilize heuristics to enable optimal FiCCO execution based on static parameters, that also works for future/unseen shapes and sizes. The heuristics can aide frameworks and runtimes to pick bespoke schedules. We present our design for this heuristic in Figure 12a. Specifically, we first observe that relative magnitude of GEMM row (M) and column dimension (K) can provide a clear guidance on choice of communication shape to pick: **1D** if $M > K$ or **2D** otherwise in order to minimize resultant DIL (Section IV-C1). For latter, there is a single FiCCO schedule available (*uniform-fused-2D*). For the former, first recall that each FiCCO schedule has a unique and inherent inefficiency loss signature (DIL-CIL degree) as depicted in Figure 12a. As such, we match each schedule with operations least sensitive to their inherent inefficiencies. As

an example, since *uniform-fused-1D* inherently manifests low DIL and high CIL, it is selected for scenarios with low GEMM op-to-byte (OTB) and low memory traffic (MT), since DIL negatively correlates with OTB and CIL positively correlates with MT (Section IV-E). We similarly match *hetero-unfused-1D* with high combination of OTB and MT and assign *hetero-fused-1D* schedule otherwise.

To demarcate the resulting OTB and MT tranches, we leverage machine-level hardware characteristics. We define a combined machine threshold using combined OTB and MT for underlying GPU in terms of its peak compute FLOPs (op-to-byte $\times$ memory bandwidth = FLOPs). We then assign *uniform-fused-1D* to scenarios where the combined OTB and MT is below this machine-level threshold, while *hetero-unfused-1D* is assigned when the combined metric exceeds the threshold by $5\times$. Note that this FiCCO heuristic threshold is machine-specific and requires a one-time tuning effort. The threshold of $5\times$ was chosen by comparing the performance of two FiCCO schedules (*hetero-fused-1D* and *hetero-unfused-1D*) for a set of GEMM microbenchmarks on AMD Instinct MI300X. Other machines would require a similar tuning to choose the threshold, which is a small one-time cost. We evaluate this heuristic both for scenarios under consideration and for synthetic scenarios and report efficacy in Section VI-D.

VI. FiCCO EVALUATION

A. FiCCO Realization

Our system setup is described in Section IV-B1. FiCCO schedules are implemented in PyTorch by invoking standard GEMM libraries and DMA transfers, without modifying the underlying optimized kernels. We use *hipStreamWrite* and *hipStreamWait* [3] for lightweight synchronization. To incorporate FiCCO, the user provides only the GEMM input sizes; based on the GEMM dimensions our heuristics will select and execute the optimum overlap schedule, replacing the serial communication and computation.

B. Shard-based Overlap: Limitations

We start with motivating FiCCO by demonstrating speedup potential with ideal computation-communication overlap, and limitations of current shard-overlap based techniques in Figure 13. Ideal performance assumes operator decomposition scales linearly with no slowdowns from decomposition or contention. We implement shard-overlap following PyTorch Async Tensor Parallelism [29] technique.

Ideal performance follows a bell curve relative to the GEMM/communication time ratio (along x-axis), that is, the more balanced the operators, the greater the benefit of perfect overlap. However, shard-based overlap, with serialized peer-to-peer communication, under-utilizes the available network links in AMD Instinct MI300X (we observe $7\times$ communication slowdown). We observe a negative correlation between speedup and the GEMM/communication time ratio; as GEMM execution time increases (moving rightward on the x-axis), they better mask communication inefficiencies. Regardless,

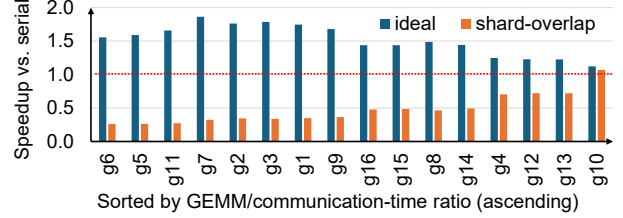


Fig. 13: Deficiencies of shard-based overlap. Due to the considerable communication slowdown of shard-based overlap, layers with a higher fraction of collective time incur greater slowdowns.

shard-overlap does not attain speedups because the communication bottleneck translates to an overall $2.3\times$ slowdown compared to serial execution. Since the performance of shard-based overlap is bottlenecked by inter-GPU communication, any optimization in intra-GPU computation is unlikely to benefit. FiCCO avoids this communication slowdown by using finer-grained sharding and all-to-all communication to exploit all available links.

C. FiCCO Schedules Comparison

Figure 12b depicts the speedups attained by the FiCCO schedules we discussed, with our heuristic selections overlaid. Since 2D DMA memory copies are currently unsupported, we emulate them using equal-sized 1D copies (the performance of 1D and 2D copies are quantified in Section VI-G). Unlike shard-based methods, FiCCO leverages all-to-all communication to improve network utilization, attaining speedups of up to $1.6\times$ with 1D schedules and $1.7\times$ with emulated 2D schedules.

D. FiCCO Heuristic Evaluation

Our proposed heuristic in Figure 12a predicts the right schedule for all scenarios we study in this work. To further evaluate the efficacy of this heuristic, we generate sixteen additional synthetic scenarios with diverse OTB and MT combinations. Across these scenarios, our proposed heuristic predicts the right schedule for 81% of scenarios, further underscoring its efficacy. For the scenarios where we mispredict, the schedule proposed by our heuristic loses approximately 14% of the optimal speedup.

E. Application Level FiCCO benefits

We use vLLM [38] to profile layer-wise execution times across four LLMs [23], [24], [57]. For each FiCCO-eligible layer, comprising all-gather and all-to-all collectives, we identify the corresponding GEMM and collective tensor sizes, measure their FiCCO speedups in isolation, and apply these to the respective layers. Figure 14 presents the resulting layer-level and end-to-end (E2E) speedups. On average, a $1.21\times$ speedup in FiCCO-eligible layers yields a $1.07\times$ E2E speedup, retaining 33.3% of the layer-level gain. The MoE models

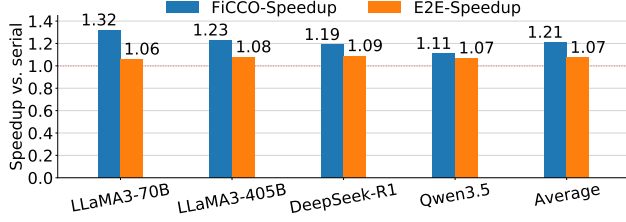


Fig. 14: LLM end-to-end speedups attained with FiCCO

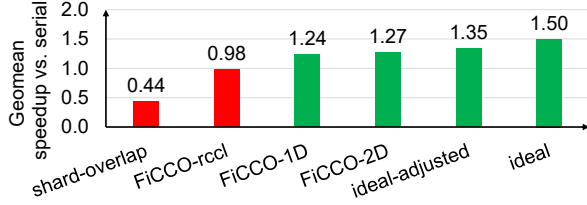


Fig. 15: Comparing FiCCO to other techniques.

benefit the most: averaged across DeepSeek and Qwen, a $1.15\times$ layer speedup translates to a $1.08\times$ E2E speedup, retaining 53.3% of the gain.

F. FiCCO Comparison To Other Overlap Techniques

Figure 15 depicts geomean speedups across all scenarios for shard-overlap, and FiCCO 1D and emulated 2D schedules. We also implement FiCCO with GPU core-based RCCL communication denoted as *FiCCO-rccl*. Ideal speedup refers to the maximum attainable speedup if sharding or overlapping operators did not cause any efficiency losses. Ideal-adjusted speedup refers to the maximum attainable speedup after empirically accounting for DIL and CIL losses in the operators. We find that FiCCO 1D and 2D schedules attain 48% and 54% of the ideal speedup, and 68% and 77% of the ideal-adjusted speedup, respectively.

Finally, we attempted comparing against Triton-Distributed [69], but code analysis revealed it to allocate memory for entire gathered buffers, causing out-of-memory (OOM) for our evaluated production GEMMs. In contrast, FiCCO optimizes memory allocations by double buffering. The Triton-Distributed paper uses smaller shapes to avoid OOM errors. Evaluating those exact sizes, FiCCO delivers a $1.05\times$ speedup over Triton-Distributed.

G. Quantifying Overheads with 2D DMA transfers

As of today, GPU DMA-based 2D copies are not exposed via the runtime, and consequently we emulated FiCCO schedules requiring 2D transfers by executing 1D transfers of the same effective size. That said, via low-level programming of AMD’s GPU engines, we offloaded 2D copies to DMA engines and compared how far off our emulation is from reality. For our evaluated sizes, we find that on average, 2D-copies incur a negligible slowdown of 0.03%

compared to 1D-copies. Though a 2D-copy, being a group of smaller 1D-transfers, can incur additional address-translations and network-inefficiencies, these overheads are minimized in FiCCO. First, GPUs utilize large 2MB pages and our transfer sizes are substantial, making the translation overhead negligible. Second, AMD DMA engines leverage back-to-back copy overlap optimization [49], overlapping the writes of one copy with the reads of the next and mitigating efficiency loss from these grouped transfers. Overall, we believe that when future runtimes expose 2D DMA transfers, FiCCO 2D schedules will deliver close to the emulated performance we report in our work.

VII. RELATED WORKS

Table II compares prior compute-communication overlap works. There has been significant research to overlap independent computation and communication [19], [20], [25], [32], [50], [51], but these solutions do not work in dependent computation-communication. CoCoNet [33], Google-Decomposition [63], Centauri [13], Flux [12], Comet [67], Domino [62], Concerto [16], FlashOverlap [28] decompose the operators, and schedule the overlap such that the dependency is satisfied. However, these solutions do not use DMAs, and therefore incur contention between communication [30] and computation kernels for the compute resources. We use DMAs for communication, thereby reducing contention. In addition, several of the solutions require writing a customized GEMM kernel to suit their implementation, and do not use the existing highly optimized GEMM kernels [2], [10]. We make no changes to the existing GEMM kernels to take advantage of their optimizations.

TileLink [70] and Fused-Operators [52] employ fused kernels of compute and communication operators. However, fused kernels often require user intervention to implement communication primitives and tiling optimizations. Furthermore, Fused-Operators does not use DMA, thereby incurring compute contention. DMA based compute communication solutions include ConCCL [5], PyTorch AsyncTP [29] and Distributed-GEMM [6]. ConCCL works only if compute and communication operators are independent. PyTorch AsyncTP, and Distributed-GEMM employ shard-overlap and are optimized for switch based P2P topologies but do not work well on full-mesh topologies. We employ finer-grain overlap that maximally saturate the bandwidth on full-mesh topologies and achieve good performance. ACE [54] and T3 [48] propose new hardware modules to handle communication to reduce compute contention. They require hardware changes, while ours is a software solution.

Finally, based on a thorough characterization of overlap inefficiencies, we provide a design space for finer grain overlap and provide heuristics to pick the optimal one. To the best of our knowledge, this design space exploration has not been done before.

TABLE II: Comparison of FiCCO with prior compute-communication overlap works.

Name	Dependent Comp/Comm	DMA- based	Use Optimized GEMM Libraries	Full-Mesh Support	Require no HW Changes	Design Space Exploration
CoCoNet [33], Concerto [16], Google-Decomp [63]	✓	✗	✗	✗	✓	✗
Centauri [13], Domino [62], Comet [67], FlashOverlap [28]	✓	✗	✓	✗	✓	✗
Flux [12], Fused-Ops. [52]	✓	✗	✗	✓	✓	✗
TileLink [70]	✓	✓	✗	✓	✓	✗
ConCCL [5]	✗	✓	✓	✓	✓	✗
AsyncTP [29], Dist-Gemm [6]	✓	✓	✓	✗	✓	✗
ACE [54]	✓	✗	✓	✓	✗	✗
T3 [48]	✓	✓	✓	✓	✗	✗
FiCCO	✓	✓	✓	✓	✓	✓

VIII. DISCUSSION

A. FiCCO Benefits For Switch-based Topologies

While we study FiCCO for direct/fully-connected topologies, its benefits carry over to other topologies as well (e.g., switch-connected topologies). As an example, on decomposing a GEMM, the tradeoff (Section IV) between increased decomposition loss and reduced contention (due to decreased memory-traffic) would persist even in a switch-topology. Specifically, Figure 12b/hetero-infused-1D shows computation sharding incurs lower contention without high decomposition loss and this is agnostic of underlying network topology. As such, FiCCO-like sharding can enable superior dataflow (Figure 5) for all topologies¹.

B. FiCCO Power Behavior

Concurrent kernel execution can utilize GPU power budget more efficiently via utilization of heterogeneous resources (compute, network, etc.). Prior work [4] analyzes concurrency power and shows that power gets re-allocated between compute and memory based on what is stressed. It also shows that concurrent power can be higher than the isolated execution of compute or communication. However, FiCCO harnesses DMA communication which is more energy-efficient [49], and thus FiCCO power drawn will be lower. Finally, overlapping computation and communication stands to mitigate problematic power swings [17], [36], which is observed in the serial execution of computation and communication.

C. FiCCO with Device-initiated Communication

Recent works such as GPU-initiated networking (GIN) [26] enable GPUs to directly initiate communication, eliminating CPU coordination overhead and allowing efficient fusion. While beneficial, they also steal GPU core cycles to initiate and manage communication and as such, both GIN and host-initiated communication are judiciously employed. That said, FiCCO is fully compatible with both device-initiated and host-initiated communication and can be coupled with either. FiCCO schedules and associated heuristics are mainly governed by static GEMM dimensions (op-to-byte and memory

traffic) and as such these heuristics should also carry over (with one-time tuning cost for thresholds) to device-initiated communication.

IX. CONCLUSION

We make a case in this work for *finer-grain* compute-communication overlap which we term FiCCO, where we argue for finer-granularity, one-level deeper sharding than the commonly deployed shard-level overlap, to unlock compute/communication overlap for wider set of network topologies, finer-grain dataflow and more. We show that FiCCO opens up a wider design space of execution schedules than possible at shard-level alone and couple this with detailed inefficiency loss characterization to provide heuristics to pick bespoke schedules. Across scenarios from real-world ML deployments, we demonstrate FiCCO delivers up to 1.6× speedup.

ACKNOWLEDGMENT

We thank our colleagues Mohamed Ibrahim, Saleel Kudchadker, Kevin Cheng, and Gowri Shankar Guttikonda for their invaluable assistance throughout the project. We also thank the anonymous IISWC reviewers for their constructive feedback in improving this work. This research was supported in part by National Science Foundation (NSF) Grant #2326894 and TACC at UT Austin. Any opinions, findings, conclusions, or recommendations are those of the authors and not of the funding agencies. AMD, the AMD Arrow logo, AMD ROCm, AMD Instinct, AMD Infinity Fabric, and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

REFERENCES

- [1] (2023) Amd Instinct™ MI300X accelerators. AMD. [Online]. Available: <https://www.amd.com/en/products/accelerators/instinct/mi300/mi300x.html>
- [2] (2025) hipblaslt. AMD. [Online]. Available: <https://github.com/ROCm/rocm-libraries>
- [3] “HIP runtime API reference: Stream memory operations,” https://rocm.docs.amd.com/projects/HIP/en/latest/doxygen/html/group__stream_m.html, Advanced Micro Devices, Inc., 2025, HIP 7.2.53211 Documentation. Accessed: May 1, 2026.
- [4] S. Aga and M. A. Ibrahim, “Comppow: A case for component-level gpu power management,” 2026. [Online]. Available: <https://arxiv.org/abs/2605.21847>

¹We avoid explicit competitive analysis in compliance with our host industry institution’s publication policy.

- [5] A. Agrawal, S. Aga, S. Pati, and M. Islam, “Conccl: Optimizing ML concurrent computation and communication with GPU DMA engines,” in *IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS 2025, Ghent, Belgium, May 11-13, 2025*. IEEE, 2025, pp. 1–11. [Online]. Available: <https://doi.org/10.1109/ISPASS64960.2025.00018>
- [6] Ali Hassani, Michael Isaev, Nic McDonald, Jie Ren, Vijay Thakkar, Haicheng Wu, and Humphrey Shi, “[Distributed GEMM: A novel CUTLASS-based implementation of Tensor Parallelism for NVLink-enabled systems,” <https://blog.shi-labs.com/distributed-gemm-88be6a481e2b>, December 2024.
- [7] “HIP: C++ Heterogeneous-Compute Interface for Portability,” <https://github.com/ROCm/HIP>, AMD, 2024.
- [8] “ROCm Communication Collectives Library (RCCL),” <https://github.com/ROCm/rccl>, AMD, 2024.
- [9] “ROCm: HIPStream,” https://rocm.docs.amd.com/projects/HIP/en/latest/reference/hip_runtime_api/modules/stream_management.html, AMD, 2024.
- [10] “ROCm/rocmBLAS: Next generation BLAS implementation for ROCm platform,” <https://github.com/ROCm/rocmBLAS>, AMD, 2024.
- [11] AMD. (2025) Hip graphs. [Online]. Available: https://rocm.docs.amd.com/projects/HIP/en/docs-develop/how-to/hip_runtime_api/hipgraph.html
- [12] L. Chang, W. Bao, Q. Hou, C. Jiang, N. Zheng, Y. Zhong, X. Zhang, Z. Song, Z. Jiang, H. Lin, X. Jin, and X. Liu, “FLUX: fast software-based communication overlap on gpus through kernel fusion,” *CoRR*, vol. abs/2406.06858, 2024. [Online]. Available: <https://doi.org/10.48550/arXiv.2406.06858>
- [13] 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,” ser. ASPLOS ’24. New York, NY, USA: Association for Computing Machinery, 2024, p. 178–191. [Online]. Available: <https://doi.org/10.1145/3620666.3651379>
- [14] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, and W. Zaremba, “Evaluating large language models trained on code,” *CoRR*, vol. abs/2107.03374, 2021. [Online]. Available: <https://arxiv.org/abs/2107.03374>
- [15] Z. Chen, S. Wang, T. Xiao, Y. Wang, S. Chen, X. Cai, J. He, and J. Wang, “Revisiting scaling laws for language models: The role of data quality and training strategies,” in *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, W. Che, J. Nabende, E. Shutova, and M. T. Pilehvar, Eds. Vienna, Austria: Association for Computational Linguistics, Jul. 2025, pp. 23 881–23 899. [Online]. Available: <https://aclanthology.org/2025.acl-long.1163/>
- [16] S. Cheng, S. Lin, L. Diao, H. Wu, S. Wang, C. Si, Z. Liu, X. Zhao, J. Du, W. Lin, and Y. You, “Concerto: Automatic communication optimization and scheduling for large-scale deep learning,” in *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1, ASPLOS 2025, Rotterdam, The Netherlands, 30 March 2025 - 3 April 2025*, L. Eeckhout, G. Smaragdakis, K. Liang, A. Sampson, M. A. Kim, and C. J. Rossbach, Eds. ACM, 2025, pp. 198–213. [Online]. Available: <https://doi.org/10.1145/3669940.3707223>
- [17] E. Choukse, B. Warrier, S. Heath, L. Belmont, A. Zhao, H. A. Khan, B. Harry, M. Kappel, R. J. Hewett, K. Datta, Y. Pei, C. Lichtenberger, J. Siegler, D. Lukofsky, Z. Kurno, G. Sahota, A. Sullivan, C. Frederick, H. Thai, R. Naughton, D. Jahnove, J. Harp, R. Carper, N. Mahalingam, S. Varkala, A. G. Kumbhare, S. Desai, V. Ramamurthy, P. Gottumukkala, G. Bhatia, K. Wildstone, L. Olariu, I. Incorvaia, A. Wetmore, P. Ram, M. Raghuraman, M. Ayna, M. Kendrick, R. Bianchini, A. Hurst, R. Zamani, X. Li, M. Petrov, G. Oden, R. Carmichael, T. Li, A. Gupta, P. Patel, N. Dattani, L. Marwong, R. Nertney, H. Kobayashi, J. Liott, M. Enev, D. Ramakrishnan, I. Buck, and J. Alben, “Power stabilization for ai training datacenters,” 2025. [Online]. Available: <https://arxiv.org/abs/2508.14318>
- [18] W. Chu, X. Xie, J. Yu, J. Wang, A. Phanishayee, C. Tang, Y. Hao, J. Huang, M. Ozdal, J. Wang, V. Goswami, N. Goyal, A. Kadian, A. Gu, C. Cai, F. Tian, X. Wang, M. Si, P. Balaji, C.-H. Chu, and J. Park, “Scaling llama 3 training with efficient parallelism strategies,” in *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, ser. ISCA ’25. New York, NY, USA: Association for Computing Machinery, 2025, p. 1703–1716. [Online]. Available: <https://doi.org/10.1145/3695053.3731410>
- [19] A. Danalis, K.-Y. Kim, L. Pollock, and M. Swany, “Transformations to parallel codes for communication-computation overlap,” in *SC ’05: Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*, 2005, pp. 58–58.
- [20] A. Danalis, L. Pollock, M. Swany, and J. Cavazos, “Mpi-aware compiler optimizations for improving communication-computation overlap,” in *Proceedings of the 23rd International Conference on Supercomputing*, ser. ICS ’09. New York, NY, USA: Association for Computing Machinery, 2009, p. 316–325. [Online]. Available: <https://doi.org/10.1145/1542275.1542321>
- [21] T. Dao, D. Y. Fu, S. Ermon, A. Rudra, and C. Ré, “Flashattention: Fast and memory-efficient exact attention with io-awareness,” in *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., 2022. [Online]. Available: http://papers.nips.cc/paper_files/paper/2022/hash/67d57c32e20fd0a7a302cb81d36e40d5-Abstract-Conference.html
- [22] DeepSeek-AI, A. Liu, B. Feng, B. Xue, B. Wang, B. Wu, C. Lu, C. Zhao, C. Deng, C. Zhang, C. Ruan, D. Dai, D. Guo, D. Yang, D. Chen, D. Ji, E. Li, F. Lin, F. Dai, F. Luo, G. Hao, G. Chen, G. Li, H. Zhang, H. Bao, H. Xu, H. Wang, H. Zhang, H. Ding, H. Xin, H. Gao, H. Li, H. Qu, J. L. Cai, J. Liang, J. Guo, J. Ni, J. Li, J. Wang, J. Chen, J. Chen, J. Yuan, J. Qiu, J. Li, J. Song, K. Dong, K. Hu, K. Gao, K. Guan, K. Huang, K. Yu, L. Wang, L. Zhang, L. Xu, L. Xia, L. Zhao, L. Wang, L. Zhang, M. Li, M. Wang, M. Zhang, M. Zhang, M. Tang, M. Li, N. Tian, P. Huang, P. Wang, P. Zhang, Q. Wang, Q. Zhu, Q. Chen, Q. Du, R. J. Chen, R. L. Jin, R. Ge, R. Zhang, R. Pan, R. Wang, R. Xu, R. Zhang, R. Chen, S. S. Li, S. Lu, S. Zhou, S. Chen, S. Wu, S. Ye, S. Ye, S. Ma, S. Wang, S. Zhou, S. Yu, S. Zhou, S. Pan, T. Wang, T. Yun, T. Pei, T. Sun, W. L. Xiao, W. Zeng, W. Zhao, W. An, W. Liu, W. Liang, W. Gao, W. Yu, W. Zhang, X. Q. Li, X. Jin, X. Wang, X. Bi, X. Liu, X. Wang, X. Shen, X. Chen, X. Zhang, X. Chen, X. Nie, X. Sun, X. Wang, X. Cheng, X. Liu, X. Xie, X. Liu, X. Yu, X. Song, X. Shan, X. Zhou, X. Yang, X. Li, X. Su, X. Lin, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. X. Zhu, Y. Zhang, Y. Xu, Y. Xu, Y. Huang, Y. Li, Y. Zhao, Y. Sun, Y. Li, Y. Wang, Y. Yu, Y. Zheng, Y. Zhang, Y. Shi, Y. Xiong, Y. He, Y. Tang, Y. Piao, Y. Wang, Y. Tan, Y. Ma, Y. Liu, Y. Guo, Y. Wu, Y. Ou, Y. Zhu, Y. Wang, Y. Gong, Y. Zou, Y. He, Y. Zha, Y. Xiong, Y. Ma, Y. Yan, Y. Luo, Y. You, Y. Liu, Y. Zhou, Z. F. Wu, Z. Z. Ren, Z. Ren, Z. Sha, Z. Fu, Z. Xu, Z. Huang, Z. Zhang, Z. Xie, Z. Zhang, Z. Hao, Z. Gou, Z. Ma, Z. Yan, Z. Shao, Z. Xu, Z. Wu, Z. Zhang, Z. Li, Z. Gu, Z. Zhu, Z. Liu, Z. Li, Z. Xie, Z. Song, Z. Gao, and Z. Pan, “Deepseek-v3 technical report,” 2025. [Online]. Available: <https://arxiv.org/abs/2412.19437>
- [23] A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Vaughan, A. Yang, A. Fan, A. Goyal, A. Hartshorn, A. Yang, A. Mitra, A. Srivankumar, A. Korenev, A. Hinsvark, A. Rao, A. Zhang, A. Rodriguez, A. Gregerson, A. Spataru, B. Roziere, B. Biron, B. Tang, B. Chern, C. Caucheteux, C. Nayak, C. Bi, C. Marra, C. McConnell, C. Keller, C. Touret, C. Wu, C. Wong, C. C. Ferrer, C. Nikolaidis, D. Allonsius, D. Song, D. Pintz, D. Livshits, D. Wyatt, D. Esiobu, D. Choudhary, D. Mahajan, D. Garcia-Olano, D. Perino, D. Hupkes, E. Lakomkin, E. AlBadawy, E. Lobanova, E. Dinan, E. M. Smith, F. Radenovic, F. Guzmán, F. Zhang, G. Synnaeve, G. Lee, G. L. Anderson, G. Thattai, G. Nail, G. Mialon, G. Pang, G. Cucurell, H. Nguyen, H. Korevaar, H. Xu, H. Touvron, I. Zarov, I. A. Ibarra, I. Kloumann, I. Misra, I. Evtimov, J. Zhang, J. Copet, J. Lee, J. Geffert, J. Vranes, J. Park, J. Mahadeokar, J. Shah, J. van der Linde, J. Billock, J. Hong, J. Lee, J. Fu, J. Chi, J. Huang, J. Liu, J. Wang, J. Yu, J. Bitton, J. Spisak, J. Park, J. Rocca, J. Johnstun, J. Saxe, J. Jia, K. V. Alwala, K. Prasad, K. Upasani, K. Plawiak, K. Li, K. Heafield, K. Stone, K. El-Arini, K. Iyer, K. Malik, K. Chiu, K. Bhalla,

- K. Lakhotia, L. Rantala-Yearly, L. van der Maaten, L. Chen, L. Tan, L. Jenkins, L. Martin, L. Madaan, L. Malo, L. Blecher, L. Landzaat, L. de Oliveira, M. Muzzi, M. Pasupuleti, M. Singh, M. Paluri, M. Kardas, M. Tsipoukelli, M. Oldham, M. Rita, M. Pavlova, M. Kambadur, M. Lewis, M. Si, M. K. Singh, M. Hassan, N. Goyal, N. Torabi, N. Bashlykov, N. Bogoychev, N. Chatterji, N. Zhang, O. Duchenne, O. Çelebi, P. Alrassy, P. Zhang, P. Li, P. Vasic, P. Weng, P. Bhargava, P. Dubal, P. Krishnan, P. S. Koura, P. Xu, Q. He, Q. Dong, R. Srinivasan, R. Ganapathy, R. Calderer, R. S. Cabral, R. Stojnic, R. Raileanu, R. Maheswari, R. Girdhar, R. Patel, R. Sauvestre, R. Polidoro, R. Sumbaly, R. Taylor, R. Silva, R. Hou, R. Wang, S. Hosseini, S. Chennabasappa, S. Singh, S. Bell, S. S. Kim, S. Edunov, S. Nie, S. Narang, S. Rappaport, S. Shen, S. Wan, S. Bhosale, S. Zhang, S. Vandenbende, S. Batra, S. Whitman, S. Sootla, S. Collot, S. Gururajan, S. Borodinsky, T. Herman, T. Fowler, T. Sheashta, T. Georgiou, T. Scialom, T. Speckbacher, T. Mihaylov, T. Xiao, U. Karn, V. Goswami, V. Gupta, V. Ramanathan, V. Kerkez, V. Gonguet, V. Do, V. Vogeti, V. Albiero, V. Petrovic, W. Chu, W. Xiong, W. Fu, W. Meers, X. Martinet, X. Wang, X. E. Tan, X. Xia, X. Xie, X. Jia, X. Wang, Y. Goldschlag, Y. Gaur, Y. Babaei, Y. Wen, Y. Song, Y. Zhang, Y. Li, Y. Mao, Z. D. Coudert, Z. Yan, Z. Chen, Z. Papakipos, A. Singh, A. Srivastava, A. Jain, A. Kelsey, A. Shajnfeld, A. Gangidi, A. Victoria, A. Goldstand, A. Menon, A. Sharma, A. Boesenberg, A. Baevski, A. Feinstein, A. Kallet, A. Sangani, A. Teo, A. Yunus, A. Lupu, A. Alvarado, A. Caples, A. Gu, A. Ho, A. Poulton, A. Ryan, A. Ramchandani, A. Dong, A. Franco, A. Goyal, A. Saraf, A. Chowdhury, A. Gabriel, A. Bharambe, A. Eisenman, A. Yazdan, B. James, B. Maurer, B. Leonhardt, B. Huang, B. Loyd, B. D. Paola, B. Paranjape, B. Liu, B. Wu, B. Ni, B. Hancock, B. Wasti, B. Spence, B. Stojkovic, B. Gamido, B. Montalvo, C. Parker, C. Burton, C. Mejia, C. Liu, C. Wang, C. Kim, C. Zhou, C. Hu, C.-H. Chu, C. Cai, C. Tindal, C. Feichtenhofer, C. Gao, D. Civin, D. Beaty, D. Kreymer, D. Li, D. Adkins, D. Xu, D. Testuggine, D. David, D. Parikh, D. Liskovich, D. Foss, D. Wang, D. Le, D. Holland, E. Dowling, E. Jamil, E. Montgomery, E. Presani, E. Hahn, E. Wood, E.-T. Le, E. Brinkman, E. Arcaute, E. Dunbar, E. Smothers, F. Sun, F. Kreuk, F. Tian, F. Kokkinos, F. Ozgenel, F. Caggioni, F. Kanayet, F. Seide, G. M. Florez, G. Schwarz, G. Badeer, G. Swee, G. Halpern, G. Herman, G. Sizov, Guangyi, Zhang, G. Lakshminarayanan, H. Inan, H. Shojanazeri, H. Zou, H. Wang, H. Zha, H. Habeeb, H. Rudolph, H. Suk, H. Aspegren, H. Goldman, H. Zhan, I. Damlaj, I. Molybog, I. Tufanov, I. Leontiadis, I.-E. Veliche, I. Gat, J. Weissman, J. Geboski, J. Kohli, J. Lam, J. Asher, J.-B. Gaya, J. Marcus, J. Tang, J. Chan, J. Zhen, J. Reizenstein, J. Teboul, J. Zhong, J. Jin, J. Yang, J. Cummings, J. Carvill, J. Shepard, J. McPhie, J. Torres, J. Ginsburg, J. Wang, K. Wu, K. H. U. K. Saxena, K. Khandelwal, K. Zand, K. Matosich, K. Veeraraghavan, K. Michelena, K. Li, K. Jagadeesh, K. Huang, K. Chawla, K. Huang, L. Chen, L. Garg, L. A. L. Silva, L. Bell, L. Zhang, L. Guo, L. Yu, L. Moshkovich, L. Wehrstedt, M. Khabza, M. Avalani, M. Bhatt, M. Mankus, M. Hasson, M. Lennie, M. Reso, M. Groshev, M. Naumov, M. Lathi, M. Keneally, M. Liu, M. L. Seltzer, M. Valko, M. Restrepo, M. Patel, M. Vyatskov, M. Samvelyan, M. Clark, M. Macey, M. Wang, M. J. Hermoso, M. Metanat, M. Rastegari, M. Bansal, N. Santhanam, N. Parks, N. White, N. Bawa, N. Singhal, N. Egebo, N. Usunier, N. Mehta, N. P. Laptev, N. Dong, N. Cheng, O. Chernoguz, O. Hart, O. Salpekar, O. Kalinli, P. Kent, P. Parekh, P. Saab, P. Balaji, P. Rittner, P. Bontrager, P. Roux, P. Dollar, P. Zvyagina, P. Ratanchandani, P. Yuvraj, Q. Liang, R. Alao, R. Rodriguez, R. Ayub, R. Murthy, R. Nayani, R. Mitra, R. Parthasarathy, R. Li, R. Hogan, R. Battey, R. Wang, R. Howes, R. Rinott, S. Mehta, S. Siby, S. J. Bondu, S. Datta, S. Chugh, S. Hunt, S. Dhillon, S. Sidorov, S. Pan, S. Mahajan, S. Verma, S. Yamamoto, S. Ramaswamy, S. Lindsay, S. Lindsay, S. Feng, S. Lin, S. C. Zha, S. Patil, S. Shankar, S. Zhang, S. Zhang, S. Wang, S. Agarwal, S. Sajuyigbe, S. Chintala, S. Max, S. Chen, S. Kehoe, S. Satterfield, S. Govindaprasad, S. Gupta, S. Deng, S. Cho, S. Virk, S. Subramanian, S. Choudhury, S. Goldman, T. Remez, T. Glaser, T. Best, T. Koehler, T. Robinson, T. Li, T. Zhang, T. Matthews, T. Chou, T. Shaked, V. Vontimitta, V. Ajayi, V. Montanez, V. Mohan, V. S. Kumar, V. Mangla, V. Ionescu, V. Poenaru, V. T. Mihailescu, V. Ivanov, W. Li, W. Wang, W. Jiang, W. Bouaziz, W. Constable, X. Tang, X. Wu, X. Wang, X. Wu, X. Gao, Y. Kleinman, Y. Chen, Y. Hu, Y. Jia, Y. Qi, Y. Li, Y. Zhang, Y. Zhang, Y. Adi, Y. Nam, Yu, Wang, Y. Zhao, Y. Hao, Y. Qian, Y. Li, Y. He, Z. Rait, Z. DeVito, Z. Rosnbrick, Z. Wen, Z. Yang, Z. Zhao, and Z. Ma, “The llama 3 herd of models,” 2024. [Online]. Available: <https://arxiv.org/abs/2407.21783>
- [24] D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, X. Bi *et al.*, “Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning,” *arXiv preprint arXiv:2501.12948*, 2025.
- [25] J. Guo, Q. Yi, J. Meng, J. Zhang, and P. Balaji, “Compiler-assisted overlapping of communication and computation in mpi applications,” in *2016 IEEE International Conference on Cluster Computing (CLUSTER)*, 2016, pp. 60–69.
- [26] K. Hamidouche, J. Bachan, P. Markthub, P.-J. Gootzen, E. Agostini, S. Jeagey, A. Shafi, G. Theodorakis, and M. G. Venkata, “Gpu-initiated networking for nccl,” 2025. [Online]. Available: <https://arxiv.org/abs/2511.15076>
- [27] B. Hanindhito, B. Patel, and L. K. John, “Bandwidth characterization of deepspeed on distributed large language model training,” in *IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS 2024, Indianapolis, IN, USA, May 5-7, 2024*. IEEE, 2024, pp. 241–256. [Online]. Available: <https://doi.org/10.1109/ISPASS61541.2024.00031>
- [28] K. Hong, X. Li, M. Liu, Q. Mao, T. Wu, Z. Huang, L. Chen, Z. Wang, Y. Zhang, Z. Zhu, G. Dai, and Y. Wang, “Efficient and adaptable overlapping for computation and communication via signaling and reordering,” 2025.
- [29] Horace He, Less Wright, Luca Wehrstedt, Tianyu Liu, Wanchao Liang, “[Distributed w/ TorchTitan] Introducing Async Tensor Parallelism in PyTorch,” <https://discuss.pytorch.org/t/distributed-w-torchtitan-introducing-async-tensor-parallelism-in-pytorch/209487>, September 2024.
- [30] Z. Hu, S. Shen, T. Bonato, S. Jeagey, C. Alexander, E. Spada, J. Dinan, J. Hammond, and T. Hoefler, “Demystifying nccl: An in-depth analysis of gpu communication protocols and algorithms,” 2025. [Online]. Available: <https://arxiv.org/abs/2507.04786>
- [31] Y. Huang, Y. Cheng, A. Bapna, O. Firat, D. Chen, M. X. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu, and Z. Chen, “Gpipe: Efficient training of giant neural networks using pipeline parallelism,” in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. B. Fox, and R. Garnett, Eds., 2019, pp. 103–112. [Online]. Available: <https://proceedings.neurips.cc/paper/2019/hash/093f65e080a295f8076b1c5722a46aa2-Abstract.html>
- [32] K. Ishizaki, H. Komatsu, and T. Nakatani, “A loop transformation algorithm for communication overlapping,” *Int. J. Parallel Program.*, vol. 28, no. 2, p. 135–154, Apr. 2000. [Online]. Available: <https://doi.org/10.1023/A:1007554715418>
- [33] 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*, ser. ASPLOS ’22. New York, NY, USA: Association for Computing Machinery, 2022, p. 402–416. [Online]. Available: <https://doi.org/10.1145/3503222.3507778>
- [34] A. Q. Jiang, A. Sablayrolles, A. Roux, A. Mensch, B. Savary, C. Bamford, D. S. Chaplot, D. de Las Casas, E. B. Hanna, F. Bressand, G. Lengyel, G. Bour, G. Lample, L. R. Lavaud, L. Saulnier, M. Lachaux, P. Stock, S. Subramanian, S. Yang, S. Antoniak, T. L. Scao, T. Gervet, T. Lavril, T. Wang, T. Lacroix, and W. E. Sayed, “Mixtral of experts,” *CoRR*, vol. abs/2401.04088, 2024. [Online]. Available: <https://doi.org/10.48550/arXiv.2401.04088>
- [35] J. Jiang, F. Wang, J. Shen, S. Kim, and S. Kim, “A survey on large language models for code generation,” *CoRR*, vol. abs/2406.00515, 2024. [Online]. Available: <https://doi.org/10.48550/arXiv.2406.00515>
- [36] Z. Jiang, J. Garrigus, A. Seigler, E. Syed, Y.-L. Huang, M. Sadi, T. Rahal-Arabi, and L. K. John, “Exploration of llm workload reliability based on di/dt effects and voltage droops,” in *2026 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2026, pp. 1–15.
- [37] V. A. Korthikanti, J. Casper, S. Lym, L. McAfee, M. Andersch, M. Shoeybi, and B. Catanzaro, “Reducing activation recomputation in large transformer models,” in *Proceedings of the Sixth Conference on Machine Learning and Systems, MLSys 2023, Miami, FL, USA, June 4-8, 2023*, D. Song, M. Carbin, and T. Chen, Eds. mlsys.org, 2023. [On-

- line]. Available: https://proceedings.mlsys.org/paper_files/paper/2023/hash/80083951326cf5b35e5100260d64ed81-Abstract-mlsys2023.html
- [38] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. Gonzalez, H. Zhang, and I. Stoica, "Efficient memory management for large language model serving with pagedattention," in *Proceedings of the 29th Symposium on Operating Systems Principles*, ser. SOSP '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 611–626. [Online]. Available: <https://doi.org/10.1145/3600006.3613165>
- [39] D. Lepikhin, H. Lee, Y. Xu, D. Chen, O. Firat, Y. Huang, M. Krikun, N. Shazeer, and Z. Chen, "Gshard: Scaling giant models with conditional computation and automatic sharding," in *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. [Online]. Available: <https://openreview.net/forum?id=qwrw7XHTmYb>
- [40] S. Li, Y. Zhao, R. Varma, O. Salpekar, P. Noordhuis, T. Li, A. Paszke, J. Smith, B. Vaughan, P. Damania, and S. Chintala, "Pytorch distributed: Experiences on accelerating data parallel training," *Proc. VLDB Endow.*, vol. 13, no. 12, pp. 3005–3018, 2020. [Online]. Available: <http://www.vldb.org/pvldb/vol13/p3005-li.pdf>
- [41] H. Liu, M. Zaharia, and P. Abbeel, "Ring attention with blockwise transformers for near-infinite context," *CoRR*, vol. abs/2310.01889, 2023. [Online]. Available: <https://doi.org/10.48550/arXiv.2310.01889>
- [42] —, "Ring attention with blockwise transformers for near-infinite context," 2023. [Online]. Available: <https://arxiv.org/abs/2310.01889>
- [43] MLCommons, "MLPerf Inference Results v5.0," https://github.com/mlcommons/inference_results_v5.0, 2025, accessed: 2025-11-16.
- [44] D. Narayanan, M. Shoeybi, J. Casper, P. LeGresley, M. Patwary, V. Korthikanti, D. Vainbrand, P. Kashinkunti, J. Bernauer, B. Catanzaro, A. Phanishayee, and M. Zaharia, "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*, ser. SC '21. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: <https://doi.org/10.1145/3458817.3476209>
- [45] M. Osama, D. Merrill, C. Cecka, M. Garland, and J. D. Owens, "Stream-k: Work-centric parallel decomposition for dense matrix-matrix multiplication on the GPU," in *Proceedings of the 28th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming, PPoPP 2023, Montreal, QC, Canada, 25 February 2023 - 1 March 2023*, M. M. Dehnavi, M. Kulkarni, and S. Krishnamoorthy, Eds. ACM, 2023, pp. 429–431. [Online]. Available: <https://doi.org/10.1145/3572848.3577479>
- [46] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," 2019. [Online]. Available: <https://arxiv.org/abs/1912.01703>
- [47] S. Pati, S. Aga, M. Islam, N. Jayasena, and M. D. Sinclair, "Tale of two cs: Computation vs. communication scaling for future transformers on future hardware," in *IEEE International Symposium on Workload Characterization, IISWC 2023, Ghent, Belgium, October 1-3, 2023*. IEEE, 2023, pp. 140–153. [Online]. Available: <https://doi.org/10.1109/IISWC59245.2023.00026>
- [48] —, "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*, ser. ASPLOS '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 1146–1164. [Online]. Available: <https://doi.org/10.1145/3620665.3640410>
- [49] S. Pati, S. Aga, M. Islam, R. Quach, S. Kudchadker, and M. A. Ibrahim, "Dma-latte: Expanding the reach of dma offloads to latency-bound ml communication," 2026. [Online]. Available: <https://arxiv.org/abs/2511.06605>
- [50] S. Pellegrini, T. Hoeffer, and T. Fahringer, "Exact dependence analysis for increased communication overlap," in *Recent Advances in the Message Passing Interface*, J. L. Träff, S. Benkner, and J. J. Dongarra, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 89–99.
- [51] J. A. Pienaar, S. Chakradhar, and A. Raghunathan, "Automatic generation of software pipelines for heterogeneous parallel systems," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '12. Washington, DC, USA: IEEE Computer Society Press, 2012.
- [52] K. Punniyamurthy, K. Hamidouche, and B. M. Beckmann, "Optimizing distributed ML communication with fused computation-collective operations," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis, SC 2024, Atlanta, GA, USA, November 17-22, 2024*. IEEE, 2024, p. 88. [Online]. Available: <https://dl.acm.org/doi/10.1109/SC41406.2024.00094>
- [53] S. Rajbhandari, C. Li, Z. Yao, M. Zhang, R. Y. Aminabadi, A. A. Awan, J. Rasley, and Y. He, "DeepSpeed-moe: Advancing mixture-of-experts inference and training to power next-generation AI scale," in *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, ser. Proceedings of Machine Learning Research, K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvári, G. Niu, and S. Sabato, Eds., vol. 162. PMLR, 2022, pp. 18 332–18 346. [Online]. Available: <https://proceedings.mlr.press/v162/rajbhandari22a.html>
- [54] 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 *48th ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2021, Virtual Event / Valencia, Spain, June 14-18, 2021*. IEEE, 2021, pp. 540–553. [Online]. Available: <https://doi.org/10.1109/ISCA52012.2021.00049>
- [55] S. Singh, O. Ruwase, A. A. Awan, S. Rajbhandari, Y. He, and A. Bhatele, "A hybrid tensor-expert-data parallelism approach to optimize mixture-of-experts training," in *Proceedings of the 37th International Conference on Supercomputing, ICS 2023, Orlando, FL, USA, June 21-23, 2023*, K. A. Gallivan, E. Gallopoulos, D. S. Nikolopoulos, and R. Beivide, Eds. ACM, 2023, pp. 203–214. [Online]. Available: <https://doi.org/10.1145/3577193.3593704>
- [56] B. Slechta, N. Comly, A. Eassa, J. DeLaere, and S. Raj. (2024, Aug) Nvidia nvlink and nvidia nvswitch supercharge large language model inference. NVIDIA Technical Blog. [Online]. Available: https://developer.nvidia.com/blog/nvidia-nvlink-and-nvidia-nvswitch-supercharge-large-language-model-inference/#nvswitch_is_critical_for_fast_multi-gpu_llm_inference
- [57] Q. Team, "Qwen3.5: Accelerating productivity with native multimodal agents," Alibaba Group, Tech. Rep., 2026.
- [58] P. Tillet, H. Kung, and D. D. 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, MAPL@PLDI 2019, Phoenix, AZ, USA, June 22, 2019*, T. Mattson, A. Muzahid, and A. Solar-Lezama, Eds. ACM, 2019, pp. 10–19. [Online]. Available: <https://doi.org/10.1145/3315508.3329973>
- [59] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, "Llama: Open and efficient foundation language models," 2023. [Online]. Available: <https://arxiv.org/abs/2302.13971>
- [60] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. C. Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao, V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. Kloumann, A. Korenev, P. S. Koura, M.-A. Lachaux, T. Lavril, J. Lee, D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Mihaylov, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, E. M. Smith, R. Subramanian, X. E. Tan, B. Tang, R. Taylor, A. Williams, J. X. Kuan, P. Xu, Z. Yan, I. Zarov, Y. Zhang, A. Fan, M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, S. Edunov, and T. Scialom, "Llama 2: Open foundation and fine-tuned chat models," 2023. [Online]. Available: <https://arxiv.org/abs/2307.09288>
- [61] P. Vaithilingam, T. Zhang, and E. L. Glassman, "Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models," in *CHI '22: CHI Conference on Human Factors in Computing Systems, New Orleans, LA, USA, 29 April 2022 - 5 May 2022, Extended Abstracts*, S. D. J. Barbosa, C. Lampe, C. Appert, and D. A. Shamma, Eds. ACM, 2022, pp. 332:1–332:7. [Online]. Available: <https://doi.org/10.1145/3491101.3519665>
- [62] G. Wang, C. Zhang, Z. Shen, A. Li, and O. Ruwase, "Domino: Eliminating communication in LLM training via generic tensor slicing and overlapping," *CoRR*, vol. abs/2409.15241, 2024. [Online]. Available: <https://doi.org/10.48550/arXiv.2409.15241>
- [63] S. Wang, J. Wei, A. Sabne, A. Davis, B. Ilbeyi, B. Hechtman, D. Chen, K. S. Murthy, M. Maggioni, Q. Zhang, S. Kumar,

- T. Guo, Y. Xu, and Z. Zhou, "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*, ser. ASPLOS 2023. New York, NY, USA: Association for Computing Machinery, 2022, p. 93–106. [Online]. Available: <https://doi.org/10.1145/3567955.3567959>
- [64] Y. Wang, H. He, and L. Wehrstedt, "Pytorch symmetricmemory: Harnessing nvlink programmability with ease," Feb 2025, pyTorch Developer Forum. [Online]. Available: <https://dev-discuss.pytorch.org/t/pytorch-symmetricmemory-harnessing-nvlink-programmability-with-ease/2798>
- [65] E. P. Xing, Q. Ho, W. Dai, J. K. Kim, J. Wei, S. Lee, X. Zheng, P. Xie, A. Kumar, and Y. Yu, "Petuum: A new platform for distributed machine learning on big data," *IEEE Trans. Big Data*, vol. 1, no. 2, pp. 49–67, 2015. [Online]. Available: <https://doi.org/10.1109/TBDATA.2015.2472014>
- [66] A. Yang, J. Yang, A. Ibrahim, X. Xie, B. Tang, G. Sizov, J. Reizenstein, J. Park, and J. Huang, "Context parallelism for scalable million-token inference," *CoRR*, vol. abs/2411.01783, 2024. [Online]. Available: <https://doi.org/10.48550/arXiv.2411.01783>
- [67] S. Zhang, N. Zheng, H. Lin, Z. Jiang, W. Bao, C. Jiang, Q. Hou, W. Cui, S. Zheng, L. Chang, Q. Chen, and X. Liu, "Comet: Fine-grained computation-communication overlapping for mixture-of-experts," *CoRR*, vol. abs/2502.19811, 2025. [Online]. Available: <https://doi.org/10.48550/arXiv.2502.19811>
- [68] Y. Zhao, A. Gu, R. Varma, L. Luo, C. Huang, M. Xu, L. Wright, H. Shojanazeri, M. Ott, S. Shleifer, A. Desmaison, C. Balioglu, P. Damania, B. Nguyen, G. Chauhan, Y. Hao, A. Mathews, and S. Li, "Pytorch FSDP: experiences on scaling fully sharded data parallel," *Proc. VLDB Endow.*, vol. 16, no. 12, pp. 3848–3860, 2023. [Online]. Available: <https://www.vldb.org/pvldb/vol16/p3848-huang.pdf>
- [69] S. Zheng, W. Bao, Q. Hou, X. Zheng, J. Fang, C. Huang, T. Li, H. Duanmu, R. Chen, R. Xu, Y. Guo, N. Zheng, Z. Jiang, X. Di, D. Wang, J. Ye, H. Lin, L.-W. Chang, L. Lu, Y. Liang, J. Zhai, and X. Liu, "Triton-distributed: Programming overlapping kernels on distributed ai systems with the triton compiler," 2025. [Online]. Available: <https://arxiv.org/abs/2504.19442>
- [70] S. Zheng, J. Fang, X. Zheng, Q. Hou, W. Bao, N. Zheng, Z. Jiang, D. Wang, J. Ye, H. Lin, L.-W. Chang, and X. Liu, "Tilelink: Generating efficient compute-communication overlapping kernels using tile-centric primitives," in *Eighth Conference on Machine Learning and Systems*, 2025. [Online]. Available: <https://openreview.net/forum?id=ccjvBkTRRe>