Hiding Communication Cost in Distributed LLM Training via Micro-batch Co-execution

Haiquan Wang*
University of Science and Technology of China
whq1516@mail.ustc.edu.cn

Chaoyi Ruan*
University of Science and Technology of China, Mohamed bin Zayed University of Artificial Intelligence
rcy@mail.ustc.edu.cn

Jia He
University of Science and Technology of China, Mohamed bin Zayed University of Artificial Intelligence
hej148@mail.ustc.edu.cn

Jiaqi Ruan
University of Science and Technology of China, Mohamed bin Zayed University of Artificial Intelligence
jiaqiruan@mail.ustc.edu.cn

Chengjie Tang
Shanxi University
tangcj@sxu.edu.cn

Xiaosong Ma
Mohamed bin Zayed University of Artificial Intelligence
xiaosong.ma@mbzuai.ac.ae

Cheng Li
University of Science and Technology of China
chengli7@ustc.edu.cn

Abstract

The growth of Large Language Models (LLMs) has necessitated large-scale distributed training. Highly optimized frameworks, however, suffer significant losses in MFU (Model FLOPS Utilization) due to communication.

This paper introduces DHelix, a novel micro-structure inspired by DNA that significantly enhances the efficiency of LLM training. Central to DHelix is *Strand Interleaving (SI)*, which treats the continuous stream of training micro-batches on a GPU as two interleaved *strands*. DHelix co-schedules their forward and backward passes using operator-level overlap profiling and a dynamic programming-based search. It enables the two strands to share model states and activation memory, requiring $< 3\%$ additional HBM space. With its unique *model folding* design, DHelix seamlessly integrates with all forms of data and model parallelism, including the challenging pipeline parallelism (with a W-shaped pipeline).

We evaluate DHelix training with the popular Llama and GPT dense models, plus the Phi Mixture of Expert (MoE) model, across 3 GPU clusters (A40, A800, and H100). Results show that it achieves 12-40% throughput (up to 58% MFU) and 2-29% throughput (up to 71% MFU) improvement on the 64-card A40 and A800 clusters respectively, significantly outperforming state-of-the-art methods. On the 32-card H100 cluster, though the faster network reduces DHelix's profit margin, it makes cross-node tensor parallelism promising, a practice currently prohibitive due to communication costs.

*Haiquan Wang and Chaoyi Ruan equally contributed to this work. This work was done when Chaoyi Ruan, Jia He and Jiaqi Ruan were visiting students at MBZUAI. .

1 Introduction

Recent advancements in Generative AI, particularly in areas such as chatbots [4] and text generation [5, 15, 35], have driven a significant trend in Large Language Model (LLM) training. These LLMs, exemplified by models like Llama [15] and GPT [5], are predominantly based on transformer architectures. As model sizes escalate from billions to trillions of parameters, distributed model training has become indispensable. However, current training methods for LLMs face a critical challenge: overall GPU throughput remains suboptimal, with end-to-end Model FLOPS Utilization (MFU) falling below 50% [21], wasting precious GPU cluster resources.

A primary limiting factor performance is the bottleneck created by *intra-layer communication*. Such communication emerges from various parallelism strategies, including Tensor Parallelism (TP) [48], Sequence Parallelism (SP) [22], Context Parallelism (CP) [30, 37], and Expert Parallelism (EP) [17, 29]. They are crucial for mitigating the rapidly increasing activation memory consumption associated with growing model sizes and input parameters (sequence length and batch size). While effectively distributing computational tasks across multiple devices, these strategies introduce numerous communication operators (e.g., `AllGather` [40] and `ReduceScatter` [41]) into the critical training execution path. As the LLM layer size increases and the number of parallelism methods grows, such communication consumes considerable portions of the total execution time (Figure 2) and delays subsequent computation.

To mitigate such communication overhead, overlapping communication with computation becomes ever more crucial

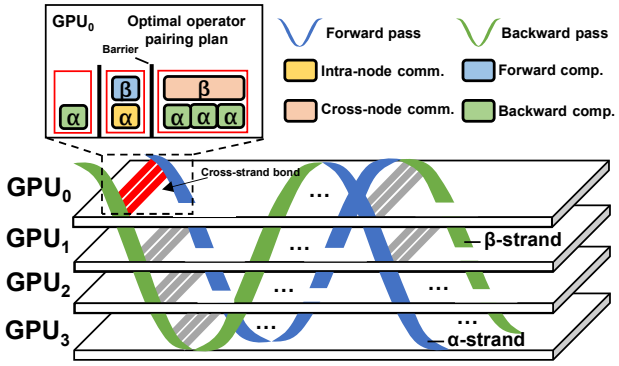

Figure 1. Double-strand execution in DHelix on 4 GPUs

in large-scale LLM training. Previous efforts on *intra-batch optimization* [6, 19, 21, 53] have aimed to overlap computation and communication within a single micro-batch by breaking down computation operators into smaller units and then overlapping the subsequent communications. However, it has two significant drawbacks: (1) limited overlapping potential due to data dependencies within a micro-batch, (2) degraded computational efficiency due to splitting well-optimized computation operators.

The *inter-batch* techniques [12, 51], in contrast, enable the concurrent execution of two micro-batches, allowing communication operators to overlap with computation operators across micro-batches. However, as to be detailed later, existing approaches have fundamental limitations that hamper the application of inter-batch co-execution to frameworks using pipeline parallelism (PP), a major mechanism for scaling out LLM training. In addition, prior solutions, including the DualPipe technique used in the very recent DeepSeek [12], pose high memory requirement due to model replication and adopt coarse-granule interleaving between two concurrent micro-batches, scheduling tasks between them in a hand-customized or round-robin manner.

In this research, we start by comprehensively analyzing the trends in intra-layer communication growth and conduct operator-level profiling to study their pairwise performance behavior when overlapped. Our results reveal that though communication sizes continue to grow, there are plenty of opportunities so far unexplored by existing frameworks.

Based on these results and inspired by the DNA structure, we propose DHelix ("Double-Helix LLM" training), a novel micro-structure that significantly improves the efficiency of LLM training. The main idea in DHelix's design is *Strand Interleaving (SI)*, which views the continuous stream of micro-batches through a GPU as two *strands*, α-strand and β-strand, to be co-executed as illustrated in Figure 1. DHelix juxtaposes their forward (blue) and backward (green) passes together, which *share model states* and have complementary memory space usage for activation data.

Within each GPU (top-left zoom-in window in Figure 1), the operators that we profiled earlier now form a core set of *operator bases*, computation-intensive or communication-intensive. These operators (colored blocks in two rows, belonging to the α-strand and β-strand respectively) are to be "paired" with cross-strand "bonds", like nucleotide bases on the DNA strands. DHelix systematically searches for an optimal SI plan that co-schedules the operators from the opposite strands as allowed by the dependency within each strand. The search is based on its operator-level overlap profiling results and assisted by a dynamic-programming algorithm. Segments of these paired operators (within the red boxes) are co-scheduled by DHelix-inserted cross-strand barriers, with their actual co-execution administered via three separate CUDA streams performing computation, intra-node communication, and cross-node communication, respectively.

However, it is challenging to co-locate the two strands going in opposite directions (forward and backward) in one GPU in the first place, with the prevalent pipeline parallelism (PP) adopted today. Earlier inter-batch interleaving techniques, such as Wavelet [51], simply cannot run with PP. To this end, we introduce a novel *model folding* technique that transforms the double-strand structure into an abstract U-shaped model. Under PP, this folding technique leads to a W-shaped pipeline schedule that significantly reduces memory overhead by allowing both strands to reuse the same parameter set on each GPU instead of requiring model replication.

We have implemented and open-sourced DHelix [13] on top of NVIDIA's state-of-the-art Megatron-LM[38] distributed LLM training framework. Our comprehensive evaluation used popular models such as Llama, GPT, and Phi (under both standard and long-sequence settings). Results show significant improvement in overall training throughput across three NVIDIA GPU clusters: up to 40% on a 64-card A40 cluster and up to 29% on a 64-card A800 one, also considerably outperforming existing optimizations that we implemented following literature. On the H100 cluster, though the fast interconnection reduces our improvement margin, DHelix makes cross-node tensor parallelism promising, a practice currently prohibitive due to communication costs. Regarding memory efficiency, DHelix supports a maximum model size of up to 97.5% of the ideal Megatron-LM single-strand limit, while providing up to 39% performance improvement and achieves up to 71% MFU.

2 Background

2.1 Distributed LLM training

In LLM training with trillions of data, input tokens are split into global or micro batches and fed into GPUs one by one. GPUs process micro-batches, accumulate gradients, and synchronize them after completing a global batch for model updates. From the perspective of a single GPU, it alternates

between forward and backward passes unless idle. The computation forms a virtual execution stream, resembling a single strand of micro-batches processing.

To meet the demands of increasing model sizes, training jobs are distributed across GPU clusters, adopting hybrid parallelism strategies [28, 38, 61] that combine data and model parallelism. In **Data Parallelism (DP)** [46, 47], the model is replicated across GPUs/nodes, with each replica processing a subset of the training data. Gradients are synchronized after each iteration. However, due to memory limitations, DP must be combined with model parallelism.

Model Parallelism can be categorized into *Inter-layer parallelism* and *Intra-layer parallelism*. The inter-layer parallelim, **Pipeline Parallelism (PP)** [18, 33, 38] splits model layers into stages, which are assigned to different GPUs/nodes, and follows a one-forward one-backward (1F1B) execution pattern. Among intra-layer parallelisms, **Tensor Parallelism (TP)** [38] splits tensors along dimensions like hidden size (e.g., partitioning `GEMM`). **Sequence Parallelism (SP)** [22] complements TP by further distributing remaining operations like `LayerNorm` and `Dropout` by partitioning the input sequences across GPUs. **Context Parallelism (CP)** [30] processes the token sequence in parallel, invoking `AllGather` communication to collect the key/value tensors from neighboring GPUs for long sequences (*e.g.*, 16K). For sparse models such as Mixture of Experts (MoE), distinct GPUs are responsible for processing different MLP components (referred to as experts) under the **Expert Parallelism (EP)** [17, 58].

2.2 Communication Bottleneck

Distributed training inevitably incur communication like *collective communication*, blocking the GPU computation and hindering efficiency and scalability.

In the context of distributed LLM training, intra-layer parallelism involves collective communication, whereas inter-layer parallelism (PP) transfers data between adjacent stages. For instance, TP involves data aggregation using `AllReduce`, which is transformed to `AllGather` and `ReduceScatter` when SP is incorporated. The addition of CP further adds more `AllGather` operators to collect key/value tensors. Finally, MoE introduces `All-to-All` operators, with tokens shuffled by router gates between GPU experts.

We analyze the communication overhead using the state-of-the-art MegatronLM framework on a 64-card A40 GPU cluster. Figure 2 gives a sample distribution of the total pre-training time among computation and three types of communication operators for multiple transformer types and size combinations. Communication already becomes time-dominant at such a scale, especially with the larger models. The major contributors are the collective communication operators brought by model parallelism, namely TP/SP, CP, and EP. DP and PP, on the other hand, incur much lower communication volumes. For instance, with the Llama-39B model,

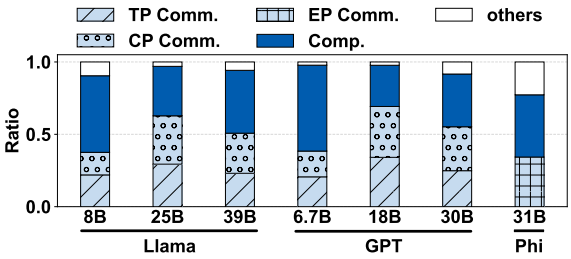

Figure 2. Sample execution time breakdown in training different transformer-based models and parameter sizes

TP- and CP-incurred communication occupies 55% of the execution time; with Phi-31B, EP-incurred communication occupies 34.3%.

Table 1. Cross-node vs. intra-node communication time/volume of Llama3.1-405B with hybrid parallelism

GPUs	8192	16384	
Parallelism	DP:64, TP:8, PP:16	DP:128, TP:8, PP:16	DP:8, TP:8, PP:16, CP:16
Intra-node	0.98 s / 441 GB	0.3 s / 220 GB	7.84 s / 3528 GB
Cross-node	0.17 s / 8.6 GB	0.15 s / 7.7 GB	7.52 s / 376 GB

As larger models require more GPUs for training, the proportion of cross-node communication will inevitably increase. At the same time, intra-node communication becomes faster with higher inter-GPU bandwidth within a machine. These two factors make the issues of cross-node communication more prominent compared to intra-node communication.

To understand this, we estimated the communication distribution between intra- and cross-node communication for the LLM training on clusters using 10000+ GPUs. More specifically, we follow the distributed training configurations and model architecture reported for Llama3.1 405B [15], utilizing H100 GPU [36] connected by NVLINK (900 GB/s) and InfiniBand (3200 Gbps) as network setup. We calculate the cross-node communication time based on communication volume and network bandwidth. Table 1 shows that the cross-node over total communication ratio in time increases from 14.78% to 33.33% when moving from 8192 to 16384 nodes by simply scaling the DP group size. However, if CP is enabled, the cross-node communication ratio is further increased to 48.96% as more cross-node Send/Recv is incurred. This indicates that communication remains a significant overhead during LLM training despite faster networks.

2.3 Computation-communication Overlap

Computation-communication overlapping is a common optimization in model parallelism, such as TP and PP. Existing methods [7, 8, 19, 21, 25, 53] decompose computation and communication into fine-grained tasks for interleaving. For instance, MegaScale [21] partitions large `AllGather` and subsequent `GEMM`/`Dgrad` operators into smaller `Send/Recv`

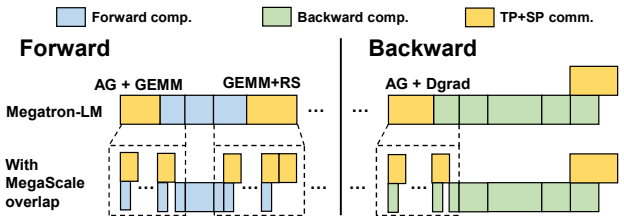

Figure 3. Sample result operator overlapping schedule by methods proposed in MegaScale [21], captured using the NVIDIA nsight profiling tool [43], in comparison to the execution follow achieved by Megatron-LM (top)

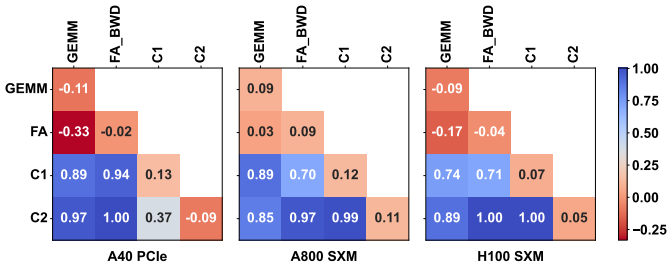

Figure 4. The overlap effectiveness is achieved through operator overlap. C1 represents intra-node AllGather, while C2 denotes cross-node All-to-All. All operators, except for C2, are derived from the Llama 70B.

tasks and matrix tiles, overlapping them within a micro-batch when no data dependence exists. Similarly, Megatron-LM [38] and Ring-attention [30] replace `AllGather` with `Send/Recv` to overlap tiled attention computation. PP optimizations [24, 62] focus on overlapping pipeline communication (e.g., tensor transmission) with computation to reduce pipeline bubbles.

Despite these efforts, the pattern of single micro-batch processing still restricts overlap opportunities due to data dependence. Our implementation of MegaScale in Megatron-LM shows that intra-layer communication only overlaps adjacent `GEMM`, leaving 73.9% of execution un-overlapped. As shown in Figure 3, `AllGather` is not fully overlapped due to `GEMM`'s short lifespan, and much backward computation lacks communication to overlap with.

Fortunately, the concept of *micro-batch co-scheduling*, first introduced by Wavelet [51], opens up new opportunities for optimization. By allowing two micro-batches to simultaneously execute independently, one micro-batch's computation operators may overlap with another micro-batch's communication ones. This significantly enhances the chance of hiding the communication cost. However, such naive co-execution suffers three key limitations: (1) incompatibility with pipeline parallelism (PP), (2) model replication, (3) coarse-granule interleaving, as to be detailed next in Section 3.

3 Motivation and Approach Overview

3.1 LLM Training Operator Overlap

Though, as mentioned earlier, existing work has eagerly enabled the overlap between computation and communication activities to hide the cost of the latter [6, 7, 21, 50, 52], there lacks systematic evaluation of the performance behavior in overlapping commonly used LLM training operators.

In this work, we performed extensive profiling to understand the performance impact when we co-schedule two operators on the same GPU, with complete pairwise benchmarking, across multiple GPU types. Here we show results from 4 representative operators in Figure 4, two computation-intensive and two communication-intensive: `GEMM`, `FA` (Flash Attention [9, 10], a highly optimized attention implementation), `C1` (node-local `AllGather`), and `C2` (cross-node `All-to-All`).

We measure the operator-level overlap behavior using a *Overlap Effectiveness Factor (OEF)*, defined as

$$OEF_{i,j} = (T_i + T_j - P_{i,j})/min(T_i, T_j) \quad (1)$$

where T_i is the sequential execution time of a single operator op_i, while $P_{i,j}$ is the overlapped execution time of op_i and op_j. In other words, OEF measures how much the overlap could hide the shorter operator's execution.

Figure 4 gives the pairwise OEF across three NVIDIA GPU platforms: A40 (PCIe connection among GPUs on the same node), as well as A800 SXM and H100 SXM (both with NVLink for intra-node, cross-node communication). The major takeaways are:

- Computation-intensive operators (`GEMM`, `FA`) do not overlap well in LLM.
- The `FA` operator is particularly unforgiving when overlapped with `GEMM`, likely due to the interference from the latter that breaks `FA`'s carefully choreographed fine-granule interleaving between computation and memory I/O.
- Both types of communication operators overlap well with both types of computation ones, though the degree of communication hiding achieved varies.
- Neither node-local nor cross-node communication operators overlap well with themselves, but benefit from overlapping with each other due to simultaneous use of different network resources (NVlink and Infiniband, for example).

These results inspire us to build an inter-batch execution interleaving scheme with systematic, fine-granule overlap optimization performed at the operator level. They also indicate that cross-node communication may be hidden well by other operators. This allows for future model size scaling by relaxing the TP scaling constraint, currently limited to 8 (only among GPUs within the same node) in production model training due to its large communication volume.

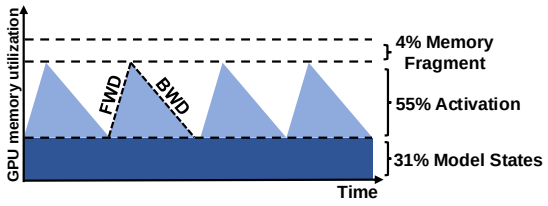

Figure 5. Sample memory allocation breakdown in training Llama-25B model, with 8192 sequence length and micro-batch size 1, on 64 A40 GPUs with parallelism strategy of DP=8 and TP=8.

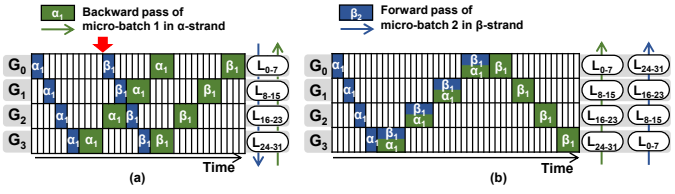

Figure 6. Pipeline scheduling with two strands: (a) V-shape from 1F1B and (b) Bi-direction

3.2 Strand Interleaving Overview

DHelix performs systematic interleaving at the operator level to accommodate two concurrent strands, α-strand and β-strand, each processing one micro-batch, for maximizing GPU utilization.

Note that the common practice today, processing a single strand, already maximizes the use of GPU memory by increasing the micro-batch size as much as possible for maximum training throughput. Figure 5 illustrates the memory usage profile of a sample training run for a few micro-batches. Given the model size and parallel training parameters (Llama 25B, DP=8, TP=8), users typically bump up the micro-batch size (to 1), stopping right before the system runs out of memory.

The only way to squeeze in two strands for SI would be to introduce a time lag between them so that α-strand's forward pass and β-strand's backward pass are co-scheduled, or vice versa, as shown in Figure 1. The reason is that their execution has complementary memory consumption patterns: while the strand running the forward pass steadily allocates memory space for activation data as it advances through layers, the one backward releases it. By fitting the two activation data "triangles" together, the total activation memory footprint stays close to the peak value from either single strand.

As mentioned in Section 2.3, micro-batch co-scheduling (explored first by Wavelet [51] and more recently DualPipe [12]) allows frameworks to hide communication overhead. However, existing systems fall short when applied to modern distributed LLM training setups for several reasons:

- *Incompatibility with PP.* The latter has become essential for scaling LLM training, especially with large-scale training jobs, due to the relatively loosely coupled computation and lower overall communication volume (only across the boundary of pipeline stages). However, with PP, neighboring micro-batches' forward and backward passes (the "Tick" and "Tock" waves in Wavelet) move in opposite directions. As marked in Figure 6(a) with a red arrow, when the forward (blue) pass of β-strand starts at GPU G_0, the backward (green) pass of α-strand starts at GPU G_3. The two strands cross each other only once during the pass, leaving little opportunity for co-execution on the same GPU.
- *Model replication.* One solution to this incompatibility is to replicate model states (*i.e.*, parameters, gradients, and optimizer states), enabling the bi-directional pipeline shown in Figure 6(b). This approach, adopted by DualPipe, allows α-strand and β-strand to move in the same direction, from G_3 to G_0. Unfortunately, with typical distributed training settings, model states occupy a considerable portion of GPU memory (over 30% in the sample profiling result in Figure 5). Storing two copies of the model would significantly forfeit the profit of micro-batch interleaving.
- *Coarse-granule interleaving.* With existing approaches, two micro-batches are co-scheduled in a communication-oblivious or coarse-granule manner. More specifically, Wavelet co-schedules its Tick and Tock waves to follow their original sequential workflow without operator reorganization to exploit overlap opportunities provided by communication activities. DualPipe divides the forward and backward processes into fixed components based on their specific MoE execution schedule: `Attention`, `All-to-All Dispatch`, `MLP`, and `All-to-All Combine`, enabling a manually fixed co-execution plan to aggressively hide all-to-all communication.

DHelix eliminates the above limitations with its general-purpose SI mechanism, which unlocks the cycle-efficient and memory-save co-scheduling of two micro-batches' processing on each GPU, regardless of the training parallelism or framework adoption. To help picture DHelix's working, imagine the double-helix DNA structure in a 2D space, whose two strands couple into a single stream of training computation, processing two micro-batches together.

The coupled "DNA strands", as a whole, get folded into a U-shape, with model layers doubling back across the GPUs participating in a PP pipeline. With this arrangement, we solve *both* the PP incompatibility and the model replication problems listed above. As described in more detail in Section 4.1, the β-strand forward and α-strand backward passes always move in the same direction in their lifetime, allowing their perfect co-execution from one device to the next in the PP scheme. Meanwhile, the U-shaped folding allows the model layers resident on each GPU to be shared by α-strand and β-strand, simultaneously serving forward and backward passes with a single copy of model parameters.

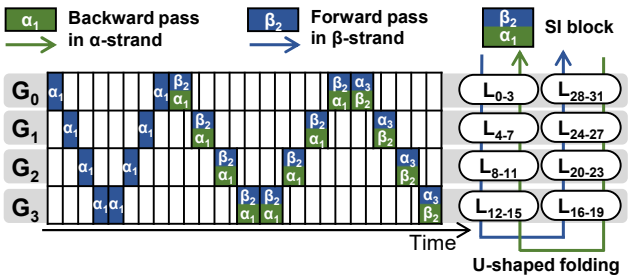

Figure 7. SI with model folding (right side) in DHelix and the resulted W-shaped scheduling (left side), in comparison to Figure 6

Within the coupling of β-strand and α-strand, the two strands move up and down opposite each other, alternating between forward and backward passes. Utilizing offline profiling results measuring inter-operator overlap compatibility, DHelix searches for efficient co-scheduling of the coupled strands using dynamic programming (to be discussed in Section 4.2). In each pass, the strands follow the search result to rearrange their operators and co-schedule compatible operators that overlap well (*e.g.*, computation with communication), producing a double-strand execution anchored by these coupling points similar to the base pairs in DNA strands.

As a result, DHelix's SI design enhances GPU utilization by enabling the training pathway to accommodate two neighboring micro-batches simultaneously, effectively hiding communication overhead in the critical path of LLM training, significantly boosting overall performance. Meanwhile, SI operates below the existing levels of parallelism, seamlessly integrating with DP, TP, SP, CP, and EP. Our U-shaped folding technique further enables its compatibility with PP, where these forms of parallelism act as a basic GPU mesh for complete model training.

4 DHelix Design

4.1 Model Folding

Here, we describe *model folding*. This key DHelix technique enables SI to work with pipeline parallelism, where we fold the original linear layout of model layers across GPUs to a U-shaped one. The right side of Figure 7 gives the layout of a 32-layer LLM after folding. The number of layers hosted per GPU remains at 8, but rather than hosting the eight consecutive layers L_0-L_7, G_0 now hosts two segments, L_0-L_3 plus L_{28}-L_{31}.

The difference is that now the flow of the two strands across the GPUs are always heading the same way instead of opposite. The α-strand backward (green) and β-strand forward (blue) passes start from G_0, reach G_3, and then return to G_0. In other words, as shown by the left side of Figure 7, the familiar "V" shape of 1F1B becomes a "W" shape, where the forward and backward passes each make an identical "V" shape, allowing α-strand and β-strand to perfectly overlap with each other in moving across the GPUs and their forward-backward passes to be co-scheduled on each GPU.

Compared with the model replication method discussed earlier (Figure 6), DHelix's model folding does not change the model parameter size per GPU. Therefore, with SI, it accommodates both strands with the same copy of the model parameter, effectively processing two micro-batches on each GPU, while consuming almost the same GPU memory capacity as the state-of-the-art distributed training frameworks in processing one strand.

Figure 7 gives a simplified rendering of the double-strand interleaved pass schedule, with the blue and green blocks possessing the same width (in time). It is well known that the backward passes are slower, with the green blocks almost twice as wide as the blue ones. When SI co-schedules the α-strand and β-strand operators on the same GPU, one might wonder if α-strand's backward memory freeing could not keep up with β-strand forward memory requests and a space-induced dependency between the two strands would be created. In reality, we found it to be a non-issue, and the difference would at most require a few hundred MBs of additional idle memory, as we overlap the forward and backward passes at the layer granularity instead of the whole model.

To give a more comprehensive picture, Figure 8 illustrates the complete W-shaped pipeline schedule with 12 micro-batches. It progresses through the same three phases as the classical 1F1B pipeline: *warmup*, *steady*, and *cooldown*. In the warmup phase, DHelix populates the pipeline with α_1-α_4 (blue blocks) from α-strand. Once they complete their forward passes, DHelix injects β_5-β_8 from β-strand, beginning prefilling the pipeline via SI blocks (top blue and bottom green), with the backward passes in α-strand and the forward passes in β-strand "fused" together. Upon the issuing of the SI block $\begin{bmatrix}\beta_5\\ \alpha_1\end{bmatrix}$, DHelix moves into the steady phase. Here DHelix launches α_9-α_{12} in α-strand and SI blocks occupy all GPUs. Finally, during the cooldown phase, DHelix runs out of forward tasks to create SI blocks and falls back to backward blocks (green) as the pipeline empties.

Both the original 1F1B and DHelix's W-shaped schedule enable fully occupied GPUs during the steady phases. Let p be the number of pipeline stages, m the number of micro-batches, and s the speedup of the SI block compared with the sum of execution time of forward block and backward block. In other words, each of the SI blocks $\begin{bmatrix}\beta_i\\ \alpha_j\end{bmatrix}$ takes $\frac{1}{s}$ of the time to complete than their sequential execution $\beta_i + \alpha_j$. The bubble ratios of DHelix and 1F1B are $\frac{p-1}{m-1+p\times s}$ and $\frac{p-1}{m-1+p}$, respectively. Typically, the value of s ranges from 1.2 to 1.4, which results in the bubble ratio of DHelix being slightly lower than that of 1F1B.

The performance benefit of DHelix mainly comes from enabling pipeline execution with SI in the first place (unlike

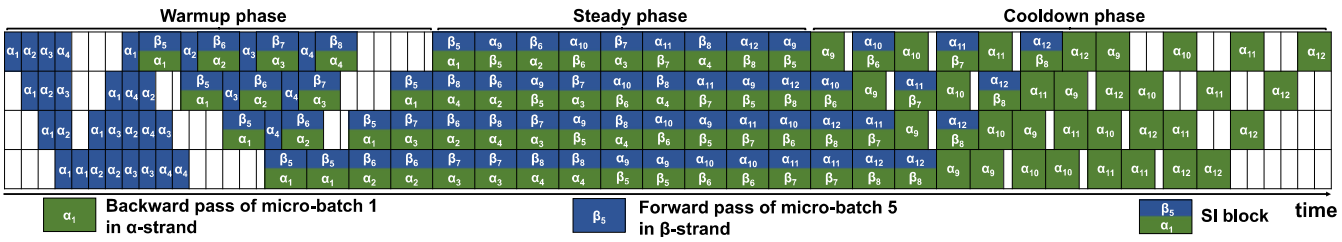

Figure 8. Sample W-shaped pipeline schedule in DHelix

Wavelet), then shortening the operator execution time by overlapping the two strands.[1]

Also, as shown in the figure, the W-shaped pipeline requires two down-up trips for each micro-batch, doubling the `Send/Recv` communication volume from the original 1F1B scheduling. However, the share of volume attributed to such pipeline `Send/Recv` is minimal in common distributed LLM training and has little performance implication.

4.2 Strand Coupling with Operator Pairing

Now, we take a closer look at a coupled double-strand forward+backward pass (one SI block in Figure 7). While the U-shaped model folding enables their co-execution, the operator-level overlap performed next brings DHelix its major performance gain.

The gain comes from aggressively harvesting hardware parallelism whenever possible in co-executing the two strands. As mentioned earlier, the individual operators in today's LLM training workflows, performing computation or communication, have been intensively optimized, enabling the overlap between these two activities [6, 21]. As a direct consequence, the current operators have little chance to further overlap within each strand due to data dependence that forces their sequential scheduling.

However, SI opens up new space for overlapping α-strand's communication operators with activities of β-strand, and vice versa, as there is no data dependence between the α-strand and the β-strand. Here, in addition to the memory benefit of interleaving the forward and backward passes of the two strands, there are two more performance advantages:

- The forward and backward passes have different operator layouts, leaving more space for co-scheduling computation with communication operators (as compared to forward-forward or backward-backward interleaving).
- The backward pass tends to be more computation-intensive, in certain cases, providing more opportunities to hide the higher ratio of communication activities in the forward one.

Rather than simply unleashing two micro-batches and letting the GPU do its best-effort co-scheduling (as with Wavelet [51]), DHelix carefully adopts a systematic and adaptive approach to intentionally align the two strands' execution at the operator level, to overlap computation and communication operators between two strands as possible. Figure 9 illustrates its overall workflow: (1) generating all possible *operator sequences* based on the appropriate DAG, (2) generating *operator segments* by partitioning a pair of forward/backward operator sequences into contiguous pieces, and (3) using a dynamic programming algorithm to search for the optimal SI pairing scheme, which are administered by inserting barriers during the two strands' co-execution. Below we discuss these steps in more detail.

Table 2. Major operator bases in transformer workflow

Computation (comp)		Communication (comm)
GEMM	Fused BDA	AllGather
FlashAttention	Layernorm	ReduceScatter
Group-GEMM	Router	AlltoAll
	Permute	Send/Recv

Operator Base Characterization. Again, one could return to the DNA structure analogy, where the operators "pair" with appropriate peers in the opposite strand to form a "bond", in this case, co-execution that offers significant performance gain by utilizing heterogeneous GPU resources. Unlike the DNA strands with four types of nucleobases, here conceptually, one could picture the transformer operators in only two types: computation or communication, as listed in Table 2.

Conceptually, the "base pairing" happens only between profitable base types: comp-comm or comm-comm, as seen in Section 3.1. In DHelix design, to account for the intricate interleaving behavior between operators, we exhaustively perform pair-wise measurement of the operators listed above in Table 2. Such offline pre-profiling must be performed on a new hardware setup or when the training workflow is modified (such as updated operator implementation), and takes around 10-30 min.

Operator Sequencing and Partitioning. DHelix has no control over the CUDA scheduling of individual operators. What it can do, however, is to throw barriers strategically across the execution of the α-strand and the β-strand, forcing their synchronization. This way, communication operators could be co-executed with peers from the other strand,

[1] In practice, with large models that fit only one pair of strands and m typically over 100, pipelines have long steady phases and low bubble ratios.

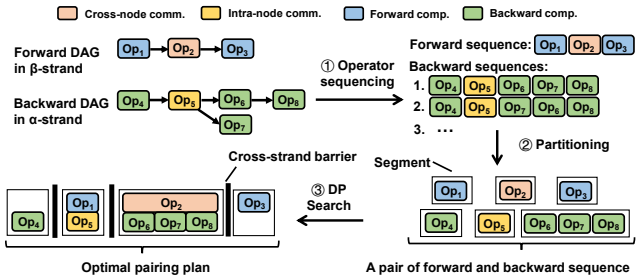

Figure 9. Strand coupling workflow

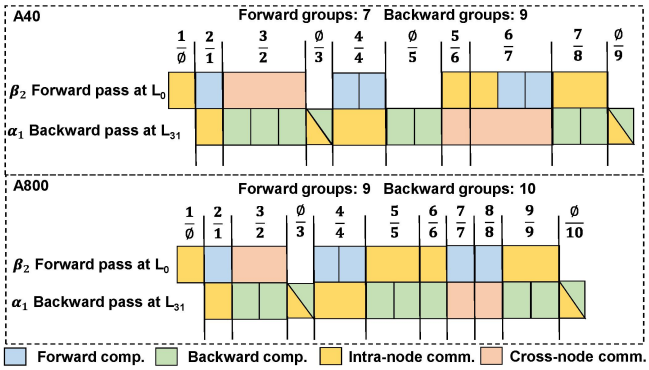

Figure 10. SI pairing results on different hardware. Here the dual-color blocks note computation/communication operators in the backward pass that can be overlapped (no inter-dependency). Such "self-overlapping" operators within a strand is considered a single operator in DHelix's pairing search.

hoping to maximize the chance of their cost being hidden by computation. In other words, the linear operator execution sequence of a single strand is *partitioned* into segments, which are then *paired* across strands, both via the DHelix injected barriers.

DHelix starts the strand pairing process by constructing the DAG for *computing a single layer in the forward and backward passes*, composing 14 and 18 operators for one transformer layer if enabling tensor/sequence parallelism. For brevity, in the rest of our description, we often use terms like "forward sequence", though the DAG and resulting sequences are from one layer of the pass. Section 4.3 will also discuss the current and future expansion of the pairing and search scope. Based on these DAGs, as shown in Figure 9, DHelix firstly generates the forward/backward candidate operator sequences by enumerating their topological orderings.

Naturally, the next step would be to partition each pair of candidate sequences into *operator segments* (with barriers inserted in between), and co-schedule the segments across the two sequences (strands). Given a pair of candidate forward-backward sequences, positioning the cross-strand barriers effectively generates a *candidate pairing plan*.

These two DAGs are simple with current transformer workflows, with a few operators that could move around in the result sequences. These include the backward weight gradient (`wgrad`, performing `GEMM`) and the `router` computation in MoE. Still, the number of candidate sequences and their partitioning already result in a search space too large to be explored manually. Fortunately, the search for an optimal pairing plan can be formulated as a dynamic programming problem, as outlined below.

Pairing by Dynamic Programming. Given a pair of forward and backward pass candidate operator sequences, S_f and S_b, each partitioned into N_f and N_b segments, DHelix searches for the optimal operator pairing plan that yields the shortest total execution make-span $T_{opt}(N_f, N_b)$.

Intuitively, a pairing sub-solution for a pair of prefix sequences in the optimal solution, containing i ($0 \le i < N_f$) and j ($0 \le j < N_b$) operator segments from S_f and S_b, must also be optimal:

$$T_{opt}\ (i,j) = \min \left\{ \begin{array}{c} T_{opt}(i-1,j) + P(i,\emptyset), \\ T_{opt}(i,j-1) + P(\emptyset,j), \\ T_{opt}(i-1,j-1) + P(i,j) \end{array} \right\} \qquad (2)$$

Here $P(i, j)$ gives the overlapped execution time of overlapping the ith operator segment from the forward sequence with the jth operator segment from the backward one, calculated from the offline profiling results. $P(i, \emptyset)$ gives the ith operator segment's solo execution time, as it is scheduled to run alone.

4.3 Discussion

Adaptivity. Compared with DualPipe, which generates a *fixed strategy designed specifically* for MoE models under a parallelism strategy combining DP, EP, and PP, DHelix's SI scheme is more general, as can be seen from the system design descriptions given in Section 4.1 and 4.2. It is independent of the specific distributed training framework or underlying hardware. By performing offline profiling and re-optimizing the strand interleaving plan, it can adapt to new training frameworks or hardware platforms.

Figure 10 demonstrates this with the search result of the same training workflow on two different platforms, on NVIDIA A40 (40 GB) and A800 (80 GB) clusters, respectively, showing quite different SI plans. Though we do not have space here, it needs to be pointed out that the pairing schedule would also vary on the same hardware when one changes distributed training parameters (group size for TP, PP, SP, etc.), as the computation and communication operators' "weight" would change.

Both AI model training frameworks and the underlying accelerator architectures are experiencing steady iterations of improvement in the years and even decades to come. Meanwhile, DHelix's basic approach makes little assumption on

their specific designs. More specifically, the model folding technique (Section 4.1) applies to systems adopting neural network layers with backpropagation. The strand pairing technique (Section 4.2) is even more general, applicable to all training/inference workflows composed of operators that have complementary resource usage (such as tasks dominantly utilizing the GPU cores or the network connections).

Scalability. Finally, we highlight that aside from the operator characterization part (including the one-time offline profiling), SI remains *transparent* to model training users. If a framework executes with a set of multi-dimensional parallelism settings, the same settings can have SI enabled without changing the parameters or requiring new SI-specific parameters. Therefore, this makes SI capable of optimizing the overall training throughput with very limited impact on user-visible software complexity.

Testbed environments used in our evaluation (Section 6), as limited by the hardware resources available to this research, can be viewed as a small patch of "tiles" that form stencil units to be replicated and connected by expanding in the PP, SP/CP, and EP dimensions for larger-scale training. Incorporating SI does not affect a baseline framework's scaling out.

5 Implementation Details

We implemented DHelix with around 5000 lines of Python code, on top of Megatron-LM [38], NVIDIA's popular distributed LLM training framework. The original transformer-based implementation in Megatron-LM consists of three modules: pre-processing, transformer, and post-processing. In DHelix, we replaced the transformer module with the our SI-enabled Transformer Block, to be connected with the other two Megatron-LM pre/post-processing modules for double-strand execution. All DHelix components are based on `torch.nn.Module`, enabling users to create custom transformer models and take advantage of SI using standard PyTorch APIs, requiring no user-level code changes.

At runtime, DHelix processes the operator segment pairs sequentially according to the generated pairing plan. This sequential execution semantic is ensured by calling the `torch.cuda.synchronize` (i.e., barrier) between two segments. Within the execution of one segment, operators are further categorized into three types: computation, local-node communication, and cross-node communication. We launch three CUDA streams and dispatch each type of operator to their dedicated stream for processing. Additionally, PyTorch allows different CUDA streams to independently allocate and release memory, which can lead to fragmentation and out-of-memory errors. To address this, DHelix uses only the default CUDA stream for all memory allocation and deallocation operations.

Table 3. Llama settings used in our evaluation. We reduce the number of layers in Llama3.1-70B and Llama3.1-405B, resulting in models referred to as Llama-25B, Llama-39B, and Llama-66B, respectively.

Type	Hidden Size	Intermediate Size	#Layers	Seq Len
Llama-8B	4096	14336	32	8192,16384
Llama-25B	8192	28672	28	8192,16384
Llama-39B	16384	53248	12	8192,16384
Llama-66B	8192	28672	76	16384,32768

In addition, we follow the existing practice in mitigating contention between computation and communication kernels [14], by tuning the NCCL environment variables `NCCL_NTHREADS` and `NCCL_MAX_NCHANNELS`.

6 Evaluation

6.1 Experimental Setup

Testbed. We conducted experiments on three NVIDIA GPU clusters. The A40 cluster consists of 8 servers, connected through 100 Gbps InfiniBand, each with 8 A40 (48GB) GPU cards interconnected via PCIe4.0 at 32GB/s. The A800 cluster also has 8 servers with faster 4x200 Gbps InfiniBand interconnection, which has higher computing power and bandwidth than the A40 cluster. Each server has 8 A800 (80GB) GPUs , whose difference with the A100 cards lies in the weaker NVlink 4.0 local-node connection (400 GB/s). The A100 cluster has 1 server with 8 A100(80GB) GPUs and NVLINk (600GB/s). The H100 cluster has 4 servers connected with the fastest 8x400 Gbps InfiniBand, each with 8 H100 (80GB) GPUs and NVLINK (900GB/s).

The CUDA version used is 12.2. Due to the difficulty in requesting high-end GPU resources, the A800/H100 clusters were available to us for a very short period. We replicated our overall performance tests across testbeds and conducted the rest on the A40 one.

LLM/MoE Models. As shown in Table 3, 4, 5, we test three mainstream open-source LLMs with nine different model sizes, ranging from 6.7 to 66 billion, including Llama [15], GPT [5], and Phi MoE [1]. Llama and GPT are two families of dense transformer-based models, while Phi is a sparse mixture-of-experts (MoE) LLM. Here, we set the top-k value of MoE to 2, following its original setup, where the router gate forwards each token to top-2 relevant experts.

In specific experiments, we adjust the number of model layers to fit in the cluster's aggregate GPU memory capacity based on the original model architecture. The performance results are expected to be very similar to the un-trimmed model size when the training scales out in the PP dimension, given the relatively small PP communication size.

Baseline systems. We compare DHelix with several baselines. Naturally, we compare with the vanilla **Megatron-LM**, the widely adopted distributed training framework supporting all forms of data/model parallelism discussed, including DP, TP/SP, CP, PP, and EP. Megatron-LM already includes

Table 4. GPT settings in evaluation: the original GPT-6.7B and GPT-18B, plus GPT-30B (by reducing the number of layers from GPT-175B)

Type	Hidden Size	Intermediate Size	#Layers	Seq Len
GPT-6.7B	4096	16384	32	8192,16384
GPT-18B	6144	24576	40	8192,16384
GPT-30B	12288	49152	16	8192,16384

Table 5. MoE settings in evaluation: the original Phi-42B and its reduced-layer version Phi-31B

Type	Hidden Size	Intermediate Size	#Layers	Seq Len	#Experts
phi-16B	4096	6400	12	3072	16
Phi-31B	4096	6400	24	3072	16
Phi-42B	4096	6400	32	3072	16

communication overlap optimizations for CP, which overlap `Send/Recv` with attention computation. The second baseline **Intra-batch** extends Megatron-LM by incorporating intra-batch communication overlapping, following the design of the industrial-strength MegaScale [21]. It partitions `GEMM` operators and interleaves them with communication operators within a single batch for TP and SP.

While these two baselines following single-strand, the next one, **Wavelet+**, extends Wavelet [51], the only existing approach exploring micro-batch interleaving to our best knowledge. As discussed earlier, it is applied only to DP and adopts model replication. For a fair comparison, we implement Wavelet's simple round-robin scheduling to interleave operators within DHelix to form a pipelined Wavelet+. Note that Wavelet+ has DHelix's model folding design, to remove its model replication and enable scheduling comparison using the same model sizes.

Since DualPipe is not open source, we did not compare with DualPipe.

Metrics. Following existing practice [38], for Llama, GPT, and their variants, we report per-GPU training performance (TFLOPS/GPU) by dividing the job's total floating-point operation count on a single GPU by the total end-to-end execution time. For the sparse Phi MoE models, we also follow related literature [58] to measure by tokens/s, indicating the generation speed. This is considering that some tokens may be dropped to balance computation among experts, the end-to-end TFLOPS cannot be calculated as accurately as with the dense models. Finally, DHelix does not modify the semantics of distributed LLM training and thus does not affect the model convergence/accuracy.

6.2 Overall Performance: A40 Cluster

First, we evaluate DHelix with three tasks: dense model training, a dense model with long sequences and context parallelism (CP), and MoE model training. The global batch size is fixed at 8 million tokens for Llama/GPT, while the micro-batch size is adjusted to utilize GPU memory fully. Similarly, we set the global batch size to 1600 for Phi variants.

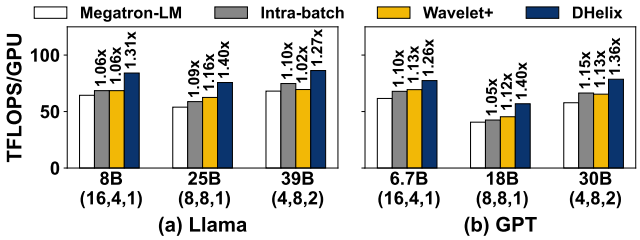

Figure 11. Overall performance w. Llama/GPT models. The tuple (x, y, z) under the bars gives DP, TP, and PP group sizes.

6.2.1 Dense Models with Normal Sequence Length.

Figure 11 compares the performance of DHelix against baselines on the Llama/GPT model with the default sequence length of 8192. The group sizes for tensor parallelism (TP) and pipeline parallelism (PP) are scaled according to the model size. For the Llama model, DHelix consistently outperforms two baselines across different parallelism strategies and model sizes and achieves 27-40%, 15-28%, and 21-25% improvement compared to Megatron-LM, Intra-batch, and Wavelet+, respectively. GPT model training shows similar results, where DHelix outperforms Megatron-LM, Intra-batch, and Wavelet+ by 26%-40%, 15%-33%, and 12%-25% improvement, respectively.

Compared with the baseline Megatron-LM, both Intra-batch and Wavelet+ show much lower performance improvement (only 5-15% and 6-16%, respectively). Our profiling finds that Intra-batch hides only 26.1% of the TP communication overhead by overlapping it with decomposed GEMM operators due to the sequential dependency within each micro-batch. Wavelet+ performs micro-batch interleaving, but with its naive round-robin scheduling, it still only hides 39.9% of the communication cost. In contrast, DHelix hides almost 83%. Section 6.4 gives a detailed analysis.

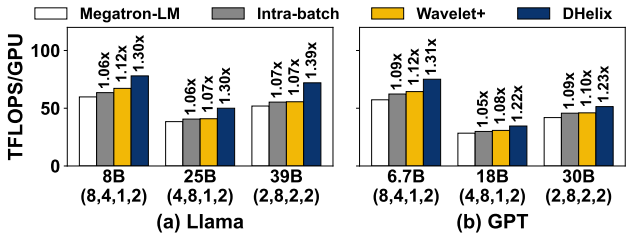

Figure 12. Llama/GPT training performance with long sequence. w in tuple (x, y, z, w) denotes CP group size.

6.2.2 Dense Models with Long Sequences.

Considering the growing demand for long-context LLMs, we next evaluate Llama and GPT model training with long sequences, doubling the sequence length from 8192 to 16384 and adding context parallelism (CP) to all configurations. Note that adding CP leads to a reduction in the DP degree, as the total GPU number stays constant.

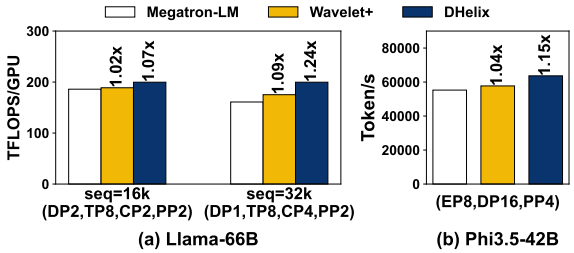

Figure 13. A800 cluster training performance of (a) Llama 66B w. varied CP sizes, and (b) Phi-42B

Figure 12 shows that again DHelix consistently outperforms Megatron-LM, Intra-batch, Wavelet+, with improvement ranging from 22%-39%, 13%-30%, 11%-30% respectively. Here, DHelix's performance gains become even more pronounced because CP introduces additional intra-layer `Send/Recv` communication for exchanging KV blocks in attention computation, where DHelix leverages its powerful and flexible pairing search to identify inter-strand overlap opportunities.

6.2.3 MoE Models. Now we move to a sparse MoE model, Phi-31B, requiring EP to be enabled and the DP group size to be divisible by the EP one. Following common practice [11, 32], we set the EP group size to 8, with DP and PP group sizes set to 16 and 4 respectively. What's more, we set the capacity factor for each expert to 6. Note that the global parallelism group size is equal to 16×4 as the EP group is a subset of the DP group.

Table 6. The training performance table for Phi-31B with EP size of 8, DP size of 16, PP size of 4 and the capacity of the expert of 6 in A40 cluster.

Metric	Megatron-LM	Wavelet+	DHelix
Token/s	59941	62454	75386
Improvement	-	5%	27%

Here, Wavelet+ can overlap communication with one short computation operation, bringing an improvement of 5% over the Megatron-LM. DHelix, on the other hand, delivers a 27% gain due to its capability to effectively overlap the time-consuming `All-to-All` intra-layer communication brought by EP.

6.3 Overall Performance on High-end Clusters

To evaluate the generality of our DHelix's performance, We then extend our evaluation to the higher-end A800 cluster, with 80GB memory per GPU, NVLINK interconnects, and higher cross-node bandwidth. In our following experiments, we adapt model/sequence sizes and data/model parallelism settings accordingly to make full use of the resources. Note that our results here drop the "Intra-batch" bar, as it brings neglectable improvement with fast NVLink.

6.3.1 Dense Models with Long Sequences. We can now raise the Llama size to 66B, the largest in our evaluation. The sequence lengths are also expanded to 16384 and 32768, with CP group size to 2 and 4 in accordance.

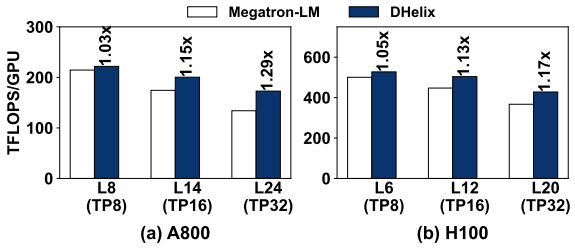

Figure 14. Scaling model training with cross-node TP, w. transformer layer configuration from Llama3.1-405B [15]. For example, "L8 (TP8)" refers to a transformer model with 8 layers and a TP group size of 8.

Figure 13-(a) presents the per-GPU training throughput. With high-speed local-node NVLINK, Megatron becomes a strong baseline, delivering impressive performance with approximately 186 TFLOPS (60% MFU) and 160.9 TFLOPS (52% MFU) in both cases. This is attributed to a reduction in local-node TP communication cost, which now accounts for only 10% of the total training time, compared to 29% on the A40 cluster, with CP size at 2. Additionally, Megatron-LM can overlap part of CP-related communication.

Wavelet+ shows slight improvement over Megatron-LM due to its simplistic communication overlap strategies. In contrast, DHelix still delivers a 7-24% improvement over Megatron-LM (5-14% over Wavelet+). Also, increasing the CP size from 2 to 4 introduces more cross-node `Send/Recv`, hurting the performance of both Megatron-LM and Wavelet+. However, DHelix effectively hides the increased cross-node communication, sustaining a throughput of 199.7 TFLOPS (64% MFU).

6.3.2 Larger MoE Model. With twice the GPU memory capacity, the A800 cluster can accommodate a full Phi-42B model and we can set the capacity factor for each expert to 8, with performance reported in Figure 13-(b). Again, Wavelet+ achieves only 4% improvement over Megatron-LM, while DHelix delivers 15%.

6.3.3 Increasing TP Group Sizes. Large-scale training efforts mostly limit the TP size to 8 (within a node), due to the prohibitive cost of TP-induced communication volume. Recently, however, cross-node TP started to be examined [52], driven by growing model size and faster networking (e.g., Nvidia DGX and InfiniBand [42] offering up to 800GB/s aggregated bandwidth). To this end, we obtain a short window of a 4-node H100 cluster to test DHelix's potential in unlocking cross-node TP.

Figure 14 lists results with increasing TP size from 8 to 32. Note that higher TP size enables larger models (more layers),[2] trading off per-GPU throughput.

[2]Due to a runtime problem that we did not have chance to debug during our access to the clusters, we were not able to enable the same number of layers on H100 as on A800

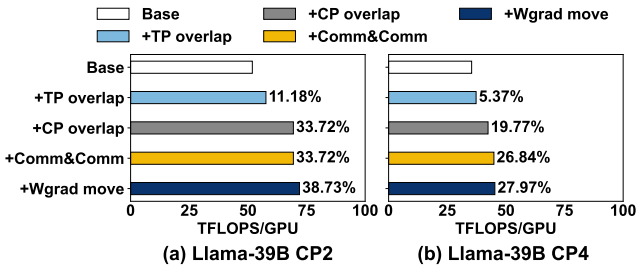

Figure 15. Breakdown of DHelix improvement by incrementally overlapping communication types

DHelix sees lower profit over Megatron-LM on H100 (up to 17%) than on A800 (up to 29%) due to the former's much powerful interconnection (hence smaller visible communication cost). However, on both GPU platforms it demonstrates the same advantage: the loss of TFLOPS from TP=8 to TP=32 is much slower while the model is over 3× larger.

To evaluate DHelix's operator overlapping effectiveness on H100, we dive into communication overhead when training with cross-node TP. Although we have hidden 50%-70% of the visible communication cost, further optimization is possible. Table 7 highlights two main factors limiting this overlap: (1) **kernel slowdown** – running computation and communication kernels concurrently on multiple CUDA streams reduces performance by 20%-30%, limiting potential performance gain; (2) **launch interval** – the small intervals between the launches of computation and communication kernels at the start of each paired segment pose an overhead amounting to 10%-20% of communication cost. A promising approach is to fuse these kernels into a single, monolithic kernel that manages GPU resources more precisely, as suggested in recent studies [23, 31].

Table 7. Composition of communication cost measured in DHelix, with different TP sizes on H100

TP size	Hidden	Visible (slowdown)	Visible (launch interval)
8	57.87%	32.48%	9.65%
16	66.73%	23.01%	10.26%
32	51.72%	29.45%	18.83%

6.4 Improvement Breakdown

Next, we analyze the individual sources of DHelix's performance gain over the Megatron-LM baseline ("Baseline") of various communication overlap strategies we introduced to DHelix. We are conducting breakdown experiments on the A40 cluster using the Llama-39B model in two scenarios: (a) CP at 2 and sequence length at 16384, and (b) CP at 4 and sequence length at 32768.

Results are given in Figures 15-(a) and (b), respectively. Test (a) uses the same setup as Llama-39B in Figure 12, while test (b) further evaluates the latency breakdown of DHelix in a longer sequence length.

TP Overlap. Starting from Baseline, we first add TP overlap, which brings 11.18% and 5.37% improvements to Test (a) and (b), respectively. Typically, having higher communication costs is good news to DHelix, as it sees a higher "profit margin". In Test (b), however, it turns into a curse as the communication overhead caused by the longer sequence length of 32768 is too high to overlap fully. Faster interconnection could change the situation and allow DHelix to gain more in these cases.

CP Overlap. We then turn on CP overlap, further hiding the CP-induced `Send/Recv` overhead, leading to another 22.5% and 14.4% improvement over the TP overlap bars, respectively. This step contributes the most significant gain from DHelix because CP-induced intra-layer communication accounts for 28% and 53% total training time in (a) and (b). Again, too much communication reduces the profit of CP overlap in the latter test.

Communication and Communication Overlap. The next addition of the overlap between local- and cross-node communication operators brings a further gain of about 7% to Test (b), helping with its abundant communication activities. It does not improve Test (a), though, as most communication there has already been overlapped by computation.

Wgrad Overlap. Finally, we show the impact of operator reordering by enabling the `Wgrad` operator to move around as allowed by the DAG sorting. Test (a) yields an approximately 5% improvement over the previous bar, which helps with "overlap bubbles" due to data dependency. In Test (b), there is insufficient computation to hide communication, and moving `Wgrad` only brings about 1% performance gain.

6.5 Memory Space Utilization with SI

Finally, we verify DHelix's memory optimization, which is crucial for supporting training jobs with model sizes as large as possible. We compare DHelix's SI scheme against Megatron-LM, the single-strand baseline, and another straightforward but memory-unfriendly Bi-directional double-strand implementation (illustrated in Figure 6). The single-strand represents the ideal maximum model size that GPU could support. In our experiment, we use the layer setup of the Llama-25B model in table 3 under the setup DP=4, TP=8, and PP=2, with micro-batch size set to 1. We increase the number of model layers to determine the maximum model parameter size supported by the system.

Our results find that DHelix's maximum supported model size (39B) is close to Megatron-LM (40B), a mere 2.5% model size reduction for a training throughput gain of 40%. This slight decrease is due to DHelix's scheduling certain forward operators in one strand being executed before the backward operators of the other, producing marginally higher peak memory usage. In contrast, without SI's model folding solution, the bi-directional double-strand implementation is forced to use model-state replication and, therefore, can accommodate models only up to 32B.

7 Other Related Work

Communication overlapping optimization Communication overlapping techniques are widely used to mitigate communication bottlenecks in various parallelism strategies. For data parallelism(including ZeRO-family parallelism) [16, 20, 45], prior work has explored to better overlap the computation and communication by priority-based scheduling and tensor partition. Centauri [7] improves upon these works by graph-level scheduling. However, these existing communication overlap techniques are applied in single-strand scheduling and are orthogonal to the double-strand scheduling used by DHelix.

Other communication optimizations Some research[3, 56] focuses on optimizing collective communication operators based on hardware topology, improving network bandwidth utilization. Despite these improvements, intra-layer communication still constitutes a significant portion of end-to-end large language model training. MiCS [60] minimizes the communication size and reduces network traffic over slower links by leveraging a hierarchical model partition strategy. Other techniques[2, 49, 54, 55] leverage the robustness of neural network training by compressing gradients or selectively transmitting part of the gradients during synchronization, thereby reducing the volume of communication. However, such communication optimization methods may negatively impact model training quality.

GPU resource sharing. Existing research [27, 57, 59] has extensively explored GPU sharing techniques to fully harness GPU computing potential, focusing on temporal and spatial dimensions. Temporal GPU sharing involves software-based time-sharing mechanisms, where Gandiva [57] introduced a GPU time-slicing mechanism primarily to accelerate hyperparameter tuning jobs. This technique initiates job switching at iteration boundaries, reducing CPU-GPU communication overhead.

In contrast, spatial sharing techniques are equally crucial. Zico [27] monitors memory usage patterns across training jobs, dynamically allocating and deallocating memory to ensure that reclaimed memory is globally available. MPS [39] partitions GPU memory statically for concurrent jobs, while TensorRT [44] Inference Server enables simultaneous deep learning inference in parallel on a single GPU using GPU streams. Salus [59] supports both temporal and spatial sharing by allowing rapid switching among DNN jobs and providing fine-grained memory abstraction. Additionally, pipeline-oriented optimizations in frameworks like Megatron-LM [34], PipeDream, and Chimera [26] execute computations for multiple microbatches simultaneously on the same set of GPU devices.

Compared to these approaches, DHelix focuses on optimizing a single job by employing a sophisticated model to search for better scheduling strategies, thereby significantly enhancing GPU utilization.

8 Conclusion

We introduce DHelix, a DNA-like abstraction for distributed LLM training that facilitates flexible overlaps between computation and communication of two co-executed micro-batches. DHelix demonstrates that there is still considerable room for aggressively overlapping operators in the LLM training pipelines, and communication for cross-node tensor parallelism may be more manageable than it appears if we have a systematic mechanism to exploit the co-scheduling optimization space.

References

[1] Marah Abdin, Sam Ade Jacobs, Ammar Ahmad Awan, Jyoti Aneja, Ahmed Awadallah, Hany Awadalla, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Harkirat Behl, et al. Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219*, 2024.

[2] Youhui Bai, Cheng Li, Quan Zhou, Jun Yi, Ping Gong, Feng Yan, Ruichuan Chen, and Yinlong Xu. Gradient compression supercharged high-performance data parallel dnn training. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, pages 359–375, 2021.

[3] Prithwish Basu, Liangyu Zhao, Jason Fantl, Siddharth Pal, Arvind Krishnamurthy, and Joud Khoury. Efficient all-to-all collective communication schedules for direct-connect topologies. In *Proceedings of the 33rd International Symposium on High-Performance Parallel and Distributed Computing*, pages 28–41, 2024.

[4] Rajesh Bhayana. Chatbots and large language models in radiology: a practical primer for clinical and research applications. *Radiology*, 310(1):e232756, 2024.

[5] Tom B Brown. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[6] Liwen Chang, Wenlei Bao, Qi Hou, Chengquan Jiang, Ningxin Zheng, Yinmin Zhong, Xuanrun Zhang, Zuquan Song, Ziheng Jiang, Haibin Lin, et al. Flux: Fast software-based communication overlap on gpus through kernel fusion. *arXiv preprint arXiv:2406.06858*, 2024.

[7] 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*, pages 178–191, 2024.

[8] Ping Chen, Wenjie Zhang, Shuibing He, Yingjie Gu, Zhuwei Peng, Kexin Huang, Xuan Zhan, Weijian Chen, Yi Zheng, Zhefeng Wang, et al. Optimizing large model training through overlapped activation recomputation. *arXiv preprint arXiv:2406.08756*, 2024.

[9] Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning (2023). *arXiv preprint arXiv:2307.08691*, 2023.

[10] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.

[11] deepseek. deepseek-hg. https://huggingface.co/deepseek-ai/DeepSeek-V2.5, 2024. "[accessed-Oct-2024]".

[12] DeepSeek-AI. Deepseek-v3 technical report. https://github.com/deepseek-ai/DeepSeek-V3/blob/main/DeepSeek_V3.pdf, 2024. "[accessed-Dec-2024]".

[13] DHeLlam. Dhellam. https://github.com/DHeLlam-502/dhellam/tree/main, 2024. "[accessed-Dec-2024]".

[14] Jiangsu Du, Jinhui Wei, Jiazhi Jiang, Shenggan Cheng, Dan Huang, Zhiguang Chen, and Yutong Lu. Liger: Interleaving intra- and inter-operator parallelism for distributed large model inference. In *Proceedings of the 29th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*, PPoPP '24, page 42–54, New York, NY, USA, 2024. Association for Computing Machinery.

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

[16] Sayed Hadi Hashemi, Sangeetha Abdu Jyothi, and Roy Campbell. Tictac: Accelerating distributed deep learning with communication scheduling. *Proceedings of Machine Learning and Systems*, 1:418–430, 2019.

[17] Jiaao He, Jiezhong Qiu, Aohan Zeng, Zhilin Yang, Jidong Zhai, and Jie Tang. Fastmoe: A fast mixture-of-expert training system. *arXiv preprint arXiv:2103.13262*, 2021.

[18] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in neural information processing systems*, 32, 2019.

[19] 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*, pages 402–416, 2022.

[20] Anand Jayarajan, Jinliang Wei, Garth Gibson, Alexandra Fedorova, and Gennady Pekhimenko. Priority-based parameter propagation for distributed dnn training. *Proceedings of Machine Learning and Systems*, 1:132–145, 2019.

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

[22] Vijay Anand Korthikanti, Jared Casper, Sangkug Lym, Lawrence McAfee, Michael Andersch, Mohammad Shoeybi, and Bryan Catanzaro. Reducing activation recomputation in large transformer models. *Proceedings of Machine Learning and Systems*, 5:341–353, 2023.

[23] Ao Li, Bojian Zheng, Gennady Pekhimenko, and Fan Long. Automatic horizontal fusion for gpu kernels. In *2022 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, pages 14–27. IEEE, 2022.

[24] Fanxin Li, Shixiong Zhao, Yuhao Qing, Xusheng Chen, Xiuxian Guan, Sen Wang, Gong Zhang, and Heming Cui. Fold3d: Rethinking and parallelizing computational and communicational tasks in the training of large dnn models. *IEEE Transactions on Parallel and Distributed Systems*, 34(5):1432–1449, 2023.

[25] Shengwei Li, Zhiquan Lai, Yanqi Hao, Weijie Liu, Keshi Ge, Xiaoge Deng, Dongsheng Li, and Kai Lu. Automated tensor model parallelism with overlapped communication for efficient foundation model training. *arXiv preprint arXiv:2305.16121*, 2023.

[26] Shigang Li and Torsten Hoefler. Chimera: efficiently training large-scale neural networks with bidirectional pipelines. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–14, 2021.

[27] Gangmuk Lim, Jeongseob Ahn, Wencong Xiao, Youngjin Kwon, and Myeongjae Jeon. Zico: Efficient GPU memory sharing for concurrent DNN training. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pages 161–175, 2021.

[28] Zhiqi Lin, Youshan Miao, Quanlu Zhang, Fan Yang, Yi Zhu, Cheng Li, Saeed Maleki, Xu Cao, Ning Shang, Yilei Yang, et al. nnScaler:Constraint-Guided Parallelization Plan Generation for Deep Learning Training. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 347–363, 2024.

[29] Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, et al. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024.

[30] Hao Liu, Matei Zaharia, and Pieter Abbeel. Ring attention with blockwise transformers for near-infinite context. *arXiv preprint arXiv:2310.01889*, 2023.

[31] Yunzhuo Liu, Bo Jiang, Shizhen Zhao, Tao Lin, Xinbing Wang, and Chenghu Zhou. Libra: Contention-aware gpu thread allocation for data parallel training in high speed networks. In *IEEE INFOCOM 2023-IEEE Conference on Computer Communications*, pages 1–10. IEEE, 2023.

[32] mistrialai. mixtral. https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1, 2024. "[accessed-Oct-2024]".

[33] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R Devanur, Gregory R Ganger, Phillip B Gibbons, and Matei Zaharia. Pipedream: Generalized pipeline parallelism for dnn training. In *Proceedings of the 27th ACM symposium on operating systems principles*, pages 1–15, 2019.

[34] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, et al. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15, 2021.

[35] Jianmo Ni, Gustavo Hernandez Abrego, Noah Constant, Ji Ma, Keith B Hall, Daniel Cer, and Yinfei Yang. Sentence-t5: Scalable sentence encoders from pre-trained text-to-text models. *arXiv preprint arXiv:2108.08877*, 2021.

[36] NVDIA. H100 hardware configuration. https://resources.nvidia.com/en-us-tensor-core/nvidia-tensor-core-gpu-datasheet, 2024. "[accessed-Oct-2024]".

[37] NVIDIA. Context parallelism overview. https://docs.nvidia.com/megatron-core/developer-guide/latest/api-guide/context_parallel.html, 2024. "[accessed-Sept-2024]".

[38] NVIDIA. Megatron-lm and megatron-core. https://github.com/NVIDIA/Megatron-LM, 2024. "[accessed-Sept-2024]".

[39] NVIDIA. Mps service. https://docs.nvidia.com/deploy/pdf/CUDA_Multi_Process_Service_Overview.pdf,, 2024. "[accessed-Aug-2024]".

[40] NVIDIA. Nccl and mpi, 2024. https://docs.nvidia.com/deeplearning/nccl/user-guide/docs/mpi.html?highlight=alltoall#other-collectives-and-point-to-point-operations [Accessed: September 2024].

[41] NVIDIA. Nccl operations. https://docs.nvidia.com/deeplearning/nccl/user-guide/docs/usage/operations.html#allgather, 2024. "[accessed-Sept-2024]".

[42] Nvidia. Nvidia connectx infiniband adapters. https://www.nvidia.com/en-us/networking/infiniband-adapters/, 2024. "[accessed-Sept-2024]".

[43] NVIDIA. Nvidia nsight systems. https://developer.nvidia.com/nsight-systems, 2024. "[accessed-Sept-2024]".

[44] NVIDIA. Nvidia triton inference server boosts deep learning inference. https://developer.nvidia.com/blog/nvidia-serves-deep-learning-inference, 2024. "[accessed-Aug-2024]".

[45] Yanghua Peng, Yibo Zhu, Yangrui Chen, Yixin Bao, Bairen Yi, Chang Lan, Chuan Wu, and Chuanxiong Guo. A generic communication scheduler for distributed dnn training acceleration. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pages 16–29, 2019.

[46] Jie Ren, Samyam Rajbhandari, Reza Yazdani Aminabadi, Olatunji Ruwase, Shuangyan Yang, Minjia Zhang, Dong Li, and Yuxiong He. {Zero-offload}: Democratizing {billion-scale} model training. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pages 551–564,

2021.

[47] Alexander Sergeev and Mike Del Balso. Horovod: fast and easy distributed deep learning in tensorflow. *arXiv preprint arXiv:1802.05799*, 2018.

[48] 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.

[49] Jaeyong Song, Jinkyu Yim, Jaewon Jung, Hongsun Jang, Hyung-Jin Kim, Youngsok Kim, and Jinho Lee. Optimus-cc: Efficient large nlp model training with 3d parallelism aware communication compression. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 560–573, 2023.

[50] Guanhua Wang, Heyang Qin, Sam Ade Jacobs, Connor Holmes, Samyam Rajbhandari, Olatunji Ruwase, Feng Yan, Lei Yang, and Yuxiong He. Zero++: Extremely efficient collective communication for giant model training. *arXiv preprint arXiv:2306.10209*, 2023.

[51] Guanhua Wang, Kehan Wang, Kenan Jiang, Xiangjun Li, and Ion Stoica. Wavelet: Efficient dnn training with tick-tock scheduling. *Proceedings of Machine Learning and Systems*, 3:696–710, 2021.

[52] Guanhua Wang, Chengming Zhang, Zheyu Shen, Ang Li, and Olatunji Ruwase. Domino: Eliminating communication in llm training via generic tensor slicing and overlapping. *arXiv preprint arXiv:2409.15241*, 2024.

[53] 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*, pages 93–106, 2022.

[54] Yiding Wang, Decang Sun, Kai Chen, Fan Lai, and Mosharaf Chowdhury. Egeria: Efficient dnn training with knowledge-guided layer freezing. In *Proceedings of the Eighteenth European Conference on Computer Systems*, pages 851–866, 2023.

[55] Zhuang Wang, Haibin Lin, Yibo Zhu, and TS Eugene Ng. Hi-speed dnn training with espresso: Unleashing the full potential of gradient compression with near-optimal usage strategies. In *Proceedings of the Eighteenth European Conference on Computer Systems*, pages 867–882, 2023.

[56] William Won, Midhilesh Elavazhagan, Sudarshan Srinivasan, Ajaya Durg, Samvit Kaul, Swati Gupta, and Tushar Krishna. Tacos: Topology-aware collective algorithm synthesizer for distributed machine learning. *arXiv preprint arXiv:2304.05301*, 2023.

[57] Wencong Xiao, Romil Bhardwaj, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, Zhenhua Han, Pratyush Patel, Xuan Peng, Hanyu Zhao, Quanlu Zhang, et al. Gandiva: Introspective cluster scheduling for deep learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 595–610, 2018.

[58] Leyang Xue. Moe-infinity: Offloading-efficient moe model serving. *arXiv preprint arXiv:2401.14361*, 2024.

[59] Peifeng Yu and Mosharaf Chowdhury. Salus: Fine-grained gpu sharing primitives for deep learning applications. *arXiv preprint arXiv:1902.04610*, 2019.

[60] Zhen Zhang, Shuai Zheng, Yida Wang, Justin Chiu, George Karypis, Trishul Chilimbi, Mu Li, and Xin Jin. Mics: near-linear scaling for training gigantic model on public cloud. *arXiv preprint arXiv:2205.00119*, 2022.

[61] Lianmin Zheng, Zhuohan Li, Hao Zhang, Yonghao Zhuang, Zhifeng Chen, Yanping Huang, Yida Wang, Yuanzhong Xu, Danyang Zhuo, Eric P Xing, et al. Alpa: Automating Inter-and Intra-Operator parallelism for distributed deep learning. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 559–578, 2022.

[62] Yonghao Zhuang, Lianmin Zheng, Zhuohan Li, Eric Xing, Qirong Ho, Joseph Gonzalez, Ion Stoica, Hao Zhang, and Hexu Zhao. On optimizing the communication of model parallelism. *Proceedings of Machine Learning and Systems*, 5, 2023.

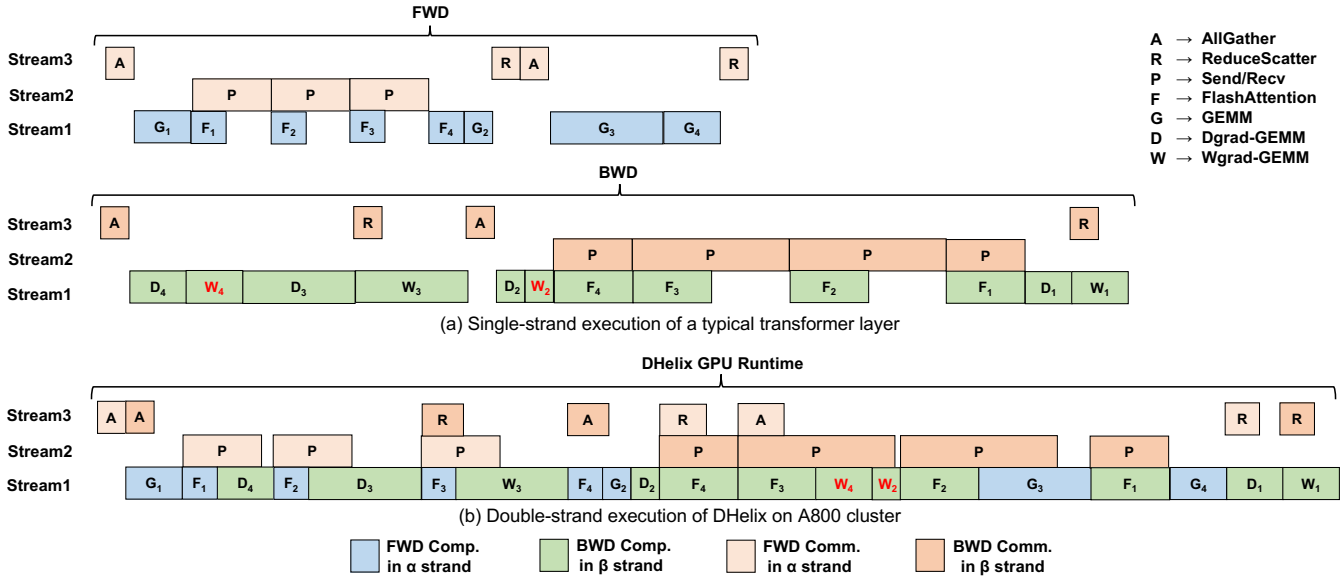

Figure 16. Visualization of the single-strand and double-strand CUDA schedule of SI on our A800 cluster, based on traces from the experiment shown in Figure 13 training Llama-66B (TP=8, CP=4). The top two rows "(a) " show the forward and backward passes executed without SI, respectively (where computation and communication have already been overlapped within each strand.) The bottom row "(b)" shows the SI double-strand execution, based on the optimal pairing found by dynamic programming. For clarity, operators with negligible execution time (e.g., `LayerNorm`, `Dropout`) are omitted. Here, DHelix efficiently overlaps over 82% of the total communication cost (*e.g.*, TP-triggered local-node `AllGather` and `ReduceScatter` and CP-induced cross-node `Send/Recv` communication). The overall speedup of DHelix is 1.24×, compared to single-strand execution, as a result.