Centauri: Enabling Efficient Scheduling for Communication-Computation Overlap in Large Model Training via Communication Partitioning

Chang Chen
charlie_chen@pku.edu.cn
Peking University
China

Xiuhong Li*
lixiuhong@pku.edu.cn
Peking University
China

Qianchao Zhu
dysania@pku.edu.cn
Peking University
China

Jiangfei Duan
dj021@ie.cuhk.edu.hk
The Chinese University of Hong Kong
China

Peng Sun
sunpeng@pjlab.org.cn
Shanghai AI Lab
China

Xingcheng Zhang
zhangxingcheng@pjlab.org.cn
Shanghai AI Lab
China

Chao Yang*
chao_yang@pku.edu.cn
Peking University
China

Abstract

Efficiently training large language models (LLMs) necessitates the adoption of hybrid parallel methods, integrating multiple communications collectives within distributed partitioned graphs. Overcoming communication bottlenecks is crucial and is often achieved through communication and computation overlaps. However, existing overlap methodologies tend to lean towards either fine-grained kernel fusion or limited operation scheduling, constraining performance optimization in heterogeneous training environments.

This paper introduces *Centauri*, an innovative framework that encompasses comprehensive communication partitioning and hierarchical scheduling schemes for optimized overlap. We propose a partition space comprising three inherent abstraction dimensions: primitive substitution, topology-aware group partitioning, and workload partitioning. These dimensions collectively create a comprehensive optimization space for efficient overlap. To determine the efficient overlap of communication and computation operators, we decompose the scheduling tasks in hybrid parallel training into three hierarchical tiers: operation, layer, and model. Through these techniques, *Centauri* effectively overlaps communication latency and enhances hardware utilization. Evaluation results demonstrate that *Centauri* achieves up to 1.49× speedup over prevalent methods across various parallel training configurations.

ACM Reference Format:
Chang Chen, Xiuhong Li, Qianchao Zhu, Jiangfei Duan, Peng Sun, Xingcheng Zhang, and Chao Yang. 2024. Centauri: Enabling Efficient Scheduling for Communication-Computation Overlap in Large Model Training via Communication Partitioning. In *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3 (ASPLOS '24), April 27-May 1, 2024, La Jolla, CA, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3620666.3651379

*Corresponding author.

ASPLOS '24, April 27-May 1, 2024, La Jolla, CA, USA

ACM ISBN 979-8-4007-0386-7/24/04…$15.00
https://doi.org/10.1145/3620666.3651379

1 Introduction

Recent advancements in LLMs have brought about a revolution in the field of natural language processing (NLP) [1, 5, 7, 35]. According to the scaling law of LLMs [14], there exists a robust correlation between the sizes of models and their performance capabilities. However, the pre-training cost of LLMs is also notably escalating with the growth of model sizes, demanding thousands of GPU-days for computation and substantial memory capacity [1].

Therefore, diverse parallelism methods [21, 29, 30] are applied to distribute the resource-intensive LLMs training tasks across a GPU cluster. These methods encompass a range of intricate parallelism paradigms, including data parallelism (DP) [18], tensor parallelism (TP) [21, 33], pipeline parallelism (PP) [10, 19, 21], fully sharded data parallelism (FSDP)

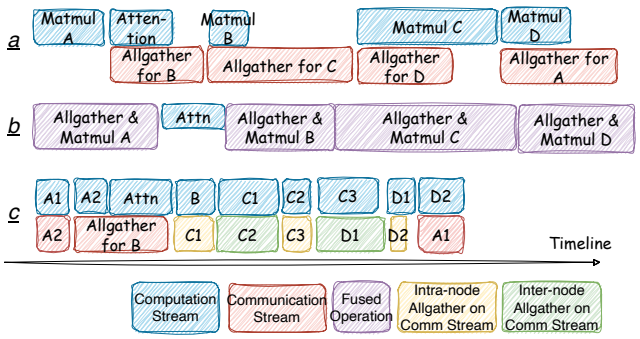

Figure 1. Different overlapping strategies on FSDP training of a simplified Transformer structure: *a* is direct scheduling that weights are gathered before MatMuls; *b* is MatMul and Allgather kernel fusion; *c* is Centauri scheduling that communication is partitioned from group and workload dimensions for better overlapping

[42] and Zero [29], each introducing distinct communication patterns to the distributed tasks. Consequently, communication overhead occupies a substantial portion of the end-to-end execution time in parallel LLMs training, particularly within heterogeneous network environments [11, 40]. Thus, minimizing communication overheads is crucial for the efficient scaling-out of parallel LLMs training tasks.

Simultaneous execution of computation and communication operations is vital for overlapping the communication overhead during parallel LLMs training. Previous efforts have aimed to enhance this overlap through scheduling and kernel fusion approaches. However, some frameworks [9, 17, 18, 27, 28] focus on optimizing the scheduling of a single parallel method and fall short in addressing the intricate overlap challenges within hybrid parallel methods [21, 29]. Notably, even for the forward and backward passes, optimal overlapping patterns may vary. Certain strategies [20, 32] rely on concurrent communications across multiple GPU streams but may lack exploration into a broader overlap space involving computation. Meanwhile, graph-level overlapping [16, 17, 42] in a coarse-grained manner might under-utilize hardware resources, as depicted by the notable gaps of resource idling shown in Fig 1 *a*. On the other hand, some sophisticated compiler-style works [11, 40] segment collectives and their adjacent computations, generating fused kernels at the operation level. Fine-grained kernel fusion may potentially overlook a wider range of graph-level scheduling opportunities. This is illustrated in Fig 1, where MatMul B in *a* completes ahead of fused kernel B in *b*.

These limitations underscore two fundamental challenges in optimizing overlap in LLMs training. The first challenge revolves around systematically and comprehensively exploring the full potential of overlapping space. To tackle this, we propose a key insight: communication inherently is a mapping transformation (primitive) of workload across a group of devices. Appropriately partitioning communication operations can expand the optimization space for communication overlap (see Fig 1 *c*). Thus, we propose an abstraction of three dimensions: 1) primitive, 2) group, and 3) workload. This comprehensive abstraction forms our partitioning space. Table 1 demonstrates the expressiveness of our abstraction, revealing the optimization spaces in most recent related works are subsets of our space. The second challenge is efficiently scheduling the partitioned communication and computation operations in the optimization space for complex hybrid parallel tasks, rather than from a single parallel perspective. To address this challenge, we recognize that the repetitive layer-wise architectures of LLMs and the iterative nature of gradient accumulation during training allow us to simplify the scheduling challenge into three hierarchical tiers: 1) operation, 2) layer, and 3) model. Each tier focuses on a different level of granularity, thereby preventing excessive entanglement between them.

Table 1. Overlapping Capability of Popular Frameworks

Works	Primitive	Group	Workload	Scheduling
Better Together [20]	✓	-	-	✓
Breadth-First [17]	-	-	-	✓
CoCoNet [11]	✓	-	✓	-
Dist-Einsum [40]	-	-	✓	-
DeepSpeed Zeros [29, 30]	✓	-	✓	-
Megatron-LM [21, 33]	-	-	-	✓
OOO-Backprop [27]	-	-	-	✓
Torch DDP, FSDP [18, 42]	-	-	✓	✓

This paper introduces *Centauri*, a training system designed to facilitate efficient scheduling for overlapping communication and computation in LLMs training through communication partitioning. *Centauri* contains two key components: partitioning and scheduling. Firstly, to broaden the optimization space for overlapping, we perform communication partitioning with consideration of primitive, group, and workload dimensions. Primitive partitioning primarily focuses on the equivalence of substitution and the scalability of substituted collectives. A collective group of ranks can be subdivided into several groups in a topology-aware manner, maximizing the utilization of high intra-node bandwidths. Workload partitioning requires the selection of compatible dimensions based on the workload analysis of a sequence of communication and dependent computation operations.

With a comprehensive partition space, efficient scheduling of hybrid parallel training tasks can unlock the full performance potential of overlapping with consideration of operation, layer, and model levels. At the operation level, orchestrating the execution order of communication among different partitioned groups with appropriate workload granularity is essential to facilitate the overlapping of partitioned operations. For layer-level backward overlapping, varying

三类维度的通信划分分别是什么？

combinations of hybrid parallel methods result in diverse communication patterns. Scheduling operations with adaptive priority according to the communication patterns assumes promising layer-level performance improvements. Model-level focuses on maximizing the overlap of forward, backward, and model update phases across different hybrid configurations. This hierarchical disentanglement of scheduling space ensures the exploration of promising overlapping strategies at a holistic level with small searching overheads.

This paper makes the following contributions:

- We put forward a novel communication partition abstraction, comprising three dimensions: primitive, group, and workload. The adaptable combinations of these dimensions encompass prevalent partition strategies.
- We decouple the complex scheduling optimization space of hybrid parallel LLMs training into 3 hierarchical tiers: operation, layer, and model. They focus on fine-granularity operation overlapping, backward adaptive scheduling, and elastic cross-phase scheduling, respectively.
- We implement *Centauri* within a widely adopted LLMs training framework Megatron-LM [33]. We encapsulate the transformer layer with different partitioning and scheduling strategies and develop a new distributed reducer, optimizer, and pipeline scheduler.
- We evaluate the proposed techniques using an open-source LLM LLaMA [35] across various parallel training methods in two heterogeneous environments. The results demonstrate that *Centauri* achieves a remarkable speedup of up to 1.45× on single parallel tasks and 1.49× on hybrid tasks with strong scalability, compared to prevalent methods.

2 Background

2.1 Collective Communication

MPI-style [8, 34] collective operations serve as essential abstractions for data exchange patterns, widely adopted to facilitate distributed tensor computation across P devices for each volume D. Three commonly used collectives in hybrid parallel training are:

- **Allreduce** aggregates data from different ranks by applying a reduction operation for a result of volume D.
- **Reduce-Scatter** reduces input values across ranks, with each rank receiving a subpart of the result corresponding to a volume of D/P. This is typically employed after intensive computation operations.
- **Allgather** gathers values from all ranks and distributes the result of volume PD to all ranks. This is typically followed by intensive computation operations.

Communication libraries, such as the NVIDIA Collective Communications Library (NCCL) [24], offer high-performance implementations of these collectives on specific hardware links. In a GPU cluster, GPUs within a node are commonly connected to high-bandwidth and low-latency NVSwitches [26] with NVLink Network interconnects [25]. Collectives are often implemented using a ring topology [24] to maximize bandwidth usage within a node. Conversely, GPUs across nodes are typically connected through networks with limited bandwidth and high latency. For multi-machine collective implementations with full bandwidth and logarithmic latency, the double binary tree approach [22] is commonly employed.

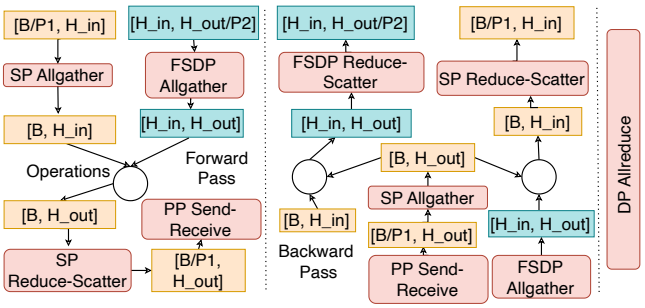

Figure 2. Simplified forward and backward workflows of hybrid parallelism, encompassing sequence parallelism, fully sharded data parallelism, pipeline parallelism, and data parallelism.

2.2 Hybrid Parallel Training

LLM training often demands a large amount of memory that exceeds the capacity of a single accelerator. As a result, LLM tasks are usually distributed across multiple devices. The choice of distributed strategies varies, each characterized by unique communication patterns, contingent on specific configurations.

An illustration of a hybrid parallel methods combination is depicted in Fig 2. **Data parallelism** (DP) [18], a widely used parallel training approach, involves distributing data samples across devices. In this paradigm, a time-consuming allreduce operation is added to accumulate gradients, which is usually scheduled to be overlapped with back-propagation [9, 17, 18]. **Fully sharded data parallelism** (FSDP) [42] partitions model parameters across devices and introduces an allgather operation to pre-fetch weights before launching dependent computations. The input tensor shape is $[H_{in}, H_{out}/P]$, producing an output tensor of shape $[H_{in}, H_{out}]$, where H_{in} is the contraction dimension and H_{out} is the output dimension. Post-computation, the memory allocated to store the global weights is released, addressing memory constraints in LLMs tasks. In the backward pass, FSDP weight gradients are also reduce-scattered. **Tensor parallelism** (TP) [21] has evolved into **sequence parallelism** (SP) [15] for enhanced memory optimization. With SP, activations are collectively gathered before a column-sharded MatMul operation within the transformer structure, and output activations of a row-sharded

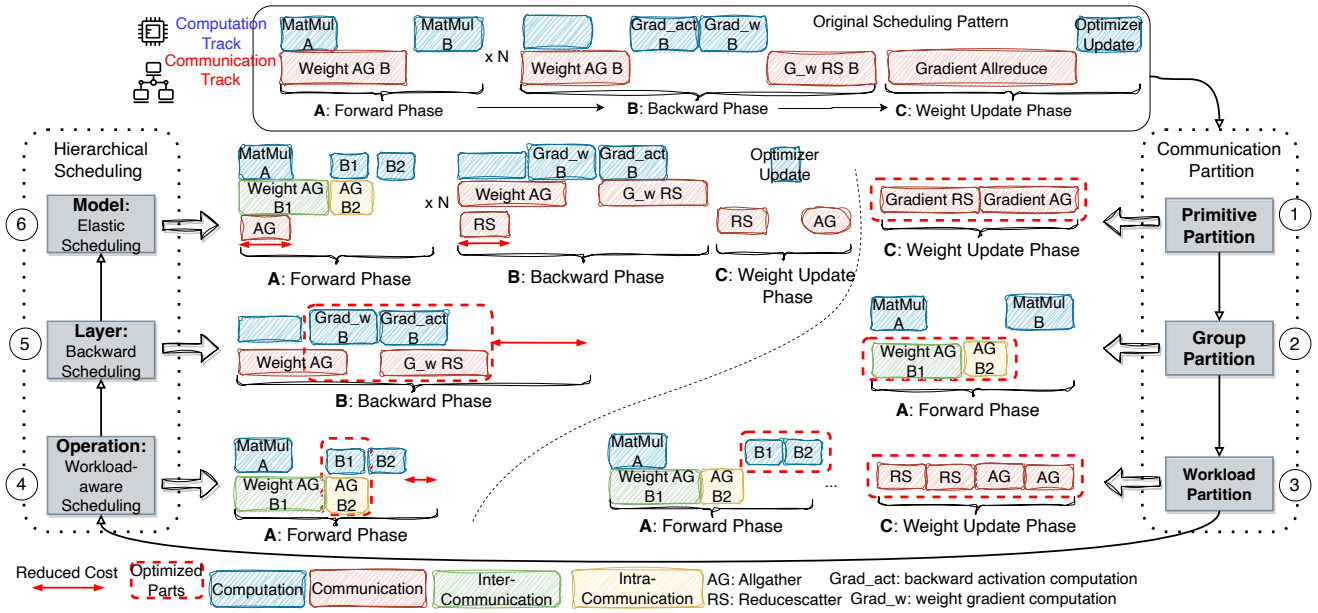

Figure 3. *Centauri* workflow overview for a hybrid parallel training example of DP and FSDP. ① Primitive substitution: Allreduce is split into reduce-scatter and allgather. ② Group partition: Allgather in the forward phase is split into inter-node group and intra-node group communication. ③ Workload partition: This step focuses on splitting collective and computation tasks with proper granularity. ④ Operation scheduling: overlap between two split collective and computation operations. ⑤ Layer scheduling: The execution is adjusted according to the critical path within a layer. ⑥ Model scheduling: Overlapping between different phases enhances overall training efficiency.

MatMul are subsequently reduce-scattered. For activation gathering of SP, the input shape is $[B/P, H_{in}]$, while the output tensor is of shape $[B, H_{in}]$, where B represents the token size dimension, and H_{in} is the hidden size dimension. The shapes for reduce-scatter are vice versa. In inter-layer **pipeline parallelism** (PP) [6, 10, 21], activation tensors are delivered among devices through point-to-point communication primitives.

2.3 Motivations

Based on the hybrid parallel patterns, we discern two types of dependencies between computation and communication:

- Sequential dependency: This involves a strict sequential order of computation and communication execution. An example is illustrated in the original task in Fig 3, where the global gradient allreduce of DP in the weight update phase depends on the backward phase.
- Non-sequential dependency: Computation and communication are independent, allowing for direct overlapping space. However, the coarse granularity of overlapping might be inefficient. An example can be found in the forward phase, where the weight pre-fetch operation allgather B overlaps with MatMul A, as depicted in the original scheduling pattern in Fig 3, with a notable idle gap on the computation track.

However, given the communication-computation constraints outlined above, the optimization space for overlapping is inherently limited for both types of dependencies. To break up these constraints, it is crucial to effectively partition communication primitives, along with their dependent computation operations. Such partitioning significantly expands the optimization space for overlapping. Take Fig 3 as an example. To break up the sequential dependency of gradient allreduce of DP, ① first partitions allreduce into reduce-scatter (RS) + allgather (AG) with primitive substitution. Then the workload of AG is further partitioned into two chunks in ③, allowing for overlapping with the forward phase in ⑥. For non-sequential dependency, the weight AG for B is initially independent of MatMul A, leading to insufficient overlap, as seen in Fig 3. To address this, steps ② and ③ involve partitioning AG B and its dependent computation for optimized scheduling, as demonstrated in ④.

3 Overview

The workflow of *Centauri* includes two integral parts: communication partition and hierarchical scheduling, as illustrated in Fig 3. The communication partition phase generates the potential partition space and selects an efficient strategy for each communication collective, considering three fundamental dimensions. The hierarchical scheduling step disentangles the complex hybrid parallel communication

Table 2. Primitive Substitution List

Primitive	Sub-primitives	Scalability
AllReduce	Reduce + Broadcast	x
	Reduce-scatter + AllGather	✓
Reduce-Scatter	Reduce + Scatter	x
	Reduces of distinct roots	✓
AllGather	Gather + Broadcast	x
	Broadcasts of distinct roots	✓

collectives into three tiers. Each collective is assigned to a specific scheduling tier. Each tier selects the partition and scheduling scheme with lower overhead. This approach aims to achieve an optimized overall overlapping scheme.

3.1 Communication Partition

A collective operation fundamentally acts as a mapping transformation of workload involving a data volume, denoted as D, across a group of devices designated as G. In this context, we define three essential dimensions: primitive, group, and workload, forming the basis for generating our partition space. The combination space of these dimensions is expansive. To ensure searching efficiency among these partition patterns without losing generality, we choose to evolve the partition space in the sequential order of the three dimensions. Primitive substitution establishes the fundamental communication patterns and cost models for group and workload partitions. With a primitive pattern in place, devices group can be subdivided into sub-groups. Workload partitioning is the final step to determine the computation operations involved in the overlapping chain and the partition dimensions of operations in the chain.

3.1.1 Primitive Substitution. A collective primitive can be substituted by several sub-primitives, maintaining the same communication effects, as listed in Table 2. Some of these sub-primitives may operate at a finer granularity with negligible partition overhead, thereby enhancing the potential for overlapping with additional computation operations. For instance, in the context of DP, the allreduce operation can be split into reduce and broadcast, or reduce-scatter and allgather, enabling them to be overlapped separately by backward and forward phases. Reduce-scatter can be further replaced by multiple reduce operations with distinct root ranks, while allgather can be considered equivalent to multiple broadcast operations. From the point of primitive substitution, an interesting observation emerges concerning the Zero-1,2 and sequence parallelism algorithms. Specifically, following the substitution of allreduce with reduce-scatter and allgather, the allgather operation can strategically undergo repositioning through elementwise operations. This transformation leads to the evolution of TP and DP into SP and Zero-1,2.

Ensuring optimal performance scalability is a key factor in deciding on operation substitutions. Some substitutions, which could potentially increase the total theoretical latency, are pruned from partition space. A case in point is the choice to favor reduce-scatter over reduce + scatter, as the latter doubles the latency of the former. While some rooted sub-primitives may exhibit nearly similar theoretical latency as the original primitive, it is notable that communication contention on the root ranks can significantly increase the overhead of rooted collectives. Consequently, the unrooted reduce-scatter and allgather pair are usually favored when replacing the bottlenecked allreduce communication, especially as the number of devices largely increases.

3.1.2 Group Partition. Collective communications are conducted within a rank group G, which can be further subdivided into smaller groups $\{G_1, G_2, G_3, ...\}$ to achieve finer communication granularity. Specific iterative executions across those groups can output results identical to those obtained from the original group. Due to the intricate nature of group partition space, a hierarchical group partition becomes necessary for aligning with the hierarchical topology of underlying networks [37]. In a group partition scheme, devices can be arranged into a d-dimension mesh with a shape denoted as $[N_1, N_2, ..., N_d]$, where $\prod_{i=1}^{d} N_i = |G|$. Each device of group rank r is assigned a d dimensional index. The k^{th} dimension of index(r) is computed as $(r/\prod_{i=k+1}^{d} N_i)\%N_k$. With this abstraction, rank r is affiliated with d partitioned sub-groups of $\{G_1^r, G_2^r, ..., G_d^r\}$, where $G_i^r = \{k | index(k)[j] = index(r)[j], \forall j \neq i\}$. Through group partition, a collective is executed on each sub-group with an identical traversing order for each rank.

When pruning a group partition space with various mesh abstractions, the primary consideration is the utilization of topology. Parallel methods like FSDP or DP typically involve collective communications across a group of inter-node devices, leading to heterogeneity of device connections. In a sub-group with unbalanced bandwidths, bottlenecks on low-bandwidth links can offset the performance benefit of high-bandwidth links. To adhere to the locality principle of double binary tree algorithm [22], group topology partitioning should leverage the high bandwidth of local connections and limit cross-node communication volume to improve communication performance. Consequently, any group partition schemes involving sub-groups with unbalanced bandwidths are excluded from the partition space. Even with balanced bandwidths, finer sub-group partition carries the risk of bandwidth under-utilization. Within each node, NCCL establishes multiple ring channels based on the detected link topology. For certain architectures, like the DGX-1, finer group partition may disrupt the well-designed link topology within a node, resulting in ring channels with significantly reduced bandwidth. Consequently, the communication cost on these

Table 3. Computation Dimension Types

Operation	Workload	Dims Types	Partition Dims
MatMul	$[A, B]x[B, C]$	OD, CD, OD	A, B, C
+, −, *, /, Dropout, ReLU, GeLU	$[A, B]$	OD, OD	A, B
SwiGLU, Softmax, LayerNorm	$[A, B]$	OD, ND	A

finely partitioned sub-groups may surpass that of direct execution on an intra-node group, rendering finer partition unnecessary. These considerations from topology utilization filter out some inefficient finer group partition schemes.

3.1.3 Workload Partition. Given a primitive and group partition scheme, the workload partition of communication triggers the partitioning of its dependent computation chain if the overlapping space with independent computation proves insufficient. In the context of LLMs training, communication can overlap with a sequential chain of dependent computations. For example, in FSDP training, the allgather operation can overlap with the subsequent MatMul and activation functions, such as GeLU and Dropout. The overall workload partition of a chain results from combining each operation's partition scheme.

Workload partitioning of communication and computation operations entails considering two aspects: the operations included in the partition and their partition dimensions. The former takes into account the chain length while the latter focuses on the compatibility of partition dimension within the chain. Partition dimensions are related to the patterns of dependent computation operations. The granularity of partitioning is closely intertwined with operation-level scheduling optimization, which is discussed in Section 3.2.1. We categorize three dimension types of computation operations in LLMs training tasks:

- Contraction dimension (CD): This category involves the contraction dimensions of operations like MatMuls and Einsums. When a computation operation is partitioned along this dimension, it necessitates an additional reduce operation to aggregate the partial results.
- None-split dimension (ND): This type requires a highly coupled computation along this dimension and is not preferable for splitting. It includes reduced dimensions of normalization functions.
- Other dimension (OD): This type encompasses the remaining workload dimensions, such as the batch dimension and none-contraction dimensions of MatMuls.

With these categories of dimension types, the possible partition space of typical computation operations is outlined in Table 3.

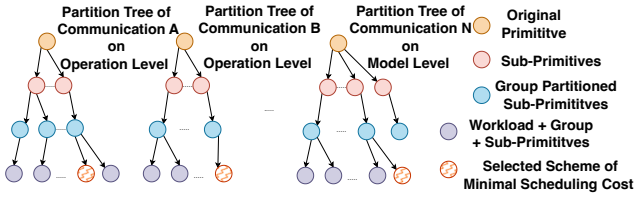

Figure 4. Communication partitioning workflow for a hybrid training task containing N communication operations.

The partition strategies for each operation in a dependent computation chain yield multiple combinations of partition dimensions. It's crucial to select feasible partition combinations that are compatible. For example, the output of a MatMul partitioned along batch-dimension (first dimension) is incompatible with a subsequent element-wise add operation partitioned along hidden size dimension (second dimension). After that, to optimize the overlapping effect, the guiding principle when choosing from various compatible partition dimensions is to identify a partition scheme that encompasses a computation chain. This chain's cost, combined with independent computation cost, should not be less than that of the communication operations intended for overlapping if possible, or maximal otherwise.

In summary, the workflow abstraction of communication partitioning is illustrated in Fig 4. Each communication in hybrid training generates a partition space in a tree structure. Each leaf node in a tree represents a feasible partition scheme. The selected partition scheme is chosen to have minimal scheduling cost. The partition strategies at each node constitute a large forest of possible partition schemes for a hybrid training task.

3.2 Hierarchical Scheduling

With a comprehensive but large partition space of Sec 3.1, optimizing the overlap scheduling of an entire graph becomes a complex task. To simplify the intricate scheduling task, we categorize the communication patterns into three hierarchical scheduling tiers: operation, layer, and model levels. The operation level focuses on the communication operation overlapping within a forward layer. It calculates the scheduling overheads of all feasible partition schemes. The partition and scheduling scheme with minimal overhead is selected as the overlapping scheme for that operation. The layer level concentrates on the optimal execution priority of a backward layer, considering the hybrid parallel method (TP & FSDP) applied. It aims to optimize the overlap of backward computations. The model level contains forward, backward, and DP weight update phases. It focuses on partition and flexible overlap scheduling for DP allreduce and the other phases. With that, a lightweight hierarchical scheduling optimization at each level contributes to a comprehensive and optimized overall scheduling scheme. The

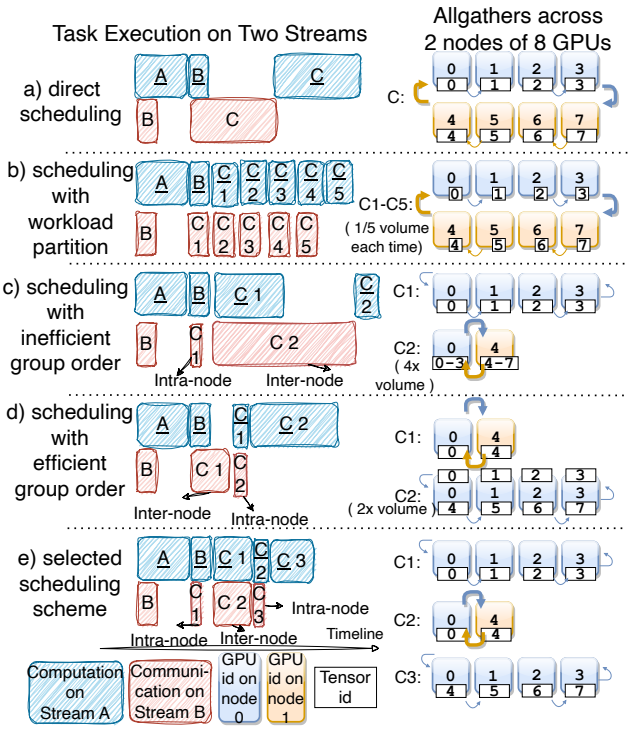

Figure 5. a) scheduling with coarse granularity results in a large idle gap. b) fine granularity scheduling introduces large overhead due to excessive workload partition. c) scheduling with a group order of large inter-node communication overheads. d) group partition with a group order of small inter-node and acceptable intra-node communication overheads. e) the selected partition and scheduling scheme with no idle gaps.

operation and layer levels scheduling focus on the forward and backward phases separately. The execution cost of optimized forward and backward phases jointly influences the model-level scheduling.

3.2.1 Operation Level. Fine-grained scheduling at the operation level aims to efficiently overlap communication and computation operations within each forward transformer layer. Each layer consists of a sequence of operations. Here, each collective is partitioned and scheduled to achieve overlap with dependent operations. This optimization process ensures that the overlap strategy for each collective is decided in a sequential order, thereby improving the efficiency of the overall layer in a greedy manner.

For each collective, different scheduling schemes based on various partitioning patterns result in distinct overall performance. Overly fine-grained workload partitioning can result in nearly full overlap of communication and computation, but it may negatively impact overall performance due to multiple small GPU kernel launches and data movement overheads, as shown in Fig 5.b. Therefore, strategies with

Algorithm 1: Operation Level Strategy

Input: comp_cost, comm_cost, cost_inter, cost_intra,
// cost of inter&intra group communication
inter_size, intra_size, //inter&intra group sizes
in_cost // cost of independent operations

```
num_chunks = MAX_CHUNKS;
if in_cost < comm_cost then
    // even workload partition
    if 0 < in_cost then
        num_chunks = min(comm_cost / in_cost, num_chunks) ;
    end
    granularity = [1 / num_chunks] x num_chunks;
    // uneven group workload partitioning
    if is_group_partitioned then
        if in_cost > cost_inter then
            // inter-node launched first
            granularity = [inter_size / world_size, 1- inter_size / world_size];
        else
            // intra-node launched first
            if comp_cost / inter_size > intra_size * cost_inter
              then
                granularity=[intra_size / world_size ,1- intra_size / world_size];
            else
                // finer workload partitioning
                granularity=[intra_size / world_size , inter_size-1 / world_size ,
                  1 - inter_size+intra_size-1 / world_size ] ;
            end
        end
    end
else
    // no need for partitioning
    num_chunks = 1, granularity = [1];
end
```

Output: granularity

larger granularity are preferable. Meanwhile, with group partitioning, it is crucial to have bandwidth-aware scheduling with an appropriate order of inter- and intra-node communication, as highlighted in the comparison of Fig 5.c and d. A proper interleaved intra- and inter-node scheduling scheme, based on group and workload partitioning, strikes a balance for maximum performance improvement in Fig 5.e. Therefore, proper interleaved execution of intra- and inter-node communication with proper partition granularity is the key issue.

With the above considerations, operation-level scheduling with appropriate partition granularity is presented in Algorithm 1. If the communication cost is lower than the independent computation cost *in_cost*, communication can be

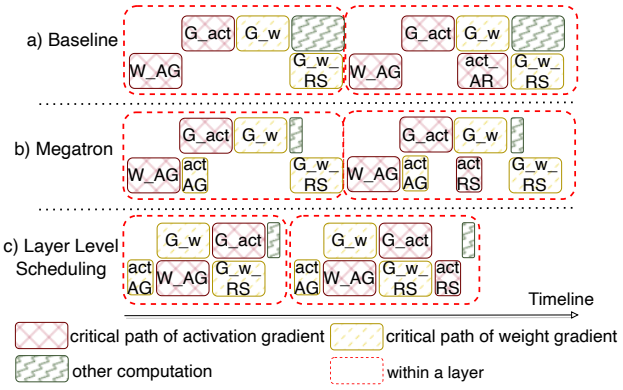

Figure 6. Layer-level Backward Scheduling of simplified TP and FSDP methods. a) OOO propagation [27] that activation computation is scheduled with higher priority. b) Megatron-LM sequence parallel method [15] that aims at overlapping activation communication. c) *Centauri* schedules critical path to maximize overlapping.

fully overlapped by independent operations and is not partitioned (line 24-25). Otherwise, partitioning is considered, and a maximum number of partitioned chunks MAX_CHUNKS is set to prevent excessive partitioning. In line 3-7, to fully utilize the independent computation, collectives are evenly partitioned according to in_cost. For group partitioning of allgather, the bandwidth-limited inter-node commutation is always the bottleneck. In line 10-12, if inter-node communication can be directly overlapped by independent operations, the allgather is partitioned into an inter-node allgather and an intra-node allgather. The output of inter-node allgather represents $\frac{inter\_size}{world\_size}$ of the total workload. The intra-node allgather gathers the output of the inter-node allgather. Therefore, the workload partition granularity is $[\frac{inter\_size}{world\_size}, 1 - \frac{inter\_size}{world\_size}]$. In line 14-20, the intra-node allgather is launched first. If inter-node communication can be overlapped by the computation of intra-node allgather output, finer workload partition is unnecessary in line 15-16. Otherwise, inter-node communication is further partitioned to minimize cross-node communication for better overlap.

3.2.2 Layer Level. Unlike the forward phase, where efficient overlap relies on partitioning, the backward phase has a natural scheduling space. The backward computation of a forward operation involves two independent parts: activation gradient computation and weight gradient computation. Traditionally, activation computation has been assigned higher scheduling priority in previous works [21, 27] since its output serves as input for the back-propagation of the preceding operation as depicted in Fig 6 a, b. Nevertheless, in hybrid parallel configurations, different execution priorities of these two parts lead to varying latency. With that, we distinguish two critical paths of activation and weight gradient computation. Different scheduling priorities of these two interleaved paths within a layer result in different costs:

$$
\begin{aligned}
T_1 &= T(W\_AG) + max(T(G\_act), T(act\_AG)) + \\
&\quad max(T(G\_w), T(act\_RS)) + max(T(G\_w\_RS), T(others)) \\
T_2 &= T(act\_AG) + max(T(G\_w), T(W\_AG)) + \\
&\quad max(T(G\_act), T(G\_w\_RS)) + T(act\_RS) + T(others)
\end{aligned} \tag{1}
$$

T_1 corresponds to the related cost of activation gradient computation scheduled with higher priority, which is adopted in a single TP scenario. The four terms in T_1 delineate the time costs associated with weight pre-fetch, overlapped computation with TP collectives, and overlapped weight gradient reduction with other operations, respectively. T_2 represents the corresponding cost when weight gradient computation is assigned a higher priority, for better overlapping in the FSDP case as shown in Fig 6 c. The four terms in T_2 encapsulate the costs related to activation communication, overlapped computation and FSDP collectives, and other computational costs. The final scheduling scheme within a backward layer hinges on the minimal cost between T_1 and T_2.

3.2.3 Model Level. Model level overlapping aims to hide the communication of gradients and weights with forward and backward phases. In a single DP scenario of *Centauri*, after the partition process, allgather is overlapped with the forward phase, while reduce-scatter is overlapped with the backward phase chunk-by-chunk in a fine-grained manner. In hybrid DP and PP training, fine-grained pipeline scheduling strategies aim to reduce pipeline bubbles, which influence the overlap of computation and communication. The overheads of a micro-batch of model chunk computation are usually smaller than the related gradients or weights communication. Launching several micro-batches of an identical model chunk releases the potential of overlapping, but activation memory consumption also grows with the number of micro-batches launched together. Memory and time cost are two factors influencing the design of the pipeline scheduling scheme. To save memory consumption, depth-first scheduling chooses a minimal number of micro-batches launched together, equal to the pipeline stage depth. It ignores the overlapping potential for end-to-end performance improvement, as depicted in Fig 7.b. Meanwhile, breadth-first scheduling goes to the other radical direction of launching all micro-batches of size *mb* per batch for overlapping with large peak memory consumption, as illustrated in Fig 7.c. The trade-off lies in the memory-minimized scheduling and overlapping maximized scheduling.

Model level overlapping scheduling of PP+DP mainly focuses on launching proper numbers of forward and backward micro-batches for maximal overlap with partitioned allreduce. It is constrained by the remaining memory capacity M of devices, apart from other memory consumption of

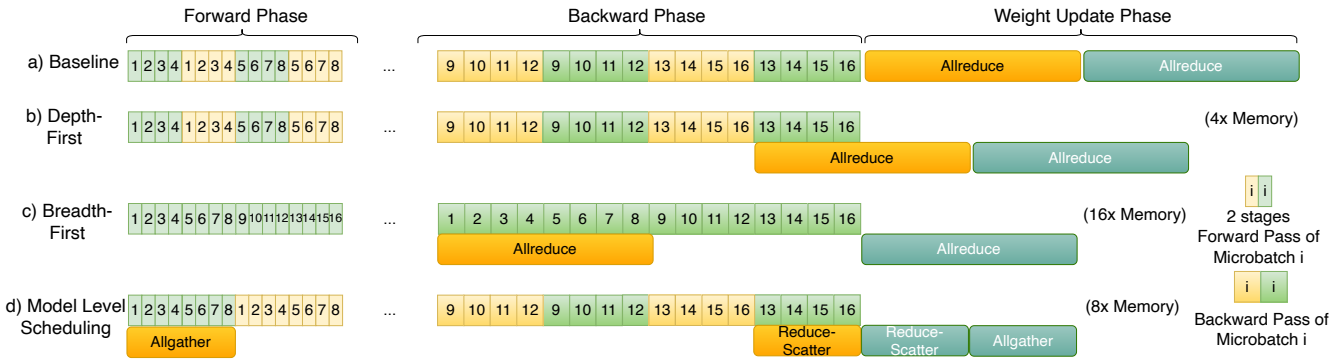

Figure 7. Model level scheduling of DP and PP: each case shows micro-batch scheduling and memory consumption of the first stage of PP group within each iteration. a) A sequential execution of forward, backward, and weight update phases, with an interleaved pipeline of 2 stages for each device, 16 micro-batches per batch, and a depth of 4. b) Direct overlap allreduce of the second stage with backward of the first stage, with minimal memory cost of 4x activation. c) All micro-batches (16) launched together for maximal overlapping, with a maximal memory cost of 16x activation. d) Minimal number (8) of micro-batches launched together for well overlapping, with medium memory cost of 8x activation.

weights and gradients buffer, etc. Due to the different computation patterns and overheads of forward and backward phases, they may demand different launched micro-batches. The overall overlapping optimization problem can be set as follows:

$$\begin{aligned} \min \quad & T = max\{C_{AG} - L_1 * C_{fw}, 0\} + \\ & \quad max\{C_{RS} - L_2 * C_{bw}, 0\} \\ s.t. \quad & L_1 * m \leq M, L_2 * m \leq M, L_1 + L_2 \leq mb \end{aligned} \tag{2}$$

$C_{op}(op \in \{AG, RS, fw, bw\})$ represents the relative operation cost of each stage, and $L1$ and $L2$ are the optimization variable numbers of micro-batches launched together in forward and backward phases for overlapping. The first term of the objective function is the un-overlapped overheads of forward computation and allgather, while the second term is of backward computation and reduce-scatter. The constraints, where m is the memory overhead of each forward execution, make the memory consumption under the remaining device capacity M.

4 Implementation

We implement *Centauri* within Megatron-LM [21], a widely-used training framework for LLMs. We encapsulate the forward and backward passes of the transformer [38] layer with different partition and scheduling strategies. For partitioning, we organize the hybrid parallel communication groups into topology-aware sub-groups and designate workload partition strategies. For the operation level, the segmented computation chain is executed with asynchronous communication in a loop style. The executed critical path is scheduled according to the designation for max overlapping. For the model level, we develop a new distributed reducer, optimizer, and pipeline scheduler. The reducer and optimizer support the dynamic overlapping between phases with the hook mechanism of PyTorch, while the pipeline scheduler elastically launches required micro-batches together. The overlapping optimization scheme is compatible with a range of hybrid parallel training methods of DP, FSDP, TP, and PP. The general hardware-agnostic idea can be applied to other LLM training frameworks.

5 Evaluation

Testbed Our experiments were conducted on two GPU clusters: Cluster A and Cluster B. In Cluster A, each node comprises 8 NVIDIA A100-80G GPUs [23] interconnected with NVSwitch [26], providing a bandwidth of 600 GB/s. The devices within Cluster A nodes are connected to the host via PCIe 3.0 x16, offering a total data transfer bandwidth of 16 GB/s. 8 devices within a Cluster A node share an InfiniBand HDR port with 200 Gb/s, with 25 Gb/s per device. On the other hand, Cluster B nodes are equipped with 4 InfiniBand HDR ports, resulting in significantly higher cross-node bandwidths of 100 Gb/s per device. Our software environment includes CUDA 11.7, NCCL 2.14 [24], Pytorch 2.0.0 [42], and Megatron-LM (git-hash e6d7e09).

Table 4. 3 LLaMA size settings

Size	Dimension	n heads	n layers	Seq len
Small	4096	32	18	2048
Medium	5120	40	15	2048
Big	6656	52	8,10,30	2048
Large	8192	64	8	2048

Model and Metric Numerous open-source large models are constructed on transformer-based architectures [1, 5, 7], among which LLaMA [35] stands out as one of the most competitive and widely adopted structures. For our evaluation, we selected four representative sizes of the LLaMA

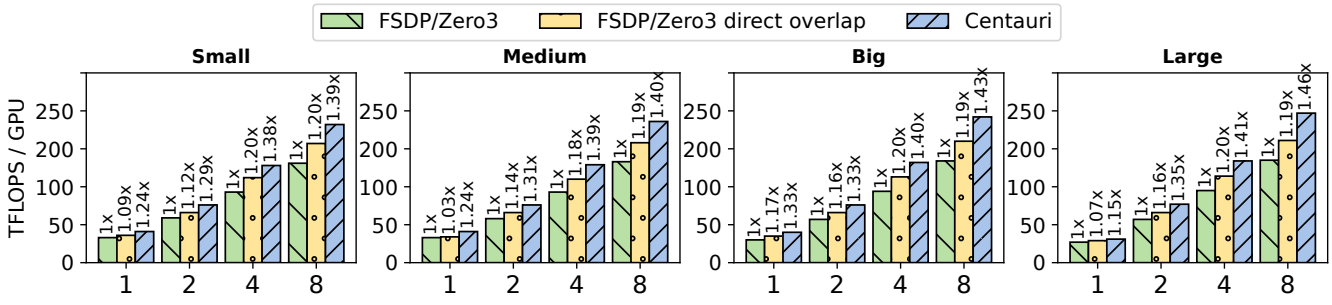

Figure 8. Performance of FSDP/Zero3 tasks on 2 nodes of Cluster A, with FSDP group size 16.

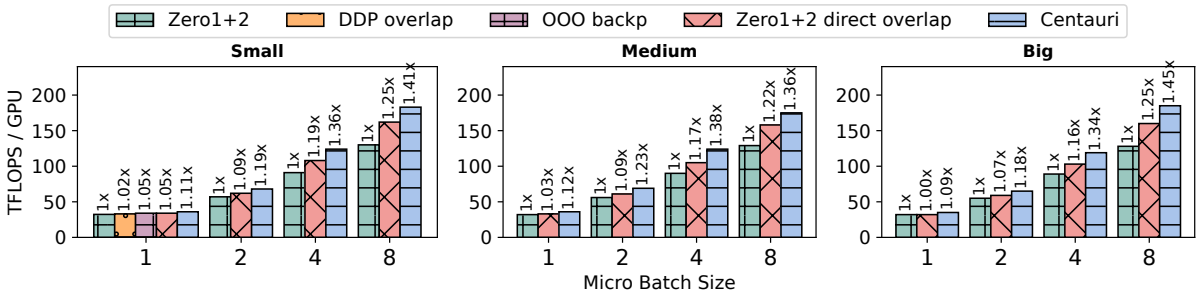

Figure 9. Performance of DP tasks on 2 nodes of Cluster A, with DP group size of 16.

model to showcase the effectiveness of *Centauri*, as outlined in Table 4. We calculated the evaluation metric by dividing the attention and MLP computation FLOP per GPU by the end-to-end time cost to gauge performance.

Baseline We compare *Centauri* with several baselines.

- **Zero1,2,3 [29]**: Zero1,2,3 partition strategies are extensively employed in LLM training. At each level, these strategies partition gradients, optimizer states, and parameters across DP groups.
- **PyTorch DDP [18]**: PyTorch's DistributedDataParallel (DDP) serves as an implementation facilitating backward computation and gradient allreduce overlapping strategies.
- **OOO back-propagation [27]**: Out-of-order backpropagation orchestrates the sequence of backward operations to optimize the overlap of gradient allreduce.
- **Megatron-LM [21]**: Megatron-LM is a widely adopted framework for high-performance hybrid training, integrating TP (SP), DP, and PP.

To ensure fairness in comparison, the fundamental methodologies of listed baselines are transposed onto Megatron-LM to neutralize the impact of unrelated frameworks' techniques.

The evaluation of *Centauri* includes four parts: performance enhancement for bottleneck single parallel tasks like FSDP and DP, an enhancement ablation study focusing on two prevalent hybrid parallel tasks (FSDP+DP and TP+PP+DP), a scalability case study of communication-demanding tasks (FSDP and FSDP+DP), and a detailed case study.

5.1 Single Parallel Performance

DP and FSDP groups typically span across nodes. Consequently, to comprehensively evaluate the combination of three partition dimensions in a single parallel method, we conduct an in-depth analysis of the performance improvements in FSDP and DP tasks.

For FSDP training, both group and workload partitioning strategies are utilized. Across various model configurations and overlapping methods, similar trends in throughput enhancement are observed with respect to batch size per device, as depicted in Fig 8. Smaller batch sizes result in computation overheads that are significantly smaller than communication latency, resulting in substantial idle periods on the computation resource and limited throughput even after enabling overlapping techniques. Therefore, the scope for performance improvement is limited, primarily depending on optimizing layer-level backward FSDP scheduling rather than finer partitioning. In contrast, for larger batch sizes with a reasonable ratio between communication and computation cost, the speedup over the FSDP/Zero3 baseline is amplified. In this scenario, the adopted partition strategy involves a looped intra-node and inter-node workload scheduling approach. Moreover, the improvements are also more

notable in larger model sizes, showing robust weak scalability concerning both data and model sizes and resulting in over 1.4× throughput enhancements.

In DP training, both primitive and workload partitioning are adopted. The allreduce operation is partitioned into reduce-scatter and allgather. The allgather phase can traverse through the element-wise optimizer operations, minimizing redundant optimizer memory usage and computation overheads. By employing a customized reducer for the partitioned allreduce, the backward phase is overlapped with the gradient reduce-scatter, and the forward phase is overlapped with weight allgather. Similar to FSDP, larger batch sizes yield higher computation granularity for overlapping and demonstrate promising throughput enhancement compared to the other baselines, as shown in Fig 9. However, in the cases of Torch DDP and OOO back-propagation, they frequently encounter out-of-memory (OOM) issues due to excessive memory consumption and the OOM cases are omitted in Fig 9.

5.2 Hybrid Parallel Performance

In hybrid parallel performance evaluation, we assess two typical hybrid parallel setups: FSDP+DP (Fig 10) and TP+PP+DP (Fig 11). In FSDP+DP training tasks, *Centauri* demonstrates throughput enhancements across various FSDP and DP group configurations compared to Zero1+2+3 baselines. In settings such as DP3+FSDP16, FSDP and DP communications span across nodes, resulting in substantial overheads. A weight tensor requires 2 pre-fetch allgather and a gradient reduce-scatter for full tensor in the FSDP setting. Meanwhile, DP requires an allreduce for weight-sharded tensors. The bandwidth latency for FSDP surpasses that of DP by a near factor of 3/2, plus the size of FSDP group. Consequently, the enhancements attributed to FSDP overlapping largely dictate the overall performance improvements. Conversely, in DP6+FSDP8 and DP12+FSDP4 settings where FSDP's communication group is deployed on intra-node devices with higher bandwidths, DP overlapping plays a more substantial role in the performance improvement. *Centauri* leverages the overlapping benefits of both methods to achieve overall high-performance results.

In TP+PP+DP training tasks, *Centauri* presents maximal throughput over Megatron-LM and other overlapping methods across various TP, PP, and DP group configurations. DP collective operations distinctly dominate the communication bottleneck compared to the minor communication overheads of intra-node TP communications and PP activation transfers (less than 5% in our settings). In configurations with larger batch sizes such as TP4+PP3+DP4+BS4, *Centauri* overlaps the segmented gradient reduce-scatter and weight allgather with a PP depth size of micro-batches, which is a direct overlapping with interleaved 1F1B scheduling. Conversely, in small batch size settings, *Centauri* increases the number of overlapping micro-batches to 6, ensuring acceptable memory consumption for fully overlapping. Overall, *Centauri* exhibits elastic micro-batch scheduling for maximal DP overlapping efficacy across different settings.

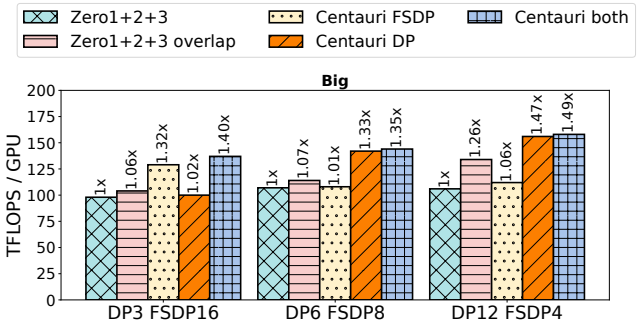

Figure 10. Performance of FSDP+DP tasks on 6 nodes of Cluster A, with a total group size of 48.

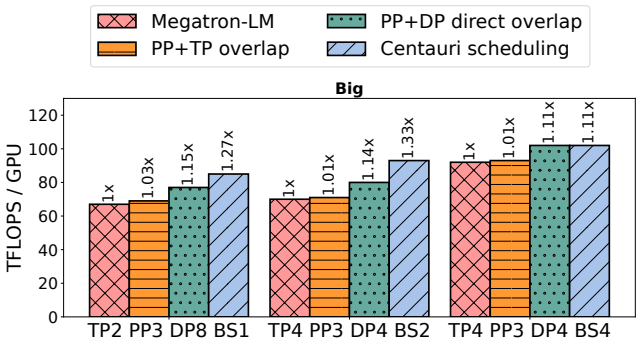

Figure 11. Performance of TP+PP+DP tasks on 6 nodes of Cluster A, with a total group size of 48 and total gradient accumulation steps of 6.

5.3 Performance Scalability

To evaluate the scalability of *Centauri*, we carry out experiments on two distinct network environments: Cluster A and B, each characterized by distinct bandwidth conditions. In Cluster A, which represents a bandwidth-limited environment, *Centauri* significantly increased the throughput of FSDP/Zero3 configurations, as depicted in Fig 13. The achieved throughput levels were comparable to those in high-performance environments (Cluster B), highlighting the bandwidth-agnostic capability of *Centauri*. Despite the limited potential for performance improvement in Cluster B, *Centauri* still managed to enhance the throughput by nearly 5% on 256 GPUs. Fig 14 illustrates the scalability of *Centauri* in communication-demanding FSDP+DP configurations. In the initial phase, all six settings experience a drop in throughput due to increased DP communication overheads. However, *Centauri* consistently maintains a high speedup compared to other baselines in Cluster A and achieves double the speedup rates of Zero1+2+3 direct overlapping in Cluster B.

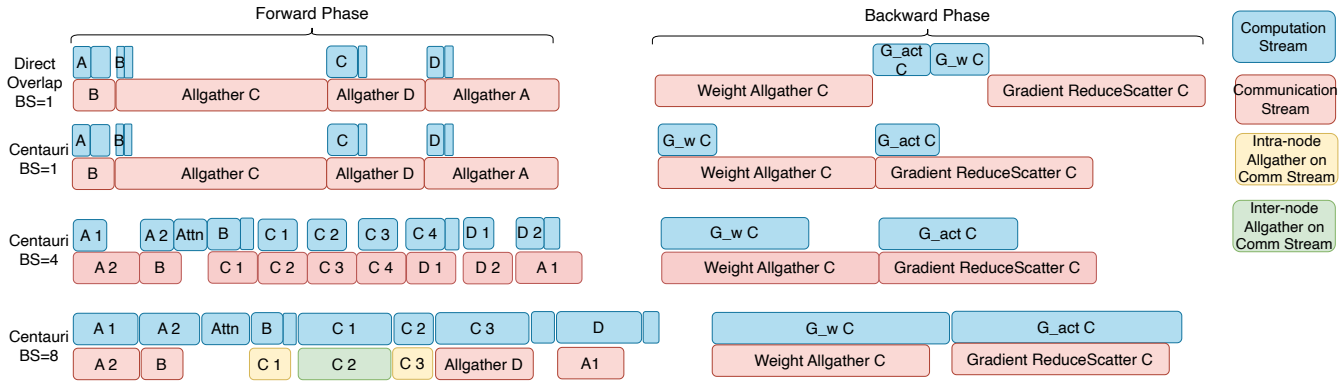

Figure 12. For a Big transformer layer in FSDP training, different communication partition strategies are employed under various batch sizes. Computation chunks, denoted as A, B, C, D, are MatMuls with large weight inputs. Communication chunks involve allgathers responsible for gathering the shared weights.

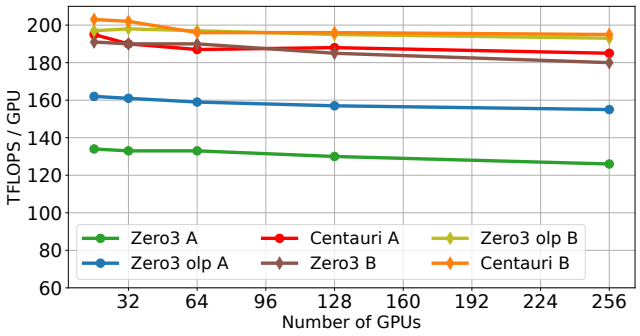

Figure 13. Scalability of FSDP/Zero3 on Cluster A&B

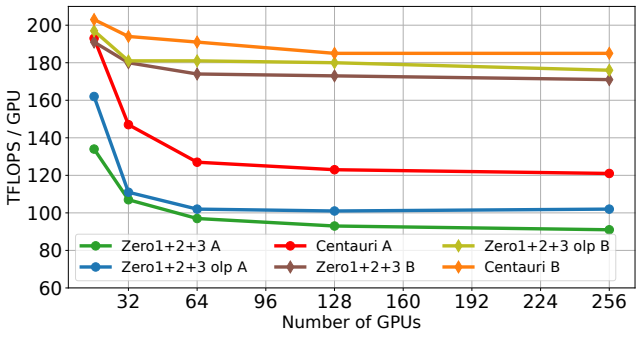

Figure 14. Scalability of FSDP16+DP on Cluster A&B

5.4 FSDP Case Study

FSDP tasks are overlap-demanding, given their frequent and large volumes of communication. Speedup upper bound is determined by several factors, including the communication cost ratio α, data dependencies, and hardware capacity. We conduct a case study using FSDP to explore the impact of different communication cost ratios. The theoretical speedup upper bound is given by $\min\{\frac{1}{\alpha}, \frac{1}{1-\alpha}\}$. A speedup of $\frac{1}{\alpha}$ indicates that computation can be fully overlapped by communication when α is larger than 0.5. On the other hand, a speedup of $\frac{1}{1-\alpha}$ suggests that communication is fully overlapped when α is smaller than 0.5. Specifically, we analyze the forward and backward phases of a transformer layer with varying batch sizes, as illustrated in Fig 12. For a small batch size (BS=1), α is large, limiting the achievable speedup. As the batch size increases to 4, α decreases, allowing for a larger speedup. The overlap patterns for the operators within a transformer layer also become more diverse. As the computation ratio exceeds the communication ratio with a batch size of 8, full overlap of communication by computation is enabled with different partition and scheduling patterns.

6 Related Work

Communication Optimization To enhance training performance, various studies have aimed to reduce bottleneck communication overheads. Some auto-parallelism works output optimal parallel schemes to minimize communication overheads [12, 43]. Additionally, high-performance collective communication libraries [8, 24, 34] focus on optimizing performance based on the characteristics of underlying hardware environments. Recent implementations of hierarchical allreduce [3, 13, 36, 37] have highlighted the importance of efficient bandwidths utilization in heterogeneous environments. Several frameworks [2, 4, 31, 39] generate specific communication kernels tailored to complex network environments. Despite these customized optimizations, communication overheads still constitute a significant portion of the total cost due to the exponential scaling-up of training networks.

Communication Overlapping Some frameworks [9, 17, 18, 20, 27, 28, 41, 42] aim to optimize the overlap scheduling of a single parallel method in a coarse-grained manner, which

may under-utilize hardware resources. Conversely, some sophisticated compiler-style works [11, 40] generate code for fine-grained overlapping between two kernels, but they may miss opportunities for graph-level scheduling. Additionally, the kernel launch and data movement overheads associated with overly fine-grained kernel partitioning may counteract the benefits of overlapping. Recent studies have explored concurrent communication execution across multi-streams [20, 32], which can also enhance training performance. However, those works lack a comprehensive partition abstraction and systematic scheduling for efficient overlapping schemes in hybrid parallel LLMs training.

7 Conclusions

We put forward *Centauri*, a framework for efficient communication and computation overlap in LLMs training tasks. It includes efficient communication partition of three dimensions: primitive, group, and workload, and a hierarchical scheduling scheme of three tiers: operation, layer, and model. We show that *Centauri* significantly improves the performance of LLMs training.

In contrast to the static and intensive computation pattern observed in LLMs training, parallel LLMs inference faces challenges associated with small-volume but frequent communication overheads. In the future, we will explore communication challenges of parallel inference with two aspects: 1) coalesce small-volume communication and parallelize tasks with low-frequency communication patterns, 2) more fine-grained communication scheduling that aligns with the warp-level computation scheduling capabilities of GPUs within the NCCL framework.

8 Acknowledgments

We thank the anonymous reviewers and our shepherd, Abhinav Jangda, for their valuable insights and feedback.

References

[1] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

[2] Zixian Cai, Zhengyang Liu, Saeed Maleki, Madanlal Musuvathi, Todd Mytkowicz, Jacob Nelson, and Olli Saarikivi. Synthesizing optimal collective algorithms. In *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '21, page 62–75, New York, NY, USA, 2021. Association for Computing Machinery.

[3] C. Chen, M. Li, and C. Yang. bbtopk: Bandwidth-aware sparse allreduce with blocked sparsification for efficient distributed training. In *2023 IEEE 43rd International Conference on Distributed Computing Systems (ICDCS)*, pages 1–11, Los Alamitos, CA, USA, jul 2023. IEEE Computer Society.

[4] Meghan Cowan, Saeed Maleki, Madan Musuvathi, Olli Saarikivi, and Yifan Xiong. Gc3: An optimizing compiler for gpu collective communication. 2022.

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[6] Shiqing Fan, Yi Rong, Chen Meng, Zongyan Cao, Siyu Wang, Zhen Zheng, Chuan Wu, Guoping Long, Jun Yang, Lixue Xia, Lansong Diao, Xiaoyong Liu, and Wei Lin. Dapple: A pipelined data parallel approach for training large models. In *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '21, page 431–445, New York, NY, USA, 2021. Association for Computing Machinery.

[7] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *CoRR*, abs/2101.03961, 2021.

[8] Richard L. Graham, Galen M. Shipman, Brian W. Barrett, Ralph H. Castain, George Bosilca, and Andrew Lumsdaine. Open mpi: A high-performance, heterogeneous mpi. In *2006 IEEE International Conference on Cluster Computing*, pages 1–9, 2006.

[9] Sayed Hadi Hashemi, Sangeetha Abdu Jyothi, and Roy Campbell. Tictac: Accelerating distributed deep learning with communication scheduling. In A. Talwalkar, V. Smith, and M. Zaharia, editors, *Proceedings of Machine Learning and Systems*, volume 1, pages 418–430, 2019.

[10] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Mia Xu Chen, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, Yonghui Wu, and Zhifeng Chen. *GPipe: Efficient Training of Giant Neural Networks Using Pipeline Parallelism*. Curran Associates Inc., Red Hook, NY, USA, 2019.

[11] Abhinav Jangda, Jun Huang, Guodong Liu, Amir Hossein Nodehi Sabet, Saeed Maleki, Youshan Miao, Madanlal Musuvathi, Todd Mytkowicz, and Olli Saarikivi. Breaking the computation and communication abstraction barrier in distributed machine learning workloads. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '22, page 402–416, New York, NY, USA, 2022. Association for Computing Machinery.

[12] Zhihao Jia, Matei Zaharia, and Alex Aiken. Beyond data and model parallelism for deep neural networks. *CoRR*, abs/1807.05358, 2018.

[13] Yimin Jiang, Yibo Zhu, Chang Lan, Bairen Yi, Yong Cui, and Chuanxiong Guo. A unified architecture for accelerating distributed DNN training in heterogeneous GPU/CPU clusters. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 463–479. USENIX Association, November 2020.

[14] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *CoRR*, abs/2001.08361, 2020.

[15] Vijay Korthikanti, Jared Casper, Sangkug Lym, Lawrence McAfee, Michael Andersch, Mohammad Shoeybi, and Bryan Catanzaro. Reducing activation recomputation in large transformer models, 2022.

[16] Woosuk Kwon, Gyeong-In Yu, Eunji Jeong, and Byung-Gon Chun. Nimble: Lightweight and parallel gpu task scheduling for deep learning. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS'20, Red Hook, NY, USA, 2020. Curran Associates Inc.

[17] Joel Lamy-Poirier. Breadth-first pipeline parallelism, 2023.
[18] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, and Soumith Chintala. Pytorch distributed: Experiences on accelerating data parallel training. *CoRR*, abs/2006.15704, 2020.
[19] 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*, SC '21, New York, NY, USA, 2021. Association for Computing Machinery.
[20] Kshiteej Mahajan, Ching-Hsiang Chu, Srinivas Sridharan, and Aditya Akella. Better together: Jointly optimizing ML collective scheduling and execution planning using SYNDICATE. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 809–824, Boston, MA, April 2023. USENIX Association.
[21] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei 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*, SC '21, New York, NY, USA, 2021. Association for Computing Machinery.
[22] NVDIA. Massively scale your deep learning training with nccl 2.4. https://developer.nvidia.com/blog/massively-scale-deep-learning-training-nccl-2-4/.
[23] NVIDIA. Nvidia dgx a100 system architecture. https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/dgx-a100/dgxa100-system-architecture-white-paper.pdf, 2020.
[24] NVIDIA. NVIDIA Collective Communication Library (NCCL) Documentation. https://docs.nvidia.com/deeplearning/nccl/user-guide/docs/index.html, 2022.
[25] NVIDIA. NVLINK. https://www.nvidia.com/en-us/data-center/nvlink/, 2022.
[26] NVIDIA. NVSWITCH: The world's highest-bandwidth on-node switch. https://images.nvidia.com/content/pdf/nvswitch-technical-overview.pdf, 2022.
[27] Hyungjun Oh, Junyeol Lee, Hyeongju Kim, and Jiwon Seo. Out-of-order backprop: An effective scheduling technique for deep learning. In *Proceedings of the Seventeenth European Conference on Computer Systems*, EuroSys '22, page 435–452, New York, NY, USA, 2022. Association for Computing Machinery.
[28] 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*, SOSP '19, page 16–29, New York, NY, USA, 2019. Association for Computing Machinery.
[29] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '20. IEEE Press, 2020.
[30] Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, page 3505–3506, New York, NY, USA, 2020. Association for Computing Machinery.
[31] Aashaka Shah, Vijay Chidambaram, Meghan Cowan, Saeed Maleki, Madan Musuvathi, Todd Mytkowicz, Jacob Nelson, Olli Saarikivi, and Rachee Singh. Synthesizing collective communication algorithms for heterogeneous networks with taccl. arXiv preprint, 2021.
[32] Shaohuai Shi, Xiaowen Chu, and Bo Li. Exploiting simultaneous communications to accelerate data parallel distributed deep learning. *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, pages 1–10, 2021.
[33] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *CoRR*, abs/1909.08053, 2019.
[34] Rajeev Thakur and William D Gropp. Improving the performance of collective operations in mpich. In *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting*, pages 257–267. Springer, 2003.
[35] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023.
[36] Thao Nguyen Truong, Mohamed Wahib, and Ryousei Takano. Efficient mpi-allreduce for large-scale deep learning on gpu-clusters. *Concurrency and Computation: Practice and Experience*, 33, 12 2019.
[37] Yuichiro Ueno and Rio Yokota. Exhaustive study of hierarchical allreduce patterns for large messages between gpus. In *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 430–439, 2019.
[38] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.
[39] Guanhua Wang, Shivaram Venkataraman, Amar Phanishayee, Jorgen Thelin, Nikhil R. Devanur, and Ion Stoica. Blink: Fast and generic collectives for distributed ML. *CoRR*, abs/1910.04940, 2019.
[40] Shibo Wang, Jinliang Wei, Amit Sabne, Andy Davis, Berkin Ilbeyi, Blake Hechtman, Dehao Chen, Karthik Srinivasa Murthy, Marcello Maggioni, Qiao Zhang, Sameer Kumar, Tongfei Guo, Yuanzhong Xu, and Zongwei 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*, ASPLOS 2023, page 93–106, New York, NY, USA, 2022. Association for Computing Machinery.
[41] L. Zhang, S. Shi, X. Chu, W. Wang, B. Li, and C. Liu. Dear: Accelerating distributed deep learning with fine-grained all-reduce pipelining. In *2023 IEEE 43rd International Conference on Distributed Computing Systems (ICDCS)*, pages 142–153, Los Alamitos, CA, USA, jul 2023. IEEE Computer Society.
[42] Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, Alban Desmaison, Can Balioglu, Pritam Damania, Bernard Nguyen, Geeta Chauhan, Yuchen Hao, Ajit Mathews, and Shen Li. Pytorch fsdp: Experiences on scaling fully sharded data parallel. *Proc. VLDB Endow.*, 16(12):3848–3860, aug 2023.
[43] Lianmin Zheng, Zhuohan Li, Hao Zhang, Yonghao Zhuang, Zhifeng Chen, Yanping Huang, Yida Wang, Yuanzhong Xu, Danyang Zhuo, Eric P. Xing, Joseph E. Gonzalez, and Ion Stoica. 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, Carlsbad, CA, July 2022. USENIX Association.