IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 13 April 2025, accepted 2 July 2025. Date of publication 00 xxxx 0000, date of current version 00 xxxx 0000.

Digital Object Identifier 10.1109/ACCESS.2025.3587387

AsymGroup: Asymmetric Grouping and Communication Optimization for 2D Tensor Parallelism in LLM Inference

KI TAE KIM, SEOK-JU IM, AND EUI-YOUNG CHUNG, (Member, IEEE)
Department of Electrical and Electronic Engineering, Yonsei University, Seoul 03722, South Korea

Corresponding author: Eui-Young Chung (eychung@yonsei.ac.kr)

This work was supported in part by the National Research Foundation of Korea(NRF) grant funded by the Korea government [Ministry of Science and Information and Communication Technology (MSIT)] (No.RS-2024-00405495, Plug&Play (P&P) Chiplet Integration research center), in part by the Technology Innovation Program (No. RS-2024-00420541, 2410000802) funded by the Ministry of Trade, Industry and Energy of Korea, in part by Institute of Information & communications Technology Planning & Evaluation(IITP) grant funded by the Korea government (MSIT) (No.2022-0-00050, Development of Processing-in-Memory (PIM) Computing Architecture based on Data-Flow), and in part by the Samsung Electronics Company, Ltd., Hwaseong, Korea.

ABSTRACT Recent advances in Large Language Models (LLMs), such as GPT and LLaMA, have demonstrated remarkable capabilities across a wide array of natural language processing tasks. Despite these successes, efficient inference at scale remains challenging, particularly in heterogeneous computing environments characterized by variations in GPU counts, and computational capacities across nodes. Conventional tensor parallelism approaches typically assume homogeneous hardware, resulting in significant performance degradation under asymmetric conditions. To address this challenge, we propose AsymGroup, which is an asymmetric 2D tensor parallelism framework that dynamically constructs groups of uneven sizes based on node- and GPU-level metrics. AsymGroup proportionally allocates both computational and communication workloads according to individual device capabilities. In addition, we introduce a formal communication cost model to accurately quantify bottlenecks and a group optimization algorithm to systematically determine efficient group configurations. Experimental evaluations demonstrate that AsymGroup reduces inference latency by up to 26.3% and communication overhead by up to 33.4% compared to state-of-the-art frameworks, while also maintaining competitive performance under symmetric conditions.

INDEX TERMS Large language models, inference optimization, tensor parallelism, collective communication.

I. INTRODUCTION

Large Language Models (LLMs), such as GPT [1] and LLaMA [2], have become central to natural language processing, achieving state-of-the-art performance across diverse tasks. However, these models typically consist of hundreds of billions to trillions of parameters, generating substantial intermediate activations during each inference step that must be retained in memory. Consequently, executing inference on a single GPU becomes impractical because of limitations in memory capacity and computational throughput, necessitating distributed processing across multiple GPUs [1], [3], [4], [5].

Various model parallelism strategies have been developed to address these challenges, notably pipeline parallelism [6] and tensor parallelism [7]. Among these, tensor parallelism has gained widespread adoption owing to the favorable trade-off between memory efficiency and execution speed [6], [7], [8]. However, tensor parallelism requires frequent synchronization among GPUs and is typically implemented through collective communication operations such as All-Reduce. These operations introduce significant

The associate editor coordinating the review of this manuscript and approving it for publication was Zhenliang Zhang.

overhead, particularly in multi-GPU systems with limited interconnect bandwidth [9], [10], [11]. As the number of GPUs scales, the communication overhead increases disproportionately, emerging as a critical scalability bottleneck.

To mitigate this issue, two-dimensional (2D) tensor parallelism has been introduced as a scalable alternative. Unlike 1D tensor parallelism, which confines computation and communication along a single axis, 2D tensor parallelism distributes operations across both the row and column dimensions. This dual-axis partitioning effectively balances the communication traffic and reduces both the communication volume and latency. Frameworks such as Optimus [12] and AutoDDL [13] have adopted 2D tensor parallelism to enhance communication efficiency during LLM inference. In particular, AutoDDL [13] achieves performance improvements by efficiently coordinating internode and intranode communication.

Despite these advantages, most existing 2D tensor parallel frameworks assume uniform group sizes ($p = p_0 \times p_1$) and homogeneous GPU configurations [12], [13], [14]. Furthermore, several approaches primarily target encoder-style architectures that pass both activations and weights, making them difficult to apply to decoder-based inference, where only the activation must be exchanged [12], [14]. In addition, asymmetry is prevalent in real-world scenarios, particularly in cloud and high-performance computing (HPC) environments [15], [16]. Nodes often differ in GPU count, memory capacity, and computational performance, creating heterogeneous configurations that pose substantial challenges to conventional 2D tensor parallelism schemes.

To address these limitations, we propose AsymGroup, a novel tensor parallelism framework specifically designed for heterogeneous GPU environments. AsymGroup partitions both computational and communication tasks based on the capabilities of individual GPUs by considering factors such as computing and memory capacity. Additionally, it relaxes the symmetry constraints typically imposed on collective communication, enabling flexible data exchange patterns that align better with hardware disparities, thus enhancing scalability. We further introduce a formal communication cost model that quantifies bottlenecks within the AsymGroup framework and develop an optimization algorithm to systematically determine efficient group configurations. The main contributions of this study are as follows.

1) We propose AsymGroup, a framework that supports asymmetric grouping and communication-aware optimization for 2D tensor parallelism in LLM inference.
2) We formulated a detailed communication cost model and introduced an optimization algorithm to construct groups that minimized both intergroup and intragroup bottlenecks.
3) We experimentally validated AsymGroup on large-scale language models, demonstrating significant improvements in inference latency and communication efficiency compared to existing state-of-the-art methods.

Our experiments confirm that AsymGroup outperforms existing tensor parallel frameworks in terms of both latency and communication efficiency under heterogeneous GPU configurations. Specifically, it achieved up to 26.3% lower inference latency and up to 33.4% reduced communication overhead on large-scale models such as GPT-175B and MT-530B [3]. Furthermore, AsymGroup maintains a competitive performance under balanced (symmetric) configurations, ensuring broad applicability without compromising efficiency.

The remainder of this paper is organized as follows. Section II introduces the background of transformer-based LLM inference and tensor parallelism. Section III discusses the limitations of the existing 2D tensor parallelism in heterogeneous environments and motivates the need for AsymGroup. Section IV surveys the related work. Section V presents the design and implementation of the proposed framework, and Section VI provides the evaluation results under various system configurations. Finally, Section VII concludes the paper and discusses potential directions for future work.

II. BACKGROUND

A. TRANSFORMER DECODER BLOCK AND LLM INFERENCE FLOW

Transformer architectures serve as foundational building blocks for LLMs such as GPT and LLaMA [1], [2], [8]. As depicted in Fig. 1, each transformer decoder block consists of fully connected (linear), multi-head attention, and layer normalization layers.

During LLM inference, the decoder blocks iteratively generate one token until an end-of-sequence token is generated. Linear layers typically execute general matrix-vector multiplication (GEMV) operations, which can benefit from batch processing when multiple input tokens are grouped. Conversely, the attention mechanism involves pairwise interactions among the query (Q), key (K), and value (V) vectors, limiting its amenability to batching. In addition, attention complexity scales with sequence length.

The key and value vectors are stored cumulatively in a key-value (KV) cache across all generated tokens. For instance, in GPT-175B with a sequence length of 2048, the KV cache alone may consume over 18 GB of memory, surpassing the memory capacity of a single GPU. This memory constraint restricts batch size during inference, leading to increased latency. Thus, model parallelism is essential for distributing both computational tasks and memory-intensive activations across multiple GPUs.

B. TENSOR PARALLELISM

Tensor parallelism has emerged as an indispensable technique for distributing LLMs across multiple GPUs, thereby significantly benefiting from training and inference. By partitioning the model parameters and intermediate activations among devices, tensor parallelism alleviates memory bottlenecks

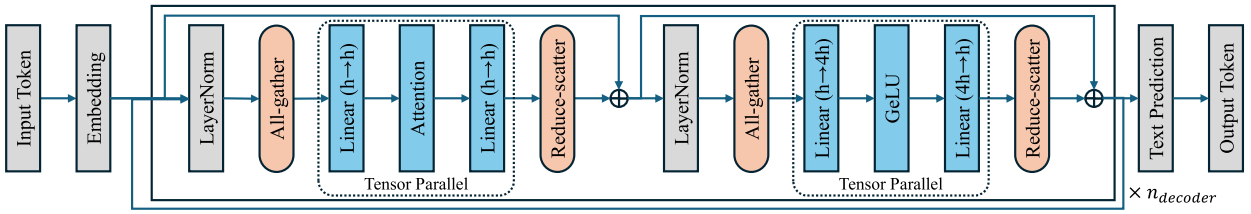

FIGURE 1. Structure of a transformer decoder block illustrating linear layers, attention layers, layer normalization steps, and the collective communication operations (all-gather and reduce-scatter) used in tensor parallelism.

and enhances computational throughput through parallel execution [7], [8].

However, tensor parallelism inherently necessitates synchronization across GPUs because intermediate computation results must be exchanged after each partitioned step. Synchronization is typically implemented through collective communication. The overhead introduced by these operations escalates with increasing data volume and is further exacerbated by the limited interconnect bandwidth, particularly across nodes.

Ideally, uniform high-bandwidth interconnects and balanced GPU counts enable ring-based or hierarchical collective algorithms to minimize communication costs. However, in practical deployments, the internode bandwidth typically falls below the intranode bandwidth and hardware heterogeneity introduces additional communication imbalances, leading to pronounced bottlenecks. Consequently, the overall performance of tensor parallelism is often constrained by the network bandwidth and collective communication algorithms.

C. COLLECTIVE COMMUNICATION

Among the collective operations used in tensor parallelism, All-Reduce plays a critical role by aggregating and distributing data across GPUs [8], [17], [18]. All-Reduce typically consists of two sub-operations: reduce-scatter, which performs distributed reduction across GPUs, and all-gather, which broadcasts the aggregated results back to all GPUs. This two-step collective operation may occur multiple times per decoder block during inference.

Communication overhead primarily depends on the volume and frequency of data transfers. A simple linear communication model can describe this cost [9], [13], [19], [20], where α represents the fixed latency per communication step, β denotes the inverse bandwidth (communication cost per byte), m is the data volume per step, and c is the number of communication steps. Thus, the total communication volume is thus $V = m \times c$, and the overall communication cost is $\alpha c + \beta V$. Given that activation data usually dominate overall communication latency, this paper primarily focuses on reducing the total communication volume V.

D. 2D TENSOR PARALLELISM AND COMMUNICATION OPTIMIZATION

1D tensor parallel methods, such as Megatron-LM, partition model parameters along a single dimension (e.g., rows or columns of linear layer weights). Although 1D approaches are straightforward and efficient for parameter distribution, significant communication overhead arises because of the frequent All-Reduce operations required to synchronize activations following partial computations.

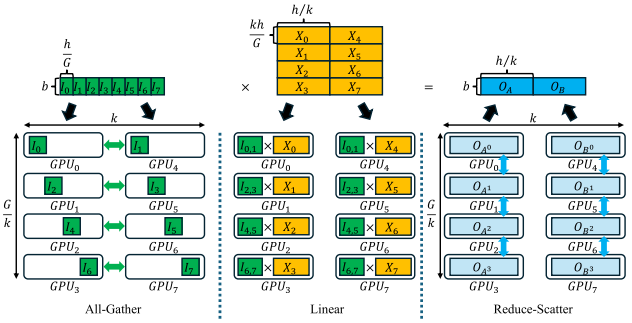

FIGURE 2. Illustration of 2D tensor parallelism, in which model weights and inputs are partitioned along row and column axes across a $k \times (G/k)$ GPU grid, effectively reducing the per-GPU communication overhead.

To address these limitations, recent frameworks including Optimus [12] and AutoDDL [13] have introduced 2D tensor parallelism schemes. In the 2D approach, the model parameters and activations are partitioned simultaneously along the row and column axes, enabling each GPU to handle a smaller subset of data arranged in a two-dimensional grid. By distributing computation and communication across two dimensions, the per-device communication volume is reduced, thereby alleviating bottlenecks.

Fig. 2 conceptually illustrates the 2D tensor parallel pipeline, where G GPUs are organized into a $k \times (G/k)$ grid. Consider the linear operation $O = IX$, where input I is divided column-wise and weight X is partitioned along both rows and columns. During computation, an all-gather among GPUs in each row ensures that devices such as GPU_0 and GPU_4 receive identical input fragments. Each then performs a linear operation using its assigned weight sub-block (e.g., X_0 for GPU_0, X_4 for GPU_4). Afterward, a reduce-scatter among GPUs in the same column, such as those using X_0, X_1, X_2, X_3, assembles the corresponding output O for each device. Let bh denote the activation data size. In traditional 1D tensor parallelism, the total communication volume can be expressed as $bh \cdot (G-1)/G$. In constrast, for 2D tensor parallel approaches, the communication volume can be formally expressed as (1). This formulation highlights how

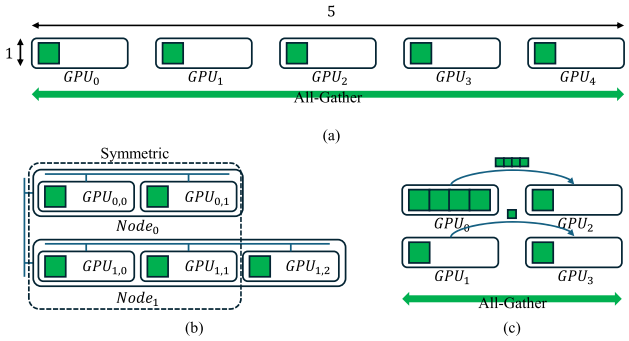

FIGURE 3. Challenges posed by asymmetric GPU distributions in 2D tensor parallelism. In (a) and (b), certain nodes cannot be evenly factored or have varying GPU counts, causing unassigned or idle GPUs and communication imbalances. In (c), GPUs with differing performance characteristics coexist within a single node, introducing potential bottlenecks when uniform partitioning is attempted.

2D tensor parallelism significantly reduces communication volume compared to the 1D tensor parallelism.

$$V_{\text{all-gather}} = \frac{bh}{G}(k-1)$$
$$V_{\text{reduce-scatter}} = \frac{bh}{G}\left(\frac{G}{k}-1\right)$$
$$V_{\text{2D}} = V_{\text{all-gather}} + V_{\text{reduce-scatter}} \quad (1)$$

III. MOTIVATION

State-of-the-art inference for LLMs typically leverages transformer architectures and tensor parallelism (Section II) to distribute massive models across multiple GPUs. In distributed environments, it is well established that inference performance is significantly constrained by communication overhead, particularly because of collective operations such as All-Reduce [9], [10], [11]. Although 2D tensor parallelism has demonstrated substantial reductions in communication volume compared to traditional 1D approaches such as Megatron-LM [7], [8] and has been effectively integrated into frameworks such as Optimus [12] and AutoDDL [13], its practical performance often degrades in real-world deployments. This deterioration primarily arises from hardware heterogeneity.

In modern cloud and HPC environments, hardware asymmetry is prevalent [15], [16]. Differences in GPU counts per node, variations in GPU computational capabilities, and non-uniform internode network bandwidth pose significant challenges, weakening the assumptions underlying conventional 2D tensor parallelism. This section investigates specific issues arising under such heterogeneous conditions and explains why standard 2D parallelization strategies fall short given LLM-specific workloads and resource constraints.

A. LIMITATIONS OF 2D TENSOR PARALLELISM IN ASYMMETRIC SYSTEMS

The first key challenge is the uneven distribution of GPUs across the nodes. Traditional 2D tensor parallelism generally presupposes a uniform mesh arrangement of GPUs in $n \times g$ grid (where n is the node count and g is the GPUs per node). However, when nodes have differing numbers of GPUs or GPU counts that do not factor cleanly into a grid, straightforward row–column partitioning becomes problematic. As illustrated in Fig. 3(a) and (b), some nodes may have unused or surplus GPUs that cannot be effectively integrated into the communication mesh, thereby significantly undermining resource utilization and diminishing the overall efficiency of computation and communication operations.

The second issue emerges from heterogeneity in GPU memory capacities and computational throughput. Many existing 2D tensor-parallel strategies assume uniform GPU specifications across nodes to streamline communication patterns, in practice, as shown in Fig. 3(c), nodes often integrate GPUs with varying computational power or memory sizes, causing specific nodes to become critical communication bottlenecks. In addition, certain all-reduce optimization methods depend heavily on evenly partitioned data to fully exploit the available bandwidth. However, with LLM workloads, where compute-intensive attention layers alternate with linear layers, the resulting uneven workload distribution can lead to significant performance degradation if GPU capacities and memory resources vary significantly among nodes.

B. THE NEED FOR AsymGroup

As highlighted above, the assumptions underpinning the traditional 2D tensor parallelism do not hold in heterogeneous computing environments. Fixed group structures optimized for homogeneous hardware frequently result in performance degradation when deployed in systems characterized by uneven resource allocation. Despite these drawbacks, the inherent advantages of 2D tensor parallelism, particularly its ability to parallelize communication and distribute data effectively—remain compelling. Thus, a more flexible and generalized structural approach is necessary to fully exploit the benefits of heterogeneous deployments.

To address these limitations, we propose AsymGroup, a generalized tensor-parallel framework designed explicitly for heterogeneous GPU environments. AsymGroup dynamically constructs capability-aware groups based on each node's GPU count, computational throughput, and memory capacity, thereby enabling proportionally balanced computational and communication workloads across the entire system. Furthermore, we introduced a formal communication cost model and an optimization algorithm to automate effective group formation and systematically mitigate bottlenecks in large-scale deployments.

IV. RELATED WORK

Significant research efforts have been dedicated to addressing the communication overhead inherent in distributed training and inference of LLMs. These efforts span multiple dimensions, including tensor parallelism, communication

scheduling, and heterogeneity-aware collective communications. This section provides an overview of representative approaches, highlighting their limitations in heterogeneous GPU environments and motivating the need for more flexible and capability-aware tensor parallel frameworks.

A. 1D TENSOR PARALLELISM

Megatron-LM [7], [8] pioneered 1D tensor parallelism by partitioning model parameters along a single axis (row or column), thereby enabling scalable execution across multiple GPUs. In addition, sequence parallelism has been proposed to enhance memory efficiency further. However, these approaches depend heavily on ring-based collective communication primitives (e.g., NCCL), which can cause substantial bottlenecks, particularly under limited internode bandwidth conditions.

B. COMMUNICATION SCHEDULING AND TOPOLOGY AWARENESS

Another line of research has focused on strategies for overlapping communication and computation [21], [22], [23], as well as employing topology-aware scheduling techniques to maximize interconnect utilization [22]. Approaches leveraging hierarchical collective communication and link augmentation for mesh or torus network topologies have also been investigated [24], [25]. However, such techniques often make low-level hardware-specific assumptions or require substantial software modifications, thereby limiting their portability and applicability across diverse computing infrastructures.

C. HETEROGENEITY-AWARE COLLECTIVE COMMUNICATION

To accommodate hardware diversity better, several studies have developed collective communication algorithms that are explicitly tailored for heterogeneous systems. FlexReduce [15], for instance, preserves uniform data partitioning but dynamically reorders communication sequences based on individual device capabilities. Similarly, TACOS [16] adapted communication patterns according to the underlying hardware topology. While these methods offer improved robustness in heterogeneous environments, they fundamentally do not address the underlying communication-computation imbalance nor alter data partitioning strategies within tensor-parallel schemes. Hence, their benefits remain limited under severe hardware asymmetry.

D. 2D TENSOR PARALLELISM AND ITS STRUCTURAL CONSTRAINTS

Recently, 2D tensor parallelism has emerged as a promising technique for further reducing communication overhead by partitioning model weights and activations across two dimensions [12], [13], [14]. Compared to 1D methods, 2D parallelism significantly mitigates communication costs and improves scalability. However, existing 2D frameworks typically assume symmetric and uniform hardware configurations, including balanced GPU counts and equal compute/network bandwidths. Such rigid assumptions restrict their applicability to realistic deployments characterized by uneven GPU distributions and hardware imbalances. In asymmetric environments, static and uniform partitioning often lead to severe communication bottlenecks and resource underutilization. Thus, there is a clear need for a more general tensor parallelism framework that can accommodate heterogeneous device capabilities without sacrificing the benefits of 2D parallelism.

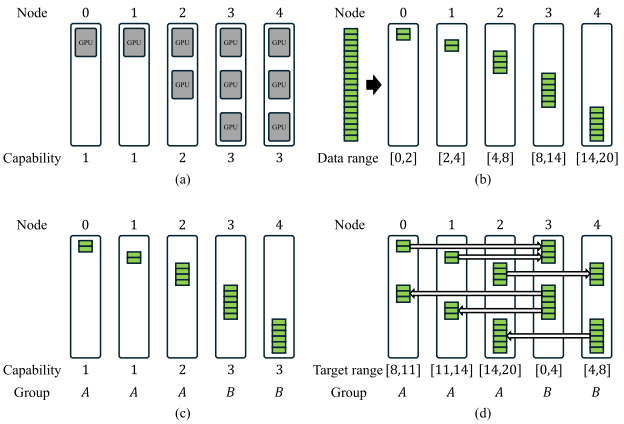

FIGURE 4. Overview of AsymGroup capability abstraction and asymmetric group formation.(a) Distribution of GPUs across nodes. (b) Proportional partitioning of input data based on node capabilities. (c) Capability-driven asymmetric grouping of nodes. (d) Capability-aware point-to-point communication between groups.

In this study, we directly addressed this challenge by introducing AsymGroup, a flexible and capability-aware tensor parallelism framework suitable for heterogeneous computing environments, as detailed in the following section.

V. AsymGroup

This section introduces AsymGroup, which is a novel tensor-parallel framework designed for scalable inference in heterogeneous GPU environments. Unlike conventional 2D tensor-parallel methods, which typically assume uniform GPU counts per node and balanced communication paths, AsymGroup explicitly accommodates both node- and GPU-level asymmetries by (i) abstracting heterogeneous hardware into logical capability units, and (ii) proportionally distributing computational and communication workloads across unevenly sized groups. This strategy preserves the structural simplicity and inherent efficiency of traditional 2D tensor parallelism, while generalizing its applicability to realistic, non-uniform deployments.

Section V-A details the capability abstraction methodology and the principles of asymmetric group formation. Section V-B elaborates on the execution pipeline, covering intergroup communication, local computation, and aggregation of results. Finally, Section V-C presents a formal cost model to quantify the communication overhead along with an optimization algorithm for automated group assignment.

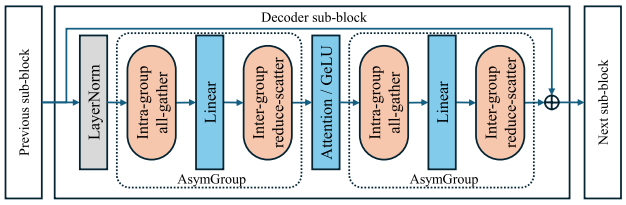

FIGURE 5. AsymGroup's three-stage pipeline–intergroup all-gather, linear computation, and intragroup reduce-scatter–applied to the two sub-blocks within each transformer decoder block.

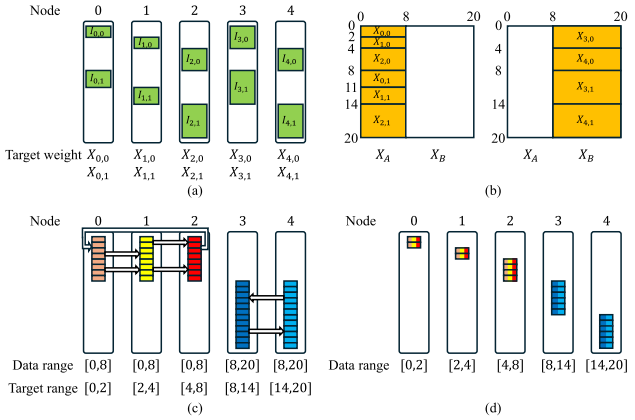

FIGURE 6. Execution pipeline of AsymGroup.(a) Capability-based proportional partitioning of input batches across nodes. (b) Two-level weight partitioning: intergroup (column-wise) and intragroup (row-wise). (c) Capability-aware intragroup reduce-scatter to redistribute partial results. (d) Final outputs proportionally match each node's capability, ensuring balanced workloads.

A. OVERVIEW OF AsymGroup

1) LOGICAL CAPABILITY MODELING

AsymGroup begins by abstracting the GPUs within each physical node into a single logical node and assigning it a relative capability score denoted by c_n. This capability measure encapsulates the total computational power of a node by accounting for both the number of GPUs and their individual performance characteristics. The total system capability, defined as $C_{\text{system}} = \sum_n c_n$, serves as the basis for proportionally allocating computational and communication workloads across the distributed system.

For instance, in a system with five nodes containing GPUs arranged as [1, 1, 2, 3, 3], as depicted in Fig. 4(a), each node receives a corresponding capability value. Then, global input batch B is partitioned proportionally among these nodes according to their capability scores, as shown in Fig. 4(b). Specifically, the batch size (B_n) assigned to an arbitrary node (n) is defined as $B_n = B \cdot c_n / C_{system}$.

This capability-based abstraction facilitates the early identification of potential performance bottlenecks and ensures balanced load distribution decisions across both computational and communication tasks.

2) ASYMMETRIC GROUPING STRATEGY

Conventional 2D tensor parallelism typically assumes uniform device groupings, resulting in fixed-size row and column partitions regardless of the actual hardware capability distributions. In contrast, AsymGroup adopts an approach to grouping based on aggregated node performance rather than the number of nodes.

Fig. 4(c) illustrates this principle, where five nodes with capabilities of [1, 1, 2, 3, 3] are divided into two groups: Group A, aggregating a total capability of 4 (1 + 1 + 2), and Group B with a total capability of 6 (3 + 3). This capability-driven grouping method ensures proportionally balanced computation and communication loads, thereby improving communication efficiency and resource utilization within heterogeneous GPU clusters.

3) HIERARCHICAL CAPABILITY MAPPING

The capability-abstraction model naturally extends from internode to intranode scenarios. Within a single node, each GPU can be similarly treated as an independent logical unit with its own distinct capability. Although the subsequent discussion predominantly addressed internode communication, the same capability-based partitioning and workload distribution logic also applies within the nodes, ensuring methodological consistency. For example, the capability vector [1, 1, 2, 3, 3] represents either multiple nodes with differing GPU counts or individual GPUs within a single node exhibiting heterogeneous performance characteristics. This hierarchical, unified abstraction allows AsymGroup to seamlessly manage both intra and internode heterogeneity.

B. EXECUTION FLOW UNDER AsymGroup

Building on the formulation presented in Section V-A, AsymGroup generalizes the 2D tensor parallel execution pipeline into three distinct stages: (i) intergroup all-gather, (ii) group-wise linear computation, and (iii) intragroup reduce-scatter.

As illustrated in Fig. 5, each transformer decoder block consists of two sub-blocks, one containing the self-attention layer and the other the feedforward layer with GeLU activation. Each sub-block includes two linear operations, and every linear layer is wrapped with one all-gather and one reduce-scatter. As a result, each decoder block performs four all-gather and four reduce-scatter operations per forward pass.

This section details the communication and computation mechanisms employed at each stage, highlighting how AsymGroup effectively mitigates bottlenecks through capability-aware load balancing.

1) INTERGROUP ALL-GATHER

In traditional 2D tensor parallelism, each computational group must gather complete input data prior to the local computation. Similarly, AsymGroup initiates an intergroup all-gather to exchange input data among groups. However, heterogeneous group sizes make conventional collective communication strategies inefficient owing to two primary

issues [26]: (i) load imbalance caused by uneven data volumes, and (ii) inefficient bandwidth utilization during broadcasts.

To mitigate these problems, AsymGroup replaces conventional collective operations with explicit capability-aware point-to-point communications. Each group calculates the send and receive data volumes proportionally based on capability ratios and transmits only the necessary data slices directly to each receiving group.

As illustrated in Fig. 4(d), if Group B sends 12 items to Group A, which consists of nodes with capabilities [1, 1, 2], then the data are proportionally partitioned into slices of [3, 3, 6]. Conversely, if Group A transmits eight items to Group B, which is composed of nodes with equal capabilities [3, 3], the data are evenly divided into [4, 4]. This capability-aware slicing ensures balanced workloads and bounds the communication latency using the slowest communication path, thereby significantly alleviating potential bottlenecks.

2) GROUP-WISE LINEAR COMPUTATION

Following input synchronization, each group independently performs local linear computations. To ensure computational balance, AsymGroup proportionally partitions the global weight matrix among the groups according to their aggregated capabilities.

Fig. 6(b) demonstrates this strategy: the global weight matrix is first partitioned column-wise between groups in proportion to each group's total capability. For example, given two groups—Group A and Group B, with capabilities $c_A = 4$ and $c_B = 6$, the weight matrix is split in a 4:6 ratio.

Within each group, the assigned weight block is further partitioned row-wise among the individual nodes based on their relative capabilities. Each node subsequently performs local matrix multiplication using its allocated input shard and corresponding weight slice. This ensures that every node handles computational workloads proportional to its performance capability, thereby maximizing utilization and efficiency.

3) INTRAGROUP REDUCE-SCATTER

After completing the local linear computations, the nodes within each group hold partial output sums, which must be aggregated to form the final results. This aggregation step was executed via an intragroup reduce-scatter operation, redistributing the partial results based on node capabilities.

Each node identifies and transmits data elements outside its final assigned range to other nodes within the same group, while simultaneously receiving data elements within their own range from peer nodes. The volume of data exchanged is determined explicitly by performance-based proportional allocation. For instance, as shown in Fig. 6(c), node 0 in Group A transmits six of its eight items, whereas node 2 transmits only four items, reflecting their respective capabilities. Group B, with uniformly capable nodes, evenly exchanged data items.

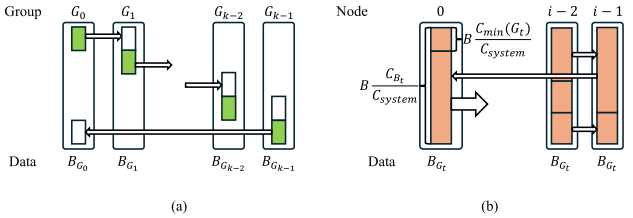

FIGURE 7. Communication bottleneck modeling in AsymGroup.(a) Ring-based intergroup All-Gather: Each group forwards received data to its neighbor in circular order until all data is shared. Latency is dominated by the group with the highest communication load. (b) Ring-based intragroup Reduce-Scatter: Each node sends data not assigned to itself. Lower-capability nodes may transmit more data and become bottlenecks.

Fig. 6(d) illustrates that, following this redistribution phase, each node holds a final output proportional to its original capability. This consistent capability-based partitioning strategy maintains balanced workloads throughout all the stages of computation and communication. Consequently, the critical communication path latency remains governed by the node with the heaviest communication load, effectively bounding the overall performance overheads.

C. GROUP OPTIMIZATION STRATEGY

Although AsymGroup provides flexible group formation through capability-aware partitioning, the choice of group assignments significantly affects the overall communication efficiency. This section introduces a detailed cost model for quantifying communication bottlenecks during both intergroup and intragroup phases, and describes an optimization algorithm designed to minimize the total communication cost through strategic node-group allocation.

1) PROBLEM FORMULATION

Consider a distributed system comprising N nodes, indexed by $n = 0, 1, \ldots, N-1$, where each node is assigned a computational capability c_n. These N nodes are partitioned into k disjoint groups, denoted as $G_0, G_1, \ldots, G_{k-1}$, where $1 \leq k \leq N$. Each node belongs to exactly one group, and no node is shared between groups. For an arbitrary group G_t, the total capability of each group C_{G_t} and the overall system capability C_{system} are defined as $C_{G_t} = \sum_{n \in G_t} c_n$ and $C_{\text{system}} = \sum_{t=0}^{k-1} C_{G_t}$, respectively.

The optimization goal is to determine the assignment of nodes to groups (G_t) that minimizes the total communication cost across both intergroup and intragroup operations.

2) INTERGROUP ALL-GATHER COST

As illustrated in Fig. 7(a), AsymGroup employs a ring-based all-gather protocol among groups, where a total of $k-1$ communication steps are performed to ensure complete data exchange. During each communication step s, the group G_t forwards the data from the originating group G_{t-s} to its clockwise neighbor G_{t+1}. The transmitted data volume at each step is $B_{G_{t-s}}$, and the communication cost is dominated

by the most heavily loaded node involved in the transfer, either the sender or receiver. Formally, this step-wise cost is expressed as:

$$Cost_s = \max_{t\in\{0,\ldots,k-1\}} \left(B_{G_{t-s}} \cdot \max \left\{ \frac{c_{\max}(G_t)}{C_{G_t}}, \frac{c_{\max}(G_{t+1})}{C_{G_{t+1}}} \right\} \right). \quad (2)$$

Summing this expression across all communication steps, the total intergroup all-gather communication cost becomes

$$Cost^{\text{all-gather}}_{\text{intergroup}} = \sum_{s=0}^{k-2} Cost_s. \quad (3)$$

Equation (3) indicates that communication costs increase owing to group capability imbalance and the presence of nodes with disproportionately high workloads.

3) INTRAGROUP REDUCE-SCATTER COST

After completing the linear computations, the nodes perform an intragroup reduce-scatter operation to aggregate the partial results (Fig. 7(b)). Here, the node with the smallest capability within each group must redistribute the largest proportion of data to achieve capability-aligned outputs, thereby becoming a critical performance bottleneck. Thus, the intragroup communication cost is defined as:

$$Cost^{\text{reduce-scatter}}_{\text{intragroup}} = B \cdot \max_t \left(\frac{C_{G_t} - c_{\min}(G_t)}{C_{\text{system}}} \right). \quad (4)$$

4) TOTAL COMMUNICATION COST

The overall communication overhead for asymmetric groupings is the sum of the intergroup and intragroup costs defined in (3) and (4), respectively, and can be expressed as:

$$Cost^{\text{asymmetric}}_{\text{total}} = Cost^{\text{all-gather}}_{\text{intergroup}} + Cost^{\text{reduce-scatter}}_{\text{intragroup}}. \quad (5)$$

The total cost is bounded by the nodes that experience the highest communication load at any stage.

5) REDUCTION TO SYMMETRIC CONFIGURATIONS

Our model generalizes to heterogeneous settings but reduces cleanly under symmetric configurations in which all nodes have identical capabilities. Let g denote the total number of GPUs in a system. Under symmetric conditions described (6),

$$C_{G_t} = \frac{C_{\text{system}}}{k}, \quad c_n = \frac{C_{\text{system}}}{g} \quad (6)$$

the total cost derived in (5) simplifies to:

$$Cost^{\text{symmetric}}_{\text{total}} = \frac{B}{g} \cdot \left((k-1) + \left(\frac{g}{k} - 1 \right) \right). \quad (7)$$

Equation (7) matches the conventional 2D tensor parallelism cost model shown in (1). Therefore, our asymmetric communication model naturally generalizes the symmetric 2D tensor parallelism as a special case.

6) GROUP OPTIMIZATION ALGORITHM

Given the formulated cost model, we propose an optimization algorithm to minimize the communication overhead through strategic node assignment to groups.

a: OBJECTIVE

We aim to partition the nodes into k groups ($\{G_0, \ldots, G_{k-1}\}$) to minimize the total asymmetric communication cost. The minimization objective defined in Equation (8) corresponds to the total cost described in (5), and can be formally expressed as:

$$\min_{\{G_t\}} \left(\sum_{s=0}^{k-2} Cost_s + B \cdot \max_t \left(\frac{C_{G_t} - c_{\min}(G_t)}{C_{\text{system}}} \right) \right). \quad (8)$$

The first term in (8) addresses intergroup communication imbalance, ensuring that no single link or node is excessively burdensome. The second term addresses the intragroup load imbalance and optimizes the distribution of computational workloads within each group. Considering (2) and (4), reducing one component can inadvertently increase the other; thus, balancing both is critical to achieve overall efficiency.

b: OPTIMIZATION PROCEDURE

We implement a greedy reassignment strategy that iteratively improves the initial node assignment:

1) **Initialization:** Nodes are initially sorted by their capabilities and evenly assigned to k groups to approximate balanced capabilities:
$$C_{G_t} \approx \frac{C_{\text{system}}}{k}, \quad c_{\min}(G_t) \approx c_{\max}(G_t).$$
2) **Iterative Reassignment:**
   - Identify the group G_{t^*} that contributes most significantly to the total cost in (8).
   - Select the node $n^* \in G_{t^*}$ whose reassignment would most reduce this dominant cost.
   - Move node n^* to the target group $G_{t'}$ such that the overall cost increases minimally, maintaining an approximate capability balance.
3) **Termination:** The algorithm concludes that the further reassignment of any single node no longer reduces the total cost.

c: SCALABILITY AND EXTENSIONS

The algorithm has an iteration complexity of $\mathcal{O}(N \cdot k)$ and typically converges rapidly. Furthermore, it can readily incorporate additional system-specific constraints, such as network topology, memory limits, or scheduling preferences, by extending the cost function in (8). Given its modular nature, our approach is suitable for both static offline planning and dynamic runtime reconfiguration.

VI. EVALUATION

This section presents a quantitative evaluation of the performance of the proposed AsymGroup framework for

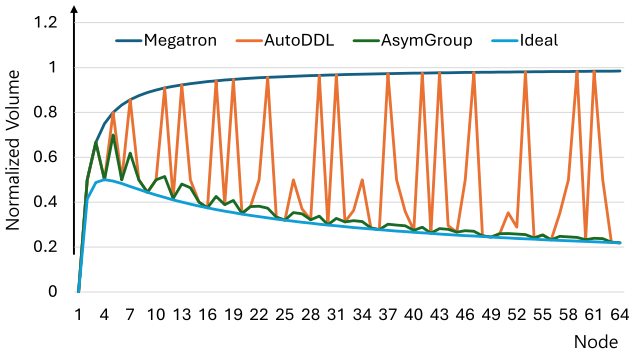

FIGURE 8. Comparison of communication times across varying node counts for Megatron, AutoDDL, and AsymGroup. As node count increases, traditional 2D tensor parallelism suffers from non-factorizable configurations. In contrast, AsymGroup flexibly adapts, achieving up to 63.5% lower communication overhead compared to Megatron, and 33.4% lower overhead compared to AutoDDL on average.

TABLE 1. System configurations S–16 used for evaluation.

System	2-GPU Nodes	1-GPU Nodes	Total GPUs	Description
S0	0	16	16	All nodes with 1 GPU
S2	2	14	18	2 nodes with 2 GPUs, 14 with 1
S4	4	12	20	4 nodes with 2 GPUs, 12 with 1
S6	6	10	22	6 nodes with 2 GPUs, 10 with 1
S8	8	8	24	8 nodes with 2 GPUs, 8 with 1
S10	10	6	26	10 nodes with 2 GPUs, 6 with 1
S12	12	4	28	12 nodes with 2 GPUs, 4 with 1
S14	14	2	30	14 nodes with 2 GPUs, 2 with 1
S16	16	0	32	All nodes with 2 GPUs

various heterogeneous GPU configurations. Specifically, we investigated its effectiveness in reducing the total communication volume, alleviating communication bottlenecks, and enhancing the model inference latency. Comparative analyses of the baseline methods demonstrated the efficiency, scalability, and general applicability of AsymGroup.

A. EVALUATION SETUP

We conducted experiments in a simulated cluster environment using closely modeling servers equipped with NVIDIA A100 GPUs. The number of nodes was varied from one to 64, with each node containing either one or two GPUs. GPUs within each node communicated via NVLink, providing 600 GB/s bidirectional bandwidth and 50 ns latency, whereas internode communication occurred over simulated network links offering 200 GB/s bidirectional bandwidth and 200 ns latency. This setup mirrors the hardware characteristics of DGX A100 systems realistically, enabling accurate assessments of communication performance under realistic conditions.

Evaluations were performed using large-scale language models with parameter counts of 175 billion (GPT-175B) and 530 billion (MT-530B). All the experiments utilized FP16 precision and a fixed batch size of 200. The input sequence length was fixed at 128 tokens, with output sequence lengths varying between 128, 512, and 2048 tokens depending on the experiment.

We note that LLaMA and similar mid-sized models typically fit within a single or dual-GPU setup (e.g., A100), and do not require tensor parallelism for inference, making them less relevant to the objectives of this study. Conversely, larger models beyond MT-530B demand significantly more hardware resources, which introduces challenges in maintaining experimental fairness. The two selected models—GPT-175B and MT-530B—thus provide a practical and representative range of model scales to observe performance trends across varying parallelization demands.

B. BASELINES AND METRICS

We compared the proposed AsymGroup framework with two state-of-the-art baseline methods.

- **Megatron-LM [7]:** Tensor parallelism with all-reduce communication, integrating FlexReduce [15] to optimize heterogeneous communication paths.
- **AutoDDL [13]:** Automatic 2D tensor parallelization framework that equally partitions input data among nodes, regardless of their GPU count.

Two primary evaluation metrics are considered.

1) **Total communication volume:** Assessed across varying node counts to quantify the reductions in communication overhead.
2) **Inference latency:** Measured under diverse GPU distributions per node, varying model sizes, and different output sequence lengths, to evaluate practical inference efficiency.

These metrics collectively demonstrate whether AsymGroup effectively mitigates communication bottlenecks and sustains high inference performance in asymmetric GPU environments.

C. NODE COUNT SENSITIVITY

To evaluate scalability, we varied the number of nodes from 1 to 64, assigning exactly one GPU per node to isolate communication performance. Communication volumes were measured for FlexReduce-enhanced Megatron, AutoDDL, and AsymGroup. In addition, an ideal 2D tensor parallel configuration arranged in $\sqrt{n} \times \sqrt{n}$ grid was included as the theoretical performance bound.

As shown in Fig. 8, AutoDDL achieved an average communication volume reduction of 45.1% relative to Megatron. AsymGroup further improved performance, reducing communication volume by an additional 33.4% over AutoDDL, corresponding to an overall reduction of 63.5% compared to Megatron. Notably, AsymGroup maintained robust performance even for node counts lacking straightforward grid factorization. For instance, at 38 nodes, AutoDDL is constrained to a rigid 2×19 structure. In contrast, AsymGroup formed six groups that approximated a $\sqrt{38}$ configuration, effectively balancing loads and eliminating bottlenecks.

This flexibility becomes especially beneficial when the node count cannot be evenly factored into a 2D grid. AutoDDL's performance fluctuates under such conditions due to rigid group formation, while AsymGroup adapts to available capability distributions, resulting in more consistent gains. These results underscore the importance

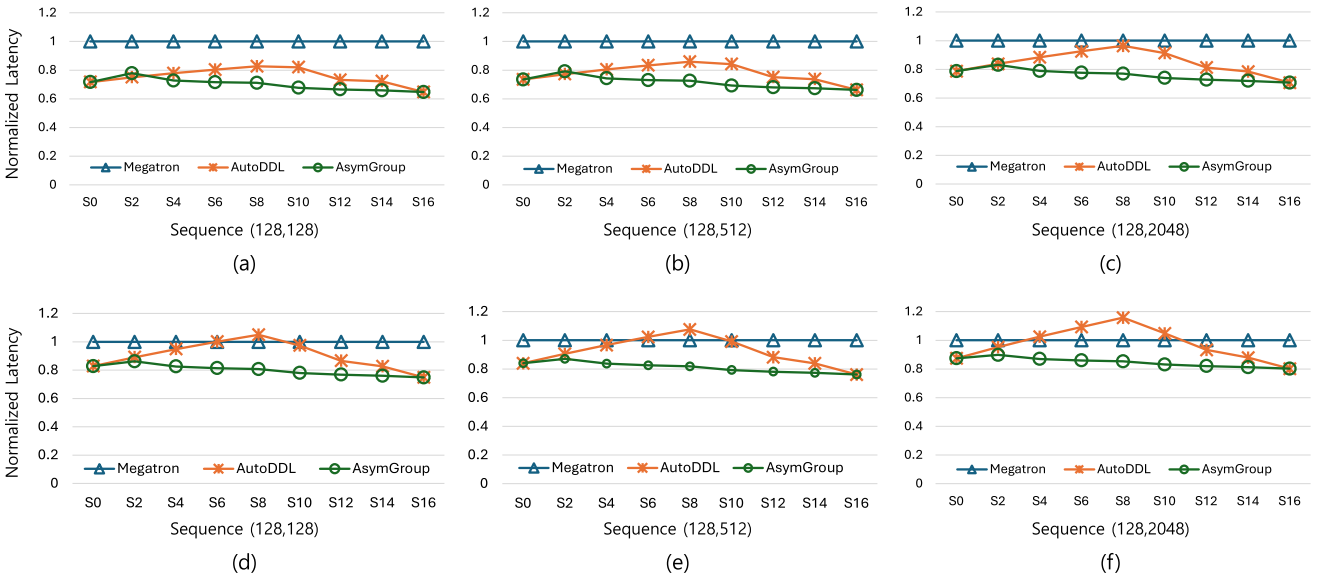

FIGURE 9. Inference latency across heterogeneous GPU configurations (S0–16) under varying sequence lengths (128, 512, 2048) and model scales (GPT-175B and MT-530B). AutoDDL experiences significant latency degradation under asymmetric GPU distributions due to load imbalance and underutilization. Conversely, AsymGroup consistently achieves balanced workloads and provides up to 26.3% lower inference latency in highly asymmetric settings.

of structure-agnostic group formation in large-scale deployments, highlighting AsymGroup's robustness under topologically constrained settings.

D. PERFORMANCE SENSITIVITY TO VARYING GPU COUNTS PER NODE

Next, we examined the performance under varying GPU counts per node, fixing the total number of nodes at 16 and varying GPU distributions from 16 to 32 GPUs, as detailed in Table 1. Intermediate configurations (e.g., S6 with six nodes containing two GPUs and 10 nodes with one GPU) represent practical scenarios involving hardware asymmetry without rigid tensor partitions.

Inference latency was evaluated for sequence lengths of 128, 512, and 2048 tokens for the GPT-175B and MT-530B models, respectively, and the results were normalized against Megatron+FlexReduce.

As illustrated in Fig. 9(a)-(c), AutoDDL suffers from significant GPU underutilization in asymmetric configurations (S0–S8). For instance, with 128-token sequences, AutoDDL reduced latency by 28.3% compared to Megatron in the symmetric S0 configuration, but only 13.3% in S8. This decline stems from AutoDDL's rigid group formation, which fails to allocate work efficiently across heterogeneous nodes. Its performance improved substantially beyond S10, peaking at 35.3% improvement in fully symmetric S16, where static 2D partitions align well with the hardware layout.

By contrast, AsymGroup flexibly utilizes all GPUs by adaptively shaping tensor partitions based on actual hardware capabilities. While it incurred a slightly higher latency (3.8%) than AutoDDL in the mildly asymmetric S2 scenario, its design becomes increasingly advantageous as asymmetry grows. For example, in S10, AsymGroup achieved up to 17.5% better performance, effectively distributing computation and minimizing idle GPU cycles. In the symmetric S16 case, both approaches converged to similar performance, suggesting that AsymGroup does not compromise efficiency even in ideal hardware conditions.

As the sequence length increases, attention computation dominates, reducing the relative impact of the communication overhead. Thus, the latency gap between AutoDDL and AsymGroup narrowed for longer sequences (2048 tokens). Still, AsymGroup maintained a 20% advantage in the highly asymmetric S8 case, highlighting the importance of capability-aware workload partitioning especially when computation-to-communication ratio shifts with larger inputs.

As shown in Fig. 9(d)-(f), similar trends appear for MT-530B, whose heavier matrix multiplications further penalize inefficient computation allocation. AutoDDL experienced up to 15.7% worse latency than Megatron at 2048 tokens under S8, as its imbalance exacerbated stalls and idle time. Conversely, AsymGroup consistently improved the performance by 10.2% (S2) and reached up to 25.2% improvement in fully symmetric S16 with shorter sequences.

These results collectively demonstrate that AsymGroup maintained consistent advantages across a diverse range of practical hardware and workload configurations. Based on the observed performance trends, AsymGroup is expected to remain effective even under broader variations in model architectures and system heterogeneity, particularly in environments which communication latency is the dominant performance bottleneck.

VII. CONCLUSION

This study introduced AsymGroup, a novel tensor parallelism framework specifically designed to address inherent hardware heterogeneity in modern HPC and cloud computing environments. AsymGroup generalizes conventional 2D tensor parallelism by dynamically forming groups based on the GPU count and computing performance, and proportionally distributing both the computation and communication workloads. To optimize group configurations systematically, we formulate an analytical communication cost model that accurately identifies and quantifies bottlenecks across nodes and individual GPUs.

Experimental evaluations of clusters with up to 64 nodes demonstrate that AsymGroup significantly reduces communication overhead compared to existing state-of-the-art frameworks. Specifically, it achieved an average communication time reduction of up to 63.5% relative to Megatron and 33.4% relative to AutoDDL, which is a representative 2D tensor parallel baseline. Furthermore, in realistically heterogeneous scenarios the number of GPUs per node varies—AsymGroup reduces the inference latency by up to 35.3% and 26.3% over Megatron and AutoDDL, respectively. Crucially, AsymGroup maintains consistent performance advantages even in situations where traditional mesh-based partitioning strategies fail owing to structural constraints, thereby highlighting its broad applicability and robustness.

We believe that AsymGroup represents an important advancement toward more efficient, scalable, and flexible LLM inference, enabling better resource utilization and performance in increasingly diverse computing infrastructures.

While AsymGroup demonstrates strong inference performance, extending it to training workloads remains an important direction. Training introduces additional challenges such as gradient synchronization and optimizer state updates, which may present new bottlenecks. However, because both inference and training are fundamentally dominated by linear operations and follow similar computational pipelines, the three-stage design of AsymGroup can be adapted naturally for training with further refinement.

Another limitation arises under scenarios with extreme capability imbalance, where proportional slicing may incur excessive communication overhead without corresponding computational benefit. Exploring heuristic or adaptive partitioning strategies may improve the performance in such cases by selectively tolerating the computation imbalance. Lastly, evaluating AsymGroup on a broader range of LLMs, including encoder-decoder models and sparse or quantized variants—would further validate its generality and reveal additional challenges in production-scale deployments.

REFERENCES

[1] OpenAI et al., "GPT-4 technical report," 2023, *arXiv:2303.08774*.

[2] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, "LLaMA: Open and efficient foundation language models," 2023, *arXiv:2302.13971*.

[3] S. Smith, M. Patwary, B. Norick, P. LeGresley, S. Rajbhandari, J. Casper, Z. Liu, S. Prabhumoye, G. Zerveas, V. Korthikanti, E. Zhang, R. Child, R. Yazdani Aminabadi, J. Bernauer, X. Song, M. Shoeybi, Y. He, M. Houston, S. Tiwary, and B. Catanzaro, "Using DeepSpeed and megatron to train megatron-turing NLG 530B, a large-scale generative language model," 2022, *arXiv:2201.11990*.

[4] X. Ren, P. Zhou, X. Meng, X. Huang, Y. Wang, W. Wang, P. Li, X. Zhang, A. Podolskiy, G. Arshinov, A. Bout, I. Piontkovskaya, J. Wei, X. Jiang, T. Su, Q. Liu, and J. Yao, "PanGu-$\sum$: Towards trillion parameter language model with sparse heterogeneous computing," 2023, *arXiv:2303.10845*.

[5] J. Lin, A. Yang, J. Bai, C. Zhou, L. Jiang, X. Jia, A. Wang, J. Zhang, Y. Li, W. Lin, J. Zhou, and H. Yang, "M6–10T: A sharing-delinking paradigm for efficient multi-trillion parameter pretraining," 2021, *arXiv:2110.03888*.

[6] Z. Li, S. Zhuang, S. Guo, D. Zhuo, H. Zhang, D. Song, and I. Stoica, "Terapipe: Token-level pipeline parallelism for training large-scale language models," in *Proc. Int. Conf. Mach. Learn.*, 2021, pp. 6543–6552.

[7] D. Narayanan, M. Shoeybi, J. Casper, P. LeGresley, M. Patwary, V. Korthikanti, D. Vainbrand, P. Kashinkunti, J. Bernauer, B. Catanzaro, A. Phanishayee, and M. Zaharia, "Efficient large-scale language model training on GPU clusters using megatron-LM," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, New York, NY, USA, Nov. 2021, pp. 1–14.

[8] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-LM: Training multi-billion parameter language models using model parallelism," 2019, *arXiv:1909.08053*.

[9] A. Bambhaniya, R. Raj, G. Jeong, S. Kundu, S. Srinivasan, S. Subramanian, M. Elavazhagan, M. Kumar, and T. Krishna, "Demystifying AI platform design for distributed inference of next-generation LLM models," 2024, *arXiv:2406.01698*.

[10] Z. Cai, Z. Liu, S. Maleki, M. Musuvathi, T. Mytkowicz, J. Nelson, and O. Saarikivi, "Synthesizing optimal collective algorithms," in *Proc. 26th ACM SIGPLAN Symp. Princ. Pract. Parallel Program.*, New York, NY, USA, Feb. 2021, pp. 62–75.

[11] Y. Zhuang, H. Zhao, L. Zheng, Z. Li, E. P. Xing, Q. Ho, J. E. Gonzalez, I. Stoica, and H. Zhang, "On optimizing the communication of model parallelism," in *Proc. Mach. Learn. Syst.*, Jan. 2022, pp. 526–540.

[12] Q. Xu and Y. You, "An efficient 2D method for training super-large deep learning models," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, May 2023, pp. 222–232.

[13] J. Chen, S. Li, R. Guo, Y. Jinhui, and T. Hoefler, "AutoDDL: Automatic distributed deep learning with near-optimal bandwidth cost," *IEEE Trans. Parallel Distrib. Syst.*, vol. 35, no. 8, pp. 1331–1344, May 2024.

[14] S. Li, H. Liu, Z. Bian, J. Fang, H. Huang, Y. Liu, B. Wang, and Y. You, "Colossal-AI: A unified deep learning system for large-scale parallel training," 2021, *arXiv:2110.14883*.

[15] J. Lee, I. Hwang, S. Shah, and M. Cho, "FlexReduce: Flexible all-reduce for distributed deep learning on asymmetric network topology," in *Proc. 57th ACM/IEEE Design Autom. Conf. (DAC)*, Jul. 2020, pp. 1–6.

[16] W. Won, M. Elavazhagan, S. Srinivasan, S. Gupta, and T. Krishna, "TACOS: Topology-aware collective algorithm synthesizer for distributed machine learning," in *Proc. 57th IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, CA. Los Alamitos, CA, USA: IEEE Computer Society, Nov. 2024, pp. 856–870.

[17] Q. Zhou, Q. Anthony, L. Xu, A. Shafi, M. Abduljabbar, H. Subramoni, and D. K. D. Panda, "Accelerating distributed deep learning training with compression assisted allgather and reduce-scatter communication," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, May 2023, pp. 134–144.

[18] A. Shah, V. Chidambaram, M. Cowan, S. Maleki, M. Musuvathi, T. Mytkowicz, J. Nelson, O. Saarikivi, and R. Singh, "TACCL: Guiding collective algorithm synthesis using communication sketches," in *Proc. 20th USENIX Symp. Networked Syst. Design Implement. (NSDI 23)*, Jan. 2021, pp. 593–612.

[19] E. Chan, M. Heimlich, A. Purkayastha, and R. van de Geijn, "Collective communication: Theory, practice, and experience," *Concurrency Comput., Pract. Exper.*, vol. 19, no. 13, pp. 1749–1783, Sep. 2007.

[20] R. Thakur and W. D. Gropp, "Improving the performance of collective operations in mpich," in *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, J. Dongarra, D. Laforenza, and S. Orlando, Eds., Berlin, Germany: Springer, 2003, pp. 257–267.

[21] G. Wang, S. Venkataraman, A. Phanishayee, J. Thelin, N. Devanur, and I. Stoica, "Blink: Fast and generic collectives for distributed ML," 2019, *arXiv:1910.04940*.

[22] C.-H. Chu, P. Kousha, A. A. Awan, K. S. Khorassani, H. Subramoni, and D. K. D. K. Panda, "NV-group: Link-efficient reduction for distributed deep learning on modern dense GPU systems," in *Proc. 34th ACM Int. Conf. Supercomputing*, New York, NY, USA, Jun. 2020, pp. 1–12.
[23] C. Chen, X. Li, Q. Zhu, J. Duan, P. Sun, X. Zhang, and C. Yang, "Centauri: Enabling efficient scheduling for communication-computation overlap in large model training via communication partitioning," in *Proc. 29th ACM Int. Conf. Architectural Support Program. Lang. Operating Syst.*, New York, NY, USA, Apr. 2024, pp. 178–191.
[24] G. Ravindran and M. Stumm, "A performance comparison of hierarchical ring- and mesh-connected multiprocessor networks," in *Proc. 3rd Int. Symp. High-Perform. Comput. Archit.*, 1997, pp. 58–69.
[25] X. Luo, W. Wu, G. Bosilca, Y. Pei, Q. Cao, T. Patinyasakdikul, D. Zhong, and J. Dongarra, "HAN: A hierarchical AutotuNed collective communication framework," in *Proc. IEEE Int. Conf. Cluster Comput. (CLUSTER)*, Sep. 2020, pp. 23–34.
[26] S. Zhang, L. Diao, C. Wu, Z. Cao, S. Wang, and W. Lin, "HAP: SPMD DNN training on heterogeneous GPU clusters with automated program synthesis," in *Proc. 19th Eur. Conf. Comput. Syst.*, New York, NY, USA, Apr. 2024, pp. 524–541.

KI TAE KIM received the B.S. degree in electrical and electronics engineering from Yonsei University, Seoul, South Korea, in 2017, where he is currently pursuing the Ph.D. degree in electrical and electronics engineering.

His research interests include flash memory applications, processing-in-memory, and high performance system architectures.

SEOK-JU IM received the B.S. degree in electrical and electronics engineering from Yonsei University, Seoul, South Korea, in 2023, where he is currently pursuing the integrated M.S. and Ph.D. degree in electrical and electronics engineering.

His research interests include memory systems, processing-in-memory, and AI accelerator.

EUI-YOUNG CHUNG (Member, IEEE) received the B.S. and M.S. degrees in electronics and computer engineering from Korea University, Seoul, South Korea, in 1988 and 1990, respectively, and the Ph.D. degree in electrical engineering from Stanford University, CA, USA, in 2002.

From 1990 to 2005, he was a Principal Engineer with the SoC Research and Development Center, Samsung Electronics, Yongin, South Korea. He is currently a Professor with the School of Electrical and Electronics Engineering, Yonsei University, Seoul. His research interests include system architecture, bio-computing, and VLSI design, including all aspects of computer-aided design with the special emphasis on low-power applications, and flash memory applications.