TASP: TOPOLOGY-AWARE SEQUENCE PARALLELISM

Yida Wang[1,3] **Ke Hong**[2,3] **Xiuhong Li**[3]

Yuanchao Xu[1] ✉ **Wenxun Wang**[2] **Guohao Dai**[4,3] **Yu Wang**[2] ✉

✉ xuyuanchao@cnu.edu.cn, yu-wang@tsinghua.edu.cn

Abstract

Long-context large language models (LLMs) face constraints due to the quadratic complexity of the self-attention mechanism. The mainstream sequence parallelism (SP) method, Ring Attention, attempts to solve this by distributing the query into multiple query chunks across accelerators and enable each Q tensor to access all KV tensors from other accelerators via the Ring AllGather communication primitive. However, it exhibits low communication efficiency, restricting its practical applicability. This inefficiency stems from the mismatch between the Ring AllGather communication primitive it adopts and the AlltoAll topology of modern accelerators. A Ring AllGather primitive is composed of iterations of ring-styled data transfer, which can only utilize a very limited fraction of an AlltoAll topology.

Inspired by the Hamiltonian decomposition of complete directed graphs, we identify that modern accelerator topology can be decomposed into multiple orthogonal ring datapaths which can concurrently transfer data without interference. Based on this, we further observe that the Ring AllGather primitive can also be decomposed into the same number of concurrent ring-styled data transfer at every iteration. Based on these insights illustrated in Figure 1, we propose **TASP**, a topology-aware SP method for long-context LLMs that fully utilizes the communication capacity of modern accelerators via *topology decomposition* and *primitive decomposition*. Experimental results on both single-node and multi-node NVIDIA H100 systems and a single-node AMD MI300X system demonstrate that TASP achieves higher communication efficiency than Ring Attention on these modern accelerator topologies and achieves up to 3.58× speedup than Ring Attention and its variant Zigzag-Ring Attention. The code is available at `https://github.com/infinigence/HamiltonAttention`.

[1] Capital Normal University
[2] Tsinghua University
[3] Infinigence-AI
[4] Shanghai Jiao Tong University
✉ Corresponding authors: Yuanchao Xu, Yu Wang

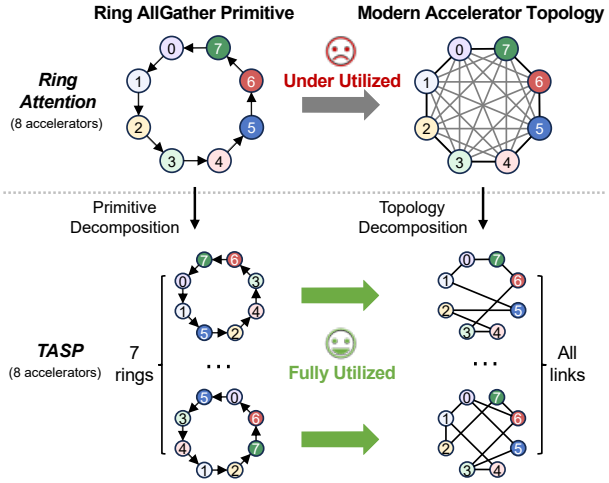


Figure 1: The mismatch between the ring-style data transfer of Ring AllGather and the fully-connected communication links of modern accelerators.

1 Introduction

The rapid advancement of large language models (LLMs) has ushered in a new era of long-context capabilities, with modern systems now supporting context windows extending to several million tokens [1, 2]. This breakthrough is critical for complex tasks that require reasoning over massive datasets, such as processing lengthy legal documents, performing whole-codebase analysis, or engaging in extended multi-modal dialogues. However, the core architecture of the popular Transformer model presents a fundamental scalability bottleneck: the quadratic computational complexity of the self-attention mechanism [3]. This poses a significant challenge for inference on these long queries, as the resource demands for computation and key-value (KV) tensor storage quickly become prohibitively expensive on a single accelerator.

To address these constraints, Sequence Parallelism (SP) methods [4, 5, 6, 7, 8] have emerged as a key technique for long-context training and inference. Through distributing the prolonged sequence dimension of long-context queries across a cluster of accelerators, they effectively shard the computational load and reduce the memory footprint on each accelerator, making long-context processing more feasible. However, these approaches suffer from notable limitations: Ulysses[5] ties the degree of parallelism to the number of Attention KV heads, which can not scale well. The other mainstream works, Ring Attention and its variants [4, 6, 7, 8], have no scalability constraint but have communication efficiency issues. These works first chunk the query into query chunks and distribute the these chunks to multiple accelerators, and then each accelerator computes the corresponding Q chunk and KV Chunk. Since each Q chunk requires all of the KV Chunks, an AllGather of these KV chunks must be performed.

Unfortunately, the standard AllGather primitive has a constraint that each accelerator needs to store all the KV Chunks aggregated from other accelerators, failing to reduce the memory consumption. Thus, Ring Attention decomposes the AllGather on n accelerators into $n-1$ iterations of the ring-style data transfer (as shown in Figure 4). In fact, this ring-style implementation of the AllGather primitive (denoted in this paper as Ring AllGather) is one of the oldest MPICH implementations of the AllGather primitive [9]. The benefits of Ring AllGather are two-fold: (1) it incurs no data duplication at each iteration of communication and computation; and (2) the communication can be fully overlapped by computation in each iteration if the compute-to-communication ratio (CCR) is above 1.0. Unfortunately, Ring AllGather suffers from the problem of low communication efficiency since each iteration of its ring-style data transfer can only utilize a small portion of communication links within the AlltoAll accelerator topology. As shown in Figure 1, the utilization of full communication capacity is $1/(n-1)$, which is approximately 14.3% when $n = 8$. This inefficiency in communication results in a low CCR of Ring Attention on modern accelerator systems, as shown in Table 1. Consequently, the communication overhead of Ring Attention fails to be overlapped with computation, thus becoming the primary bottleneck that limits its overall performance.

To this end, we propose **TASP**, a novel sequence parallelism methodology designed to overcome the communication bottleneck of Ring Attention by fully leveraging all the communication capacity in modern accelerator clusters. Our key insight is that the mismatch between the ring-style data transfer of Ring AllGather and the fully-connected communication links in modern accelerator clusters can be solved by **topology decomposition** and **primitive decomposition**, as shown in Figure 1. In terms of the topology decomposition, inspired by the Hamiltonian decomposition of complete directed graphs, we identify that the modern accelerator topology can be decomposed into multiple orthogonal ring datapaths that can concurrently transfer data in a ring-style without mutual interference. We first analyze the decomposition of the AlltoAll topology of a single-node into $n-1$ edge-disjoint Hamiltonian cycles, where vertices represent accelerators and edges represent peer-to-peer communication links. The detailed decomposition scheme is illustrated in Figure 2, and the formulation of this scheme is presented in Section 3.1. Apart from the single-node scenario, Hamiltonian decomposition schemes also exist for multi-node topologies based on IB/RoCE network switches which are commonly adopted in modern AI data-centers.

Table 1: Evaluation of CCR and speedup of TASP over Ring Attention, measured with a batch size of 48 on 8 AMD-MI300X across different sequence lengths.

SeqLen	10K	20K	40K	50K	100K
CCR	0.39	0.65	0.80	0.98	1.17
Speedup	**2.4**	**1.8**	**1.5**	**1.3**	**1.1**

Building upon the topology decomposition, we need to accordingly decompose the ring-style data transfer in each iteration of Ring AllGather into multiple orthogonal ring-style data transfer (We denote them all-together as Multi-Ring AllGather in this paper). In this way, we can achieve perfect mapping between these orthogonal ring-style data transfer and the orthogonal ring datapaths obtained from topology decomposition, as shown in Figure 1. We prove that Multi-Ring AllGather is equivalent to Ring AllGather in terms of distributed Attention computation through a comprehensive analysis of Ring AllGather's characteristics in accessibility and zero-copy. Moreover, considering the widely-adopted causal mask in LLMs, we present a query chunk placement strategy to further improve the workload balance between accelerators.

Through extensive experiments, we demonstrate that TASP provides a more communication-efficient solution for sequence parallelism, achieving significant speedup over the baseline method when its CCR is below 1.0. For single-node scenarios with 8 accelerators, TASP achieves up to $2.31\times$ speedup on the H100 server and $3.58\times$ speedup on the MI300X server over Ring Attention (non-causal baseline) and Zig-zag Ring Attention (causal baseline). For multi-node scenarios, we observe an average speedup of $1.43\times$ on two 8-H100 nodes, and an average speedup of $1.27\times$ on four 8-H100 nodes, respectively.

Overall, this paper makes the following contributions:

- We first introduce a novel methodology to improve communication efficiency via topology decomposition and primitive decomposition.
- We successfully apply the methodology in sequence parallelism optimization and propose TASP, inspired by Hamiltonian decomposition of complete graphs. It enables a perfect mapping between all the data transfer and all the communication links.

- The evaluation results demonstrate that TASP has significant performance improvement over the state-of-the-art baselines, Ring Attention and Zigzag Ring Attention.

2 Background and Motivation

2.1 Sequence Parallelism

The trend toward handling long-context LLM application introduces significant challenges, including computational overhead and memory demands for KV tensor storage across extended sequences. To address these issues, SP has emerged as a promising technique to distribute the workload efficiently across multiple accelerators, which can be broadly categorized into two types, including Ulysses [5] and Ring Attention [10].

Ulysses initially shards along the sequence length dimension and distributes across accelerators, the KV tensors are partitioned by the KV head dimension through an AlltoAll operation. This allows each accelerator to maintain a complete view of the whole KV tensors along the sequence length dimension for its assigned heads. Since each KV head computes Attention independently, each accelerator can calculate the full Attention output for its respective heads. Subsequently, another AlltoAll operation can be utilized to transpose the partial Attention outputs across accelerators to form the final result. However, Ulysses's leverage of the independence between KV heads introduces a constraint: its performance is highly dependent on the number of KV heads and the computational balance across them. Current trends in Attention mechanisms exhibit fewer KV heads, such as Grouped Query Attention (GQA) with typically 4 or 8 KV heads [11], or Multi-Query Attention (MQA) [12] and Multi-Layer Attention (MLA) [13] with only a single KV head. This reduction limits Ulysses's scalability in distributed environments, as it heavily relies on a sufficient number of KV heads for effective partitioning across accelerators. Moreover, it is hard for Ulysses to overlap its communication with computation since they are dependent to each other.

Ring Attention partitions queries (Q and KV) along the sequence length dimension across accelerators. The Q tensors remain stationary on their respective accelerators, while the KV blocks are rotated among all accelerators through a Ring AllGather communication primitive. Over iterations, each accelerator incrementally computes, via Flash Attention [14], the full Attention output for its local query segment by processing all global KV blocks. A fundamental challenge inherent to the original Ring Attention is computational imbalance. The computational cost of Attention is often unevenly distributed across the sequence length dimension, particularly under the widely used causal mask. Several optimizations have been proposed to address this issue. For instance, Striped Attention [6] employs a striped partitioning strategy to achieve better load balance. Megatron CP [8] uses an axial-symmetric partitioning scheme that can achieve perfect load balance for causal masking. Ring Attention and its causal-masking balanced variants typically employ computation-communication overlap to mitigate the critical communication overhead. However, perfect overlapping of communication overhead is not easy to achieve. As evidenced by Table 1, the CCR which indicates the ratio between computation and communication falls below 1.0 under many cases. Consequently, the overall Attention performance remains constrained by the communication overhead of the Ring AllGather primitive. Therefore, improving the efficiency of communication is crucial for enhancing the performance of Ring Attention generally.

2.2 Inter-connectivity of Hardware Platforms

With the increasing scale of LLMs, a single accelerator can no longer meet the requirements for efficient serving. As a result, deploying LLM inference services increasingly relies on clusters of accelerators utilizing various parallelization strategies such as Tensor Parallelism (TP) and Expert Parallelism (EP) [15]. These parallelization strategies, however, impose stringent demands on the interconnectivity between devices: any pair of devices must support high-bandwidth, low-latency P2P communication, and multiple such communications should be able to occur concurrently without interference, so that collective communication operations like AlltoAll and ALLGather that are utilized in TP and EP can be efficiently carried out.

To support such capabilities, fully connected topologies have become the preferred choice for networking state-of-the-art AI accelerators, which fall into two categories: full-mesh and switch-based. For example, Huawei's Ascend 910B NPUs [16], AMD's MI300X series GPUs [17], and Intel's Gaudi 3 AI accelerators [18] all employ a full-mesh topology among 8 devices. For full-mesh topologies, vanilla Ring Attention can only utilize $\frac{1}{n-1}$ of the $n(n-1)$ unidirectional communication links among accelerators. In contrast, NVIDIA's GPUs utilize a switch-based fully connected topology [19], which allows for more flexible bandwidth allocation compared to the full-mesh approach. For larger-scale clusters, multi-level switch-based architectures—such as the CloudMatrix384 [20]—provide fully P2P inter-connectivity among up to 384 Ascend 910 NPUs and 192 CPUs.

These advancements of accelerator inter-connectivity underscore that high-performance P2P communication capability is both a fundamental requirement and a key trend for next-generation AI hardware architecture. However, some communication primitives can not fully utilze the communication links in an AlltoAll topology, resulting in poor communication efficiency and consequently increases end-to-end latency. Therefore, when designing and implementing parallelization strategies, it is essential to fully account for AlltoAll inter-connectivity of hardware platforms to maximize communication efficiency.

Table 2: Notation description.

Notation	Description
n	Number of accelerators
V	Accelerator set
E	Communication link set
R_i	The i-th accelerator (rank i)
G	Directed graph expression of accelerator topology
K_n	Complete graph with degree n

3 Design of TASP

The design of TASP involves two aspects: the accelerator `topology decomposition` to unveil the potential of full communication link utilization, and the Ring AllGather `primitive decomposition` to map the chunked ring-styled data transfer workload onto all the communication links.

First, we demonstrate that modern accelerator topologies, including the single-node and multi-node ones, can be decomposed into a set of mutually orthogonal ring datapaths by Hamiltonian Decomposition. Those ring datapaths can be utilized simultaneously without interference.

Second, we demonstrate that the `Ring AllGather` primitive utilized in Ring Attention can be decomposed into `Multi-Ring AllGather`, within which a number of ring-styled data transfer are concurrently carried out at every iteration. Consequently, we can map each ring-styled data transfer to each ring datapath, fully utilizing all communication links.

3.1 Decomposition of Accelerator Topology

3.1.1 Ring Datapath Definition

We define a ring datapath based on the formulation of the accelerator topology as a directed graph. Consider a set of accelerators denoted as $V = \{R_0, R_1, \ldots, R_{n-1}\}$, where R_i denotes the i-th rank in the communication group. Each accelerator R_i has several communication ports that can be connected to other accelerators via electrical or fiber-optic cables and optionally a set of network switches. Due to duplex communication mode, a cable c_k connecting R_i and R_j establishes two logical unidirectional communication links, denoted as (R_i, R_j, c_k) and (R_j, R_i, c_k). For accelerators interconnected via switches, there exist two communication links between each pair of accelerators but these links share the same amount of aggregated bandwidth.

By treating the accelerators as vertices, the set of all unidirectional communication links forms a directed edge set by

$$E = \{(R_i, R_j, c_k) \mid \text{there exists a communication link from } R_i \text{ to } R_j \text{ via cable } c_k\}. \quad (1)$$

Thus, the topology can be represented as a directed graph $G = (V, E)$.

Algorithm 1 Decomposition of K_n into a set of Hamiltonian Cycles

Require: Integer n with $n \geq 8$ and $n\%4 = 0$
Ensure: Hamiltonian cycle list h_cycles
// Generate Hamiltonian paths
$init\_paths \leftarrow \text{GETPATH}(n-2)$
// Link the paths to form cycles
$init\_cycles \leftarrow \text{GENCYCLES}(init\_paths)$
// Find a set of edges in each cycle that forms a Hamiltonian Path
$k \leftarrow n/4 - 1$
$shifts \leftarrow \{0:1,\ k+1:4k+2,\ 2k+2:3,\ 3k+2:4k\}$
$shift \leftarrow shifts.get(i, 2k)$
$rotated[i][j] \leftarrow init\_cycles[i][(j+shift)\%(n-1)]$
// Disconnect these edges to form paths
$appender[rotated[i][-1]] \leftarrow rotated[i][0]$
$appender\_list \leftarrow [n-2]$, then append $appender[last]$ for $n-2$ steps
// Link the paths to form cycles again
$h\_cycles \leftarrow \text{GENCYCLES}(appender)$
return h_cycles

Next, we define a ring datapath. A ring datapath on G is a subgraph $D \subseteq G$ that forms a directed Hamiltonian cycle, `i.e.`, a directed cycle that visits every vertex in V exactly once. Two ring datapaths $D_i = (V, E_i), D_j = (V, E_j), i \neq j$ are said to be **orthogonal ring datapaths** if they are edge-disjoint, that is,

$$E_i \cap E_j = \emptyset. \quad (2)$$

The property ensures that a set of mutually orthogonal ring datapaths can support simultaneous data transfer without mutual interference.

3.1.2 Single-Node topology Decomposition

The most common topology in modern single-node accelerators is the AlltoAll topology, encompassing switch-based (NVIDIA GPUs) and full-mesh (AMD GPUs) ones. Theoretically, both topologies can be represented as a complete directed graph K_n of degree n. For decomposition of these AlltoAll topologies, we can directly apply the existing *Hamiltonian decomposition of complete directed graphs into edge-disjoint Hamiltonian Cycles* [21, 22, 23, 24, 25] (denoted in this paper as K_n-decomposition).

As illustrated in Figure 2, we use 8 accelerators within a single node (`i.e.`, K_8) as an example, the decomposition scheme is computed via an algorithm demonstrated in Algorithm 1 proposed by [26], which decomposes a complete graph K_n within $O(n)$ time complexity. Based on this decomposition, the $n \times (n-1)$ communication links are divided into a set of $n-1$ edge-disjoint Hamiltonian cycles. These cycles are then utilized as the mutually orthogonal ring datapaths for concurrent data transfer.

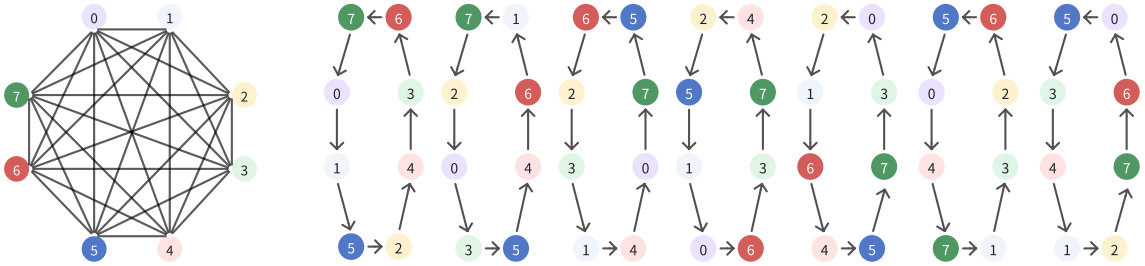
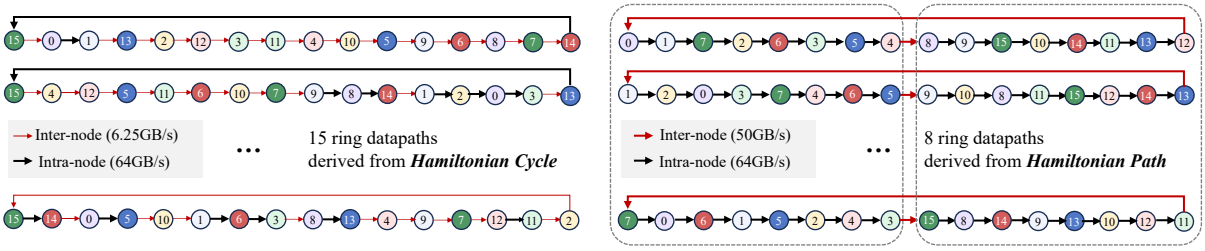


Figure 2: Decomposition of 8-accelerator AlltoAll topology graph K_8 into 7 edge-disjoint directed Hamiltonian cycles. The decomposed Hamiltonian cycles correspond to mutually orthogonal ring datapaths that traverse all 8 accelerators.



(a) Hamiltonian Cycle decomposition for two nodes

(b) Hamiltonian Path decomposition for two nodes

Figure 3: Left: The 15 Hamiltonian cycles decomposed from the topology of H100-2 via a $K_{16} - decomposition$ scheme. Right: The 8 Hamilton cycles derived from a $(8 - K_8 - 8)^2$-decomposition scheme.

3.1.3 Multi-Node topology Decomposition

In multi-node systems such as the NVIDIA DGX H100 GPU cluster [27], multiple H100 servers each equipped with 8 H100 accelerators are inter-connected via a high-bandwidth Infiniband Network. Within each node, 8 accelerators are fully-connected via four ultra-high bandwidth NVSwitches, while at the same time, each accelerator is also connected with an IB NIC, making it possible for all 8 accelerators to independently and concurrently RDMA data over their own dedicated 400 Gb/s NIC.

To analyze the decomposition of such IB-connected multi-node accelerators, we first take a 2-node H100 cluster as an example (abbreviated as H100-2). Since every two accelerators in H100-2 are either interconnected via ultra-high 7.2 TB/s (duplex) aggregated bandwidth NVLINK-based communication links or 800 GB/s (duplex) aggregated bandwidth IB-based communication links, we can simply model its topology as $K_{8\times2}$ and use the above mentioned $K_n - decomposition$ scheme for its topology decomposition, as illustrated in the left part of Figure 3.

However, this decomposition scheme will result in under-utilization of the bandwidth of intra-node NVLINK-based communication links. In a $K_{8\times2} - decomposition$ scheme there are $7\times8 = 56$ communication links within each node whereas $8\times8 = 64$ communication links from one node to the other. Consequently, each intra-node communication link will have a bandwidth of $7200/(56 * 2) \approx 64GB/s$, whereas each inter-node communication link will have a bandwidth of $\frac{800}{64*2} \approx 6.3$ GB/s, an order of magnitude lower than the intra-node bandwidth. Consequently, the utilization of intra-node bandwidth is bounded by inter-node bandwidth.

To this end, we propose to model the topology of H100-2 as two K_8 subgraphs inter-connected with 8 bidirectional or 16 unidirectional IB-based communication links (denoted as $(m - K_m - m)^u - decomposition$ where in this case $m = 8$ and $u = 2$). This decomposition scheme is based on the Hamiltonian decomposition of each node into a set of `Hamiltonian paths`. A Hamiltonian path is a path in a directed graph that visits each vertex exactly once. As illustrated in the right part of Figure 3, the theoretical bandwidth of each unidirectional intra-node communication link (gray) are roughly 64 GB/s while the theoretical bandwidth of each unidirectional inter-node communication link (blue) are approximately 50 GB/s-only a 28% difference which is significantly smaller than the over 9x difference if $K_{8\times2}$ decomposition is adopted.

Next, we prove that the 2-node decomposition scheme can be generalized to an arbitrary $u - node$ cluster. If we decompose two K_8 subgraphs into Hamiltonian paths, we can obtain 8 Hamiltonian cycles by connecting these paths end-to-end. Since the decomposed Hamiltonian paths are at the same time a row-complete Latin Square. In each row or column of a Latin Square the entries are mutually distinct. Consequently, all 8 accelerators will appear both at the start and the end of the decomposed Hamiltonian paths, utilizing all the $2 \times 8 = 16$ IB NICs.

Moreover, through mathematical induction, it is provable that the above decomposition scheme can scale up to an arbitrary number of H100 nodes inter-connected via IB switches. Suppose we have already decomposed a n-node cluster using the scheme, we can derive a decomposition of $n+1$-node cluster simply by two steps (ranks denote accelerators): (1) dis-connect the communication links from $rank\ (8n-8) \sim (8n-1)$ to $rank\ 0 \sim 7$; (2) connect $rank\ (8n-8) \sim (8n-1)$ to $rank\ (8n) \sim (8n+7)$ and connect $rank\ (8n) \sim (8n+7)$ to $rank\ 0 \sim 7$.

Other topologies, such as hypercubes, can also be decomposed into orthogonal ring data-paths via Hamiltonian decomposition. However, due to space limitations and the growing architectural emphasis on AlltoAll connectivity, this paper focuses primarily on the single-node and multi-node AlltoAll topologies analyzed in this section.

3.2 Decomposition of Ring AllGather

3.2.1 Problem Definition.

For simplicity, we assume that for each query chunk, its Q tensors keep stationary and its KV tensors are sent to or received from other accelerators in a ring-style data transfer. Based on the above assumption, the formulation of vanilla Ring Attention is composed of two separate components: a `Ring AllGather Communication Primitive` that assigns a query chunk at each accelerator and one ring-styled data transfer pattern for sending and receiving these chunks, as well as a `Chunk Placement Strategy` which defines the (initial) query chunk placement strategy on the accelerators. Thus, given a Ring AllGather primitive and a Chunk Placement Strategy, the communication and computation of the vanilla Ring Attention scheme are well formulated and fully determined for every iteration. As illustrated in Figure 4, vanilla Ring Attention has three desirable properties.

- **Accessibility**: After $n-1$ iterations of KV transfer and n iterations of Attention computation, every Q tensor can access every KV tensor (which are initially distributed among all accelerators) locally on its assigned accelerator.
- **Zero-copy**: No additional memory is consumed due to KV tensor duplication.
- **Load balance**: For bi-directional non-causal Attention, as long as the query chunks are distributed evenly among the accelerators, computation and memory load are balanced across all accelerators. For causal Attention, however, there are two types of workload balance methods, including Striped Attention [6] and Zig-zag Ring Attention [8].

The former two of the above properties correspond to the first component `Ring AllGather Communication Primitive` of the vanilla Ring Attention, ensuring the algorithmic correctness in Attention computation. The third one corresponds to the `Chunk Placement Strategy` of the vanilla Ring Attention, and affects the workload balance of each iteration. Then, we demonstrate that `Ring AllGather` can be decomposed into Multi-Ring AllGather while satisfying the first two properties (See Section 3.2.2). Lastly, the chunk placement strategies for Ring Attention (Striped Attention and Zig-zag Ring Attention) cannot be directly ported to TASP. Thus, we present a Zig-zag chunk placement strategy for perfect load-balance (See Section 3.2.3).

3.2.2 Multi-Ring AllGather

As shown in Figure 4, compared to the original Ring AllGather primitive that only assigns one query chunk for each accelerator, an m-ring Multi-Ring AllGather communication primitive assigns m query chunks for each accelerator and designates a specific ring-styled data transfer for each query chunk. The number of m can be chosen arbitrarily, as long as the original query can be evenly split into m query chunks. The ring-style data transfer for each query chunk can also be designed arbitrarily and mutually different.

Since it is shown in the previous section that the topology of modern accelerators can be decomposed into a set of mutually orthogonal ring datapaths, we can thus design a Multi-Ring AllGather communication primitive 100%-percent align with a topology decomposition. For example, we present a 7-ring Multi-Ring AllGather Communication specifically designed for $K_8 - decomposition$ of an 8-accelerator AlltoAll topology in Figure 4. Full communication link utilization can be achieved if the ring-styled data transfer are 100%-percent align with the ring datapaths.

However, to make sure this decomposition of Ring AllGather does not incur extra memory overhead or violate the algorithmic correctness of Attention calculation, we need to prove that it still adheres to the following properties:

Proof of Accessibility: For any Q_i tensor (assigned to accelerator s_0) and any KV_j tensor, since KV_j tensor traverses all accelerators along its assigned ring-styled data transfer h_0 (a Hamiltonian cycle), it will eventually reach accelerator s_0, allowing Q_i to attend to KV_j locally.

Proof of Zero-copy: Each KV tensor is assigned to exactly one ring-styled data transfer h_0 and is transferred solely along h_0, ensuring only one copy exists at any time.

3.2.3 Zig-zag Chunk Placement for TASP

The vanilla Ring Attention adopts a naive chunk placement strategy:

$$t[i] = KV[\frac{iS}{n}, \frac{(i+1)S}{n}) \text{ (S is sequence length)} \quad (3)$$

Translating into plain text, this strategy first evenly splits all the KV tensors into n continuous blocks and map each block to accelerator, given n accelerators.

However, when a causal mask is used, the naive chunk placement strategy will disrupt computation load balance.

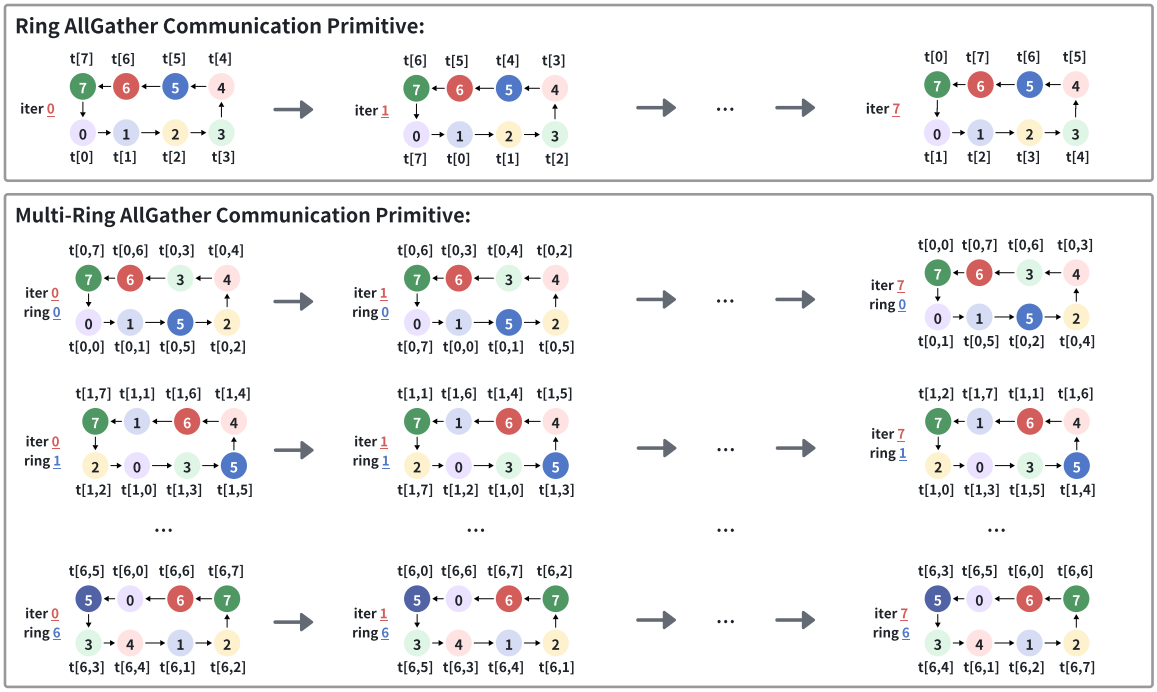


Figure 4: Top: The Ring AllGather communication primitive used in Ring Attention and its variants. Colored circles represent accelerators 0–7. Symbols $t[0] \sim t[7]$ denote the transferred KV blocks. Bottom: The Multi-Ring AllGather communication primitive designed for $K_8 - decomposition$. Each column illustrates one iteration with seven ring-styled data transfer for seven chunks of KV blocks. Symbol $t[i, j]$ represents the KV initially assigned at accelerator j and circulates across all accelerators following the i-th ring-styled data transfer. Iterations 2–6 and ring-styled transfer 2-5 are omitted for brevity.

If the naive chunk placement strategy is used together with causal masking, i accelerators will become idle while only $n - i$ accelerators perform computation at the i-th iteration. In contrast, the Zig-Zag chunk placement strategy of Megatron CP ensures that all accelerators perform an equal amount of computation at each iteration, through a more complicated mapping:

$$\begin{aligned} t[i] = & KV\left[\frac{iS}{2n}, \frac{(i+1)S}{2n}\right) \cup \\ & KV\left[\frac{(2n-i-1)S}{2n}, \frac{(2n-i)S}{2n}\right) \end{aligned} \tag{4}$$

For perfectly load-balanced causal Attention computation, the Zig-zag chunk placement for Zig-zag Attention cannot be directly utilized in TASP. However, via some necessary adjustments, similar strategy can also be derived. `Zig-zag TASP`: Each KV chunk $t[i, j]$ assigned at accelerator j and circulated via ring-style data transfer i is further split into two parts:

$$t[i,j][0] = KV[\frac{(2nj+i)S}{2n(n-1)}, \frac{(2nj+i+1)S}{2n(n-1)}) \tag{5}$$

$$t[i,j][1] = KV[S - \frac{(2nj+i+1)S}{2n(n-1)}, S - \frac{(2nj+i)S}{2n(n-1)}) \tag{6}$$

The proof of the perfect load-balance of Zig-zag TASP is as follows: Suppose for some accelerator r_0, at some iteration $iter_0$, it receives KV chunks $t_{KV}[i_0, j_0][0]$ and $t_{KV}[i_0, j_0][1]$ initially placed at accelerator j_0 via some ring data-path i_0 while at the same time the Q chunks $t_Q[i, r_0][0]$ and $t_Q[i, r_0][1]$ $(\forall i)$ are kept at r_0. If $j < r_0$, then the token indices of $t_Q[i, r_0][0]$ and $t_Q[i, r_0][1]$ $(\forall i)$ are larger than those of $t_{KV}[i_0, j_0][0]$ but smaller than those of $t_{KV}[i_0, j_0][1]$, making only $t_{KV}[i_0, j_0][0]$ needs to be attended, which is half of the transferred KV tensors via i_0 to accelerator r_0. On the contrary, if $j > r_0$, then the token indices of $t_Q[i, r_0][0]$ $(\forall i)$ are smaller than both of $t_{KV}[i_0, j_0][0]$ and $t_{KV}[i_0, j_0][1]$ while those of $t_Q[i, r_0][1]$ $(\forall i)$ are larger than both of $t_{KV}[i_0, j_0][0]$ and $t_{KV}[i_0, j_0][1]$, making only $t_Q[i, r_0][1]$ $(\forall i)$ needs to be attended, which is half of the query tokens. As a result, for any accelerator r_0, its computation load at any iteration is the same, leading to 100% load-balance.

4 Implementation of TASP

The implementation of TASP leverages two key techniques: (1) the use of AlltoAll collective communication for concurrent data transmission across multiple ring datapaths,

Algorithm 2 Calculate In/Out Mapping

Require: Looping matrix $looping[m][n]$
direction $d \in \{+1, -1\}$
Ensure: Mapping matrix $mapping[n][n]$
Initialize $mapping$ as $n \times n$ matrix filled with -1
for $i = 0$ to $m - 1$ **do**
  for $j = 0$ to $n - 1$ **do**
    $u \leftarrow looping[i][j]$
    $v \leftarrow looping[i][(j + d) \bmod n]$
    $mapping[u][v] \leftarrow i$
  end for
end for
return $mapping$

and (2) pre-computed routing tables to minimize CPU scheduling overhead.

4.1 Pre-Computed Routing Tables

To minimize CPU overhead during the distributed Attention computation, TASP pre-computes all routing tables before the forward pass. The `cal_in/out_mapping` functions as described in Algorithm 2 generate source and destination mappings for each send/receive based on a given topology decomposition scheme, eliminating the need for dynamic routing during the compute-intensive attention.

4.2 Concurrent Ring-styled Data Transfer via AlltoAll

Unlike Ring Attention's utilization of batched `sendRecv` collective, TASP can utilize a single `AlltoAll` collective operation to simultaneously transmit KV blocks across all active ring datapaths (this optimization is for $K_{m\times u}$-decomposition scheme only). This approach fully leverages the low-level optimization for `AlltoAll` collective implemented in the Communication Collectives Libraries (NCCL and RCCL).

The entire forward pass of TASP is illustrated in Algorithm 3, where each accelerator prepares send buffers according to a pre-computed routing table and receives corresponding KV blocks from all other accelerators in each iteration.

Algorithm 3 TASP Forward Pass

Require: q, k, v, *process_group*, *world_size*
Ensure: *out*, *lse*
// Initialize routing tables:
$looping \leftarrow gen\_hamilton\_circle(world\_size)$
out_mapping $\leftarrow$ `cal_out_mapping`(*looping*)
in_mapping $\leftarrow$ `cal_in_mapping`(*looping*)
para_size $\leftarrow$ *world_size* $- 1$
// Stack KV tensors:
this_kv $\leftarrow [stack(k[i], v[i])\ \forall i \in \{0, para\_size\}]$
// Initialize receive buffers:
next_kv $\leftarrow [empty\_like(this\_kv[0])\ \forall i \in \{0, para\_size\}]$
for *step* $= 0$ **to** *world_size* $- 1$ **do**
  // Extract K, V:
  $k \leftarrow$ [*this_kv*[i][0]], $v \leftarrow$ [*this_kv*[i][1]]
  if *step* $<$ *world_size* $- 1$ **then**
    // Prepare send/recv buffers using routing tables:
    send_buf $\leftarrow$ [*this_kv*[*out_mapping*[i]]]
    recv_buf $\leftarrow$ [*next_kv*[*in_mapping*[i]]]
    All-to-All(*send_buf*, *recv_buf*)
  end if
  // Concatenate KV blocks:
  $k_{\text{cat}} \leftarrow concat(k, \text{dim} = seqlen)$
  $v_{\text{cat}} \leftarrow concat(v, \text{dim} = seqlen)$
  // Compute Attention:
  output, *lse_curr* $\leftarrow$ `flash_attn`$(q, k_{\text{cat}}, v_{\text{cat}})$
  out, *lse* $\leftarrow$ `update_out_lse`(*out*, *lse*, *output*, *lse_curr*)
  if *step* $<$ *world_size* $- 1$ **then**
    // *Wait* for communication completion
    this_kv $\leftarrow$ *next_kv*, *next_kv* $\leftarrow$ *this_kv*
  end if
end for
return *out*, *lse*

5 Evaluation

5.1 Setup

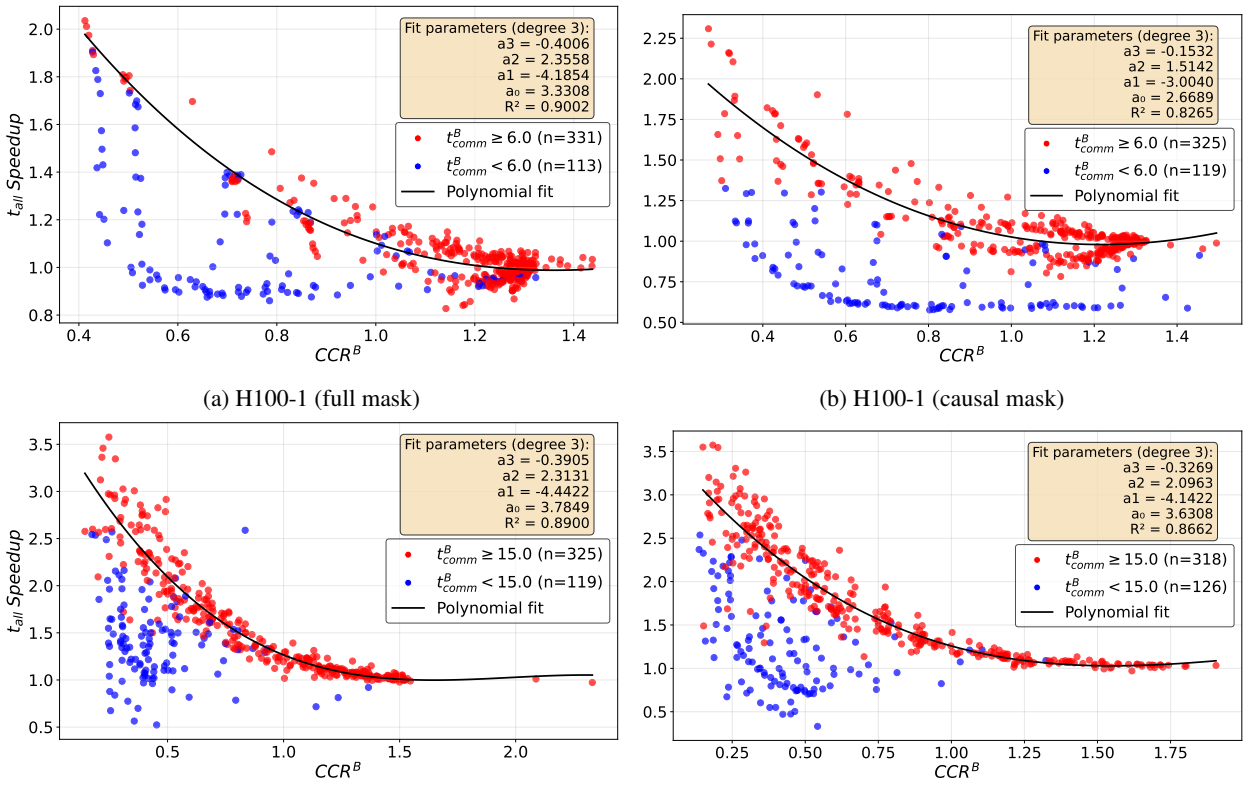
5.1.1 Testbed

We evaluate TASP on four distinct hardware platforms: (1) A single node with eight NVIDIA H100 GPUs interconnected via NVSwitch4 with 3.6TB/s bisection bandwidth. (2) A single node with eight AMD MI300X GPUs in a full-mesh topology via AMD Infinity Fabric with 896 GB/s aggregated bandwidth. (3) A two-node server, each node equipped with eight NVIDIA H100 GPUs. The two nodes are interconnected via 16 NVIDIA Mellanox Infiniband NICs (eight NICs for each node) with 800 GB/s (duplex) aggregate bandwidth per node. (4) A four-node server with the same intra-node and inter-node configurations as (3).

5.1.2 Benchmark.

We evaluate the performance of TASP on three sets of approximately 400 test cases with various input sizes (1287 test cases in total). Specifically, the batch size ranges from 1 to 128, the sequence length from 3k to 1M, and the number of Attention heads in $\{4, 12, 20\}$ with a head dimension of 64. We use BF16 precision for experiments, nevertheless, it is straightforward to adapt to other precisions.

For different hardware platforms, we set different maximum sequence lengths for test cases. For all single-node platforms, we use a maximum sequence length of 430K. For two-node and four-node platforms, the maximum sequence lengths are 490K and 1M, respectively.



(a) H100-1 (full mask)

(b) H100-1 (causal mask)

(c) MI300X-1 (full mask)

(d) MI300X-1 (causal mask)

Figure 5: t_{all} speedup v.s. CCR^B. The test cases are divided into two sets (colored red and blue) to distinguish the data transfer size impact on our communication optimization. For cases with sufficient data volume to saturate the bandwidth (dots in red), TASP achieves a significant t_{all} speedup compared to baselines. Furthermore, the speedup exhibits a decreasing trend with the increase of CCR^B, as the optimized part lies in communication.

5.1.3 Baseline.

We use Ring Attention [10] as our baseline method for cases with non-causal masks and Zig-zag Ring Attention [8] for cases with causal masks.

- Ring Attention [10]. The fundamental distributed Attention implementation specialized for sequence parallelism.
- Zig-zag Ring Attention [28]. The distributed Attention implementation modified for Attention with causal masks, to address the workload imbalance in Ring Attention.

For the fairness of comparison, all methods (including TASP) are implemented in PyTorch [29] and utilize the same Flash Attention [14] implementation. We adopt the open source code [30] of Ring Attention and Zig-zag Ring Attention for baseline comparison.

5.1.4 Metrics.

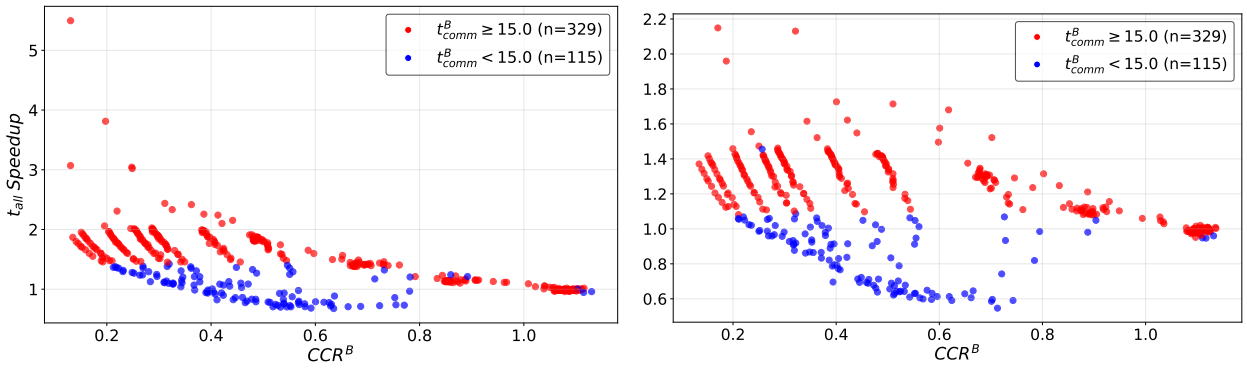
We collect three metrics for evaluation, *i.e.*, t_{comm} denoting the total communication time, t_{comp} denoting the total computation time and t_{all} denoting the overall latency of each method. Since we implement TASP and baseline methods using two CUDA/HIP streams to overlap communication and computation, we use the execution time of each stream counted via CUDA/HIP events to derive t_{comm} and t_{comp}, and adding them up to derive t_{all}.

Moreover, to explore the impact of the (latency) ratio of computation and communication on the performance of our proposed method, we use CCR^B which is the computation-to-communication ratio of the baseline method at each test case as the independent variable (x-variable) in our analysis.

5.2 Single-node Evaluation

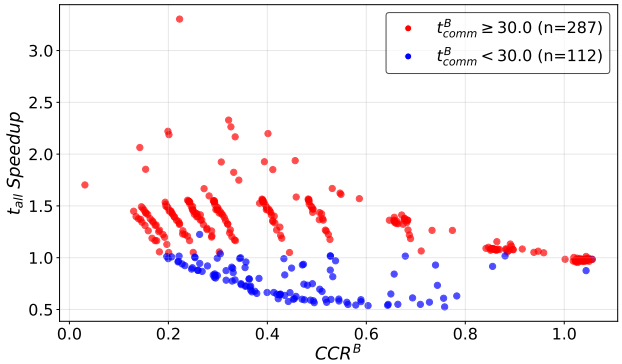
We first evaluate the single-node performance of TASP, with the eight NVIDIA H100 GPU server (abbreviated as H100-1) representing the switch-based topology and the eight AMD-MI300X GPU server (abbreviated as MI300X-1) representing the full-mesh topology.

As depicted in Figure 5, we divide the test cases into two sets to evaluate the performance of TASP. The first set includes the test cases with relatively large communication latency ($t^B_{comm} \geq T$, T is a threshold varying with

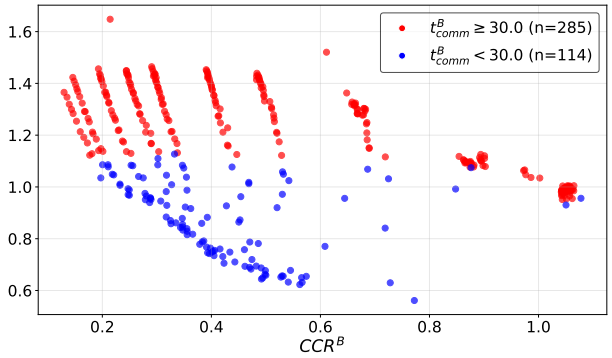


(a) H100-2 ($K_{8\times2}$)

(b) H100-2 ($(8-K_8-8)^2$)



(c) H100-4 ($K_{8\times4}$)



(d) H100-4 ($(8-K_8-8)^4$)

Figure 6: t_{all} speedup v.s. CCR^B. Similar to the single-node evaluation, the test cases are divided into two sets (colored blue and red) by communication volume size, and the speedup exhibits a decreasing trend with the increase of CCR^B. As the number of accelerator node scales up, TASP (with $(m-K_m-m)^u$-decomposition scheme) maintains an almost constant speedup ratio, demonstrating better scalability.

hardware platform's bandwith capacity), which can fully occupy the interconnection bandwidth of accelerators, and the second set includes the test cases with $t^B_{comm} < T$, failing to saturate the bandwidth. In Figure 5, the first set is depicted in red and the second set in blue.

We observe that TASP achieves significant speedups across the first set regarding the overall time t_{all}, yielding an average speedup of 1.05× and 1.08× for full-mask and casual-mask cases on H100-1, and an average speedup of 1.57× and 1.68× for full-mask and casual-mask cases on MI300X-1, respectively. Moreover, TASP achieves up to **2.31**× maximum speedup on H100-1 and **3.58**× maximum speedup on MI300X-1 over the baselines, respectively. For the communication time t_{comm} alone, TASP achieves a maximum of a **4.72**× speedup on MI300X-1 and **2.87**× speedup on H100-1 against the baseline methods (not shown in the figure).

The impact of CCR^B**.** Through evaluation, we find that a lower computation-communication ratio (CCR^B) enables TASP to be more communication efficient than baseline methods and yield a higher speedup in t_{all}. The lower the CCR^B, the baseline method is more communication-bound, and the higher the t_{all} speedup ratio is achieved.

The impact of mask pattern. TASP achieves a higher speedup on causal-masked test cases on both H100-1 and MI300X-1, since causal-masked Attention has theoretically two times lower CCR^B than full-masked Attention.

The impact of hardware platform. Besides, we note that TASP achieves much superior performance optimization on MI300X-1 over H00-1 in most test cases. This is not only attributed to the much lower intra-node communication bandwidth of MI300X-1 that makes it more communication-bound in most test cases, but also aligns with our theoretical analysis of full-mesh topology *v.s.* switch-based topology in Section 3.1.

5.3 Multi-node Evaluation

The multi-node evaluation results are depicted in Figure 6, and the two-node and four-node servers equipped with H100 GPUs are denoted as H100-2 and H100-4, respectively. To demonstrate the efficiency of the proposed topology decomposition method for multiple nodes, we compare the two distinct schemes, *i.e.*, $K_{m\times u}$-decomposition and $(m-K_m-m)^u$-decomposition, where m denotes the accelerator number within a node and u denotes the node

number. For simplicity, we only present the results of the full-masked TASP on those multi-node platforms.

Similar to the single-node evaluation, we also present the results in two sets (red and blue) to distinguish the communication data volume impact. In the following, we focus on the test cases with sufficient communication sizes (the set in red). We observe an average speedup of 1.43× with $K_{8\times2}$-decomposition and 1.20× with $(8 - K_8 - 8)^2$-decomposition on H100-2, and an average speedup of 1.27× with $K_{8\times4}$-decomposition and 1.20× with $(8 - K_8 - 8)^4$-decomposition on H100-4, respectively.

According to Section 3.1, the $(m - K_m - m)^u$-decomposition theoretically outperforms the $K_{m\times u}$-decomposition, due to the topology-aware communication link utilization. However, the experiments yield the opposite results when there are only two nodes. This is due to the difference in the implementation of these two decomposition schemes, as the $K_{m\times u}$-decomposition scheme can directly call the AlltoAll API from NCCL/RCCL while the $(m-K_m-m)^u$-decomposition scheme necessitates utilizing the batched *SendRecv* to customize the communication, which leads to communication efficiency degradation due to lack of low-level optimization. However, we observe that the $(m-K_m-m)^u$-decomposition is much more scalable than $K_{m\times u}$-decomposition, as its speedup remains almost unchanged when transferring from H100-2 to H100-4, while the speedup of $K_{m\times u}$-decomposition significantly reduces.

5.4 Overhead Analysis

The overhead of TASP lies in both computation and communication. In terms of communication, TASP employs a complex communication primitive (the AlltoAll collective). The batched *SendRecv* employed by $(m - K_m - m)^u$-decomposition-based TASP is also more complicated than the batched *SendRecv* employed in vanilla Ring Attention (m vs 2 simultaneous SendRecvs are batched). While effective in utilizing all orthogonal ring datapaths, this incurs higher runtime overhead compared. Consequently, this overhead degrades overall performance in scenarios with smaller communication volume, such as the test cases with small batch sizes and short sequence lengths, as reflected by the blue data points in Figure 5 and Figure 6. However, in other scenarios (approximately 3/4 of all test cases), this overhead is negligible compared to the communication latency.

The additional computational overhead of TASP primarily stems from its fine-grained KV Cache partitioning. For instance, in a K_8-decomposition–based TASP scheme, the KV Cache is split into 56 chunks (7 chunks per accelerator), whereas vanilla Ring Attention requires only 8 chunks. Although the routing and mapping of these KV chunks can be precomputed, the increased chunk number introduces slightly higher CPU overhead compared to vanilla Ring Attention. Fortunately, this CPU overhead does not affect the overall execution time, as the kernel launch latency of TASP can be fully overlapped with GPU computation—owing to the absence of GPU–CPU synchronization.

Beyond CPU overhead, our current implementation of TASP also introduces minor GPU computation overhead compared to vanilla Ring Attention, as all KV Cache chunks must be concatenated on each rank before invoking the flash-attn kernel for Attention computation. Empirical measurements indicate that this additional overhead is modest, resulting in only a 1%–5% slowdown in the total computation time (t_{comp}). Nevertheless, we note that such overhead can be further minimized through kernel optimization techniques.

6 Related Work

Efficient attention mechanism. The attention mechanism in Transformer models has long suffered from low device efficiency on modern accelerators. To bridge this gap, researchers have proposed various improvements. For instance, Linformer [31] and Sparse Transformer [32] attempted to replace standard attention computation with linear approximation and sparsification methods, respectively. Rabe and Staats [33] introduced a memory-efficient self-attention algorithm that reduces memory complexity from $O(n^2)$ to $O(1)$. FlashAttention [14, 34] provided an efficient implementation by leveraging custom CUDA kernels and optimizations tailored to the GPU memory hierarchy.

Attention for long sequences. As the context length of LLMs extends to the million-token level, the attention mechanism faces significant computational overhead and memory pressure. Data parallelism [35], model parallelism [36], pipeline parallelism [37], tensor parallelism [28], and expert parallelism [38], cannot meet the requirements. To address this challenge, Li et al. [4, 39] proposed sequence parallelism. Ulysses [5] improves the communication efficiency of SP by replacing all-gather and reduce-scatter operations with smaller AlltoAll operations. Similarly, LightSeq [40] achieves communication-computation overlap by partitioning computational tasks into bubbles. Ring Attention [10] involves partitioning queries and the KV Cache along the sequence dimension across different accelerators, keeping queries stationary while rotating KV Cache segments through ring communication. Striped Attention [6] addresses load imbalance by employing a striped partitioning scheme. Megatron CP [8] can achieve fully balanced loads across accelerators, but it becomes ineffective when applied to sparse attention mechanisms. In short, Ring Attention and its variants are constrained by the limited communication efficiency inherent to ring-based communication paradigms.

7 Limitation and Discussion

Our proposed algorithm's effectiveness in long-sequence multi-accelerator inference is closely tied to the CCR: when the ratio is low, our communication-focused optimizations target the primary bottleneck (communication) to deliver notable performance gains; yet in compute-intensive attention scenarios—especially with a CCR exceeding 1.5—such gains vanish.

This does not indicate flaws in the algorithm itself, but rather that the overall latency of such distributed Attention method is shaped by multiple factors. When the impact of other factors surpasses that of the communication overhead, the effectiveness of communication-oriented optimizations will naturally diminish, which clarifies the algorithm's strengths in collaborating with computation-oriented optimizations such as sparse Attention.

8 Conclusion

We present TASP, a novel SP method that leverages the Hamiltonian decomposition of complete directed graphs to optimize the attention prefill phase for long-context LLMs. To overcome the limitations of existing methods, TASP employs two steps of optimization: topology decomposition and primitive decomposition to fully utilize the native interconnect topology of AI accelerators. Through experimental evaluation, TASP achieves higher communication efficiency than Ring Attention (and Zig-zag Ring Attention) in both single-node and multi-node topologies, underscoring the critical importance of topology-aware optimization in designing high-performance SP methods.

References

[1] Jiaheng Liu, Dawei Zhu, Zhiqi Bai, Yancheng He, Huanxuan Liao, Haoran Que, Zekun Wang, Chenchen Zhang, Ge Zhang, Jiebin Zhang, et al. A comprehensive survey on long context language modeling. *arXiv preprint arXiv:2503.17407*, 2025.

[2] Yusen Zhang, Ruoxi Sun, Yanfei Chen, Tomas Pfister, Rui Zhang, and Sercan Arik. Chain of agents: Large language models collaborating on long-context tasks. *Advances in Neural Information Processing Systems*, 37:132208–132237, 2024.

[3] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[4] Shenggui Li, Fuzhao Xue, Chaitanya Baranwal, Yongbin Li, and Yang You. Sequence parallelism: Long sequence training from system perspective. *arXiv preprint arXiv:2105.13120*, 2021.

[5] Sam Ade Jacobs, Masahiro Tanaka, Chengming Zhang, Minjia Zhang, Shuaiwen Leon Song, Samyam Rajbhandari, and Yuxiong He. Deepspeed ulysses: System optimizations for enabling training of extreme long sequence transformer models. *arXiv preprint arXiv:2309.14509*, 2023.

[6] William Brandon, Aniruddha Nrusimha, Kevin Qian, Zachary Ankner, Tian Jin, Zhiye Song, and Jonathan Ragan-Kelley. Striped attention: Faster ring attention for causal transformers. *arXiv preprint arXiv:2311.09431*, 2023.

[7] Dacheng Li, Rulin Shao, Anze Xie, Eric P Xing, Xuezhe Ma, Ion Stoica, Joseph E Gonzalez, and Hao Zhang. Distflashattn: Distributed memory-efficient attention for long-context llms training. *arXiv preprint arXiv:2310.03294*, 2023.

[8] NVIDIA. *Megatron Context Parallelism*, 2023. [Online; accessed 25-Aug-2025].

[9] Rajeev Thakur, Rolf Rabenseifner, and William Gropp. Optimization of collective communication operations in mpich. *The International Journal of High Performance Computing Applications*, 19(1):49–66, 2005.

[10] Hao Liu, Matei Zaharia, and Pieter Abbeel. Ring attention with blockwise transformers for near-infinite context, 2023. *URL https://arxiv.org/abs/2310.01889*, 7, 1889.

[11] Yingfa Chen, Yutong Wu, Xu Han, Zhiyuan Liu, and Maosong Sun. Cost-optimal grouped-query attention for long-context llms. *arXiv e-prints*, pages arXiv–2503, 2025.

[12] Joshua Ainslie, James Lee-Thorp, Michiel De Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. *arXiv preprint arXiv:2305.13245*, 2023.

[13] Dingzirui Wang, Xuangliang Zhang, Keyan Xu, Qingfu Zhu, Wanxiang Che, and Yang Deng. Multi-layer attention is the amplifier of demonstration effectiveness. *arXiv preprint arXiv:2508.00385*, 2025.

[14] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems*, 35:16344–16359, 2022.

[15] Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

[16] Heng Liao, Jiajin Tu, Jing Xia, Hu Liu, Xiping Zhou, Honghui Yuan, and Yuxing Hu. Ascend: a scalable and unified architecture for ubiquitous deep neural network computing: Industry track paper. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 789–801. IEEE, 2021.

[17] Alan Smith and Vamsi Krishna Alla. Amd instinct™ mi300x: A generative ai accelerator and platform architecture. *IEEE Micro*, 2025.

[18] Roman Kaplan. Intel gaudi 3 ai accelerator: Architected for gen ai training and inference. In *2024 IEEE Hot Chips 36 Symposium (HCS)*, pages 1–16. IEEE, 2024.

[19] Jack Choquette, Edward Lee, Ronny Krashinsky, Vishnu Balan, and Brucek Khailany. 3.2 the a100 datacenter gpu and ampere architecture. In *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, volume 64, pages 48–50. IEEE, 2021.

[20] Pengfei Zuo, Huimin Lin, Junbo Deng, Nan Zou, Xingkun Yang, Yingyu Diao, Weifeng Gao, Ke Xu, Zhangyu Chen, Shirui Lu, et al. Serving large language models on huawei cloudmatrix384. *arXiv preprint arXiv:2506.12708*, 2025.

[21] Myung M Bae and Bella Bose. Edge disjoint hamiltonian cycles in k-ary n-cubes and hypercubes. *IEEE Transactions on Computers*, 52(10):1271–1284, 2003.

[22] Roman Čada, Tomáš Kaiser, Moshe Rosenfeld, and Zdeněk Ryjáček. Hamiltonian decompositions of prisms over cubic graphs. *Discrete Mathematics*, 286(1):45–56, 2004. Cycles and Colourings.

[23] Chutima Saengchampa and Chariya Uiyyasathian. On hamiltonian decompositions of complete 3-uniform hypergraphs. *Discrete Mathematics*, 347(12):114197, 2024.

[24] Lenhard L Ng. Hamiltonian decomposition of complete regular multipartite digraphs. *Discrete Mathematics*, 177(1):279–285, 1997.

[25] Jiuqiang Liu. Hamiltonian decompositions of cayley graphs on abelian groups of even order. *Journal of Combinatorial Theory, Series B*, 88(2):305–321, 2003.

[26] Timothy W Tillson. A hamiltonian decomposition of k2m*, 2m$\geq$8. *Journal of Combinatorial Theory, Series B*, 29(1):68–74, 1980.

[27] Alexander Ishii and Ryan Wells. The nvlink-network switch: Nvidia's switch chip for high communication-bandwidth superpods. In *2022 IEEE Hot Chips 34 Symposium (HCS)*, pages 1–23, 2022.

[28] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

[29] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[30] Zilin Zhu. ring-flash-attention. `https://github.com/zhuzilin/ring-flash-attention`, 2024.

[31] Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

[32] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

[33] Markus N Rabe and Charles Staats. Self-attention does not need $o(n^2)$ memory. *arXiv preprint arXiv:2112.05682*, 2021.

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

[35] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, et al. Large scale distributed deep networks. *Advances in neural information processing systems*, 25, 2012.

[36] Zhihao Jia, Matei Zaharia, and Alex Aiken. Beyond data and model parallelism for deep neural networks. *Proceedings of Machine Learning and Systems*, 1:1–13, 2019.

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

[38] Siddharth Singh, Olatunji Ruwase, Ammar Ahmad Awan, Samyam Rajbhandari, Yuxiong He, and Abhinav Bhatele. A hybrid tensor-expert-data parallelism approach to optimize mixture-of-experts training. In *Proceedings of the 37th International Conference on Supercomputing*, pages 203–214, 2023.

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

[40] Dacheng Li, Rulin Shao, Anze Xie, Eric P Xing, Joseph E Gonzalez, Ion Stoica, Xuezhe Ma, and Hao Zhang. Lightseq:: Sequence level parallelism for distributed training of long context transformers. In *Workshop on Advancing Neural Network Training: Computational Efficiency, Scalability, and Resource Optimization*, 2023.