MLSys 2026
Ninth Annual Conference on Machine Learning and Systems

db-SP: Accelerating Sparse Attention for Visual Generative Models with Dual-Balanced Sequence Parallelism

Siqi Chen*[1], Ke Hong*[1], Tianchen Zhao[1], Ruiqi Xie[1], Zhenhua Zhu[1], Xudong Zhang[1], Yu Wang[1]

[1]Tsinghua University, Beijing, China

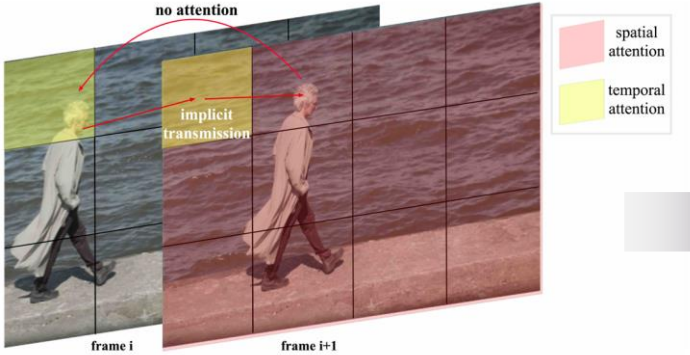
*Equal contribution

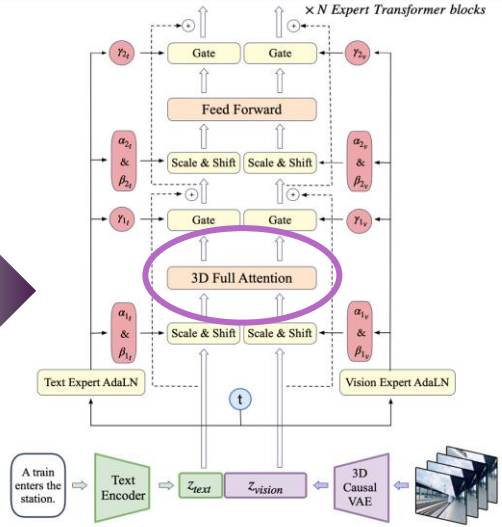
Corresponding to Yu Wang <yu-wang@tsinghua.edu.cn>

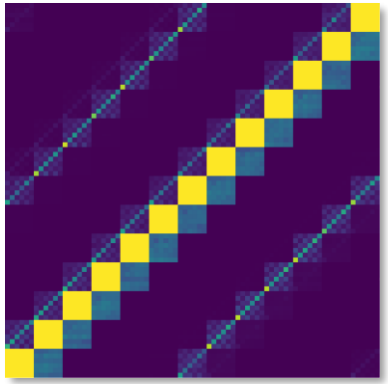
Background

Visual Generative Models

Mainstream video generation models are built on the Diffusion Transformer (DiT) architecture, with their attention mechanism evolving from factorized spatial-temporal attention to integrated 3D full attention.







Inherent sparsity of attention map

[1] Z. Yang, et al. “CogVideoX: Text-to-Video Diffusion Mod els with An Expert Transformer”.

Background

Block-wise Sparse Attention for Visual Generative Models

Previous works demonstrate that the block-wise sparse attention reduces computational costs while maintaining acceptable generation quality.

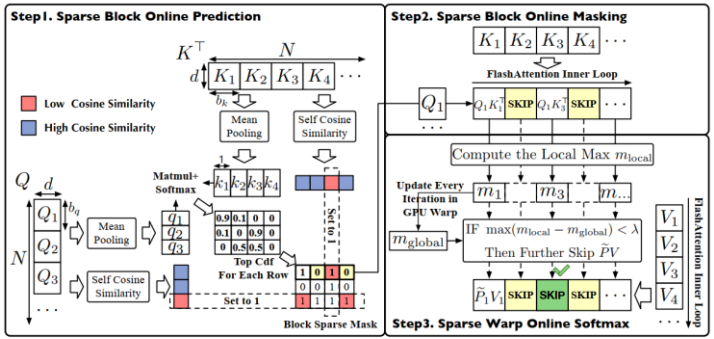
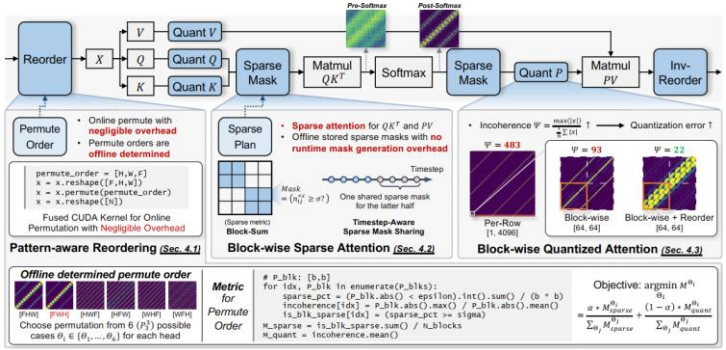


Figure 3. Workflow of SpargeAttn.

SpargeAttn (dynamic sparse attention)



PAROAttention (static sparse attention)

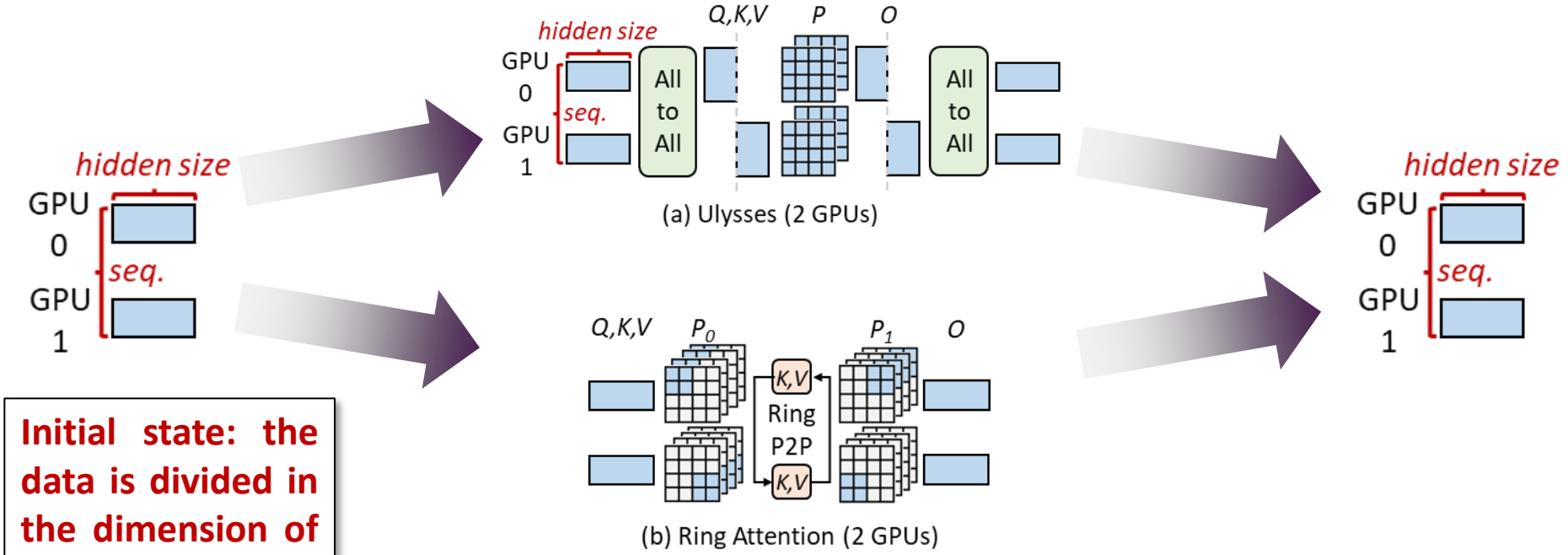
Block-wise sparse pattern is algorithm-friendly and hardware-friendly.

[1] J. Zhang, et al. “SpargeAttn: Accurate Sparse Attention Accelerating Any Model Inference”, ICML’25.

[2] T. Zhao, et al. “PAROAttention: Pattern-Aware ReOrdering for Efficient Sparse and Quantized Attention in Visual Generation Models”, NeurIPS’25.

Background

Sequence Parallelism

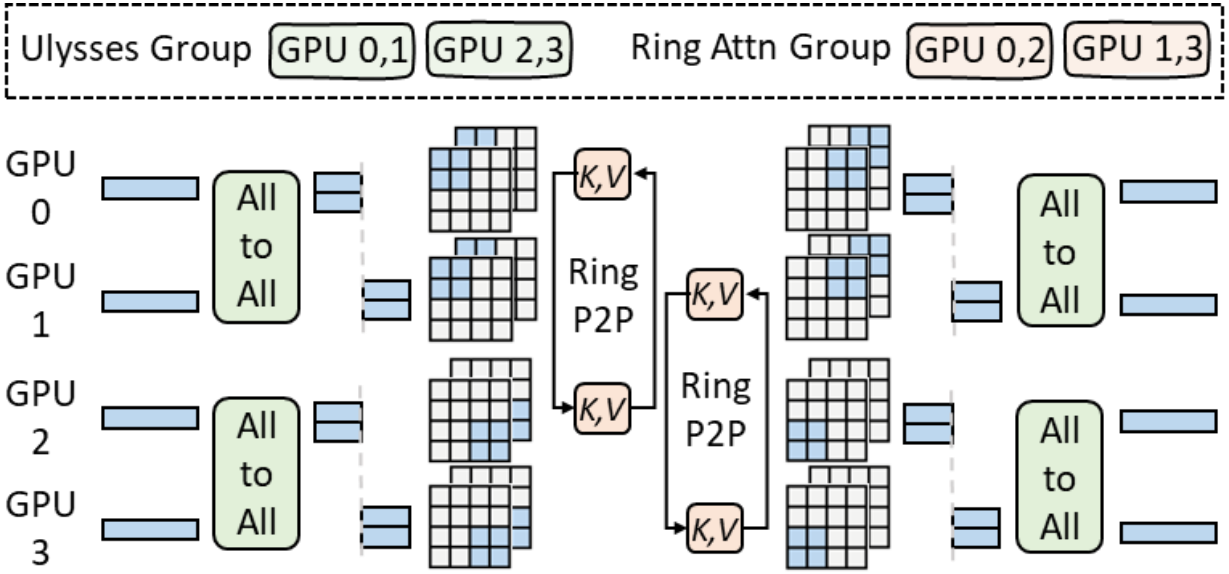


Initial state: the data is divided in the dimension of sequence

Background

Sequence Parallelism

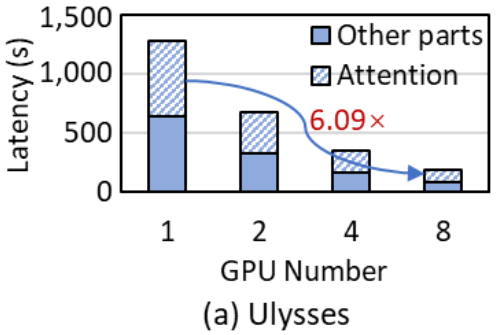
Initial state: the data is divided in the dimension of sequence



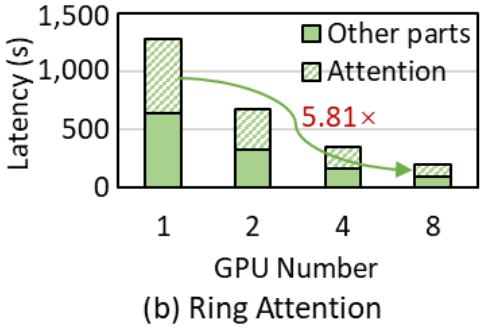
(c) Unified Sequence Parallelism on 4 GPUs (Ulysses 2 × Ring Attention 2)

Motivation

Sparse Attention with Sequence Parallelism



(a) Ulysses



(b) Ring Attention

When applied the block-wise sparsity, the sparse attention still occupies over 50% of the total inference latency.

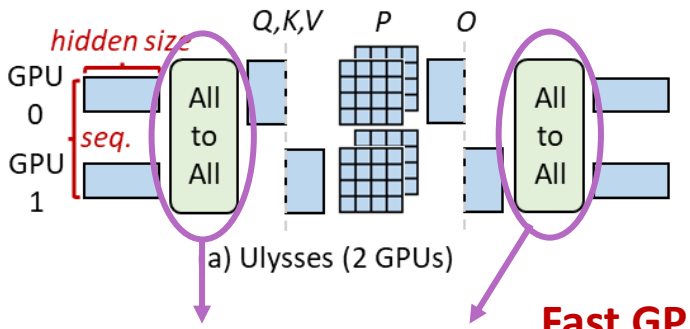
Sparse attention is still the main **bottleneck** and applying sequence parallelism is rewarding.

Existing sequence parallelism achieves limited reduction in latency when scaling the GPU number up.

Directly applying sequence parallelism to sparse attention leads to **workload imbalance** across GPUs.

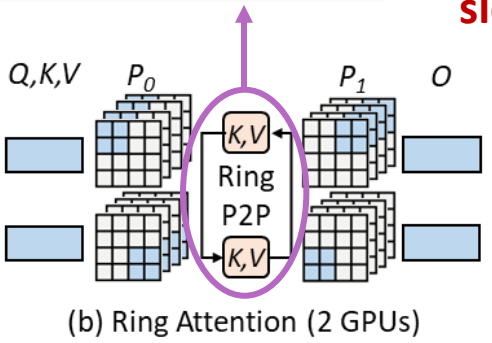
Motivation

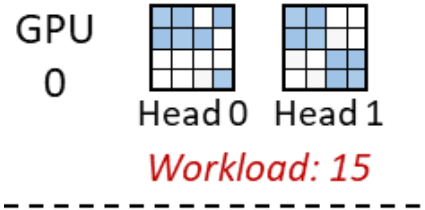
Sparse Attention with Sequence Parallelism



Data synchronization

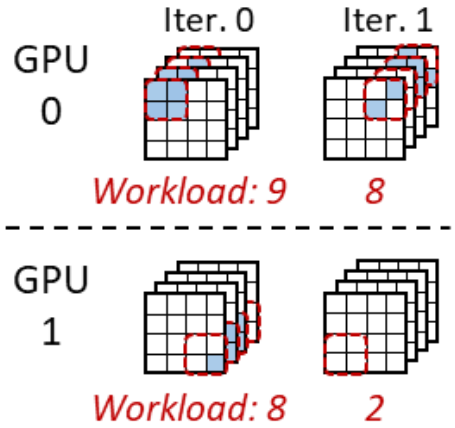
Fast GPUs need to wait the slowest GPU





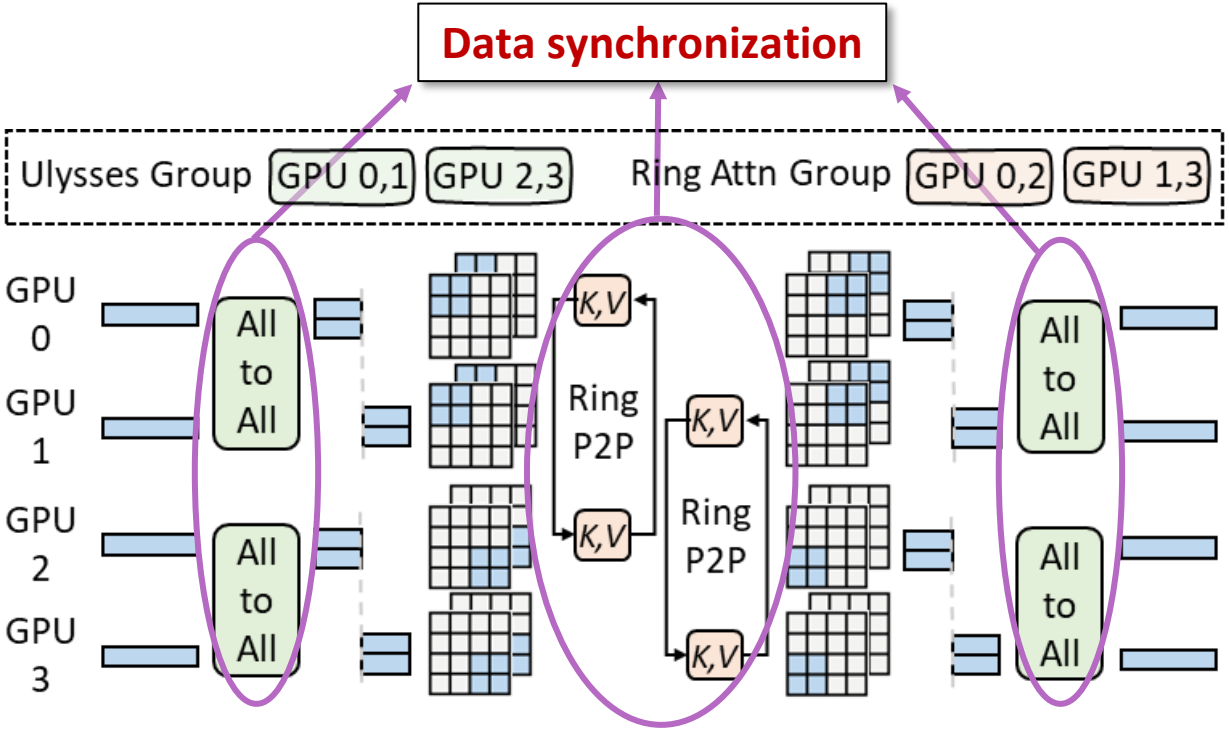
Head-level imbalance

Block-level imbalance



Motivation

Sparse Attention with Sequence Parallelism



Fast GPUs need to wait the slowest GPU

Dual-level imbalance

Motivation

Sparse Imbalance Ratio (ρs)

Key insights:
1. Data synchronization makes the fast GPUs need to wait the slowest.
2.The number of dense blocks represents the computational workload.

$$\rho_s = \frac{\sum_{i\in\{0,..,N-1\}}\left(\max_{g\in\mathcal{G}}\left(b_{i,g}(\mathbf{s},\mathbf{p})\right)\right)}{\sum_{i\in\{0,..,N-1\}}\sum_{g\in\mathcal{G}} b_{i,g}(\mathbf{s},\mathbf{p})/|\mathcal{G}|},$$

real workload

perfectly balanced workload

N synchronization points

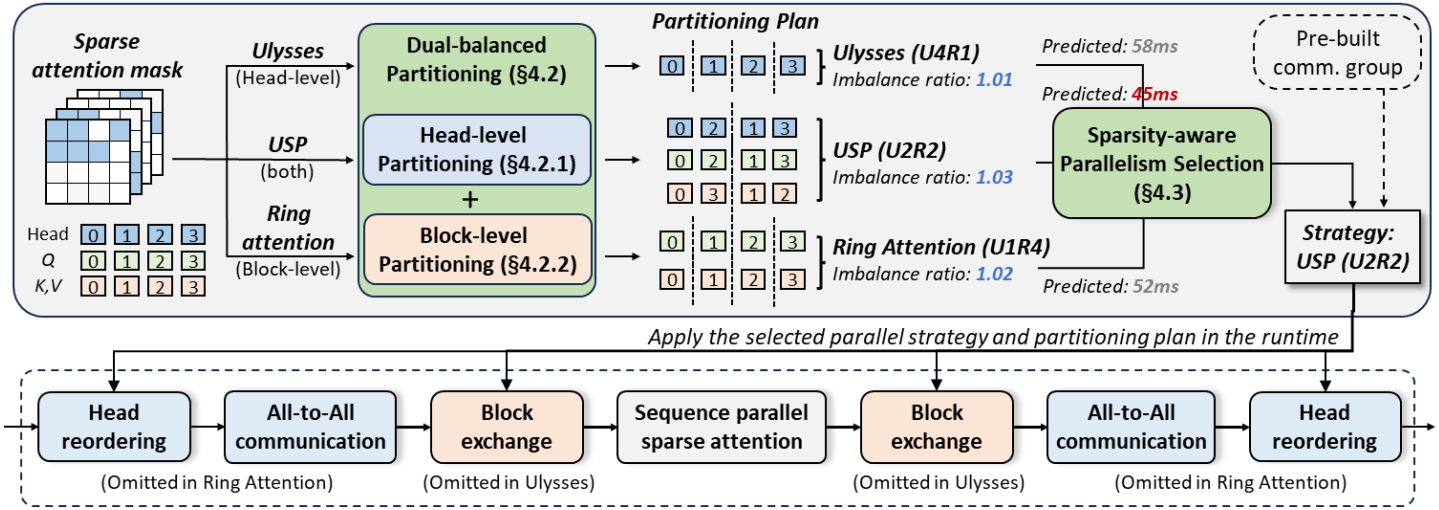
GPU number

sparse pattern s and parallelism strategy p

ρs ranges from 1.159 to 1.513, indicating potentials for acceleration

Overview

db-SP: Dual-Balanced Sequence Parallelism



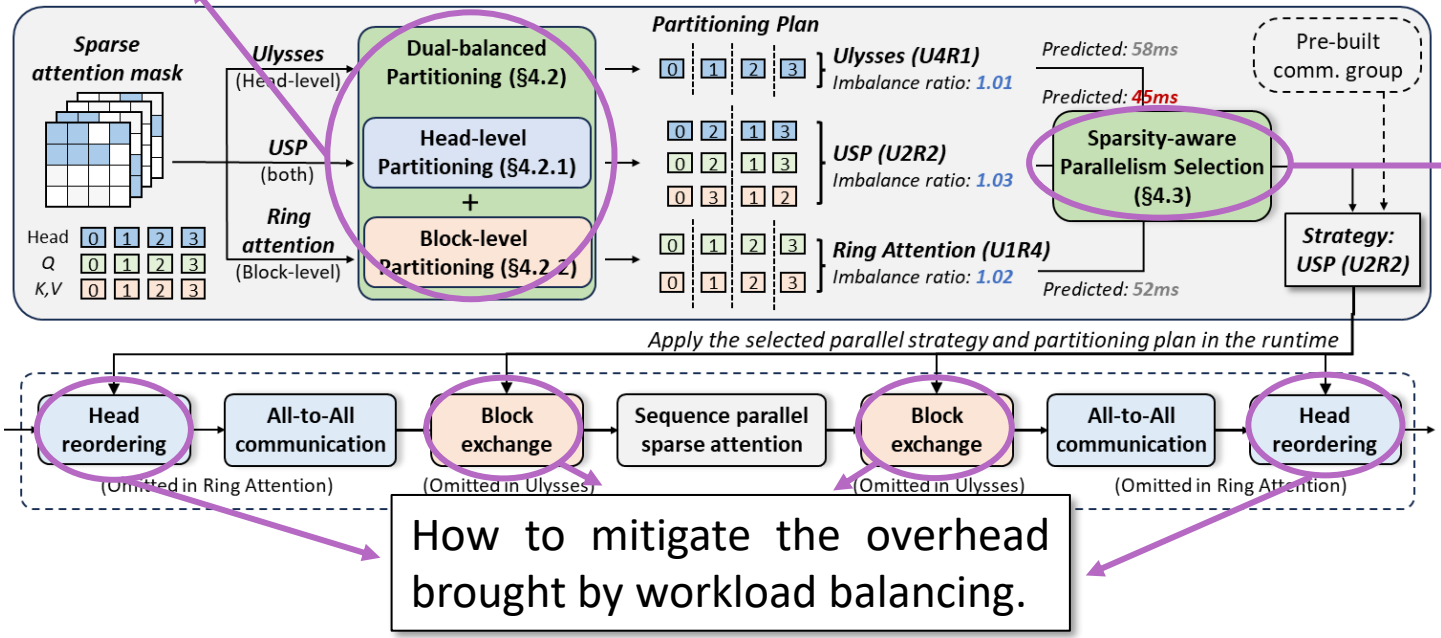
db-SP reorders Q, K, V data in dual-level to guarantee a balanced computation workload across GPUs for sparse attention.

Overview

Design space

How to design a general dual-balanced parallelism method.

Target: minimize ρ_s



How to choose parallelism strategy against dynamic sparsity.

How to mitigate the overhead brought by workload balancing.

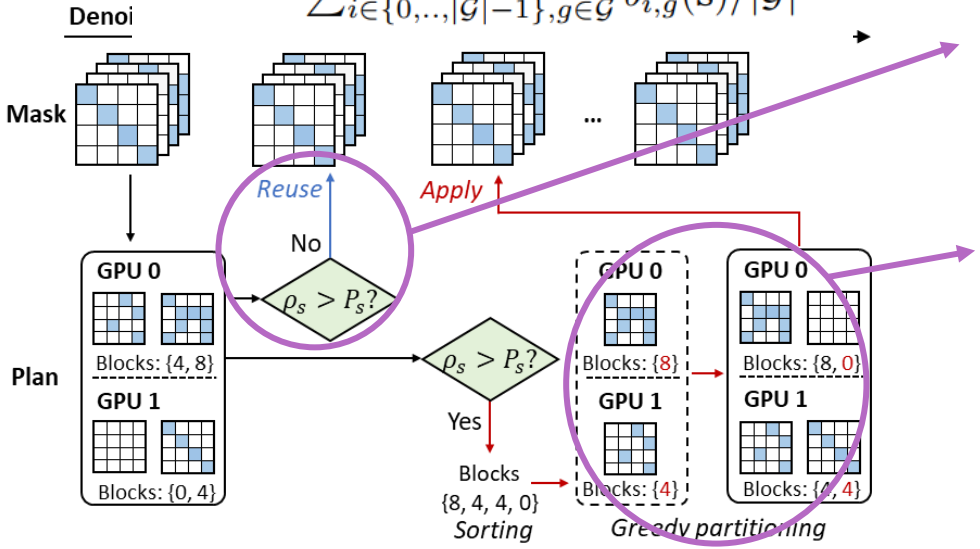
Solution

Dual-Balanced Partitioning

Target: minimize ρ_s

Number of dense blocks represents computational workload

$$\rho_s = \frac{\sum_{i\in\{0,..,|\mathcal{G}|-1\}}(\max_{g\in\mathcal{G}}(b_{i,g}(\mathbf{s})))}{\sum_{i\in\{0,..,|\mathcal{G}|-1\},g\in\mathcal{G}} b_{i,g}(\mathbf{s})/|\mathcal{G}|},$$



Head-level Partitioning (Ulysses)

Overhead mitigation:
Reuse the partitioning plan across denoising steps.

Partitioning Approach:

1. Sort the heads in descending order
2. Iteratively assign each sorted head to the specific GPU

Result: ρs: 1.025 on average (max 1.05)

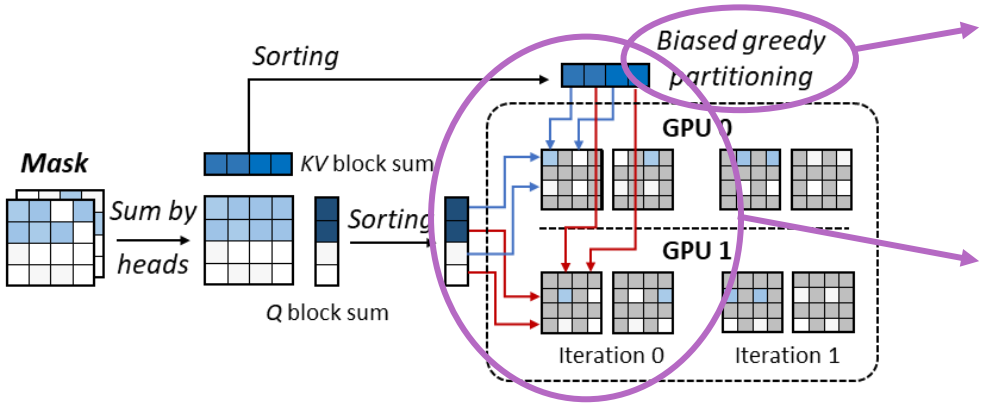
Solution

Dual-Balanced Partitioning

Target: minimize ρ_s

Number of dense blocks represents computational workload

$$\rho_s = \frac{\max_{g\in\mathcal{G}}(b_{0,g}(\mathbf{s}))}{\sum_{g\in\mathcal{G}} b_{0,g}(\mathbf{s})/|\mathcal{G}|}, b_{0,g}(\mathbf{s}) = \sum_{j\in\mathcal{H}_g} b_j^{\text{head}}(\mathbf{s}),$$



Result: ρs: <1.01 on average

Block-level Partitioning (Ring Attention)

Overhead mitigation:
Introduce a reward factor Rb to encourage the Q and K, V to reside on the initial GPU

Partitioning Approach:

1. Sum by column and row
2. Sort the sums in descending order
3. Iteratively assign each sorted head to the specific GPU

Solution

Dual-Balanced Partitioning

Target: minimize ρ_s

Number of dense blocks represents computational workload

Algorithm 1 Dual-Balancing Partitioning

Input: sparse pattern **s**, parallel strategy **p** (UxRy), reusing threshold P_s, partitioning reward R_b
head_partitioning_plan ← *old_head_partitioning_plan*
// Head-level partitioning
if ($x > 1$) and (cal_imbalance_ratio(**s**) > P_s) **then**
  blocks_per_head ← get_blocks_per_head(**s**)
  sorted_head_id ← desc_argsort(*blocks_per_head*)
  sum_per_gpu ← $\{0\}_x$
  for *head_id* in enumerate(*sorted_head_id*) **do**
    gpu_id ← argmin(*sum_per_gpu*)
    head_partitioning_plan [*gpu_id*].append(*head_id*)
    sum_per_gpu.update()
  end for
end if
q_partitioning_plan,*kv_partitioning_plan* ← ∅
// Block-level partitioning
if $y > 1$ **then**
  // Assume workload is balanced across heads
  blocks_accum_by_head ← sum_by_head(**s**)
  blocks_per_q ← sum_by_kv(*blocks_accum_by_head*)
  sorted_q_id ← desc_argsort(*blocks_per_q*)
  sum_per_gpu ← $\{0\}_y$
  for *q_id* in enumerate(*sorted_q_id*) **do**
    // Perform biased greedy partitioning with reward
    biased_sum_per_gpu ← [$s - R_b$ if g is initial_gpu
      for g, s in enumerate(*sum_per_gpu*)]
    gpu_id ← argmin(*biased_sum_per_gpu*)
    q_partitioning_plan[*gpu_id*].append(*q_id*)
    sum_per_gpu.update()
  end for
  // Repeat to partition K, V blocks
  blocks_per_kv ← sum_by_q(*blocks_accum_by_head*)
  ...
end if
Output: *head_partitioning_plan*, *q_partitioning_plan*, *kv_partitioning_plan*

Decomposed into two subproblems

Dual-level Partitioning (USP)

Partitioning Approach:

1. Greedy head-level partitioning
2. Biased greedy block-level partitioning (assume workload is perfectly balanced across different heads)

Result: ρs: <1.03 on average

Solution

Sparsity-Aware Parallel Strategy Selection

All-to-All communication latency

Attention computation latency (sparse, multi-GPU)

Ring P2P communication latency

$$\mathcal{L}(\mathbf{s},\mathbf{p}) = \mathcal{L}^{\text{all2all}}(x(\mathbf{p})) + \mathcal{L}^{\text{attn}}(\mathbf{s},\mathbf{p}),$$

$$\mathcal{L}^{\text{attn}}(\mathbf{s},\mathbf{p}) = \left[\max\left(\mathcal{L}^{\text{comp}}(\mathbf{s}), \mathcal{L}^{\text{p2p}}(y(\mathbf{p}))\right) \times (y(\mathbf{p})-1) + \mathcal{L}^{\text{comp}}(\mathbf{s})\right] \times \rho_s(\mathbf{s},\mathbf{p}),$$

$$\mathcal{L}^{\text{comp}}(\mathbf{s}) = \left(\frac{\mathcal{L}^{\text{dense}}}{|\mathcal{G}|} \times \text{density}(\mathbf{s})\right) + \mathcal{L}^{\text{launch}},$$

Kernel launch overhead

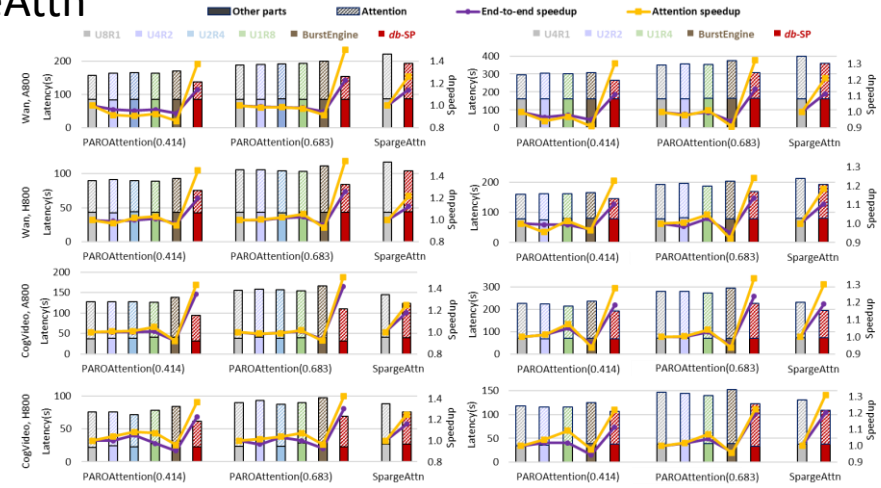
Attention computation latency (dense, single-GPU)

Dynamically select the sequence parallel strategy with least predicted latency at runtime.

Main result

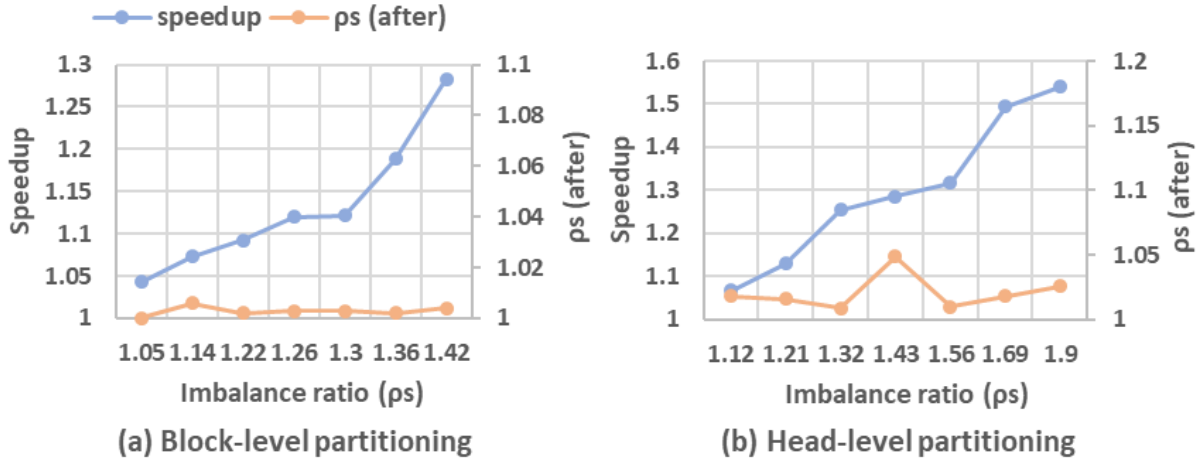
Results1: Higher Speed in Real System

- **Baseline:** Ulysses, Ring Attention, and USP
- **Testbed:** 4/8 NVIDIA A800 80GB SXM4 GPUs & 4/8 NVIDIA H800 80GB GPUs
- **Benchmark**: Wan2.1-T2V-14B (81-frame 1280P videos with 50 sampling steps) & CogVideoX1.5-5B (161-frame 1280P with 50 sampling steps); PAROAttention & SpargeAttn



1.4× attention speedup & 1.25× end-to-end speedup on average

Other Results



Results2: near perfect workload

The imbalance ratio of each layers are close to 1 after dual-level partitioning

Results3: near linear speedup

Speedup and imbalance ratio is approximately proportional.

Takeaway

The goal of this work:

Higher speed, balanced workload

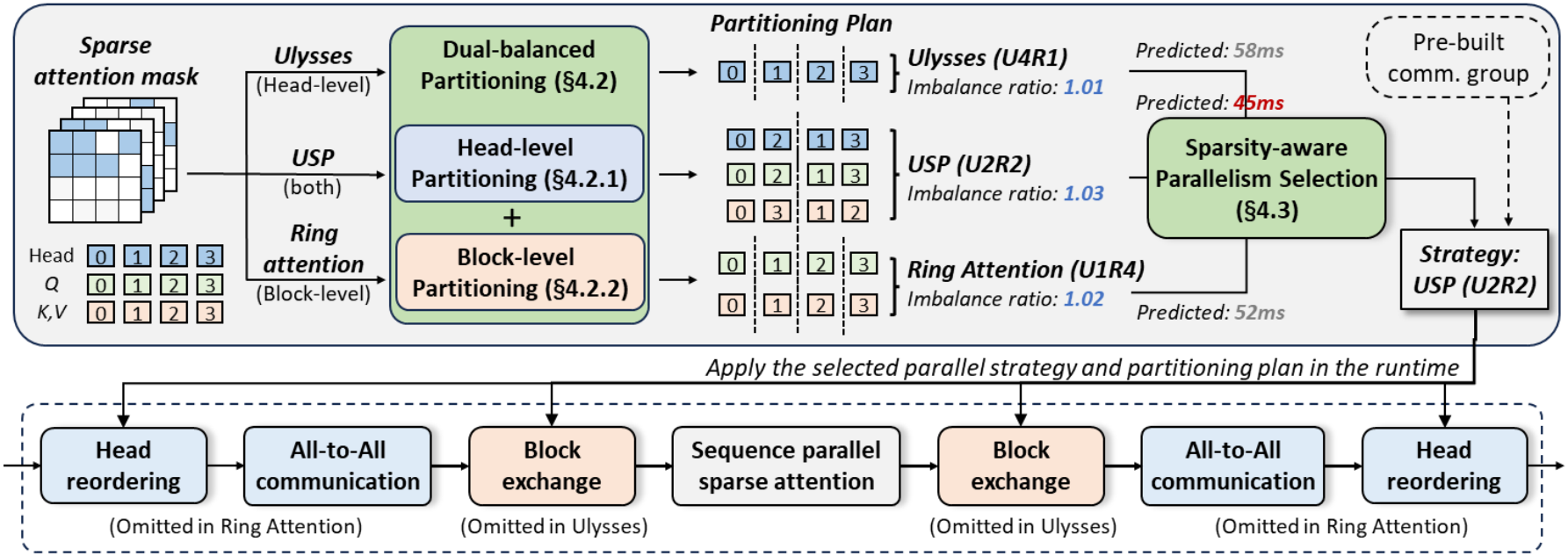
The motivation of this work:

To address the problem of workload imbalance when applying sequence parallelism to sparse attention

What we do in this work:

1. ***Quantify the problem by defining a sparse imbalance ratio***
2. ***Dual-level partitioning with reduced overhead***
3. ***Dynamic sequence parallelism selection***

db-SP: Accelerating Sparse Attention for Visual Generative Models with Dual-Balanced Sequence Parallelism



Thank you, Q&A

Siqi Chen, chensq23@mails.tsinghua.edu.cn