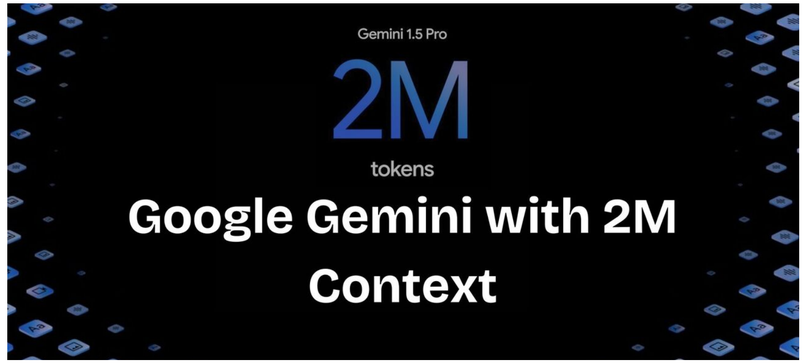
Unleashing Scalable Context Parallelism for Foundation Models Pre-Training via *FCP*

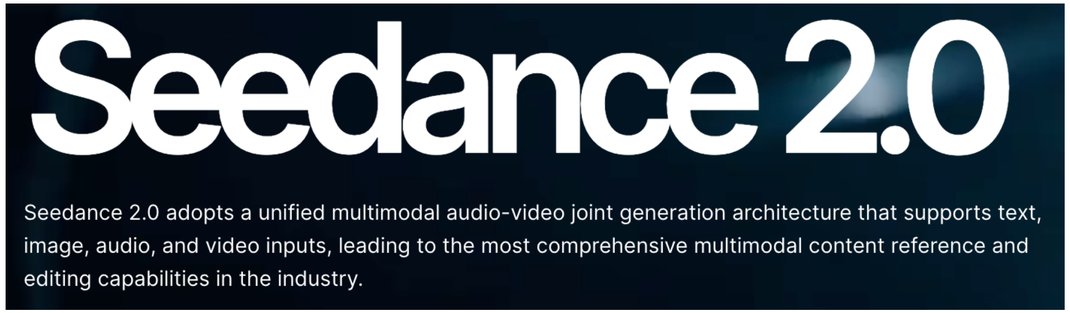
Yilong Zhao*, **Xiaonan Nie***, Kan Zhu, Shuang Ma, Zhichao Lai, Hongxiang Hao, Yang Zhou, Baris Kasikci, Ion Stoica

Background

Foundation models are moving toward longer contexts

- Context windows keep expanding for longer documents and reasoning-intensive tasks.
  - **Multimodal** inputs further amplify sequence lengths beyond text.
- This creates increasing pressure on **attention** computation and memory.

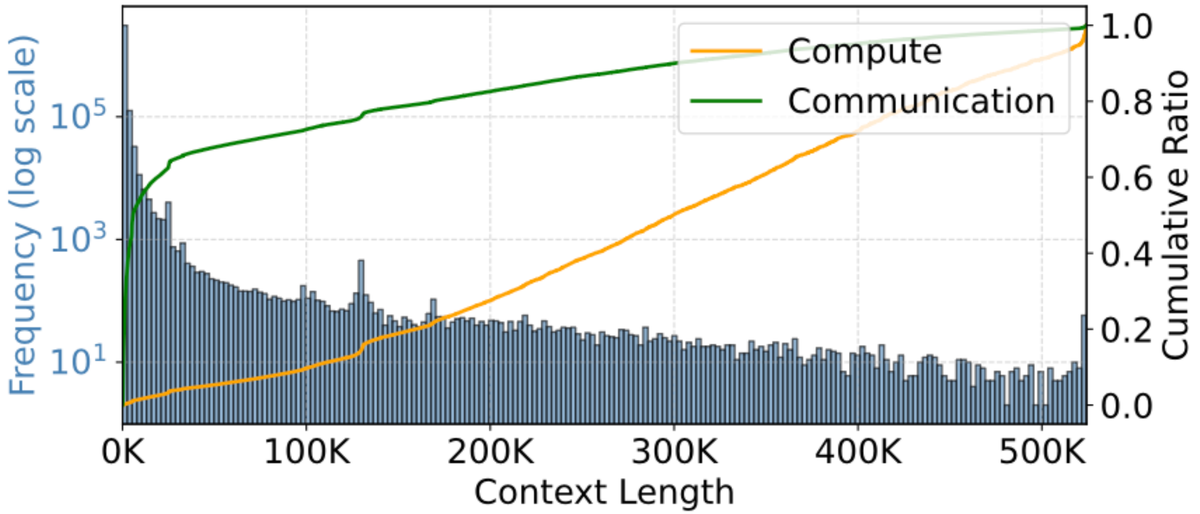




Motivation

Training samples contain heterogeneous context lengths

- Real VLM datasets are **long-tailed:**
  - Both short and long sequences take a significant portion of attention flops.
  - Workload imbalance: equal number of tokens ≠ balanced attention flops.
- **Goal:** efficiently parallelize heterogeneous sequences across GPUs with **linear scalability.**

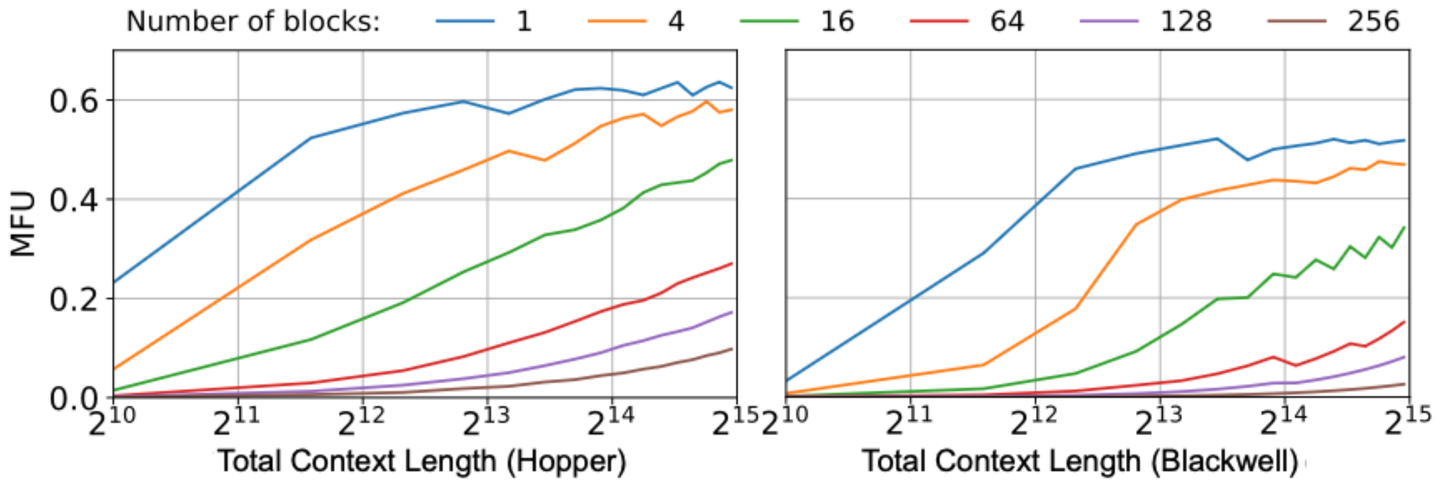


The context length distribution, and cumulative computation and communication ratio (attention-only) from our internal training data.

Challenges

Challenge 1: Over-sharding short sequences wastes compute and communication

- Sharding short sequences into tiny blocks leads to reduced kernel efficiency.
  - Modern GPUs need sufficiently large attention blocks to saturate tensor cores.
  - Additional communication volume for aggregating parallelized states.

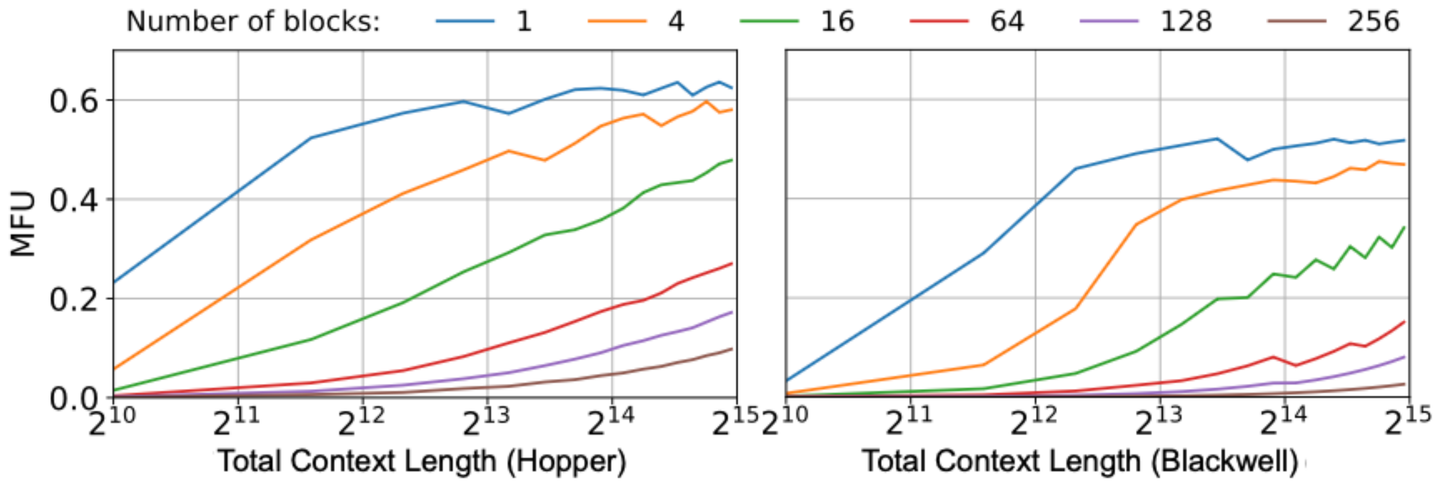


Attention-kernel MFU of FA3/4 with num-kv-heads=8, num-qo-heads=64, head-dim=128 w/ unit benchmark

Challenges

Challenge 1: Over-sharding short sequences wastes compute and communication

- Sharding short sequences into tiny blocks leads to reduced efficiency.
  - Modern GPUs need sufficiently large attention blocks to saturate tensor cores.
  - Additional communication volume for aggregating parallelized states.

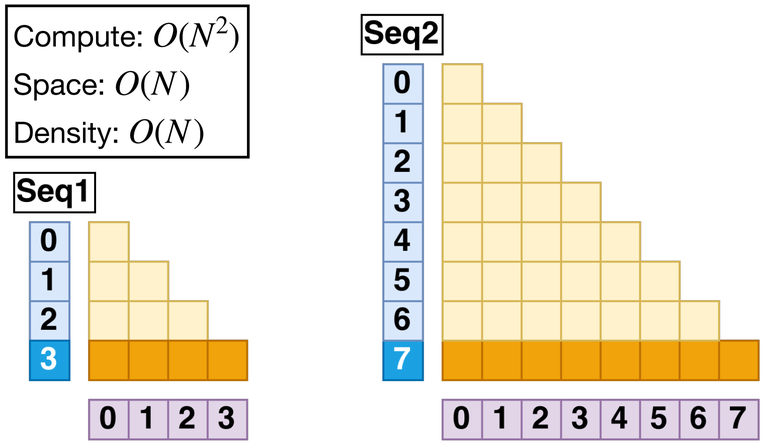


Sequence sharding should be ***length-aware*** for sharding ***efficiency***

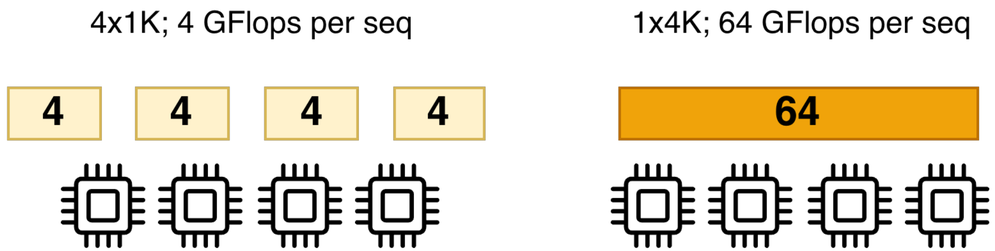
Challenges

Challenge 2: Equal tokens do not imply balanced attention workload

- Attention compute scales **quadratically** with sequence length, while memory scales **linearly**.
- Longer sequences have higher **compute density**: more attention FLOPs per token.
- Token-only bin packing creates severe stragglers.
  - E.g., 4x1K sequences and 1x4K sequences have the same memory footprint, while the latter requires **4× per-GPU compute**.



Complexity of compute, memory and density of attention

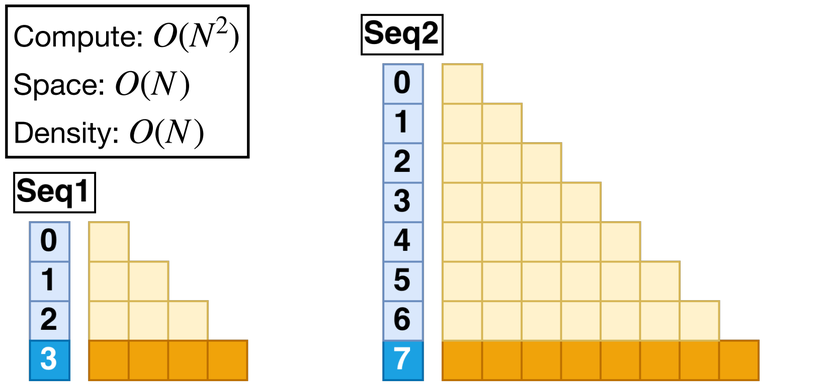


Example of naive bin-packing (assume each GPU hold 1K)

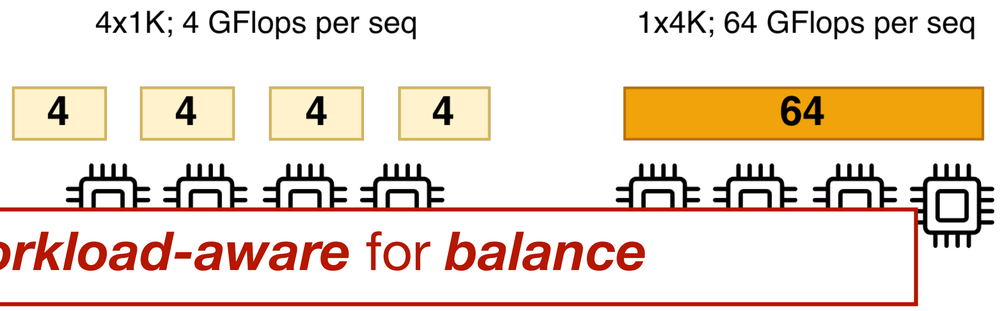
Challenges

Challenge 2: Equal tokens do not imply balanced attention workload

- Attention compute scales **quadratically** with sequence length, while memory scales **linearly**.
- Longer sequences have higher **compute density**: more attention FLOPs per token.
- Token-only bin packing creates severe stragglers.
  - E.g., 4x1K sequences and 1x4K sequences have the same memory footprint, while the latter requires **4× per-GPU compute**.



Complexity of compute, memory and density of attention



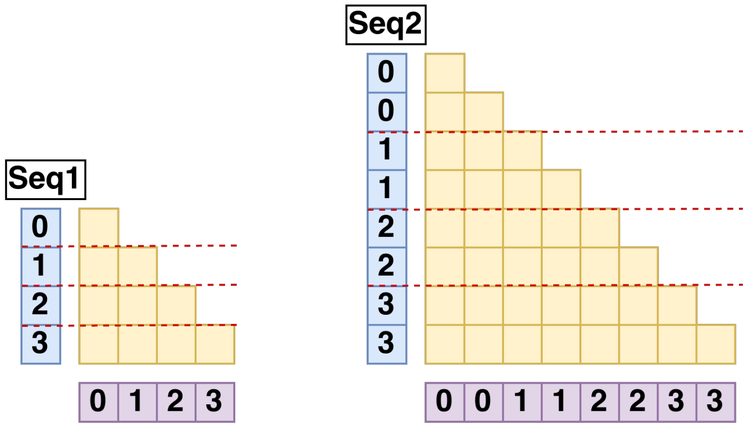
Example of naive bin-packing (assume each GPU hold 1K)

Shards packing should be ***workload-aware*** for ***balance***

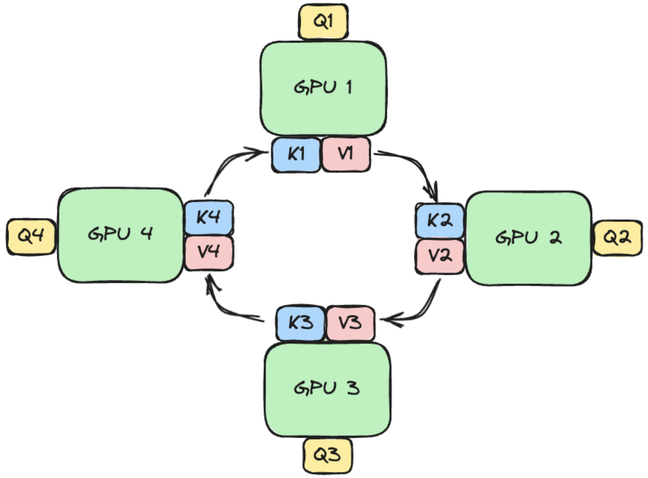
Existing Works

Ring Attention: workload balance ✅, sharding efficiency ❌

- Splits every sequence **uniformly** across all ranks within the CP group.
  - Both short and long sequences are sharded into the same number of blocks.
  - Ring execution provides perfect workload balance, both memory- and computation-wise.
- **Limited scalability**: oversharding short sequences decreases kernel efficiency and introduces excessive communication volume.



Uniform sharding of RingAttention

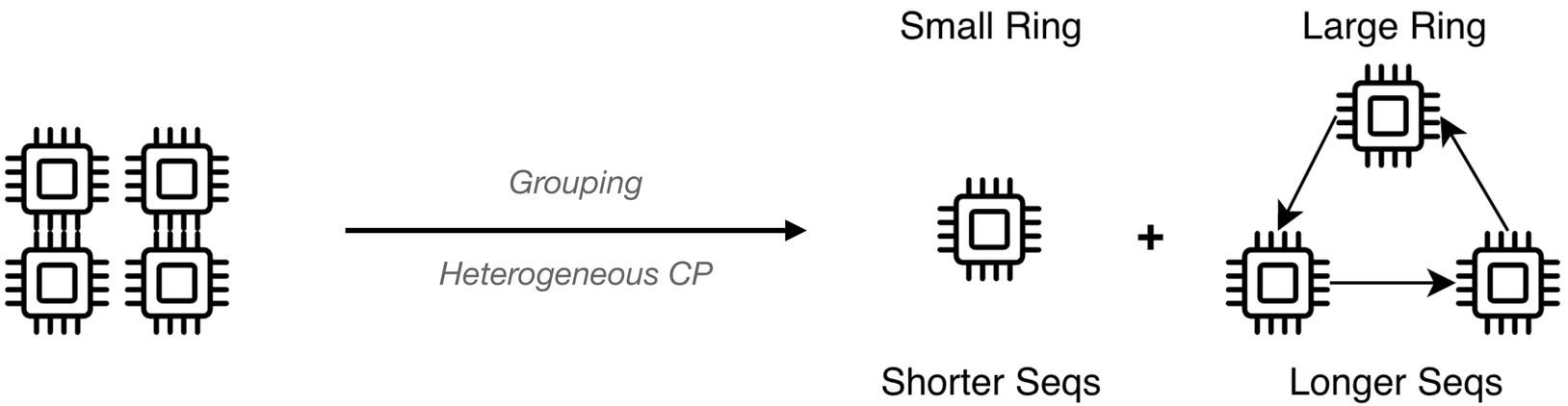


Ring topology execution

Existing Works

ByteScale: sharding efficiency ✅, workload balance ❌

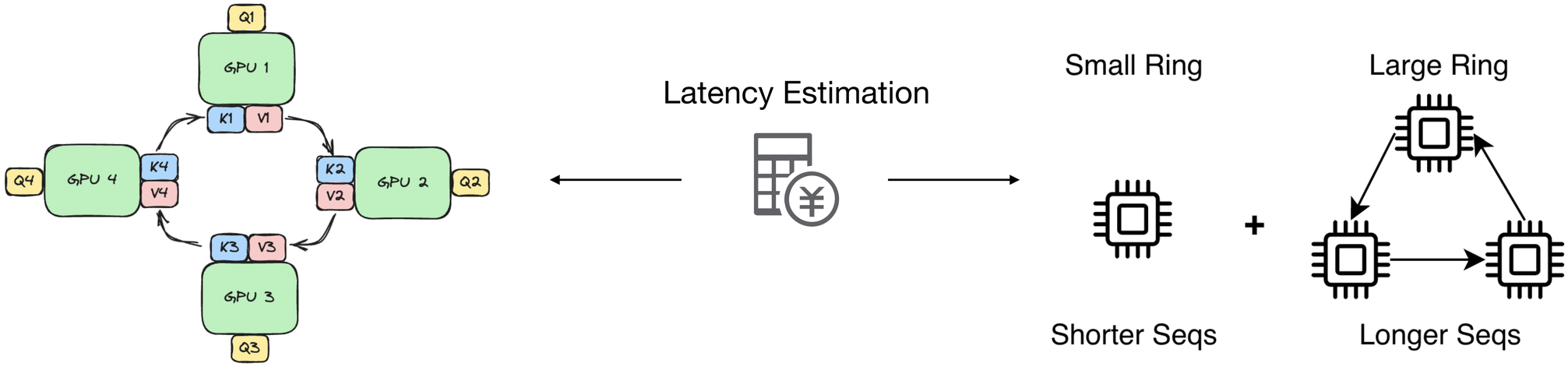
- Uses a **dynamic mesh** to allocate different-sized CP for sequences with different lengths.
- Shorter sequences use a smaller CP, avoiding **over-sharding** and redundant communication.
- However, such grouping leads to inter-group workload imbalances. Given sequence length L:
  - If CP size $\propto L$, then $O(L/L) = O(1)$ memory per rank and $O(L^2/L) = O(L)$ flops per rank
  - If CP size $\propto L^2$, then $O(1/L)$ memory per rank and $O(L^2/L^2) = O(1)$ flops per rank



Existing Works

WLB-LLM: adaptive switch for trade-off

- Uses online latency estimation to choose between two CP strategies.
- **Balance-optimized:** uniform ring-based sharding, perfect balance but worse efficiency.
- **Efficiency-optimized:** length-aware sharding, better efficiency but worse balance.
- However, switching between two sub-optimal strategies does not expand the space.



Balance-optimized (RingAttention)

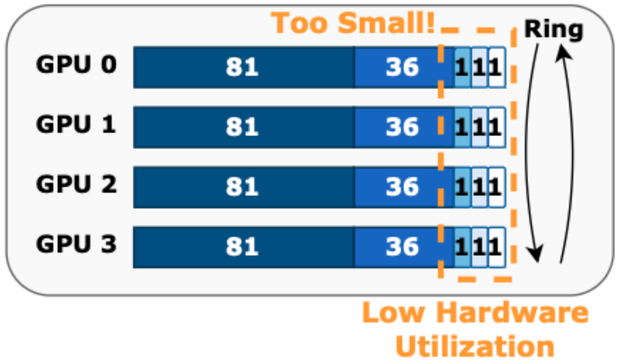
Efficiency-optmized (ByteScale)

Design Goals for Flexible Context Parallelism

To handle heterogeneous sequence lengths, CP needs three properties:

Compute efficiency:

each operation saturates GPU hardware

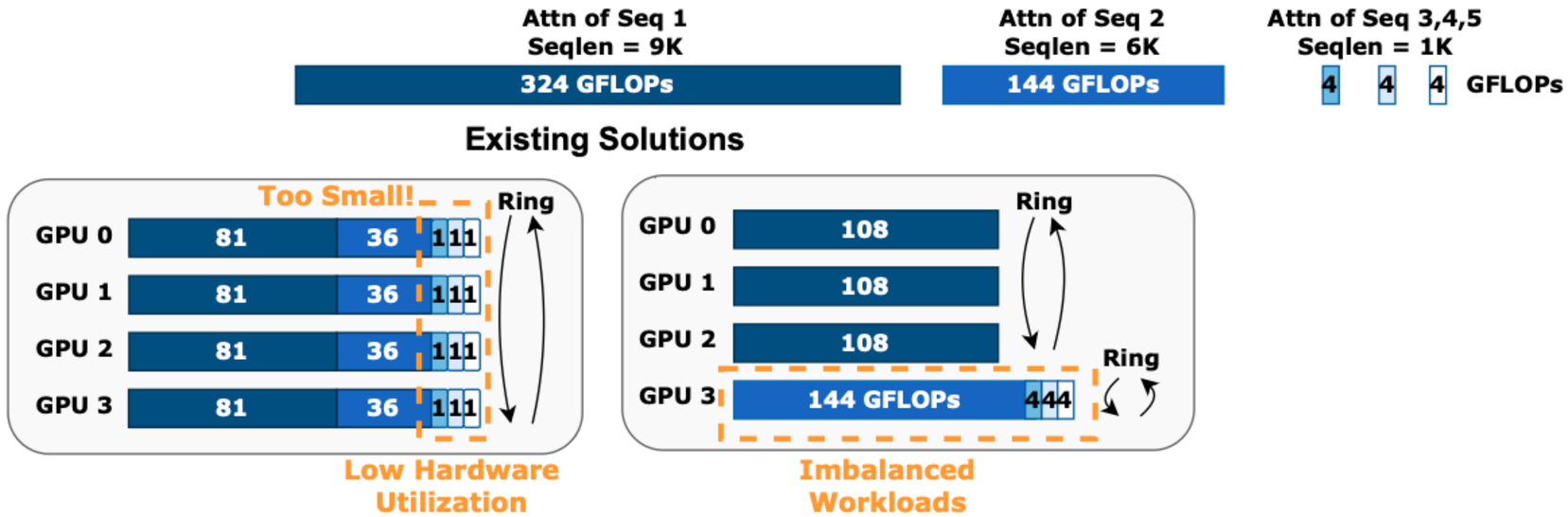


Design Goals for Flexible Context Parallelism

To handle heterogeneous sequence lengths, CP needs three properties:

Compute efficiency:
each operation saturates GPU hardware

Workload balance:
each GPU holds the same amount of workloads



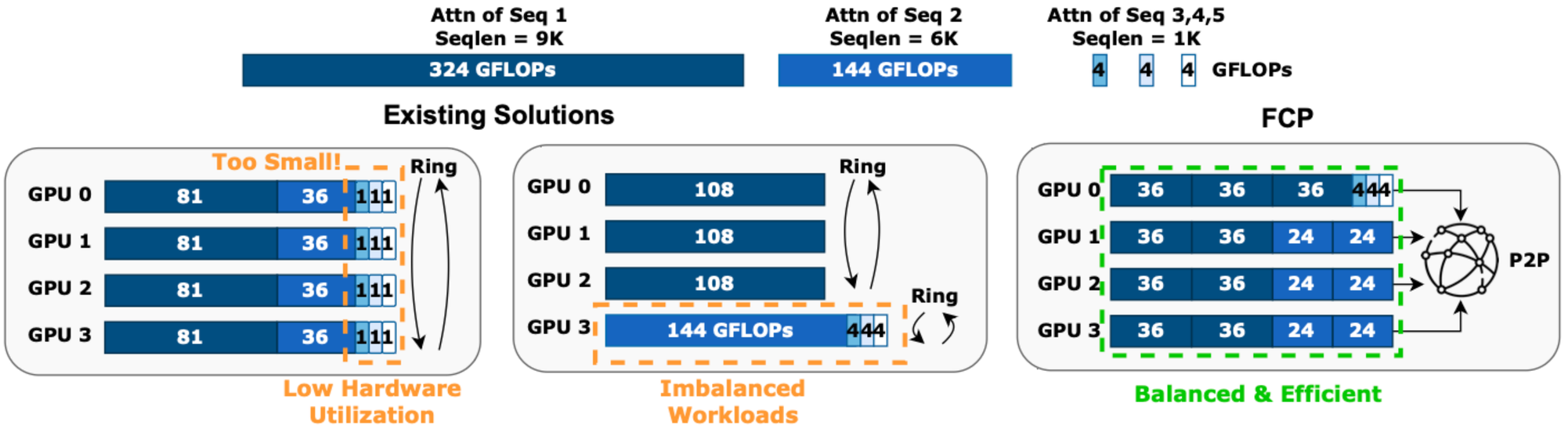
Design Goals for Flexible Context Parallelism

To handle heterogeneous sequence lengths, CP needs three properties:

Compute efficiency:
each operation saturates GPU hardware

Workload balance:
each GPU holds the same amount of workloads

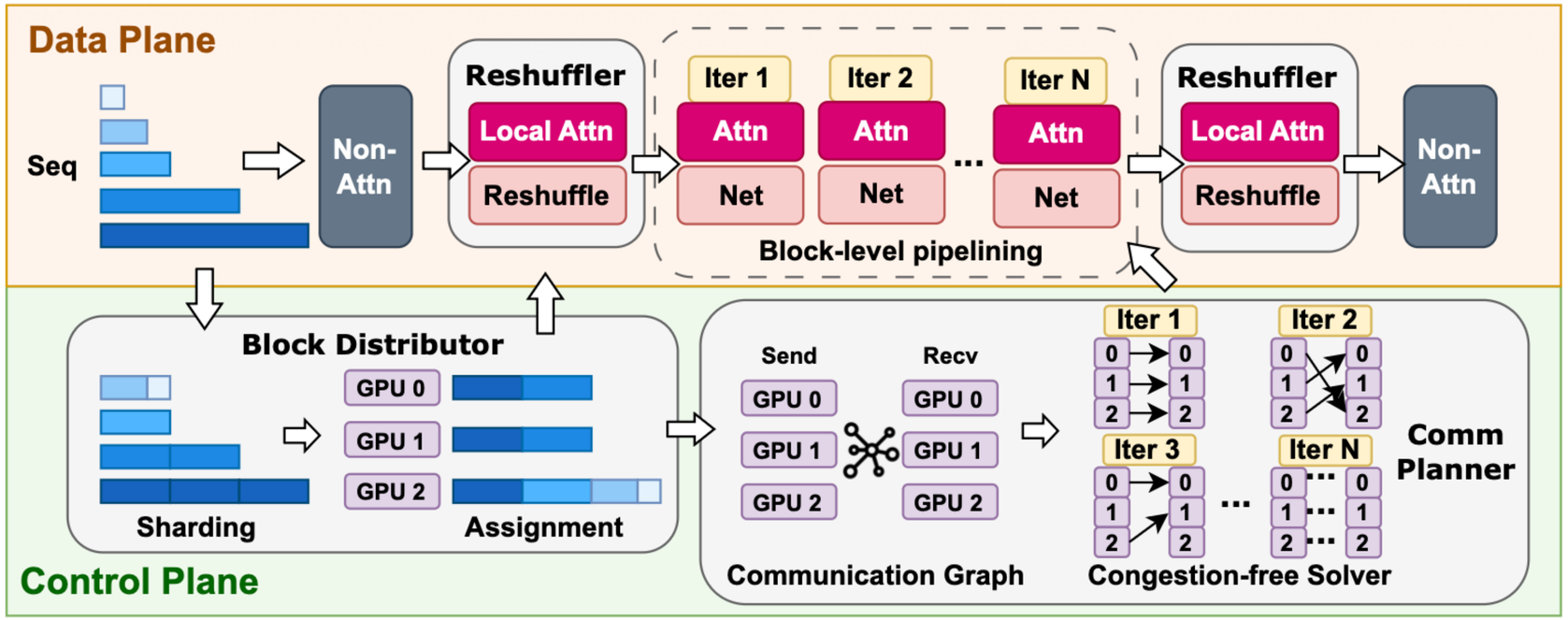
Communication overlap:
communication should be effectively overlapped with computation



System overview

FCP design overview

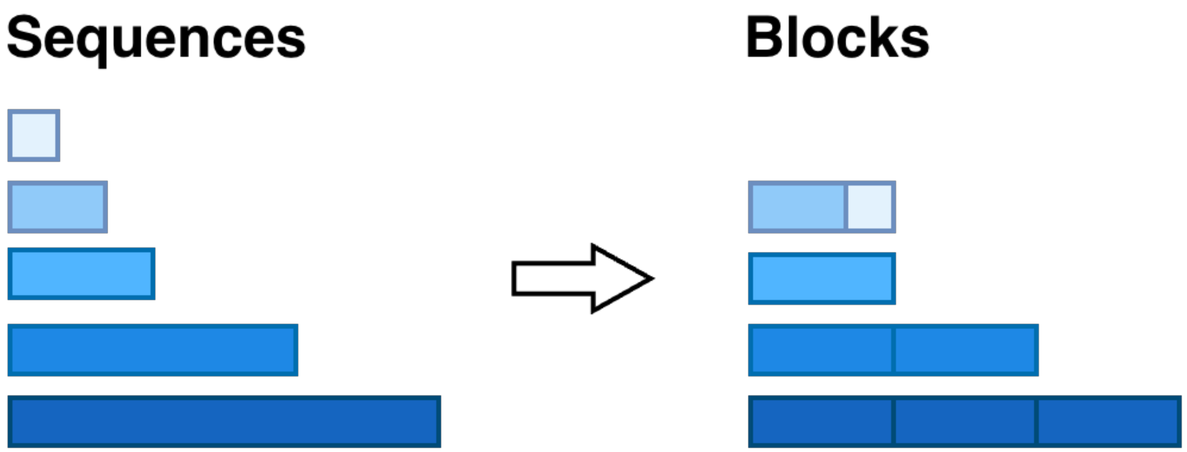
- **Block distributor**: length-aware sharding + workload-aware assignment.
- **Communication planner:** congestion-free P2P comm schedule.
- **Transparent reshuffler:** bridge user layout and FCP layout (*skipped in the talk*).



Overview of FCP Design

Design

Block distributor: fixed-size blocks + workload-aware assignment



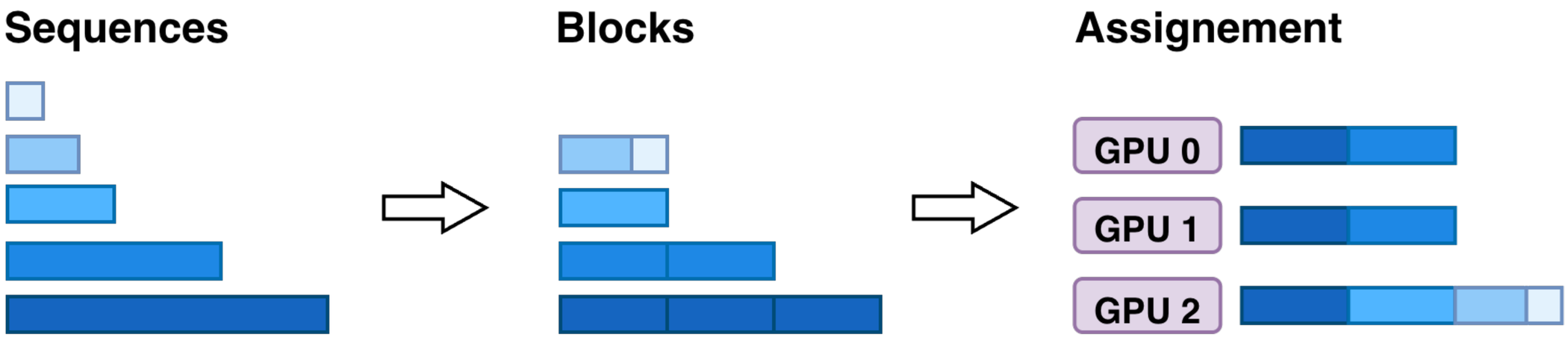
Sharding sequences into fixed-sized ***blocks***

Adaptive to various lengths;

Large enough to saturate GPU

Design

Block distributor: fixed-size blocks + workload-aware assignment



Sharding sequences into fixed-sized ***blocks***

Adaptive to various lengths;
Large enough to saturate GPU

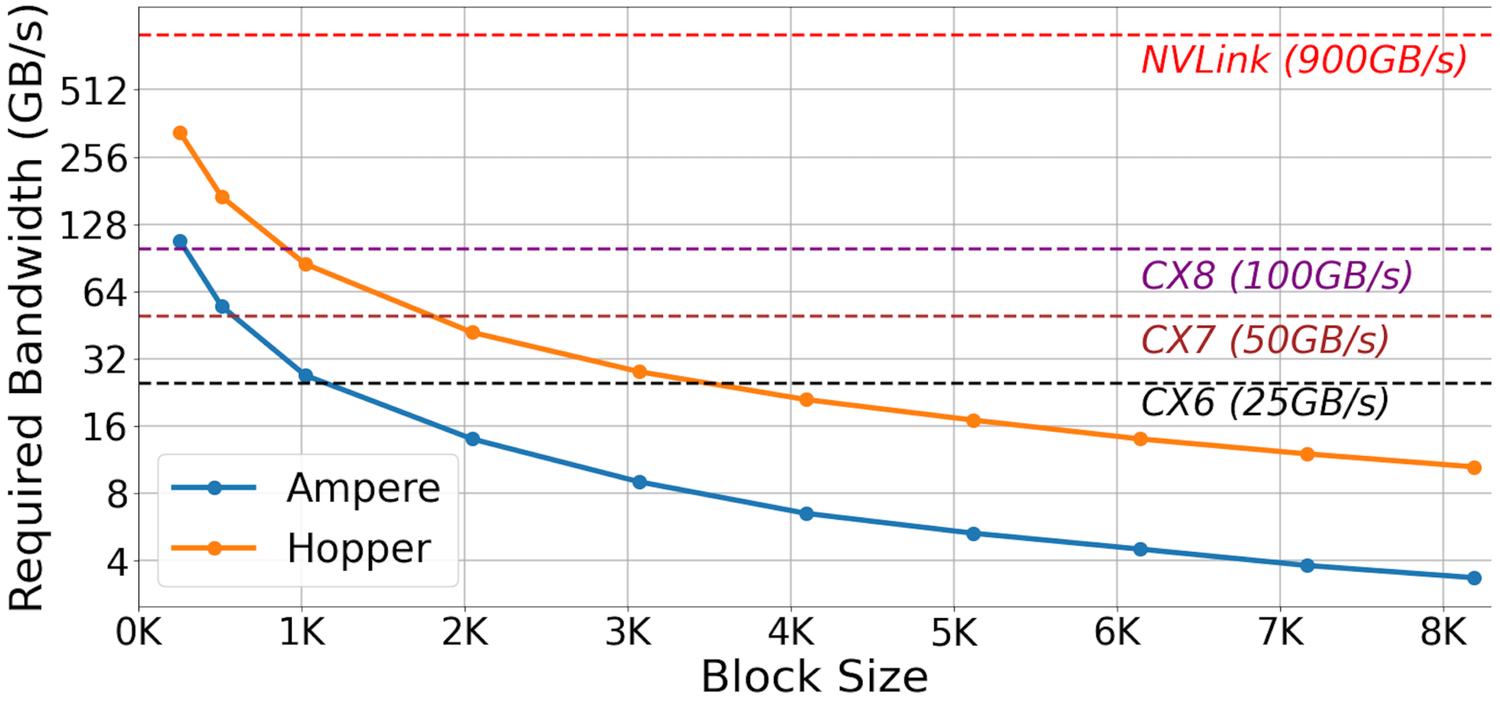
Greedy-based GPU assignment (LPT algorithm)

Compute/memory/communication balance

Design

How to overlap communication efficiently?

- FCP is a trade-off between **communication complexity** and **schedule flexibility.**
- **Block-wise pipelining** for overlapping!



Required comm bandwidth for overlapping the computation and communcation of a single sharded block (assume 100% utils)

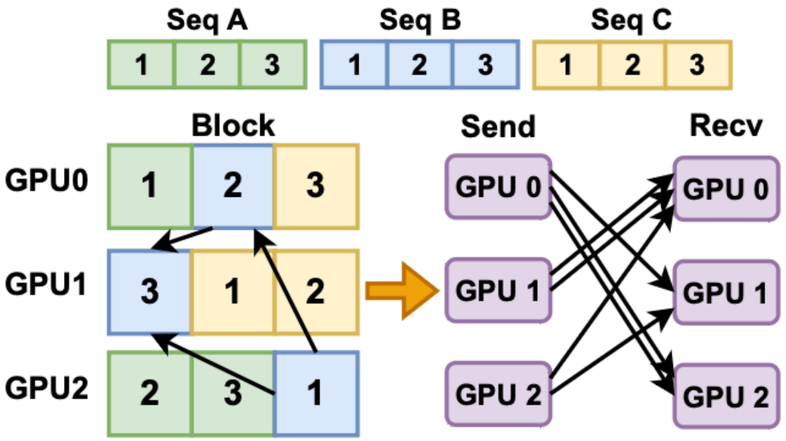
$O(N)$ communication v.s. $O(N^2)$ computation

Always tunable for overlap

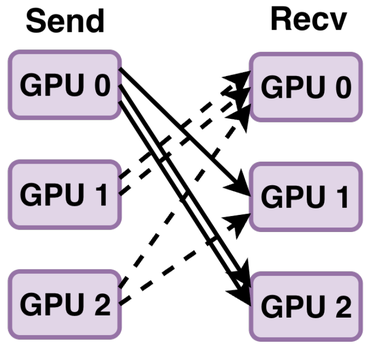
Design

How to overlap communication efficiently?

- Communication ordering matters!



Communication bipartite
constructed by block assignments

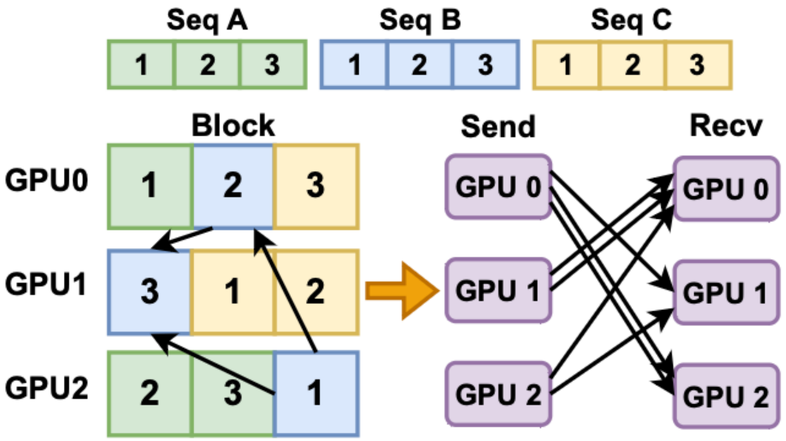


Congestion at GPU0 ❌

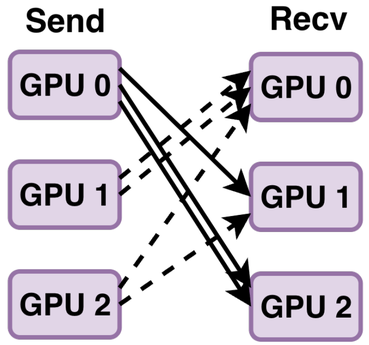
Design

How to overlap communication efficiently?

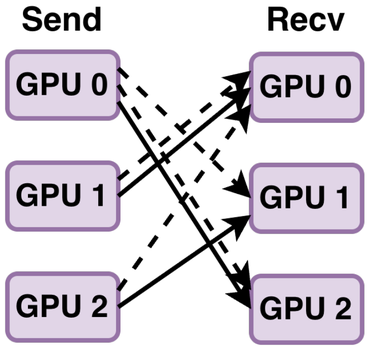
- Communication ordering matters!



Communication bipartite
constructed by block assignments



Congestion at GPU0 ❌



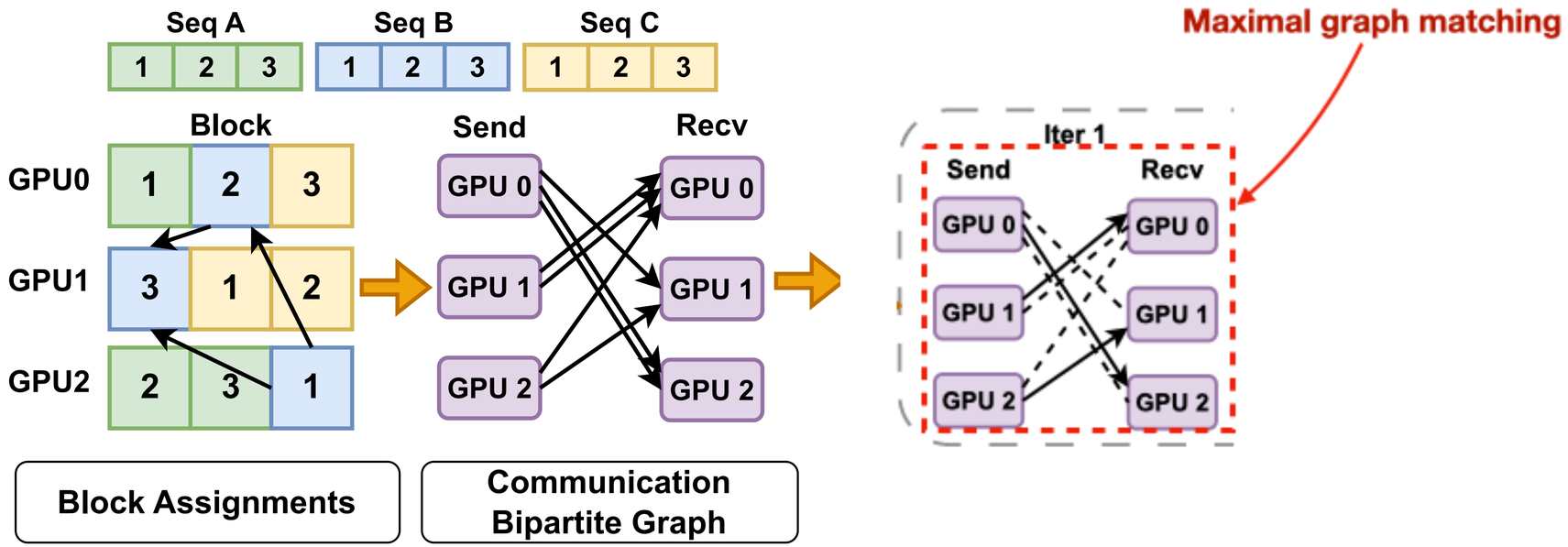
Congestion-free ✅

Congestion-free plan ✅***:***
Each GPU at most sends/receives one block, which is equivalent to a ***graph matching***

Design

How to overlap communication efficiently?

- Congestion-free ordering solver:



- Iteratively compute the ***maximal graph matching*** from the bipartite graph
  - ***Proved:*** the optimal number of iterations equals to the maximal degree
  - Hopcroft-Karp algorithm $O(N^{2.5})$

Evaluation Setup

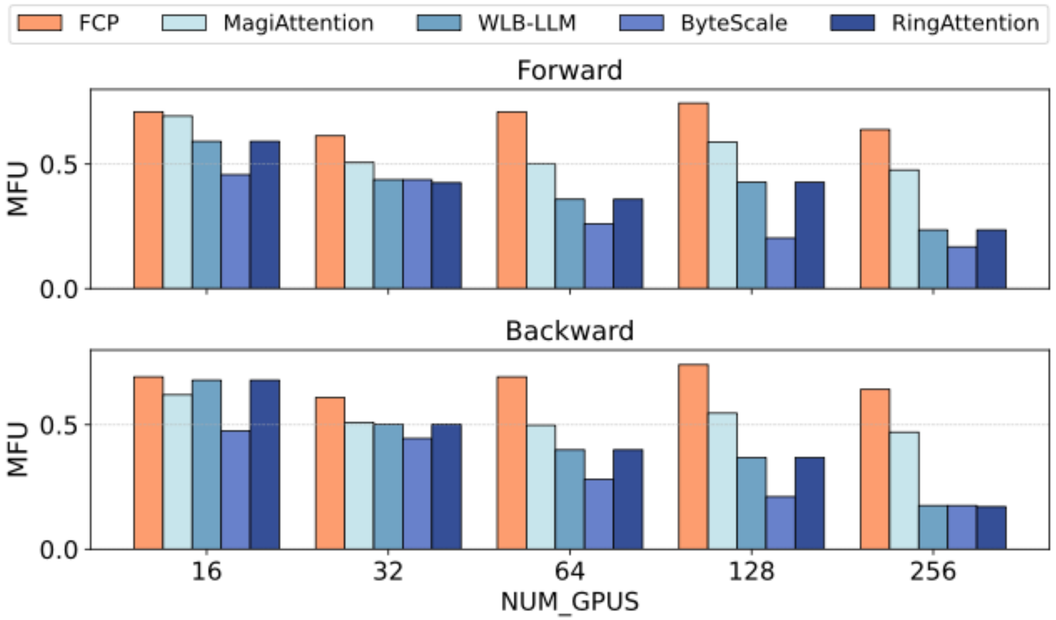
Real long-tailed workloads on modern GPU clusters

- **Hardware:** two clusters, anonymized as GPU-X and GPU-Y, with different computation-to-communication ratios
  - BF16 tensor ops and CP domain comm bandwidth
  - (Flops/s)/(byte/s): GPU-X (5920) v.s. GPU-Y (2500)
- **Workloads:**
  - Configuration from Llama-3-70B on real training traces, up to 512K context length.
  - Causal attention, scaling up to 256 GPUs.
- **Baselines:** RingAttention, ByteScale, WLB-LLM, MagiAttention
  - Implemented with FA3/4
  - Comm/Compute overlap enabled
- **Hyperparameters:** 4K tokens as block size, 32K tokens as local batch size

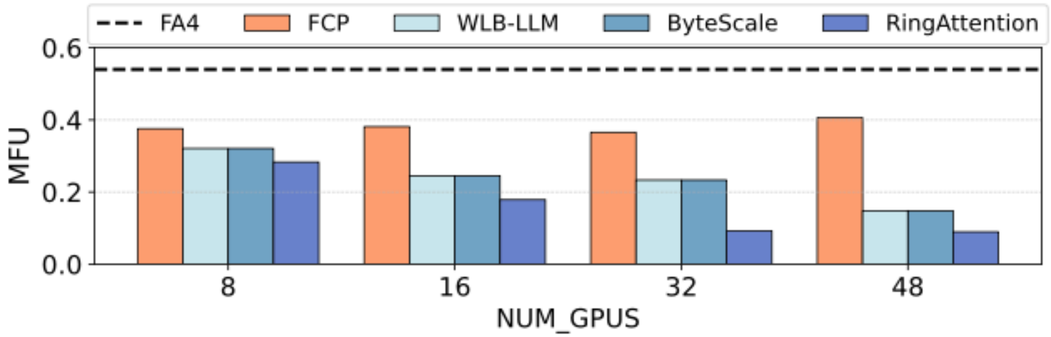
Evaluation

FCP consistently outperforms existing CP methods across scales

- Weak scaling: increasing the total number of GPUs, while keeping per-GPU tokens the same.
- FCP maintains high MFU as the CP degree increases, while baselines degrade due to over-sharding or imbalance.



Weak Scaling on GPU-X.

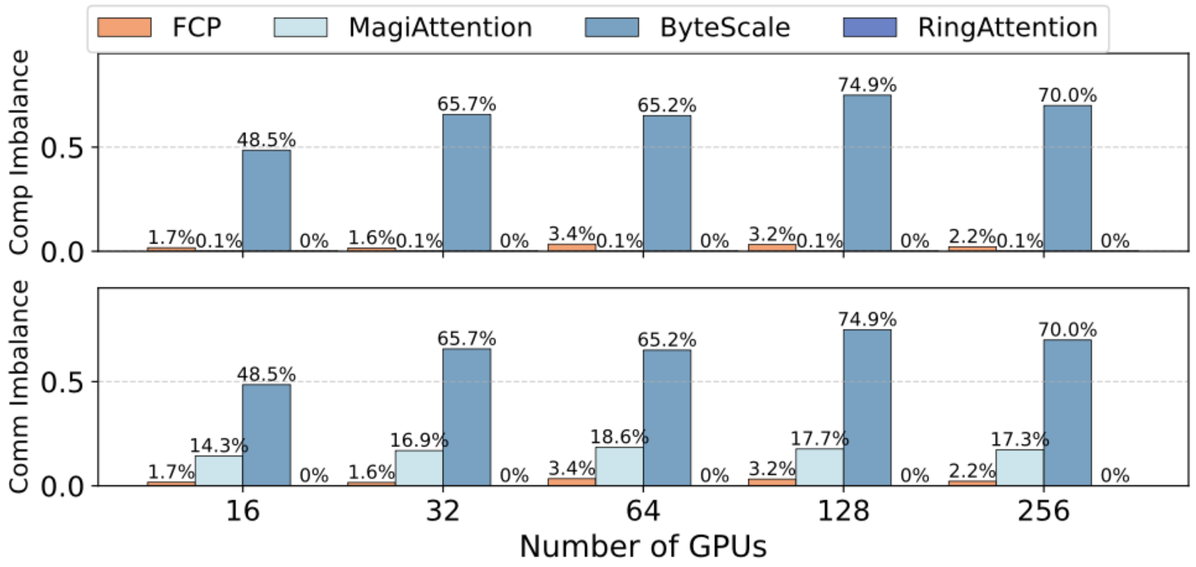


Weak Scaling on GPU-Y (Forward-only)

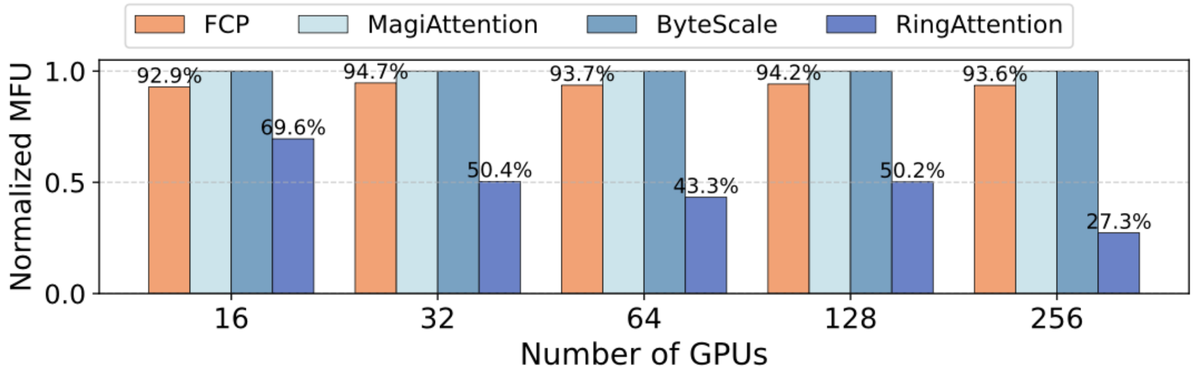
Evaluation

FCP balances compute, communication, and kernel efficiency

- **RingAttention:** balanced, but over-shards short sequences
- **ByteScale:** efficient compute, but imbalanced workloads
- **MagiAttention:** good compute balance, but communication imbalance remains
- **FCP:** balanced workloads + efficient kernels + balanced communication



Compute/Communication Imbalance Ratio

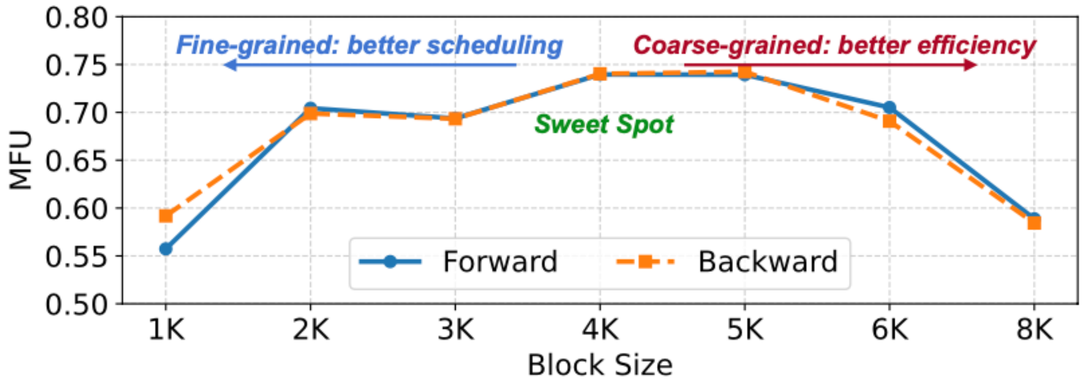


Compute Efficiency of Single Kernel Call (Normalized to FA perf)

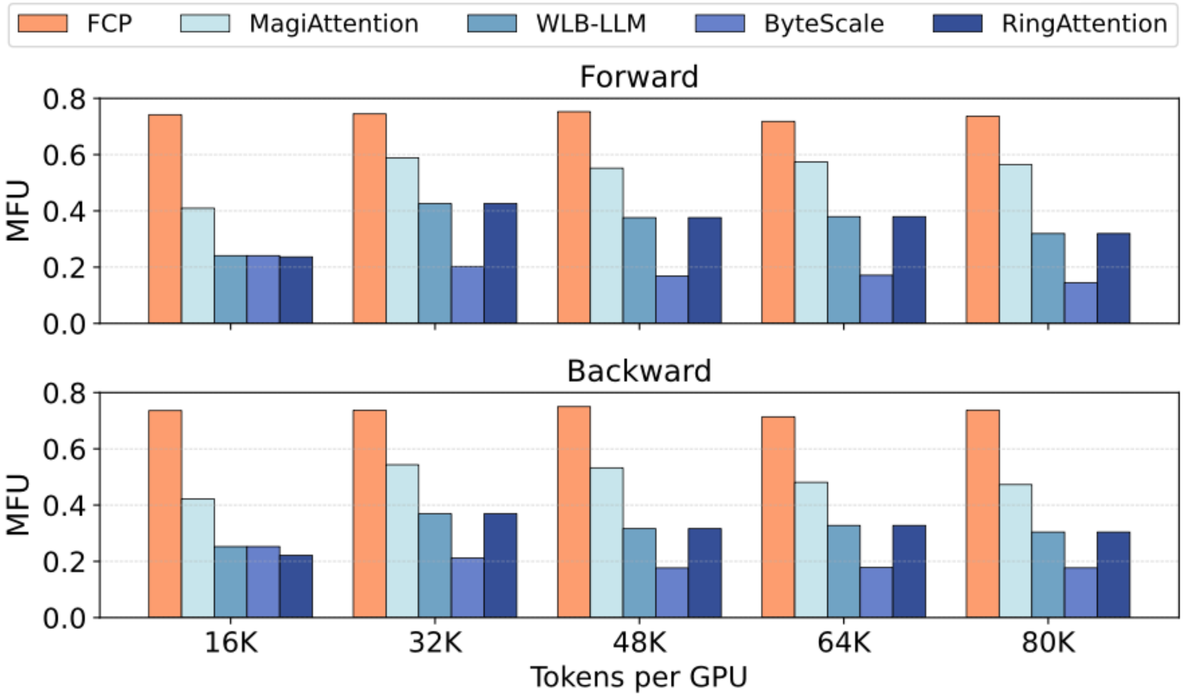
Evaluation

FCP is robust across block sizes and token budgets

- **Block size sensitivity (Left):** FCP is robust to block-size choices; precise tuning brings only marginal gains.
- **Token-budget sensitivity (Right):** FCP consistently outperforms baselines from 16K to 80K tokens/GPU.



Sensitivity test of block sizes on 128x GPU-X



Sensitivity test of per-GPU tokens on 128 x GPU-X

Conclusion

Flexible block-level scheduling makes CP scalable

- Existing CP methods trade off **compute efficiency** and **workload balance**.
- FCP removes this trade-off with **block-wise scheduling + arbitrary P2P communication**.
- FCP scales to **256 GPUs** and improves attention MFU by **1.13×–2.21×**.

Q&A

niexiaonan@bytedance.com

yilongzhao@berkeley.edu

Thanks!

Paper