Context Parallelism for Scalable Million-Token Inference

Amy (Jie) Yang, Jingyi Yang, Aya Ibrahim, Xinfeng Xie,
Bangsheng Tang, Grigory Sizov,
Jeremy Reizenstein, Jongsoo Park, Jianyu Huang

Motivation

- Demand for long context (1M) driven by use cases: e.g. multimodal, code analysis and copilot.
- Processing long context incurs long latency at inference time without optimization.
- Goal: Enable fast, scalable inference without architectural changes.

Key Contributions

- Introduced Context Parallelism (CP) for LLM inference.
- Developed two ring attention variants: pass-KV and pass-Q to support low-latency prefill and decode with long context.
- Achieved 1M-token inference with Llama3 405B in 77s with 93% parallel efficiency.

Long Context Inference Challenges

- Compute:
  - Dense attention FLOPs scale quadratically with context length.
  - Attention compute dominates
- Memory:
  - KV cache grows linearly with context.
- Communication:
  - communication latency increases when parallelized to multiple hosts.

	Prefill	Decode
Attention	O(T ^ 2 * D)	O(B * D * T)
FFN	O(T * D * hidden)	O(B * D * hidden)

Attention compute complexity

context	TTFT	TTIT
128K	42s	~100ms
8K	1.6s	<= 30ms

Single host 405B FP8 model serving latency

Parallelism Comparison

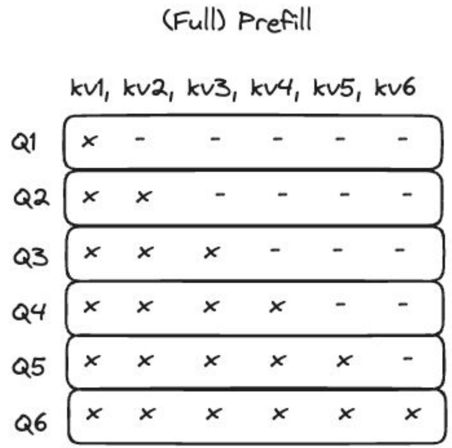
- TP: High all-reduce overhead, poor inter-node scaling.
- CP: Distributes input tokens, passes only Q or KV tensors.
- CP has significantly lower communication cost

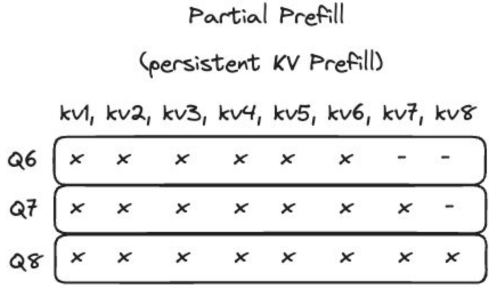
Table 2. Communication and memory cost comparison between tensor parallel (TP) and context parallel (CP) for full prefill. T: sequence length, D_H: head dimension, N_H: # of attention heads, N_{KV}: # of key/value heads, N_{TP}: TP group size, W: model parameter size. Total comm cost shows the communication cost per transformer block.

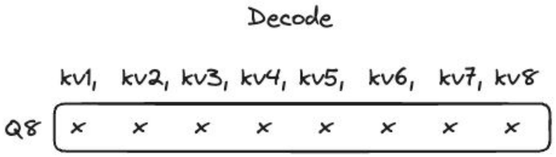
	TP	CP
Collective	AllReduce	SendRecv
Comm per 2 Linear	$T \cdot N_H \cdot D_H$	0
Comm per Attn	0	$T \cdot N_{KV} \cdot D_H$
Total comm	$2 \cdot (T \cdot N_H \cdot D_H)$	$T \cdot N_{KV} \cdot D_H$
Parameter Size	$\frac{W}{N_{TP}}$	W

Multi-Turn Chat:
Full Prefill, Partial Prefill, Decode

- Full Prefill:
  - prompt processing and KV cache generation.
- Decode:
  - autoregressive generation. Generate one token which attends to all previous tokens.
- Partial Prefill:
  - follow-up prompt that attends to previous prompts and generated tokens.







Load-Balanced Sharding

- load-balanced sharding with inputs of variable sequence lengths
- both compute and memory are balanced

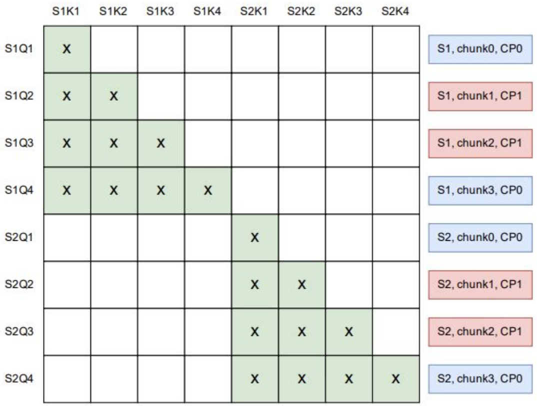


Figure 1. Load-balanced CP sharding with fused inputs in full prefill with 2 CP ranks (CP2). We have 2 input sequences: $S1$, $S2$. Each is partitioned evenly into 4 chunks: Q_i / K_i, where $i = 1, 2, 3, 4$.

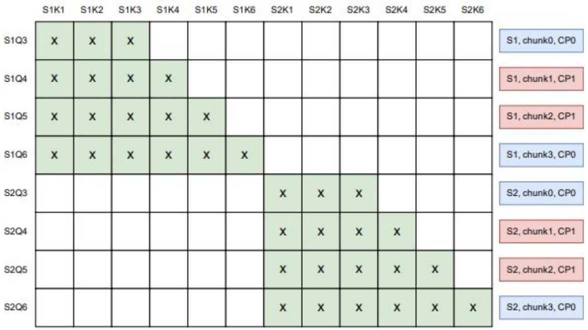


Figure 2. Load-balanced CP sharding with fused inputs partial prefill with 2 CP ranks (CP2). We have 2 input sequences: $S1$, $S2$. Load-balanced sharding is applied to the new token Q_i dimension (4 chunks), regardless of how cached token dimension K_i is partitioned in partial prefill.

Ring Attention Algorithms

- Pass-KV ring attention
  - full prefill and partial prefill with low-medium KV cache hit rate
  - communication and computation are overlapped

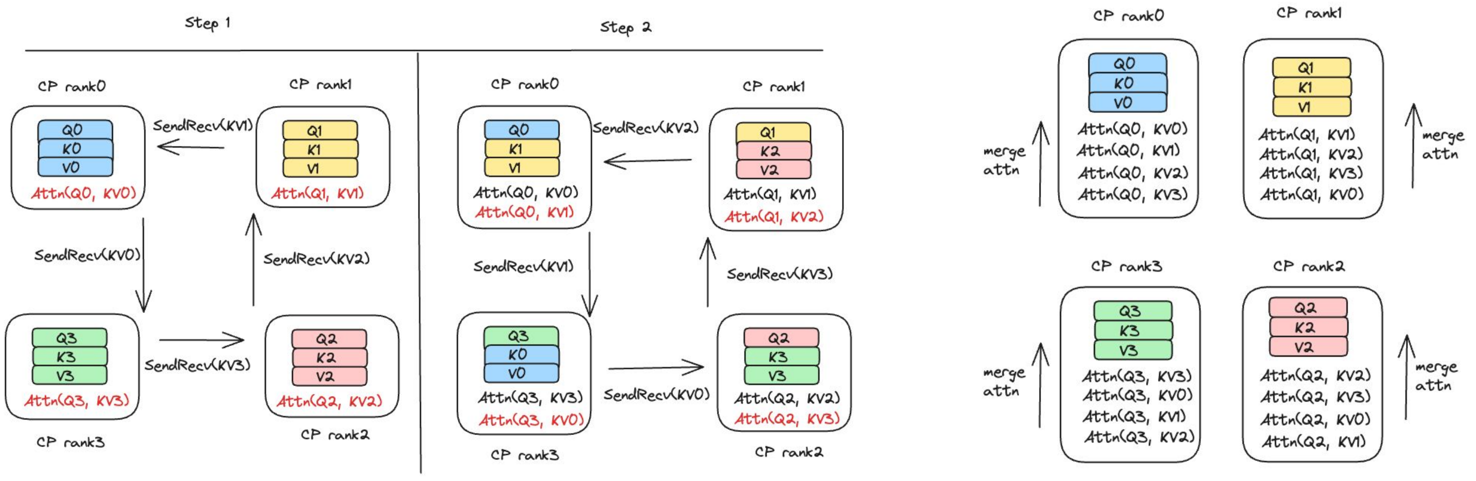
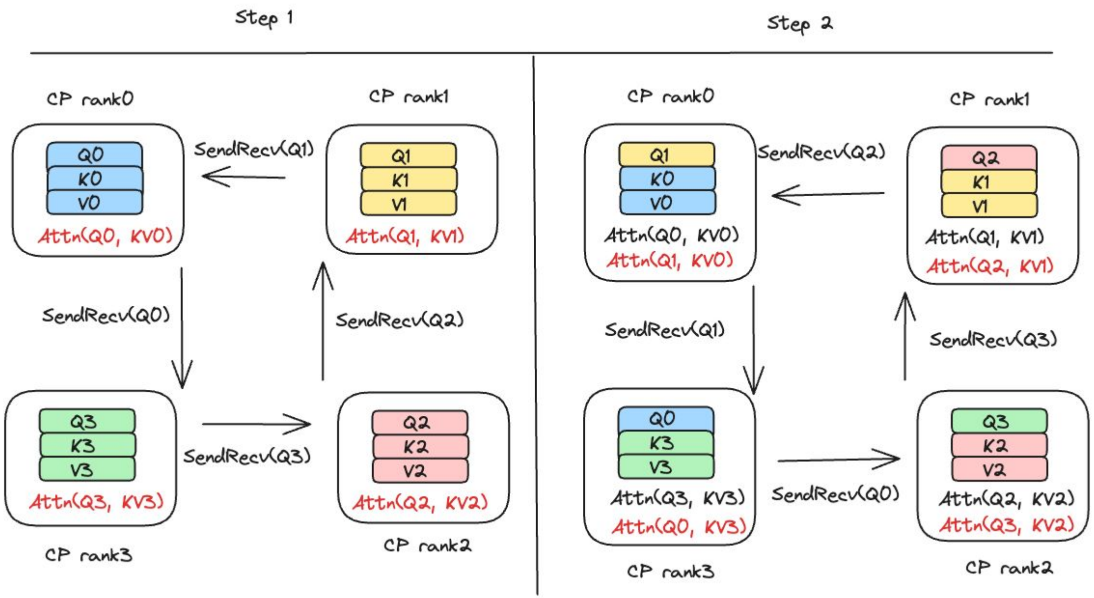


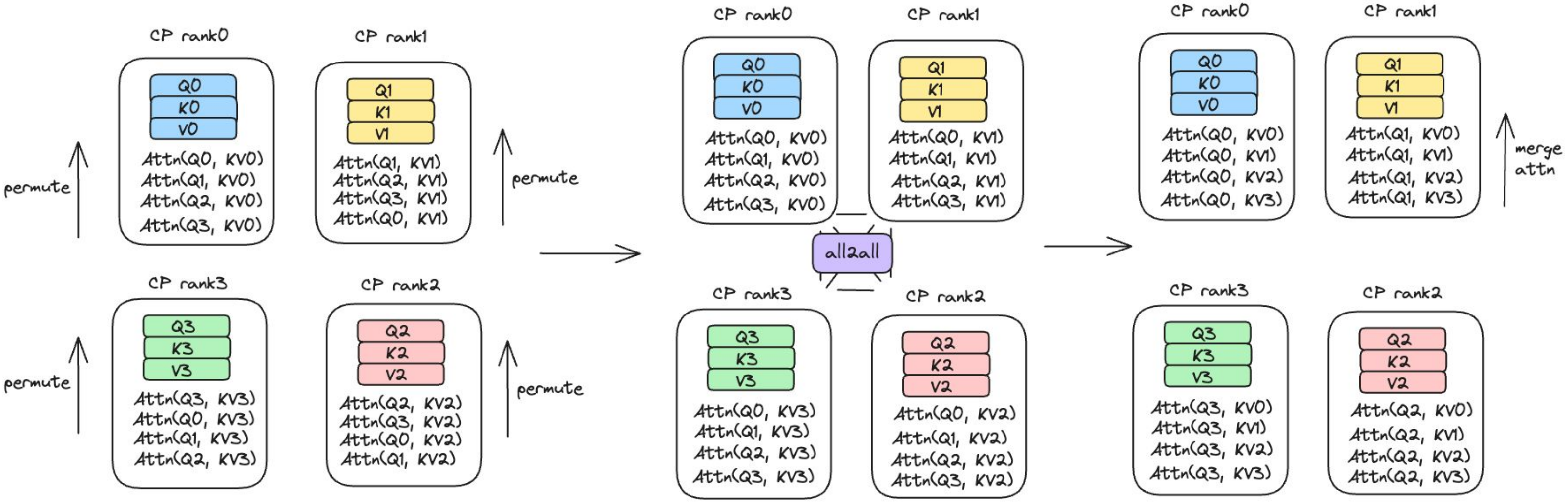
Figure 3. Ring Pass-KV Attention with 4 CP ranks (CP4).

Ring Attention Algorithms



- Pass-Q ring attention:
  - partial prefill with high KV cache hit rate
  - decode

Ring Attention Algorithms



- Pass-Q ring attention
  - permute and all2all partial attention outputs

Experiment Setup

- Model:
  - Llama3 405B, FP8 quantized
  - 8 KV heads, 128 Q heads
- Hardware:
  - Grand Teton Training (GTT)
    - Interconnect: backend RDMA 400Gb/s per GPU
  - Grand Teton Inference (GTI)
    - Interconnect: frontend TCP/IP 100Gb/s per GPU
- Partitioning:
  - 8-way tensor-parallel (TP8) intra host

Results: Prefill Scaling

- Near-linear scaling with CP
- 128K token prefill in 3.8s with CP over 16 nodes.
- 1M-token prefill in 77s.

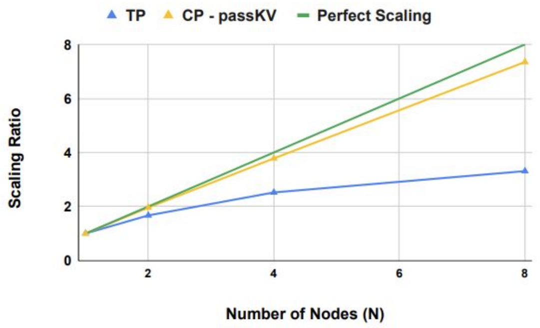
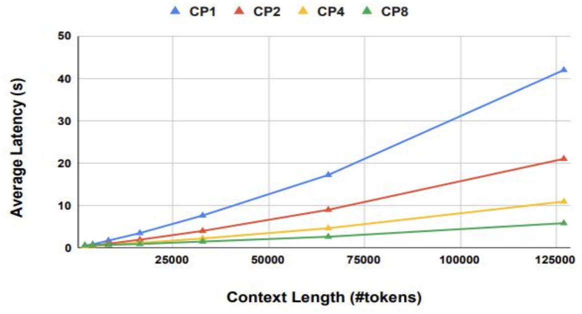
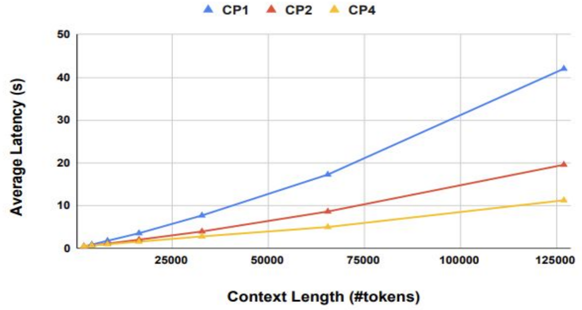


Figure 7. Scaling ratio (latency with one node over latency with N nodes) of context parallel vs. multi-node tensor parallel.



(a) GTT Latency for CP with 1, 2, 4, 8 nodes.



(b) GTI Latency for CP with 1, 2, 4 nodes

Figure 6. Llama3 405B `pass-KV` full prefill latency.

Results: Pass KV vs. Pass Q Prefill

- KV Cache miss rate <= 5%
  - Pass-Q has lower latency

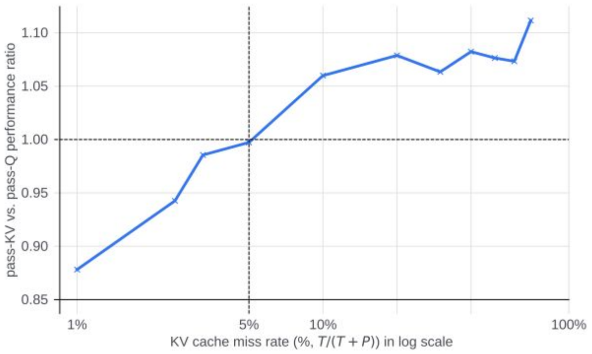


Figure 9. `pass-KV` / `pass-Q` speed ratio of 128K context with persistent KV cache miss rate, varying P and T with $P+T=128000$, on 4 CP ranks (CP4).

Table 4. TTFT (in ms) for `pass-KV` vs. `pass-Q` varying P and T with $P+T=128000$, on 4 CP ranks (CP4). P: length of existing tokens in the KV cache, T: length of new tokens.

P	T	MISS RATE	pass-KV	pass-Q
126720	1280	1.00%	1023.39	898.71
124800	3200	2.50%	1110.18	1046.43
123840	**4160**	**3.25%**	**1298.92**	**1280.1**
121600	**6400**	**5.00%**	**1305.56**	**1302.01**
115200	12800	10.00%	2080.67	2205.27
102400	25600	20.00%	3353.02	3617.02
89600	38400	30.00%	4629.23	4922.52
76800	51200	40.00%	5745.08	6217.83
64000	64000	50.00%	6845.21	7367.99
51200	76800	60.00%	7890.35	8468.66
38400	89600	70.00%	8697.27	9666.62
25600	102400	80.00%	10105.78	10652.39
12800	115200	90.00%	11136.4	11571.62
0	128000	100.00%	11462.15	12360.57

Results: Decode

- CP improves prefill, regresses decode latency.
- TTIT increases with CP ranks.
- Recommend decoupling prefill and decode systems.

Table 7. TTFT / TTIT (in ms) comparisons between TP8, CP2, TP16, CP4, TP32 with 128K context length at batch size 1.

	TTFT	TTIT
CP1+TP8	42010	46.26
CP2+TP8	21042	60.23
TP16	29917	39.52
CP4+TP8	10950	71.31
TP32	19841	47.3

Conclusion

- CP improves prefill latency for long-context inference by scaling compute to more nodes and overlapping communication with computation.
- Dynamic pass-KV/pass-Q switching optimizes latency for full prefill, partial prefill, and decode.
- Combines well with retrieval-based approximate methods.