Heterogeneity-Aware Context Parallelism in Ring Attention

William Baisi, Yujun Qian, Kanghyuk Lee, Anshul Sadh-Gauri
Computer Science Department, Columbia University, New York, USA, {wb2426, yq2432, as7798, kl3768}@columbia.edu

***Abstract*—The rapid adoption of large language models (LLMs) has made inference as a service a dominant deployment paradigm, yet the quadratic memory growth of self-attention remains a major barrier to scalability. Techniques such as FlashAttention and distributed methods like Ring Attention alleviate memory pressure by partitioning computation and data, but they typically assume homogeneous hardware. In this work, we study intra-node heterogeneity, where different ranks have unequal performance, either due to partial hardware upgrades driven by cost constraints or because one or more ranks are degraded or throttled. We first quantify the slowdown that heterogeneity introduces in ring attention execution, where performance is highly sensitive to load balance. We then propose a load-balancing methodology that explicitly accounts for rank capabilities to mitigate this slowdown and improve end-to-end inference performance.**

***Index Terms*—Distributed Inference, Heterogeneous Computing, Large Language Models (LLMs), Ring Attention, and Scalability.**

I. Introduction

A. Background and Motivation

The rapid adoption of large language models (LLMs) has accelerated the rise of inference as a service, a paradigm in which user queries are offloaded to data centers. Transformers are the backbone of the success of these models, but their scalability poses significant challenges. In particular, the memory footprint of the self-attention mechanism [1] grows quadratically with sequence length. This limitation has motivated new research directions: techniques such as FlashAttention [2], which avoid materializing the entire attention matrix by processing it in tiles combined with online softmax computation [3], [4], and distributed execution approaches such as ring attention [5], where workloads are partitioned across multiple nodes to overcome per-node memory constraints.

While many modern datacenters are still largely homogeneous clusters, it is plausible that they will become increasingly heterogeneous over time. Incremental upgrades under tight economic constraints, occasional node replacement after hardware failures, or the introduction of specialized accelerators could all gradually lead to a mix of hardware generations and capabilities within the same deployment. This heterogeneity could creates bottlenecks in synchronous distributed execution, where the overall performance is constrained by the slowest participant.

B. Problem Statement

Current implementations of Context Parallelism, including Ring Attention, implicitly assume a symmetric hardware topology. Consequently, standard partitioning strategies distribute the Key-Value (KV) cache uniformly, assigning an equal number of tokens to every rank in the process group (N_{tokens}/P_{ranks}). While optimal for homogeneous clusters, this rigid partitioning becomes a critical bottleneck in heterogeneous environments.

The core issue stems from the synchronous nature of the Ring Attention algorithm. In each step of the ring, computation (Attention calculation) and communication (P2P block transfer) are overlapped, but the transition to the next step requires a global synchronization barrier or implicit data dependency. In a cluster containing mixed accelerators (e.g., combining NVIDIA H100s with A100s), a uniform workload results in disparate execution times across ranks.

Specifically, higher-performance nodes complete their attention sub-problems rapidly and become idle, forced to wait for the slowest node (the *straggler*) to complete its computation or transmission. This "straggler effect" causes the aggregate system throughput to collapse to the speed of the weakest device multiplied by the number of nodes. As a result, the superior compute throughput and memory bandwidth of modern accelerators in the ring are wasted, and the latency of end-to-end inference remains unacceptably high despite the presence of capable hardware.

C. Objectives and Scope

Building on this prior work, our project seeks to extend context parallelism to heterogeneous environments. Specifically, we will investigate heterogeneity in ring attention within IBM's FMS. While earlier implementations demonstrated the benefits of context splitting across homogeneous GPUs, our focus is on uneven context allocation strategies: distributing query, key, and value shards across heterogeneous GPUs in proportion to their memory capacity, compute throughput, and interconnect bandwidth. By adapting context parallelism to heterogeneous clusters, we aim to mitigate performance degradation caused by weaker nodes and enable the scalability of long-context inference in heterogeneous environments.

II. Literature Review

A. Review of Relevant Literature

The transformer architecture introduced by Vaswani et al. [1] established self-attention as the core building block for modern LLMs, but with a quadratic memory and compute cost in sequence length. This limitation has motivated a line of work that reduces the memory footprint of attention on a

single GPU. Rabe and Staats show that self-attention does not fundamentally require O(n$\hat{2}$) memory and can be implemented in a streaming manner using online softmax formulations [3]. Building on these ideas, FlashAttention proposes an IO-aware attention kernel that tiles the attention computation, keeps Q,K,V in on-chip SRAM, and uses online softmax to avoid materializing the full attention matrix [2]. These kernels significantly reduce memory traffic and improve latency, but they assume the entire sequence still fits on one device.

To go beyond the memory capacity of a single GPU, recent work studies blockwise and context-parallel transformers. Blockwise Parallel Transformers process long sequences as blocks while preserving exact attention via parallelizable schedules [4]. Ring Attention further extends this idea by arranging devices in a ring topology and circulating KV blocks between them, so each device holds only a shard of the context while still computing full-sequence attention [5]. This provides a practical recipe for context parallelism across homogeneous devices, and is now a common reference design for multi-GPU long-context inference.

A separate body of work focuses on heterogeneous systems, where GPUs differ in compute and memory capabilities. Systems such as Helix study serving LLMs over heterogeneous GPUs and networks using max-flow–style formulations for layer placement and request routing, targeting pipeline and tensor parallelism across mixed accelerators rather than context sharding [6]. Other recent work on "efficient inference on heterogeneous GPUs" similarly optimizes where to place model layers and how to schedule requests across devices with different speeds and bandwidths [7], but does not modify the attention algorithm itself or the way sequence tokens are partitioned.

There is also work on algorithmic variants for long-context attention that aim to reduce the effective cost per token without changing the underlying hardware. Star Attention and Infini-attention both restructure the attention pattern to support infinite or near-infinite context via sparsity, compressed state, or special memory tokens, rather than by sharding the KV cache across devices [8], [9]. While these methods significantly improve scalability for a single model instance, they introduce architectural changes and approximations to standard attention, and they do not address how to balance work across heterogeneous GPUs within an existing exact ring-attention setup.

B. Identification of Gaps in Existing Research

Existing ringattention implementations assume symmetric hardware and therefore partition the KV cache uniformly as $\frac{N_{\text{tokens}}}{P_{\text{ranks}}}$. In a heterogeneous setting, this creates the exact straggler behavior described in our introduction: faster GPUs frequently idle while waiting for slower ranks to finish their shard of attention and communication, collapsing throughput to the speed of the weakest device.

Moreover, most ring-attention implementations combine FlashAttention-style kernels with homogeneous context splits. They exploit tiling and SRAM reuse on each device, but they do not adapt how many tokens each GPU owns based on its effective throughput, nor do they combine this with an online-softmax formulation that can safely aggregate attention across uneven KV shards.

Our work directly targets this gap: we study heterogeneity-aware context parallelism in ring attention within IBM's FMS. Concretely, we (i) quantify the slowdown that realistic intra-node heterogeneity induces under standard even context splits, and (ii) propose and evaluate uneven KV-sharding strategies that allocate longer context segments to faster GPUs, while preserving exact attention using an online-softmax-compatible ring kernel. This positions our contribution at the intersection of FlashAttention-style memory-efficient kernels, ring-based context parallelism, and heterogeneous-systems scheduling, but focused specifically on context splitting for long-sequence inference on heterogeneous GPUs.

III. Methodology

A. Model Selection

Our experiments are built on top of IBM's Foundation Model Stack (FMS) implementation of LLaMA-style decoder-only transformers. Rather than running the full LLaMA-3.1-8B model end-to-end, we focus on a single multi-head self-attention layer with the same head configuration used in the FMS LLaMA blocks (e.g., $Q, K, V \in \mathbb{R}^{1\times 8\times L\times 64}$ in our logs). This isolates the effect of context sharding and communication/computation overlap in ring attention without confounding effects from feed-forward layers or logits projection.

We introduce a new RingAttentionStrategy, which exposes a block_lens argument to control how many tokens are assigned to each rank. The strategy manages a default compute stream and a dedicated communication stream so that peer-to-peer KV transfers can overlap with attention compute in the ring. The LLaMA implementation is extended to call a custom ring_forward function, which replaces the standard attention block.

The core of the implementation is a pass-KV ring attention kernel in which queries remain local while keys and values are rotated around the ring. The kernel uses an online softmax formulation, allowing us to accumulate attention outputs across KV blocks without materializing the full attention matrix.

B. Triton Kernels for Online Softmax

To efficiently support online softmax in the ring, we implement custom Triton kernels. Our extension computes block-wise softmax statistics for tiles of size (BLOCK_Q, BLOCK_K). These kernels return per-query partial sums and maxima that are merged by the online softmax update rule. The diagonal blocks (local queries attending local keys) and off-diagonal blocks (queries attending remote KV segments) are both handled in this tiled fashion; the off-diagonal path uses the Triton kernel we added. This setup lets us keep the online softmax logic in PyTorch while offloading the heavy per-tile GEMM and reduction work to Triton.

C. Optimization Procedure

Given a heterogeneous ring where rank capabilities differ, the key challenge is determining how to partition the sequence across ranks. We investigate four partitioning strategies that vary in complexity and information requirements.

Even Split (Baseline). The standard approach assigns an equal number of tokens to each rank: $n_i = N/P$ for all ranks i, where N is the total sequence length and P is the number of ranks. This strategy is optimal when all ranks have identical performance but becomes suboptimal when heterogeneity is present.

Uneven Split. When the relative performance of each rank is known (e.g., through MPS percentage or hardware specifications), tokens can be distributed proportionally. For a two-rank system with rank 0 at full capacity and rank 1 throttled to $s\%$ of full performance, we assign:

$$n_0 = N \cdot \frac{1}{1+s}, \quad n_1 = N \cdot \frac{s}{1+s} \tag{1}$$

This ensures that both ranks complete their workload in approximately the same time, minimizing idle cycles.

LUT Split. Rather than assuming a linear relationship between MPS percentage and performance, we construct a lookup table (LUT) from offline profiling. For each combination of sequence length and MPS setting, we measure actual execution time and use these measurements to derive optimal token allocations. This approach captures non-linear effects that the simple proportional model may miss.

Formula Split. We fit a polynomial regression model to the profiling data, yielding a closed-form expression for predicted performance as a function of sequence length and MPS percentage. The fitted model achieves $R^2 = 0.922$ and allows token allocation without requiring a full lookup table.

Figure 1 illustrates how these strategies redistribute tokens across ranks as heterogeneity increases. At mild heterogeneity (90% MPS), all strategies produce similar allocations. At severe heterogeneity (10% MPS), the adaptive strategies shift substantially more tokens to the faster rank.

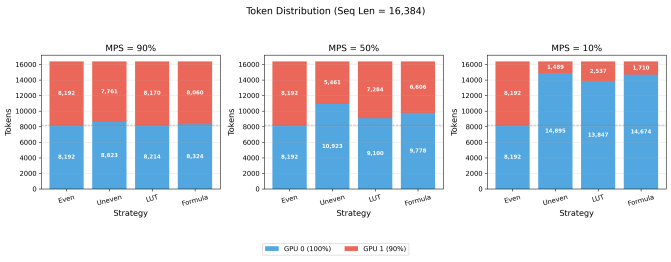

Fig. 1. Token distribution across GPUs under different partitioning strategies. As heterogeneity increases (lower MPS), adaptive strategies assign more tokens to the faster GPU to balance execution time.

D. Profiling Tools and Methods

To simulate heterogeneous GPU performance within a homogeneous cluster, we use NVIDIA's Multi-Process Service (MPS) to throttle individual GPUs to a fraction of their full compute capacity. MPS allows multiple processes to share a GPU and, critically, enables limiting the percentage of SM (Streaming Multiprocessor) resources available to each process. By setting MPS active thread percentage to values between 10% and 100%, we can emulate GPUs with varying compute capabilities.

We first characterize the relationship between MPS throttling and actual compute performance through microbenchmarks. Figure 2 shows matrix multiplication latency across different matrix sizes and MPS settings. The results confirm that MPS throttling produces predictable performance degradation that scales with problem size.

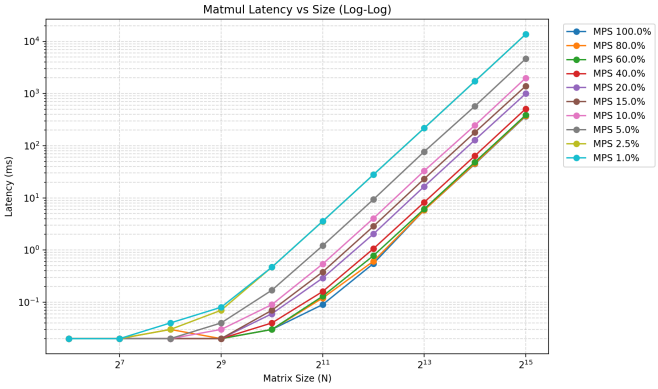

Fig. 2. Matrix multiplication latency as a function of matrix size under different MPS throttling levels. Lower MPS percentages result in proportionally higher latency for compute-bound operations.

Figure 3 shows normalized performance (relative to 100% MPS) across different MPS settings and sequence lengths.

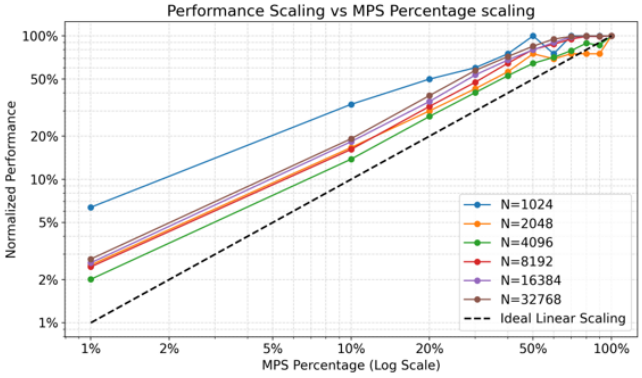

Fig. 3. Normalized performance scaling with MPS percentage. Performance scales approximately linearly with MPS allocation, enabling predictable modeling of heterogeneous configurations.

To enable the formula-based partitioning strategy, we fit a degree-2 polynomial regression model to the profiling data. The model takes sequence length and MPS percentage as inputs and predicts normalized performance. Figure 4 shows the fitted regression surface, which achieves $R^2 = 0.922$ on the training data.

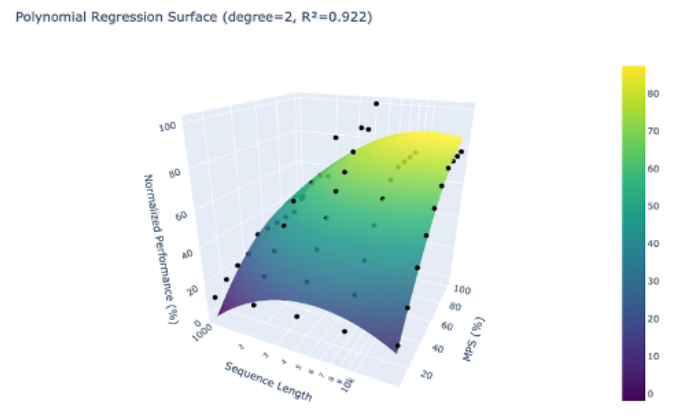

Fig. 4. Polynomial regression surface for performance prediction. The model captures the joint dependence on sequence length and MPS percentage with $R^2 = 0.922$.

E. Evaluation Metrics

We evaluate partitioning strategies using the following metrics:

Prefill latency. The wall-clock time for the attention layer to complete across all ranks, determined by the slowest rank. Since ring attention requires synchronization between ranks, overall latency is bounded by the straggler.

Slowdown factor. The ratio of heterogeneous execution time to homogeneous baseline execution time: Slowdown $= T_{hetero}/T_{homo}$. A slowdown of 1.0 indicates no performance loss from heterogeneity.

Efficiency. The inverse of slowdown, expressed as a percentage: Efficiency $= (T_{homo}/T_{hetero}) \times 100\%$. This measures how much of the ideal (homogeneous) performance is retained.

Speedup over baseline. The ratio of even-split latency to adaptive-strategy latency: Speedup $= T_{even}/T_{adaptive}$. This quantifies the improvement from using heterogeneity-aware partitioning.

IV. Experimental Results

A. Experimental Setup

All experiments are conducted on a single node equipped with two NVIDIA L40 GPUs. We use IBM's Foundation Model Stack with our modified RingAttentionStrategy implementation.

Hardware heterogeneity is simulated rather than using different GPU models. To simulate heterogeneity, we run rank 0 at 100% MPS and vary rank 1's MPS allocation from 10% to 90% in increments of 10%. This creates heterogeneity ratios ranging from 1.1:1 (mild) to 10:1 (severe). We sweep sequence lengths from 4,096 to 65,536 tokens, covering the range relevant to long-context inference workloads.

To decouple systems behavior from model quality, we use synthetic inputs throughout. For the single-layer ring-attention microbenchmarks, we directly sample random Q,K,V tensors on GPU.

Each configuration is run 5 times after a warmup pass, and we report the median latency. The reference homogeneous baseline runs both ranks at 100% MPS with even partitioning.

B. Performance Comparison

Figure 5 presents the main experimental results, showing slowdown factor relative to the homogeneous baseline across all sequence lengths and MPS configurations. Several patterns emerge from these results.

First, even split performance degrades rapidly as heterogeneity increases. At 10% MPS, the even split strategy exhibits slowdowns of 5-8x across all sequence lengths, as the faster rank spends most of its time waiting for the throttled rank to complete.

Second, the adaptive strategies substantially reduce this slowdown. The uneven split strategy consistently achieves the best or near-best performance across configurations. At extreme heterogeneity (10% MPS), uneven split reduces slowdown from 5-8x to approximately 2x, recovering a significant portion of the lost performance.

Third, the relative effectiveness of LUT and formula strategies varies with configuration. LUT split performs well at moderate heterogeneity but sometimes underperforms at extremes, likely due to interpolation artifacts in the lookup table. Formula split provides consistent middle-ground performance but rarely achieves the best results.

Figure 6 provides an alternative view of these results, showing the speedup of uneven split over even split as a heatmap. The speedup increases with both sequence length and heterogeneity, reaching 4.4x at 65K tokens with 10% MPS.

Figure 7 shows absolute latency at the most extreme heterogeneity setting (10% MPS). The difference between strategies is most pronounced here: even split requires over 6 seconds for 65K tokens, while uneven split completes in under 1.4 seconds.

V. Discussion

A. Interpretation of Results

The experimental results confirm that heterogeneity-aware partitioning can recover substantial performance in ring attention under heterogeneous conditions. The key insight is that the straggler effect in synchronous distributed execution can be mitigated by reducing the workload assigned to slower ranks.

Figure 8 shows efficiency (relative to homogeneous baseline) across MPS configurations. Even split efficiency drops below 20% at severe heterogeneity, meaning over 80% of potential performance is lost to waiting. The adaptive strategies maintain 40-50% efficiency even at 10% MPS, representing a 2-3x improvement in resource utilization.

The uneven split strategy's consistent strong performance suggests that a simple proportional allocation based on known rank capabilities is sufficient for most scenarios. The additional complexity of LUT or formula-based approaches provides marginal benefit and may introduce brittleness when the profiling conditions do not match runtime conditions.

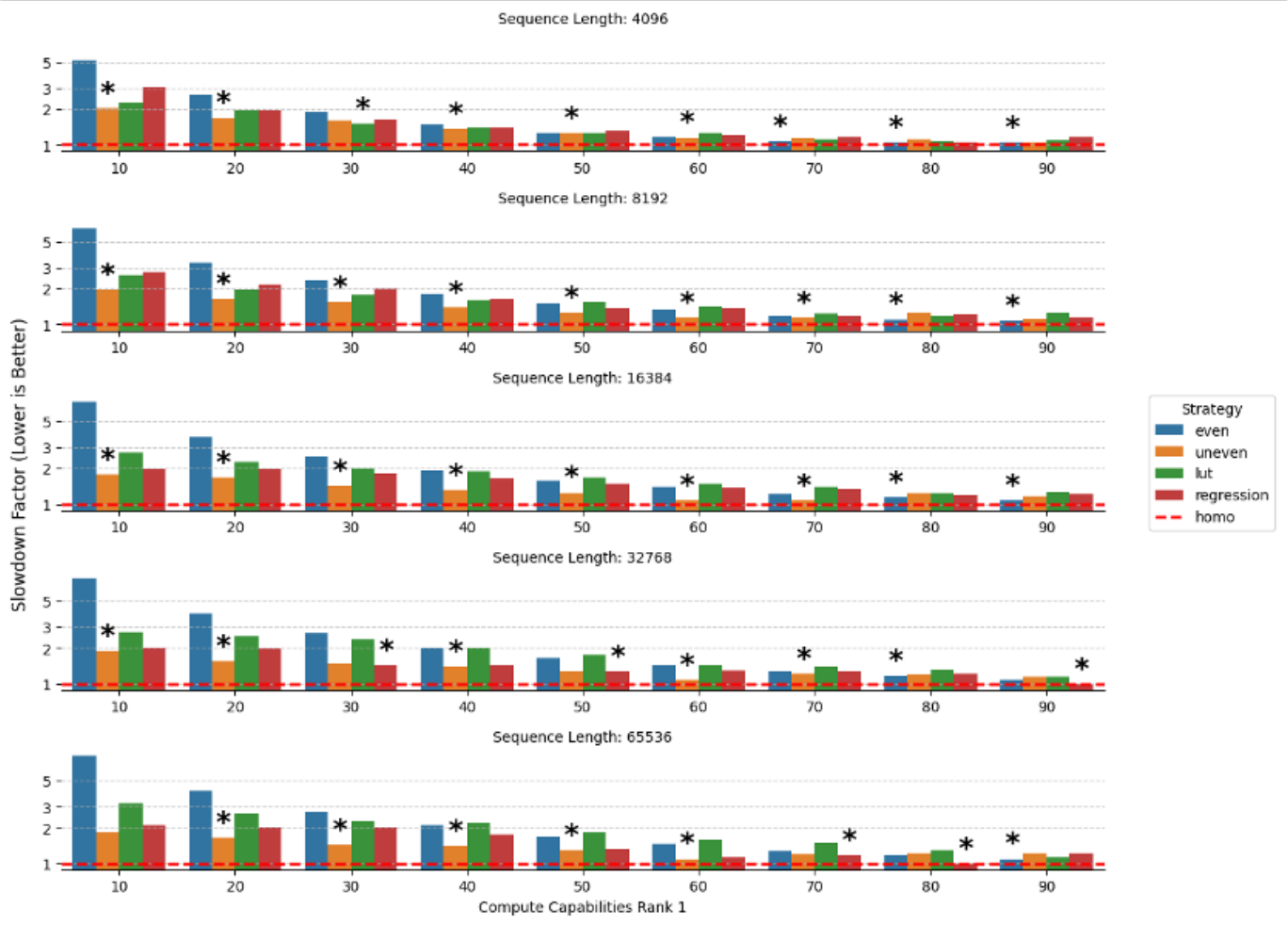

Fig. 5. Slowdown factor for each partitioning strategy across sequence lengths and MPS configurations. Lower values indicate better performance. The dashed red line at 1.0 represents ideal (homogeneous) performance. Asterisks mark the best-performing strategy for each configuration.

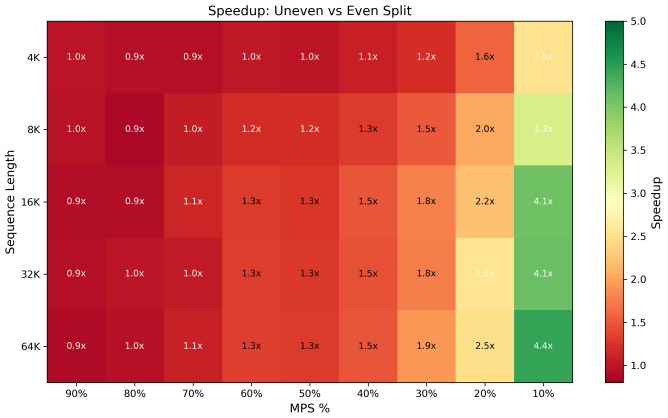

Fig. 6. Speedup of uneven split over even split. Darker green indicates larger improvements. The greatest benefits occur at high heterogeneity (low MPS) and long sequences.

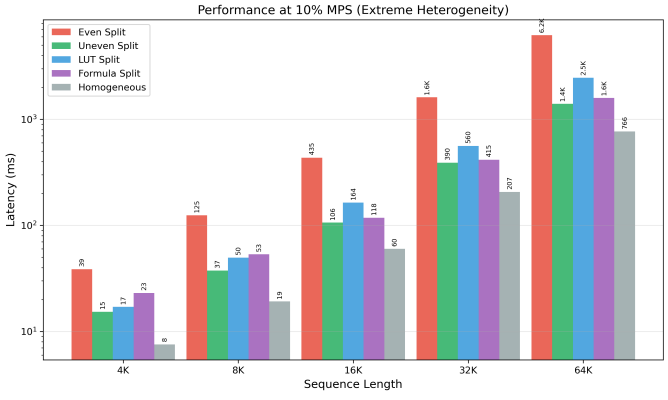

Fig. 7. Absolute latency at 10% MPS (extreme heterogeneity). Adaptive strategies reduce latency by up to 4.4x compared to even split.

B. Comparison with Previous Studies

Prior work on context parallelism, including the original ring attention paper and IBM's FMS implementation, focused

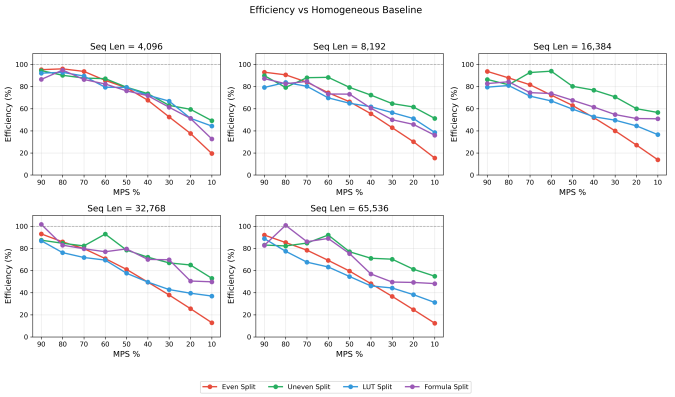

Fig. 8. Efficiency relative to homogeneous baseline. Adaptive strategies maintain substantially higher efficiency as heterogeneity increases.

exclusively on homogeneous configurations. To our knowledge, this is the first systematic study of ring attention performance under controlled heterogeneity. Our results demonstrate that the implicit homogeneity assumption in existing implementations leads to severe performance degradation when violated, and that simple modifications to the partitioning strategy can largely recover the lost performance.

C. Challenges and Limitations

Several limitations affect the generalizability of our results. First, we simulate heterogeneity using MPS throttling rather than actual mixed hardware. While MPS provides controllable and reproducible heterogeneity, real mixed-GPU deployments may exhibit different performance characteristics, particularly in communication patterns.

Second, our experiments use a two-rank configuration. Scaling to larger rings introduces additional complexity, as the optimal partitioning must account for multiple heterogeneous ranks simultaneously. The proportional allocation approach extends naturally, but the interaction effects may differ.

Third, we focus on the prefill (prompt processing) phase of inference. The decode phase, which generates tokens autoregressively, has different computational characteristics and may respond differently to heterogeneous partitioning.

D. Future Directions

Several directions merit further investigation. Dynamic rebalancing during execution could adapt to runtime performance variations rather than relying on static profiling. Integration with other parallelism strategies (tensor parallelism, pipeline parallelism) would enable heterogeneity-aware partitioning in more complex deployment configurations. Finally, extending the approach to actual mixed-hardware deployments would validate the practical applicability of our methodology.

VI. Conclusion

A. Summary of Findings

We investigated the impact of GPU heterogeneity on ring attention performance and proposed partitioning strategies to mitigate the resulting slowdown. Our experiments demonstrate that:

- Standard even partitioning causes severe performance degradation under heterogeneity, with slowdowns exceeding 5x at 10:1 capability ratios.
- Simple proportional partitioning (uneven split) recovers most of the lost performance, achieving up to 4.4x speedup over even split.
- More complex strategies (LUT, formula) provide marginal additional benefit and may introduce brittleness.

B. Contributions

This work makes three contributions. First, we quantify the performance impact of heterogeneity in ring attention, establishing that the straggler effect causes substantial slowdown even at moderate capability differences. Second, we propose and evaluate multiple partitioning strategies for heterogeneous rings, identifying proportional allocation as an effective and practical approach. Third, we provide an open-source implementation of heterogeneity-aware ring attention in IBM's FMS framework.

C. Recommendations for Future Research

Based on our findings, we recommend that distributed inference frameworks treat heterogeneity-aware partitioning as a first-class design concern. The proportional allocation scheme we evaluated is simple to implement and already recovers most of the lost throughput under heterogeneous MPS configurations, making it a low-cost addition to existing ring-attention implementations. Future work should extend these techniques to multi-node deployments and explore dynamic adaptation of shard sizes based on online measurements of per-rank performance, rather than relying on a one-time calibration pass.

On the kernel and system side, several directions are especially promising:

a. **GQA Kernel Optimization:** Our current Triton kernels target standard multi-head attention; extending them to Grouped Query Attention would be important for modern LLM architectures, where reducing KV bandwidth and memory traffic is critical.

b. **Disaggregated Prefill–Decode:** The prefill phase is predominantly compute-bound, while decode is often memory and bandwidth-bound. A disaggregated architecture that treats prefill and decode as separate services could assign long-context prefill to faster GPUs and route decode to memory-optimized or cheaper accelerators, enabling finer-grained use of heterogeneous hardware.

c. **Advanced Q/KV Partitioning Algorithms:** Our proportional scheme is intentionally simple: KV tokens are split by measured throughput ratios, and queries remain evenly distributed. A natural next step is to design more sophisticated partitioning policies that jointly optimize Q and KV sharding, take into account network topology and contention, and potentially re-balance shards at runtime as loads change.

REFERENCES

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023. [Online]. Available: https://arxiv.org/abs/1706.03762

[2] T. Dao, D. Y. Fu, S. Ermon, A. Rudra, and C. Ré, "Flashattention: Fast and memory-efficient exact attention with io-awareness," 2022. [Online]. Available: https://arxiv.org/abs/2205.14135

[3] M. N. Rabe and C. Staats, "Self-attention does not need $o(n^2)$ memory," 2022. [Online]. Available: https://arxiv.org/abs/2112.05682

[4] H. Liu and P. Abbeel, "Blockwise parallel transformer for large context models," 2023. [Online]. Available: https://arxiv.org/abs/2305.19370

[5] H. Liu, M. Zaharia, and P. Abbeel, "Ring attention with blockwise transformers for near-infinite context," 2023. [Online]. Available: https://arxiv.org/abs/2310.01889

[6] Y. Mei, Y. Zhuang, X. Miao, J. Yang, Z. Jia, and R. Vinayak, "Helix: Serving large language models over heterogeneous gpus and network via max-flow," in *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, 2025, pp. 586–602.

[7] Y. Liu, Q. Xu, and Y. C. Hu, "Cronus: Efficient llm inference on heterogeneous gpu clusters via partially disaggregated prefill," *arXiv preprint arXiv:2509.17357*, 2025.

[8] S. Acharya, F. Jia, and B. Ginsburg, "Star attention: Efficient llm inference over long sequences," *arXiv preprint arXiv:2411.17116*, 2024.

[9] T. Munkhdalai, M. Faruqui, and S. Gopal, "Leave no context behind: Efficient infinite context transformers with infini-attention," *arXiv preprint arXiv:2404.07143*, vol. 101, 2024.