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, 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, degradation of the hardware ranks, 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 create 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. 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 and communication are overlapped, but the transition to the next step requires a global synchronization barrier.

This synchronization constraint gives rise to the well-known straggler effect: faster nodes complete their local attention computations early but are forced to wait for the slowest rank before the ring can advance. Without any mitigation strategy, the entire system becomes fundamentally limited by the capabilities of the weakest device, since each step's duration is dictated by the slowest compute or communication path. As a result, the effective throughput collapses to the straggler's performance multiplied by the number of ring stages, leaving the superior compute and memory bandwidth of modern accelerators underutilized and keeping end-to-end inference latency high despite the presence of much 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 [6]. 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 compute capabilities. 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 [1] established self-attention as the fundamental operation for large language models, yet its memory and compute requirements scale quadratically with the sequence length. This limitation has motivated extensive research on reducing the memory footprint of attention within a single GPU. Prior work demonstrates that self-attention does not inherently require $O(n^2)$ memory and can be implemented in a streaming fashion using online softmax formulations [3].

Building on this insight, FlashAttention introduces an IO-aware attention kernel that tiles the computation, keeps Q, K, and V in on-chip SRAM, and uses online softmax to avoid materializing the full attention matrix [2]. These approaches significantly reduce memory traffic and improve runtime, but the entire sequence must still fit on a single device.

To overcome single-GPU memory limits, a parallel line of work investigates blockwise and context-parallel transformers. Blockwise Parallel Transformers process long sequences in blocks while preserving exact attention through parallelizable execution schedules [4]. Ring Attention extends this strategy across multiple devices by arranging them in a ring topology and circulating KV blocks so that each device holds only a shard of the context while still computing full-sequence attention [5]. This design provides a practical mechanism for distributed context parallelism on homogeneous clusters and has become a standard reference architecture for multi-GPU long-context inference.

A separate set of systems targets heterogeneous GPU clusters in which devices differ in compute or memory capabilities. These systems formulate model placement and request routing as global optimization problems, often using flow-based or scheduling-based methods to allocate layers across mixed accelerators [7]. While effective for pipeline and tensor parallelism, these solutions do not modify the attention algorithm itself or the way sequence tokens are partitioned. Similar techniques for efficient inference on heterogeneous GPUs focus on optimizing device utilization and interconnect bandwidth [8], but likewise leave context sharding untouched.

Finally, several algorithmic variants aim to support long or unbounded context lengths without changing the underlying hardware configuration. Methods such as Star Attention and Infini-attention restructure the attention pattern using sparsity, compressed state, or specialized memory tokens to achieve near-infinite context [9], [10]. Although these approaches improve scalability for single-device deployments, they introduce architectural modifications and approximations to standard attention, and they do not address how to balance work across heterogeneous GPUs within existing exact ring-attention frameworks.

B. *Identification of Gaps in Existing Research*

Existing ring attention implementations assume symmetric hardware and therefore partition the KV cache uniformly. In heterogeneous settings, this leads directly to the straggler behavior described earlier: faster GPUs frequently idle while waiting for slower ranks to complete their portion of attention and communication, reducing overall throughput to that of the weakest rank.

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 addresses this gap by studying heterogeneity-aware context parallelism for ring attention within IBM's FMS. We (i) quantify the slowdown introduced by realistic intra-node heterogeneity under standard uniform context splits, and (ii) propose and evaluate uneven KV-sharding strategies that allocate larger context segments to faster GPUs while preserving exact attention through an online-softmax-compatible ring kernel.

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. Both diagonal blocks (local queries attending local keys) and off-diagonal blocks (queries attending remote KV segments) use the same Triton kernel to ensure all ranks execute identical code paths and avoid NCCL deadlocks. This setup lets us keep the online softmax merging 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.

Lookup Table 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.

Regression-Based 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.

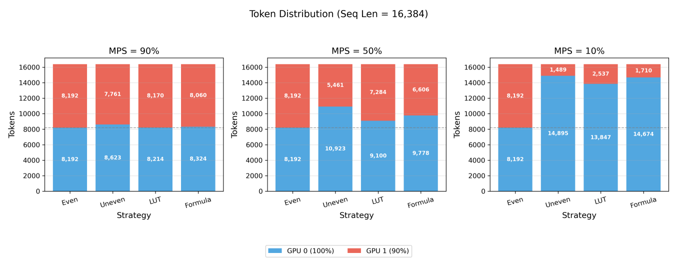

Fig. 1. Token distribution across GPUs under different partitioning strategies. As heterogeneity increases (lower MPS), adaptive strategies assign more tokens to the faster GPU to balance execution time.

D. Profiling Tools and Methods

To simulate heterogeneous GPU performance within a homogeneous cluster, we use NVIDIA's Multi-Process Service (MPS) [11] 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 Streaming Multiprocessors (SM) resources available to each process. Specifically, we set the `CUDA_MPS_ACTIVE_THREAD_PERCENTAGE` environment variable to values between 10% and 100% before launching each rank, which restricts the fraction of GPU compute resources available to that process. This allows us to emulate GPUs with varying compute capabilities on identical hardware.

We first characterize the relationship between MPS throttling and actual compute performance through a custom benchmark. 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.

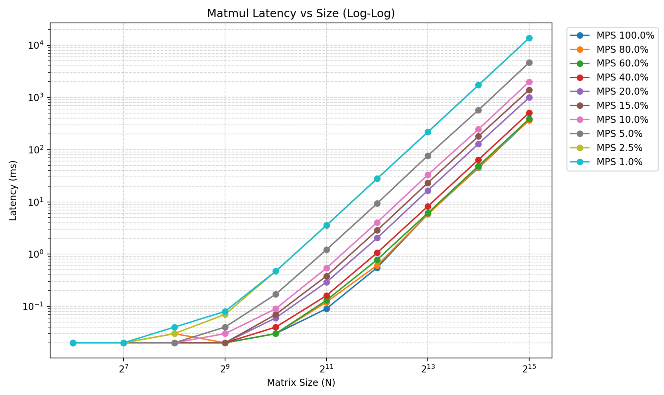

Fig. 2. Matrix multiplication latency as a function of matrix size under different MPS throttling levels. Lower MPS percentages result in proportionally higher latency for compute-bound operations.

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

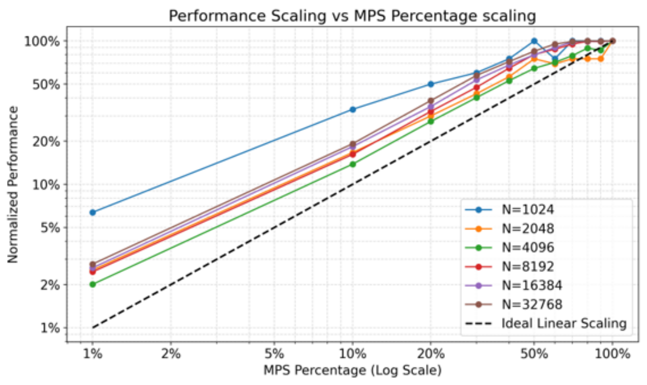

Fig. 3. Normalized performance scaling with MPS percentage. Performance scales approximately linearly with MPS allocation, enabling predictable modeling of heterogeneous configurations.

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.

E. Evaluation Metrics

We evaluate partitioning strategies using the following metrics:

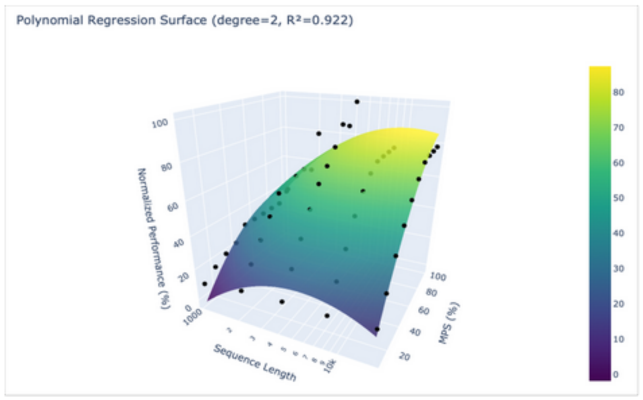

Fig. 4. Polynomial regression surface for performance prediction. The model captures the joint dependence on sequence length and MPS percentage with $R^2 = 0.922$.

Prefill latency. The wall-clock time for the attention layer to complete across all ranks.

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 performance gain from heterogeneity-aware KV partitioning.

IV. Experimental Results

A. Experimental Setup

All experiments are conducted on a single node equipped with two NVIDIA L40 GPUs using IBM's Foundation Model Stack and our custom Ring-Attention implementation. Hardware heterogeneity is emulated leveraging MPS. Specifically, we fix rank 0 at 100% MPS and vary rank 1's allocation from 10% to 90% in increments of 10%, yielding heterogeneity ratios from roughly 1.1:1 (mild) to 10:1 (severe). We sweep sequence lengths from 4,096 to 65,536 tokens to cover the range relevant for long-context inference.

To isolate systems-level behavior from model semantics, we use synthetic inputs throughout. For single-layer ring-attention microbenchmarks, we generate random Q, K, and V tensors directly on the GPU.

Each configuration is run five times following a warmup pass, and we report the median latency. The homogeneous baseline runs both ranks at 100% MPS with an even KV partition.

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.

B. Comparison with Previous Studies

Prior work on context parallelism, including the original ring attention paper and IBM's FMS implementation, focused exclusively on homogeneous configurations. To our knowledge, this is the first study of ring attention performance under controlled heterogeneity. Our results demonstrate that

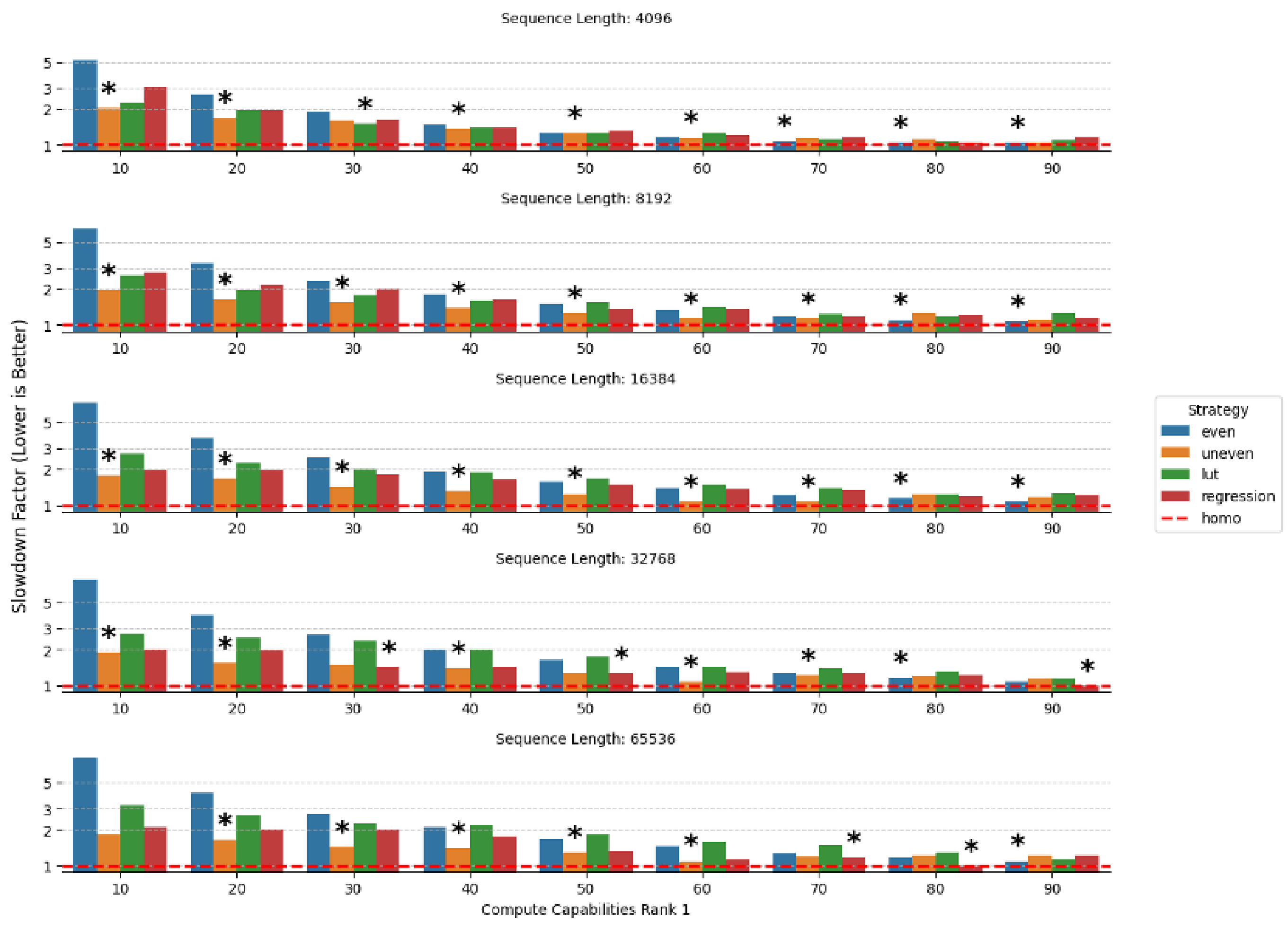

Fig. 5. Slowdown factor for each partitioning strategy across sequence lengths and MPS configurations. Lower values indicate better performance. The dashed red line at 1.0 represents ideal (homogeneous) performance. Asterisks mark the best-performing strategy for each configuration.

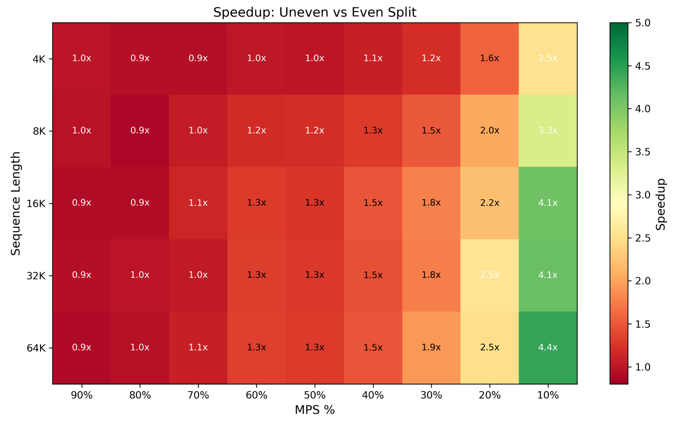

Fig. 6. Speedup of uneven split over even split. Darker green indicates larger improvements. The greatest benefits occur at high heterogeneity (low MPS) and long sequences.

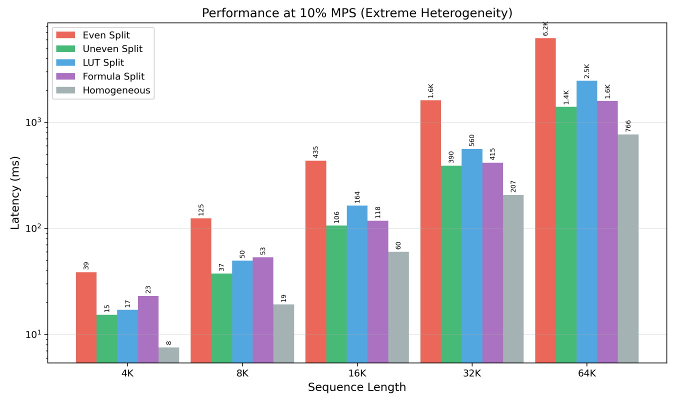

Fig. 7. Absolute latency at 10% MPS (extreme heterogeneity). Adaptive strategies reduce latency by up to 4.4x compared to even split.

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.

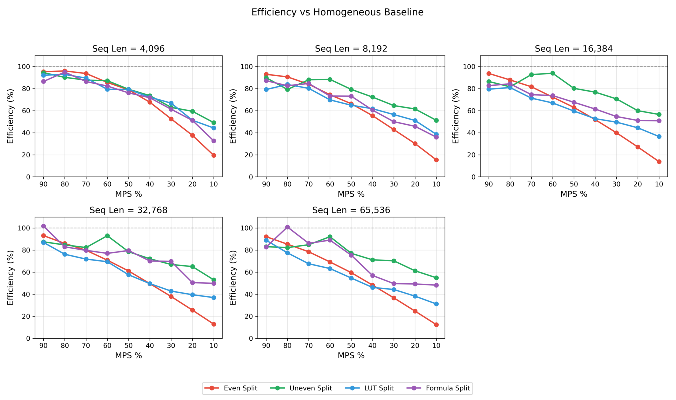

Fig. 8. Efficiency relative to homogeneous baseline. Adaptive strategies maintain substantially higher efficiency as heterogeneity increases.

C. Challenges and Limitations

Several limitations affect the generalizability of our results. First, we simulate heterogeneity via MPS throttling rather than actual mixed hardware, which may not capture all real-world performance characteristics. Second, our two-rank configuration does not address the added complexity of partitioning across larger heterogeneous rings. Third, we focus on the prefill phase; the decode phase has different computational characteristics. Fourth, we evaluate a single attention layer rather than a full end-to-end model with feed-forward layers and other components. Fifth, we measure single-request latency rather than throughput under concurrent load, which differs from production serving conditions.

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, available at https://github.com/AnshSG13/hetero_gpu_context_parallelism.

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: both Q and KV tokens are split by measured throughput ratios. A natural next step is to design more sophisticated partitioning policies that 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] IBM Research, “Foundation model stack (fms),” https://github.com/foundation-model-stack/foundation-model-stack, 2024, accessed: 2025-12-01.

[7] 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.

[8] 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.

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

[10] 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.

[11] NVIDIA Corporation, “Multi-process service (mps) documentation,” https://docs.nvidia.com/deploy/mps/index.html, 2024, accessed: 2025-12-01.