Exploring High-Bandwidth Flash for Modern LLM Inference: Opportunities and Challenges

NVRAMOS 2026

Jisung Park

POSTECH CAOS
COMPUTER ARCHITECTURE & OPERATING SYSTEMS LABORATORY

2026.10.22

Exploring HBF for LLM-Serving Systems

- D. Son et al., "Exploring High-Bandwidth Flash for Modern LLM Inference: Opportunities and Challenges," IEEE Computer Architecture Letters (IEEE CAL), 2026.

Exploring High-Bandwidth Flash for Modern LLM Inference: Opportunities and Challenges

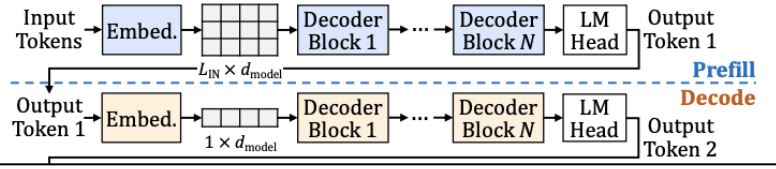
Dowon Son, Yonggon Park, Hyunuk Cho, Hyungkyu Ham, Onur Mutlu, Sungjin Lee, Gwangsun Kim, and Jisung Park

***Abstract*—This work investigates the potential benefits and technical challenges of using high-bandwidth flash (HBF) for large language model (LLM) inference. HBF has gained increasing attention as a promising solution to mitigate memory-capacity bottlenecks in modern LLM-serving systems, but its benefits and challenges remain largely uninvestigated. To address this gap, we thoroughly analyze HBF-based LLM-serving systems under diverse system configurations and operating scenarios in which HBF serves as a main GPU-memory component to handle both reads and writes. Our analysis shows that, despite its limited write performance, HBF can significantly improve the batch size, throughput, and flexibility of LLM-serving systems while reducing the minimum GPU requirements, but realizing these benefits critically depends on sustaining HBM-comparable read bandwidth and requires significant endurance improvements.**

***Index Terms*—High-bandwidth flash, LLM-serving systems**

II. Background and Motivation

LLM Inference. Fig. 1(a) depicts the overall process of LLM inference, which consists of two phases: (*i*) *prefill* and (*ii*) *decode*. Both phases pass through the same token-embedding layer, decoder blocks, and language-model (LM) head. Given a query with L_{IN} tokens, the prefill phase processes all tokens in parallel to generate the first output token. The decode phase then generates L_{OUT} tokens *auto-regressively*, using each output token as a new input token for the next iteration.



Executive Summary

- Motivation: High-bandwidth flash (HBF) is gaining significant attention as a promising alternative or complement to high-bandwidth memory (HBM)
- Goal: Understand HBF's potential benefits and technical challenges
  - In modern LLM-serving systems
  - When used as the main GPU-memory component
- Methodology: High-level simulation of HBF-enabled LLM-serving systems
  - Using recent large-scale LLMs
  - While varying GPU count, SLO constraint, and context length
- Ten key observations and five takeaways, including
  - HBF can significantly improve the batch size, throughput, and system flexibility
  - Providing larger capacity alone is insufficient; requires (1) HBM-comparable read bandwidth and (2) significant endurance enhancement (+ write performance)

Outline

- Background & Motivation
- Methodology
- Evaluation Results
- Conclusion

Memory Bottlenecks in LLM Inference: Capacity

- Continuous increase in model size and KV cache
  - Larger models for better inference quality
  - Longer context (and more users)
  - e.g., DeepSeek V4
    - Model size
      - Pro: 1.6T weight parameters → ~865 GB
      - Flash: 28 B (552B in V4.1) weight parameters → ~160 (510) GB
    - KV cache (Pro): 4.94 KB per token → 4.94 GB per 1M-context query (× batch size)

Capacity Requirements: $865 + 4.94 \times S_{\mathrm{BATCH}}$ GB

Memory Bottlenecks in LLM Inference: Bandwidth

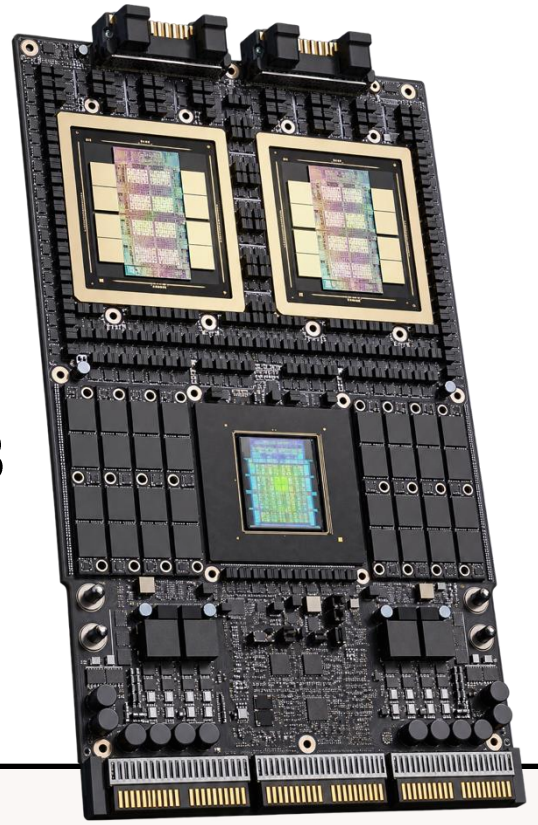
- Continuous increase in SLO-requirement level
  - 10 tokens per second (TPS) → > 100 TPS
  - Decode needs to read both weights and KV cache per token
    - 865 (model) + 4.94 × S_{BATCH} (KV cache)
  - GPU-scaling reduces the amount of model weights to read per GPU
    - 8 GPUs → ~108 GB per GPU
  - Batching amortizes model-weight accesses across queries
  - Exploiting sparsity reduces the amount of read data
    - Mixture of Experts (MoE) → ~5 GB
    - Sparse attention → ~0.7 GB per query (1M context)

Bandwidth Requirements: 2.24 TB/s
(100 iterations/sec, 1M context, per-GPU batch size: 32)

State-of-the-Art GPU: NVIDIA Rubin GPU

- HBM specifications
  - Capacity: 288 GB
  - Bandwidth: 22 TB/s

- Requirements
  - Capacity: 865 (model) + 5.2 × S_{BATCH} (KV cache + index) GB
    - 8 GPUs → 108 + 5.2 × 32 = 274.4 GB per GPU
  - Bandwidth: 100 × (5 + 0.7 × 32) = 2.24 TB/s



Memory capacity is the key bottleneck

NAND Flash-Based High-Bandwidth Memory

- SanDisk's High-Bandwidth Flash (HBF)

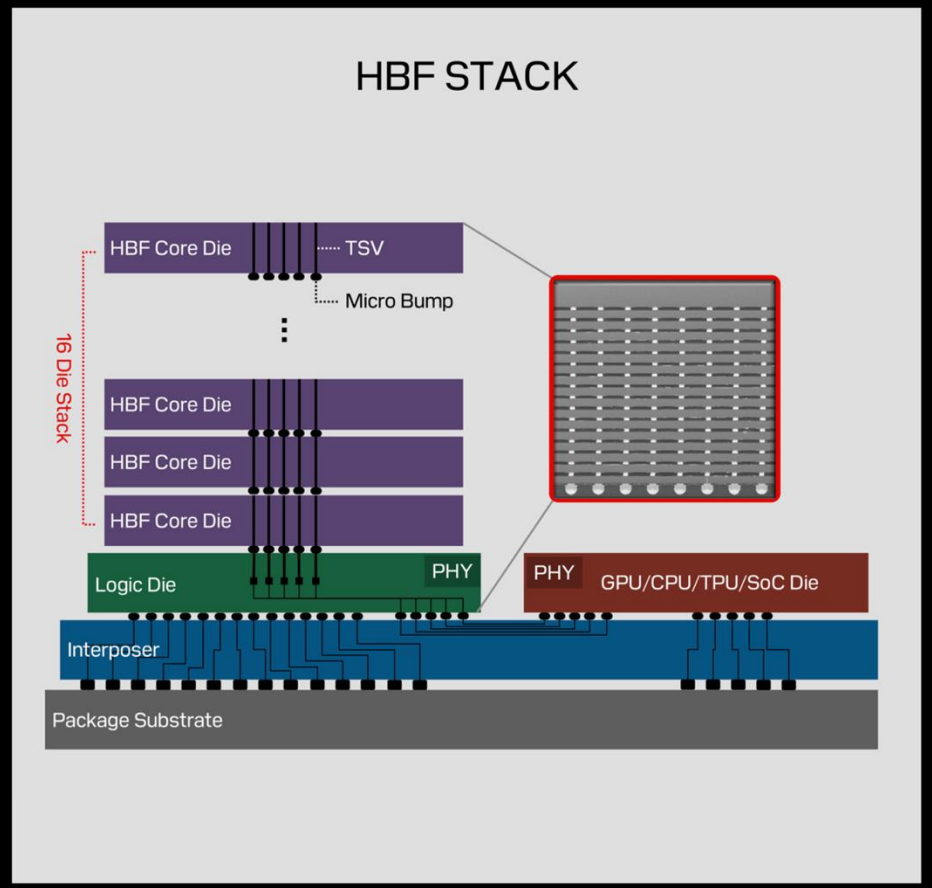
High Bandwidth Flash (HBF™)
Augmenting HBM Memory with NAND Flash for AI Inference Workloads

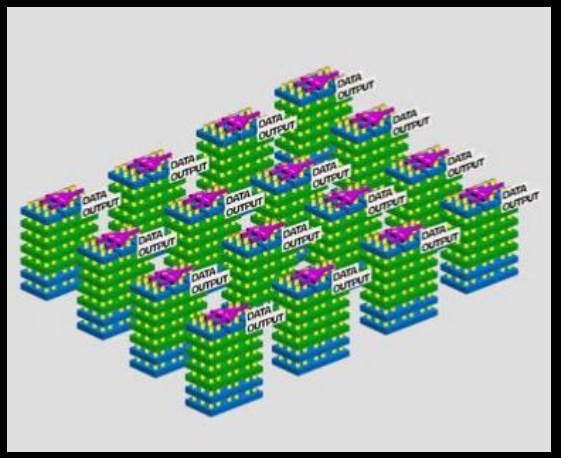
(→) Match HBM Bandwidth
Deliver 8-16X Capacity at Similar Cost

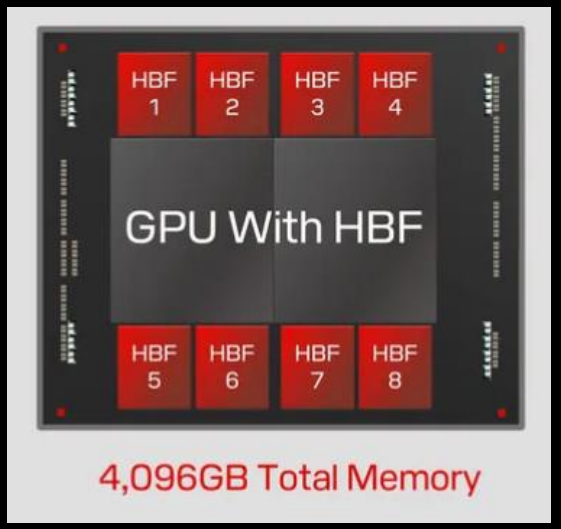
(→) Enabled by BiCS Technology
With CBA Wafer Bonding

(→) Proprietary Stacking Technology
Ultra-Low Die Warpage for 16H Stacking

(→) Architecture Developed Over the Past Year
With Inputs From Major AI Players

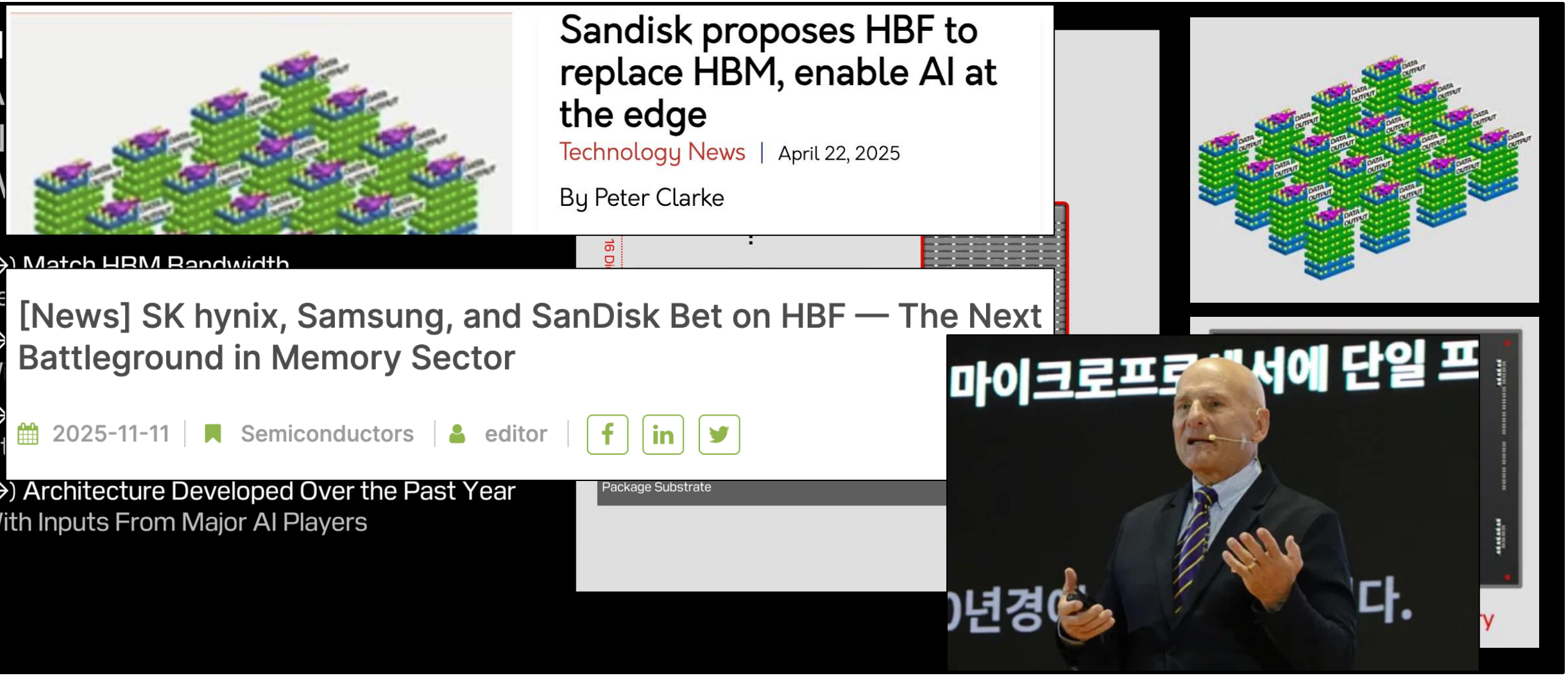






NAND Flash-Based High-Bandwidth Memory

- SanDisk's High-Bandwidth Flash (HBF)



NAND Flash-Based High-Bandwidth Memory

- SanDisk's High-Bandwidth Flash (HBF)

SanDisk's promise:
>10x capacity than & comparable bandwidth to HBM4

10% of GPUs or 10x throughput?:
We need a systematic analysis of the potential of HBF

Our Goal

Provide new insights into HBF's potential benefits and challenges through a systematic analysis of HBF-based LLM-serving systems under realistic, challenging workloads

Outline

- Background & Motivation
- Methodology
- Evaluation Results
- Conclusion

Methodology: Key Assumption

All HW components (GPU cores, memory stacks, and interconnects) operate at their specified peak performance

\+ Rapid exploration of HBF's potential benefits

\- How to achieve such performance? (future work)

Methodology: LLM-Serving Systems

- Focusing on decode nodes in disaggregated prefill-decode systems
  - Assuming high-performance prefill nodes for zero-stall batching
  - KV-cache transfer from prefill nodes to decode nodes

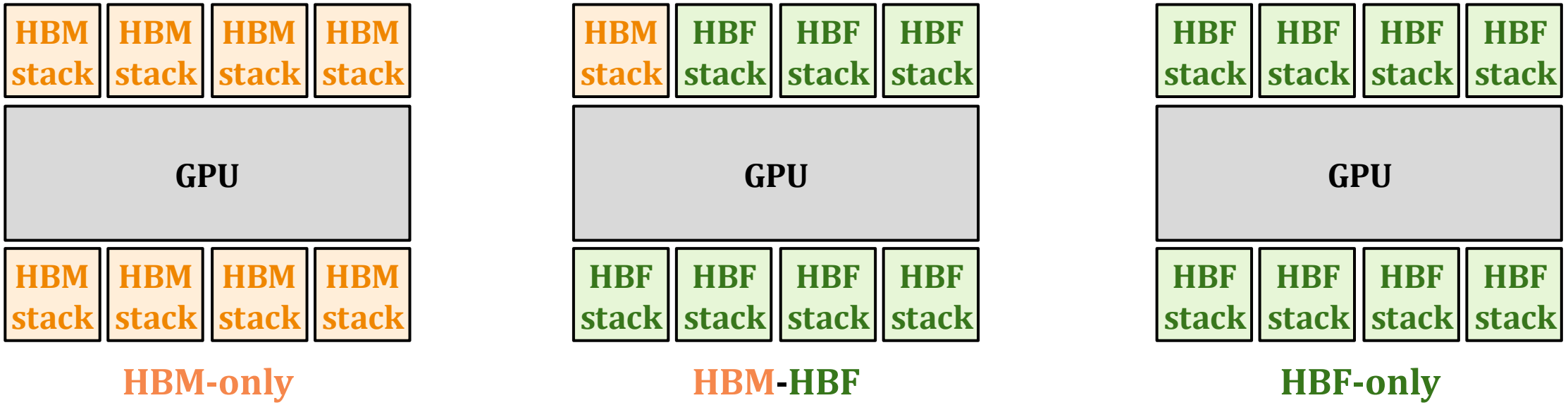
- Continuous batching, constrained by
  - GPU-memory capacity (no KV-cache spill)
  - SLO requirement (e.g., TPOT $\leq$ 0.1 s)
  - On-die SRAM capacity for HBF-only systems

- Using best parallelism scheme to maximize TPS (tokens per second)

- Modeled using an extended version of LLMSimulator [S. Yun+, MICRO'24]

GPU Configurations

- Three GPU-memory architectures
  - HBM-only: eight HBM stacks per GPU
  - HBM-HBF: one HBM stack and seven HBF stacks per GPU
    - HBM stack dedicated to intermediate data (not KV)
  - HBF-only: eight HBF stacks per GPU
    - Equipped with 40-MB SRAM on logic die to store intermediate data



Methodology: GPU Configurations (Cont'd)

- R100 GPU core (4 PFLOPS for BF16)
- Interconnects
  - Intra-node (~8 GPUs): 1,800 GB/s NVLink
  - Inter-node: 100 GB/s InfiniBand
- HBM stacks (HBM4)
  - Per-stack read bandwidth: 1.6 TB/s
  - Per-stack capacity: 36 GB
- HBF stacks (based on SanDisk's projection)
  - Per-stack read bandwidth: 1.6 TB/s
  - Per-stack capacity: 512 GB

Methodology: GPU Configurations (Cont'd)

- HBF model based on conventional low-latency NAND flash
  - It may not be easy to achieve the 1.6-TB/s per-stack bandwidth with NAND flash
  - e.g., Samsung Z-NAND and Kioxia XL Flash
    - Page size: 2 KiB and 4 KiB
    - Planes per die: 8 and 16
    - Page-read latency: 3 µs and 4 µs
    - Die bandwidth: ~5 GB/s and ~16 GB/s
    - Stack bandwidth: ~80 GB/s and ~256 GB/s (16 dies per stack)
  - Achieving target bandwidth requires 27 planes per die even with 1-µs latency
  - Optimistic modeling of conventional NAND-based HBF
    - 4-KiB page, 16 planes per die, 3-µs latency
      → ~350-GB/s per-stack bandwidth

Methodology: GPU Configurations (Cont'd)

- Five LLM-serving systems w/ different GPU-memory configurations

- HBM4 (HBM-only), HBF and CONV (HBM-HBF), HBF$^+$ and CONV$^+$ (HBF-only)
  - Assume 100-µs program latency (from Z-NAND) for all HBF configurations

GPU	# Stacks (HBM/HBF)	Capacity [GB]	RD BW [TB/s] (HBM/HBF)	WR BW [TB/s] (HBM/HBF)
HBM4	8/0	288	12.8/-	12.8/-
HBF	1/7	3,620	1.6/11.2	1.6/0.112
HBF$^+$	0/8	4,096	-/12.8	-/0.128
CONV	1/7	3,620	1.6/2.45	1.6/0.073
CONV$^+$	0/8	4,096	-/2.80	-/0.084

Methodology: Workloads

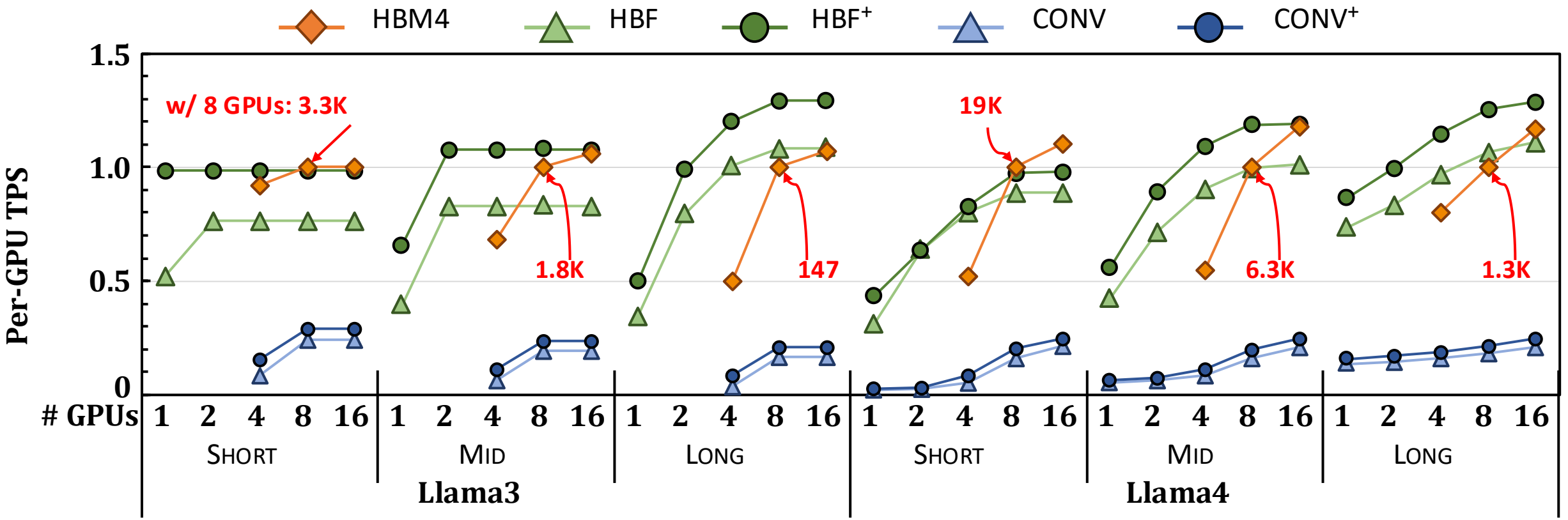
- Target models: Llama 3 405B and Llama 4 Maverick
  - Llama 3 405B (dense): 756-GB weights
  - Llama 4 Maverick (MoE): 746-GB weights

- Three workloads w/ different $\langle L_{IN}, L_{out} \rangle$ based on real-world benchmarks
  - SHORT (ShareGPT): $\langle 1660, 373 \rangle$
  - MID (LongBench): $\langle 5.9K, 499 \rangle$
  - LONG (English summarization): $\langle 103.5K, 1.1K \rangle$

Outline

- Background & Motivation
- Methodology
- **Evaluation Results**
- Conclusion

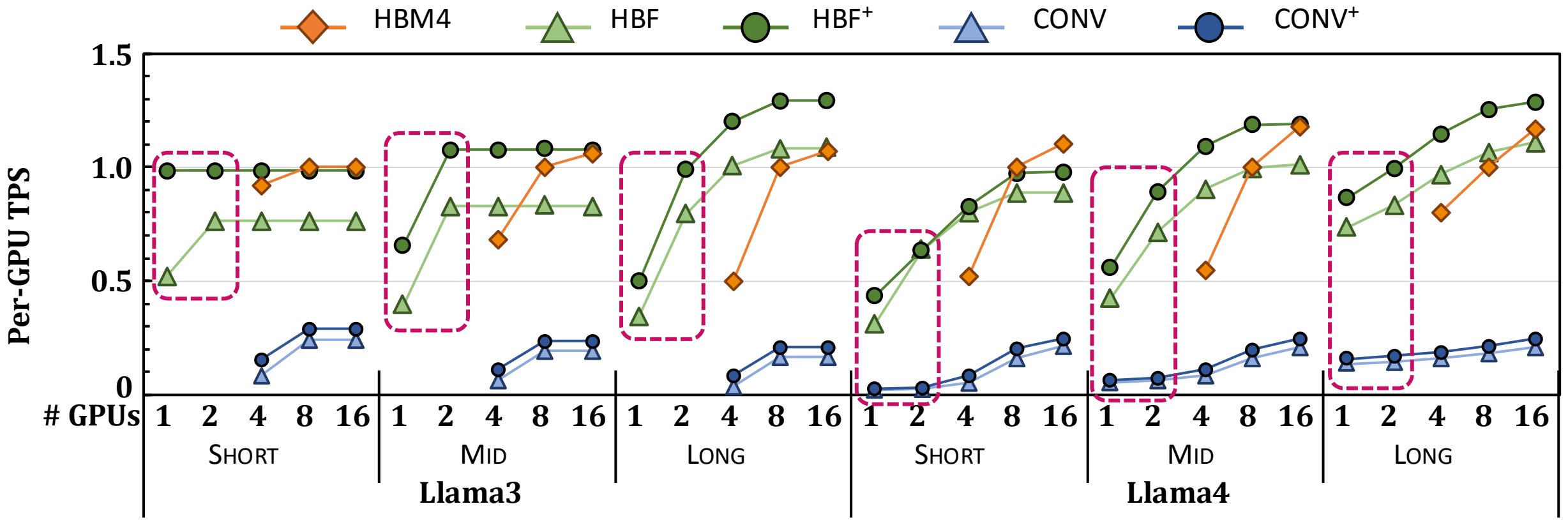
Evaluation Results: System Throughput



Per-GPU TPS under 0.1-second TPOT SLO

Normalized to HBM4 w/ 8 GPUs: Significantly higher TPS in MoE models

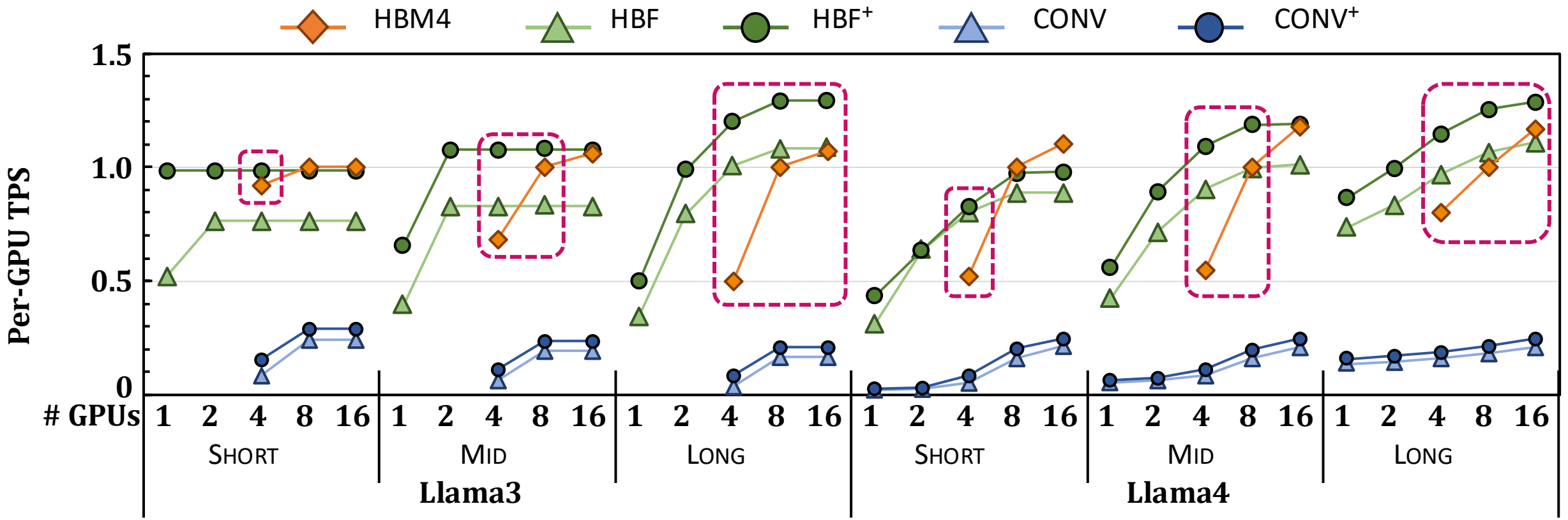
Evaluation Results: System Throughput



Per-GPU TPS under 0.1-second TPOT SLO

Benefit: HBF reduces the minimum GPU count to support the given models

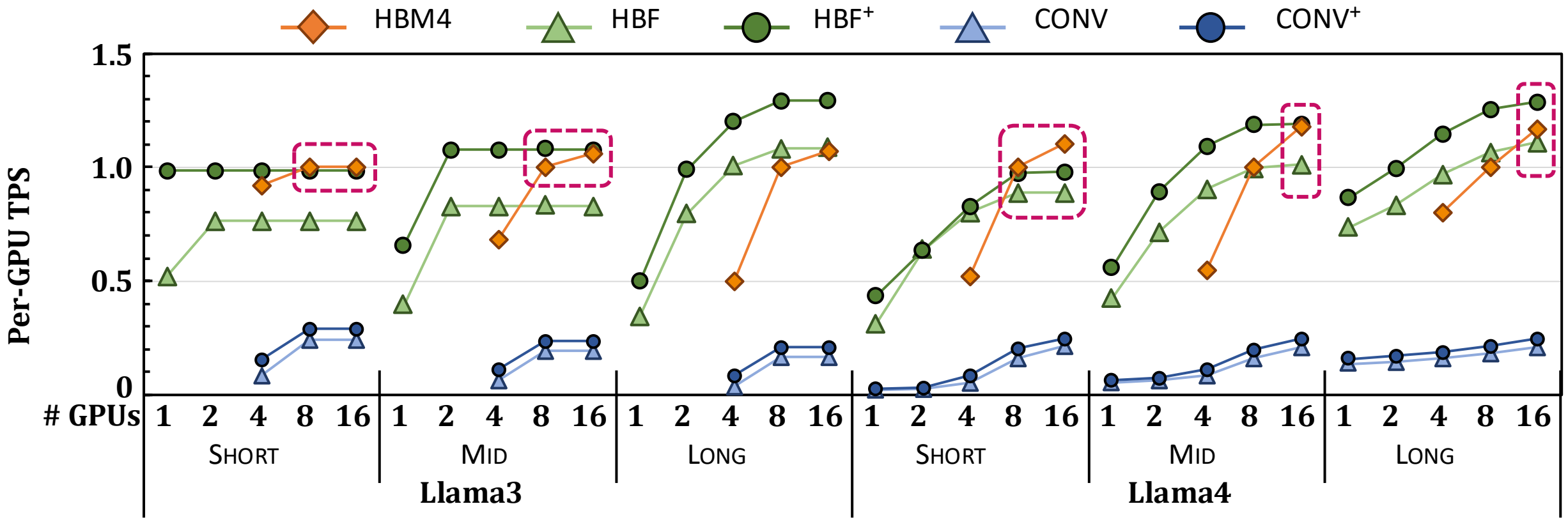
Evaluation Results: System Throughput



Per-GPU TPS under 0.1-second TPOT SLO

Benefit: HBF can improve system throughput at the same GPU count: larger improvements for longer-context workloads

Evaluation Results: System Throughput

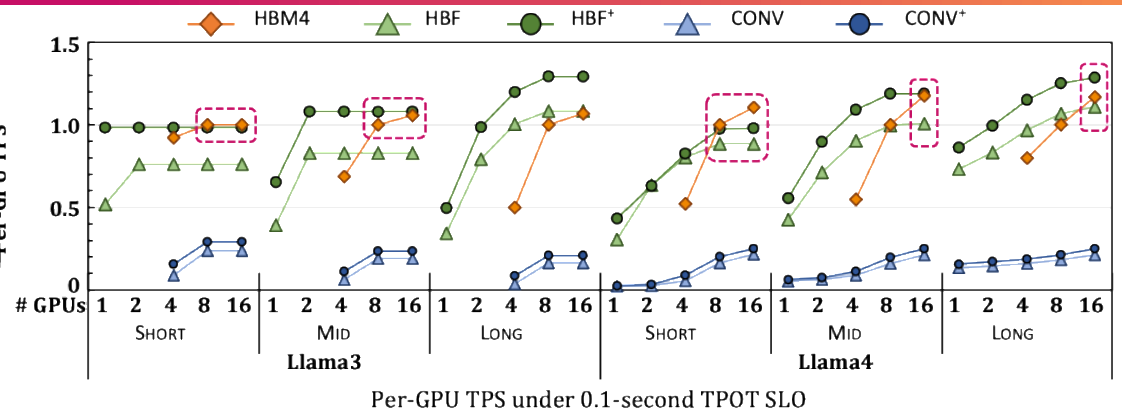


Per-GPU TPS under 0.1-second TPOT SLO

Diminishing improvements as GPU count increases
→ Is HBF effective only for small-scale systems?

Evaluation Results: Diminishing Gains w/ more GPUs

- GPU scaling
  → Larger aggregate capacity & bandwidth
  → Overcome the memory wall (at high cost)

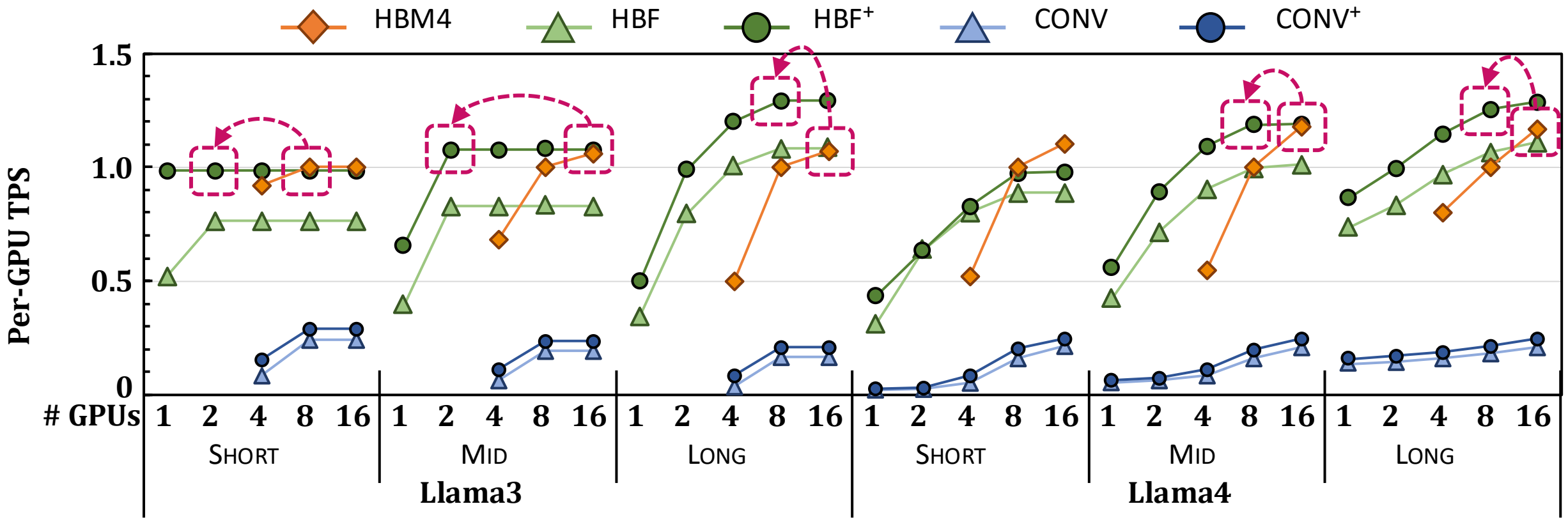


Per-GPU TPS under 0.1-second TPOT SLO

- The benefits of GPU scaling diminish under all evaluated systems
  - Regardless of the memory technology (i.e., also in HBM-based systems)
  - Larger batch → Amortizing more weight accesses (diminishing after a certain point)

- HBF enables achieving such batch sizes w/ fewer GPUs, which can potentially
  - Provide larger throughput gains for larger models (e.g., w/ trillion-parameter LLMs)
  - Allow developing even larger LLMs or leveraging multiple LLMs

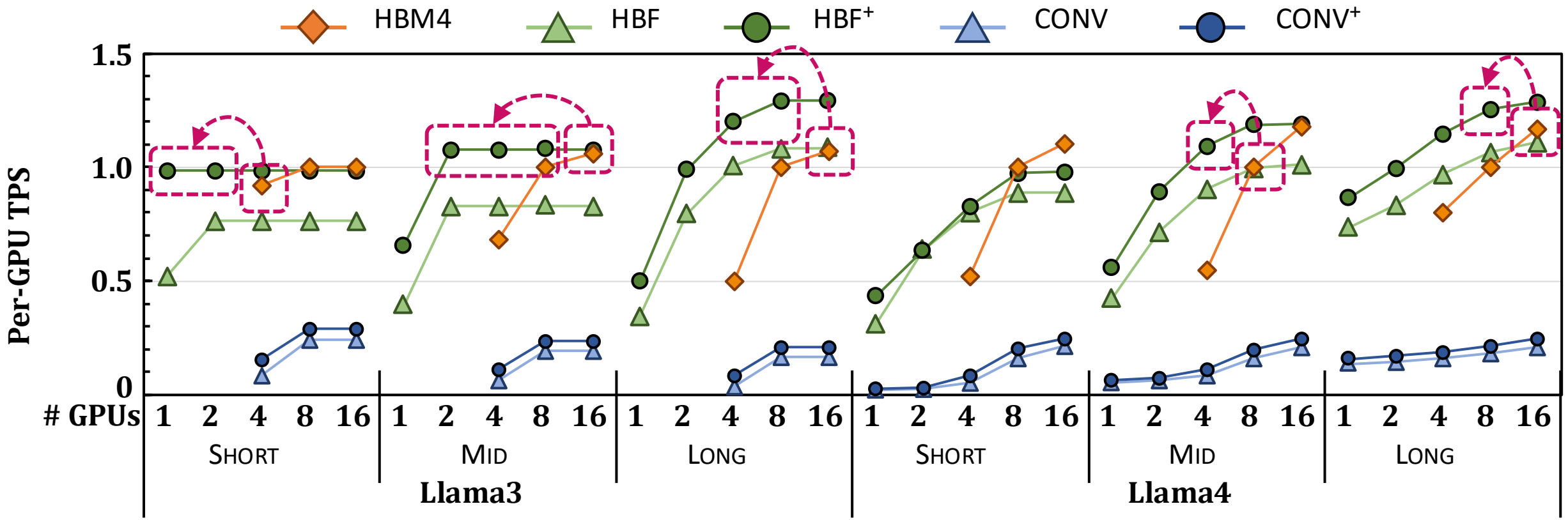
Evaluation Results: System Throughput



Per-GPU TPS under 0.1-second TPOT SLO

Diminishing improvements as GPU count increases?
→ Benefit: HBF enables achieving sufficient batch sizes w/ fewer GPUs

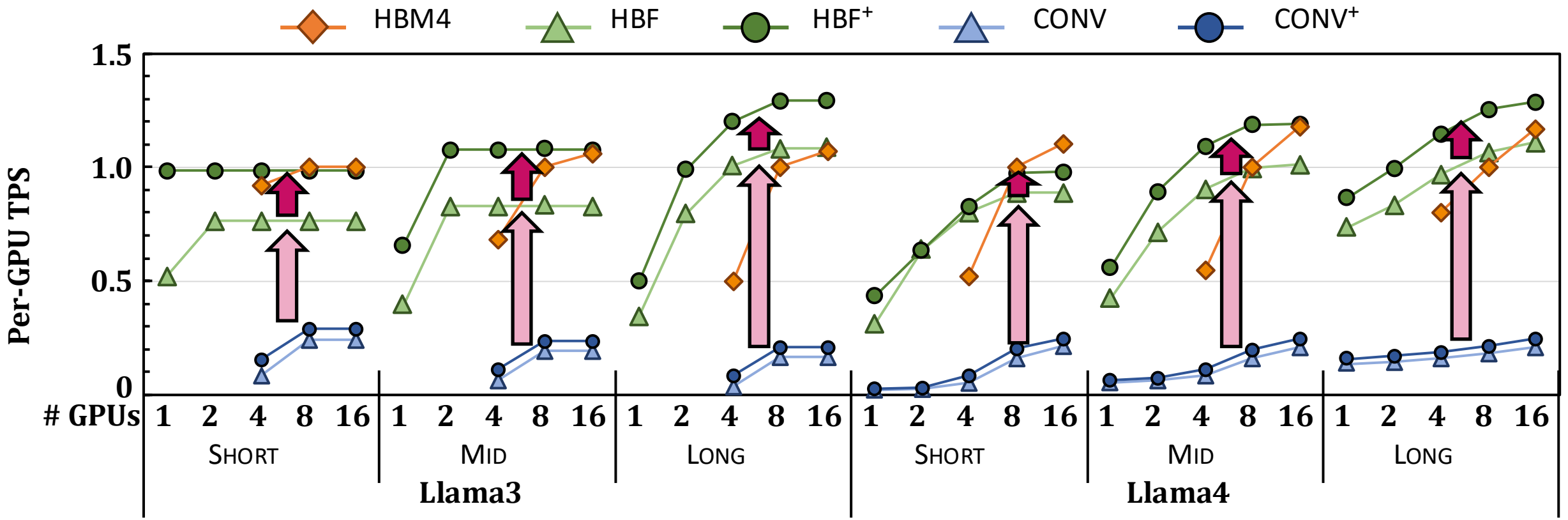
Evaluation Results: System Throughput



Per-GPU TPS under 0.1-second TPOT SLO

HBF can enable high per-GPU TPS even w/ fewer GPUs
→ Benefit: HBF can simultaneously improve performance and flexibility

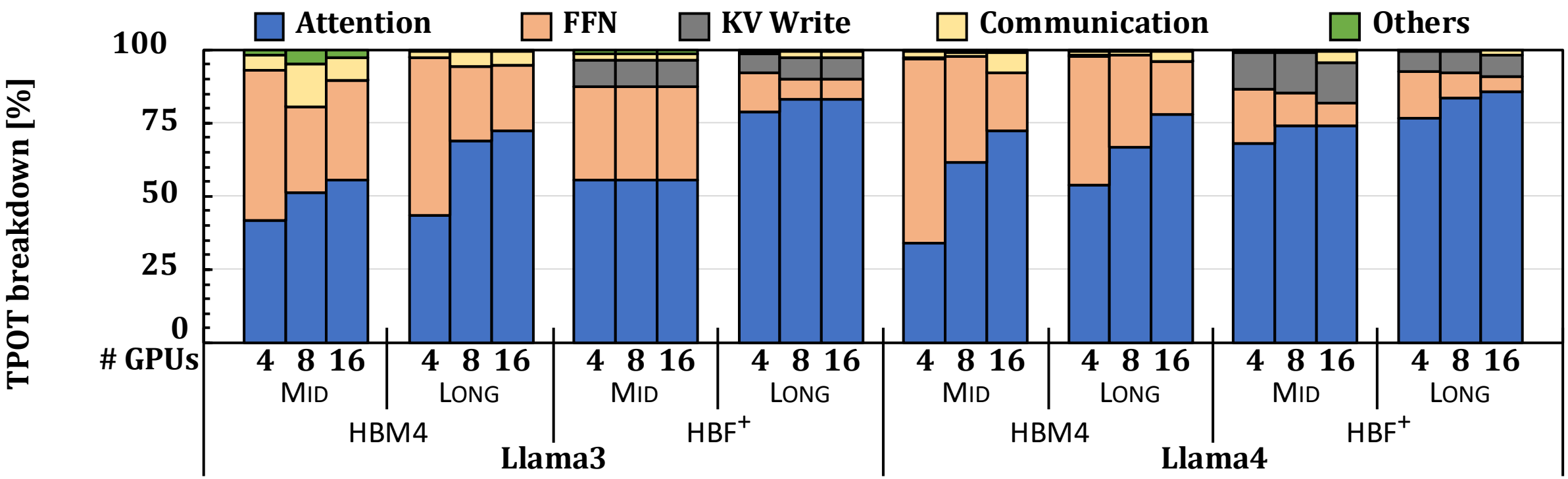
Evaluation Results: System Throughput



Per-GPU TPS under 0.1-second TPOT SLO

HBF+ significantly outperforms other HBF-based systems
→ Requirement: Achieving high read bandwidth is the key to benefits over HBM

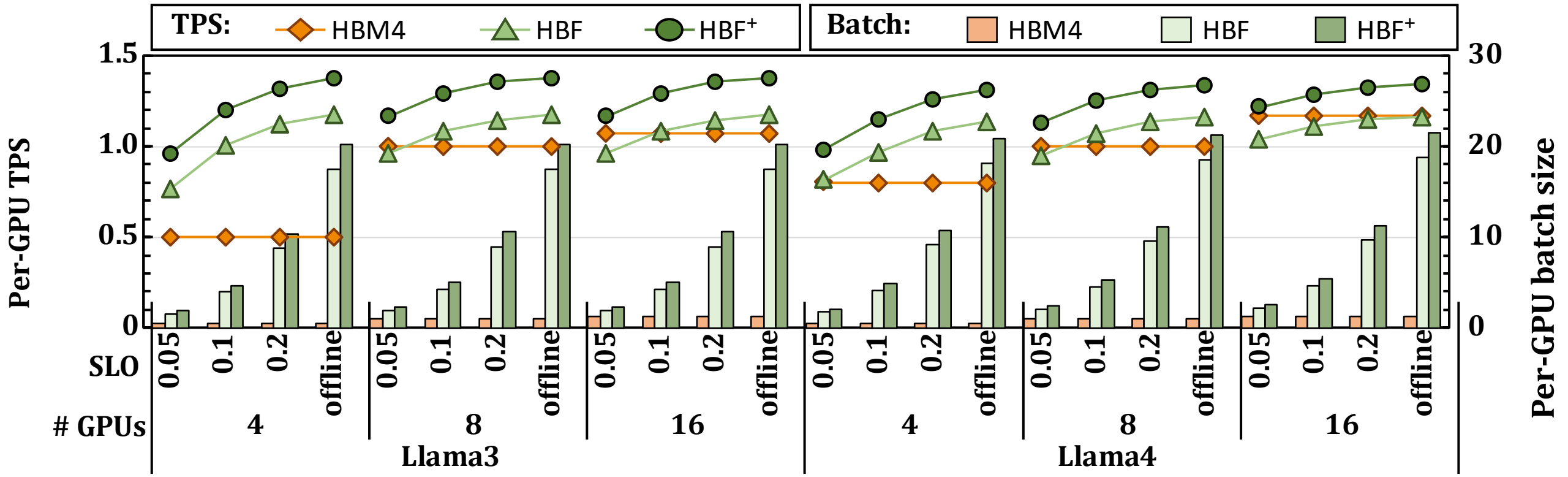
Evaluation: Performance Breakdown



Decoding time breakdown under 0.1-second TPOT SLO

KV-cache writes (from prefill nodes) introduce non-trivial performance overhead
→ Requirement: Reducing tPROG would further improve performance

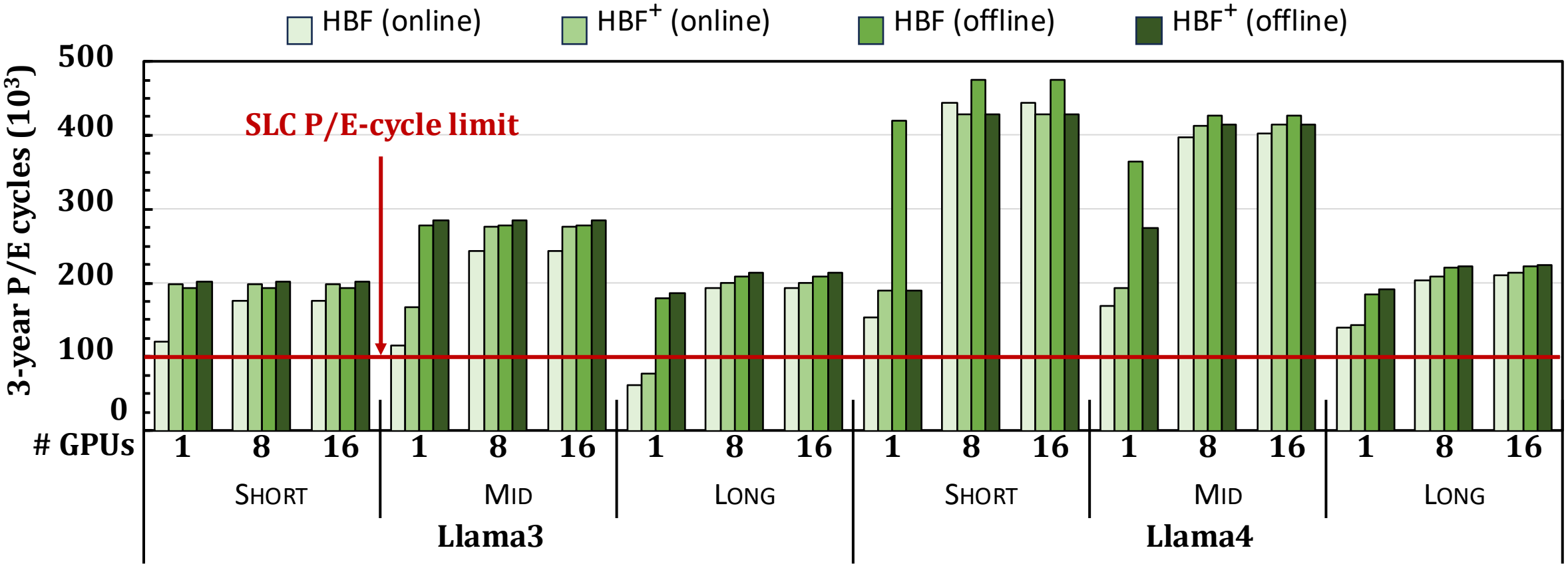
Evaluation: Effect of TPOT SLO Constraint



Performance and batch size impact of TPOT SLO

Benefit: HBF provides larger benefits under relaxed SLO constraints

Evaluation: Write-Traffic Analysis



Required PEC under different operating scenarios

Requirement: HBF requires significant endurance enhancement (2~5x)

Summary: Exploring HBF for LLM-Serving Systems

- Various potential benefits
  - Reducing the minimum GPU count
  - Improving system throughput at the same GPU count, especially
    - Before GPUs are sufficiently scaled for a given model
    - For long-context workloads
    - Under relaxed SLO requirements
  - Improving the performance and flexibility of LLM-serving systems
- Requirements
  - Maximizing read bandwidth
  - Significantly enhancing endurance
  - Improving program speed
  - w/ extensive architecture- and system-level optimizations

Exploring High-Bandwidth Flash for Modern LLM Inference: Opportunities and Challenges

NVRAMOS 2026

Jisung Park

POSTECH CAOS
COMPUTER ARCHITECTURE & OPERATING SYSTEMS LABORATORY

2026.10.22