Hardware-Managed Heterogeneous High-Bandwidth Memory and Flash in LLM Inference Systems

Hakam Atassi *Member, IEEE*, Noa Zilberman *Senior Member, IEEE*, and Amro Awad *Senior Member, IEEE*

***Abstract*—Large Language Models (LLMs) have become the driver of many modern applications, but their parameter counts have outgrown the memory capacity of a single GPU. Because GPUs rely on High-Bandwidth Memory (HBM), whose capacity is approaching a fundamental ceiling, fitting today's models often requires aggregating several accelerators, a cost that places edge deployment out of reach for many users. High-Bandwidth Flash (HBF) offers a denser alternative, providing 16x more capacity per stack at comparable bandwidth. In this work, we show that while replacing HBM with HBF can address the capacity problem, doing so naively severely impacts performance due to HBF's long tail memory latency starving GPU schedulers. To address this, we propose a Heterogeneous Memory Architecture (HMA) that combines HBM and HBF through a prediction-based migration policy to keep high latency HBF off the GPU's critical path. We demonstrate that the HMA achieves geo-mean 2.79x improved performance over an HBF Only baseline, enabling the efficient execution of 9.2x larger models on a single edge GPU.**

***Index Terms*—High-Bandwidth Flash, Heterogeneous Memory Architecture, Large Language Model, Prefetching**

I. Introduction

Large Language Model (LLM) inference has become an important workload in many modern applications. However, the growth in model sizes required to faithfully serve these applications has outpaced the memory capacity of individual accelerators, and their efficient inference is largely bound by the availability of memory bandwidth. While modern accelerators rely on High-Bandwidth Memory (HBM) to meet this demand, current state-of-the-art offerings only provide capacities on the order of 256 GiB at bandwidths of 8 TiB/s [1]. Yet the models leading multi-step reasoning benchmarks exceed these capacities. For instance, GLM-4.7 [2] requires 661 GiB for its FP16 parameters alone. With the scaling of DRAM cell density largely having stalled [3], users dependent on edge deployments are faced with making one of the following choices: sacrifice performance by relying on larger, lower bandwidth memory, sacrifice model accuracy through quantization, or sacrifice data sovereignty by relying on cloud-based deployments.

High-Bandwidth Flash (HBF) is an emerging memory technology well positioned to alleviate these capacity limitations. Where an HBM stack offers capacity on the order of 32-64 GiB at 1 TiB/s, HBF is projected to support capacities of 512-1024 GiB at similar bandwidths [4]. However, HBF inherits a major limitation from its underlying 3D NAND technology: read latencies approximately 1000x higher than DRAM [5].

Hakam Atassi, Noa Zilberman, and Amro Awad are with the University of Oxford.

Study	Memory Modules	Interposer Size	Power & Heat	Capacity	Theoretical BW
H3 [4]	8	↑	↑	∼4 TiB	∼4 TiB/s
Ours	4	↓	↓	∼1 TiB	∼4 TiB/s

TABLE I: Comparison with prior studies

In this work, we show that naively integrating HBF as a drop-in replacement for HBM exposes the GPU to its significantly larger read latency, limiting the number of warps available to schedule. While conventional approaches to mitigating high DRAM latency, such as increasing the number of memory banks, enlarging page size, and deepening the First-Ready First-Come First-Served (FR-FCFS) memory scheduler queue, offer measurable improvements, they do not attain HBM's level of performance, even when composed.

Therefore, we demonstrate that a Heterogeneous Memory Architecture (HMA) relaxes the memory capacity constraint of a single GPU at the cost of only a modest reduction in bandwidth. By migrating model parameters across memory technologies, the HMA improves performance for 230B-397B models by geo-mean 2.79x over an HBF Only integration, mitigating the impact of latency on GPU front-end performance. This positions HBF as a practical memory technology for the design and architecture of high-bandwidth high-capacity memory subsystems in GPUs. As our HMA requires only a single HBF stack, we are able to achieve the aforementioned speedups while concurrently reducing power, cost, and interposer area, as summarized in Table I. The result of the HMA is a single GPU memory subsystem capable of deploying models up to a tested 739 GiB, 9.2x the 80 GiB maximum of an HBM Only A100, while outperforming an HBF Only configuration by a geo-mean of 2.79x.

II. Background

A. LLM Execution

Large Language Models (LLMs) are auto-regressive machine learning algorithms based on the transformer architecture. The inference of the transformer architecture is dominated by General Matrix Multiplies (GEMMs) and General Matrix Vector Multiplies (GEMVs) over the model's entire parameter set, which typically ranges from hundreds to thousands of GiBs. Additionally, the inputs to these GEMM and GEMV kernels are typically relatively small, resulting in comparatively little arithmetic per byte fetched. This low arithmetic intensity makes said kernels heavily memory-bandwidth bound. The memory-bound nature of these kernels, coupled with their capacity demands, is the fundamental reason behind integrating modern accelerators with HBM.

B. Memory Technologies

Modern accelerators rely on HBM to meet the capacity and bandwidth demands of LLM inference. HBM achieves this by stacking DRAM dies vertically and routing their I/Os through Through-Silicon Vias (TSVs), enabling a single stack to provide capacities of 32-64 GiB at bandwidths of 1-2 TiB/s. Such bandwidth is achieved through very close integration of HBM to the compute die, enabling interfaces of 1024-2048 bits at GHz-range frequencies [6]. As a consequence, the aggregate HBM capacity attached to a given accelerator is bounded by the available perimeter, or shoreline, of the logic die. With DRAM cell scaling having slowed [3], and with increases in the number of dies per stack introducing significant yield and thermal challenges, the HBM capacity available to a single accelerator is approaching a fundamental ceiling.

HBF is an emerging memory technology that stacks 3D NAND flash dies into a device that is capable of offering 512-1024 GiB at a projected 1 TiB/s bandwidth per stack [4]. This is achieved through the reliance on the immense memory density of 3D NAND flash, which is largely a result of two main features: vertical 3D NAND integration [7], and multi-level cell (MLC) technology [5]. Together, these technologies enable HBF to pack immense capacity into a single stack, roughly 16x greater than its HBM counterpart [4].

Although HBF's use of 3D NAND enables an order of magnitude larger memory capacities, reads from 3D NAND tend to incur high latencies. While the time for a DRAM device to access a bitline is often on the order of 20-50ns, the latency of accessing a bitline in 3D NAND is on the order of 20µs, with further delays for evaluating and discharging said bitlines for subsequent reads [5], [8]. In the context of GPU memory subsystems, increased HBF access latency manifests as very high page read latency (HBF_{Rd_lat}). NAND flash write latency can be in the hundreds of microseconds but is out of scope for this study given the read intensity of LLM inference.

III. Motivation

The high page read time of HBF poses major challenges in its adoption in GPUs. Namely, while GPUs are typically able to hide memory latency via massive thread-level parallelism, the excessively high HBF_{Rd_lat} exceeds GPUs' latency-hiding ability. Historically, memory latencies have been improved via three primary approaches: allocating additional memory banks, increasing the size of page buffers, and increasing the depth of the FR-FCFS request queue in the memory controller.

Increasing the number of allocated memory banks reduces bank contention and conflicts, with Figure 1(a) showing slowdown decreasing roughly as $1/N$ with bank count, approaching a ∼30x limit beyond 128 banks. Increasing the page buffer size reduces incurred page reads, allowing a single HBF_{Rd_lat} to serve more requests. Figure 1(b) further shows that increasing page buffer size from 2 KiB to 32 KiB reduces slowdown from 207x to 113x, with no visible floor, suggesting larger page buffers remain beneficial beyond the tested range. Figure 1(c) indicates that increasing FR-FCFS queue depth gives the memory controller a more global view of outstanding requests, improving page reuse. However, the impact of FR-FCFS capacity is limited to a floor of ∼155x relative to the baseline, indicating diminishing returns beyond a depth of 128.

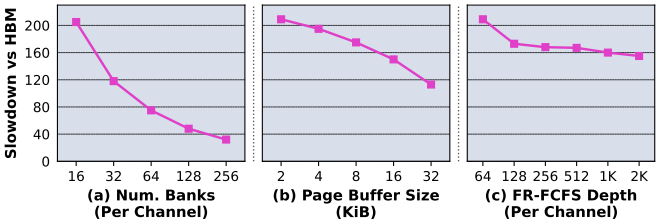

Fig. 1: Slowdown of GLM-4.7 as HBF parameters are swept, normalized to an A100 baseline with unmodified HBM

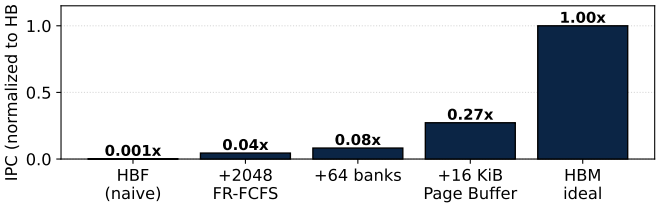

Fig. 2: Impact of composing HBF design parameters on GLM-4.7 throughput

While an HBF memory device with a huge number of banks, a large page buffer, and deep FR-FCFS queue is ideal, these design parameters are constrained by manufacturing limitations. A typical HBM configuration provides 16 banks per channel, a page buffer size of 2 KiB, and an FR-FCFS controller depth of 4 entries per bank (64 entries total) [9]. Due to HBF's sensitivity to these parameters, however, we assume they will be expanded relative to HBM while remaining within reason relative to current work [5]. We therefore assume a realistic HBF device with a 16 KiB page buffer, 64 banks per channel, and 32 FR-FCFS entries per bank. Unfortunately, composing these parameters is unable to recover lost performance, resulting in an instruction throughput of only 27% of an infinite capacity HBM Only ideal, as shown in Figure 2.

Figure 3 compares the scheduler performance of GLM-4.7 on an HBM Only system against an HBF Only system using the composed parameters from Figure 2. Under HBM, 15.4% of scheduler checks issue an instruction during a layer's execution, whereas under HBF this falls to 1.9%, a reduction of more than 8x. Over the same window, the share of checks stalled on read-after-write (RAW) dependencies rises from 64.1% to 89.0%, as the high latency of HBF leaves the schedulers with no ready instructions to issue. This establishes that despite the high maximum HBF bandwidth, the GPU front-end remains stalled, motivating a design that keeps HBF off the GPU's critical execution path.

IV. Design

In order to hide the high latency of HBF reads from the GPU, we design an HMA where HBM and HBF stacks are integrated directly adjacent to the shoreline of the GPU. In so doing, we are able to prefetch data from the HBF to the

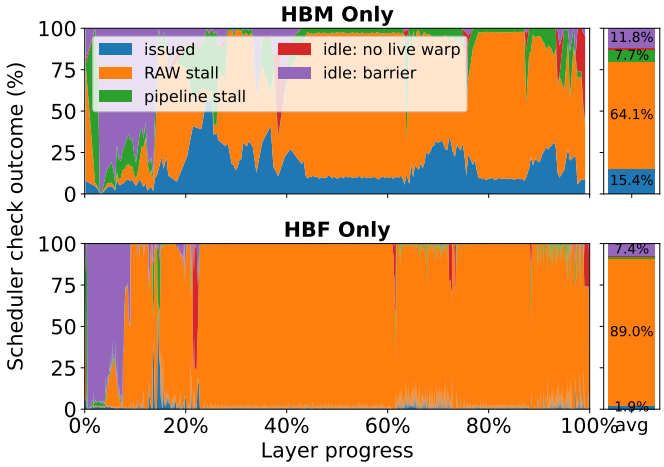

Fig. 3: Scheduler performance of GLM-4.7 on an A100 with HBF Only and HBM Only

HBM, providing lower latency memory access to the GPU, improving performance. Figure 4 depicts our design in detail.

	HBM 64 GiB			HBF 1024 GiB	Total 1088 GiB
Model (params)	Dense	Prefetch	Max tok.	Experts	(GiB)
MiniMax-M2.5 (230B)	7.47	0.42	128K	418.5	426.0
GLM-4.7 (355B)	31.35	0.70	86K	625.8	657.2
Qwen3.5 (397B)	19.00	0.47	264K	720.0	739.5
Qwen3.5 (30B)	56.87	–	78K	–	56.87

TABLE II: Model capacity requirements (FP16). All values in GiB except max tokens. Qwen3.5 (30B) fits entirely within HBM, so allocation is entirely HBM resident.

Memory Layout. Table II describes how memory is allocated in our architecture. In general, dense parameters (embedding, output layer), the KV cache, and activations are allocated on HBM statically. An HBM Expert prefetch buffer is also reserved for use by the Migration Manager.

Migration Manager. The Migration Manager is the component responsible for triggering and monitoring ongoing migrations in the memory subsystem. ❶ In Figure 4, the oracle in the Migration Manager schedules read requests from the HBF. ❷ Then, once the Migration Manager receives a response, it issues a corresponding write to the HBM. ❸ Upon the arrival of a write acknowledgment from HBM, the HBM residency indirection cache is updated with the corresponding page number, valid bit, and cache status, evicting data as needed. Future reads to memory perform a lookup in this indirection cache to determine residence in HBM. Hits, in this case, are routed to HBM, while misses are routed to HBF.

Predictor. We assume the use of an ideal oracle constructed by generating a memory trace of the LLM workloads, and issuing the migrations for layer $i{+}1$ during the execution of layer i. Incomplete migrations stall the execution until all write acknowledgments arrive and the entirety of the subsequent layer has been migrated. Through this, we prevent the predictor from falling behind the current execution (i.e., migrating data that has already been fetched by SMs). To simplify analysis,

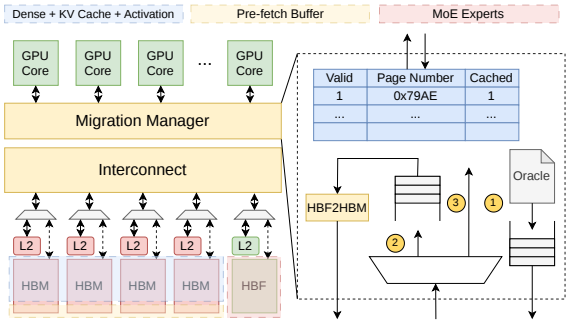

Fig. 4: Migration Manager based HMA design

we assume no reuse of pages, meaning that all page migrations are scheduled even if already available in HBM.

V. Methodology

System Parameter	Value
GPU	NVIDIA A100
Freq	1.41 GHz
SMs	108
PCIe Bandwidth	64 GiB/s
PCIe Latency	1 µs
Host DRAM	1024 GiB
Migration Page Size	2 MiB

Memory Parameter	HBM	HBF
Page Size	2 KiB	16 KiB
Banks	16	64
FR-FCFS Depth	2048 entries	
Channels per configuration		
HBM Only	40	–
HBF Only	–	40
HMA	32	8

TABLE III: Accel-Sim configuration

We perform our experiments in Accel-Sim [9] to simulate an A100 with the configuration in Table III. We collect traces for the HuggingFace [2] models using NVBit on an Ampere A5000 GPU. As this work focuses on single-user edge inference, we simulate a batch size of 1. To bound simulation time and trace sizes, we limit context size to 256 tokens. We report throughput for the decode phase, which is memory-bandwidth bound and most sensitive to the read latency of HBF. To model HBF, we apply recently proposed HBF timing parameters to the Accel-Sim memory model to reflect the latency, parallelism, and read granularity of 3D NAND flash technology [4], [5]. Specifically, we assume HBF_{Rd_lat} of 39,312 cycles.

VI. Evaluation

Figure 5 demonstrates the performance impact our HMA has on various LLMs. Notably, while CPU offloading outperforms direct HBF integration by a geo-mean of 2.05x, the HMA outperforms both direct HBF integration and CPU offloading, providing the highest speed-up of 1.70-3.62x with a geo-mean of 2.79x relative to an HBF Only integration. Moreover, while CPU offloading and HMA configurations exhibit similar computation times, HBF enables much faster migrations, enabling boosted performance.

As both the HBM Only and HMA provide at least 64 GiB of HBM, we measure the impact of small models (Qwen-30B) allocated exclusively on HBM for both configs. We find that HBM Only performs $<$3.5% better than the HMA, indicating that the HMA preserves performance for small models for our tested sequence lengths, as shown in Figure 5.

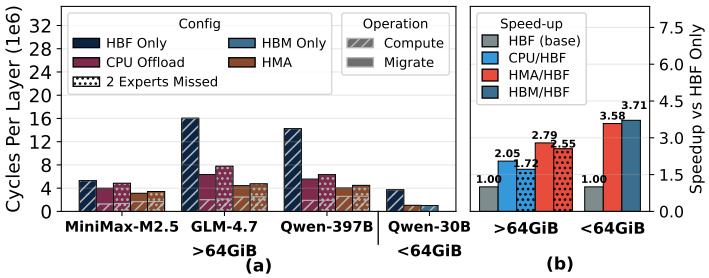

Fig. 5: (a) Per layer LLM performance, and (b) speed-ups

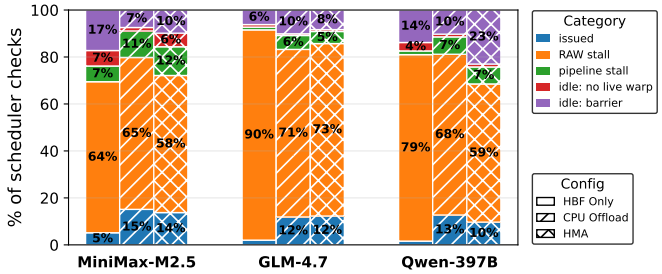

Fig. 6: Breakdown of scheduler outcome across memory configurations

We also measure the impact of mispredicting 2 experts across all models (arithmetic mean misprediction rate of 23.3%), which results in a geo-mean 9.4% overhead in the HMA case and 19.3% in the CPU offloading case (dotted bars in Figure 5), indicating HMA is less sensitive to misprediction than CPU offloading due to increased bandwidth.

Figure 6 reports the average outcome of front-end schedulers during the execution phase of LLM layers. In particular, RAW stalls indicate that the scheduler found at least one warp ready to execute, but every such instruction was blocked by a pending register write. Otherwise, the scheduler fails to issue due to a lack of live warps or barriers. Clearly, the HBF Only configuration exhibits an increased number of stalls due to either barriers or RAW hazards. The HMA and CPU offloading configurations, however, perform similarly from a scheduling perspective, as both rely on HBM for reads, but suffer from varying overhead due to migration, as shown in Figure 5.

The performance gains in Figure 5 can be further justified by the reduction in the latency of memory accesses exposed to the GPU, shown in Figure 7. While the median access latency is similar across configurations at roughly 1µs, the HBF Only system exhibits a heavy tail in which a large fraction of accesses incur the full HBF read latency. The HMA reduces this tail by more than 10x, bringing the 90th-percentile access latency from tens of microseconds to a few microseconds by serving the majority of memory requests from HBM.

VII. Future Directions

Future work may seek to explore the design space of the Migration Manager, evaluating the impact of the page migration granularity, migration scheduling/ordering, or exposure to pre-existing structures (such as translation look-aside buffers/TLBs). Other directions include novel GPU front-end architectures that are able to tolerate HBF-scale memory

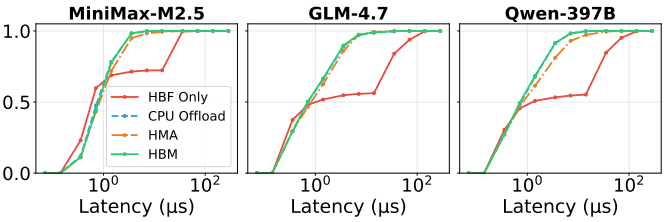

Fig. 7: Memory access latency CDF across configurations

latencies, and improved prefetching techniques to minimize high-latency memory accesses.

VIII. Conclusion

The execution of LLMs is constrained by the capacity-bandwidth limitations of current memory technologies. While emerging technologies such as HBF offer 16x greater capacity at comparable bandwidth, their 1000x higher page open time starves the GPU schedulers and severely degrades performance. In this work, we contribute a hardware-managed HMA that horizontally integrates HBM and HBF along the GPU shoreline and employs a prediction-based migration policy to keep HBF off the GPU's critical path. By migrating parameters from HBF to HBM, the HMA recovers a substantial portion of LLM performance, achieving a geo-mean speed-up of 2.79x over an HBF Only integration, primarily by reducing the memory access tail latency. These results establish hardware-managed HBM/HBF heterogeneity as a viable approach to relaxing the capacity-bandwidth constraints of modern GPU memory subsystems, enabling the execution of 9.2x larger models on a single GPU relative to an HBM Only A100.

Acknowledgment

This work was funded by the Advanced Research + Invention Agency (ARIA) through project code NACB-PR01-P04.

References

[1] NVIDIA Corporation, "NVIDIA Vera Rubin NVL72: Rack-scale agentic AI supercomputer," Product Page, 2026, accessed: 2026-06-22. [Online]. Available: https://www.nvidia.com/en-gb/data-center/vera-rubin-nvl72/

[2] Hugging Face, "Hugging face – the AI community building the future," https://huggingface.co/, 2024, accessed: 2026-06-01.

[3] M. Patel, T. Shahroodi, A. Manglik, A. Giray Yağlıkçı, A. Olgun, H. Luo, and O. Mutlu, "Rethinking the producer-consumer relationship in modern dram-based systems," *IEEE Access*, vol. 12, pp. 196 207–196 239, 2024.

[4] M. Ha, E. Kim, and H. Kim, "H3: Hybrid architecture using high bandwidth memory and high bandwidth flash for cost-efficient llm inference," *IEEE CAL*, pp. 1–4, 2026.

[5] J. Park, M. Kim, M. Chun, L. Orosa, J. Kim, and O. Mutlu, "Reducing solid-state drive read latency by optimizing read-retry," in *Proceedings of the 26th ACM ASPLOS*, 2021, pp. 702–716.

[6] K. Asifuzzaman, M. Abuelala, M. Hassan, and F. J. Cazorla, "Demystifying the characteristics of high bandwidth memory for real-time systems," in *2021 IEEE/ACM (ICCAD)*, 2021, pp. 1–9.

[7] H. Tanaka, M. Kido, K. Yahashi, M. Oomura, R. Katsumata, M. Kito, Y. Fukuzumi, M. Sato, Y. Nagata, Y. Matsuoka *et al.*, "Bit cost scalable technology with punch and plug process for ultra high density flash memory," in *IEEE Symposium on VLSI Technology*, 2007, pp. 14–15.

[8] Y. Kim, V. Seshadri, D. Lee, J. Liu, and O. Mutlu, "A case for exploiting subarray-level parallelism (salp) in dram," in *Proceedings of the 39th ISCA*, ser. ISCA '12. USA: IEEE Computer Society, 2012, p. 368–379.

[9] M. Khairy, Z. Shen, T. M. Aamodt, and T. G. Rogers, "Accel-sim: An extensible simulation framework for validated gpu modeling," in *2020 ACM/IEEE 47th (ISCA)*, 2020, pp. 473–486.