Hardware-based Heterogeneous Memory Management for Large Language Model Inference

Soojin Hwang
KAIST
Republic of Korea
sjhwang@casys.kaist.ac.kr

Jungwoo Kim
Stanford University
California, USA
jungwkim@stanford.edu

Sanghyeon Lee
KAIST
Republic of Korea
leesh6796@casys.kaist.ac.kr

Hongbeen Kim
KAIST
Republic of Korea
hbkim@casys.kaist.ac.kr

Jaehyuk Huh
KAIST
Republic of Korea
jhhuh@kaist.ac.kr

Abstract

A large language model (LLM) is one of the most important emerging machine learning applications nowadays. However, due to its huge model size and runtime increase of the memory footprint, LLM inferences suffer from the lack of memory capacity in conventional systems consisting of multiple GPUs with a modest amount of high bandwidth memory. Moreover, since LLM contains many bandwidth-intensive kernels, only focusing on the memory capacity without considering the bandwidth incurs a serious performance degradation. To handle such conflicting memory capacity and bandwidth demands in a cost-effective way, this study investigates the potential of heterogeneous memory systems, proposing H2M2. It uses an asymmetric memory architecture consisting of capacity-centric and bandwidth-centric memory with computation units attached to each memory device. With the asymmetric memory, we first analyze the effect of kernel-memory mapping for the asymmetric memory. Second, we propose a dynamic runtime algorithm that finds a mapping solution considering the characteristics of LLM operations and the change of footprint during LLM inference. Third, we advocate the need for memory abstraction for the efficient management of the asymmetric memory. H2M2 outperforms the conventional homogeneous memory system with LPDDR by 1.46×, 1.55×, and 2.94× speedup in GPT3-175B, Chinchilla-70B, and Llama2-70B, respectively.

Keywords: Large Language Models (LLMs), Heterogeneous memory system, Hardware accelerators

1 Introduction

Recently, decoder-only transformer-based large language models (LLMs) are widely used for their simple structure and powerful performance [5, 7, 40, 46, 51]. However, the characteristics of LLMs make it challenging to accelerate their computation with conventional systems: First, the large sizes of model parameters and activations require significant memory capacity and the infrequent tensor reuse leads to low locality, demanding high memory bandwidth. Second, decoder-based LLMs store the KV (Key-Value) cache to prevent repetitive computation, which expands in size with each token generation. Unlike the prior ML inferences, this leads to significant dynamic changes in memory footprint during inference runtime. Third, the opportunity for batching is limited, as the attention layer cannot be batched with multiple requests. Furthermore, the attention layer of a token generation phase contains numerous memory intensive GEMV (GEneral Matrix-Vector multiplication) kernels.

To address the substantial memory footprint and high bandwidth requirement, multi-GPU systems are extensively used for LLM acceleration. However, the limited locality of memory accesses presents difficulties in fully utilizing the high computation power of GPUs, because multiple GPUs are needed to store weights and KV cache in relative small HBMs (High Bandwidth Memory). The capacity requirement of LLMs increases LLM service costs significantly with many GPUs necessary to store the weights and KV cache in a distributed way. Furthermore, applying the model/tensor parallelism to multi-GPU systems incurs considerable communication and synchronization overheads [2, 4].

To alleviate the memory capacity challenge of LLMs, recent studies used the host memory as the second-level memory for the KV cache and/or weights [42, 53]. However, these approaches incur significant performance degradation since they need to access CPU-side memory through limited PCIe bandwidth. A recent alternative HW-based solution integrates capacity-centric memory combined with a custom accelerator chip [36]. The approach showed that an accelerator with a large capacity of LPDDR memory can outperform multi-GPU systems in LLM performance for non-batched operations, as using many GPUs for larger models increases communication and synchronization overheads [36]. Although this approach accommodates the capacity demand of LLM, the low bandwidth of LPDDR memory still limits the performance of LLM computation.

Two major memory demands of LLMs, *large capacity* and *high bandwidth* must be satisfied to improve the performance by overcoming the limitation of prior approaches. To reach

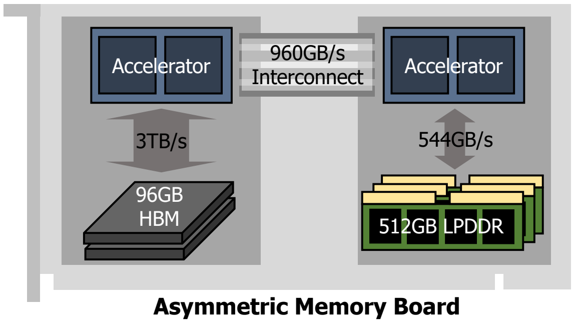


Figure 1. Accelerator substrate used by H2M2.

the goal, this paper investigates a heterogeneous memory architecture with the capacity-centric memory module and the bandwidth-centric memory module. However, the strict hierarchical organization of conventional heterogeneous memory which uses the high bandwidth memory as the first level memory backed by the low bandwidth memory is not suitable for LLMs, as the low locality of LLMs makes the performance bounded by the low bandwidth of capacity-centric memory.

In this paper, we employ an *asymmetric memory* architecture for LLM acceleration combining the bandwidth-centric and capacity-centric memory modules in a parallel way, as supported in a recent Grace Hopper architecture [32]. The asymmetric memory adds accelerators to both of the bandwidth-centric and capacity-centric memory modules. This study investigates how the mapping of data and computation of LLMs can properly exploit the potential of the asymmetric memory architecture and what hardware supports are needed to hide the complexity of asymmetric memory. We explore mapping techniques such as kernel fusion and layer split to the asymmetric memory, and analyze patterns of the best *Kernel-Memory Mapping* to unleash the potential of performance.

This paper proposes a dynamic mapping algorithm for accelerator-based asymmetric memory system with hardware support for memory abstraction, called H2M2. Based on the best mapping analysis, we propose a runtime policy for the kernel-memory mapping, which can be achieved by solving a simple linear problem. It can adjust the memory mapping effectively when the KV cache size dynamically changes for different batch sizes and token lengths. In addition, we propose an efficient dynamic memory management for asymmetric memory to address the KV cache memory allocation problem raised by a recent study [24], and to provide a unified memory view under dynamic changes of the memory mapping.

We evaluate the performance of H2M2 using a cycle-accurate simulator for generation phases of three LLMs: GPT3-175B, Chinchilla-70B, and Llama2-70B. H2M2 outperforms the capacity-centric memory architecture with the same number of computation units by 1.46×, 1.55×, and 2.94× speedup for GPT3-175B, Chinchilla-70B, and Llama2-70B, respectively. The hardware support for memory abstraction of H2M2 incurs up to only 1.36% performance overhead in three LLM workloads. The kernel-memory mapping based on the greedy mapping policy of H2M2 shows less than 5% additional degradation compared to the optimal kernel-memory mapping strategy in all three LLM workloads, which is negligible for the overall performance gain from asymmetric memory architecture.

The contribution of this paper are as follows:

- This study applies an asymmetric memory architecture to LLM acceleration, and analyzes the effect of kernel-memory mapping policies to find the best one.
- It proposes a near-optimal runtime algorithm under dynamically changing KV cache sizes.
- It proposes an efficient memory abstraction scheme to address the challenges for the KV cache size and mapping changes.

2 Background

2.1 Large Language Models

Language modeling entails predicting the forthcoming sequence of output tokens (typically words or subwords) from given a set of input tokens. Recently, transformer-based generative language models have been extensively explored owing to their inherent scalability and self-attention mechanisms [47]. These transformer-based language models with a significant number of parameters are commonly known as Large Language Models (LLMs) [52].

Decoder-based LLM: Recent LLMs are based on on a *transformer decoder layer* that generates subsequent tokens autoregressively (*generation phase*), after processing the prompt with multiple tokens (*prompt phase*) [3, 5, 7, 34, 40, 45, 46]. These LLMs are constructed with multiple decoder layers, each sharing the same topology in terms of tensor dimensions and operations, albeit with different parameter values. Figure 2 illustrates the decoder layer topology for the autoregressive generation phase in the GPT3 model, which dominates inference execution time [5]. Key hyperparameters include H, the dimension of each attention head, N, the number of heads, D, the dimension of the bottleneck layer following multi-head attention, and O, the dimension of the feed-forward layer. The sequence length S represents the total length of the prompt and previously generated tokens.

As illustrated in Figure 2, the decoder in the generation phase includes green-colored batching-compatible operations and pink-colored batching-incompatible operations. Batching-compatible operations are computed as GEneral Matrix-Matrix multiplication (GEMM) kernels when the batch size is greater than one, or as GEneral Matrix-Vector multiplication (GEMV) kernels for a batch size of one. In contrast, batching-incompatible operations are always computed as GEMV kernels, regardless of batch size. In the rest

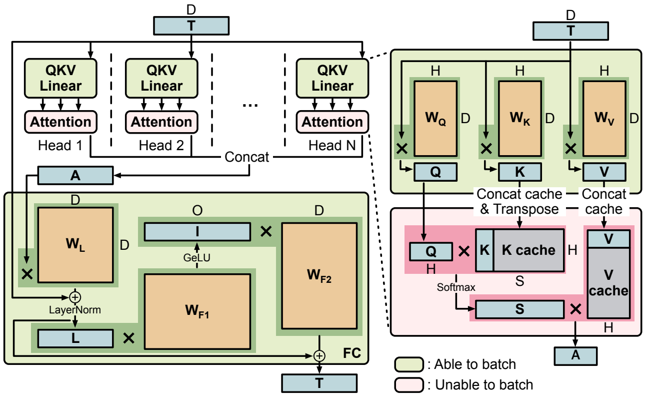


Figure 2. The topology of the decoder layer in GPT3. Orange boxes with W labels represent the weight parameter tensors, blue boxes represent the input activation tensors, and gray boxes represent the KV cache tensors.

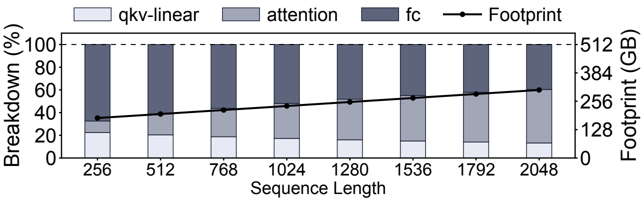


Figure 3. Footprint breakdown for batch size 32, sequence length growing from 256 to 2048 in GPT3-175B. The portion of *attention* increases due to the increase of KV cache size.

of this paper, we classify these decoder kernels into three groups: *qkv-linear*, *attention*, and *fc* layers. During the computation of transformer decoder, the embedding vector T is first passed to *qkv-linear* sublayer to generate LLM contexts: Query (Q), key (K), and value (V). Next, *attention* sublayer computes attention value A using Q, K, and V. This sublayer utilizes not only the contexts of the current input token, but also the contexts of prior tokens - called as KV cache. Lastly, *fc* sublayer computes the final output, and this sublayer includes projection and feed-forward network. While the weight - activation multiplication kernels of *qkv-linear* and *fc* can be batched with multiple requests, it is usually infeasible to batch KV cache - activation multiplication kernels in *attention*. This is because of the nature of the KV cache; Due to the difference of user requirements between requests, requests in a batch rarely have common values in the KV cache, not allowing the batched computation.

2.2 Challenges of Accelerating LLM Inference

2.2.1 Large Footprint and Limited Locality. Modern LLMs often require hundreds of gigabytes [5, 7, 40, 51]. Moreover, the rarity of data repetition among kernels and layers leads to strict dependencies among operations, as described in Figure 2. The considerable memory footprint and strict dependencies in LLMs contribute to excessively long data reuse distances, which limits temporal locality. For instance, in GPT3-175B with FP16 precision, the reuse distance of weight parameters is at least 350GB. Several prior works have introduced the methodology of increasing inter-kernel temporal locality through batching, but their limited data reuse still results in the lack of temporal locality for the overall model [24, 49].

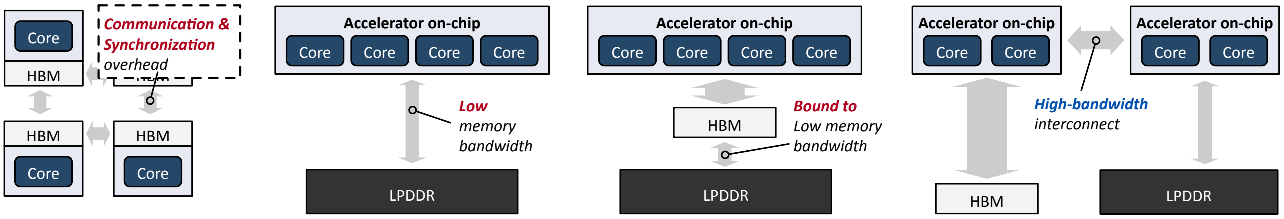
2.2.2 Unique Characteristics of KV Cache. As depicted in Figure 2, the *attention* key and value tensors are preserved in memory for future reuse, forming what is known as *KV cache* [37]. KV cache continuously expands throughout the auto-regressive generation phase until the generation of the end-of-sequence (eos) token. Therefore, as shown in the Figure 3, the model footprint dynamically changes during inference, resulting in significant storage overhead. Thus, accelerating LLM inference requires a unique approach distinct from traditional DNN models. For example, vLLM introduced efficient memory management for key and value tensors within *attention* layers by employing virtual memory and paging techniques with software supports [24]. However, vLLM presupposed that GPU device memory is large enough to store all weight parameters, still facing the limitation when running large models.

2.2.3 Limitation on Batching. GEMV serves as a basic building block of the decoder due to batching-incompatible operations. However, its arithmetic intensity (i.e. # of operations per memory traffic) is $O(1)$, whereas GEMM has an arithmetic intensity of $O(n)$. Due to limited data reuse, GPUs handle GEMV less efficiently than GEMM, making it necessary to convert GEMV into GEMM through batching to fully utilize GPU computation units. To address this, Orca selectively batches compatible operators at the iteration level [49]. However, batching requests within *attention* layers is nearly impossible, as these layers do not have weight parameters and compute each request independently using inputs such as query, key, and value. Prior studies have proposed integrating GEMV-optimized processing-in-memory (PIM) technology to efficiently accelerate attention layers [13, 35, 41]. However, PIM architectures are limited by their small memory capacity due to the inclusion of computation units inside memory banks.

2.3 Memory Systems for LLMs

Figure 4 illustrates four types of memory systems applicable to LLM inference acceleration. The bandwidth-centric and the capacity-centric memory modules are represented as HBM and LPDDR, general solutions for modern AI acceleration [25, 32, 36].

Multiple Bandwidth-Centric Memory: Modern GPU frameworks for LLMs typically configure a system resembling Figure 4 (a) by employing model parallelism across multiple GPUs, each equipped with tens of gigabytes of HBMs [24, 49]. Nevertheless, such systems may encounter significant communication overhead from frequent synchronization among



(a) Multiple Bandwidth-Centric Memory (b) Capacity-Centric Memory (c) Strict Hierarchical Memory (d) Asymmetric Memory

Figure 4. Four possible configurations of memory systems applicable for LLM inference acceleration.

devices [2, 4]. Moreover, although incorporating hundreds of gigabytes of HBM into a single module would satisfy both memory demands of capacity and bandwidth for LLMs, this approach is heavily constrained by cost factors and the nature of HBM architecture [20, 36].

Capacity-Centric Memory: Another feasible solution is depicted by Figure 4 (b), configuring the system entirely with capacity-centric memory. This configuration provides scalability in capacity without incurring communication overhead. CXL-PNM suggested employing cost-effective LPDDR5X memory due to its advantageous balance among memory bandwidth, capacity, and power consumption [36]. However, the capacity-centric memory can suffer from limited memory bandwidth compared to the bandwidth-centric memory.

Strict Hierarchical Memory: Traditional heterogeneous memory systems often assume a *strict hierarchical memory configuration*, where computation units are exclusively attached to the bandwidth-centric memory, as illustrated in Figure 4 (c). However, this approach requires data migration for every access to capacity-centric memory, resulting in power and latency overheads for workloads with limited locality [19].

Asymmetric Memory: To overcome the limitation of hierarchical memory, a heterogeneous memory system without a hierarchy between memory modules presents a viable solution. As illustrated in Figure 4 (d), computation units (i.e. accelerators) are connected to both bandwidth-centric and capacity-centric memory in this system. For example, NVIDIA's Grace Hopper Superchip and Grace Blackwell Superchip feature an architecture where a GPU with HBM and a CPU with LPDDR are connected via a high-speed interconnect [32, 33]. By providing equal computational power to both memory devices, the system eliminates the necessity for frequent data migration between memory modules or direct access for computation. In the following sections, we refer to this configuration as an *asymmetric memory*.

2.4 Host Memory Offloading Methods for LLMs

Several approaches have explored using host memory, a capacity-centric memory with relatively long access latency, for GPU-based ML computation [19, 26, 31, 39, 42, 44]. DeepPlan uses a pipeline of load and execution, replacing some data migration with direct host-to-GPU memory access [19]. FlexGen extends this by utilizing both the computational power and memory capacity of the host CPU [42]. While the large capacity of host CPU memory meets capacity demands, its low bandwidth and the limited bandwidth and high latency of PCIe interconnects significantly restrict overall performance. Consequently, system performance largely depends on which operation's data is offloaded to the host memory. For instance, FlexGen determines the placement of weight parameters, KV cache, and activation vectors by solving a linear optimization problem for the system with GPU, CPU, and disk [42]:

$$\begin{array}{lrcl}
\min\limits_{p} & & \mathit{Execution\ Time} & \\
\text{s.t.} & \mathit{gpu\ peak\ memory} & < & \mathit{gpu\ mem\ capacity} \\
& \mathit{cpu\ peak\ memory} & < & \mathit{cpu\ mem\ capacity} \\
& \mathit{disk\ peak\ memory} & < & \mathit{disk\ mem\ capacity} \\
& w_g + w_c + w_d & = & 1 \\
& c_g + c_c + c_d & = & 1 \\
& h_g + h_c + h_d & = & 1
\end{array} \tag{1}$$

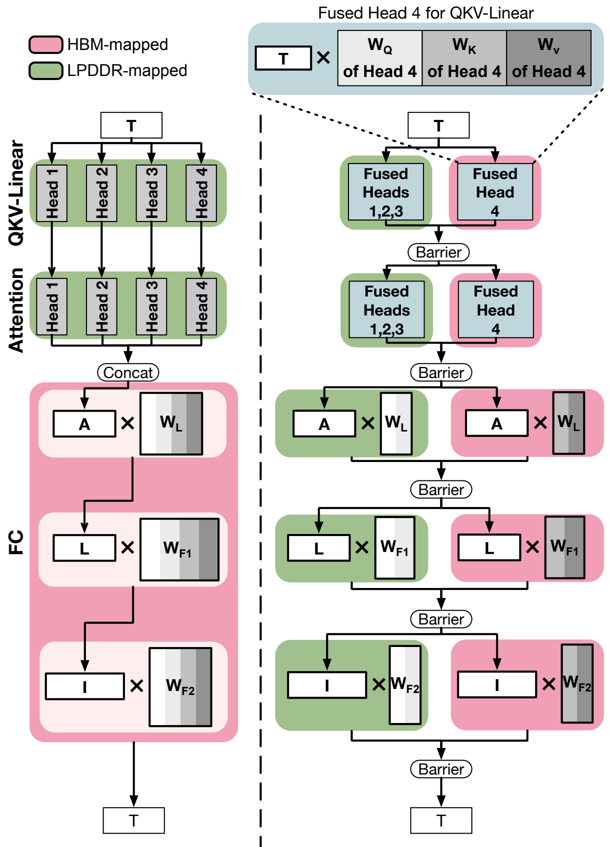
FlexGen addresses the challenge of efficiently utilizing host memory by optimizing the placement of data across GPU, CPU, and disk storage. The placement p is defined by nine variables: (w_g, w_c, w_d) for weights, (h_g, h_c, h_d) for activations, and (c_g, c_c, c_d) for KV cache. These variables are relaxed to real values between 0 and 1 in the cost model, simplifying the optimization process. The problem is solved by adjusting these placement variables to minimize the objective function while adhering to memory constraints. This linear programming approach, outlined in Eq. (1), balances memory usage and performance demands and can be extended to include additional considerations such as latency or compression techniques.

3 Analysis of Mapping Space

Serving LLMs imposes significant demands on bandwidth and memory capacity. Section 2.3 examines four potential strategies, three of which exhibit limitations in practicality or efficiency. To address these challenges, this paper proposes an asymmetric memory approach as an effective solution.

3.1 Granularity of Kernel-Memory Mapping

To minimize redundant data migration, all LLM data should be mapped to asymmetric memory modules before each generation phase iteration. Figure 5(a) shows a naive sublayer-granular mapping approach with model parallelism and KV



(a) Naïve **(b) Head-aware Mapping**

Figure 5. Techniques for supporting optimal mapping granularity with asymmetric memory, for a single decoder layer. For clarity in explanation, a simplified view of a decoder layer is presented, visualizing GEMM/GEMV kernels mainly.

cache parallelism, where decoder layers consist of *qkv-linear*, *attention*, and *fc*. As HBM-mapped operations must wait for LPDDR-mapped operations to complete due to strict dependencies, parallelization is not available and the accelerator cores are under-utilized.

To enhance LLM computation efficiency, various parallelization and kernel fusion techniques have been proposed [2, 8, 12, 21, 22, 27, 28, 42, 43]. Building on these techniques, we propose *head-aware mapping granularity*, an advanced kernel-memory mapping granularity for asymmetric memory systems, as depicted in Figure 5(b). In *head-aware mapping granularity*, each sublayer is split into two parallel partitions and mapped to the HBM side and the LPDDR side, to maximize the utilization of both accelerator chips.

As shown in Figure 5 (b), *qkv-linear* and *attention* are partitioned at the head granularity due to their independent heads. In contrast, the *fc*, lacking a head-based structure, partitions GEMM kernels into two. To prevent partial sum accumulation, weight matrices are split column-wise, and activation matrices are fully copied into both memory modules.

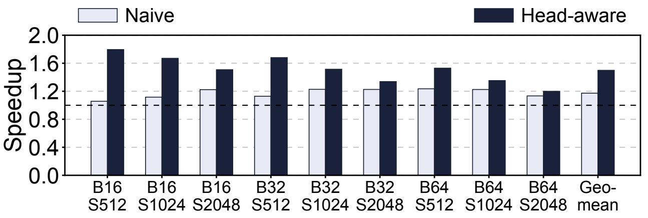


Figure 6. Comparison of the performance of the asymmetric memory with two mapping granularity options in GPT3-175B.

A synchronization barrier ensures computational correctness after each kernel. Furthermore, kernels from multiple heads mapped to the same memory module can be fused into a single kernel call, as depicted in the upper part of Figure 5(b). This fusion increases matrix sizes for efficient blocked GEMM, improves on-chip memory utilization, and reduces kernel launch overhead.

Figure 6 compares the speedup of asymmetric memory with two mapping granularities, normalized to the baseline: LPDDR-only homogeneous memory (Figure 4(b)) with sublayer-granular mapping (Figure 5(a)). The x-axis represents batch size (B) and sequence length (S) (e.g., *B16 S512* indicates a batch size of 16 and a sequence length of 512). The *head-aware granularity* achieves a 1.50× speedup, outperforming the 1.27× speedup of the naïve granularity. This underscores the effectiveness of the proposed *head-aware mapping granularity*, which is adopted as a foundational assumption for all asymmetric memory variants discussed in the paper. Note that the performance of both options is evaluated using their optimal kernel-memory mapping, defined as the configuration yielding the best performance.

The key concepts of the proposed head-aware mapping granularity are (1) overlapping the execution time of two sides, (2) reducing the capacity consumption by distributing model parameters and KV cache exclusively, and (3) reducing the synchronization overhead by exploiting the parallelism of heads. Therefore, the head-aware mapping granularity is applicable to other variants of LLMs as long as there exists an independency between kernels inside a sublayer for most sublayers (e.g. 'head' and 'expert' of mixture-of-expert (MoE) models).

3.2 Limitation of Mapping Policy from Host Memory Offloading

Another important feature that makes significant impact on the LLM inference performance in asymmetric memory is, *kernel-memory mapping* decision. That is, the number and type of kernels mapped to each side of the asymmetric memory decides the overall performance of the system. Prior work, FlexGen proposed an analytical performance model-based mapping strategy to guarantee performance in systems with host memory offloading [42], which is described in Equation 1.

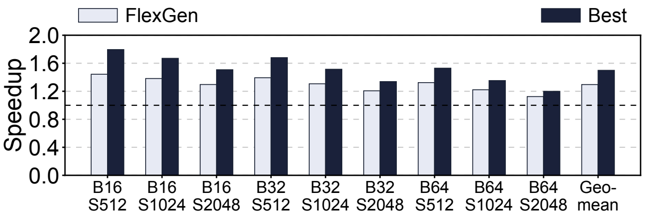


Figure 7. The relative speedup over the baseline with two mapping policies of asymmetric memory in GPT3-175B, batch size 32. `FlexGen` follows the mapping found by the model of Equation 1, and `Best` follows the mapping decision found by exhaustive search with N^3 times profiling.

Figure 7 compares two mapping strategies: Mapping decision from FlexGen's performance model (Equation 1, denoted as `FlexGen`), and the optimal mapping determined through N^3 times profiling and exhaustive search (`Best`). The x axis represents the batch size and sequence length pair, and the y axis is a normalized speedup over the baseline. We modified FlexGen's performance model illustrated in Equation 1 to suit asymmetric memory systems instead of single-GPU systems. As shown in Figure 7, while `Best` shows 1.50× speedup over the baseline on average, `FlexGen` reports 1.30× speedup on average - 0.87× of `Best`.

The mapping decisions of the FlexGen model are suboptimal in the asymmetric memory system for two key reasons. First, since FlexGen is designed for offline inference, it does not account for changes in batch size or sequence length, leading to notable performance degradation in our asymmetric memory system due to static mapping. Second, its performance model fails to account for the distinct characteristics of the three sublayer types. As explained in Section 2.4, FlexGen model's partitioning scheme is based on three groups: Weight, activation, and KV cache. Therefore, *qkv-linear* and *fc* are considered as a same group, partitioned within the same ratio as well as ignoring their different characteristics such as dimensions of weight matrices, number of GEMM kernels, parallelism between heads, and existence of the barrier. Moreover, as the FlexGen model only considers the total capacity and FLOP assigned to each side of memory device, the high memory intensiveness of *attention* is ignored. Because of this, in particular, mapping *attention* to LPDDR causes significant performance degradation, as the slowdown of memory-intensive GEMV kernels in *attention* affects the entire system. In addition, the mapping of `FlexGen` remains static when batch sizes and sequence lengths change, which might be efficient for the offline inference that FlexGen was originally designed for, but not efficient for the scenario with dynamic changes of batch size and sequence lengths. Note that the growth of batch size and sequence length incurs the narrower gap between `FlexGen` and `Best` due to the decrease of relative portion of HBM in total footprint: As the relative portion of HBM in total memory consumption of the system gets smaller, the performance of the systems with both mapping gets closer to the baseline (i.e. LPDDR-only system), resulting in the smaller gap between two mapping strategies.

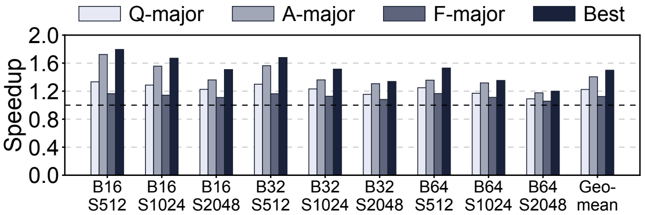


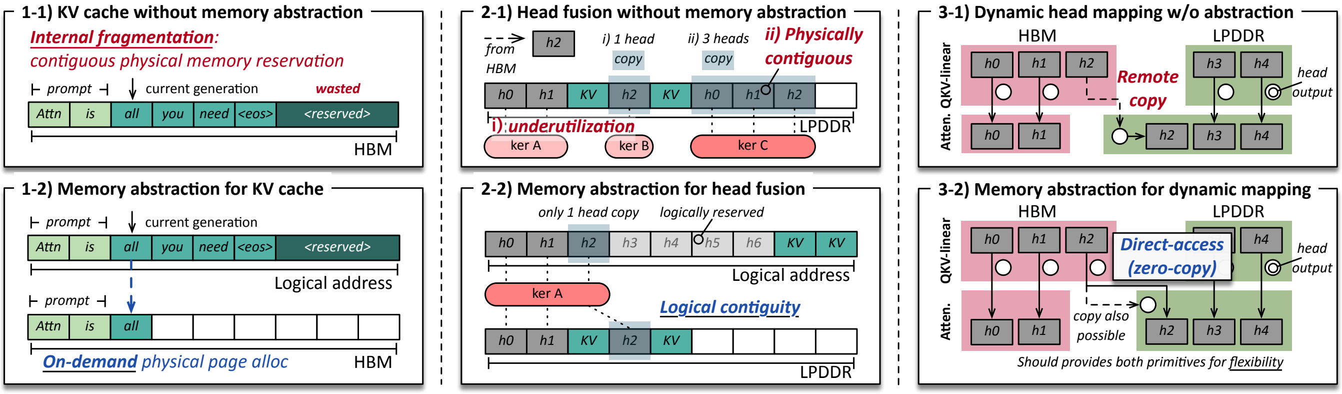
Figure 8. The relative speedup over the baseline for asymmetric memory variants in GPT3-175B, batch size 32. `Q-major`, `A-major` and `F-major` follows the mapping decision found by exhaustive search among result of N^2 times profiling, each favoring HBM for *qkv-linear*, *attention*, and *fc*, respectively. `Best` follows the mapping decision found by exhaustive search with N^3 times profiling.

Although using FlexGen's performance model to determine a mapping for asymmetric memory is unsuitable, finding the optimal mapping through exhaustive search is also impractical due to its high profiling cost. Assuming the number of attention heads is N, the search space is N^3 (*qkv-linear* × *attention* × *fc* decisions), amounting to 884,736 options for the GPT3-175B model with 96 heads. Profiling such a vast space for each LLM inference iteration introduces significant overhead, emphasizing the need for a new mapping policy. In the following, we quantitatively analyze the correlation between kernel-memory mapping decisions and performance.

3.3 Importance of Sublayer Characteristic Consideration

Figure 8 compares the performance of the best mapping (`Best`) with three alternative mappings with smaller search space. Each of `Q-major`, `A-major`, and `F-major` means an alternative best mapping decision with keeping as much *qkv-linear*, *attention*, and *fc* in HBM as possible, respectively (i.e. Requiring N^2 profiling followed by exhaustive search each). The performance is measured as a speedup over the baseline, for each batch size and sequence pair. While `Best` shows the performance 1.50× faster than the baseline, `Q-major` and `F-major` only reaches 1.22× and 1.12× speedup over the baseline, respectively. On the other hand, `A-major` shows notable performance improvement compared to other two options: 1.40× speedup over the baseline, 0.94× of `Best`. Such difference is originated by distinct characteristics of three sub-layers: Each sublayer of LLM have a notable difference in the correlation between the available memory bandwidth and their performance.

Importance of Attention Sublayer Mapping: Since GEMV kernels of *attention* makes it highly bandwidth-bounded, the performance of *attention* with HBM shows higher performance boost over LPDDR than other two sublayers. In addition, the impact of *attention* on overall performance is



(a) Growing KV cache

(b) Mapping changes

(c) Flexible memory accesses

Figure 9. The need for dynamic memory management in asymmetric memory systems: three motivations for memory abstraction to decouple the logical address space from the physical address space.

proportional to the batch size and sequence length, as these values are directly related to the size of KV cache. This is the reason why A-major consistently showed good performance in Figure 8.

Other Sublayers: Compared to the *attention* layer, *qkv-linear* and *fc* share similar characteristics. While the *attention* layer maintains a nearly constant arithmetic intensity, the arithmetic intensity of *qkv-linear* and *fc* increases with the batch size. Additionally, their weight parameters and activations remain stable regardless of sequence length, and their overall footprint varies minimally with batch size, leading to a relatively consistent memory demand. As the LLM footprint grows due to KV cache expansion, the contribution of *qkv-linear* and *fc* to overall performance decreases. This trend explains the consistently suboptimal performance of Q-major and F-major, as shown in Figure 8. These characteristics emphasize the need for carefully tailored kernel-memory mapping strategies for each sublayer type.

3.4 Need for Dynamic Memory Management

Along with the unique characteristics of LLM operations and fine-grained kernel-memory mapping, the distinct traits of memory modules in an asymmetric memory introduce significant challenges for deploying LLMs. Efficient LLM execution requires dynamic memory management to handle growing KV caches, adapt to runtime changes in kernel-memory mapping, and optimize data placement. Figure 9 presents three common challenges in using asymmetric memory for LLMs and demonstrates how our dynamic memory management resolves them. To avoid underutilization and migration overhead from direct physical memory allocation, we propose decoupling logical and physical address spaces. This approach enables flexible memory access and abstraction, ensuring effective use of both memory modules and accelerators, ultimately maximizing LLM performance on asymmetric memory systems.

(a) Growing KV cache: Since the KV cache grows with more generated tokens, using physical memory addresses directly requires pre-allocating contiguous memory to accommodate the maximum sequence length for each request. However, as shown in Figure 9 (1-1), much of this reserved memory is wasted, as not all requests reach the maximum sequence length. This inefficiency severely impacts asymmetric memory systems, where HBM's limited memory space is crucial for performance. vLLM previously addressed this issue by introducing a software-based virtualization layer for the KV cache, allowing more flexible memory allocation [24].

Building on this concept, as shown in Figure 9 (1-2), we propose a hardware-integrated memory virtualization approach tailored for asymmetric memory systems. Unlike vLLM's software-based solution, our method dynamically manages memory at the hardware level, enabling the KV cache to be allocated contiguously in logical address space while assigning physical memory only as needed for generated tokens [38]. This eliminates the complexity of software-based translation while optimizing memory utilization. Moreover, our method is applicable for all tensors and all sublayers of LLM, while PagedAttention approach of vLLM is designed only for KV cache (i.e. only applicable for *attention* sublayer).

(b) Mapping changes: In asymmetric memory systems with the fine-grained kernel-memory mapping proposed in Section 3.1, the best mapping may change at runtime. For instance, a tensor composed of three heads (*h0*, *h1*, *h2*) might initially be partitioned between LPDDR (*h0*, *h1*) and HBM (*h2*). Figure 9 (2-1) describes a scenario where the entire tensor needs to be stored in LPDDR due to a runtime change in the best mapping, requiring *h2* to be moved from HBM to LPDDR. In such cases, the tensor must be stored in a contiguous region in LPDDR to prevent performance degradation caused by dividing a matrix operation into multiple kernels.

Without memory abstraction, transferring *h2* requires allocating a new physically contiguous region and copying all three heads to the new location. Such data movement leads to significant under-utilization of LPDDR, as each head is stored in LPDDR twice. As illustrated in Figure 9 (2-2), we address this under-utilization problem by dynamically managing asymmetric memory with memory abstraction. The proposed approach enables each kernel to access data through a contiguous logical memory region, even when physical pages are scattered across the memory.

(c) Flexible memory accesses: When using fine-grained mapping in asymmetric memory systems, the output of a layer may be computed across accelerators attached to two different memory modules. For the next layer, each accelerator may require access to the output stored in the other side. Figure 9 (3) describes two scenarios where the accelerator on the LPDDR side requires *h2*, which is stored on the HBM side. Depending on the degree of data reuse, the data can either be copied to a different memory module (Figure 9 (3-1)) or direct-accessed remotely (Figure 9 (3-2)). To enable such dynamic memory management, memory abstraction provides a flexible memory access mechanism, allowing kernels to operate within a logical address space without requiring modifications to the kernel code.

4 Design

4.1 Overview

This paper proposes a dynamic mapping algorithm and a hardware support for memory abstraction to maximize the potential of asymmetric memory for LLM processing. Figure 10 shows the overall organization of our system H2M2.

Hardware Substrate: In our system, the asymmetric memory consists of two different memory modules: bandwidth-centric HBM3 with 3TB/s bandwidth, and capacity-centric LPDDR5X with 544GB/s bandwidth [32]. On each side of the asymmetric memory, there is an accelerator chip processing kernel computations of LLMs. Each accelerator chip is comprised of four cores, and all accelerator units in each core share the on-chip scratchpad memory (SPM) with double buffering mechanism. The memory modules and accelerators are placed in a single board with PCIe interconnect to the host. Figure 11 illustrates the architecture of each accelerator chip. Considering various types of operations in LLM, the accelerator core contains four types of accelerator units: matrix-matrix (MM) unit for GEMMs, matrix-vector (MV) unit for GEMVs, vector unit for layer normalization and residual connections, and special function unit (SFU) for activation functions. Section 5.1 explains the detailed hardware parameters.

Memory abstraction support: Instead of using physical memory directly, H2M2 provides memory abstraction to address the KV cache allocation problem and the dynamically changing kernel-memory mapping (Section 4.2). Each accelerator chip in H2M2 includes an MMU for address translation to support memory abstraction. A simple flat page table for each side is maintained by the host driver. The page tables in the HBM and LPDDR sides may have different contents, as the same virtual page containing weights can be duplicated in both HBM and LPDDR.

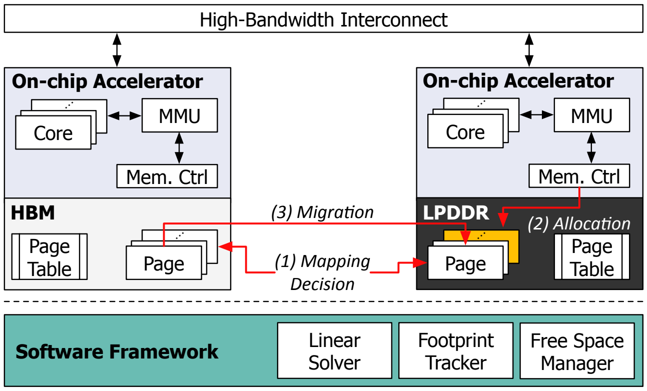


Figure 10. Overview of H2M2 with hardware substrates.

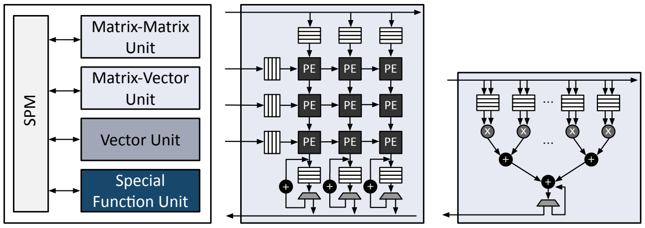
Dynamic mapping support: The memory mapping can change for two major reasons. First, the KV cache can grow with increasing sequence lengths, requiring the addition of a new physical page. Second, a request can complete earlier than the other requests in the same batch, prompting the addition of a new request to the batch. This can shrink or expand the KV cache size, leading to a change in the mapping decision (Section 4.3). When such a mapping change occurs, the host driver updates both page tables and invalidates the TLBs in the MMUs.

Kernel synchronization: As shown in Figure 5, kernels executed in two memory sides often need to be synchronized before advancing to the next operation. To optimize synchronization efficiency while reducing kernel launch overhead, we adopt a hardware-based synchronization mechanism similar to the CUDA event for NVIDIA GPUs [30]. The H2M2 driver initiates all necessary LLM kernels on both accelerators, managing dependencies via the H2M2 hardware controller rather than explicit host-side synchronization. Kernels register their dependencies upon launch and are temporarily removed from the scheduling pool, to be reintroduced when their prerequisites are complete. This approach minimizes kernel launch overhead and accelerator idle time, thereby enhancing overall synchronization efficiency.

4.2 Dynamic Memory Management

4.2.1 Page-based Memory Virtualization.

We apply traditional page-based virtualization for the asymmetric memory system, to support dynamic memory management through memory abstraction. Two page tables are maintained for the HBM and LPDDR sides, which are maintained by the host driver. Each accelerator has an MMU with 2048 TLB entries, referring to the prior work [16, 17]. To reduce the complexity and latency of page table walks, we use a simple flat page table. Unlike conventional CPUs, the logical address space is



(a) Accelerator Core (b) Matrix-Matrix Unit (c) Matrix-Vector Unit

Figure 11. Architecture of the accelerator core: (a) Overview, (b) Matrix-Matrix multiplication unit, and (c) Matrix-Vector multiplication unit. An accelerator chip of each memory module includes 4 cores.

not huge for the LLM accelerator, as the entire capacity is bounded by the sum of HBM and LPDDR capacity plus the extra KV cache space for maximum sequence length. Therefore, instead of employing a radix-tree page table, a flat table allows a single memory access for a TLB miss. Assuming 1TB of logical address space and 2MB page size, the page table size is only 4MB.

Page Size: Considering the overhead of address translation, our system employs a **2MB huge page** for our system. However, the use of large page sizes may present a risk of internal fragmentation. Hence, we investigate the potential data fragmentation using GPT3-175B with a batch size of 32, to estimate the impact of internal fragmentation. The size of internal fragmentation for each type of tensor is calculated using Equation 2.

$$fragmentation = ((tensor\_size)\ mod\ (page\_size)) \times (\#\ tensors) \tag{2}$$

The term *tensor_size* denotes the minimum size of consecutive data in each tensor, which is consistently mapped to the same memory module (either bandwidth-centric or capacity-centric) and cannot be merged with other tensors. By accumulating the sizes of internal fragmentation calculated using Equation 2, we find that the total maximum internal fragmentation amounts to 156MB for GPT3-175B, which occupy only 0.16% of the HBM capacity. Based on the analysis, a 2MB page size is sufficient for current system and target workloads.

4.2.2 HW/SW Support for the Change of Mapping.

While the batch size and sequence length of LLMs can change at runtime, such changes appear only at the end of each iteration in the generation phase, meaning they occur infrequently. As illustrated in Figure 10, three types of events can occur at the end of each iteration: (1) mapping decision, (2) allocation, and (3) migration. These events are executed under the control or assistance of software frameworks (i.e., *linear solver*, *footprint tracker*, *free space manager*) with support from the hardware components described in the hardware substrate. Note that due to the hardware support and relatively long intervals between batch size or sequence length update, these events have minimal impact on overall system performance.

Algorithm 1 Mapping Decision Algorithm

```
def initialize_mapping($N$: Number of heads):
    for sublayer in [attention, qkv-linear, fc]:
        Find $n$ such that
        • Possible to map $n$ heads of the sublayer to HBM
        • Possible to map $(N - n)$ of the sublayer to LPDDR
        • Minimize the peak execution time on both
          HBM and LPDDR sides
```

Mapping Decision: *Linear solver* is the software implementation of the mapping algorithm explained in Section 4.3. The mapping decision is updated when the *linear solver* determines that a change is required based on the information from the *footprint tracker*. In such cases, the subsequent allocation and migration processes are triggered, resulting in changes to the kernel-memory mapping.

Allocation: Physical page allocation is required for newly generated tokens or additional requests. The location of newly allocated physical pages is determined by the *free space manager*, and the allocation mechanism also updates the information of MMUs.

Migration: When the mapping changes, pages previously mapped to HBM need to be migrated to LPDDR, or vice versa. In this process, the MMU's information is updated accordingly, and the allocation mechanism is applied if a new page allocation is required at the destination.

4.3 Runtime Dynamic Mapping Decision

4.3.1 Requirements of Mapping Decision Strategy.

As explained in Section 3, an effective kernel-memory mapping can significantly boost performance in asymmetric memory systems. However, finding the *best-mapping* through exhaustive search and profiling is impractical, as head-aware mapping in asymmetric memory involves $N \times N \times N$ choices, where N is the number of heads per each decoder. To maximize performance in asymmetric memory, the mapping strategy must satisfy three key conditions. First, the mapping decision should consider the **runtime change of performance trend** caused by changes in batch size and the sequence length. Second, the mapping decision should **reflect the characteristics of each sublayer**. As discussed in Section 3.3, differences in arithmetic intensity and memory footprint imply that the impact of the mapping decision on overall performance varies across sublayers. Lastly, the **cost** of the mapping decision process should be **low**. Even if a mapping decision is optimal, excessive decision-making costs (e.g., requiring extensive profiling) can negate the performance gains by introducing significant overhead.

Arch	Capacity	Bandwidth
HBM3	96GB	3TB/s
LPDDR5X	512GB	544GB/s
Interconnect	-	960GB/s

Table 1. Heterogeneous memory system configuration, following the configuration of [32].

4.3.2 Mapping Algorithm. Algorithm 1 shows our mechanism for runtime dynamic mapping decision, satisfying all three conditions explained above. Algorithm 1 aims to maximize the performance of both HBM and LPDDR sides without encountering out-of-memory issues. Since *attention* requires more HBM capacity than *qkv-linear*, we prioritize mapping optimization in the order of *attention*, *qkv-linear*, and *fc* as shown in line 2. The first and second bullet of line 3 aim to prevent out-of-memory issues. For the third bullet of line 3, we employ the min-max algorithm to ensure balanced execution times across both memory sides for each sublayer. To reflect the characteristics of each sublayer, the peak execution model first calculates the *ideal execution time* by dividing the total number of arithmetic operations by the maximum throughput of the accelerator chip. Next, a hyperparameter that reflects the arithmetic intensity is multiplied to the *ideal execution time* for additional elaboration. The cost of solving Algorithm 1 with single-threaded C++ implementation is 0.05ms with Intel i7-6700 CPU, which is nearly negligible compared to the cost of LLM inference. Compared to the exhaustive search-based mapping strategies (i.e. mappings of Figure 8, our mapping algorithm does not requires profiling in $O(N^2)$ or $O(N^3)$ space. Instead, it only needs to solve simple polynomials to find the nearly-optimal mapping, which leads to the low temporal overheads.

Basically, Algorithm 1 operates as a greedy algorithm that prioritizes HBM allocation in the order of *attention*, *qkv-linear*, and *fc* sublayers. Even if the batch size and sequence length change at runtime, the mapping decisions determined by this algorithm remain relatively stable, as long as the changes do not cause significant change in HBM utilization. Therefore, even if a new (i.e. changed) mapping decision necessitates altering the existing mapping, the amount of data migration required is relatively small. For instance, in typical LLM inference scenarios where sequence length continually increases with token generation, eviction from HBM to LPDDR occurs in order of *fc*, *qkv-linear*, and *attention*. By following the Algorithm 1, once a layer is evicted, there is no need to bring it back into HBM in this scenario. This minimizes a redundant data migration during runtime.

5 Evaluation

5.1 Methodology

Architecture Configuration: Table 1 shows the major parameters of two memory modules. The interconnect refers to the connection between accelerators in two memory modules. We adopt the memory configuration of NVIDIA Grace Hopper Superchip for the heterogeneous memory configuration [32]. Table 2 defines the detailed hardware configuration of the accelerator units. We adopt the systolic array configuration of Google Cloud TPU for the MM units [11]. We refer to the vector unit configuration of DFX [15], but scale up the number of computation units considering the size of the systolic array.

MV Unit	
PE Configuration	32 × 1D Array (128 × 1)
Algorithm	Dot product
MM Unit	
PE Configuration	Systolic Array (128 × 128)
Dataflow	Weight stationary
SFU & Vector Unit	
ADD/SUB/MUL/DIV	1D Array (128 × 1)
Adder Tree	128 Adders
Lookup Table	128 req/cycle
Common	
Core Frequency	1GHz
On-chip SPM Size	(16MB × 2) per core
HBM Access Latency	32ns
LPDDR Access Latency	45ns
TLB Miss Latency	300ns

Table 2. Accelerator parameter configuration.

Baseline: We adopt the capacity-centric memory system in the prior work as a baseline [36]. All comparison configurations, including the baseline system configuration, consist of two accelerator chips that provide the same computational power with an asymmetric memory system.

Simulation: We model homogeneous and heterogeneous memory systems by developing a cycle-level performance simulator, cross-validated by profiling the open-source multi-core NPU simulator and DRAM simulator [16, 29]. We apply the parameters defined in Table 1 and 2 for the simulation. We collect memory access latency data from Ghose et al. [10].

Benchmarks: We use three modern decoder-based LLMs: GPT3, Chinchilla and Llama2 as our evaluation benchmark [5, 14, 46]. Among variants of these models, we select GPT3-175B with 175 billion parameters, Chinchilla-70B and Llama2-70B with 70 billion parameters each to model the scenario with insufficiency of HBM capacity. Note that we select Llama2-70B in addition to Chinchilla-70B to analyze the effect of grouped-query attention (GQA), despite their similar model parameter sizes. We assume INT8 precision for all three models, as we use ASIC accelerator chips to provide the computational power.

Performance Measurement: We only include decoder layers in our evaluation, as decoder layers occupy a significant portion of computations in decoder-based LLMs. Additionally, referring to the prior work, we measure the performance for a single iteration of the generation phase with a given

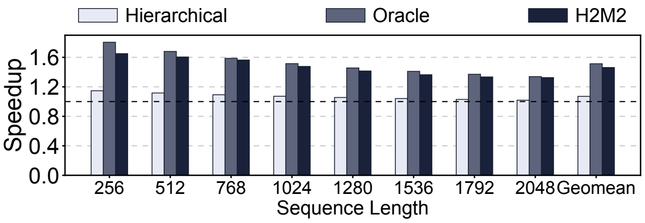


Figure 12. The relative speedup over the baseline (LPDDR-only) for hierarchical memory and asymmetric memory in GPT3-175B, batch size 32.

batch size and sequence length [13]. In this section, we fix the batch size for each model: 32 for GPT3, 64 for Chinchilla, and 128 for Llama2.

Evaluation Metric: We use the relative speedup as a main metric of performance evaluation, which is defined as (time elapsed for a single iteration of the baseline)/(time elapsed for a single iteration of H2M2). The relative speedup has same meaning with relative throughput and relative latency improvement in our evaluation scenarios: First, as a relative throughput of) H2M2 can be calculated as (the throughput of H2M2)/(the throughput of the baseline) and the throughput is equal to (batch size)/(time elapsed for a single iteration), the relative throughput has the same value with the relative speedup. Similarly, the latency improvement for time-between-token (TBT) can be calculated by (time elapsed for a single iteration of the baseline)/(time elapsed for a single iteration of H2M2), sharing a same definition with the relative speedup. Note that we do not include time-to-first-token (TTFT) metric in our evaluation, as TTFT is a latency evaluation metric for prompt phase, which is out of our scope.

5.2 Speedup

5.2.1 Effectiveness of Asymmetric Memory. We first compare the performance of H2M2 with two alternative configurations: A strict hierarchical memory architecture (`Hierarchical`) and an asymmetric memory architecture always using the best kernel-memory mapping (`Oracle`). Note that `Oracle` assumes no overhead from memory abstraction, by setting the cost of PTW/TLB access as zero.

GPT3-175B: Figure 12 visualizes the performance of GPT3-175B with the four configurations, translated to the relative speedup over the baseline LPDDR-only system. While the strict hierarchical memory shows a 1.07× speedup over the baseline on average, H2M2 reports a 1.46× speedup on average, reaching 0.97× of the ideal asymmetric memory performance. Due to the lack of temporal locality in LLM, on-demand migration of `Hierarchical` yields migration overhead, leading to relatively low performance boost. On the other hand, H2M2 takes an advantage of heterogeneous memory through nearly-optimal kernel-memory mapping without on-demand migration cost. This result demonstrates that H2M2 is a promising cost-efficient solution for LLM acceleration.

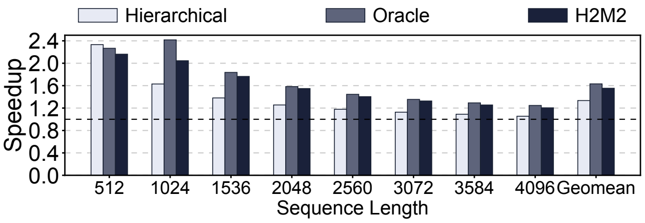


Figure 13. The relative speedup over the baseline (LPDDR-only) for hierarchical memory and asymmetric memory in Chinchilla-70B, batch size 64.

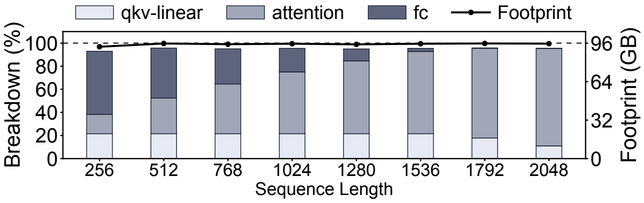


Figure 14. HBM footprint breakdown of H2M2 in GPT3-175B, batch size 32.

Chinchilla-70B: Figure 13 shows the performance of Chinchilla-70B with an asymmetric memory and alternative memory configurations, represented as a speedup over the baseline. Note that we use longer sequence lengths and larger batch size for Chinchilla-70B to model a scenario where heterogeneous memory is required due to the lack of bandwidth-centric memory capacity. Because of the small model size of Chinchilla-70B, both strict hierarchical memory and H2M2 shows higher performance boost over the baseline compared to GPT3-175B. In addition, strict hierarchical memory outperforms asymmetric memory with the sequence length smaller than 512: When the total footprint of LLM inference task is smaller than the HBM capacity, the performance of strict hierarchical memory become equivalent to the multi-HBM memory without communication cost. However, this is only a corner case for short sequence length and H2M2 still notably outperforms strict hierarchical memory and closely follows the performance of ideal asymmetric memory in general; strict hierarchical memory shows a 1.33× speedup over the baseline on average, and H2M2 shows a 1.55× performance boost, which is 0.95× of the ideal performance. This implies that in real-world LLM serving with dynamically growing sequence length, H2M2 is still efficient even with small model size.

5.2.2 Analysis with Footprint Breakdown. Figure 14 illustrates the breakdown of HBM utilization during the execution of GPT3-175B with a batch size 32, corresponding to Figure 12. The bars divided into three colors represent the HBM occupancy for *qkv-linear*, *attention*, and *fc*, while the line plot indicates the overall HBM footprint. HBM is nearly fully utilized across various sequence lengths as shown in

Figure 14, demonstrating the effectiveness of H2M2's mapping in optimizing HBM utilization. Furthermore, *attention* consumes a larger portion of HBM as a sequence length increases, while *fc* exhibits the opposite trend. Meanwhile, *qkv-linear* consistently occupies a moderate amount of HBM compared to *attention* and *fc*. These patterns in HBM utilization demonstrate that H2M2's mapping strategy effectively adapts to the runtime performance trends discussed in Section 3.3.

5.2.3 Effect of Grouped-Query Attention (GQA). Lllama2-70B employs grouped-query attention (GQA), an effective approach to address memory bandwidth limitations by allowing multiple heads to share the KV cache, with the number of heads determined by the parameter 'group size'. Consequently, GQA exhibits two distinctive characteristics compared to multi-head attention (MHA): (1) the storage and computational overhead for generating K and V tensors is reduced by a factor of 1/(group size), and (2) the size of KV cache in *attention* is also scaled down by the same factor. We incorporate the group size parameter into our performance model within H2M2 to account for the impact of GQA.

Figure 15 illustrates the impact of GQA on utilizing the asymmetric memory architecture. We use a batch size of 128, and the x-axis and y-axis respectively represent the speedup over the baseline and the sequence length. Note that a heterogeneous memory architecture becomes meaningful only when the application generates a sufficiently large memory footprint. Since the memory footprint in GQA is smaller due to the scaled-down size of the KV cache, we use larger batch size and sequence lengths compared to those in Figure 13. The average speedup over the baseline is 2.75× for `Hierarchical`, 3.00× for `Oracle`, and 2.94× for H2M2, excedding the speedups observed in Chinchilla-70B with a similar parameter size. This difference can be explained by the effects of GQA in reducing the memory footprint of *attention* and *qkv-linear*, which have the highest priority in HBM usage, resulting in improved HBM utilization. Although H2M2 generally outperforms `Hierarchical`, the opposite trend is observed for shorter sequence lengths, as the reduced KV cache size in GQA mitigates the migration cost of `Hierarchical`. Although H2M2 requires longer sequence length and larger batch size to be effective in smaller LLMs, this requirement is same as the escape condition for the corner case that the heterogeneous memory system is less effective than single-HBM system. Overall, the results presented in Figure 15 show that H2M2 remains effective even when grouped-query attention is applied.

5.2.4 Performance Overheads. While H2M2 follows up the performance of `Oracle` in all three LLMs used in our evaluation closely, still there exists some gap. We evaluate the temporal overhead introduced by *memory abstraction* and the reliability of the *greedy mapping policy* to justify the proposed architectural design of H2M2. Table 3 summarizes the average performance overhead introduced by applying memory abstraction (i.e. adding TLB access and miss latency) and replacing the oracle mapping policy (i.e. always finding the best) to greedy mapping policy. The performance gap between H2M2 and `Oracle` in Figure 12 and 13 comes from these performance overhead. As shown in the first column of the table, the memory abstraction incurs negligible temporal overhead of which is less than 2% for all three models used in the evaluation. In addition, the greedy mapping policy adds only 2.56%, 3.76%, and 0.60% additional performance overhead for GPT3-175B, Chinchilla-70B, and Llama2-70B, respectively. This result shows the robustness of the asymmetric memory architecture with memory abstraction and proposed mapping decision algorithm, with various model sizes and characteristics.

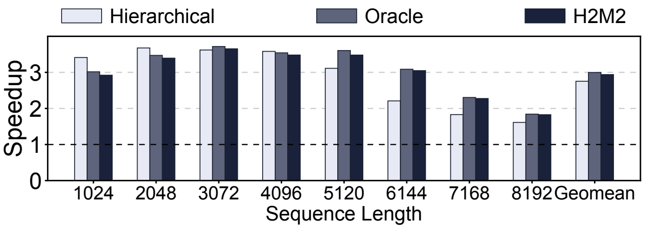


Figure 15. The relative speedup over the baseline (LPDDR-only) for hierarchical memory and asymmetric memory in Llama2-70B, batch size 128.

Model	Temporal Overhead		
	Abstraction	Mapping	Total
GPT3-175B	0.80%	2.56%	3.36%
Chinchilla-70B	1.01%	3.76%	4.78%
Llama2-70B	1.36%	0.60%	1.96%

Table 3. Average temporal overhead of H2M2.

5.3 Dynamic Sequence Length

Experiments of Section 3 and Section 5.2 assume all requests in a batch have equal sequence length without early termination of any requests. Here, to evaluate the performance of H2M2 and the efficiency of greedy mapping policy in realistic and dynamic situation, we consider another scenario where requests in a batch can have different sequence length and can be terminated in random moment. We assume that requests are consecutively fed to the system, and we fix the batch size to 32. For every iteration, each request can either generate the next token (i.e. sequence length increases by 1), or can be finished and replaced to the new request (i.e. sequence length changes randomly). Such a dynamic scenario leads KV cache size to fluctuate in runtime, making optimal mapping change frequently.

Figure 16 visualizes the performance of H2M2, `Oracle`, and `FlexGen` with dynamic sequence length scenario, measured during 128 iterations in GPT3-175B. The x-axis is the number of cumulative iterations, and the y-axis is a speedup

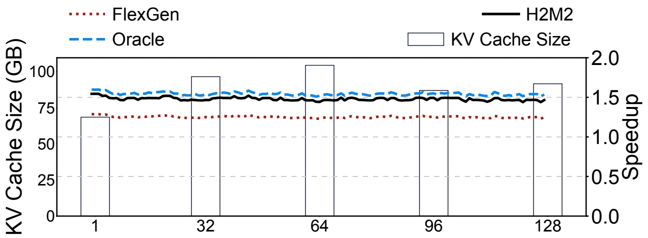


Figure 16. The relative speedup over the baseline for asymmetric memory, in dynamic sequence length scenario in GPT3-175B.

Name	Component	Configuration
`Original`	No modification	
`HBMcap-Less`	HBM capacity	48GB (0.5×)
`HBMcap-More`	HBM capacity	192GB (2×)
`HBMbw-Less`	HBM bandwidth	2.25TB/s (0.75×)
`HBMbw-More`	HBM bandwidth	4TB/s (1.3×)
`LPDDRbw-Less`	LPDDR bandwidth	408GB/s (0.75×)
`LPDDRbw-More`	LPDDR bandwidth	680GB/s (1.25×)
`HBMChip-More`	HBM-side compute units	2 chips (2×)
`LPDDRChip-More`	LPDDR-side compute units	2 chips (2×)

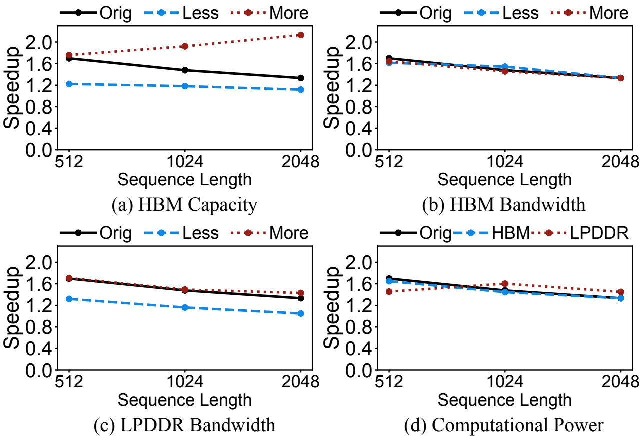
Table 4. Memory configuration variants for sensitivity study.

over the baseline. Moreover, we measure the total size of KV cache for five checkpoints: the 1st, 32nd, 64th, 96th, and 128th iteration, respectively. As shown in Figure 16, H2M2 shows a stable speedup over the baseline even with the dynamic change of KV cache size as well as following the performance of `Oracle` closely - While `FlexGen` only reports 1.25× speedup over the baseline on average, H2D2 shows 1.48× speedup over the baseline on average, reaching 0.96× performance of `Oracle`. This implies the robustness of *greedy mapping policy*. Note that `FlexGen` reports larger performance gap under H2M2 in Figure 16 compared to Figure 7, due to the difference of the batch size.

5.4 Sensitivity Study

To demonstrate the robustness of H2M2 design to different hardware configurations, we conduct a sensitivity study with eight different hardware configurations outlined in Table 4. While `Original` is an original memory configuration of Table 1, the other configurations are created by changing one of the configuration parameters from `Original`: Either deducting (`Less`) or increasing (`More`) HBM capacity, HBM bandwidth, LPDDR bandwidth, HBM-side computational power, and LPDDR-side computational power.

Figure 17 presents the performance trend of various hardware configurations compared to `Original`. As shown in Figure 17(a), the relative performance of H2M2 over the baseline is highly sensitive to the capacity of HBM (`HBMcap`). This is because the capacity of HBM is directly related to the amount of kernels that can be mapped to HBM, deciding the performance of overall system. The bandwidth of LPDDR (`LPDDRbw`) and the computational power of LPDDR-side accelerator (`LPDDRChip`) also shows some impacts on performance, as illustrated in Figure 17(c) and (d). As the performance of H2M2 is bounded to the LPDDR-side execution, the change of LPDDR-side configuration affects the performance of overall system. Meanwhile, the sensitivity of performance to other parameters such as HBM bandwidth are almost negligible, as shown in Figure 17(b). This is because other parameters do not make significant impact on the performance of the critical path.



(a) HBM Capacity (b) HBM Bandwidth (c) LPDDR Bandwidth (d) Computational Power

Figure 17. Sensitivity study for H2M2 with GPT3-175B.

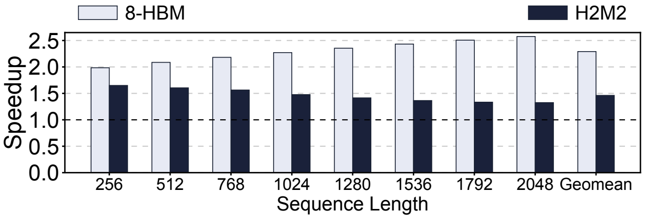


Figure 18. The performance improvement of H2M2 and multi-HBM system (`8-HBM`) over the baseline (LPDDR-only) for GPT3-175B, batch size 32.

Despite the variance on performance trends of eight alternative configurations, analyzing the distinctions in each configuration indicates the reasonableness of diverse patterns. This sensitivity study highlights the reliability of deploying the H2M2 for efficient LLM acceleration in various hardware configurations.

5.5 Comparison with Multi-HBM System

To compare H2M2 with multi-HBM system, we introduce a new configuration called `8-HBM`, an HBM-only system with eight HBM devices that offers total capacity of 768GB. As multiple HBM devices require communication between memory modules, we add communication cost to `8-HBM` which was measured by profiling multi-GPU system with eight NVIDIA A100 GPUs.

Speedup: Figure 18 compares the relative speedup of H2M2 and `8-HBM` over the baseline LPDDR-only system for GPT3-175B, with batch size of 32. While H2M2 achieves average

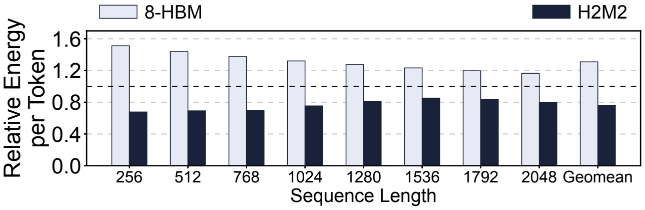


Figure 19. The relative memory energy per token H2M2 and 8-HBM over the baseline (LPDDR-only) for GPT3-175B, batch size 32. Lower is better.

speedup of 1.46× over the baseline, 8-HBM outperforms the baseline with an average speedup of 2.29×, which can be trasnlated into 1.57× additional speedup over H2M2. Such performance gap between 8-HBM and H2M2 comes from from the limited capacity of HBM as well as low locality of LLM.

Energy Efficiency: To evaluate the energy efficiency, we compare the energy consumption per token of H2M2 with that of 8-HBM. Note that the computational capability of both H2M2 and 8-HBM remains unchanged, and memory access overhead generally dominates power consumption compared to computational overhead [6, 9, 23, 48]. Accordingly, our static power model focuses solely on memory accesses, assuming equal energy consumptions for read and write operations. The relative energy consumption of HBM and LPDDR in our static power model is derived from the prior work [36]. Figure 19 visualizes the relative memory energy consumption per token of H2M2 and 8-HBM over the baseline LPDDR-only system. As shown in the figure, H2M2's energy consumption per token is 0.76× of the baseline on average, while 8-HBM reports 1.31× of the baseline on average. Considering that the relative energy consumption of the baseline LPDDR-only system is translated to 1, this result justifies the asymmetric memory system design for LLMs as H2M2 is more energy-efficient than both multi-HBM and LPDDR-only systems.

Cost and Scalability: Recall Section 2.3 , larger memory capacity requires more devices with multi-HBM systems, leading multi-HBM systems to become increasingly affected by communication overhead. Because of this, the cost-to-performance efficiency of multi-HBM systems tends to degrade as the system's memory capacity increases [54]. In contrast, asymmetric memory systems combine highly scalable LPDDR with less scalable HBM, allowing memory capacity to be expanded on the LPDDR side. Based on this design, H2M2 makes it easier to mitigate the decline in cost-to-performance efficiency as memory capacity increases.

6 Related Work

LLM Serving Systems: The research into systems for efficient LLM execution has been primarily carried out on GPUs. Orca improved the throughput of LLM inference by adopting iteration-level scheduling [49]. FlashLLM reduced data traffic by using flash memory and accessing data within larger granularity [1]. FlashAttention lowered memory usage and I/O costs of attention layers through input block organization and kernel optimization [8]. PowerInfer alleviates GPU memory limitations by uploading only the active (hot) neurons of the model's weights to the GPU, while delegating the remaining computations to the host CPU [44]. ALISA and InfiniGen reduces memory demands and swap overhead in LLM inference by employing sparse attention, processing only critical key-value pairs during attention operations [26, 53].

LLM Accelerators: LLM accelerators have been deeply researched for power efficiency and performance via specialized hardware and parallel architectures. DFX implemented a specialized hardware accelerator architecture for LLM with FPGA [15]. FlightLLM used mixed precision to place activations in the on-chip memory during the decode stage to accelerate LLM inference [50].

Near-memory Processing for LLMs: Several studies have investigated near-memory processing to accelerate LLMs. IANUS, NeuPIMs and AttAcc integrated GEMV-optimized PIM technology to accelerate the attention layer efficiently [13, 35, 41]. Smart-Infinity utilized computational storage devices for updating parameters in LLM training [18].

7 Conclusion

With asymmetric memory for LLM acceleration, this study investigated the mapping decision algorithm, and proposed a memory abstraction scheme for the efficient management. The propose H2M2 outperformed the conventional LPDDR-based homogeneous memory system by 1.46× speedup in GPT3-175B on average, and traditional strict hierarchical memory by 1.36×.

References

[1] Keivan Alizadeh, Iman Mirzadeh, Dmitry Belenko, Karen Khatamifard, Minsik Cho, Carlo C Del Mundo, Mohammad Rastegari, and Mehrdad Farajtabar. 2023. Llm in a flash: Efficient large language model inference with limited memory. *arXiv preprint arXiv:2312.11514* (2023).

[2] Reza Yazdani Aminabadi, Samyam Rajbhandari, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Olatunji Ruwase, Shaden Smith, Minjia Zhang, Jeff Rasley, et al. 2022. Deepspeed-inference: enabling efficient inference of transformer models at unprecedented scale. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–15.

[3] Rohan Anil, Andrew M. Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, et al. 2023. PaLM 2 Technical Report. *arXiv preprint arXiv:2305.10403* (2023). arXiv:2305.10403 [cs.CL]

[4] Alexander Borzunov, Max Ryabinin, Artem Chumachenko, Dmitry Baranchuk, Tim Dettmers, Younes Belkada, Pavel Samygin, and Colin A Raffel. 2024. Distributed Inference and Fine-tuning of Large Language Models Over The Internet. *Advances in Neural Information Processing Systems* 36 (2024).

[5] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss,

Grechen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[6] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. 2016. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. *ACM SIGARCH computer architecture news* 44, 3 (2016), 367–379.

[7] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2022. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311* (2022).

[8] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems* 35 (2022), 16344–16359.

[9] Mingyu Gao, Jing Pu, Xuan Yang, Mark Horowitz, and Christos Kozyrakis. 2017. Tetris: Scalable and efficient neural network acceleration with 3d memory. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*. 751–764.

[10] Saugata Ghose, Tianshi Li, Nastaran Hajinazar, Damla Senol Cali, and Onur Mutlu. 2019. Demystifying Complex Workload-DRAM Interactions: An Experimental Study. 3, 3 (2019).

[11] Google. 2018. CloudTPU. https://cloud.google.com/tpu/docs/system-architecture-tpu-vm.

[12] Albert Gu and Tri Dao. 2023. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752* (2023).

[13] Guseul Heo, Sangyeop Lee, Jaehong Cho, Hyunmin Choi, Sanghyeon Lee, Hyungkyu Ham, Gwangsun Kim, Divya Mahajan, and Jongse Park. 2024. NeuPIMs: A NPU-PIM Heterogeneous Acceleration for Batched Inference of Large Language Model. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'24)*.

[14] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbi, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jaek W. Rae, Oriol Vinyals, and Laurent Sifre. 2022. Training Compute-Optimal Large Language Models. *arXiv preprint arXiv:2203.15556* (2022).

[15] Seongmin Hong, Seungjae Moon, Junsoo Kim, Sungjae Lee, Minsub Kim, Dongsoo Lee, and Joo-Young Kim. 2022. DFX: A Low-latency Multi-FPGA Applicance for Accelerating Transformer-based Text Generation. In *2022 55th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*.

[16] Soojin Hwang, Sunho Lee, Jungwoo Kim, Hongbeen Kim, and Jaehyuk Huh. 2023. mNPUsim: Evaluating the Effect of Sharing Resources with Multi-core NPUs. In *2023 IEEE International Symposium on Workload Charcterization (IISWC)*.

[17] Bongjoon Hyun, Youngeun Kwon, Yujeong Choi, John Kim, and Minsoo Rhu. 2020. NeuMMU: Architectural Support for Efficient Address Translations in Neural Processing Units. In *Proceedings of the 25th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'20)*. 1109–1124.

[18] Hongsun Jang, Jaeyong Song, Jaewon Jung, Jaeyoung Park, Youngsok Kim, and Jinho Lee. 2024. Smart-Infinity: Fast Large Language Model Training using Near-Storage Processing on a Real System. In *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 345–360.

[19] Jinwoo Jeong, Seungsu Baek, and Jeongseob Ahn. 2023. Fast and Efficient Model Serving Using Multi-GPUs with Direct-Host-Access. In *Proceedings of the Eighteenth European Conference on Computer Systems* (Rome, Italy) *(EuroSys '23)*. 249–265. https://doi.org/10.1145/3552326.3567508

[20] Hongshin Jun, Jinhee Cho, Kangseol Lee, Ho-Young Son, Kwiwook Kim, Hanho Jin, and Keith Kim. 2017. HBM (High Bandwidth Memory) DRAM Technology and Architecture. In *2017 IEEE International Memory Workshop*.

[21] Sheng-Chun Kao, Suvinay Subramanian, Gaurav Agrawal, Amir Yazdanbakhsh, and Tushar Krishna. 2023. FLAT: An Optimized Dataflow for Mitigating Attention Bottlenecks. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. 295–310.

[22] Vijay Anand Korthikanti, Jared Casper, Sangkug Lym, Lawrence McAfee, Michael Andersch, Mohammad Shoeybi, and Bryan Catanzaro. 2023. Reducing activation recomputation in large transformer models. *Proceedings of Machine Learning and Systems* 5 (2023), 341–353.

[23] Hyoukjun Kwon, Prasanth Chatarasi, Michael Pellauer, Angshuman Parashar, Vivek Sarkar, and Tushar Krishna. 2019. Understanding Reuse, Performance, and Hardware Cost of DNN Dataflow: A Data-Centric Approach. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. 754–768.

[24] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles* (Koblenz, Germany) *(SOSP'23)*. https://doi.org/10.1145/3600006.3613165

[25] Sukhan Lee, Shin-haeng Kang, Jaehoon Lee, Hyeonsu Kim, Eojin Lee, Seungwoo Seo, Hosang Yoon, Seungwon Lee, Kyounghwan Lim, Hyunsung Shin, Jinhyun Kim, O Seongil, Anand Iyer, David Wang, Kyomin Sohn, and Nam Sung Kim. 2021. Hardware Architecture and Software Stack for PIM Based on Commercial DRAM Technology : Industrial Product. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. 43–56.

[26] Wonbeom Lee, Jungi Lee, Junghwan Seo, and Jaewoong Sim. 2024. InfiniGen: Efficient Generative Inference of Large Language Models with Dynamic KV Cache Management. In *Proceedings of the 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*.

[27] Dacheng Li, Rulin Shao, Anze Xie, Eric P Xing, Joseph E Gonzalez, Ion Stoica, Xuezhe Ma, and Hao Zhang. 2023. Lightseq: Sequence level parallelism for distributed training of long context transformers. *arXiv preprint arXiv:2310.03294* (2023).

[28] Shenggui Li, Fuzhao Xue, Chaitanya Baranwal, Yongbin Li, and Yang You. 2021. Sequence parallelism: Long sequence training from system perspective. *arXiv preprint arXiv:2105.13120* (2021).

[29] S. Li, Z. Yang, D. Reddy, A. Srivastava, and B. Jacob. 2020. DRAMsim3: A Cycle-Accurate, Thermal-Capable DRAM Simulator. 19, 2 (2020), 106–109.

[30] NVIDIA. 2006. CUDA Toolkit Document. https://docs.nvidia.com/cuda/cuda-runtime-api.

[31] NVIDIA. 2014. Unified Memory Programming. https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#um-unified-memory-programminghd.

[32] NVIDIA. 2023. Grace Hopper Superchip. https://www.nvidia.com/en-us/data-center/grace-hopper-superchip/.

[33] NVIDIA. 2024. Blackwell Superchip. https://www.nvidia.com/en-us/data-center/gb200-nvl72/.

[34] OpenAI. 2023. GPT-4 Technical Report. *arXiv preprint arXiv:2303.08774* (2023). arXiv:2303.08774 [cs.CL]

[35] Jaehyun Park, Jaewan Choi, Kwanhee Kyung, Michael Jaemin Kim, Yongsuk Kwon, Nam Sung Kim, and Jung Ho Ahn. 2024. AttAcc!

Unleashing the Power of PIM for Batched Transformer-based Generative Model Inference. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'24)*.

[36] Sang-Soo Park, KyungSoo Kim, Jinin So, Jin Jung, Jonggeon Lee, Kyoungwan Woo, Nayeon Kim, Younghyun Lee, Hyungyo Kim, Yongsuk Kwon, et al. 2024. An LPDDR-based CXL-PNM Platform for TCO-efficient Inference of Transformer-based Large Language Models. In *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 970–982.

[37] Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. 2023. Efficiently scaling transformer inference. *Proceedings of Machine Learning and Systems* 5 (2023).

[38] Ramya Prabhu, Ajay Nayak, Jayashree Mohan, Ramachandran Ramjee, and Ashish Panwar. 2024. vAttention: Dynamic Memory Management for Serving LLMs without PagedAttention. *arXiv preprint arXiv:2405.04437* (2024).

[39] Minsoo Rhu, Natalia Gimelshein, Jason Clemons, Arslan Zulfiqar, and Stephen W Keckler. 2016. vDNN: Virtualized Deep Neural Networks for Scalable, Memory-Efficient Neural Network Design. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 1–13.

[40] Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, et al. 2023. BLOOM: A 176B-Parameter Open-Access Multilingual Language Model. *arXiv preprint arXiv:2211.05100* (2023). arXiv:2211.05100 [cs.CL]

[41] Minseok Seo, Xuan Truong Nguyen, Seok Joong Hwang, Yongkee Kwon, Guhyun Kim, Chanwook Park, Ilkon Kim, Jaehan Park, Jeongbin Kim, Woojae Shin, Jongsoon Won, Haerang Choi, Kyuyoung Kim, Daehan Kwon, Chunseok Jeong, Sangheon Lee, Yongseok Choi, Wooseok Byun, Seungcheol Baek, Hyuk-Jae Lee, and John Kim. 2024. IANUS: Integrated Accelerator based on NPU-PIM Unified Memory System. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'24)*.

[42] Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. 2023. FlexGen: High-Throughput Generative Inference of Large Language Models with a Single GPU. In *Proceedings of the 40th International Conference on Machine Learning* (Honolulu, Hawaii, USA) *(ICML'23)*.

[43] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2019. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053* (2019).

[44] Yixin Song, Zeyu Mi, Haotong Xie, and Haibo Chen. 2024. PowerInfer: Fast Large Language Model Serving with a Consumer-grade GPU. In *Proceedings of the 30th Symposium on Operating Systems Principles (SOSP)*.

[45] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).

[46] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. arXiv:2307.09288 [cs.CL]

[47] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. In *31st Conference on Neural Information Processing Systems (NIPS 2017)*.

[48] Yannan Nellie Wu, Joel S Emer, and Vivienne Sze. 2019. Accelergy: An architecture-level energy estimation methodology for accelerator designs. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 1–8.

[49] Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. 2022. Orca: A distributed serving system for Transformer-Based generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. 521–538.

[50] Shulin Zeng, Jun Liu, Guohao Dai, Xinhao Yang, Tianyu Fu, Hongyi Wang, Wenheng Ma, Hanbo Sun, Shiyao Li, Zixiao Huang, et al. 2024. FlightLLM: Efficient Large Language Model Inference with a Complete Mapping Flow on FPGA. *arXiv preprint arXiv:2401.03868* (2024).

[51] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. 2022. OPT: Open Pre-trained Transformer Language Models. arXiv:2205.01068 [cs.CL]

[52] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. 2023. A Survey on Large Language Models. *arXiv preprint arXiv:2303.18223* (2023). arXiv:2303.18223 [cs.CL]

[53] Youpeng Zhao, Di Wu, and Jun Wang. 2024. ALISA: Accelerating Large Language Model Inference via Sparsity-Aware KV Caching. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*.

[54] Lianmin Zheng, Zhuohan Li, Hao Zhang, Yonghao Zhuang, Zhifeng Chen, Yanping Huang, Yida Wang, Yuanzhong Xu, Danyang Zhuo, Eric P. Xing, Joseph E. Gonzalez, and Ion Stoica. 2022. Alpa: Automating Inter- and Intra-Operator Parallelism for Distributed Deep Learning. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. USENIX Association, Carlsbad, CA. https://www.usenix.org/conference/osdi22/presentation/zheng-lianmin