McPAL: Scaling Unstructured Sparse Inference with Multi-Chiplet HBM-PIM Architecture for LLMs

Placeholder for Double Blind Submission

***Abstract*—Large language models (LLMs) have gained significant attention recently. However, executing LLM is memory-bound due to the extensive memory accesses. Process-in-memory (PIM) emerges as an energy-efficient solution for LLMs, delivering high memory bandwidth and compute parallelism. Nevertheless, the trend towards larger LLMs introduces escalating memory footprint challenges for monolithic PIM chips. This paper proposes McPAL, which tackles this challenge by emphasizing unstructured sparse compute within PIM and hierarchical multi-chiplet scaling. McPAL decomposes arbitrary sparse weight matrix into multiple irregular sparse vectors. The non-skipped computations in each vector are then routed via an in-memory butterfly network to the standard PIM array, enhancing the PIM utilization. In addition, we scale McPAL vertically by strategically organizing the 3D-HBM hierarchy to minimize the internal long-distance data travel. Meanwhile, a 2.5D IO chiplet scales McPAL horizontally, reducing die-to-die (D2D) data transfer and ensuring sparse workload balance. We conducted extensive experiments from Llama-7B to Llama-70B. The results show that McPAL achieves 1.57× to 3.12× speedup and 10.43× to 35.66× energy efficiency over Nvidia A100 GPU. Compared to SOTAs, McPAL also achieves 1.08× to 2.15× speedup and 1.65× to 5.14× energy efficiency.**

***Index Terms*—HBM, Chiplet, Large Language Model, Unstructured Sparsity, Processing-in-Memory**

I. Introduction

LLMs exhibit superior capability to learn long-range contextual dependencies and remarkable scalability as parameters grow, from GPT-2 1.5 billion [1] to GPT-4 1.76 trillion [2] in just 5 years. As LLMs continue to expand, a simultaneous need arises to scale up compute throughput, memory capacity, and inter-chip bandwidth. Conventional architectures like GPUs typically store all parameters in an external memory chip (e.g. DRAM) and perform LLMs on another separate processor chip, moving large-scale weights back and forth. Unfortunately, CMOS compute logic scales much faster than inter-chip links and memory capacity [4]. As a result, the performance of GPU-DRAM systems is often constrained by the limited bandwidth between memory and processor chips.

PIM is a promising solution for memory-intensive LLMs. It features ultra-tight coupling between compute and storage cells, avoiding the performance bottleneck limited by the memory-processor bandwidth. PIM pioneers leverage dataflow optimization [10]–[12] and attention sparsity [13]–[15] to accelerate self-attention in LLMs, but leave linear layers unexplored. These linear layers frequently become bottlenecks during LLM inference. For example, the Llama-7B model [3] with 4K tokens requires 6.47 billion weights for its linear layers, compared to just 1.07 billion for self-attention's key-value caching. The extensive weight parameters take up 85.8% of model storage, placing significant memory demands on LLM systems.

Sparsification Techniques [16]–[18] are widely adopted to reduce the memory footprint of linear layers. However, the irregular memory access and compute pattern of sparse weight matrices are inherently incompatible with the regular PIM structure. Consequently, previous PIM designs primarily support structured sparsity, where weights are pruned in block-wise [21], [22] or channel-wise [23] patterns. However, SparseGPT [6] proves that structured sparsity could lead to performance degradation in LLMs. In addition, ***Multi-Chiplet Integration*** emerges as another approach to manage memory footprint, breaking the monolithic limitation of a single PIM chip. Heterogeneous integration enables the creation of a virtual "*super-die*" by dividing it into multiple chiplets. Thanks to the flexibility and scalability, various chiplet architectures [24], [25] were proposed for high-performance computing. However, these works primarily optimize compute-bound tasks, such as matrix multiplication, but lack specific optimizations tailored for memory-bound LLMs, resulting in redundant storage and inefficient data transfer [19].

In summary, efficient LLM execution in PIM requires satisfying the following requirements: (1) The overall architecture should support the full scope of LLM operations, including not only self-attention but also linear layers and other components. (2) The PIM chiplet should accommodate unstructured weight sparsity to minimize the memory overhead without compromising LLM accuracy and hardware utilization. (3) The PIM-chiplet system meticulously optimizes LLM workflows, maximizing compute/storage utilization while minimizing inter-chiplet communication. This paper proposes McAPL to address these challenges through the following contributions,

- McPAL explores unstructured sparsity in HBM-PIM through a novel matrix decomposition algorithm. It decomposes arbitrary sparse weight matrices into irregular sparse vectors, which are then routed through an in-memory butterfly network for regular vector products in PIM. This method delivers nearly 2× speedup at 50% sparsity, significantly reducing memory footprint without compromising LLM accuracy or hardware utilization.
- McPAL introduces a hierarchical multi-chiplet scaling topology. We first scale McPAL vertically by strategically organizing the 3D-HBM hierarchy to minimize HBM's internal long-distance data travel by through-silicon vias (TSVs). A dedicated 2.5D IO chiplet furthers scales McPAL vertically to reduce D2D data synchronization and enables sparse workload balance.
- We conduct extensive experiments to test McPAL scalability from Llama-7B to Llama-70B models. The results show that McPAL achieves 1.57× to 3.12× speedup and 10.43× to 35.66× energy efficiency over Nvidia A100 GPU. Compared to SOTA accelerators, McPAL also achieves 1.08× to 2.15× speedup and 1.65× to 5.14× energy efficiency.

II. Background and Motivation

A. LLM Preliminaries

Fig. 1 illustrates the LLM structure, comprising an embedding layer, a stack of transformer layers and a language model (LM) head. The embedding converts discrete tokens into feature vectors, while the LM head predicts the next token based on the transformer output. Between these lies the transformer layer, which is the core component of LLMs. It consists of two consecutive blocks: self-attention and feed-forward. The self-attention first projects input vectors into three intermediate features: Query (Q), Key (K) and Value (V). The Q×K yields attention scores (S), then normalized by a softmax to produce attention probabilities (P). Next, the P×V yields attention outputs. These outputs are fed into the feed-forward network, which comprises several linear layers separated by non-linear activation functions.

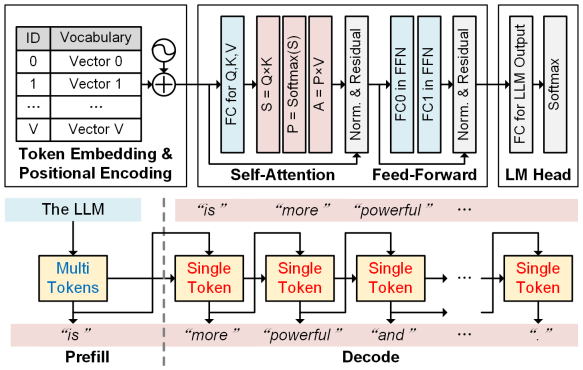

Fig. 1. LLM and transformer topology.

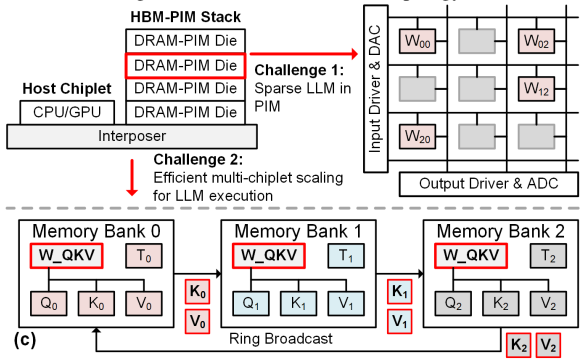

Fig. 2. Challenges for HBM-PIM chiplet architecture.

Most LLMs are pretrained for causal modelling, essentially as the next-token predictors. As shown in Fig. 1, LLMs typically contain two distinct phases. The prefill phase processes the input prompt in a single inference to generate the first new token. Since all tokens in the prompt are predetermined, LLMs perform matrix-matrix multiplication, making the prefill compute-bound. In contrast, the decode phase aggressively generates a single token per inference, serving as the input in the next iteration. The KV vectors of the input prompt and the generated tokens are stored to construct the "*KV Cache*", which is reused in the subsequent decode inference. The decode phase involves vector-matrix multiplication and is memory-bound. According to S3 [28], A100 GPU utilization reaches 68.4% during GPT-J model inference with a batch size of 512 (compute-bound), but drops to 0.4% with a single batch (memory-bound). It indicates that bandwidth limitation is the main bottleneck in LLM acceleration. Therefore, the PIM, offering high memory bandwidth, is a promising solution for accelerating memory-bound LLMs.

B. Motivations for HBM-PIM-Chiplet Architecture

The chiplet architecture integrates multiple smaller dies through 2.5D/3D advanced packaging technologies. This modular approach offers several advantages, including improved yield, reduced development time, and enhanced customization. Combining large-capacity HBMs, the HBM-PIM-chiplet architecture is developed. While HBM-PIM [19] and chiplet [24] are individually prevalent, their integration into a multi-chiplet HBM-PIM architecture raises new challenges:

❶ **Efficient LLM storage in HBM-PIM.** The billions of parameters in LLMs create substantial memory demands, far exceeding the photolithographic limitation of a single HBM chiplet. While sparsification can minimize LLM size, most HBM-PIM designs struggle with under-utilization when mapping sparse weight matrices. Given the PIM crossbar in Fig. 2, the randomly distributed sparsity patterns lead to DRAM cells filled with zeros, resulting in low utilization. Consequently, most PIM designs only support block-wise [21], [22] or channel-wise [23] structured sparsity. SparseGPT [6] proves that structured sparsity could degrade LLM performance. Therefore, a PIM design that supports unstructured sparsity is necessary, distinguishing our work from other HBM-PIM works. ❷ **Efficient multi-chiplet scaling for LLM execution.** TransPIM [19] is the first PIM design to explore LLM scaling by distributing the input prompt across multiple HBM banks. As shown in Fig. 2, it duplicates all weight parameters in bank groups to enable parallel computing, which is impractical for larger LLMs. Moreover, it ring-broadcasts and synchronizes the KV cache for each input token. Since the KV cache size scales with the number of tokens, TransPIM incurs massive data transfers, rendering it inefficient for handling long input sequences, such as those with 4K tokens. Finally, TransPIM optimizes matrix multiplication within the HBM-PIM die, but overlooks auxiliary operations such as normalization, embedding, etc. In practical LLM applications, these remaining operations necessitate an additional master, which diminishes PIM efficiency by introducing substantial long-distance data travel within HBM or even inter-chip communication. To mitigate this issue, an efficient multi-chiplet scaling scheme tailored for LLMs is essential, setting McPAL apart from other chiplet-based architectures.

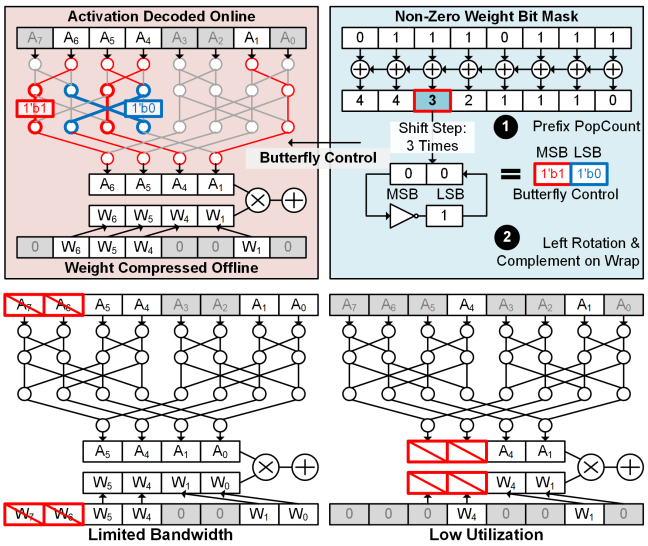

Fig. 3. (a) BFLY network to map M:N sparsity in PIM. and (b) The challenge of mapping unstructured sparsity in PIM using BFLY.

III. Sparsity-Aware McPAL PIM Design

A. Local Bank-Balance M:N Sparsity in PIM

Mapping unstructured sparsity to PIM is the knob to reduce memory footprint without sacrificing LLM performance. We first review the local-balanced M:N sparsity [8], [9], which lays the foundation for the subsequent exploration of the fully unstructured sparsity in McPAL HBM-PIM. In M:N sparsity, the weight matrix typically reserves $\boldsymbol{M}$ non-zero weights (NZWs) out of $\boldsymbol{N}$ elements within each weight vector. The sparse vector is then compressed and encoded by a bitmask. Guided by the bitmask, the activations associated with the NZWs are routed from the original inputs. The product between the compressed NZWs and the extracted activations effectively converts the M:N sparsity into regular vector products, which is well-supported in many PIM designs.

Fig. 3 depicts the M:N sparse compute flow and the butterfly network (BFLY) to route the non-skipped activations to NZWs [9]. The BFLY decoder contains two primary operations: prefix popcount and left-rotation complement-on-wrap (LRCW). The popcount

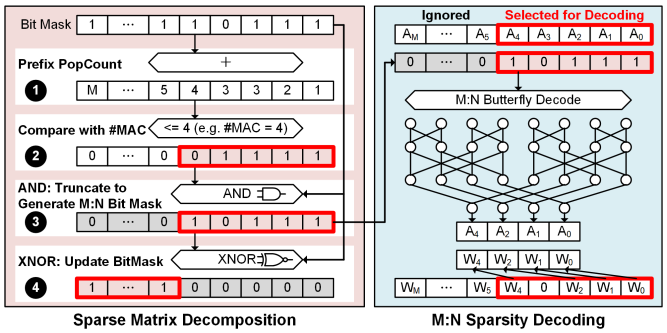

Fig. 4. Decompose sparse weight matrix to avoid limited bandwidth in BFLY.

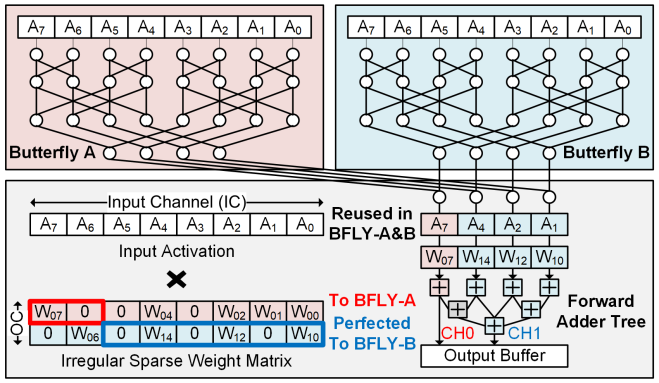

Fig. 5. Double-buffering BFLY to address low utilization issue.

performs cumulative sum on the given bitmask. The count of bit-1 indicates the number of NZWs up to a certain position. LRCW is a standard left rotation except that the MSB is complemented whenever it wraps around. Fig. 3 shows how to generate the control bits for two BFLY switches in stage 2. The two LSBs of the popcount result, PP[5], are used as the rotation step to execute LRCW on two zeros ('00'), which yields control bits '10'. Other switches follow a similar decoding method just with different popcount results. BFLY accurately directs the activations to the paired NZWs in M:N sparsity. However, due to the random distribution of NZWs in arbitrary unstructured sparsity, BFLY may cause limited bandwidth or low utilization. As shown in Fig. 3, the limited bandwidth arises when the number of NZWs exceeds BFLY's output bandwidth, preventing simultaneous extraction of all required activations. Conversely, low utilization happens when the weight vector is highly sparse, resulting in insufficient workloads to fully utilize the downstream MACs.

B. Unstructured Sparsity in McPAL PIM

McPAL leverages an algorithm-architecture co-design to enable arbitrary unstructured sparsity in PIM. ❶ Sparse weight matrix decomposition (SMD) and ❷ double-buffering BFLY (DB^{2}FLY) address the limited bandwidth and low utilization issues, respectively.

*1) **Sparse Weight Matrix Decomposition**:* The essence of SMD is to break down any sparse weight matrix into multiple M:N sparse vectors, which are then decoded as described in Sec. III-A. As shown in Fig. 4, SMD includes four steps to align the excess NZWs with both BFLY bandwidth and the downstream MACs (#MAC). ❶ Perform prefix popcount on the input bitmask to count the number of NZWs up to each position. ❷ Compare the popcount result with #MAC. The resulting bit-1s define the range of the first #MAC NZWs. ❸ Perform element-wise AND operation between the original bitmask and the comparison result. It truncates the bitmask for the first #MAC NZWs and sets the remaining to zero for the excess NZWs. The truncated bitmask is then forwarded to the standard BFLY decoder. With the bitmask for the excess NZWs set to zero, BFLY will only route the first #MAC NZWs. ❹ Finally, perform element-wise XNOR operation between the truncated bitmask and the original bitmask. It updates the bitmask for the next SMD iteration. In summary, the four-step SMD decomposes arbitrary sparse weight matrix to multiple M:N sparse vectors, enabling effective PIM mapping using the BFLY decoding logic.

*2) **Double Buffering Butterfly Network**:* McPAL uses DB^{2}LFY to tackle the low utilization when handling highly sparse weight matrices. As shown in Fig. 5, the DB^{2}FLY comprises two independent regular BFLYs. If the sparse weight vector has enough NZWs to fill all MACs, only one BFLY is activated to reduce dynamic power. Otherwise, DB^{2}FLY activates both BFLYs to allocate adequate workload from a boarder region. Take the sparse weight matrix in Fig. 5 as an example. At the first cycle, McPAL employs SMD to produce M:N sparse compute for the first four NZWs, W_{00} to W_{04}. As a result, the remaining weight vector contains only one NZW, W_{07}, leading to insufficient data to fully utilize all MACs in the next iteration. To prevent low utilization, the DB^{2}FLY prefetches and then routes the NZWs from two consecutive output channels. Each BFLY handles M:N sparsity for its respective channel. In addition, both BFLYs reuse the same input activations, thereby reducing the input bandwidth requirement. The partial sums from these two channels cannot be accumulated directly. To resolve this issue, we leverage the forwarding adder tree in Sigma [17] for parallel accumulations within a single adder tree.

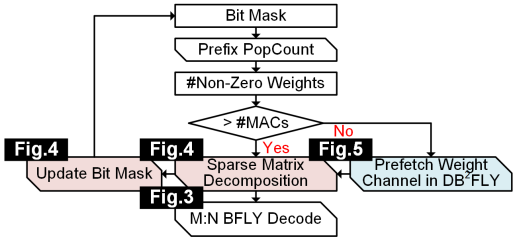

Fig. 6. The overall workflow of McPAL to map unstructured sparsity in PIM.

The integration of SMD and DB^{2}FLY enables McPAL to deal with unstructured sparsity in PIM. Fig. 6 summarizes the overall flow of the unstructured sparse computing. First, McPAL conducts prefix popcount on the input bitmask to calculate the number of NZWs and compares it with #MAC. As long as the NZWs can fill all MACs, SMD is then executed. Otherwise, the DB^{2}FLY prefetches the next weight channel to mitigate the low utilization. McPAL supports prefetching only two consecutive channels. That's because more channels would require a larger BFLY, forwarding adder tree and additional memory for partial sum storage, resulting in a higher hardware overhead. Although NZWs in these two channels may still fail to fully utilize the downstream MACs, the experimental results in Sec.V demonstrate that DB^{2}FLY is enough to achieve nearly a 2× latency reduction at 50% weight sparsity.

IV. Multi-Chiplet McPAL Scaling

A. 3D Vertical McPAL Scaling with Hierarchical HBM-PIM

To overcome the monolithic limitations of a single PIM chip, we first realize the sparsity-aware McPAL PIM and scale it vertically using 3D HBM technology. Fig. 7 gives the integration of the McPAL process engine (PE) into the HBM architecture. McPAL includes four DRAM dies and a logic die, all 3D stacked via TSVs. Each DRAM die contains four pseudo channels (pCH) that operate independently, with each pCH comprising four bank groups (BGs). BG is further divided into four banks and every two banks share

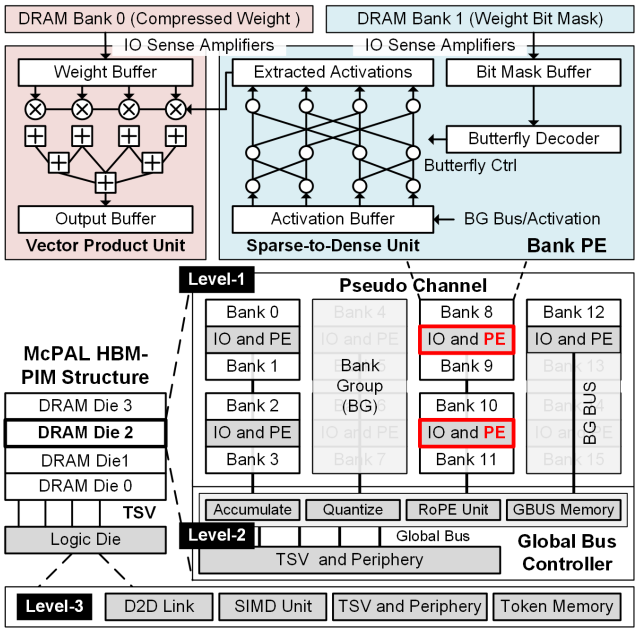

Fig. 7. 3D vertical McPAL scaling with hierarchical HBM-PIM.

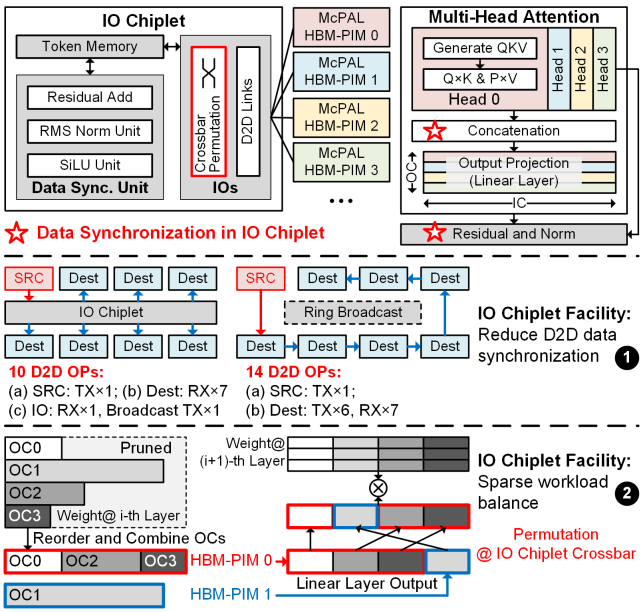

Fig. 8. 2.5D horizontal McPAL scaling with IO chiplet.

one McPAL PE. Each McPAL PE can execute 32 INT-8 MACs in parallel, leveraging the 256-bit local bank I/O. Once McPAL issues a sparse PIM operation, the PE reads 32 NZWs and a 128-bit bitmask from two adjacent banks via I/O sense amplifiers, while activations are delivered from the logic die through TSVs, global bus and BG bus. The bitmask is decoded to control a 128×2-input-32-output DB^{2}FLY, which dispatches the required activations to NZWs. For circuit implementation, the embedded MACs and transmission-gate-based BFLY switches are derived from IMPERA [9].

FGDRAM [27] reveals that data transfer from BGs to the logic die via TSVs accounts for 56.6% of the HBM power consumption. To minimize long-distance travel via TSVs, we strategically organize the hardware hierarchy within HBMs to align with the distinct operational characteristics in LLMs. Besides the bank-level PEs, McPAL has two other hierarchical hardwares in the global bus controller (GBUS CTRL) of the DRAM die and the logic die. For linear layers, the activations from the logic die are broadcast to multiple BGs, which then process different weight output channels in parallel. For self-attention, the KV cache is stored in DRAM dies. Different BGs reuse the same Q to calculate Q×K or P×V for various KV vectors. This approach enables McPAL to fully leverage the abundant internal BGs of HBMs for vector-matrix multiplication, maximizing PIM utilization during the LLM decode phase. As shown in Fig. 7, the partial sums are accumulated and quantized in the GBUS CTRL per BG before writing back to the logic die through TSVs. It consumes less power as partial sums travel a shorter distance. The GBUS CTRL also includes a memory to buffer activations from the logic die and a RoPE unit for positional embedding. The activations are reused locally to avoid direct access to token memory in the logic die, while the RoPE unit locally embeds the positional information into the QK vectors. Then the prepared KV vectors are directly written back to the DRAM banks. Therefore, the GBUS CRTL further minimizes the data transfer through TSVs, improving the power efficiency of McPAL HBM. Finally, we place a 128-way SIMD in the logic die to compute softmax, where the exponential and division are approximated by Taylor expansion. That's because the softmax collects Q×K results from all BGs. Placing softmax at the GBUS or bank level would involve complex transfers among pCHs. Compared to HAIMA-like [11] HBM-PIM, which assigns matrix multiplication to the bank-level PE and allocates all other operations in the host logic die, the hierarchical McPAL reduces the power consumption by 34.3%.

B. 2.5D Horizontal McPAL Scaling with IO Chiplet

We further scale McPAL HBM-PIM horizontally to fit larger LLMs using an IO chiplet. The IO chiplet minimizes D2D communication during data synchronization and ensures balanced sparse workload across various McPAL devices. As shown in Fig. 8, the IO chiplet comprises a token memory, a synchronization unit and abundant IO functionalities. The token memory stores input tokens which are then dispatched to McPAL HBM-PIMs. The synchronization unit receives results from HBM-PIMs, and concurrently performs auxiliary operations in LLMs such as residual addition, RMS normalization or SiLU. Similarly, the non-linear functions are approximated by Taylor expansion. Following tensor parallel [20], McPAL divides self-attention by heads and linear layers by output channels across multiple McPAL HBM-PIMs. Each device independently processes the designated workload segment, contributing to a portion of the entire output. Therefore, output synchronization is essential for subsequent operations. For instance, the attention results from different heads must be concatenated before proceeding to the output projection. The synchronization is also required after each linear layer.

In contrast to ring broadcast [19], [20], the IO chiplet can minimize the D2D communication during data synchronization. Take 8 McPAL HBM-PIMs in Fig. 8 as an example. To move data from one source to the others, HBM-PIMs issue one transmission (TX from source) and 7 receptions (RX to destinations), while the IO chiplet receives data from the source and broadcasts it to the destinations. Consequently, the IO chiplet employs 7 distinct RX channels, which encompass the physical layer, D2D adapter and protocol layer, while reusing a single TX channel. Therefore, a total of 10 D2D transmissions are required. In contrast, the ring broadcast necessitates 7 TX and 7 RX transmissions, 1.4× higher than McPAL. In addition, the IO chiplet enables workload balance across different McPAL HBM-PIMs. Due to the unstructured sparsity within McPAL, directly distributing the linear layers to various McPAL devices could result in unbalanced workloads. The HBM-PIM, which is assigned the heaviest workload, dominates the total latency, leading to low utilization of both PIM

TABLE I
HBM CHARACTERISTICS.

HBM Organization	DRAM Die=4, pCHs/Die=4, Banks/pCH=16, Rows/Bank=32k, Row Size=1KB, IOs/pCH=64bits
HBM Timing (ns)	t_{RC}=45, t_{RCD}=16, t_{RAS}=29, t_{CL}=16, t_{RRD}=2, t_{WR}=16, t_{CCD}=2
HBM Energy (pJ)	e_{ACT}=909, $e_{Pre-GSA}$=1.51, $e_{Post-GSA}$=1.17, $e_{I/O}$=0.80

TABLE II
POWER AND AREA CHARACTERISTICS OF MCPAL.

	Module	Parameter	Power (mW)	Area (mm^2)
Bank PE	DB^2FLY	128×2-Input, 32-Output	4.00	27080.34
	Multiplier	INT-8 ×32	5.60	33195.99
	Adder Tree	×1	3.21	19294.95
	Output Buffer	32b×128	3.68	17588.08
GBUS CTRL	Accumulator	INT-16 ×16	5.46	29273.13
	Quantization	×16	14.98	82532.26
	RoPE MAC	INT-16 ×16	11.46	67967.75
	GBUS Memory	64KB ×16	27.23	347042.07
Logic Die	SIMD	INT-8 MACs ×128	35.25	59942.40
	Token Memory	16MB SRAM	108.94	27423405.82
	D2D Link	256Gb/s UCIE	195.94	470831.04
IO Chiplet	SIMD	INT-8 MACs ×128	35.25	59942.40
	Token Memory	16MB SRAM	108.94	27423405.82
	D2D Link	256Gb/s UCIE ×8	1567.48	3766648.32

logic and D2D bandwidth. To address this issue, as shown in Fig. 8, we combine several highly sparse output channels and assign them to McPAL-0, while a single dense channel is assigned to McPAL-1. Note that the reordered outputs are misaligned and thus cannot directly multiply with the weight in the next layer. The IO chiplet utilizes a crossbar that permutes the output during data synchronization, ensuring correct LLM execution.

V. EXPERIMENTAL RESULTS

We developed a McPAL prototype using 28nm technology. The 8GB HBM used in McPAL follows the standard HBM2 protocol. Its hardware characteristic is drawn from FGDRAM [27] and summarized in Table. I. The embedded multiplier and transmission-gate BFLY switches are derived from IMPERA [9], and redesigned with Cadence Virtuoso. The digital modules in McPAL PE and logic die are synthesized using Synopsys Design Compiler and then placed and routed with Synopsys IC Compiler. The post layout provides area and power estimations. Similar to TransPIM [19], we consider the process difference between the CMOS logic and DRAM, where the DRAM process incurs about 50% area overhead to the logic process. We evaluate McPAL across various Llama models (7B, 33B and 70B). To facilitate McPAL execution, these models are pruned to 50% sparsity by Wanda [5] and quantized to INT-8 by SmoothQuant [7]. The perplexities on the WikiText dataset are 7.45, 5.86 and 4.07, respectively, resulting in perplexity losses of 1.77, 1.09 and 0.95 compared to the original models. We run Llama models on Nvidia A100 40GB GPU, serving as the baseline and compare with McPAL. To fit the entire parameters on McPAL, we utilize 1, 4 and 8 McPAL HBM-PIMs to run Llmam-7B, 33B and 70B models, respectively. Finally, **McPAL-S** enables sparse compute, whereas **McPAL-D** stands for the dense counterpart without sparsity.

A. McPAL Performance and Design Space Exploration

1) ***Power, Area and Latency Breakdown****:* Table II reports McPAL specification. To match the column access time of HBM ($t_{CCD} = 2ns$), McPAL PEs and other digital modules run at 500MHz frequency and 1.0V power supply. As described in Sec. IV, every two DRAM banks share a single 32-MAC McPAL PE. Therefore, the total of 32 PEs and the GBUS CTRL occupy a 3.64mm^2 area, incurring 6.8% overhead to the original HBM die (53.15mm^2) [19]. As shown in Table I, each pCH provides a 64-bit data bus to

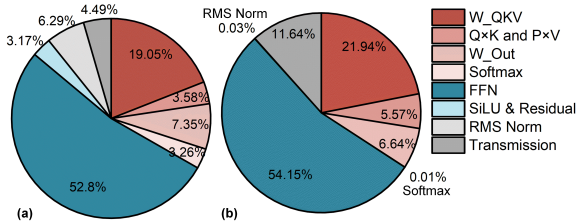

Fig. 9. (a) Latency and (b) Energy breakdown of McPAL with 8 McPAL HBM-PIMs on Llama-70B model.

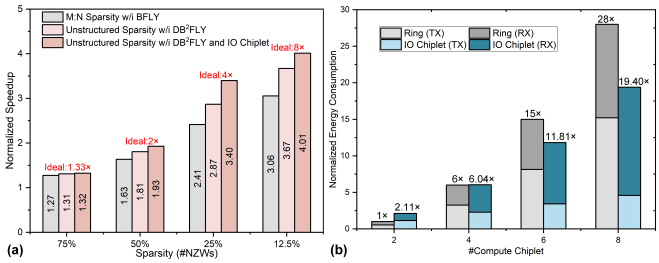

Fig. 10. Design space exploration: (a) The impact of unstructured weight sparsity on McPAL performance, and (b) Transmission energy reduction in IO-chiplet-based McPAL compared to ring broadcast.

transfer 16-bit partial sums. To avoid congestion, only half of the McPAL PEs are activated at the same time. As a result, a single McPAL HBM-PIM yields a peak throughput of 2.048TOPs. Both the logic die and IO chiplet contain a 16MB token memory, capable of storing up to 2K input tokens for the Llama-70B model. Finally, the D2D link adheres to the standard UCIe protocol [26], providing a total transmission speed of 256Gb/s with 16 lanes. Fig. 9 gives the latency and energy breakdown of running the Llama-70B model on 8 McPAL HBM-PIMs with sparsity enabled. The model processes an input sequence of 2K input tokens and sequentially generates an additional 2K tokens. The matrix multiplication in HBM-PIM dominates the inference, accounting for 82.78% latency and 88.31% energy cost. Consequently, the average throughput is 13.56TOPS (82.78%×2.048TOPS×8 McPAL HBM-PIMs). The IO chiplet orchestration reduces D2D communication, which only contributes to 4.49% latency and 11.64% energy consumption. The remainder is for the non-linear functions.

2) ***McPAL Speedup with Sparsity****:* Fig. 10(a) shows the impact of unstructured weight sparsity on McPAL performance, with comparison between BFLY and DB^2FLY. In this experiment, we exclusively compute the feed-forward network of the Llama-70B model. With a lower sparsity where 75% of NZWs are reserved, the BFLY and DB^2FLY shows similar speedup. At a higher sparsity of 50%, BFLY fails to fully exploit the sparsity benefit, resulting in only 1.63× speedup. In contrast, DB^2FLY reduces the data congestion during sparsity decoding and achieves 1.81× speedup. The IO chiplet further enables sparse workload balance across various McPAL HBM-PIMs. As a result, the ultimate McPAL achieves 1.93× speedup, approaching the 2× speedup upper bound. It even achieves 3.40× speedup with only 25% of NZWs preserved. Finally, McPAL struggles to maintain utilization at higher sparsity. Nevertheless, McPAL well supports 50% sparsity, which is the highest compression ratio attainable by the prevalent sparsification algorithms [5], [6] without compromising LLM accuracy.

3) ***IO Chiplet Efficiency****:* We evaluate the efficiency of the IO chiplet for data synchronization and compare it with a baseline, where McPAL HBM-PIMs are organized as a ring [20]. To fit all parameters on fewer McPAL devices, a smaller Llama-7B model is used to generate 2K tokens with another 2K input tokens. Fig. 10(b)

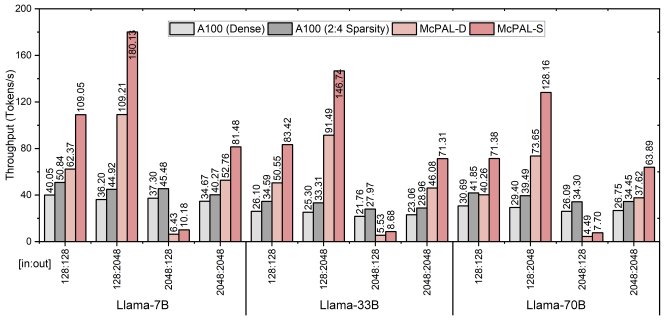

Fig. 11. McPAL throughput improvement over Nvidia A100 GPU. Higher value is better.

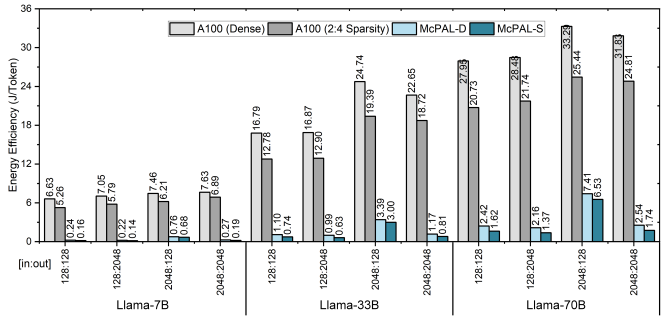

Fig. 12. McPAL energy efficiency improvement over Nvidia A100 GPU. Lower value is better.

reports the normalized energy cost of D2D communication. The ring topology exhibits better energy efficiency when configured with 2 or 4 McPAL HBM-PIMs. It indicates that direct communication among HBM-PIMs is more efficient in such cases. Nevertheless, the IO chiplet proves to be more efficient for coordinating more McPAL HBM-PIMs, making it more compatible with LLM evolution towards large model sizes. Specifically, the IO chiplet consumes 1.27× less energy to coordinate 6 McPAL HBM-PIMs and 1.44× less energy for 8 McPAL HBM-PIMs. As described in Sec. IV-B, the I/O chiplet broadcasts data to all destinations simultaneously, thereby reducing TX transmission requirement with a minimal RX transmission overhead.

B. Comparison to Baseline

1) ***Comparison to Nvidia A100 GPU****:* We first compare McPAL with the Nvidia A100 GPU. To accommodate the entire model on devices, we use 1, 4 and 8 8GB HBM-PIMs to run Llama-7B, 33B and 70B models, respectively, while utilizing 1, 2 and 4 A100 GPUs to run the same models. Even using fewer GPUs, the peak TOPS of a single A100 GPU remains 76.2× higher than that of 8 McPAL HBM-PIMs. Fig. 11 and Fig. 12 show McPAL's improvements in throughput and energy efficiency across various prefill and decode sequence lengths, [in:out]. For throughput comparison, the latency of both prefill and decode phases are considered and divided by the number of output tokens. The approach to calculate energy efficiency is similar. Compared to the A100 GPU, McPAL-D shows an average speedup of 1.57×, 1.95× and 1.35× on Llama-7B, 33B and 70B models, respectively. It also achieves 24.50×, 14.75× and 10.43× energy efficiency. Besides the inherent advantages of HBM-PIM, the proposed hierarchical multi-chiplet scaling method ensures >80% utilization under various workloads, while the A100 GPU achieves only about 3.2% utilization under the same scenarios. By pruning the weights to 50% sparsity, McPAL-S results in further speedup of 2.58×, 3.12× and 2.34×, along with energy efficiency improvements of 35.66×, 21.50× and 15.34×. For a fair comparison, we also explore 2:4 sparsity (described in Sec. III-A) on A100 GPUs, which

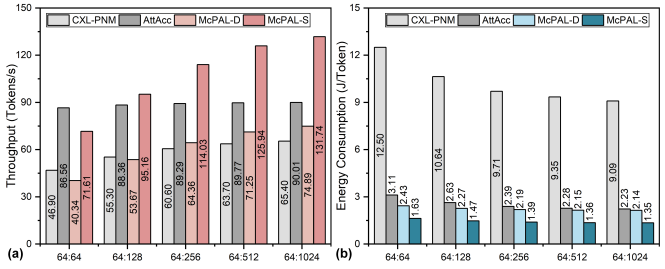

Fig. 13. Comparison with CXL-PNM and AttAcc: (a) Throughput and (b) Energy Consumption. McPAL and AttAcc run Llama-70B model and CXL-PNM runs OPT-65B model.

is inherently supported by Nvidia sparse tensor core. Nevertheless, McPAL-S still achieves 2.10×, 2.39× and 1.76× average speedup, and 29.86×, 16.90× and 11.69× energy efficiency at the same sparsity. Finally, McPAL shows better performance with longer input sequences, but worse performance with fewer input tokens. Take Llama-70B model as an example. Compared to A100 GPU, McPAL-S achieves 3.24× speedup and 15.88× energy efficiency when [in:out] is [128:2048]. That's because the memory-bound decode stage exacerbates the overhead caused by the back-and-forth weight movement in GPUs. In contrast, McPAL-S demonstrates only 0.21× throughput and 3.89× energy efficiency with [2048:128] workload, where the compute-bound prefill phase saturates GPU utilization.

2) ***Comparison to SOTA LLM Accelerators****:* We also compare McPAL with SOTAs LLM accelerators. CXL-PNM [29] is a near-memory architecture based on LPDDR5, utilizing 8 devices to run the OPT-65B model. Each device delivers the same peak throughput as one McPAL HBM-PIM. AttAcc [30] is a hybrid attention accelerator for the Llama-70B model, where A100 GPU handles the linear layer and HBM-PIM processes the self-attention. Fig. 13 compares the performance of McPAL HBM-PIM with CXL-PNM and AttAcc. AttAcc's results are obtained from its open-sourced simulator, while CXL-PNM provides explicit performance. McPAL-D achieves a similar throughput as CXL-PNM, but delivers 4.25× to 5.14× energy efficiency. In addition, McPAL-S further achieves 1.53× to 2.01× speedup and 6.75× to 7.68× energy efficiency. AttAcc demonstrates a speedup of 1.20× to 2.15× over McPAL-D across all workloads, and even a speedup of 1.21× over McPAL-S under [64:64] workload. That's because AttAcc allocates more compute resources, using A100 GPUs and 40 HBM3-PIMs to process LLMs. Nevertheless, McPAL-S still achieves 1.08× to 1.46× speedup with very few compute logic, alongside energy efficiency improvements of 1.65× to 1.91×. The data communication of self-attention between A100 GPU and HBM-PIM negatively impacts the energy efficiency of AttAcc, accounting for about 43.3% of energy consumption. In contrast, McPAL eliminates the need for inter-chip transfer for the KV cache by the three-level hierarchical HBM-PIM design.

VI. Conclusion

The paper introduces McPAL, a novel HBM-PIM-chiplet architecture that enhances sparse compute and multi-chiplet scalability for LLM inference. The sparse matrix decomposition enables unstructured sparse compute in PIM without compromising LLM accuracy or hardware utilization. Further, the chiplet scaling reduces D2D data transferor and enables sparse workload balance. The experimental results show that McPAL achieves 1.57× to 3.12× speedup and 10.43× to 35.66× energy efficiency over Nvidia A100 GPU. Compared to SOTA accelerators, McPAL also achieves 1.08× to 2.15× speedup and 1.65× to 5.14× energy efficiency.

REFERENCES

[1] A. Radford, et al. "Language models are unsupervised multitask learners." OpenAI blog, 2019.

[2] J. Achiam, et al. "Gpt-4 technical report." arXiv preprint arXiv:2303.08774, 2023.

[3] H. Touvron, et al. "Llama 2: Open foundation and fine-tuned chat models." arXiv preprint arXiv:2307.09288, 2023.

[4] A. Gholami, et al. "AI and memory wall." IEEE Micro, 2024.

[5] M. Sun, et al. "A simple and effective pruning approach for large language models." arXiv preprint arXiv:2306.11695, 2023.

[6] E. Frantar and A. Dan. "Sparsegpt: Massive language models can be accurately pruned in one-shot." International Conference on Machine Learning, 2023.

[7] G. Xiao, et al. "Smoothquant: Accurate and efficient post-training quantization for large language models." International Conference on Machine Learning (ICML), 2023.

[8] M. Zhang, et al. "FNM-Trans: Efficient FPGA-based Transformer Architecture with Full N: M Sparsity." ACM/IEEE Design Automation Conference (DAC), 2024.

[9] S. Liu, et al. "A 28nm 53.8 TOPS/W 8b sparse transformer accelerator with in-memory butterfly zero skipper for unstructured-pruned NN and CIM-based local-attention-reusable engine." IEEE International Solid-State Circuits Conference (ISSCC), 2023.

[10] X. Yang, et al. "ReTransformer: ReRAM-based processing-in-memory architecture for transformer acceleration." International Conference on Computer-Aided Design (ICCAD), 2020.

[11] Y. Ding, et al. "HAIMA: A hybrid SRAM and DRAM accelerator-in-memory architecture for transformer." ACM/IEEE Design Automation Conference (DAC), 2023.

[12] Z. Chen, et al. "An in-memory computing accelerator with reconfigurable dataflow for multi-scale vision transformer with hybrid topology." ACM/IEEE Design Automation Conference (DAC), 2024.

[13] F. Tu, et al. "TranCIM: Full-digital bitline-transpose CIM-based sparse transformer accelerator with pipeline/parallel reconfigurable modes." IEEE Journal of Solid-State Circuits (JSSC), 2022.

[14] F. Tu, et al. "Multcim: Digital computing-in-memory-based multimodal transformer accelerator with attention-token-bit hybrid sparsity." IEEE Journal of Solid-State Circuits (JSSC), 2023.

[15] A.F. Laguna, et al. "In-memory computing based accelerator for transformer networks for long sequences." Design, Automation & Test in Europe Conference & Exhibition (DATE), 2021.

[16] G. Li et al. "Ristretto: An atomized processing architecture for sparsity-condensed stream flow in cnn." IEEE/ACM International Symposium on Microarchitecture (MICRO), 2022.

[17] E. Qin, et al. "Sigma: A sparse and irregular gemm accelerator with flexible interconnects for dnn training." IEEE International Symposium on High Performance Computer Architecture (HPCA), 2020.

[18] Z. Li, et al. "Spada: Accelerating sparse matrix multiplication with adaptive dataflow." ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2023.

[19] M. Zhou, et al. "Transpim: A memory-based acceleration via software-hardware co-design for transformer." IEEE International Symposium on High-Performance Computer Architecture (HPCA), 2022.

[20] S. Hong, et al. "Dfx: A low-latency multi-fpga appliance for accelerating transformer-based text generation." IEEE/ACM International Symposium on Microarchitecture (MICRO), 2022.

[21] J. Yue, et al. "An energy-efficient computing-in-memory NN processor with set-associate blockwise sparsity and ping-pong weight update." IEEE Journal of Solid-State Circuits (JSSC), 2023.

[22] C. Chu, et al. "PIM-prune: Fine-grain DCNN pruning for crossbar-based process-in-memory architecture." ACM/IEEE Design Automation Conference (DAC), 2020.

[23] J. Kim, et al. "Z-PIM: A sparsity-aware processing-in-memory architecture with fully variable weight bit-precision for energy-efficient deep neural networks." IEEE Journal of Solid-State Circuits (JSSC), 2021.

[24] J. Cai, et al. "Gemini: Mapping and architecture co-exploration for large-scale DNN chiplet accelerators." IEEE International Symposium on High-Performance Computer Architecture (HPCA), 2024.

[25] Y.S. Shao, et al. "Simba: Scaling deep-learning inference with multi-chip-module-based architecture." IEEE/ACM International Symposium on Microarchitecture (MICRO). 2019.

[26] D. Sharma, et al. "Universal chiplet interconnect express (UCIe): An open industry standard for innovations with chiplets at package level." IEEE Transactions on Components, Packaging and Manufacturing Technology, 2022.

[27] M. O'Connor, et al. "Fine-grained DRAM: Energy-efficient DRAM for extreme bandwidth systems." IEEE/ACM International Symposium on Microarchitecture (MICRO), 2017.

[28] Y. Jin et al. "S3: Increasing GPU utilization during generative inference for higher throughput." Advances in Neural Information Processing Systems (NIPS), 2023.

[29] S. Park, et al. "An LPDDR-based CXL-PNM platform for TCO-efficient inference of transformer-based large language models." IEEE International Symposium on High-Performance Computer Architecture (HPCA), 2024.

[30] J. Park, et al. "AttAcc! Unleashing the power of PIM for batched transformer-based generative model inference." ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2024.