Redefining PIM Architecture with Compact and Power-Efficient Microscaling

Yoonho Jang*, Hyeongjun Cho†, Seokin Hong*
*Department of Electrical and Computer Engineering, Sungkyunkwan University, Suwon, Korea
†Department of Semiconductor Convergence Engineering, Sungkyunkwan University, Suwon, Korea
{jyh8807, cho0624ck, seokin}@skku.edu

2025 International Conference on Electronics, Information, and Communication (ICEIC) | 979-8-3315-1075-6/25/$31.00 ©2025 IEEE | DOI: 10.1109/ICEIC64972.2025.10879742

***Abstract*—With advances in neural network technology, Processing-In-Memory (PIM) has emerged as a solution to performance bottlenecks between processors and memory. Among various PIM design techniques, integrating processing units within memory banks has demonstrated high performance in accelerating neural network models. However, prior architectures based on this approach have required sacrificing a large portion of the cell array to accommodate complex floating-point processing units. In this paper, we propose a new architecture that incorporates a unique quantization technique called microscaling. This technique efficiently converts high-precision data to integer types with minimal accuracy loss during model inference. By adopting microscaling, we reduce the processing unit overhead by replacing floating-point units with integer-based ones. As a result, our approach achieves a 50% reduction in area while maintaining equivalent performance and reduces energy consumption to approximately 70% of that in prior architectures.**

I. Introduction

In the past decade, Deep Neural Networks (DNNs) have become a foundational technique in artificial intelligence, driven by advances in computing performance. As computers process increasingly large datasets at high speeds, DNN models have become complex, containing vast weight parameters. This growth has led to a significant memory traffic bottleneck, where the speed of data movement lags behind the processor's speed, causing performance stalls—commonly known as the memory wall. Addressing this memory wall has become a critical challenge in accelerating DNN workloads across most computing systems.

Processing-In-Memory (PIM) is a promising solution to the memory wall challenge, aiming to reduce data movement. By enabling simple data computations within or near the memory die, PIM minimizes the volume of data transferred to the processors [1, 2]. Leveraging this advantage, leading memory manufacturers such as Samsung and Hynix have developed their own PIM architectures by integrating processing units directly into memory banks [3, 4]. This approach maximizes internal memory bandwidth, making them a state-of-the-art technique for accelerating large DNN models, including Large Language Models (LLMs).

However, both architectures suffer from area and energy consumption due to the need to support floating-point precision. Unlike simpler integer units, floating-point computation units demand many logic gates because of their complex computational steps. Both architectures reduce the memory cell by 50% to place them in the memory bank, which incurs permanent capacity loss [4, 5]. Unfortunately, prior research has not replaced these units with integer-based ones, as many LLM workloads require high-precision computation while classification models can often successfully quantize weight parameters to integer types. Since the goal is to accelerate LLMs, reducing the complexity of the processing units conflicts with these precision requirements.

To address this issue, we leverage a unique quantization technique called microscaling [6]. This technique converts FP32 data into 4 to 8-bit precision, including integer types, without compromising the accuracy of DNN model inference. The quantization process involves a straightforward dataset alignment corresponding its largest value. Modifying the architecture to replace floating-point units with integer types and integrating alignment logic effectively reduces overhead and enables data type conversion without host intervention.

We evaluate our approach by implementing it on the baseline architecture from prior research and conducting a comparative analysis. Our modifications achieve a 50% reduction in area overhead without compromising performance and nearly a 70% reduction in energy consumption.

II. Background & Motivation

A. DRAM organization

It is essential to review the DRAM organization first to understand PIM architectures, as shown in Figure 1. A DRAM device consists of several DRAM dies, with one or more dies forming a memory channel. Each DRAM channel operates independently, using its dedicated I/O bus without relying on other channels. A channel comprises multiple banks, which serve as independent units of operation. Each bank contains a two-dimensional array of memory cells and a hierarchical sense amplifier called the row buffer. Upon receiving a row index, the row decoder activates the corresponding row about row index, transferring its data into the row buffer. The row buffer amplifies the charge of memory cells in an activated row during read operations. It also temporarily stores this data before enabling access to other rows within the same bank. Users can retrieve data from the activated row by selecting columns through the column decoder.

B. PU-based PIM architectures

The PU-based PIM architecture integrates processing units directly within or near memory banks, allowing operands

Authorized licensed use limited to: Sungkyunkwan University. Downloaded on September 02,2025 at 07:49:24 UTC from IEEE Xplore. Restrictions apply.

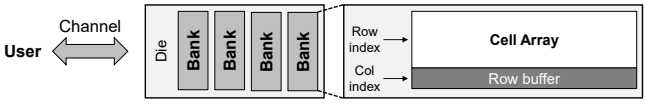


Fig. 1: DRAM Organization

to move from memory cells to the processing units without involving an external I/O bus. This approach significantly reduces data movement latency and boosts throughput by enabling all memory banks to operate concurrently, like SIMD processing. With these advantages, PU-based PIM has become the state-of-the-art architecture for PIM systems, and representative memory manufacturers have adopted it for commercial use, as shown in Figure 2.

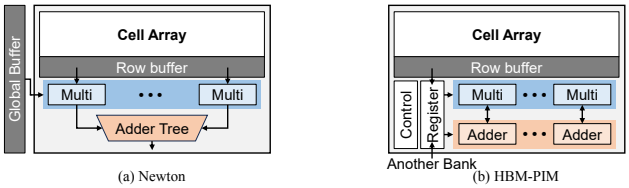


Fig. 2: A bank of PU-based PIM architectures

Newton employs multipliers and an adder tree to accelerate the Multiply-and-Accumulate (MAC) operations fundamental to General Matrix-Vector Multiplication (GEMV). Each memory bank in Newton contains dedicated processing units, which deliver exceptional performance, particularly in speeding up Large Language Models (LLMs). HBM-PIM features a similar architecture to Newton, but pairs a single processing unit—independent multipliers and adders—between two memory banks. These independent processing units enable support for a broader range of operations beyond GEMV, adding flexibility for handling DNN workloads. As a result, both Newton and HBM-PIM effectively accelerate DNN computations.

C. Overhead of Floating Point

Both architectures support floating-point precision operations, which result in considerable area overhead. Many large Language Models (LLMs) use floating-point weight parameters such as FP32, FP16, and BF16. These formats can represent a wide range of values, improving model accuracy during inference. However, as shown in Table I, the Arithmetic Logic Units (ALUs) for these precisions are significantly larger than those for integer types, particularly INT8, which is commonly used in Deep Neural Network (DNN) workloads. Replacing current floating-point units with integer-based units can substantially reduce area overhead.

Unfortunately, replacing floating-point units in PU-based PIM architectures with integer-based units poses significant challenges, primarily due to the accuracy degradation associated with integer quantization. Quantization reduces the precision of weight parameters, thereby lowering the size of weight storage and alleviating hardware overhead in processing units. While many classification models successfully apply this technique with minimal accuracy loss, in LLMs like GPT-3 or Llama-3, quantization is difficult due to the vast number of parameters and layers, where accuracy degradation occurs with each layer, making it impractical. As prior PU-based PIM research aimed to accelerate LLM, this trial does not match their goal.

TABLE I: The gate counts for each ALU and the number of ALUs in a processing unit. We use the Synopsys Design Compiler with a 22nm technology library [7]. The total size of the ALUs within a processing unit is determined by multiplying the gate count by the number of ALUs.

Technique	Newton		HBM-PIM		INT
Precision	BF16		FP16		
Counts	gate	unit	gate	unit	gate
Multi	431	16	695	16	349 (INT8)
Adder	427	32	324	16	82 (INT16)

D. Mircoscaling

We adopt a microscaling quantization technique in both architectures to address this challenge. Algorithm 1 illustrates how this technique reduces the bit size, providing a clearer understanding of the process [6]. The core concept of microscaling is aligning the mantissa with the largest exponential value in the block, a portion of the dataset, called the shared exponent. Since the exponent plays a dominant role in representing values in floating-point formats, this technique preserves the accuracy of high-precision models by focusing on the most significant weight parameters. Microscaling reduces FP32 data to 4 to 8-bit precision, including the MXINT8 type, which converts operands with INT8 type. By utilizing this approach, microscaling enables the replacement of floating-point units with integer units in PU-based PIM architectures, thereby significantly reducing the area overhead associated with processing units.

Algorithm 1 MXINT8 Scaling Algorithm

1: **Input:** Dataset V, block size k
2: $max\_exponential \leftarrow$ exponent of the largest element in the data format
3: $shared\_exp \leftarrow \log_2(\max(V)) - max\_exponential$
4: $X \leftarrow 2^{shared\_exp}$
5: **for** $i \leftarrow 1$ to k **do**
6: $\quad P_i \leftarrow \frac{V_i}{X}$
7: **end for**
8: **Output:** X, P

III. Overview of Re-designing Architecture

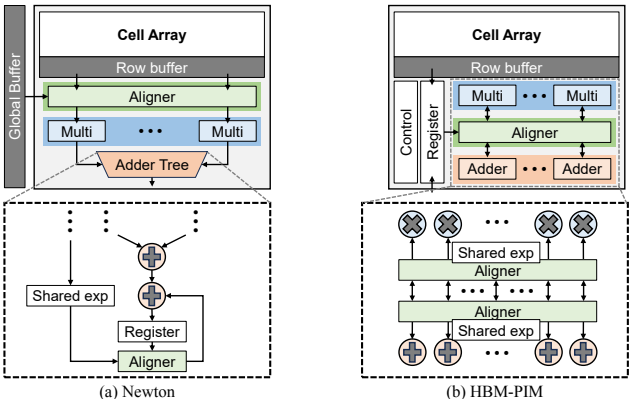
In this section, we present the overall concept of our approach, covering both prior architectures. For clarity, we base our architecture on a 256-bit prefetch size, commonly used in modern memory products such as HBM2, HBM2E, and LPDDR5 [8]. This configuration builds on prior PU-based PIM designs. The choice is straightforward, as it aligns with the configuration of HBM-PIM and shares the same prefetch size that serves as the foundation for Newton [3, 9].

Authorized licensed use limited to: Sungkyunkwan University. Downloaded on September 02,2025 at 07:49:24 UTC from IEEE Xplore. Restrictions apply.

A. Architecture

Figure 3 illustrates the modified architectures incorporating microscaling. As shown in the figure, the design employs an aligner to quantize bit precision from the original type within the memory bank. Further details on this process are provided in the following subsection. By placing the quantization unit before the processing units, they receive operands in MXINT8 format rather than floating-point, allowing both architectures to replace the multiplier and adder with integer-based units effectively.

In Newton, an additional aligner positioned beneath the adder tree converts results back to BF16 precision. The shared exponent the aligner generates passes through the multiplier and adder tree in lock-step with the operands to achieve this. This process ensures that GEMV processing maintains accuracy when handling consecutive columns as separate quantization blocks. In contrast, HBM-PIM uses a dedicated aligner for each computation step. Since HBM-PIM supports simultaneous multiplication and addition during GEMV processing, the aligner must differentiate between new input blocks and the intermediate multiplication results entering the adder. One aligner quantizes the input block, while the other aligns the accumulation's multiplication results and partial sums. Additionally, they back the results of the operation to an original FP16 precision like Newton.



(a) Newton (b) HBM-PIM

Fig. 3: A bank of modified PU-based PIM architectures

B. Microscaling Aligner

For MXINT8 scaling, the microscaling aligner identifies the most prominent exponent in a block of data and shifts the mantissa bits of operands within that block accordingly. We drew inspiration from Newton's adder tree architecture to implement this approach. In Newton, the architecture aligns multiplication results with the most significant value using the exponent bits before starting accumulation while performing actual floating-point operations. This method efficiently reduces the number of accumulation cycles by skipping alignment steps at each accumulation stage, resulting in minimal accuracy loss. Figure 4 shows the modified aligner based on this approach. We set the block size for microscaling to 32 operands, matching the prefetch size (16 times the number of operands in a prefetch). Additionally, Newton reduces the complexity of the original aligner in its adder tree by placing the proposed aligner in front of the multiplier, simplifying hardware and allowing data quantization without external intervention. This optimized design not only enhances processing efficiency but also minimizes hardware complexity.

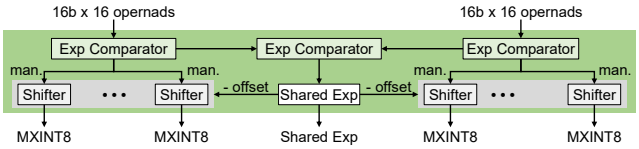


Fig. 4: An architecture of aligner

IV. Evaluation

A. Methodology

We evaluate our approach using a variety of DNN workloads, specifically targeting PIM architectures. We select transformer based models that demand high performance in GEMV processing due to their large number of fully-connected layers. In addition, we include some classification models, such as CNNs, in our evaluation. While CNNs are not primarily memory-intensive, their last layer is fully-connected, making it relevant for our testing. Therefore, we also apply our evaluation to the final layer of these CNN models. The specific size of workloads are provided in Table II.

TABLE II: Workloads

Workload	GEMV size	Type
BERTs1	1K x 1K	Transfomer based models
BERTs2	1K x 4K	
ViT	3072 x 768	
ResNet50	1K x 2K	CNN
VGG16	4K x 4K	

We use Ramulator, an open-source DRAM cycle simulator [10, 11]. We modify the simulator to implement both our benchmark and proposed approach. We select Newton as the baseline for the evaluation benchmark, given its dedicated architecture and significant speed in GEMV processing. Thus, our evaluation is based on the Newton architecture. We use the 8Gb-2000 HBM2 memory configuration for both architectures, as detailed in Table III. We reference the timing parameters and power information for this memory configuration from the JEDEC standard [8].

TABLE III: Configuration

HBM2 Configuration	
Channel & Rank	8 channels & 1 rank per channel
Bank group & Bank	4 per channel & 4 per bank group
Row size & Prefetch	1KB & 256-bit

Authorized licensed use limited to: Sungkyunkwan University. Downloaded on September 02,2025 at 07:49:24 UTC from IEEE Xplore. Restrictions apply.

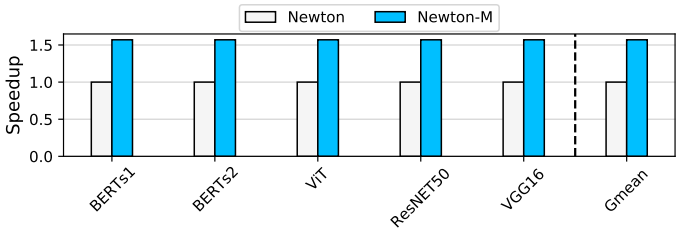


Fig. 5: Normalized speed per area

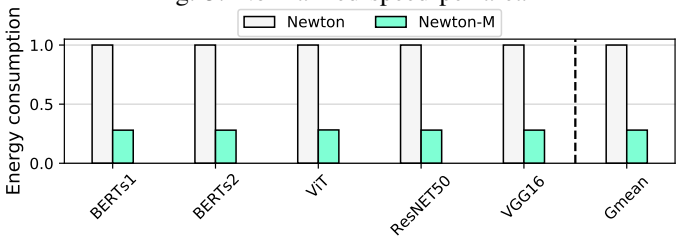


Fig. 6: Normalized energy consumption

B. Performance

Figure 5 presents the evaluation results, showing the ratio of processing time to the processing unit area. We designed and modified the baseline Newton architecture using the Synopsys Design Compiler, detailed in Table I (Although Newton uses BF16 precision, the adder tree consists of INT16 adders, as discussed in Section III-A). As expected, the overall performance remains comparable to the original Newton architecture despite replacing the floating-point units with integer-based units. This is because we assume each processing unit follows the tCCD timing parameter, even though its critical path is slightly faster. Additionally, since the number of ALUs remains the same to match the prefetch size, there is no improvement in throughput. However, thanks to the smaller size of integer units, particularly the adders, our approach reduces the processing unit size by 50%, as illustrated in the figure.

C. Energy Consumption

The simpler design of integer processing units allows for a significant reduction in energy consumption. Figure 6 compares the energy consumption of the processing units in each architecture, normalized against Newton's results. Floating-point multipliers involve complex operations, such as comparing exponent values and using large buffers for mantissa multiplication and shifting logic for alignment. In contrast, integer multiplication only requires fixed-point logic, resulting in lower energy consumption. Although the aligner in our approach consumes slightly more energy than Newton's, the overall energy efficiency of the processing unit improves by nearly 70%.

V. Conclusion

This paper presents a modified PU-based PIM architecture designed to reduce the overhead of processing units. We integrate the microscaling quantization technique, which efficiently converts high-precision data to INT8 with minimal accuracy loss. To implement this technique, we introduce additional hardware that aligns the dataset with the largest exponent value. As a result, our approach reduces the area overhead by 50% and energy consumption by 70% compared to prior architectures.

Acknowledgment

This work was partly supported by the National Research Foundation of Korea (NRF) grant (No.2022R1C1C1012154, 50%), Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No.2021-0-00863, Intelligent in-memory error correction device for high-reliability memory, 25%) and the Competency Development Program for Industry Specialists of the Korean Ministry of Trade, Industry and Energy (MOTIE), operated by Korea Institute for Advancement of Technology (KIAT) (No. P0023704, 25%). Seokin Hong is the corresponding author.

References

[1] S.-S. Park, K. Kim, J. So, J. Jung, J. Lee, K. Woo, N. Kim, Y. Lee, H. Kim, Y. Kwon, J. Kim, J. Lee, Y. Cho, Y. Tai, J. Cho, H. Song, J. H. Ahn, and N. S. Kim, "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)*, 2024, pp. 970–982.

[2] F. Devaux, "The true processing in memory accelerator," in *2019 IEEE Hot Chips 31 Symposium (HCS)*. Los Alamitos, CA, USA: IEEE Computer Society, aug 2019, pp. 1–24. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/HOTCHIPS.2019.8875680

[3] M. He, C. Song, I. Kim, C. Jeong, S. Kim, I. Park, M. Thottethodi, and T. N. Vijaykumar, "Newton: A dram-maker's accelerator-in-memory (aim) architecture for machine learning," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 372–385.

[4] S. Lee, S.-h. Kang, J. Lee, H. Kim, E. Lee, S. Seo, H. Yoon, S. Lee, K. Lim, H. Shin, J. Kim, O. Seongil, A. Iyer, D. Wang, K. Sohn, and N. S. Kim, "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)*, 2021, pp. 43–56.

[5] S. Lee, K. Kim, S. Oh, J. Park, G. Hong, D. Ka, K. Hwang, J. Park, K. Kang, J. Kim, J. Jeon, N. Kim, Y. Kwon, K. Vladimir, W. Shin, J. Won, M. Lee, H. Joo, H. Choi, J. Lee, D. Ko, Y. Jun, K. Cho, I. Kim, C. Song, C. Jeong, D. Kwon, J. Jang, I. Park, J. Chun, and J. Cho, "A 1ynm 1.25v 8gb, 16gb/s/pin gddr6-based accelerator-in-memory supporting 1tflops mac operation and various activation functions for deep-learning applications," in *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 65, 2022, pp. 1–3.

[6] B. D. Rouhani, R. Zhao, A. More, M. Hall, A. Khodamoradi, S. Deng, D. Choudhary, M. Cornea, E. Dellinger, K. Denolf, S. Dusan, V. Elango, M. Golub, A. Heinecke, P. James-Roxby, D. Jani, G. Kolhe, M. Langhammer, A. Li, L. Melnick, M. Mesmakhosroshahi, A. Rodriguez, M. Schulte, R. Shafipour, L. Shao, M. Siu, P. Dubey, P. Micikevicius, M. Naumov, C. Verrilli, R. Wittig, D. Burger, and E. Chung, "Microscaling data formats for deep learning," 2023. [Online]. Available: https://arxiv.org/abs/2310.10537

[7] Synopsys, "Synopsys design compiler," https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/dc-ultra.html.

[8] JEDEC, "High bandwidth memory dram (hbm1, hbm2)," https://www.jedec.org/sites/default/files/docs/JESD235D.pdf, 2021.

[9] J. H. Kim, S.-H. Kang, S. Lee, H. Kim, Y. Ro, S. Lee, D. Wang, J. Choi, J. So, Y. Cho, J. Song, J. Cho, K. Sohn, and N. S. Kim, "Aquabolt-xl hbm2-pim, lpddr5-pim with in-memory processing, and axdimm with acceleration buffer," *IEEE Micro*, vol. 42, no. 3, pp. 20–30, 2022.

[10] S. R. Group, "Ramulator: A dram simulator," https://github.com/CMU-SAFARI/ramulator, 2015.

[11] Y. Kim, W. Yang, and O. Mutlu, "Ramulator: A fast and extensible dram simulator," *IEEE Computer Architecture Letters*, vol. 15, no. 1, pp. 45–49, 2016.

Authorized licensed use limited to: Sungkyunkwan University. Downloaded on September 02,2025 at 07:49:24 UTC from IEEE Xplore. Restrictions apply.