FlutPIM: A Look-up Table-based Processing in Memory Architecture with Floating-point Computation Support for Deep Learning Applications

Purab Ranjan Sutradhar
Department of Computer Engineering
Rochester Institute of Technology
Rochester, New York, USA
ps9525@rit.edu

Sathwika Bavikadi
Department of Electrical and Computer Engineering
George Mason University
Fairfax, Virginia, USA
sbavikad@gmu.edu

Mark Indovina
Department of Electrical and Microelectronic Engineering
Rochester Institute of Technology
Rochester, New York, USA
maieee@rit.edu

Sai Manoj Pudukotai Dinakarrao
Department of Electrical and Computer Engineering
George Mason University
Fairfax, Virginia, USA
spudukot@gmu.edu

Amlan Ganguly
Department of Computer Engineering
Rochester Institute of Technology
Rochester, New York, USA
axgeec@rit.edu

ABSTRACT

Processing-in-Memory (PIM) has shown great potential for a wide range of data-driven applications, especially Deep Learning and AI. However, it is a challenge to facilitate the computational sophistication of a standard processor (*i.e.* CPU or GPU) within the limited scope of a memory chip without contributing significant circuit overheads. To address the challenge, we propose a programmable LUT-based area-efficient PIM architecture capable of performing various low-precision floating point (FP) computations using a novel LUT-oriented operand-decomposition technique. We incorporate such compact computational units within the memory banks in a large count to achieve impressive parallel processing capabilities, up to 4× higher than state-of-the-art FP-capable PIM. Additionally, we adopt a highly-optimized low-precision FP format that maximizes computational performance at a minimal compromise of computational precision, especially for Deep Learning Applications. The overall result is a 17% higher throughput and an impressive 8-20× higher compute Bandwidth/bank compared to the state-of-the-art of in-memory acceleration.

CCS CONCEPTS

• **Computer systems organization** → **Architectures**;

KEYWORDS

Processing in Memory; Floating Point; DRAM; Deep Learning

GLSVLSI '23, June 5–7, 2023, Knoxville, TN, USA.

ACM ISBN 979-8-4007-0125-2/23/06…$15.00
https://doi.org/10.1145/3583781.3590313.

ACM Reference Format:
Purab Ranjan Sutradhar, Sathwika Bavikadi, Mark Indovina, Sai Manoj Pudukotai Dinakarrao, and Amlan Ganguly. 2023. FlutPIM: A Look-up Table-based Processing in Memory Architecture with Floating-point Computation Support for Deep Learning Applications. In *Proceedings of the Great Lakes Symposium on VLSI 2023 (GLSVLSI '23), June 5–7, 2023, Knoxville, TN, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3583781.3590313.

1 INTRODUCTION

For data-driven and highly memory-intensive applications such as Deep Learning, the memory bandwidth limitation of traditional computers has become a major performance bottleneck [2]. Lately, this phenomenon has motivated a dramatic shift of interest toward memory-centric processing architectures. Processing in Memory (PIM) is an emerging memory-centric computing paradigm that enables massively parallel computing at minimal data-movement overheads by integrating computing circuitry within the memory chip itself [7, 8, 16, 18].

Although PIM, to a certain degree, addresses the memory bandwidth bottleneck problem faced by traditional high-performance processors such as Graphical Processing Units (GPUs) and GPU Computing Processors, it has its own challenges. Memory chips are traditionally designed with the goal of minimizing power and design complexity as well as maximizing memory cell density [16]. This essentially limits the level of sophistication of the computing units that can be incorporated within the memory chip to develop a PIM system. Nevertheless, recent commercial PIM prototypes such as the Samsung FIMDRAM [13] and Hynix AiM [7] demonstrate half-precision floating-point (FP) computation capability on a limited number of operations. However, such sophistication comes with the cost of a large area overhead from the computational units. Overall, there is a noticeable trend of PIM computing units becoming increasingly larger in the past couple of years [8, 14, 18].

Such a large footprint of computing units inevitably comes with several disadvantages. First, the number of computing units that can be incorporated into a memory chip is inversely proportional

to the footprint of individual units. For example, the UPMEM PIM, which features large RISC-oriented computing pipelines (DPUs) [16], can currently locate only eight units of these per chip. Second, the larger the computing units are, the lengthier is the effective datapath between the memory cells and the computing unit. Also, a lengthier datapath implies a narrower data bandwidth available to the computing units. For example, the near-bank accelerators, with large computing units, [14, 16, 18] can only access the Bank I/O bandwidth. In contrast, the simplest PIMs with logic-on-bit line processing architecture [11, 12] can expose 'DRAM page(s)' of data at a time to the computing logic located within a bank.

Therefore, a significant design trade-off between the complexity of the computing unit and the operational parallelism of a PIM exists. Nevertheless, due to application demands, a large number of works are gravitating towards sophistication (*e.g.* FP-precision computing) at the expense of underutilized data bandwidth and limited operational parallelism [8, 14, 18]. In this work, we aim to diverge from this trend and propose an alternative approach that enables us to optimize computing unit footprint without sacrificing computational sophistication (*i.e.* FP-computing capability). In order to make this possible, we adopt the following strategies:

Strategy 1: To minimize the processing footprint we introduce a highly efficient, Look-up Table (LUT)-based computing architecture capable of performing FP-precision. While conventional LUT-based processing architectures are prone to contributing significant area overhead [6, 8, 10], we adopt careful architectural and data-path designs to overcome this challenge. The proposed architecture consists of a group of re-writable and interconnected tiny Look-up Tables (LUT) that can be programmed to perform multiple logic/arithmetic operations.

Strategy 2: To minimize the datapath length and widen the effective datapath, the computing units are located within the memory banks and interfaced to the subarray bitlines. These units can access data concurrently from neighboring memory subarrays and collectively access a significantly higher data-bandwidth compared to the near-bank accelerators.

Strategy 3: To optimize the FP computation overheads, we adopt 'tiny float', a highly optimized low-precision FP format that has been demonstrated to achieve comparable accuracy to full-precision FP format for Deep Learning applications at a remarkably lower computing cost [4].

With these goals, we propose **FlutPIM**, a LUT-based flexible PIM architecture featuring within-the-bank computing units that support energy-efficient, low-precision FP arithmetic operations. In this architecture, complex FP operations (*e.g.* multiply-and-accumulate or MAC) are carried out by decomposing the operands into fixed-size sub-operands and then processing these in multiple stages of operation across a group of tiny LUTs. By adopting highly-efficient implementation algorithms that maximize resource utilization and minimize sequential operational stages, we achieve impressive performance gains. The proposed FlutPIM architecture achieves up to 20× higher FP computation performance compared to a state-of-the-art memory-centric accelerator by integrating 4× as many compact, parallel computing units per memory bank.

2 FLUTPIM ARCHITECTURE

In this work, we propose a LUT-based PIM capable of performing low-precision FP and fixed-point computations. An overview of the architecture of the proposed FlutPIM is presented in Figure 1. FlutPIM consists of many computing units/ processing elements (PE) that are integrated within the memory banks and arranged in rows of four between pairs of subarrays. We use the term 'subarray' to refer to a row of memory cell 'tiles' (*i.e.* 1k×1k cells) in the horizontal direction that share the same set of global word lines, and therefore are activated and precharged in lockstep [17]. As shown in Figure 1(a), each quadruplet of PEs, arranged in a row, has access to a shared buffer via a common PE bus. The buffer reads/writes data from/to the primary sense amplifiers (PSA) of the neighboring subarray pair via a common crossbar. Due to the proximity of the buffer to the PSAs of the subarrays, a) the memory access latency can be minimized [4], and b) the crossbar switch can have a significantly wider bit-width than the bank I/0 bus, resulting in reduced communication overhead for memory-bound tasks (*e.g.* matrix-vector (GEMV) and matrix-matrix (GEMM) multiplications).

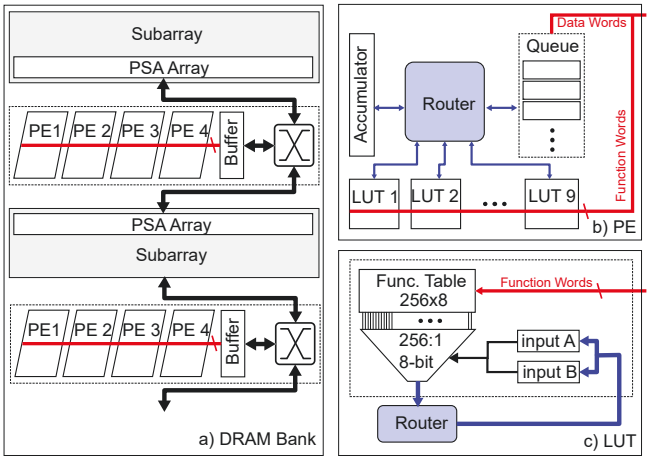

Figure 1: An overview of the FlutPIM architecture, with a)bank-level organization of the PEs, b) architecture of the PE, and c) architecture of a LUT.

Inside each PE, the FP/ fixed-point operations are carried out by employing a group of tiny, identical LUTs programmed to execute various constituent logic/arithmetic operations (*i.e.* multiplication, addition, bit shifting, comparison, increment/decrement, etc.) with a view to collectively performing complex operations in multiple clock stages. As shown in Figure 1(b), a PE contains 9 LUTs, interconnected by a router. Additionally, the PE contains a queue for holding input and output data operands and an internal register called Accumulator for holding intermediate data/ output during computations. The router of the PE is a node-to-node interconnect with 8-bit communication channels, implemented using a crossbar. It can establish concurrent communications as well as multicast data across the LUTs inside the PE.

The LUTs inside a PE are programmable and perform any logic/ arithmetic operation on a pair of 4-bit data or a single 8-bit data. We specifically choose this size of LUT since this combination provides an optimal balance between overhead and computational precision required for the desired application. Figure 1(c) shows the architecture of a LUT which consists of a latch-based 8-bit $2^4 \times 2^4$=256-entry Function Table, indexed with a 256-1 multiplexer. It can be reprogrammed on-the-fly via the PE bus to support different logic/arithmetic operations. By programming the LUTs in

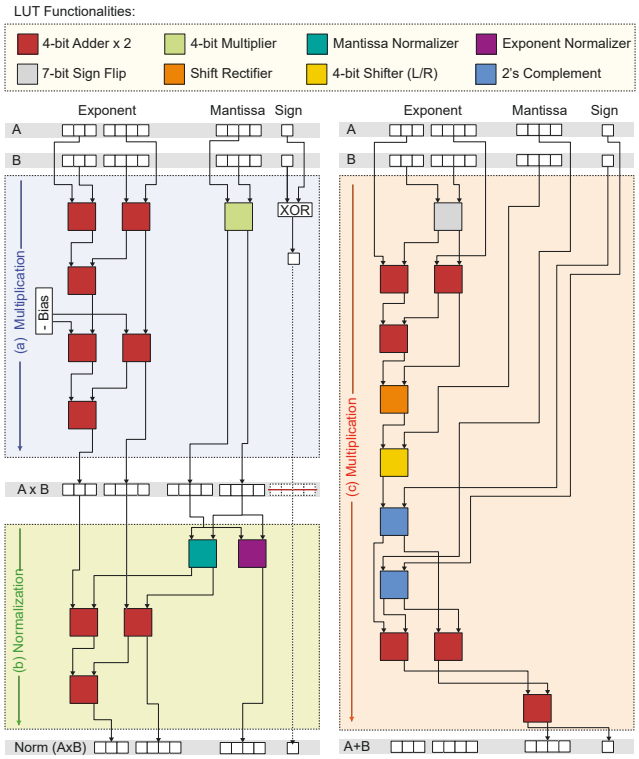

Figure 2: Step-wise execution scheme of different stages of multiply-and-accumulate (MAC) operation with tinyfloat-12 precision: (a) Multiplication of two operands, a and b, (b) Normalization of the product, and (c) Addition of two operands, a and b. The Normalization following the addition has been omitted for brevity.

a PE to perform specific operations and devising step-wise data routing patterns among these LUTs via the router, various FP and fixed-point operations can be implemented, discussed in section 3.

3 OPERATIONS EXECUTION DATAFLOW

The AI training and inferences applications are essentially translated into a set of large matrix and vector multiplications (*i.e.* GEMV and GEMM) consisting of repeated MAC operations. This makes MAC the most frequently appearing arithmetic operation in these workloads. In this section, we present and discuss the implementation of the MAC operation with tinyfloat-12 precision in detail. The proposed PE can also support 8-bit MAC as well as other operations such as max-pooling, average pooling, and ReLU activation via reprogramming of the LUTs.

The tinyfloat-12 consists of a sign bit, a 4-bit unsigned mantissa, and a 7-bit exponent [4]. The MAC operation is performed in four distinct stages: multiplication, normalization of the product, accumulation (*i.e.* recursive addition of products), and normalization of the accumulation. To perform these operations, the LUTs are to be programmed in a specific pattern. The LUT functionalities and the step-wise execution scheme for the MAC operation are shown in Figure 2 and discussed in detail below.

LUT Functionalities: The LUTs operate on a pair of 4-bit data or a single 8-bit data and generate an 8-bit output. The 9 LUTs are programmed to perform the following logic/arithmetic operations in order to perform a tinyfloat-12 MAC operation. For performing multiplication, two Adder LUTs and one Multiplier LUTs are required. Both the Adder and the Multiplier LUTs operate on a pair of 4-bit inputs and generate one 8-bit output. Normalization of a tinyfloat number requires two Adder LUTs, alongside a) a Mantissa Normalizer LUT that shifts the mantissa such that the MSB is a '1', and b) an Exponent Normalizer LUT that adjusts (*i.e.* increments/decrements) the 7-bit exponent corresponding to the adjustment made to the mantissa. For tinyfloat addition, two Adder LUTs are required, along with a) a Sign Flip LUT that changes the sign of the input value by multiplying it by '-1', b) a Shift Rectifier LUT that leverages the difference of exponents of two inputs to generate the shift-value for the mantissa, c) a 4-bit Shifter LUT that utilizes the output of the Shift Rectifier LUT in order to shift the Mantissa by 0-4 bit positions, and d) an 8-bit 2's Complement LUT. The functionalities of the Mantissa Normalizer LUT, and the Exponent Normalizer LUTs are shown in Table 1, and the functionality of the Shift Rectifier LUT is shown in Table 2. The LUT-functionalities are identified with unique color codes in Figure 2.

Table 1: Truth Table for Mantissa and Exponent Normalizer

Mantissa	Normalized Mantissa	Exponent Inc./Dec.
0 0001	1000	0111 1101
0 001x	1x00	0111 1110
0 01xx	1xx0	0111 1111
0 1xxx	1xxx	0000 0000
1 xxxx	1xxx	0000 0001

Multiplication: This step involves the multiplication of both the mantissa and the exponents of the FP operands (A and B), as shown in Figure 2(a). Since the LUTs only operate on 4-bit data, the 7-bit exponents are split into a pair of 4-bit sub-operands and added with the aid of two Adder LUTs. This addition is then adjusted for bias. Meanwhile, the 4-bit mantissa is multiplied using a 4-bit multiplier. The Multiplier LUT is programmed such that it does not generate the four least-significant bits of the product and instead appends the overflow bit to the product in order to aid the normalization stage that follows multiplication.

Normalization: This stage involves appropriate shifting of the mantissa such that the MSB contains a '1'. Therefore, the mantissa undergoes respectively a right-shift and a left-shift for overflow and underflow. This is paired with a corresponding increment/decrement of the exponent, as shown in Figure 2(b). The Exponent Normalizer LUT determines the increment/decrement value for the exponent, which is then added to the original exponent value via a pair of Adder LUTs.

Accumulation: FP addition operation, shown in Figure 2(c), requires the adjustment of exponents. In order to perform this, the exponent of operand B is subtracted from the exponent of operand A. The subtraction is carried out via multiplication by -1, followed by addition. Next, the difference of the exponents is translated into a shift-value by the Shift Rectifier LUT, which is utilized by the 4-bit Shifter LUT to shift the mantissa of B. Both mantissas are then sign-extended via a 2's Complement LUT and added using two Adder LUTs to generate the accumulation. Finally, the accumulation also undergoes the Normalization stage of Figure 2(b).

The multiplication, normalization, and accumulation steps of the tinyfloat-12 MAC operation can be performed together in 19

Table 2: Truth Table for Shift Rectifier

Cases	Shift-Value	Cases	Shift-Value
A-B > 4	4 (Right)	- 4 < A-B < 0	B-A (Left)
0 < A-B < 4	A-B (Right)	A-B < - 4	4 (Left)

Table 3: Attributes of various FlutPIM components

Component	Latency (nS)	Clock (GHz)	Power (mW)	Area (mm^2)
LUT	0.63	1.59	0.93/0.45 (Dyn./Leak)	0.002
PE	11.34 (MAC)	1.59	13.77	0.021
PCU	0.392	1.59	0.079	0.0005

clock cycles. Alongside this, FlutPIM can also support other operations such as max-pooling and ReLU Activation Function. Further, these operations can be also performed with 8-bit fixed-point data-precision without any modification to the PE architecture. However, since similar implementation techniques have already been explored in prior works [9], we only discuss the performance evaluation of the proposed FlutPIM architecture for 8-bit fixed point alongside the 12-bit FP precision computations in Section 5.

4 CONTROL ARCHITECTURE

The PEs within each Bank is controlled by a Bank Controller Unit (BCU) that performs the task of decomposing and scheduling operational instructions and the pertinent data to the PEs across the bank. Alongside, each PE is equipped with a low-level PE Controller Unit (PCU) that decodes and executes operational instructions sent from the BCU. The PCU consists of an instruction decoder and a micro-coded control signal generator that coordinates the dataflow in the I/O registers as well as implements the clock-wise data routing patterns required for performing a specific operation (*e.g.* the MAC operation discussed in section 3).

5 EXPERIMENTAL RESULTS

Experiment Setup: The PEs and the PCU of FlutPIM were designed and verified in the HDL environment. The device parameters are reported in the 20nm technology node and are obtained from ASIC physical synthesis performed using the Synopsys Design Compiler tools. The synthesis was performed in compliance with memory chip specifications, *e.g.* only four metal layers were utilized in the synthesis of the PEs. Various attributes of the PIM components obtained from the synthesis are presented in Table 3.

Hardware Configurations: The PEs of the proposed FlutPIM are to be integrated within the banks of a memory organization which makes it highly modular. Therefore FlutPIM can be incorporated in various memory architectures such as the 2-D Dual Inline Memory Module as well as 3-D stacked platforms such as HMC and HBM/HBM2. The key difference between the DIMM and the 3-D stacked configurations is in the significantly higher global I/O bus bandwidth for the 3-D stacked memory banks which enables faster data relocation within bank groups. We present an overview of the integration of the FlutPIM architecture in the HBM2 memory

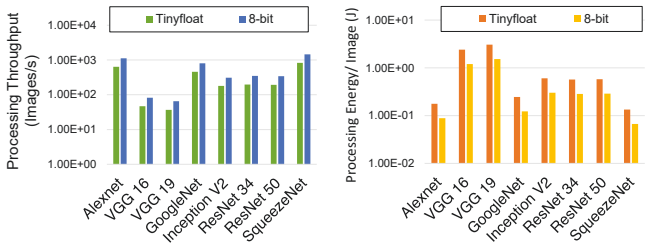

Figure 3: Performance benchmark of the proposed FlutPIM architecture for various CNN algorithms using tinyfloat and 8-bit precisions of computations, presented in terms of a) processing throughput (Images/s) and processing energy for individual images (Joule) of the ImageNet dataset.

Table 4: Specification of the HBM2-based FlutPIM

Attributes	Values
# PE/Bank	4×8=32
# Subarrays/Bank w/PEs	32
Bank Capacity w/PEs (MB)	64
# Bank/Channel	32
# Channels	8
Max. Power/Die (W)	30

organization in Table 4. HBM2 consists of multiple DRAM dies stacked vertically with a base logic die and interconnected via a large number of high-speed TSVs [1]. Similar to Samsung's FIM-DRAM [18], we consider an 8Hi-stack with 4-stacks repurposed as FlutPIM. FlutPIM is distributed across eight memory channels that are further split into a pair of pseudo channels (pCH). Each pCH consists of sixteen banks that are modified to include the PEs of FlutPIM. An arrangement of 32 PEs, along with respective PCUs, collectively occupy roughly half the bank area. Therefore, these PEs are fitted within the bank by replacing half (*i.e* 32 out of 64) of the total subarrays and the associated circuitry.

Application Benchmarking: We benchmark the performance of the proposed FlutPIM for CNN inference application which is the most popular and widely explored application domain for PIM devices [3, 9–12, 15]. Our benchmark consists of a variety of CNN architectures, including the feed-forward CNNs: AlexNet, VGGNets-16,19, the Residual CNNs: ResNets-34,50, CNNs with inception layers: GoogleNet, Inception V3, and a depth-wise separable CNN: SqueezeNet, evaluated for the widely popular ImageNet dataset. For each network, the computation workload for the NN layers is distributed across the PEs inside the banks. This is performed by adopting the weight-stationary approach in which the activation matrices are dispatched to the subarrays that also contain the corresponding pre-trained weight matrices. For executing different NN layers, the activation matrices are relocated across different banks in the pCH via the bank-group bus.

The results of CNN performance bench-marking are presented in Figures 3(a) and (b) which respectively demonstrate the maximum parallel processing throughput (Image/s) and the energy consumption for processing each image for 12-bit tinyfloat and 8-bit fixed-point precision computations. It can be observed that the tiny float-precision causes only minor degradation of performance in comparison to the 8-bit precision performance for all the CNNs.

6 COMPARATIVE EVALUATIONS

Comparison with FP-Capable Accelerators: We compare FlutPIM with FP-supported, 3-D stacked memory-based near-bank accelerators, FIMDRAM [18] and DLUX [8]. DLUX combines LUT-based computing with CMOS logic to support FP computations for AI Training applications while FIMDRAM processing engines contain multiple (*i.e.* 16) FP-pipelines to support diverse application

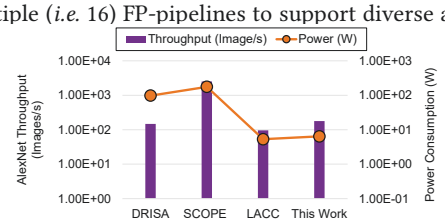

Figure 4: Comparison of proposed FlutPIM with several DRAM-based PIMs for AlexNet acceleration with 8-bit precision of operations and power consumption.

Table 5: Comparison with FIMDRAM

Attributes	FIMDRAM	FlutPIM
PE Area (mm^2)	0.712 (0.045 /ALU)	0.021
# ALU to Bank ratio	8:1	32:1
# Bank/Channel	16	32
Bank Capacity (MB)	128	64
Operational Precision	16-bit FP	12-bit FP
Datapath-width/bank	64b	8×512b
Compute BW (TB/s)	1-1.29	11.5-26
MAC Throughput (TFlops/s)	1.23	1.44

domains. Unlike FlutPIM, the processing engines of these devices are prohibitively large for within-the-bank integration, limiting the datapath between the memory and the processing engine to the width of the bank I/O bus (64-bit). In contrast, the proposed FlutPIM architecture integrates significantly smaller PEs within the banks in large count. For example, the FlutPIM PEs are respectively 33.9× and 19.7× smaller than the FIMDRAM and DLUX processing engines respectively. Also, the FlutPIM PEs can access data directly from the subarray PSAs within the bank via local bitlines. This significantly cuts down the data access energy (*i.e.* 90%) [5] compared to the near-bank accelerators. Moreover, it opens up a remarkably wider aggregated datapath between the memory and the PEs, as demonstrated in the side-by-side comparison of FlutPIM with FIMDRAM in Table 5. Table 5 also shows that FlutPIM fits a larger number of PEs/bank and offers higher operational parallelism.

Comparison with Low-Precision PIMs: We compare the performance of the proposed FlutPIM with several DIMM-based PIM accelerators [15] such as DRISA [11], SCOPE [12], and LACC [10] for 8-bit fixed-point precision computing performances. DRISA and SCOPE are bitwise parallel processors with very high parallel throughput while LACC is a LUT-based accelerator that repurposes memory subarrays for performing multiplications. As presented in Table 6, this comparison shows that the proposed FlutPIM is the fastest at MAC operation, at only 11.34ns of latency, and also has the second lowest power consumption. Therefore, this is significantly more energy-efficient at computing than the bitwise processor PIMS (*i.e.* DRISA and SCOPE) and similarly efficient as LACC. Also, unlike LACC, FlutPIM does not re-purpose the memory subarrays for computing and therefore has higher on-chip memory capacity which is essential for executing large-scale, data-intensive applications such as CNN.

Figure 4 presents a performance comparison of the proposed FlutPIM with the aforementioned PIMs for the inferences of standard AlexNet on ImageNet with 8-bit precision in Figure 4 [10]. For a fairer comparison, we consider a single DIMM DRAM chip with 32 banks for each PIM. It can be observed that both of the LUT-based PIMs are very power efficient. Also, FlutPIM respectively offers 18.6×, 1.95×, and 1.55× higher energy-efficiency for equivalent computing performance than DRISA, SCOPE, and LACC.

Table 6: Comparison of PE attributes of DRAM-based PIMs

Device	#PEs	PE Area (mm^2)	PE Power (W)	MAC **Delay**(ns)
DRISA-3T1C	32768	0.001	0.003	1768
DRISA-1T1C-NOR	16384	0.0025	0.006	2110
SCOPE-Vanilla	65536	0.0028	0.0026	56
SCOPE-H2D	65536	0.0029	0.0026	200
LACC	16384	– – –	0.0003	231
FlutPIM	8192	0.021	0.014	11.34

7 CONCLUSIONS

In this work, we present a PIM architecture that supports floating point (FP)-precision computations within the memory chip by leveraging a group of flexible, tiny, programmable look-up tables (LUT). The proposed computing units are integrated within memory banks to minimize datapath latency and maximize data bandwidth between the computation units and the memory. We also adopt and demonstrate the use of a highly efficient low-precision FP format for achieving optimal computing performance. Overall, the proposed PIM architecture achieves 20× higher maximum compute bandwidth than a state-of-the-art memory-centric accelerator using relatively 2.1× compact equivalent PEs.

ACKNOWLEDGMENT

This work was supported in part by the US National Science Foundation (NSF) CAREER Grant CNS-1553264 and NSF Grant CNS-2228239.

REFERENCES

[1] 2021. High Bandwidth Memory (HBM)DRAM. *JEDEC SOLID STATE TECHNOLOGY ASSOCIATION* (2021).
[2] A. Nowatzyk et al. 1996. Missing the Memory Wall: The Case for Processor/Memory Integration. In *23rd Annual International Symposium on Computer Architecture (ISCA'96)*. 90–90. https://doi.org/10.1109/ISCA.1996.10008
[3] A. Ramanathan et al. 2020. Look-Up Table based Energy Efficient Processing in Cache Support for Neural Network Acceleration. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 88–101.
[4] C. Sudarshan et al. 2022. A Critical Assessment ofnbsp;DRAM-PIM Architectures - Trends, Challenges andnbsp;Solutions. In *Embedded Computer Systems: Architectures, Modeling, and Simulation: 22nd International Conference, SAMOS 2022, Samos, Greece, July 3–7, 2022, Proceedings* (Samos, Greece). 362–379.
[5] C. Sudarshan et al. 2022. Optimization of DRAM based PIM Architecture for Energy-Efficient Deep Neural Network Training. In *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*. 1472–1476.
[6] J. Ferreira et al. 2021. pLUTo: In-DRAM Lookup Tables to Enable Massively Parallel General-Purpose Computation. *CoRR* abs/2104.07699 (2021). arXiv:2104.07699
[7] M. He et al. 2020. Newton: A DRAM-maker's Accelerator-in-Memory (AiM) Architecture for Machine Learning. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 372–385.
[8] P. Gu et al. 2020. DLUX: a LUT-based Near-Bank Accelerator for Data Center Deep Learning Training Workloads. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2020), 1–1.
[9] P. Sutradhar et al. 2020. pPIM: A Programmable Processor-in-Memory Architecture With Precision-Scaling for Deep Learning. *IEEE Computer Architecture Letters* 19, 2 (2020), 118–121.
[10] Q. Deng et al. 2019. LAcc: Exploiting Lookup Table-based Fast and Accurate Vector Multiplication in DRAM-based CNN Accelerator. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*.
[11] S. Li et al. 2017. DRISA: A DRAM-based Reconfigurable In-Situ Accelerator. In *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 288–301.
[12] S. Li et al. 2018. SCOPE: A Stochastic Computing Engine for DRAM-Based In-Situ Accelerator. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 696–709.
[13] S. Lee et al. 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.
[14] S. Lee et al. 2022. 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. 1–3.
[15] S. Shivanandamurthy et al. 2021. ATRIA: A Bit-Parallel Stochastic Arithmetic Based Accelerator for In-DRAM CNN Processing. In *2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. 200–205.
[16] T. Morgan et al. 2019. Accelerating Compute By Cramming It Into DRAM Memory. https://www.upmem.com/nextplatform-com-2019-10-03-accelerating-compute-by-cramming-it-into-dram/
[17] Y. Kim et al. 2012. A case for exploiting subarray-level parallelism (SALP) in DRAM. In *2012 39th Annual International Symposium on Computer Architecture (ISCA)*. 368–379.
[18] Y. Kwon et al. 2021. 25.4 A 20nm 6GB Function-In-Memory DRAM, Based on HBM2 with a 1.2TFLOPS Programmable Computing Unit Using Bank-Level Parallelism, for Machine Learning Applications. In *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, Vol. 64. 350–352.