A High Throughput, Energy-Efficient Architecture for Variable Precision Computing in DRAM

Gian Singh
Arizona State University
Tempe, AZ, USA
gsingh58@asu.edu

Ayushi Dube
Arizona State University
Tempe, AZ, USA
adube9@asu.edu

Sarma Vrudhula
Arizona State University
Tempe, AZ, USA
svrudhul@asu.edu

***Abstract*—DRAM-based near-memory architectures are recognized for their ability to deliver substantial energy efficiency and throughput to execute data-intensive tasks. However, the inherent limitations regarding area, power, and timing within DRAM allow the integration of only primitive processing elements with limited operations and application support. This paper introduces a near-memory processing architecture based on DRAM featuring a novel computing unit termed the neuron processing element (NPE). NPEs are capable of performing multiple arithmetic, logical, and predicate operations. With a well-defined instruction set, the NPEs can be programmed to support standard data formats for floating point and fixed point precision used in different AI/ML and signal processing applications. They can be dynamically reconfigured to switch operations during run-time without increasing overall latency or power consumption. The NPEs have a small area and power footprint compared to conventional MAC units and other functionally equivalent implementations, making them suitable for integration with DRAM without compromising its organization or timing constraints. Furthermore, this paper shows a substantial improvement in latency and energy consumption compared to prior in-memory architectures and demonstrates the efficacy of the proposed architecture for the acceleration of neural network inference.**

***Index Terms*—Processing-in-Memory, Low-power, Deep Neural Networks, DRAM, Memory Wall, Energy Efficiency.**

I. Introduction

Machine learning (ML) is fast becoming a dominant paradigm of computing in almost every domain. ML algorithms are realized as parametric *function graphs* (i.e., deep neural networks–DNN) in which nodes represent the composition of inner products and non-linear functions, and connections represent function composition. The major internet companies like Google, Meta, Microsoft, and others deploy DNNs with hundreds of billions of parameters, performing trillions of large dimensional matrix operations. Thus, DNNs are often both memory and compute-bound. Consequently, they require massive amounts of memory and large server farms with thousands of high-performance GPU processors. The electricity usage of such server farms is approaching that of whole industries and some nation states, and for such systems to be sustainable, at least one to two orders of magnitude improvements in energy-efficiency are required [1]–[3].

Existing CPU/GPU processors based on the traditional von Neumann processor architecture in which the computation units and main memory (i.e. DRAM) are separated are wholly inadequate [4] for large-scale memory and compute-intensive applications such as ML because the energy required to data transfer between the CPU and DRAM is nearly two orders of magnitude greater than that required to perform computations on a CPU [5]. Processing-in-memory (PIM), in which the computation units are integrated with the DRAM, can eliminate the energy consumption of the data transfers and achieve the required improvement in energy efficiency.

While the concepts of underlying PIM are not new, the recent proliferation of ML has led to a rapid development of PIM architectures. Both SRAM and DRAM have been proposed for the design of PIM architectures. SRAM implementations either utilize analog multiply and accumulate (MAC) with small ADCs (quantization < 4 bits) [6], or bit-serial digital computation [7]. SRAM-based compute-in-memory (SRAM-CIM) is usually implemented in the cache of CPU/GPU of small size (< 100 MB), requiring at least one-time data transfer from an external larger capacity memory such as a DRAM. Thus, for large models, data transfers on the memory channel dominate the energy consumption of the system.

A DRAM-based PIM architecture eliminates all the data movement on the memory channel. A DRAM, with its much larger capacity (> 10 GB), can provide various amounts of parallelism depending on where the compute elements are placed. The closer they are to the memory array, the more available parallelism can be exploited.

In-array architectures modify the memory array itself [8]–[10]. This exploits the maximum available parallelism by performing bit-wise operations on entire rows of banks. Simple logic operations are performed by exploiting the charge-sharing characteristics of the transistors that comprise the memory cells or by modifying the sense amplifiers of the memory banks. These require changes to the memory access protocol and timing, and incurs high delay when performing arithmetic operations. Tools like DRAM bender [11] are used to implement and test In-array PIM architectures on commercial DRAM chips.

Near-array processing places the compute elements just outside the memory array after the bit line sense amplifiers (BLSA) [12], [13]. Here, the compute elements can receive the maximum number of bits (e.g., 8K to 16K) from a memory array. Unfortunately, due to stringent constraints on how much memory capacity can be sacrificed, only compute elements of

This work was supported in part by the NSF I/UCRC IDEAS center and from the NSF grant #2324945.

979-8-3315-3967-2/24/$31.00 ©2024 IEEE

limited functionality (e.g. a full adder as in [12]) can be used.

Near-bank processing places compute elements further away from the memory array after the local I/O circuits. Here the bandwidth is reduced to 64 bits for a DDR memory or 256 bits in the case of 3D-HBM (High Bandwidth Memory) [4], while still sacrificing nearly 50% of the memory capacity. Finally, compute elements can be placed entirely outside the memory chip at the RANK level, as in [14]. This is essentially a CPU and a small DRAM memory on a separate chip. All of these approaches present trade-offs between memory capacity and the amount of parallelism. The near-array design [13] places LUTs and the near-bank design [4], with reduced bandwidth, places 16-bit floating-point MACs–both incurring 50% reduction in memory capacity.

A. Main Contributions

In this paper, we present a PIM design that addresses some of the basic disadvantages of all existing PIM designs. As explained above, the fundamental challenge is to introduce compute elements that have high *compute density*, very low area, and low latency near the memory array so as to access the maximum number of bits in parallel. The low area should allow many compute elements to be deployed near the array to operate in a SIMD mode so as to maximize throughput. With conventional CMOS logic, the requirements of high compute density, low area and latency are conflicting.

We propose a novel solution to this problem by using a new circuit element, referred to as a *configurable neuron* (CN), that can realize complex functions within an ultra-small area. For instance, the carry-out of a 5-input carry-lookahead adder can be realized in an area the size of a single D-flipflop. A small network of CNs forms the basic compute unit that can be configured, without any area or delay penalty, to compute arithmetic, bit-wise logic, and comparison functions. The area and power of a CN is 50% of a functionally equivalent, equi-delay version using conventional logic. A combination of the compute unit and additional logic forms a novel processing element, called NPE (neuron processing element), that can be introduced into a DRAM at the output of each memory array to maximally exploit all the available parallelism.

The proposed PIM architecture supports multiple data formats (integer and floating point) and multiple bit-precision (4, 8, 12, 16, 32 bits) as the instruction set of the NPEs supports all logic, arithmetic, and comparison operations. The proposed architecture achieves on an average 1.78× improvement in throughput against [13], and 2.64× against [4], and 9.27× improvement in energy efficiency as compared to [13]. [4] and [13] are state-of-the-art PIM architectures.

II. Proposed PIM Architecture

A. Top-Level Architecture

Fig. 1(a) shows the top-level architecture of the proposed PIM design. It consists of a High Bandwidth Memory (HBM) cube, which has multiple DRAM chips and a base logic layer connected using the through silicon vias (TSVs) to form a 3D integrated memory with high density. Each DRAM layer in the HBM consists of a collection of 2D memory arrays called banks, as shown in Fig. 1(b). The main innovation of this paper is the design of the Neuron Processing Elements (NPEs) (shown in Fig. 1(c)), which can be interfaced directly to the bit-line sense amplifier arrays of the DRAM banks without interfering with the timing constraints or access protocols of the memory as shown in Fig. 1(b). The array of NPEs, *NPE-Array*, connected to the BLSA operates in a SIMD fashion to enable massive compute parallelism inside the DRAM array. The NPE is the basic computing element that can be configured using the control signals generated by an instruction decoder in the logic layer of the HBM. These signals are broadcast to all the NPEs to operate in a SIMD fashion. NPEs perform different operations without any delay penalty and without having to include separate units for different functions. The instruction set of the NPE consists of multi-bit carry-lookahead operation, multi-bit logic, and comparison operations, which provide an ability to map any higher-level arithmetic, linear, or non-linear functions (multiplication, pooling, ReLU, etc.) in different data formats (integer and floating point) to the NPE. The NPE-Array enables the parallel execution of the matrix-matrix (MM) and matrix-vector (MV) multiplications which are common to many workloads (DNNs, LLMs, etc.) that drive much of the applications today. The integration of the NPE-Array to the memory banks can be easily adapted to all the 2D (DDR, GDDR, LPDDR, etc.) and 3D (HMC and HBM) DRAM organizations, making such a PIM architecture scalable from edge devices to high-end servers. This paper uses the HBM organization to demonstrate the efficacy of the NPEs for DRAM-based PIM architectures.

TABLE I: Comparison of different Processing Elements/cores used in PIM architectures.

PE Type	Area (20nm)	#PEs/Bank	Relative Throughput /Bank
MAC INT 16 [4] (w/ 48-bit Acc.)	61.77	16	1.00
MAC INT 8 [4] (w/ 48-bit Acc)	27.80	32	2.00
MAC INT 8 [4] (w/ 32-bit Acc)	21.62	32	2.00
MAC FP 16 [4]	81.54	16	1.00
MAC BFLOAT 16 [4]	71.04	16	1.00
NPE-INT 16	1	1024	3.70
NPE-INT 8	1	1024	11.11
NPE-BFLOAT 16	1	1024	4.49

Why is NPE suitable for BLSA integration? The DRAM provides maximum memory parallelism (8K bits) at the array level or at the output of bit-line sense amplifiers (BLSAs). The major constraint in adding logic near BLSA is the high area overhead of the compute elements. Therefore, prior architectures have been able to place only primitive gates near BLSA [12]. Such architectures have high latency for multi-bit arithmetic operations, which makes them unsuitable for many data-intensive applications such as DNNs. On the other hand, several PIM architectures have used multiply and accumulate (MAC) units of different precision to accelerate ML applications inside DRAM [4], [15]. These architectures, however, place the MAC units outside the bank I/O, thus operating on a much lower data width (64 or 256 bits). Furthermore, these

979-8-3315-3967-2/24/$31.00 ©2024 IEEE

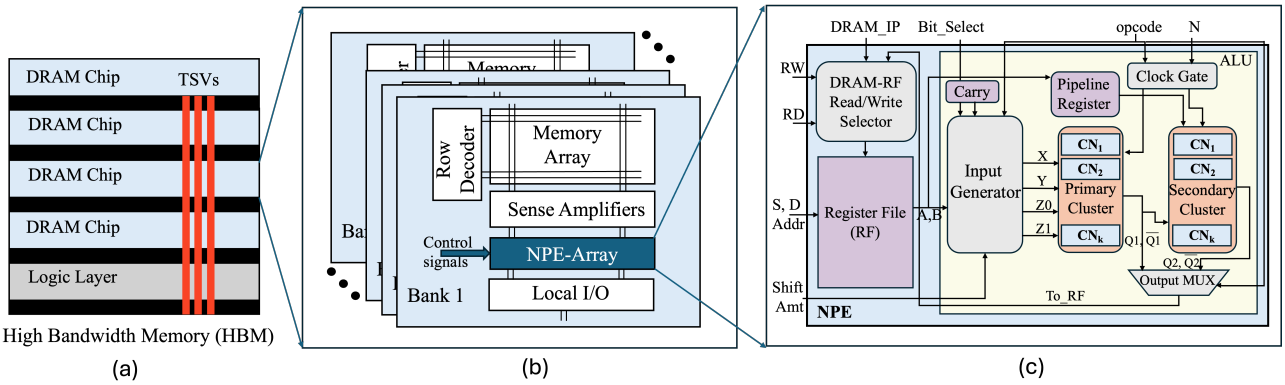

Fig. 1: (a) High Bandwidth Memory (HBM) with multiple DRAM chips; (b) Bank-level description of the proposed PIM architecture showing the proposed NPE array interfaced with a bank; (c) The proposed NPE microarchitecture.

architectures require more hardware for other operations of the DNNs such as pooling and ReLU. McDRAM [15] showed that adding 8-bit MAC units near BLSA in LPDDR4 leads to an area overhead of 120%. Thus, such an architecture is not feasible.

The unique design of the NPE, as explained later in this section, provides the flexibility to perform multi-bit operations in a very low area footprint. Table I provides an area and throughput/bank comparison of NPE and different MAC units [4]. The MAC units are placed outside the DRAM bank with parallel access of 256 bits, and NPEs are placed near BLSA with parallel access of 16K bits [15]. The area advantage of the NPE is clearly evident, and moreover, the NPE can be configured to perform MAC operations of different bit-width, unlike conventional MAC units.

B. Neuron Processing Element (NPE)

Fig. 1(c) shows the major components of the proposed NPE microarchitecture. At the microarchitecture level, the NPE resembles an execution unit of a conventional microprocessor. The NPE is designed to issue a single instruction in every cycle. The three main components of the NPE include **DRAM-RF Read/Write Selector** (referred to as the Selector), **Register File (RF)** with associated circuitry, and the **Arithmetic Logic Unit (ALU)**. The design of ALU is what distinguishes the NPE from the conventional CMOS implementations.

The main computing elements in the ALU are the *Primary Cluster (PC)* and the *Secondary Cluster (SC)*. These clusters consist of k *Configurable Neurons (CNs)* which can perform k-bit operations (for some k) in a single clock cycle. The CNs are the basic compute elements that perform the operations on each bit of the operands. The main operations of the cluster are k-bit carry look ahead, k-bit sum, k-bit comparison, and k-bit logic operations. The cluster can implement all these functions just by using k-CNs. No additional hardware is required for any of the functions. The function on the cluster is selected by simply generating the appropriate set of inputs to the neurons. This leads to a very compact design of the primary and secondary clusters. For instance, for k=5 as used in this paper, in 40nm technology, the CN-based primary cluster is **4**$\times$ smaller than an equivalent CMOS implementation.

The other components of the ALU comprise an Input Generator (referred to as the Generator), a Clock Gate (CG) module, a Carry register, and an output multiplexer (Output_MUX).

The Instruction Set Architecture (ISA): The ISA of the proposed NPE design is formally defined in Fig. 2. It consists of the 11 functions represented by the 4 bits of the opcode field of the instruction encoding as shown in Fig 2. The instructions of the NPE include arithmetic (ADD (addition)), COMP (comparison) and logical (AND, OR, NOT, XOR, XNOR, LADD (addition after logical left shifting second operand), RADD (addition after logical right shifting second operand), RCAR (reset the carry register to zero) and MAND (AND of all bits of first operand with a single bit)).

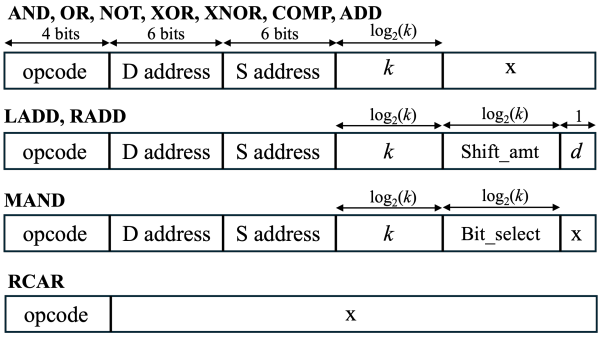

Fig. 2: ISA for the NPE (D and S denote the destination and source register respectively; k is the bit-width of the operands; d is a bit denoting left or right direction; x denotes unused bit(s).

NPE Operation: The primary inputs to the NPE are control signals (*RW and RD*), input data (*DRAM_IP*), and the opcode. The **Selector** interfaces the data from DRAM to the register file (RF) in the NPE. The NPE operates in two modes: *functional mode* and *buffer mode*. In the buffer mode, the NPE either reads data from the DRAM and writes to the RF or writes data back to the DRAM from the RF. In this mode, the ALU of the NPE is clock-gated to save power consumption. In functional mode, the write port of the RF is connected to data coming from the ALU to write the output of the operation performed by the ALU. The control signals *RW and RD* are used to select the mode of the operation of the NPE.

In the ALU, the Input Generator is responsible for formatting the ALU operands A and B based on the opcode such that they can be directly interfaced with the individual inputs of the CNs in the clusters. The function performed by a CN is concisely described by Equation 1 below.

$$Q(p, Z_0, X, Z_1, Y) = Z_0 + \sum_{j=0}^{p-1} 2^j X_j \geq Z_1 + \sum_{j=0}^{p-1} 2^j Y_j \qquad (1)$$

For k-bit ALU operands A and B, the primary and secondary clusters each consist of k CNs and each with $2k+2$ inputs. Therefore, the total number of configuration bits per cluster is $2k^2 + 2k$ (X (k^2 bits), Z0 (k bits), Y (k^2 bits) and Z1 (k bits)) to evaluate the Q function as defined in Equation 1. These configuration bits for different functions are listed in Table II.

TABLE II: Arguments to a Q function for k-bit operations.

operation	p	Z0	X	Z1	Y
AND	1	0	A_k	1	$\sim B_k$
OR	1	0	A_k	0	$\sim B_k$
NOT	1	0	A_k	1	0
COMP	k	$COMP_{out}$	A[k-1:0]	1	B[k-1:0]
ADD	k-1, 2	$Carry_{in}, A_k$	A[k-1:0], B_k	1, 0	$\sim$B[k-1:0], {$Carry_k$,$\sim Carry_{k-1}$}
XOR	2	A_k	B_k	1	{AND_k, 0}
XNOR	2	A_k	B_k	1	{AND_k, 0}
LADD	k-1, 2	$Carry_{in}, A_k$	A[k-1:0], B_k	1, 0	$\sim$B[k-1:0], {$Carry_k$,$\sim Carry_{k-1}$}
RADD	k-1, 2	$Carry_{in}, A_k$	A[k-1:0], B_k	1, 0	$\sim$B[k-1:0], {$Carry_k$,$\sim Carry_{k-1}$}
MAND	1	0	A_k	1	$\sim B_k$
RCAR	0	0	0	0	0

This explains how different functions can be computed by simply setting the appropriate bits. All functions except for SUM and XOR/XNOR need only the primary cluster (PC) for execution. A carry-look-ahead adder is implemented for SUM operation wherein the PC computes the carry bits C_k. The output of PC, $Q1$, and $\overline{Q1}$ are supplied to the secondary cluster (SC) and the Output MUX. The SC evaluates the SUM bits where each of the k bits is computed using one out of k CNs. The outputs to the SC are $Q2$ and $\overline{Q2}$. For the XOR/XNOR operation, the PC evaluates the AND operation between operands A and B, which is used as an intermediate operation to compute XOR/XNOR. A clock gate module operates only the necessary clusters and/or CNs. The output multiplexer (output_MUX) selects the required output among $Q1$, $\overline{Q1}$, $Q2$, and $\overline{Q2}$ as per the opcode.

Values of the inputs to the Q function depend on the operation to be performed, as shown in Table II. For example, consider an AND operation between two 1-bit operands A and B, which can be calculated using Q(1,0,A,1,$\overline{B}$). By substituting the appropriate values into Equation 1, results in $0+A \geq 1+\overline{B}$, which in turn can be rewritten as $A+B \geq 2$. Other k-bit logic operations are similarly defined. They are computed in one cycle using a neuron cluster in an NPE. For $XOR(A_{[i]}, B_{[i]})$, where $i \leq k$, there is a two-level cluster network and therefore requires two cycles.

For N-bit operands ($N > k$), addition, comparison, logic, **multiplication** in **integer or floating point** format can be decomposed into a sequence of k-bit operations, executed sequentially on a single NPE. In this paper, $k = 5$.

The implementation of floating-point operations requires the additional step of *Normalization*. The normalization procedure is dependent on the specific output value of the operation on the input operands which can break the SIMD operation of the NPEs. This paper uses the Normalization algorithm presented in [16] for the SIMD operation. This Normalization procedure depends only upon the bit-width of the operands; thus, the same operations are performed for all the NPEs irrespective of the value of the input operands. The algorithm is mapped onto the ISA of the proposed NPE to be performed at the end of each floating point operation.

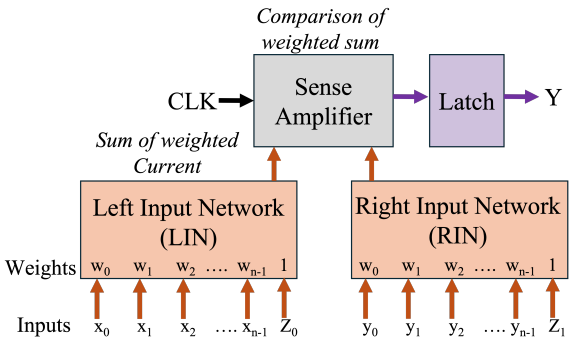

Fig. 3: Configurable Neuron (CN): a mixed-signal circuit implementation based on Threshold Logic (TL) [18].

In summary, the NPE can: (1) implement multiple functions on the same hardware structure, (2) instantaneously switch between the various functions by activating specific inputs of the CN, and (3) utilize the exact number of CNs required in a cluster depending upon the operand bit-width. All of these factors result in extremely low power and a very small area footprint of the NPE.

C. Circuit Implementation of Q (Configurable Neuron)

The Q function in the Equation 1 is realized by a *mixed-signal* circuit called Configurable Neuron (CN) as shown in Fig. 3. The inputs x_i and Z_0 are mapped to the left input network (LIN) of the structure, where each x_i has an associated weight w_i. The w associated with Z_0 is 1. Similarly, the inputs y_i and Z_1 are mapped to the right input network (RIN). For the NPE design $w_i = 2^i$. When the clock is enabled, LIN and RIN evaluate the weighted sum of the inputs in the form of a cumulative current and connect to the Sense Amplifier (SA) as two differential signals, as shown in Fig. 3. The SA evaluates to 1 (0) if the LIN current is greater (lesser) than the RIN and stores the result in the SR latch. A detailed discussion on the threshold functions is provided in [17]. **Note:** The weights w_i in a CN are implemented by programmable resistors, whose resistance values are set after fabrication. The details of the CN's circuit architecture and the realization of the w_i are beyond the scope of this paper. The circuit level details of the CN design are presented in [18]. It is not the main contribution of this paper.

The advantages of a single CN over a functionally equivalent CMOS are substantial [18]. At the individual cell level, a 5-input CN results in improvements in area, power, and delay of $[80\%, 60\%, 40\%]$ respectively, over the performance-optimized, functionally equivalent CMOS circuit.

979-8-3315-3967-2/24/$31.00 ©2024 IEEE

III. Experiments and Results

A. Design and evaluation methodology

The NPE is designed in Verilog and synthesized in 40nm technology using the TSMC standard cell library of the CMOS and the configurable neuron data obtained from the work [18]. The frequency of the NPE is 300 MHz, which is equal to the internal HBM frequency [4]. The area, power, and timing information of the NPEs extracted using the industry standard CAD tools and the HBM are listed in Table III. A behavior-level simulator is designed using the NPE power and timing data along with DRAMPower [19] simulator to characterize the latency and the energy consumption of the proposed and baseline PIM architectures FlutPIM [13] and FIMDRAM [4] for different workloads.

Hardware Configuration: The proposed NPEs can be integrated with memory banks in both 2D-DIMM and 3D-HBM memory. In this paper, the parameters of the HBM are used according to Samsung's HBM2-based PIM industrial product [4]. The PIM-HBM2 consists of 8 DRAM dies stacked using the TSVs. The memory bus frequency of the PIM-HBM2 is 1.2 GHz, and the operating frequency of the DRAM is 4× slower than the bus frequency (= 300 MHz). In Samsung's PIM-HBM2, half of the DRAM dies consist of compute elements, and the capacity of each DRAM die with compute elements is equal to 4 Gb. Therefore, this paper simulates a 4 Gb DRAM die with the bank organization and the timing parameters as listed in Table III. The area of each DRAM die is 84.4 mm^2 at 20nm technology [4] and therefore, the area overhead of the proposed design with 16384 NPEs/chip is only **10.6%**. All the presented results of the proposed and the baseline architecture are based on the workload mapping and simulation on a single DRAM chip.

TABLE III: Configuration of the platforms used.

Attributes	FIMDRAM [4]	FlutPIM [13]	This Work
PE Area (20 nm) in mm^2	0.045	0.021	0.00055 (0.0026 in 40 nm)
PE Power (mW)	N.A.	14	0.051
PE Operating Freq	300 MHz	1.69 GHz	300 MHz
PIM HBM Freq	1.2 GHz	1.2 GHz	1.2 GHz
PIM HBM Die Capacity	4 Gb	4 Gb	4 Gb
#PEs/bank	16	32	1024
#Banks/die	16	16	16
MAC Latency (Tinyfloat-12)	N.A.	19 Cycles	67 Cycles
MAC Latency (INT-8)	N.A.	6 Cycles	33 Cycles
MAC Latency (FP-16)	2 Cycles	N.A.	96 Cycles
Data-width/bank	256 bits	4096 bits	8192 bits
PIM HBM timing	tRAS = 29, tRP = 14, tRCD = 16, tCCD_S = 4, tCCD_L = 2, tWR = 16, tRC = 45, tRRD = 2		

B. Workloads

This paper evaluates the performance and energy efficiency of the proposed PIM architecture on CNN inference workloads. CNN inference is a popular workload for PIM architectures as it requires high computing and data parallelism, which is available in PIM designs. Furthermore, the CNNs are executed at different data precision in both integer (INT) and floating-point (FP) data formats. Hence, the CNN workloads are ideal for testing the efficacy of the proposed PIM architecture, which can be programmed to support multiple data formats. This paper evaluates popular CNN architectures such as ALEXNET [20], RESNET-18, RESNET-50 [21], VGG-16, and VGG-19 [22] on the Imagenet dataset. Fig. 4 shows how the weight matrix (shown in different colors on top of the bank) and input vector V are mapped to the DRAM bank in the proposed architecture for parallel computation of the vector-matrix product, a common operation in CNN inference.

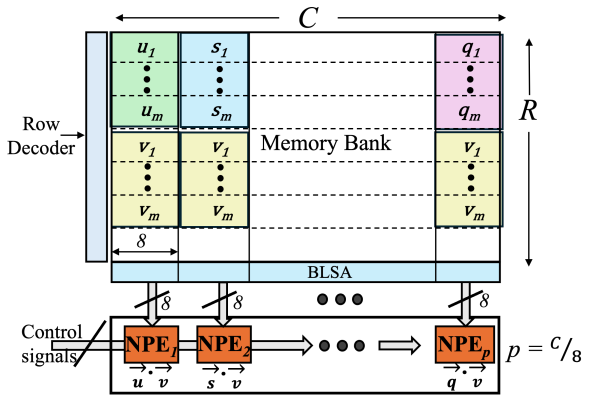

Fig. 4: Mapping of a matrix of weights and activation vectors to a DRAM bank for parallel computation.

C. Results and Discussion

Support For Variable Precision: The instruction set of the NPE consists of basic logic functions, carry-lookahead addition, and comparison operations. Hence, a micro-program consisting of NPE instructions can be designed to implement different arithmetic and logic operations involving various data formats. The proposed architecture's throughput (Images/s) and energy efficiency (Images/J) are evaluated for all workloads at popular integer formats for inference such as INT-4 and INT-8. Popular floating point formats for machine learning, including Tinyfloat (12 bits) and brain floating point (BFLOAT-16), are also evaluated. Fig. 5 shows the throughput and Fig. 6 shows the energy efficiency of the CNN workloads for different data formats. It is observed that as the size (number of parameters) of the CNN architecture increases, both the throughput and energy efficiency of the proposed architecture decrease. The same trend can be observed with respect to the data format and precision and higher precision involves computing more NPE instruction per operation.

High Throughout and Energy Efficiency: Table IV compares throughput and the energy efficiency of the proposed PIM architecture with another LUT-based PIM architecture FlutPIM [13]. The throughput of the proposed architecture is also compared with Samsung FIMDRAM [4]. The processing elements of the FlutPIM consist of multiple lookup tables that

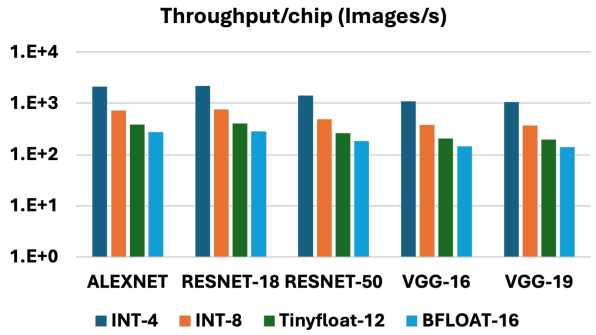

Fig. 5: Demonstrating variable precision. Throughput of the proposed PIM architecture for computing different CNNs at different data formats.

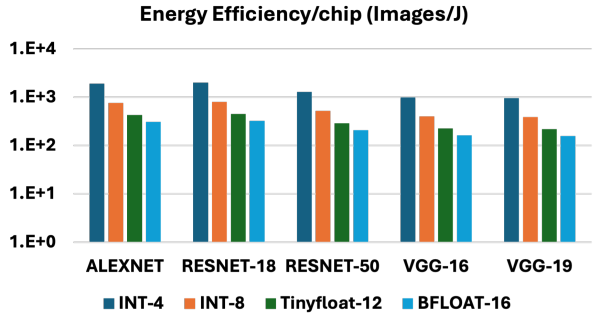

Fig. 6: Demonstrating variable precision. Energy efficiency of the proposed PIM architecture for computing different CNNs at different data formats.

implement 8-bit functions and, therefore, are about **38**× larger than the NPEs used in the proposed PIM architecture when scaled to 20 nm process node. To limit the area overhead of the PEs, only 32 of the FlutPIM PEs are used per bank as opposed to 1024 NPEs. This results in a larger compute bandwidth and compute parallelism per bank for the proposed PIM architecture. As a result, the proposed PIM architecture is able to achieve improvements in throughput over FlutPIM in spite of NPE being slower as compared to FlutPIM PE[1].

The NPEs are compact and do not overprovision hardware to achieve flexibility; hence, they achieve about **274**× lower power than a FlutPIM PE. This results in an order of magnitude improvement in the energy efficiency of the proposed PIM architecture as compared to FlutPIM, as shown in Table IV.

IV. Conclusion

This paper presents a PIM architecture that integrates novel neuron processing elements (NPEs) into a conventional DRAM to utilize the maximum available parallelism inside the memory. The aim of this work is to design a general-purpose computing element with a very small area and power footprint that is, therefore, non-invasive to the DRAM design and makes the PIM architecture easily adaptable. This paper shows the NPE can perform both integer and floating point operations and can be integrated into BLSA to extract maximum memory parallelism. When compared to an existing PIM architecture with floating point support, the proposed PIM architecture achieves about 1.56× to 2.69× higher throughput and about 6.94× to 11.75× higher energy efficiency.

[1]The FlutPIM uses a clock frequency of 1.69 GHz for its PE which is greater than the HBM frequency (300 MHz). This is not a feasible design according to Samsung Industrial PIM product [4]. Any components inside the HBM have to operate at the HBM frequency, or else it will lead to timing violations.

TABLE IV: Throughput of the proposed architecture normalized to FlutPIM [13] and Samsung FIMDRAM [4] and Energy-Efficiency normalized to FlutPIM [13].

	Throughput Ratio			Energy-Efficiency Ratio	
	FlutPIM = 1		FIMDRAM = 1	FlutPIM = 1	
	INT-8	Tinyfloat-12	FP-16	INT-8	Tinyfloat-12
ALEXNET	1.56	2.01	2.55	6.94	11.41
RESNET18	1.56	2.01	2.59	6.97	11.75
RESNET50	1.56	2.01	2.66	6.96	11.43
VGG16	1.56	2.01	2.69	6.95	11.71
VGG19	1.56	2.01	2.69	6.95	11.71

References

[1] C. Lai et al. AI is harming our planet: addressing AI's staggering energy cost. https://numenta.com/blog/2022/05/24/ai-is-harming-our-planet, 2022.
[2] E. Strubell et al. Energy and Policy Considerations for Deep Learning in NLP, 2019.
[3] C. Wu et al. Sustainable AI: Environmental Implications, Challenges and Opportunities, 2021.
[4] S. Lee et al. Hardware architecture and software stack for pim based on commercial dram technology : Industrial product. In *ACM/IEEE ISCA*, 2021.
[5] Mark Horowitz. Computing's energy problem (and what we can do about it). In *IEEE ISSCC*, 2014.
[6] S. Yin et al. Vesti: Energy-Efficient In-Memory Computing Accelerator for Deep Neural Networks. *TVLSI'20*.
[7] H. Kim et al. Colonnade: A Reconfigurable SRAM-Based Digital Bit-Serial Compute-In-Memory Macro for Processing Neural Networks. *IEEE JSSC*, 2021.
[8] V. Seshadri et al. Ambit: in-memory accelerator for bulk bitwise operations using commodity DRAM technology. In *MICRO'17*.
[9] Q. Deng et al. DrAcc: a DRAM based accelerator for accurate CNN inference. In *DAC'18*.
[10] İ. Yüksel et al. Functionally-Complete Boolean Logic in Real DRAM Chips: Experimental Characterization and Analysis. In *IEEE HPCA*, 2024.
[11] A.and others Olgun. DRAM Bender: An Extensible and Versatile FPGA-Based Infrastructure to Easily Test State-of-the-Art DRAM Chips. *IEEE TCAD*, 2023.
[12] S. Li et al. DRISA: a DRAM-based Reconfigurable In-Situ Accelerator. In *IEEE/ACM MICRO'17*.
[13] P.R. Sutradhar et al. FlutPIM: A Look-up Table-based Processing in Memory Architecture with Floating-point Computation Support for Deep Learning Applications. In *ACM GLSVLSI*, 2023.
[14] J. Gomez-Luna et al. Benchmarking a New Paradigm: Experimental Analysis and Characterization of a Real Processing-in-Memory System. *IEEE Access*, 2022.
[15] H. Shin et al. McDRAM: Low Latency and Energy-Efficient Matrix Computations in DRAM. *IEEE TCAD*, 2018.
[16] O. Leitersdorf et al. AritPIM: High-Throughput In-Memory Arithmetic. *IEEE Transactions on Emerging Topics in Computing*, 2023.
[17] S. Muroga. *Threshold logic and its applications*. Wiley-Interscience, New York, 1971.
[18] A. Wagle et al. A Novel ASIC Design Flow Using Weight-Tunable Binary Neurons as Standard Cells. *IEEE TCAS I: Regular Papers*, 2022.
[19] K. Chandrasekar et al. DRAMPower: Open-source DRAM Power and Energy Estimation Tool,. http://www.drampower.info/.
[20] A. Krizhevsky et al. ImageNet Classification with Deep Convolutional Neural Networks. In *NeurIPS*, 2012.
[21] K. He et al. Deep residual learning for image recognition. *CoRR*, 2015.
[22] S. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, 2015.

979-8-3315-3967-2/24/$31.00 ©2024 IEEE