2021 IEEE 39th International Conference on Computer Design (ICCD) | 978-1-6654-3219-1/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICCD53106.2021.00022

# Flexible Instruction Set Architecture for Programmable Look-up Table based Processing-in-Memory

Mark Connolly, Purab Ranjan Sutradhar, Mark Indovina, Amlan Ganguly
Department of Computer Engineering, Rochester Institute of Technology, Rochester, NY, USA
{mfc5867, ps9525, maieee, axgeec}@rit.edu

***Abstract*—Processing in Memory (PIM) is a recent novel computing paradigm that is still in its nascent stage of development. Therefore, there has been an observable lack of standardized and modular Instruction Set Architectures (ISA) for the PIM devices. In this work, we present the design of an ISA which is primarily aimed at a recent programmable Look-up Table (LUT) based PIM architecture. Our ISA performs the three major tasks of i) controlling the flow of data between the memory and the PIM units, ii) reprogramming the LUTs to perform various operations required for a particular application, and iii) executing sequential steps of operation within the PIM device. A microcoded architecture of the Controller/Sequencer unit ensures minimum circuit overhead as well as offers programmability to support any custom operation. We provide a case study of CNN inferences, large matrix multiplications, and bitwise computations on the PIM architecture equipped with our ISA and present performance evaluations based on this setup. We also compare the performances with several other PIM architectures.**

***Index Terms*—Instruction set architecture, microcode, processing in memory, look-up table, convolutional neural network, deep neural network, DRAM**

## I. Introduction

The ever-increasing growth of the processing capability and efficiency of modern processors is strikingly contrasted by the performance of the memory. The memory devices suffer from significant data-access latency and poor energy efficiency which end up causing the major performance bottleneck in the state-of-the-art of computing devices. Moreover, the physical separation between the processor and the memory imposed by the von Neumann Computing model essentially limits the achievable data-transfer bandwidth between these two units. This prevents a processing device from maximizing its performance and thereby creates a 'memory wall' bottleneck [1]. At the same time, the data communication between the processor and the memory chip also accounts for a dominating share of the total power consumption.

The memory wall issue is increasingly motivating investigation into the alternative non-von Neumann computing models. Processing-in-memory (PIM) has lately emerged as a viable solution that minimizes latency and power consumption from the data communication by implementing the processor and the memory inside the same chip. Although PIM is not a new concept [2], it is lately being re-explored under a new light for a versatile range of applications [3].

The earliest PIM works endeavored to implement complete traditional processing units inside the memory chip [4]. However, the complication associated with implementing complex logic blocks inside the memory chip hindered the practicality of such designs. Recent PIMs are integrating processing circuitry with simpler construction deeply within the memory chip. For example, the bit-wise parallel PIM architectures [5]–[8] feature logic gates on the memory bitlines so that the data can undergo processing without leaving the memory subarray inside which it is located. Such a PIM architecture also has to be accompanied by an instruction set architecture (ISA) that enables it to access part or entirety of the memory micro-architecture for performing the operations. This may involve taking over the control of memory row access from the memory controller in order to perform computations [5]. Such a custom ISA also requires an accompanying software interface to receive instructions from a host device [5], [7].

Although most of the recent PIM designs mention an accompanying ISA and software interface [5], [7], there has been an observable lack of a detailed discussion of the ISA architecture in these works. A previous work called 'PIM-enabled Instructions' [9] presented a generalized, architecture-level abstraction of PIM and offered functional access to the PIM device via the memory controller. However, it is not compatible with later PIMs which feature a significantly deeper integration of the logic with the memory micro-architecture [5], [6], [10]. These PIMs require ISA support with significantly finer-grained control over the memory architecture. This calls for an initiative towards custom-designed ISAs targeted at specific PIM architectures so as to capitalize on the strengths and quirks of each particular PIM architecture.

In this work, we present a microcoding-based programmable ISA for a recent PIM architecture called pPIM [11], [12]. Our choice of a micro-coding-based architecture for the ISA is inspired by several factors. First, a microcoding-based Controller unit is fully programmable which allows complete functional flexibility of operations to be performed by the PIM architecture. Moreover, this programmable nature will allow the inclusion of yet unexplored operations and applications on this platform with significant ease that would be not possible with a custom-designed ISA. This flexibility of the proposed micro-coded ISA can be fully utilized by the programmable lookup table (LUT) based pPIM architecture.

Authorized licensed use limited to: Rochester Institute of Technology. Downloaded on March 17,2025 at 18:46:43 UTC from IEEE Xplore. Restrictions apply.

pPIM consists of a parallel processing element (PE) termed as 'clusters' each of which contains several programmable look-up table cores to perform micro-operations. With the aid of a sophisticated routing mechanism that interconnects all the LUT cores inside, each cluster of pPIM is capable of performing more complex operations such as matrix multiplications. We choose pPIM as the platform for our ISA design for four main reasons. First, pPIM is a highly flexible processing architecture with LUTs which can be programmed to perform virtually any logic/arithmetic operation. Second, the performance and energy efficiency of pPIM is impressive thanks to its unique LUT-based computing paradigm. Third, pPIM performs data communication with the adjacent memory subarrays in large batches which allows for a comparatively simpler data organization scheme. Fourth, pPIM clusters feature an internal data-routing mechanism that, along with the programmable LUTs, allows a user to design custom operations. Our microcoding-based ISA is also programmable which makes it highly suitable for operating the pPIM Architecture.

Our ISA supports the baseline pPIM functionalities of Multiplication & Accumulation (MAC) and the ReLU activation filter with a view to performing CNN inferences. We also explore a diverse range of applications, including large matrix multiplications and bitwise logic operations. Our ISA design space leaves room for further expansion in functionality.

## II. Background & Motivation

Bitwise processing PIMs make up the majority of the works in the PIM domain. These PIM architectures perform logic/arithmetic operations on each memory bitline either by charge sharing [13] [14] or by appending logic circuitry on the local sense amplifiers [7], [10]. In order to perform such bitwise operations, multiple memory rows containing the operands are activated simultaneously. Therefore, the ISAs of these PIMs need to facilitate simultaneous activation of multiple memory rows which generally involves the implementation of additional custom row-decoders [13].

The bitwise processing PIMs, however, are not suitable for scaling-up the data-precision of operations. A few works, such as Drisa [5] and DrAcc [6] support operations on larger data-precisions, albeit at the expense of significant area overhead and operational complexity. Several bitwise processing PIM architectures rely on bit-serial computing to support larger data-precision of operations [8], [15], but at the same also suffers from significant operational latency.

LUT-based PIMs [11], [12], [16], on the other hand, are inherently capable of performing operations with larger data granularity. The data look-ups do not require multi-row accesses, unlike the bitwise PIMs. This simplifies the complexity associated with ISA design and the ISA hardware overhead. Therefore, a LUT-based PIM has been chosen over the bitwise PIMs as a platform PIM in this work.

Moreover, the LUT-based PIMs feature individual Processing Elements (PE) that operate in parallel like in a Single-Instruction-Multiple-Data (SIMD) architecture. These PEs are also interconnected via an in-memory communication architecture [11], [12], [16]which enables these to communicate data-operands internally during the operations. Therefore, these PIMs also leverage a mechanism resembling the Systolic Array architectures. However, the amalgamation of all these features into a PIM architecture is a task left for the ISA to perform. Therefore, in this work, we present an ISA for a look-up Table based PIM (pPIM [11], [12]) inspired by the technical aspects from the aforementioned computing models and bring these features together in one single ISA design.

## III. pPIM Architecture

Our ISA design is wrapped around the pPIM architecture. pPIM is suitable for hardware acceleration on data-centric applications, especially AI applications, such as DNNs & CNNs. The architecture of pPIM is presented in Figure 1 in a hierarchical manner.

The top-level element within the pPIM architecture is the pPIM cluster. Figure 1 (a) shows the arrangement of the clusters in a DRAM bank and the architecture of a single cluster is shown in Figure 1 (b). Within the cluster are nine pPIM cores that communicate their output through an all-to-all router architecture. The micro-architecture of this router is shown in Figure 1 (c). The bulk of the processing power of the pPIM architecture is contained within the pPIM core. A core is a reprogrammable component that can facilitate any operation between a pair of 4-bit inputs and produces 8-bit outputs. Figure 1 (d) shows the architecture of a single core. The LUT in a core is implemented with eight 256-to-1 multiplexers, accompanied by eight 256-bit latch/register files. The 8 select bits of the multiplexers are controlled by two 4-bit input registers. The LUT can be reprogrammed by re-writing the latch/register files.

Through the use of multiple pPIM cores, the architecture is able to process more complex functions that a single core alone could not. Complex functions can be broken down into multiple functions that can be handled individually at the pPIM core level. The router allows the orchestration of multistage data-flow schemes to implement such complex operations. A cluster can perform a chain of operations on a pair of 8-bit operands. A 16-bit Accumulator located inside a cluster captures the output of an operation so that it can be re-utilized during the operation of the following cycles if required.

## IV. Instruction Set Architecture

### A. ISA Design

The proposed ISA is primarily designed to drive the pPIM architecture for performing data-intensive applications such as CNN acceleration. For this purpose, we develop a set of instructions that are required for implementing different CNNs layers, *i.e.* Convolutional Layer, Activation Layer, etc. The proposed ISA is depicted in Figure 2. The ISA consists of an Instruction Register/Decoder unit, a group of Pointers, and a Controller/Sequencer unit. Decoded instruction bits are distributed among an Address Pointer, a Core Pointer, Read/Write Pointers, and the Program Counter inside the

Authorized licensed use limited to: Rochester Institute of Technology. Downloaded on March 17,2025 at 18:46:43 UTC from IEEE Xplore. Restrictions apply.

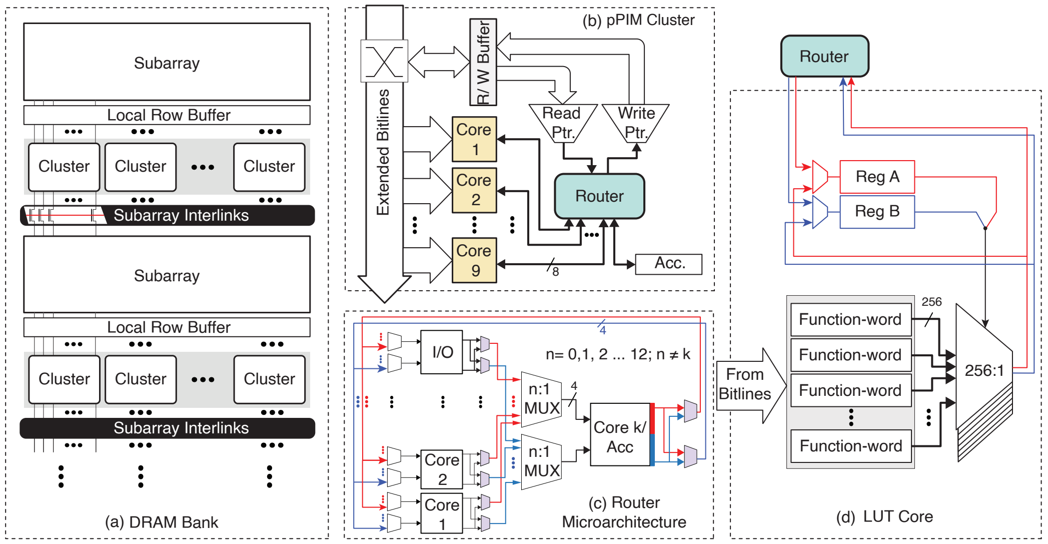

Fig. 1. Hierarchical view of the pPIM architecture including (a) arrangement of pPIM clusters in a DRAM Bank, (b) cluster architecture, (c) cluster router design, and (d) LUT-Core architecture

Controller/Sequencer unit. The Address pointer is used for accessing data-operands from the memory subarray. The Core pointer is used for selecting specific cores inside a cluster during the programming stage of operation. Additionally, the read/write pointers facilitate sequential reads and writes of data-operands from and to the clusters.

We have adopted a microcoding-based implementation of the Controller/Sequencer unit where control signals are generated from a microcode table, in the form of 'control-words'. These control words may perform any or all the following operations: a) programming the core LUTs inside a cluster with newer functionalities, b) routing data-operands among the cores via the router during an operation, and c) reading/writing data from/to the memory subarray.

The choice of a microcoding-based architecture of the Controller/Sequencer unit is inspired by several factors. First, the alternative to microcoded 'control-words' would have been a 'hard-wired' logic-based ISA which does not allow in-situ modification. This would essentially restrict the functional flexibility and scope for functional expansion of the pPIM architecture. Moreover, the adoption of such a logic-based ISA would involve a certain amount of CMOS logic circuitry which is challenging to implement on a DRAM memory chip. In contrast, the microcoding based Controller/Sequencer Unit can be implemented on SRAM, ROM, or non-volatile memory table. In fact, the microcode table for the Controller/Sequencer of our proposed ISA is left with empty slots which can be programmed to support an additional set of instructions that are compatible with the pPIM architecture. The microcoded Controller makes our ISA highly modular and flexible.

## B. ISA Connectivity

The connectivity pattern of the ISA units with the pPIM clusters in a DRAM bank is shown in Figure 3. Only the vertically aligned pPIM clusters are capable of having data communications via the interlinked bitlines. This essentially results in a number of 'Process Threads' consisting of vertically aligned clusters only. Each Process Thread can run in parallel and perform identical operations on entirely different sets of data-operands. This makes a perfect candidate for implementing a SIMD processing model inside a pPIM bank. Therefore, our ISA units are designed to receive a stream of instructions from a host CPU and in turn, drive a group of clusters each of which belongs to a different Process Thread, as shown in Figure 3.

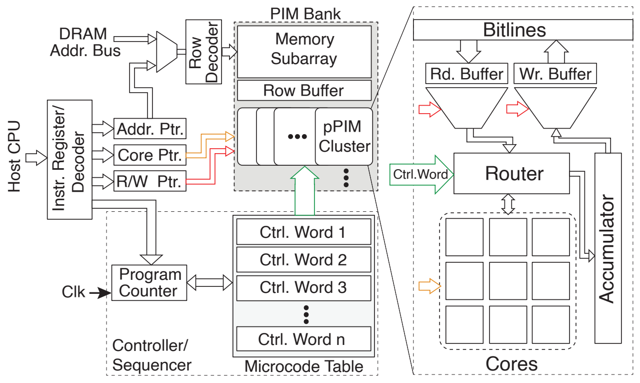

Fig. 2. Proposed instruction set architecture for pPIM

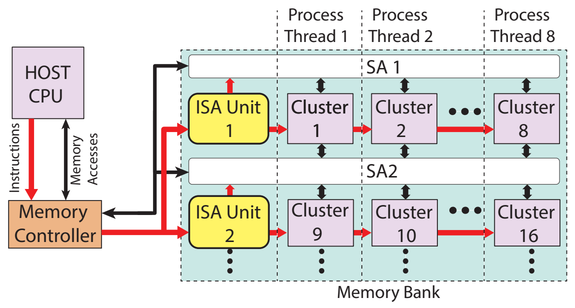

Fig. 3. Interfacing of the proposed ISA with pPIM clusters in a bank and the host CPU

Authorized licensed use limited to: Rochester Institute of Technology. Downloaded on March 17,2025 at 18:46:43 UTC from IEEE Xplore. Restrictions apply.

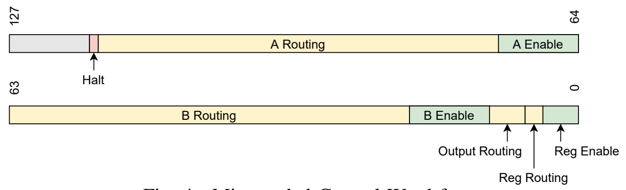

Fig. 4. Microcoded Control-Word format

## C. Controller/Sequencer Design

The Controller/Sequencer unit consists of a Program Counter and a microcode table that drives the control bus. The microcode table is a 2-D array of memory cells where each row contains one complete control-word. The structure of a control-word is shown in Fig. 4. Each control word contains ISA dataflow control signals, routing signals for the pPIM cluster, and register-enable signals for the registers contained within the clusters. A control word also contains a 1-bit 'stop' signal that denotes the end of an instruction. This bit resets the program counter so that it can terminate the ongoing instruction and fetch the next instruction sent from the host CPU. These signals total up to a length of 120 bits or 15 -byte length of a control word.

In its default state, the Program Counter (PC) points to a control word that sets the cluster to an 'idle state' and waits for further instructions from the host CPU. A valid instruction from the host sets the program counter to point at the initiating control word in the microcode table. In the following clock cycles, the program counter progresses through the next consecutive control words in the table, one by one. This progression continues until it reaches a control word with the 'stop' bit set to high. In such a case, the counter is reset to either to the idle state or to the initiating control word. The latter is performed when there are more instructions waiting in the queue to be executed.

## D. Instructions

The pPIM ISA features a fixed-length 24-bit instruction format shown in Figure 5. The instruction-word has two distinct segments: the upper 8-bit segment is dedicated to the execution of different operations while the lower 10-bit segment is dedicated to accessing the memory. The additional 6-bits are left blank for facilitating further expansion of the functionality if required. The most significant 8-bit segment consists of a 2-bit sub-segment for defining the type of instruction, accompanied by a 6-bit pointer value. This segment of the instruction-word can be set to either of the three possible instructions: **'PROG'**, **'EXE'**, and **'END'**. A **PROG** instruction reprograms a core identified by the pointer bits with new functionality. This is done by re-writing the latch/register files in that core. An **EXE** instruction is used for initiating a particular operation inside the cluster. The

| Encoding | Op |
|---|---|
| 00 | NoOp |
| 01 | PROG |
| 10 | EXE |
| 11 | END |

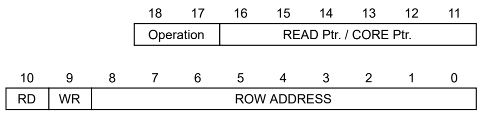

Fig. 5. Instruction-word Format

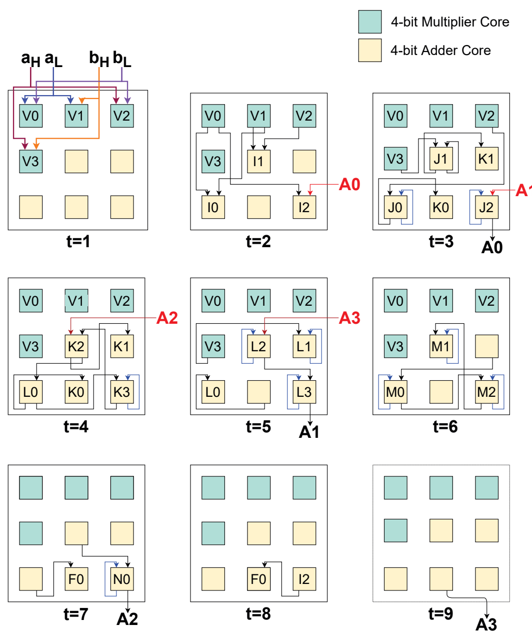

Fig. 6. Sequential model of a MAC operation implemented on the pPIM architecture. The inputs **'a'** and **'b'** indicate input to the cluster from memory with the high and low segments of that memory dictated by the subscripts **'H'** and **'L'**, respectively. Input and output of the accumulator is dictated by A with the segment represented by the following number.

**EXE** instruction causes the Program Counter to jump to a specific control-word location in the microcode table that designates the first of a set of consecutive control-words associated with that operation. During several following clock cycles, the Program Counter increments through a number of consecutive control-words, until it reaches the final one which is an **END** instruction. It designates the end of the ongoing operation by scheduling a Synchronous reset of the Program Counter. The **END** instruction also synchronously resets the computing registers located inside the clusters (excluding the latch/register arrays inside the LUTs).

The lower 10-bits of the instruction are dedicated to managing any sort of memory accesses made by the pPIM clusters. This segment contains a read bit, write-bit, and 9-bit row address pointer for pointing at any of the 512 memory rows in a subarray. By setting the read bit to High, data can be read from the specified row of the subarray, into the read buffer via the bitlines. Conversely, by setting the write bit to High, the contents of the write buffer can be dispatched to a subarray row designated by the 9-bit row address segment of the instruction. In the event that both the read and write bits are high, the read bit is given priority- the data will first be read from the subarray row buffer by the cluster read buffer. Then the data contained in the write buffer of the cluster is written into the memory.

## E. Operation Mapping

Our proposed ISA is capable of orchestrating the routing of data-operands among the cores in a pPIM cluster in several consecutive steps to perform complex operations. We demon-

Authorized licensed use limited to: Rochester Institute of Technology. Downloaded on March 17,2025 at 18:46:43 UTC from IEEE Xplore. Restrictions apply.

TABLE I
MICROCODE SEQUENCES

| Operation | Stage | Control Word |
|---|---|---|
| MAC | 1 | 00 00 00 11 8C 20 0F 00 00 00 4E 53 90 3C 00 |
| | 2 | 00 00 80 60 00 01 50 80 04 02 80 00 05 42 20 |
| | 3 | 30 D2 24 C0 00 01 F0 88 30 0C 00 00 05 42 21 |
| | 4 | 18 00 02 00 00 01 10 C3 50 97 80 00 07 C2 30 |
| | 5 | 30 06 1F 00 00 01 70 40 1D 88 80 00 04 C2 32 |
| | 6 | 18 0C 01 E0 00 01 50 C0 14 0C 00 00 05 42 00 |
| | 7 | 30 C0 00 00 00 01 80 40 00 00 00 00 04 02 04 |
| | 8 | 00 00 00 00 00 00 00 03 80 00 00 00 02 01 C0 |
| | 9 | 80 00 00 00 00 00 00 00 00 00 00 00 00 01 C8 |

strate this capability with the example of an 8-bit unsigned Multiply-and-Accumulate (MAC) operation. The 8-bit MAC operation itself is the most frequently performed operation during the execution of a convolutional or fully connected layer of a CNN. The process begins with programming all the cores with functionalities required for the operation. The programming is performed by transporting eight 256-bit function-words to the latch/register file of a particular core. These function-words cover all the possible outcomes of the 8-bit operation For the 8-bit MAC operation, four cores are programmed as 4-bit multipliers and the other five cores are programmed as 4-bit adders. The upper 4-bits of the output of an adder core contains zero-padded carry-out.

The 8-bit inputs, A & B, of the MAC operation are each split into pairs of 4-bit segments $A_H$, $A_L$ & $B_H$, $B_L$ respectively. Partial products $V_0$-$V_3$ are generated from cross multiplication of these four 4-bit operands:

$$V_0 = a_L * b_L \tag{1}$$
$$V_1 = a_L * b_H \tag{2}$$
$$V_2 = a_H * b_L \tag{3}$$
$$V_3 = a_H * b_H \tag{4}$$

These 8-bit partial products are then aggregated in seven consecutive steps, as shown in Figure6, to perform MAC operation on the input pair. The 8-stage routing pattern shown in Figure 6 is encoded in the eight consecutive control-words dedicated to this operation. Table 1 shows the portion of each of these control words responsible for the routing patterns.

The fetching of the data-operands required for the MAC operation precedes the actual MAC operation. A memory read instruction reads data-operands from the memory row and stores it in the cluster read-buffer. Once the data-operands are available, the execution of the MAC operation is initiated by reading the first control-word of the MAC operation from the microcode table via the control bus. During the next seven clock cycles, the seven following control-words for the MAC operation are read one by one. While performing the MAC operation, the ISA is also able to perform a write-back operation for depositing the results from the previous round of operations. Once finished, the output of the MAC operation is forwarded to the write-buffer from which it is written back into a memory row later on. For the case of a chain of consecutive MAC operations, more memory read requests can be executed prior to a write-request. This protocol is outlined in Figure 7.

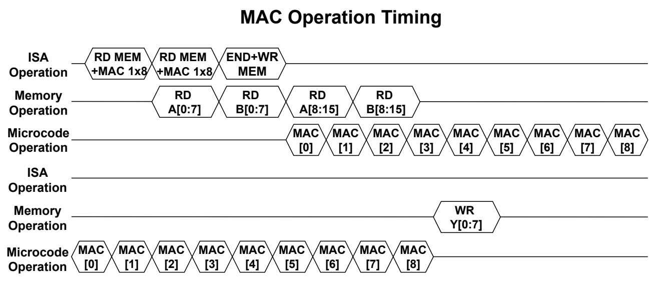

Fig. 7. Instruction protocol for initiating MAC operation in the pPIM clusters

## F. Compatible Operations

We further demonstrated the versatility of the ISA through the mapping of additional operations. Table II lists the number of steps/clock cycles required for various operations. This includes both the microcode sequences and the core configurations for programming the pPIM cores. The size of the microcode sequences is determined by the number of control words within the sequence. The size of the core configurations is determined by the number of unique functions that are used for an operation in a cluster. This demonstrates the ability of this architecture to be reconfigured for not only various applications but also precision scaling.

We also map various linear algebraic operations on the pPIM architecture which are similar to how convolutions are performed for ML applications. In this calculation, each pPIM cluster in the architecture can be dedicated to an output value in the resulting matrix. Therefore the throughput is directly proportional to the number of pPIM cluster engaged in parallel on a particular operation.

## G. Compiler Support

A low-level compiler is required to be integrated the proposed ISA into the host system. The compiler performs two primary functions. First, it converts a program written in a high-level language (*i.e.* Python) into a stream of instruction-words in the format shown in Figure 5. Second, it optimizes the distribution of those instructions among the pPIM clusters in such a way that minimizes the overhead associated with accessing and transporting the data-operands from the memory subarrays. Figure 8 gives an overview of the interaction of such a Compiler with the proposed ISA.

In order to generate the instructions, the Compiler translates the high-level code blocks into corresponding machine codes. These codes are then appended with the addresses to the

TABLE II
PPIM COMPATIBLE OPERATIONS

| Operation | No. of Steps | No. of Different Core Configurations |
|---|---|---|
| Unsigned MAC (8-bit) | 9 | 2 |
| Unsigned MAC (4-bit) | 5 | 2 |
| Signed MAC (8-bit) | 13 | 5 |
| ReLu (16-bit) | 4 | 1 |
| ReLu (8-bit) | 2 | 1 |
| Max Index (16-bit) | 13 | 4 |
| Max Index (8-bit) | 7 | 4 |

Authorized licensed use limited to: Rochester Institute of Technology. Downloaded on March 17,2025 at 18:46:43 UTC from IEEE Xplore. Restrictions apply.

memory rows containing the operands to form the instructions. An optimization algorithm is envisioned that tries to ensure that each instruction is executed in a pPIM cluster located next to the memory subarray that contains the corresponding operands. When such a case is not possible, the data operands are to hop across multiple subarrays via the subarray-interlinks. The compiler sends subarray-interlink controlling bits to the ISA units for this purpose.

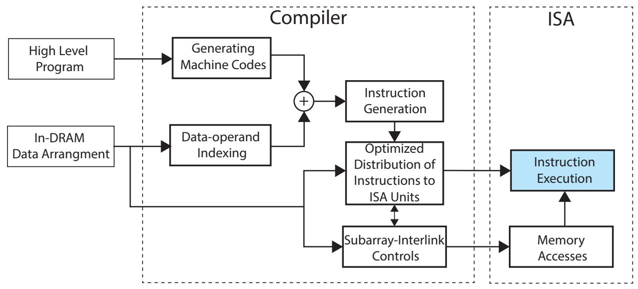

Fig. 8. Overview of the functionality of the proposed Compiler

## V. Results

In this section, we discuss the performance of the ISA in conjunction with the pPIM architecture.

### A. *ISA and pPIM architecture characteristics*

The ISA and pPIM are characterized using post-synthesis models using the Synopsys Design Compiler at the 28nm technology node. The results of the hardware synthesis and the pPIM architecture are outlined in Table III. It can be observed that the area overhead from the ISA is minimal thanks to the microcode-based implementation of the Control Unit. The microcode table can support 128 control-words each of 120 bits and contributes only 15.24 $\mu m^2$ of area overhead. Moreover, since each ISA unit is in charge of multiple (eight) pPIM clusters, the incremental area overhead from the inclusion of ISA is minimal ($<1\%$). The operational clock speed of the ISA is determined by the critical latency of a pPIM Core (0.8ns).

### B. *System-Level Performance Evaluation*

We evaluate the performance of the pPIM architecture equipped with our proposed ISA. This involves a pPIM configuration with eight parallel process threads in a bank [12] where each ISA unit is in charge of eight parallel SIMD clusters. We further scale up the evaluations for configurations with a higher number of clusters (256 & 512) [11], [12]. We evaluate the pPIM ISA for DL applications, Matrix, and vector operations as well as bit-wise logical operations to demonstrate the flexibility of the microcode-based ISA in the next subsections.

TABLE III
Synthesis Results

| Component | Delay (ns) | Power (mW) | Active Area $\mu m^2$ |
|---|---|---|---|
| PIM ISA | 0.549 | 0.155 | 968.16 |
| PIM Core | 0.8 | 2.7 | 4616.85 |
| PIM Cluster (MAC Operation) | 6.4 | 5.2 | 41551.66 |

*1) Performance with DL applications:* The evaluation of the DL applications on the pPIM architecture is performed both for 8-bit and 4-bit fixed point precisions. We compare the performance of the ISA-pPIM setup for CNN inferences with several other computing architectures. This includes high-end computing devices such as Intel Knights Landing server CPU (KNL) and Nvidia Tesla P100, AI accelerators Edge TPU & Intel Neural Compute Stick 2 [17], as well as two contemporary PIM architectures, DRISA [5], and LAcc [16]. Fig. 9 presents the comparison of throughput and power consumption respectively for Alexnet inferences on these devices. It can be observed that the two AI accelerators and the PIM devices, in general, outperform the general-purpose computing architectures of CPU and the GPU by a huge margin both in terms of performance and power efficiency. However, the PIM devices also perform noticeably better than the AI accelerators: TPU and Neural Compute Stick 2, at a similar range of power consumption. This highlights the merit of a PIM- based hardware solution for AI acceleration which effectively eliminates the performance and energy bottleneck from the data communications.

Among the PIM devices, LAcc [16] and pPIM, both of which are Look-up Table (LUT) based architectures, offer very high performances for the least amount of power consumption. DRISA, which is a DRAM-based bitwise processing accelerator, outperforms LAcc and the pPIM-256 configuration, albeit at a significantly higher power consumption rate. pPIM equipped with our ISA achieves slightly better performance at a slightly lower rate of power consumption than LAcc This is possible due to the comparatively more efficient, clustered LUT-based architecture of the pPIM which also enables our ISA to perform massively parallel operation mapping. pPIM achieves nearly double the performance for inferences with 4-bit fixed point precision (pPIM-256A) compared to the 8-bit fixed-point operation mode. We further scaled up the pPIM architecture to 512 clusters (pPIM-512) which achieves a similar level of performance as the pPIM-256A.

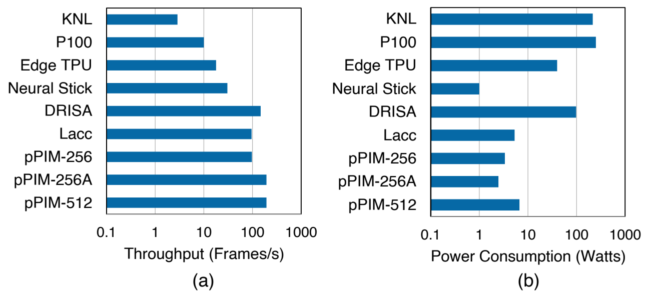

Fig. 9. Comparison of (a) throughput (Frames/s) and (b) power consumption (Watts) of the ISA-equipped pPIM architecture with general purpose processors, AI accelerators and other PIM architectures for AlexNet inferences.

Authorized licensed use limited to: Rochester Institute of Technology. Downloaded on March 17,2025 at 18:46:43 UTC from IEEE Xplore. Restrictions apply.

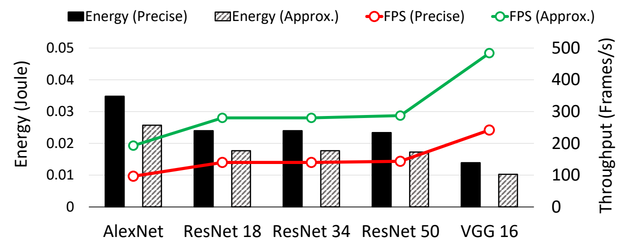

Fig. 10. Inference throughput and total energy consumption for 4-bit and 8-bit precision inferences of several CNN algorithms by the pPIM architecture equipped with the porposed ISA

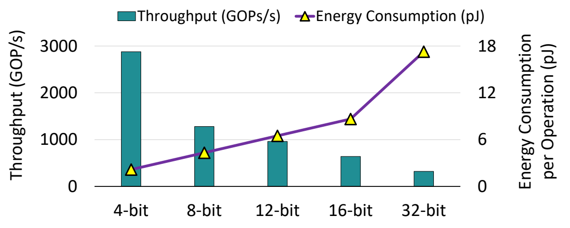

Fig. 12. Evaluation of throughput (GOPs/s) for bitwise logic operations such as AND/NAND, OR/NOR, NOT, XOR/XNOR with different data-precision (4-bit to 32-bit) on 256 pPIM clusters equipped with the proposed ISA, as well as energy consumption (pJ) per each operation.

Alongside AlexNet, we evaluate the performance of the pPIM-256 and pPIM-256A configurations for four other CNN algorithms: ResNet 18, ResNet 34, ResNet 50, and VGG 16. The performance throughput and total energy consumption for these inferences are presented in Figure 10. It can be observed that the maximum inference throughput and the least energy consumption are achieved for the VGG 16 inferences for both setups due to the least computational workload from this algorithm. Overall, the efficient mapping of operations by our ISA across the parallel processing threads inside the pPIM architecture enables us to achieve a superior CNN inference throughput and energy-efficiency from this device than all the other devices in comparison.

*2) Performance with matrix Operations:* The proposed ISA is also capable of implementing linear algebraic operations such as multiplications and additions of large-scale vectors and matrices on the pPIM architecture. For example, large-scale matrix multiplications can be performed by leveraging the same MAC Operation configuration used for CNN inferences, without any further reprogramming of the pPIM cores. We evaluate the performance of the pPIM architecture equipped with the proposed ISA for several linear algebraic operations. Our evaluations include latency of addition and multiplication operations on a wide range of dimension of matrices and vectors with 8-bit and 4-bit data-precision, as shown in Figure 11. Both the matrices (N×N) and the vectors (1× N) are represented on the same axis in terms of the value of N. A step-wise growth of the operational latency with the increase of the dimension of the matrices/ vectors is visible in Figure 11. This can be traced back to the 256 pPIM Cluster configuration that has been used for performing this simulation. Since the PIM can perform up to 256 computations simultaneously at a time, additional calculations only impact the performance of the architecture once it reaches an additional 256 computations.

*3) Performance with bit-wise operations:* The LUT-based architecture of pPIM can support any bit-wise logic operation, *i.e.* bitwise inversion, AND/NAND, OR/NOR, XOR/XNOR, etc. This is performed by programming each LUT-core of a pPIM cluster with an identical set of function-words corresponding to the specific bitwise operation. The programming is performed using a PROG instruction in the proposed ISA. Although these operations have a single-bit granularity, each LUT-core performs these operations on 4-bit segments of operands. By combining operations across multiple LUT-cores in parallel, bitwise operations can be performed on larger data operands. Figure 12 shows performance evaluation of bitwise logic operations in the pPIM architecture with 256-cluster configuration for operands ranging from 4-bit, up to 32-bit precision. For lower-precision of operations such as 4-bit or 8-bit precision, multiple bitwise operations are performed in parallel across the LUT-cores in a cluster for improved throughput. It can be observed that the pPIM architecture can achieve very high throughput (up to 2880 GOPs/s) at low energy consumption (2.16 pJ/OP) for the 4-bit bitwise operations, thanks to the single clock of operational latency, irrespective of how complex the bitwise operation is.

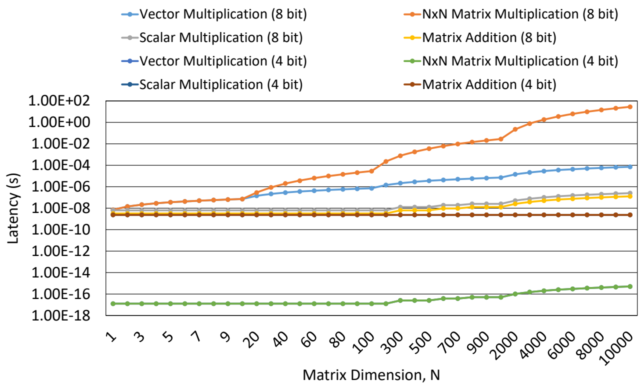

Fig. 11. Evaluation of throughput of calculating various linear algebra operations on the pPIM architecture

## VI. Conclusions

In this paper, we propose an ISA for reconfigurable and programmable PIM architectures through the use of microcode sequences that promotes the adaptability of PIMs. The full capability of the ISA design is demonstrated in cooperation with the pPIM architecture which delivers very high throughput and re-programmability at a low area overhead. We evaluate the ISA by comparing the combined pPIM and ISA power figures against other PIM architectures. The ISA is capable of instructing programmable processing elements at a negligible cost to the performance of the PIM in a small form factor, so as not to detract from the DRAM architecture. The ISA also demonstrates a level of adaptability through the reformatting of microcode control words that can be reassigned to other existing PIMs.

Authorized licensed use limited to: Rochester Institute of Technology. Downloaded on March 17,2025 at 18:46:43 UTC from IEEE Xplore. Restrictions apply.

## References

[1] S. L. et al., “Scaling the “memory wall”: Designer track,” in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2012.

[2] H. S. Stone, “A logic-in-memory computer,” *IEEE Transactions on Computers*, vol. C-19, no. 1, pp. 73–78, Jan 1970.

[3] S. Bavikadi, P. R. Sutradhar, K. N. Khasawneh, A. Ganguly, and S. M. Pudukotai Dinakarrao, “A review of in-memory computing architectures for machine learning applications,” ser. GLSVLSI ’20. New York, NY, USA: Association for Computing Machinery, 2020, p. 89–94. [Online]. Available: https://doi.org/10.1145/3386263.3407649

[4] D. P. et al., “Intelligent ram (iram): the industrial setting, applications, and architectures,” in *Proceedings International Conference on Computer Design VLSI in Computers and Processors*, Oct 1997, pp. 2–7.

[5] S. L. et al., “Drisa: A dram-based reconfigurable in-situ accelerator,” in *IEEE/ACM International Symposium on Microarchitecture*, 2017.

[6] Q. D. et al., “Dracc: a dram based accelerator for accurate cnn inference,” in *ACM/ESDA/IEEE Design Automation Conference (DAC)*, 2018.

[7] S. Li and et al., “Scope: A stochastic computing engine for dram-based in-situ accelerator,” in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Oct 2018, pp. 696–709.

[8] C. E. et al., “Neural cache: Bit-serial in-cache acceleration of deep neural networks,” in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, June 2018, pp. 383–396.

[9] J. Ahn, S. Yoo, O. Mutlu, and K. Choi, “Pim-enabled instructions: A low-overhead, locality-aware processing-in-memory architecture,” in *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, June 2015, pp. 336–348.

[10] S. Angizi and D. Fan, “Redram: A reconfigurable processing-in-dram platform for accelerating bulk bit-wise operations,” in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019.

[11] P. R. Sutradhar, M. Connolly, S. Bavikadi, S. M. Pudukotai Dinakarrao, M. A. Indovina, and A. Ganguly, “ppim: A programmable processor-in-memory architecture with precision-scaling for deep learning,” *IEEE Computer Architecture Letters*, vol. 19, no. 2, pp. 118–121, 2020.

[12] P. R. Sutradhar, S. Bavikadi, M. Connolly, S. K. Prajapati, M. A. Indovina, S. M. Pudukotaidinakarrao, and A. Ganguly, “Look-up-table based processing-in-memoryarchitecture with programmable precision-scalingfor deep learning applications,” *IEEE Transactions on Parallel and Distributed Systems*, pp. 1–1, 2021.

[13] V. S. et al., “Ambit: In-memory accelerator for bulk bitwise operations using commodity dram technology,” *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2017.

[14] F. Gao, G. Tziantzioulis, and D. Wentzlaff, “Computedram: In-memory compute using off-the-shelf drams,” in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO ’52. New York, NY, USA: Association for Computing Machinery, 2019, p. 100–113. [Online]. Available: https://doi.org/10.1145/3352460.3358260

[15] H. et al., “Simdram: A framework for bit-serial simd processing using dram extended abstract.”

[16] Q. D. et al., “Lacc: Exploiting lookup table-based fast and accurate vector multiplication in dram-based cnn accelerator,” in *ACM/IEEE Design Automation Conference (DAC)*, 2019.

[17] “Edge tpu performance benchmarks,” coral. [online].” November 2020. [Online]. Available: https://coral.ai/docs/edgetpu/benchmarks/

Authorized licensed use limited to: Rochester Institute of Technology. Downloaded on March 17,2025 at 18:46:43 UTC from IEEE Xplore. Restrictions apply.