Project:

MNEMOSENE

(Grant Agreement number 780215)

"Computation-in-memory architecture based on resistive devices"

Funding Scheme: Research and Innovation Action

Call: ICT-31-2017 "Development of new approaches to scale functional performance of information processing and storage substantially beyond the state-of-the-art technologies with a focus on ultra-low power and high performance"

Date of the latest version of ANNEX I: 11/10/2017

D3.3– Refined CIM architecture

Project Coordinator (PC):	Prof. Said Hamdioui Technische Universiteit Delft - Department of Quantum and Computer Engineering (TUD) Tel.: (+31) 15 27 83643 Email: S.Hamdioui@tudelft.nl
Project website address:	www.mnemosene.eu
Lead Partner for Deliverable:	Eindhoven University of Technology (TUE)

Report Issue Date:	27/03/2020

Document History

(Revisions – Amendments)

Version and date	Changes
17/03/2020	Outline updated
24/03/2020	Draft prepared
27/03/2020	Final version

Dissemination Level

PU	Public	**X**
PP	Restricted to other program participants (including the EC Services)	
RE	Restricted to a group specified by the consortium (including the EC Services)	
CO	Confidential, only for members of the consortium (including the EC)	

The MNEMOSENE project aims at demonstrating a new computation-in-memory (CIM) based on resistive devices together with its required programming flow and interface. To develop the new architecture, the following scientific and technical objectives will be targeted:

- Objective 1: Develop new algorithmic solutions for targeted applications for CIM architecture.
- Objective 2: Develop and design new mapping methods integrated in a framework for efficient compilation of the new algorithms into CIM macro-level operations; each of these is mapped to a group of CIM tiles.
- Objective 3: Develop a macro-architecture based on the integration of group of CIM tiles, including the overall scheduling of the macro-level operation, data accesses, inter-tile communication, the partitioning of the crossbar, etc.
- Objective 4: Develop and demonstrate the micro-architecture level of CIM tiles and their models, including primitive logic and arithmetic operators, the mapping of such operators on the crossbar, different circuit choices and the associated design trade-offs, etc.
- Objective 5: Design a simulator (based on calibrated models of memristor devices & building blocks) and FPGA emulator for the new architecture (CIM device combined with conventional CPU) in order demonstrate its superiority. Demonstrate the concept of CIM by performing measurements on fabricated crossbar mounted on a PCB board.

A demonstrator will be produced and tested to show that the storage and processing can be integrated in the same physical location to improve energy efficiency and also to show that the proposed accelerator is able to achieve the following measurable targets (as compared with a general purpose multi-core platform) for the considered applications:

- Improve the energy-delay product by factor of 100X to 1000X
- Improve the computational efficiency (#operations / total-energy) by factor of 10X to 100X
- Improve the performance density (# operations per area) by factor of 10X to 100X

Table of Contents

1. Introduction

As the data, which is required to be processed, is growing at a galloping pace, the need for brand-new processor architectures, which enjoy new paradigms of computing, becomes vital. Existing processors are mostly based on the von-Neumann architecture, in which the processing units and memory banks are placed apart from each other Figure 1.a. Therefore, every time a data is needed to be processed, it must be fetched from memory, transferred to the processor, and the produced result, again, must be transferred back to the memory to be stored. Considering this behaviour and given the data which is needed to be processed is significantly large, this paradigm not only elongates the process-time but also consumes a significant amount of energy. Adding to the levels of the memory hierarchy and introducing small-sized caches, close to the processing unit, have alleviated the problem (Figure 1.b). Nonetheless, such techniques still fail to meet ever-increasing, performance and energy requirements of the emerging applications, e.g. neural networks. Computation in-memory, on the other hand, suggests processing the data at the very same site where it resides [14, 16, 18]. Eliminating a huge part of data transfers, CIM reduces the data bandwidth constraints, decreases energy budget requirements, and improves the performance.

An approach to realize CIM is to exploit emerging non-volatile memories (NVM), e.g., resistive random access memory (ReRAM), phase change memory (PCM), and spin-transfer torque magnetic random access memory (STT-MRAM) (Figure 1.d). These devices offer promising features like compact realizations, ultra-low static power, and non-volatility –eliminating the required energy to refresh cells– which makes them favourable candidates to realize CIM. Being organized in a crossbar structure, they can operate as extremely dense memory units. Employing the very same structure, several operations such as vector matrix multiplication (VMM), and bulk Boolean bitwise operation (BBB), can be executed on them exploiting their analogue behaviour. Although memristor based CIM is considered one of the most promising options for the next generation of processors, the implications associated with their deployment at the computer architecture level are rarely studied.

In this document, we first start in section 2 with presenting background information on memristors and memristor crossbar. Then, in Section 3, we introduce the micro-architecture that consist of Micro-engine and Calculator. In section 4, PULP architecture, the CIM unit deployment as an accelerator, and the necessary modification –applied to CIM unit— are explained. Integration of CIM unit as an accelerator into PULP is interesting as PULP is a reasonable representative for prevalent MPSoCs. By integration we can figure out if memristor crossbars show promising results in commercialized architecture. In Section 5, CIM is integrated as a functional unit into TTA, a data path aware architecture. The promise of configuring data path in a fashion that minimizes the memory accesses motivated us to consider integration into such architectures. In Section 6, the results that are obtained from CIM integration into TTA are presented. Section 7 concludes the document.

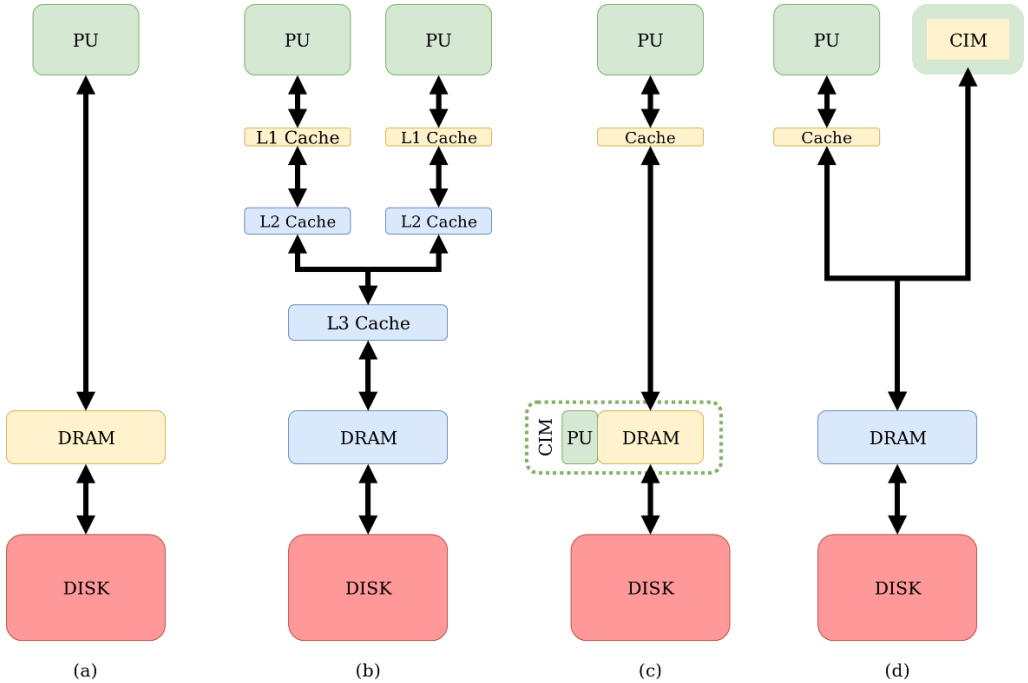


Figure 1. a) Early computers, b) Multi-processor with multi-level caches, c) computation in memory (DRAM), d) computation in memory (memristor) [20]

2. Background

For a good understanding a proper background is needed. This section presents the basics of non-volatile memories, and their organization in a crossbar structure are presented.

2.1 Memristors and Memristor Crossbars

Device engineers have been seeking for alternative technology for the post-CMOS era since the device scaling is reaching the atomic realm. Non-volatile memories, e.g. phase change memories (PCM), resistive RAMs (ReRAM), are considered a promising instance of such devices. Information maps to the conductance states of a device. The device retains its value until it is (RE)SET by a voltage higher than its threshold voltage.

Being organized in an area-efficient crossbar structure, they can be exploited as dense blocks of memories (Figure 4.a). To further increase the density –bits per area– multi-level cells are used. The non-linear I-V characteristics of the memristors allow them to hold multiple conductance states. In multi-level cells, multiple bits of information can be stored on a single device with multi-conductance states. To fine-tune the conductance level, in a write-verify scheme, pulses with different width (amplitude) are applied across the target device.

However, the non-linear characteristics of the device itself, and the sneak path current in a crossbar –an undesired current that flows through the memory cells parallel to the desired one– make it hard to use the device in practice. To make the programming stage more controllable, the 1T1R structure (1-Transistor 1-Resistor) is proposed [26] (Figure 4.b). In the 1T1R structure, a transistor is placed in series with a memristor. Depending on the direction in which the gates of the transistors are connected, row/column-wise, different functionalities can be achieved. For instance, if the gates are connected horizontally, bulk bit-wise operations (BBB) can be executed, but not hyper-dimensional computation (HDC) [8]. Whereas, if the gates are connected vertically, the platform is suitable for HDC and not BBB.

2.2 Memristor Crossbars Functions

Memristors, either individually or in a crossbar layout, have been employed for various use cases. They have performed as physically unclonable functions [10], radiofrequency switches [17], dot product engines [9], and memory blocks [7]. In this section, we focus on memristor crossbar-based use-cases.

The most appealing feature of memristors is that they can perform vector-matrix multiplication (VMM), which is the core operation of many applications especially neural networks, in a single shot. To perform a VMM ($Y_{1\times n} = W_1 \times X_{m\times n}$), where W, X, and Y are weight matrix, input vector, and output vector, respectively, several measures should be taken (Figure 2.c). First, the elements of the weight matrix are mapped to the conductance states of the crossbar cells. Then, elements of the input vector are encoded to the amplitude

or the length of pulses to be applied to rows. Note that the amplitude of the input voltages should not exceed the threshold voltage of the device, as this destructs the stored information. Next is to drive the input voltages into the word-lines while the bit-lines are virtually grounded. According to Ohm's law, a current equal to the product of input voltage and the cell conductance flows through the device ($I_{ij} = V_i \times G_{ij}$). The currents, which are resulted of all element-by-element multiplications, are accumulated and sensed simultaneously at the end of active columns ($I_i = \sum I_{ij}$). Thanks to this attribute, memristor crossbars are considered one of the most promising platforms to realize neuromorphic computing architectures [2, 9, 19, 22]. It is worth mentioning that if the required analogue peripheries –which drive the crossbar in the reverse direction– are available, it is possible to do VMM on the transpose matrix using the very same crossbar [5].

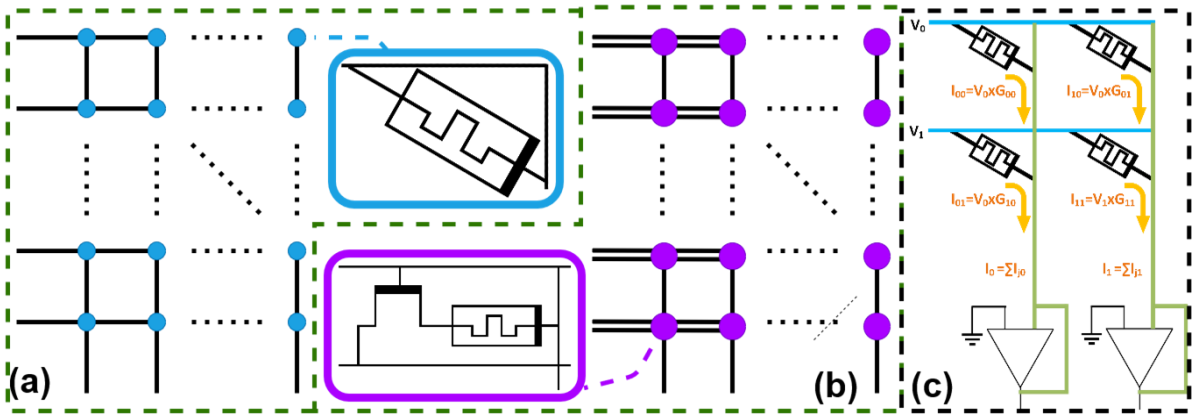


Figure 2. a) Passive Crossbar, b) Active Crossbar (1T1R), c) Vector Matrix Multiplication

The other use-case, in which memristor crossbars have shown promising results, is in memory bulk Boolean bitwise operation [13]. BBB is frequently performed in database queries and DNA sequence alignment [18]. To perform a bitwise operation, reference [24] programs every single bit of an operand to cells in a row. Then, by applying a read voltage to two desired rows, where targeted operands are stored, currents flow in bit-lines. Currents are sensed using scouting logic. The reference current of the sense amplifier is determined based on the operation, i.e. AND, OR, XOR to be done. Comparing the sensed current with the reference current(s), different Boolean operations can be realized.

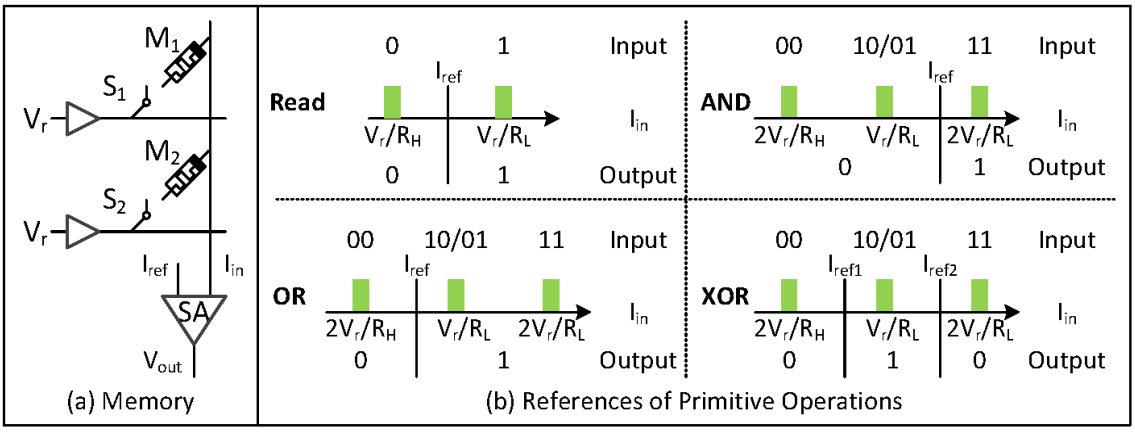


Figure 3. In Memory Bulk Boolean Bitwise Operation

3. Micro Architecture

As memristor technologies are hardly available, to develop an architecture for processors based on memristor crossbars we designed a simulator that replicates the behaviour of memristor crossbars. The simulator also covers the operation of the peripheral mixed-signal circuitry. We call this mixed-signal part “Calculator” (Figure 4.a). For the calculator to communicate with other processing elements, we have expanded the simulator, by adding necessary digital components, e.g. buffers, register files, controller. The added digital components all together comprise the “Micro-engine” (Figure 4.a). In the following of this section, first, the architecture of the CIM unit is illustrated; then, the proposed micro-ISA is introduced; and in the end, the tasks of the controller are described.

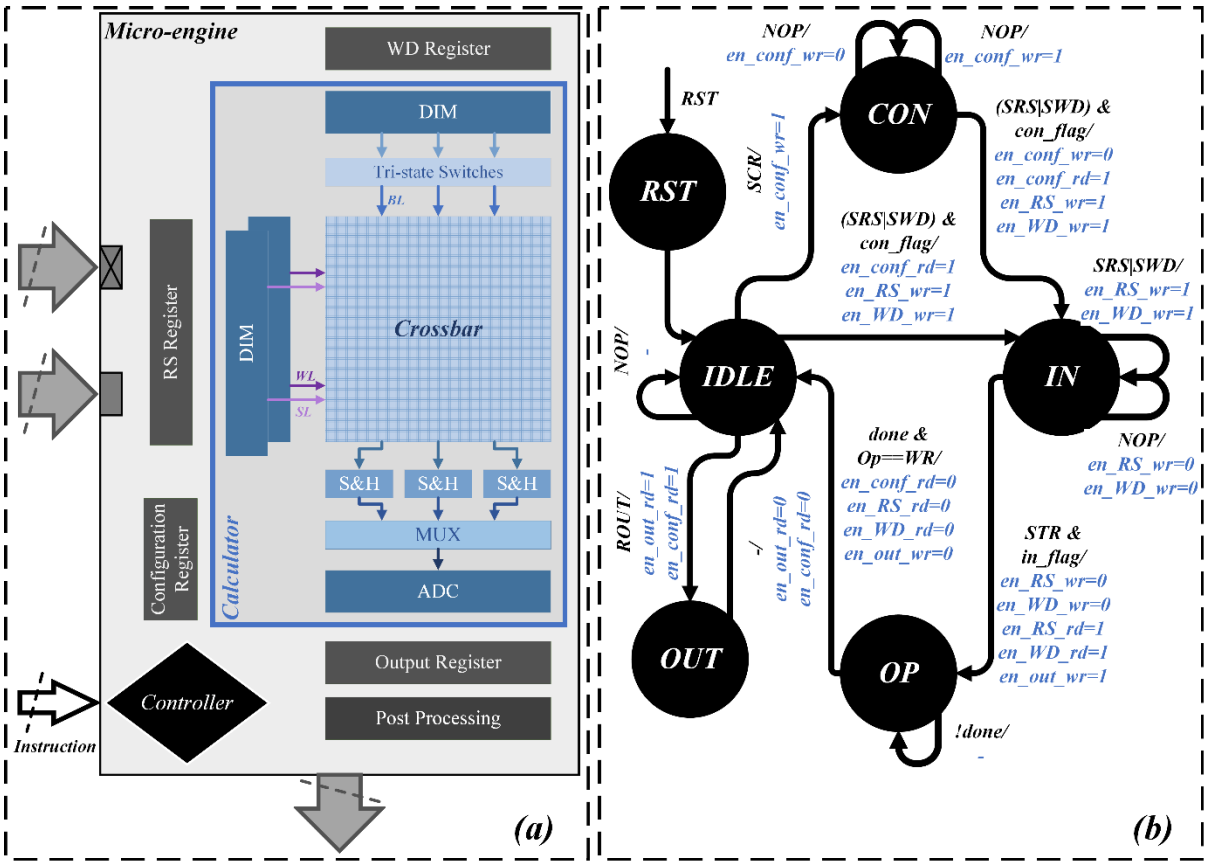


Figure 4. Overall look of CIM unit. a) The tile structure, b) The Controller, FSM (only memory units enable/disable signals are presented)

3.1 CIM Architecture

CIM tile, a cycle-accurate simulator that is developed in C++, replicates the function of a memristor crossbar, the driving mixed-signal circuits, and necessary digital elements. The tile includes two different domains, mixed-signal, the Calculator, and digital, the Micro-engine (Fig. 2.a).

In Calculator, various modules are defined to imitate the behaviour of every single analogue component, i.e. the crossbar, analogue/digital converters (ADC), digital input modulators (DIM), sample and holds (S&H). This modular design enables a designer to change the configuration of the Calculator, e.g. crossbar size, number of ADC/DIMs, or add new modules with a negligible effort.

DIMs and ADCs, at the edges of Calculator, are connected to digital buffers, i.e. RS/WD/Output. DIMs convert the raw data into amplitude/width of pulses that are to drive the crossbar. Crossbar results, if operation produces any, are converted to digital values by ADCs and stored in Output buffer. Simple digital logics, present inside the Micro-engine, carry out simple operations like weighted sum if desired. To instruct the CIM unit to carry out different operations, we propose a micro-instruction set architecture (**Error! Reference**

source not found.). The micro-instructions promote the nano-instructions that are presented in [4]. Although the nano-instructions offer full control over the tile, there is quite some room to enhance them. For example, the process of fetching raw data into buffers requires the whole buffer to be overwritten, even if only a few entries are supposed to change. The full control offered by nano-architecture is not desirable as it comes with many dependencies between instructions. This not only make the compilation complex, long, and inefficient but may lead to unreliable code. Therefore, we designed a micro-ISA to avoid interfering with any pre/post-processing stage inside the CIM unit, thus eliminating the chance of external error. The controller (Figure 4.b) reads the operation parameters that are written into the configuration register (**Error! Reference source not found.**) via SCR instruction and manage the whole operation based on these registers.

3.2 Micro-Engine and Micro-ISA

To avoid long micro-ISA, that include several execution parameters, a configuration register is introduced to hold these parameters (**Error! Reference source not found.**). The controller, a mealy machine (FSM), conducts the correct execution of an instruction, e.g. filling buffers, enabling/disabling the analogue elements, etc (**Error! Reference source not found.**). To carry out an operation, first, the configuration should be filled using the set configuration register (SCR) instructions. Based on the contents of the configuration register, the FSM calculates the address of the buffers, aligns the data, triggers the operation, collects the crossbar output, and post-processes the output (if needed). In the end, the FSM controls the process of sending the final results out. As mentioned, the controller is a mealy machine that issues the control signal based on received micro-instructions, configuration register contents, the state of the FSM, and internal flags (Figure 4.b). The digital buffers, the configuration registers, and the controller, all together, constitute the Micro-engine.

Table 1. Micro instructions

Class	Mnemonic	Description	Operands
Initialization	SCR	Set Configuration Register	address, data
	SRS	Set Row Select buffer	data
	SWD	Set Write Data buffer	data
Compute	STR	Start operation, e.g. VMM	-
Read	ROUT	Read results out	-

Table 2. Configuration Registers. Controller conducts an operation based on the content of configuration register

Register Index	0	1	2	3	4	5	6	7	8	9-15
Register content	Start Row	Start Column	Number of Rows	Number of Cols	Input Precision	Weight Precision	Output Precision	Truncate Bits	Operation	Reserved

According to a prototype developed by IBM [12], to write a phase change memory (PCM), a type of memristors, or to perform a VMM it takes 2.5µs, 1µs, respectively. Considering that these delays are considerably long, we attempted to schedule some tasks into these very long time slot. The data that is to be processed on the CIM unit is a vector. Hence, we propose to add an extra set of buffers in the CIM unit (double buffering) to fetch the $(n+1)^{th}$ vector while n^{th} vector is being processed (**Error! Reference source not found.**.b). To ensure that no data is lost, we add an extra level to the controller that supervises the correct redirection of the control signals. This approach of having a top layer in the controller enables us to perform the operations that are targeted in [8].

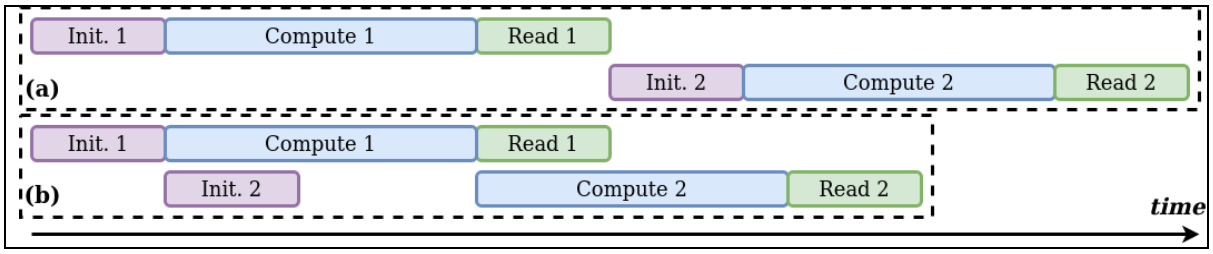


Figure 5. CIM unit execution flow, a) without DB, b) with DB (Init., Compute and Read are three classes of micro instructions in Table 1)

4. PULP Macro Architecture

In this section we first briefly introduce PULP. Then details of introducing an accelerator to PULP is elaborated. In the end, the overview of the accelerator and the integration of CIM tile is described.

4.1 PULP

Figure 6 depicts the architecture of PULP. The system is split into two different power/clock domains: 1) The Cluster domain contains several general-purpose processing cores, CIM Accelerators and shared memory. 2) The SoC domain on the other hand contains IO blocks for communication with off-chip peripherals and external memory, a power management unit and clock generators. All these components as well as the cluster domain is controlled by a tiny processor, the so-called fabric controller.

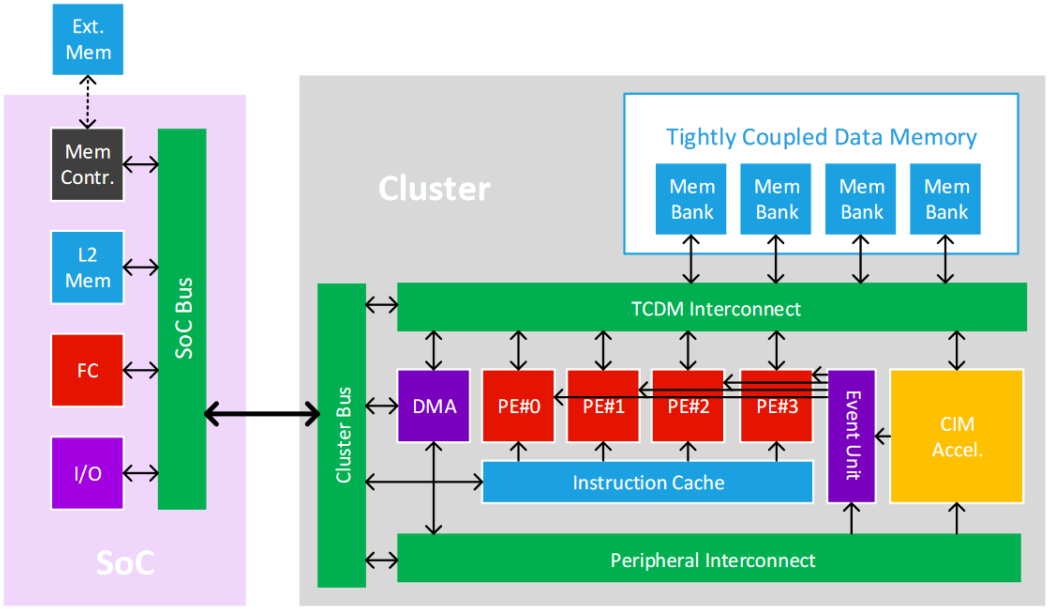


Figure 6. PULP architecture

Fabric Controller: The fabric controller is responsible for orchestrating I/O peripherals the external memory controller as well as the cluster. To process data on the high-performance cluster the fabric controller can fork execution to the general-purpose cores within the cluster and can dynamically adjust the clusters frequency for optimal performance and energy efficiency.

Cluster Architecture: The cluster consists of several general-purpose processing elements (PE) based on ETH's RI5CY RV32IMF core and one or multiple CIM accelerators. All processing elements share access to the tightly coupled data memory, a multi-banked software managed scratchpad memory for data exchange between the processing elements. The DMA module

within the cluster can be programmed by the PEs to transfer data between the larger L2 memory in the SoC domain and an event unit provides the means for synchronization between the processing elements. Each processing element is connected via two interconnects: The peripheral interconnect is a low bandwidth bus based on AMBA APB1 used for configuration of the DMA, event unit and most importantly the CIM accelerators. The TCDM interconnect provides low latency and high-bandwidth access to the shared TCDM memory and is based on the so-called Logarithmic Interconnect, a forest of arbitration trees providing parallel single cycle access to the multi-banked memory with transparent arbitration in the case of bank conflicts between different processing elements.

4.2 CIM Accelerator Interface

TCDM Interface: CIM accelerators are connected to external L1/L2 shared memory by means of a simple memory protocol, using a request/grant handshake. The protocol used is called Tightly Coupled Data Memory (TCDM) protocol, and it is the same as the one used by cores and DMAs operating on memories. It supports neither multiple outstanding transactions nor bursts, as the accelerators are designed to be closely coupled to memories. The TCDM protocol is used to connect a master to a slave.

Peripheral Interface: To enable the control of the CIM Accelerators, they typically expose a slave port to the peripheral system interconnect. The slave port follows an extension of the TCDM protocol which we can call PERIPH. The PERIPH protocol is the same exposed by most peripherals in a PULP system and used by the GP cores to communicate with them. The PERIPH protocol is distinguished by the TCDM protocol by the id and r_id side channels. They are used in load operations issued through a PERIPH interface: the id identifies the master during the request phase, is buffered by the slave peripherals and accompanies the response phase as r_id. In this way, multiple masters can distinguish which traffic is related to themselves.

4.3 CIM Unit Integration into CIM Accelerator

The CIM accelerator comprises some load/store units –to calculate the address of input/output data and to interact with TCDM interface, as well as a CIM unit. The controller of the CIM unit should change slightly to carry out an operation without keeping the core busy. Hereunder it is explained what the CIM tile would look like after applying the modifications:

1) To interface with the Peripheral Interconnect for configuration access by the general-purpose core(s), it should have a 4-bit address port and a 32-bit read/write port. These ports are used by the core(s) to read/write any address in the configuration register of the CIM unit. A similar configuration register is required for the load/store unit(s).
2) A 32-bit data stream input and 32-bit data stream output needs to be added with the means to stall accelerator in the presence of memory contention in the scratchpad

memory. The load/store units will translate between the stream-based data-path of the CIM-tile and the transaction-based data-path of the TCDM interconnect.

3) To keep core(s) free as much as possible, the core(s) will program the CIM unit, and the load/store units and trigger the operation as soon as the units are configured. Therefore, the configuration register would be extended according to Table 3. Load/store units will interact with the TCDM interconnect to fetch/store data into TCDM memory bank according to the linear memory access scheme configured by the core via the Peripheral Interconnect.
4) A handshaking protocol would be devised to replace SRS, SWD, and ROUT micro-instructions in Table 1. This is required to ensure a valid data is on the CIM unit I/O port.

Table 3. CIM unit configuration register in PULP

Register Index	0	1	2	3	4	5	6	7	8	9	10	11-15
Register content	Start Row	Start Column	Number of Rows	Number of Cols	Input Precision	Weight Precision	Output Precision	Truncate Bits	Operation	Status	Trigger Operation	Reserved

5. TTA Macro Architecture

In this section, first, Transport Triggered Architectures (TTAs) are introduced. It is followed by describing TTA Co-design Environment (TCE), an open-source toolset that allows adding special functional units like the CIM unit to the architecture. Then, the details of the CIM special functional unit (CIM-SFU) integration into TTA architecture is explained. Lastly, Low-power TTA (LoTTA) [15], a processor core based on TTA that is developed for energy-efficient execution of always-on applications, is described.

5.1 TTA and TCE

Transport Triggered Architectures are a class of Exposed Data Path Architectures (EDAPs), where the data path of the processor instance is exposed to the programmer. Fine-grain control over data path allows compile-time bypassing of data between processing-elements, i.e. software bypassing, without dependency checking hardware circuitry, which improves energy-efficiency. Furthermore, static scheduling of instructions on EDAPs open up new optimization possibilities on the software as well as on the hardware side. With a data path aware compiler, a) the register file data-bandwidth requirements of EDAPs do not need to satisfy the worst-case design requirements of the execution pipeline (FUs) and thereby relaxes the read/write port requirements of RF design and improves scalability over VLIWs, b) the interconnection network can be customized for an application/domain to prioritize the most-used data path over rarely used one to avoid excessive redundant connections, and c) the EDAPs specific optimization at soft-ware level, such as the ability to directly forward data from one FU to another(software bypassing), and eliminating a result move to RF when all the uses of the result are software bypassed (dead result elimination) to avoid costlier RF access. The above described primary optimizations allow EDAPs to reach high energy-efficiency without compromising programmability.

Compared to traditional operation triggered architectures, where operation triggers data path activity, the TTA instruction represents the data transports (TTA data-move) and the computations are triggered as a side effect of the data-move. Functional Units of TTA comprise one or more operand ports and optional result ports that can communicate data over the interconnect network. In TTAs, one of the operation ports of FUs is a fixed as special port named “trigger-port”. A data-move to trigger-port starts the execution of an operation on FUs. Figure 7 (bottom) depicts an example of a TTA processor instance with three communication resources (bus network) and one CIM unit as an added special functional unit (CIM-SFU). The trigger-port of the FUs are marked as “T”. The CIM-SFU is the functional model of the CIM unit with CIM-ISA defined in the section 3. The Figure 7 (top) shows a TTA program for simple increment operation (i.e. a=a+ 4) on the processor instance [15] to illustrate a programming model of the TTAs. Three buses in the instance imply that three data transport can happen in parallel during each clock cycle. Therefore, both operands move (a and 4) can happen in parallel for the considered example and the whole operation takes two cycles in total as shown by the color-coded data moves. The data move *4→ALU1.add.in1t* triggers

execution of an add operation in the ALU unit with operands on input ports *in1t* and *in2* at cycle-1. At cycle-2, the result of the add operation (assuming add latency is one cycle) is written back to RF.

A mature open-source toolset, TTA Co-design Environment (TCE), enables users to design and freely customize TTAs for their purpose. It includes a re-targetable instruction-set compiler, cycle-accurate ISA simulator, RTL generator, and support for adding special compute-units for dedicated functions. TTAs are an ideal candidate for energy-efficient application-specific instruction-set processors (ASIPs), and a sensible choice for prototyping experimental platforms such as CIM Tile.

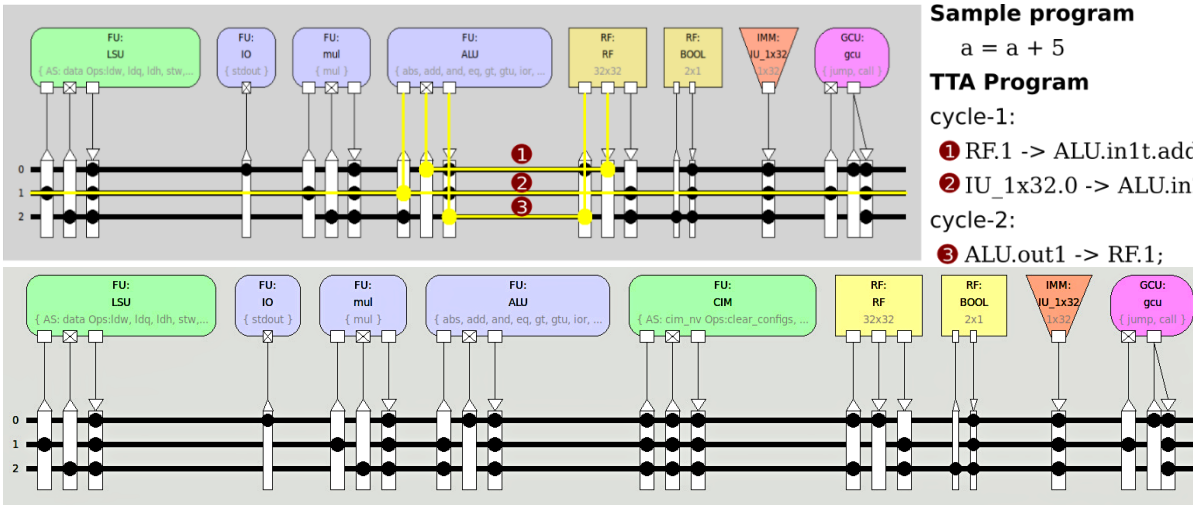


Figure 7. TTA based processors overview. LoTTA (Top). LoTTA+1×CIM (Bottom)

5.2 CIM unit integration into TTA

As pointed out before in Section 3, the latency of the CIM computations (VMM, Write, etc.) is much higher than the regular operations on the traditional FUs such as ALU, MUL, etc. This gives rise to two main requirements for CIM-SFU design, 1) A multi-cycle FU model to hide the latency of the CIM-SFU from other units, and 2) Pipelined FU model to separate the CIM compute unit (Calculator) from the TTA interface (operand and result port) to hide data-latency with the double-buffering concept (**Error! Reference source not found.**.b). The semi-virtual time latching model of the FUs in the TCE allows multi-cycle, pipelined TTA-SFU model possible in the TCE tool-set. Figure 4.a depicts the designed CIM-SFU. The CIM-SFU is modelled as a three-stage pipeline with the first-stage the first-stage fetching the input data, the second stage covering the core CIM mixed-signal computation logic (*Calculator*), and the third stage sending the results out (**Error! Reference source not found.**.a). The pipeline of the SFU is controlled by a trigger move i.e. when the trigger move happens, the operands ($input_{1t}$ and optional $input_{2-n}$) are latched to the Micro-engine and the opcode is issued to the controller of the Micro-engine. The FSM, then, issues necessary control signals for the rest of the components. Calculator, which is modelled cycle-accurately, produces the result on the output buffer once the operation latency is elapsed for the triggered operation. The data on

the output buffer is then serialized to the output port via an explicit trigger commands to controller. The architecture template of the TCE requires that each operation in the FU to have a deterministic latency such that the resulting read for the operation can be scheduled at compile time.

5.3 LoTTA

Low-power TTA (LoTTA) [15] is a processor core aimed for always-on processing and efficient execution of both signal processing and control-oriented programs. The core used for evaluations in this paper is a variant of the original work. Functional units of the core and its interconnection network are presented in Figure 7.a.

6. Experiments

In this section, we present the effects of adding CIM unit in a quantitative manner. To do so, we employ LoTTA, without a CIM-SFU, as a base setup. Then, CIM-SFU(s) are added to LoTTA to explore its effect on various parameters such as performance, energy, and area. For evaluation, we used gemm as well as deep learning LeNet kernels.

6.1 Experimental Setup

The energy and area estimations for the LoTTA core are obtained after synthesis with Synopsys Design Compiler, version 2016.12. A 28 nm process is used at 0.95 V operating voltage and 25°C temperature process corner. For power consumption analysis, switching activity information files (SAIFs) are generated with ModelSim 10.5.

The weight to be mapped on NVM crossbar are 8-bit values which are mapped on IBM's 4-bit PCM [11]. To mimic an 8-bit weight with 4-bit cells, two columns are used, one for four MSBs and the other for four LSBs. The final result is computed by a weighted sum of MSB and LSB columns in the digital logic block. The models for the crossbar and mixed-signal circuitry are from [11] and [19].

Table 4. CIM and LoTTA Configuration

Crossbar Parameter	**Value**	
Memristor Technology	IBM PCM	
Cell precision	8-bit (implemented by 2×(4-bit) PCMs)	
Compute and Write Latency/8-bit	1 µs and 2.5 µs	
Compute Energy/8-bit	200 fJ (2x100 fJ/4-bit PCM)	
Write Energy/8-bit	200 pJ (2x100 pJ/4-bit PCM)	
Area (128×128)	50 μm^2	
Peripheral Circuitry	**Energy**	**Area**
Mixed Signal	2.1 nJ/cycle (@1.2GHz)	1252 μm^2
Micro-engine (Digital)	64.8 pJ/byte	865 μm^2

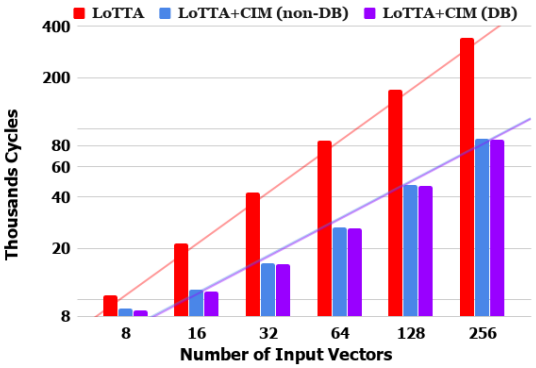
6.2 Evaluation

As mentioned earlier, we evaluated gemm and LeNet kernels. For gemm, we studied the impact of different input and matrix sizes. For LeNet, we evaluated performance, accuracy, energy, as well as area for different crossbar sizes.

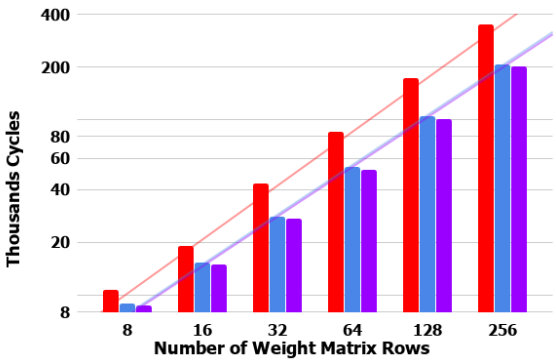
gemm: To evaluate how memristor crossbars perform on VMM, we implemented the gemm kernel on a crossbar of size 256×256. Figure 8.a shows that by increasing the number of input vectors the speedup increases from 1.2X to 3.9X for basic CIM unit without double buffering. This was expected since the computation dominates the initial overhead of programming the crossbar. Increasing the number of the rows of the weight matrix, i.e. the columns of the input

matrix, it is observed that although the speed-up increases, the improvement rate is less significant compared to the previous case (Figure 8.b). This happens as the number of cycles required to program the crossbar dominates the overall execution time. Considering these two experiments, and the fact that memristors still suffer from low endurance, the read/write ratio shall be taken into account to assess if it is reasonable to use a memristor crossbar or not. Lastly, in Figure 8.c we observe that increasing the number of weight matrix columns increases the speed-up rate of LoTTA+CIM over LoTTA.

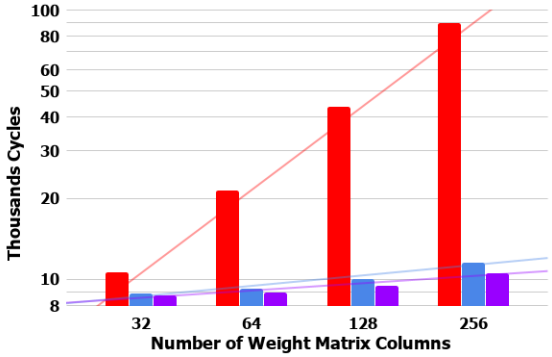
Double Buffering: Figure 8.a shows that although by deploying double buffering performance improves compared to non-DB, relative speedup goes down from 1.02X to 1.01X. This happens since the size of input vector is too small, thus, a limited number of instructions can be scheduled to a VMM execution period (Compute stage in **Error! Reference source not found.**.) However, in Figure 8.c we observe that as the number of columns of weight matrix increases the relative speedup caused by deployment of DB goes from 1.02X to 1.10X. This happens since the size of the vector to be loaded and programmed on the memristor crossbar grows; therefore, more instructions can be scheduled to a write execution period.



(a) Varying Input Vectors



(b) Varying Weight Matrix Rows



(c) Varying Weight Matrix Columns

Figure 8. Performance for gemm kernel

LeNet: To assess the suitability of memristor based CIMs for deep learning applications, we implemented the LeNet architecture on LoTTA and LoTTA+CIM unit(s). The LeNet architecture comprises both convolutional as well as fully connected neural network layers that makes it a perfect data-intensive application instance to study with respect to implications associated

with deploying memristor crossbars in a full-blown system. One of the challenges that memristor crossbars should address is the possibility that the weight matrix exceeds the memristor crossbar in size, i.e., either in the number of columns or rows. If the number of columns of the weight matrix is bigger than that of the actual memristor crossbar, the only measure that should be taken is to divide the weight matrix over *M* CIM units, where $M = \left\lceil \frac{WeightMatrix_{col}}{MemristorCrossbar_{col}} \right\rceil$, or to divide the task over time and use a CIM unit *M* times. The second solution is not quite desirable due to the low endurance of memristors. Obviously, the RS buffer in each and every CIM unit has to hold the exact same data. Since columns results are independent, they can be handled without any dependency. In case the number of rows of the weight matrix is bigger than that of the actual memristor crossbar, like in the previous case, either *N* CIM units are required, where $N = \left\lceil \frac{WeighMatrix_{row}}{MemristorCrossbar_{row}} \right\rceil$, or the operation should be carried out in *N* time steps with one CIM unit. Unlike the previous case the results of each part are only partial results and should be accumulated to produce the final result. To study this, we assume various sizes for the memristor crossbar. The biggest memristor crossbar has 512 rows and the smallest has 128. The number of columns of all the layers of LeNet are always smaller than the matrix size. Considering that the weight matrix of the second and the third layers of LeNet after unrolling are 150 and 400 rows, respectively, these layers should be either distributed temporally, if the resources are limited, or spatially, if enough CIM units are available. Figure 9 shows the results of various implementations. Using only one CIM unit, even one that is big enough to map a whole layer to the crossbar, performance is worse than any other implementation with multiple CIM units due to parallelism. Distributing weights matrix amongst several CIM units saves quite a few cycles while programming the crossbar, since this is the most time-consuming step of performing an operation on a memristor crossbar. The reason that the implementation with two units of size 256×256 yields worse results compared to the one with two crossbars of size 128×128 is that we map a whole layer to one crossbar if it was possible. This is important as splitting weight degrades the accuracy (Figure 10). As can be seen in Figure 9, we have mapped the architecture for the DB version as well. The best performance is achieved when the four CIM units that are deployed in the design, do the calculation in parallel.

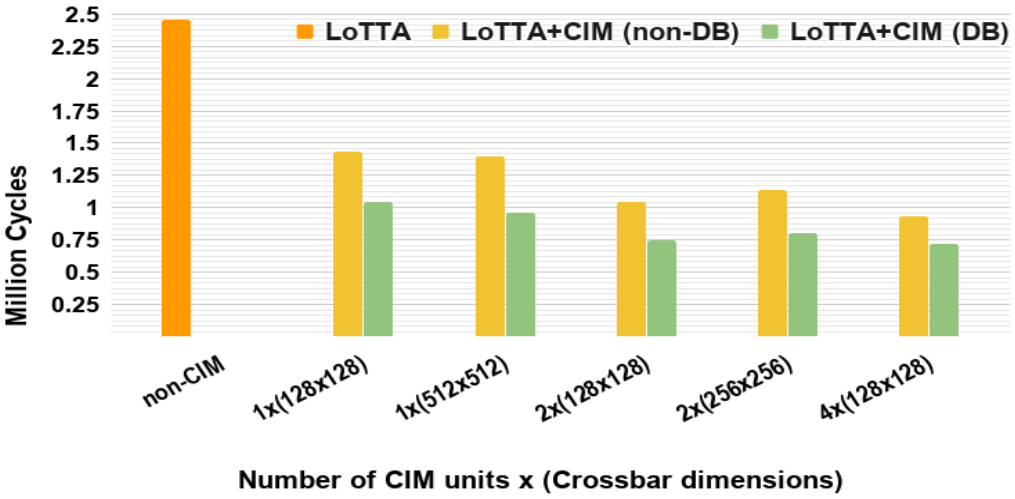
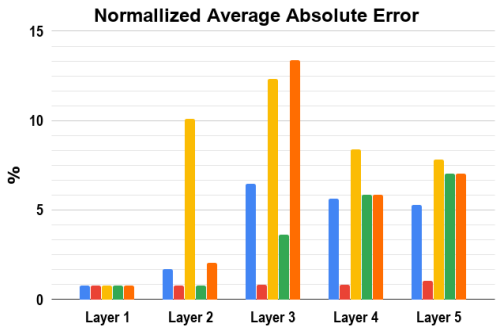
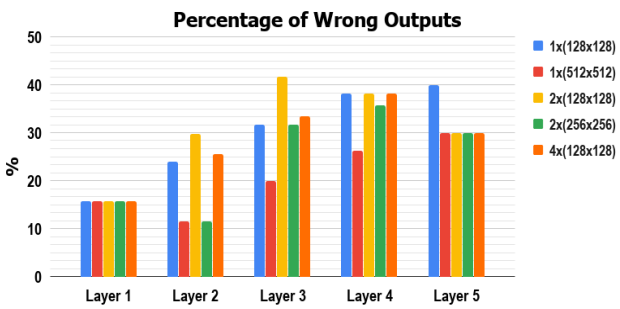


Figure 9. Performance comparison of architectures for LeNet deep neural network

Accuracy: The error sources that we have spotted are two folds; 1) MSB and LSB columns are truncated by the DC offset of the ADC before being summed up together; therefore, the expected carry bit from the addition of the lower bits is lost, 2) The saturation voltage of ADC clips the high/low voltages. In case a crossbar is distributed amongst several CIM units or it is multiplexed in time, a partial sum may exceed the saturation voltage while other partial sums do not reach the saturation voltage. If all parts were together this would be a saturated result while in the distributed case partial sums produce a different result. In a mathematical terminology $\sum f(x_i^j) \neq f(\sum x_i^j)$, where f is a non-linear function – characteristics of ADC– and x_i^j is the output of ith column of jth CIM crossbar. Looking at Figure 10, we observe that splitting the weight matrix in different manners results in different errors. As an instance, for crossbar(s) of size 128×128, although the inputs are the same, just by splitting the second layer weight differently different number of faulty errors with different averages are reported (see blue, yellow, and orange bars in the figure). Although these errors degrade the final result, due to the resilient nature of LeNet the input images are still classified correctly. To address the degradation of the results, itis required to retrain the network with crossbar size being taken into account.



a) Sum of absolute errors normalized by the maximum value



b) Percentage of faulty outputs

Figure 10 . Errors in results using basic weight without retraining

Energy and Area: One of the most important concerns in the deployment of CIM units is their analogue nature which requires data conversion that can be costly. Although digital/analogue converters (DACs) are relatively cheap, in terms of energy/area, analogue/digital converters (ADCs) can be extremely costly– more than 500X more power hungry and more than 7000X bulkier compared to DACs [19]. One of the techniques that is commonly used to reduce the energy/area overhead is to share an ADC amongst several columns. To do so, sample and hold circuits (S&Hs) are introduced between the ADC and crossbar. With such modifications the required energy budget falls into a reasonable scale. Table 5 shows that in the best case up to 69% energy reduction can be achieved, while area is increased by 80%. Looking at EDPA –energy, delay, area product–we observe that in almost all cases, except for a crossbar of size 512×256, the double buffered version performs better than all other implementation in the basic CIM unit. Also, it is spotted that in the DB version the 2×(128×128) yields the best EDAP, while in the basic version the best EDAP is obtained by 4×(128×128).

Table 5 . Energy and area results for different CIM unit configurations

Core	**LoTTA**	**1×(128×128)**		**1×(512×512)**		**2×(128×128)**		**2×(256×256)**		**4×(128×128)**	
		non-DB	**DB**	**non-DB**	**DB**	**non-DB**	**DB**	**non-DB**	**DB**	**non-DB**	**DB**
Energy (mJ)	1.54	0.92	0.68	0.89	0.62	0.68	0.49	0.74	0.52	0.61	0.48
Area (μm^2)	10009	12175	12460	17534	18674	13761	14331	16934	18074	16934	18074
EDAP (10^9)	13.25	5.66	7.61	3.43	4.97	3.36	3.07	3.88	1.86	2.65	2.18

7. Conclusion

In this deliverable, we presented a cycle-accurate simulator for a memristor based CIM unit to scrutinize the challenges that are associated with their integration into a full system. The simulator has two version: 1) one that offers a micro-ISA that allows a memristor crossbar to be integrated into a transport triggered architecture, 2) one that is configured via its configuration register and together with load/store units serves as an accelerator in PULP. The integration not only enhances the crossbar with general-purpose functional units/load store units to execute complex kernels but also enables us to study the challenges that are associated with a memristor crossbar deployment in a full-blown system.

Deploying CIM unit(s) –in TTA— shows huge improvements in terms of performance and energy, up to 3.9X speedup and 69% energy reduction. However, including CIM units has a price. The extra units increase the overall area. In our examples, the area increases between 21% and 86%. In addition, accuracy of the results may degrade since the architecture is not taken into account while the networks are trained. Hence, training should take the architecture properties into account. Of course, it would elongate the training process [3]. The huge reduction of EDAP (up to 84%) is an extremely motivating point, though.

We expect that the integration of a CIM unit into PULP also result in a considerable energy saving and performance improvement. Obviously, the area and accuracy penalty are expected to be bigger here. However, as accelerators in PULP can only send data via TCDM memory banks, the improvement that is gained in a multi-CIM unit implementation in TTA, where the data-path can be specified in a wat that the units can pass data directly, is expected to be smaller. In the other hand, the integration into PULP is quite interesting as PULP is far more similar to commercialized classical MPSoCs thereby can help to understand the challenges associated with deployment of CIM units as accelerators.

8. References

1. Ankit, A., Hajj, I.E., Chalamalasetti, S.R., Ndu, G., Foltin, M., Williams,R.S., Faraboschi, P., Hwu, W.m.W., Strachan, J.P., Roy, K., et al.: Puma: A programmable ultra-efficient memristor-based accelerator for machine learning inference. In: Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems. p. 715–731. ASPLOS '19, Association for Computing Machinery, New York, NY, USA (2019). ,https://doi.org/10.1145/3297858.3304049

2. Ansari, M., Fayyazi, A., Banagozar, A., Maleki, M.A., Kamal, M., Afzali-Kusha,A., Pedram, M.: Phax: Physical characteristics aware ex-situ training framework for inverter-based memristive neuromorphic circuits. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 37(8), 1602–1613 (2017)

3. BanaGozar, A., Maleki, M.A., Kamal, M., Afzali-Kusha, A., Pedram, M.: Robust neuromorphic computing in the presence of process variation. In: Design, Automation Test in Europe Conference Exhibition (DATE), 2017. pp. 440–445 (March2017). https://doi.org/10.23919/DATE.2017.7927030

4. BanaGozar, A., Vadivel, K., Stuijk, S., Corporaal, H., Wong, S., Lebdeh, M.A., Yu, J., Hamdioui, S.: Cim-sim: computation in memory simulator. In: Proceedings of the 22nd International Workshop on Software and Compilers for Embedded Systems. pp. 1–4. ACM (2019)

5. Cai, F., Correll, J.M., Lee, S.H., Lim, Y., Bothra, V., Zhang, Z., Flynn, M.P., Lu, W.D.: A fully integrated reprogrammable memristor–cmos system for efficient multiply–accumulate operations. Nature Electronics 2(7), 290–299 (2019)

6. Chen, P.Y., Peng, X., Yu, S.: Neurosim+: An integrated device-to-algorithm framework for benchmarking synaptic devices and array architectures. In: 2017 IEEE International Electron Devices Meeting (IEDM). pp. 6–1. IEEE (2017)

7. Chi, P., Li, S., Xu, C., Zhang, T., Zhao, J., Liu, Y., Wang, Y., Xie, Y.: Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory. In: Proceedings of the 43rd International Symposium on Computer Architecture. pp. 27–39. ISCA '16, IEEE Press, Piscataway, NJ, USA (2016).https://doi.org/10.1109/ISCA.2016.13, https://doi.org/10.1109/ISCA.2016.13

8. Hamdioui, S., Du Nguyen, H.A., Taouil, M., Sebastian, A., Gallo, M.L., Pande, S., Schaafsma, S., Catthoor, F., Das, S., Redondo, F.G., Karunaratne, G., Rahimi, A., Benini, L.: Applications of computation-in-memory architectures based on memristive devices. In: 2019 Design, Automation Test in Europe Conference Exhibition (DATE). pp. 486–491 (March 2019). https://doi.org/10.23919/DATE.2019.8715020

9. Hu, M., Strachan, J.P., Li, Z., Stanley, R., et al.: Dot-product engine as computing memory to accelerate machine learning algorithms. In: 2016 17th International Symposium on Quality Electronic Design (ISQED). pp. 374–379. IEEE (2016). https://doi.org/10.1109/ISQED.2016.7479230

10. Jiang, H., Belkin, D., Savel'ev, S.E., Lin, S., Wang, Z., Li, Y., Joshi, S., Midya,R., Li, C., Rao, M., et al.: A novel true random number generator based on a stochastic diffusive memristor. Nature communications 8(1), 882 (2017)

11. Le Gallo, M., Sebastian, A., Cherubini, G., Giefers, H., Eleftheriou, E.: Compressed sensing with approximate message passing using in-memory computing. IEEE Transactions on Electron Devices 65(10), 4304–4312 (2018)

12. Le Gallo, M., Sebastian, A., Mathis, R., Manica, M., Giefers, H., Tuma, T., Bekas,C., Curioni, A., Eleftheriou, E.: Mixed-precision in-memory computing. Nature Electronics 1(4), 246 (2018). https://doi.org/10.1038/s41928-018-0054-8

13. Li, S., Xu, C., Zou, Q., Zhao, J., Lu, Y., Xie, Y.: Pinatubo: A processing-in-memoryarchitecture for bulk bitwise operations in emerging non-volatile memories. In: 2016 53rd ACM/EDAC/IEEE Design Automation Conference (DAC). pp. 1–6 (June2016). https://doi.org/10.1145/2897937.2898064

14. Mittal, S.: A survey of reram-based architectures for processing-in-memory and neural networks. Machine learning and knowledge extraction 1(1), 75–114 (2018)

15. Multanen, J., Kultala, H., J̈äaskel̈ainen, P., Viitanen, T., Tervo, A., Takala, J.:Lotta: Energy-efficient processor for always-on applications. In: 2018 IEEE Inter-national Workshop on Signal Processing Systems (SiPS). pp. 193–198. IEEE (2018)

16. Nair, R., Antao, S.F., Bertolli, C., Bose, P., Brunheroto, J.R., Chen, T., Cher, C.Y.,Costa, C.H., Doi, J., Evangelinos, C., et al.: Active memory cube: A processing-in-memory architecture for exascale systems. IBM Journal of Research and Development 59(2/3), 17–1 (2015)

17. Pi, S., Ghadiri-Sadrabadi, M., Bardin, J.C., Xia, Q.: Nanoscale memristive radiofrequency switches. Nature Communications 6, 7519 (2015)

18. Seshadri, V., Lee, D., Mullins, T., Hassan, H., Boroumand, A., Kim, J., Kozuch,M.A., Mutlu, O., Gibbons, P.B., Mowry, T.C.: Ambit: In-memory accelerator forbulk bitwise operations using commodity dram technology. In: Proceedings of the50th Annual IEEE/ACM International Symposium on Microarchitecture. pp. 273–287. ACM (2017)

19. Shafiee, A., Nag, A., Muralimanohar, N., Balasubramonian, R., Strachan, J.P.,Hu, M., Williams, R.S., Srikumar, V.: Isaac: A convolutional neural networkaccelerator with in-situ analog arithmetic in crossbars. In: Proceedings of the 43rd International Symposium on Computer Architecture. pp. 14–26. ISCA '16, IEEE Press, Piscataway, NJ, USA (2016). https://doi.org/10.1109/ISCA.2016.12,https://doi.org/10.1109/ISCA.2016.12

20. Singh, G., Chelini, L., Corda, S., Awan, A.J., Stuijk, S., Jordans, R., Corporaal, H., Boonstra, A.J.: A review of near-memory computing architectures: Opportunities and challenges. In: 2018 21st Euromicro Conference on Digital System Design (DSD). pp. 608–617. IEEE (2018)

21. Upadhyay, N.K., Jiang, H., Wang, Z., Asapu, S., Xia, Q., Joshua Yang, J.: Emerging memory devices for neuromorphic computing. Advanced Materials Technologies 4(4), 1800589 (2019)

22. Wang, Z., Joshi, S., Savel'ev, S., Song, W., Midya, R., Li, Y., Rao, M., Yan, P., As-apu, S., Zhuo, Y., et al.: Fully memristive neural networks for pattern classification with unsupervised learning. Nature Electronics 1(2), 137 (2018)

23. Xia, L., Li, B., Tang, T., Gu, P., Chen, P.Y., Yu, S., Cao, Y.,Wang, Y., Xie, Y., Yang, H.: Mnsim: Simulation platform for memristor-based neuromorphic computing system. vol. 37, pp. 1009–1022 (2018). https://doi.org/10.1109/TCAD.2017.2729466

24. Xie, L., Du Nguyen, H.A., Yu, J., Kaichouhi, A., Taouil, M., AlFailakawi, M., Ham-dioui, S.: Scouting logic: A novel memristor-based logic design for resistive com-puting. In: 2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI). pp. 176–181. IEEE (2017). https://doi.org/10.1109/ISVLSI.2017.39

25. Yang, J.J., Strukov, D.B., Stewart, D.R.: Memristive devices for computing. Naturenanotechnology 8(1), 13 (2013)

26. Zangeneh, M., Joshi, A.: Performance and energy models for memristor-based 1t1r rram cell. In: Proceedings of the Great Lakes Symposium on VLSI. p. 9–14. GLSVLSI '12, Association for Computing Machinery, New York, NY, USA (2012). https://doi.org/10.1145/2206781.2206786,https://doi.org/10.1145/2206781.2206786

27. Zidan, M.A., Jeong, Y., Shin, J.H., Du, C., Zhang, Z., Lu, W.D.: Field-programmable crossbar array (fpca) for reconfigurable computing. IEEE Transactions on Multi-Scale Computing Systems 4(4), 698–710 (Oct 2018). https://doi.org/10.1109/TMSCS.2017.2721160