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

D1.2– First report on new algorithmic solutions

Project Coordinator (PC):	Prof. Dr. 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:	Delft University of Technology (TUD)
Report Issue Date:	22/07/2018

Document History *(Revisions – Amendments)*	
Version and date	**Changes**
10/4/2019	First version issued by TUDelft
24/06/2019	Arm's section updated
15/07/2019	IBM's section updated
19/07/2019	Final version updated by Delft

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)	

European Commission

The MNEMOSENE project has received funding from the European Union's Horizon 2020 Research and Innovation Programme under

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

Current processors are usually based on the von Neumann architecture in which a processing unit and its memory are logically *separated* from each other and even fabricated. Consequently, when the processor needs data (or when the data generated by the processor needs to be stored), communication between off-chip memory has to be performed in order to receive (or send data). This communication imposes a huge performance and power overhead especially when a large amount of data has to be transferred. As emerging applications are mainly driven by big-data, and hence require massive data movement, the current computing architectures are just not able to deal with such applications due to energy and time constraints. Therefore, to be able to provide services for these kinds of applications, alternative computing architectures have to be explored. Computation-in-Memory (CIM) based on memristor devices is one of the most promising approaches, which combines the memory and processing functionalities in a single device. Hence, it can significantly reduce data movement, and consequently reduce the energy consumption as well as increase the overall performance while supporting massive parallelism.

In this deliverable, we will describe several applications that clearly and significantly benefit from CIM architecture based on non-volatile memories. We have identified applications from three different domains:

- Database and encryption
- Image processing
- Machine learning

For each application (domain), we will briefly introduce it, how it is traditionally implemented, how it can be implemented using a CIM architecture, and how much is the potential improvement in e.g., energy, computing efficiency, etc.. With a small modification on the source code of database application, we will partially implement it on CIM tile and measure the performance improvement by developing an analytical method. In addition, in the signal processing domain, the implementation of a new compressed sensing algorithm using CIM will show not only how CIM can reduce the complexity of computing but also how it saves energy and increases the computing efficiency. Finally, we will show how crossbar based CIM can be used to train a neural network and use weights in different ranges to achieve efficient network in terms of accuracy and power.

2. Architecture Overview

2.1 Terminology

The concept of computation-in-memory (CIM) based on memristive device aims at performing computations within the memory core. It is well-known that any memory core consists of a memory array and peripheral circuits. Memristive circuit designs, aiming at implementing CIM, produce the computing results either within the array or within the periphery. Hence, depending on where the memristive circuit produces the result, the CIM architectures can be classified into two classes:

- **CIM-Array (CIM-A):** the computing result is produced within the array. As the memory array is based on memristive devices in our targeted architecture, the output is stored in form of resistance state.

- **CIM-Periphery (CIM-P):** the computing result is produced within the periphery. When the periphery is based on CMOS technology, the output is produced (in a form of a voltage or current), and stored in a form of voltage.

The memristive circuits that enable these architectures are confined to hold at least part of the inputs to be processed (operands) in the array. In other words, the operator executed within the memory core must either hold all operands in the array (hence their logical states are all resistive), or hold only part of the operands in the array and the other part is provided via the periphery (hence their logical states are hybrid; i.e., partly resistive and partly voltage). As a result, CIM architectures can be further categorized into four subclasses:

1. CIM-A resistive inputs (**CIM-Ar**): the computing result is produced within the **array**, while the inputs are **all resistive stored in the array**. These designs have the following advantages
   - They perform computations independently from the sense amplifiers; hence, they allow high parallelism.
   - They enable logic cascading of universal functions (e.g., NAND) without reading or writing; hence, they can enable any computing kernel (function) within the memory array.

   On the other hand they have the following disadvantages:
   - They switch the output devices during computation; hence, they affect the endurance.
   - They require high control voltages to switch the output device; hence, they require large drivers (higher energy consumption).
   - They require multiple steps to cascade more complex functions, such as addition; hence, they impact the performance, energy as well as endurance.
   - They are dependent on programming (writing) the output devices conditionally (i.e., based on inputs); hence, they are sensitive to programming voltage variations.

2. CIM-A hybrid inputs (**CIM-Ah**): the computing result is produced within the **array**, while the inputs are ***partly*** *resistive stored in the* ***array*** *and partly voltage (or current) provided via the* ***periphery***. These designs have the following advantages:
   - They can perform parallel computations with all devices in the array; i.e., they are independent from sense amplifiers and are not affected by the sneak path currents.
   - They can enable partial cascading in the array; i.e., the computation results are produced within the array.
   - They can perform complex functions (i.e., ADD) primitively.

   On the other hand, CIM-Ah designs suffers from the following

   - They switch the output devices during computation; hence, affecting the endurance.
   - They require high control voltages to switch the output devices; hence requiring large drivers.

3. CIM-P resistive inputs (**CIM-Pr**): the computing result is produced within the **periphery**, while the inputs are **all resistive stored in the array**. These designs have the following advantages:
   - They produce the output in the periphery without switching memristive devices; hence, they do not affect the endurance.
   - They require low-magnitude control voltages; hence, they require small drivers.

- They are only based on sensing the BL current or voltage; hence, they are tolerant to programming voltage variations.

On the other hand, CIM-Pr design have the following disadvantages:

- They may share one SA per multiple BLs; hence, they have limited parallelism.
- They require the output to be produced in the periphery; hence, they cannot cascade functions without performing read or write operations.

4. CIM-P hybrid inputs (CIM-Ph): the computing result is produced within the **periphery**, while the inputs are **partly** resistive stored in the **array** and partly voltage (or current) provided via the **periphery**. The CIM-Ph have the following advantages:
   - They perform complex operations compared to other designs.
   - They produce the output in the periphery without switching the memristive devices; hence they do not affect the endurance.
   - They require low-magnitude control voltages; hence they require small drivers.
   - They are only based on sensing the currents or voltages; hence, they are tolerant to voltage variations.

   On the other hand CIM-Ph designs have the following disadvantages
   - They may share multiple BLs per each sensing circuit (i.e., SA, S&H or ADC); hence, the parallelism is limited by the area of the sensing circuits.
   - They require the output to be produced in the periphery; hence, they cannot cascade functions without performing cascading without performing read or write operations.

Based on the above discussion, one can clearly conclude that **CIM-P is the prime candidate** for further study. This is mainly driven by the pros related to the *endurance, reduced complexity,* and *performance*.

2.2 CIM architecture

In this section, we will shortly summarize the CIM-tile organization and its nano-ISA (ISA=instruction Set Architecture) based on the initial definition in Deliverable 3.1 and what is presented in Deliverable 4.6. CIM architecture consists of memristor array and peripheral analog as well as digital circuits. As stated in section 2.1, by using proper peripheries, memristor arrays can be used as non-volatile storage as well as powerful computation unit, to work as an on-chip accelerator alongside the general-purpose processors. Several operations can be executed within the crossbar such as write, read, vector-matrix multiplication, and logical operations. In order to do that, specialized circuits together with proper control signals and data have to be provided for targeted operations. Hence, our nano-instructions are responsible to drive the controller inside the crossbar tile to issue proper commands to configure periphery circuits. Figure 1 depicts the general architecture of our CIM tile. In the following, we will briefly explain the peripheries and their corresponding nano-instructions.

1. Data for the rows and columns of the crossbar are provided by two registers which are fed by their corresponding nano-instructions, *row select* (RS) and *write data* (WD). In addition, another register is employed to mask some columns which are not going to be used during the write operation for which new nano-instruction, *write data select* (WDS), was defined as well.
2. Since crossbar works with analog data, another important periphery, called *Digital Input Modular* (DIM), is used to translate digital data to the analog domain for the crossbar

rows and columns. In order to configure and activate DIMs, we use two other nano-instructions called *function select* (FS) and *Do Array* (DoA).

3. After a certain delay, the memory array will produce its results in the output. The *Do Sample (DoS)* nano-instruction is used to activate the sample and hold circuitry to capture the results and hold them, until the next Do Sample nano-instruction is issued. This gives time for the ensuing SAs and/ADCs to read out the results while the array can perform the "next" operation. This conceptual distinction between execution and read-out of results will allow for future code scheduling optimizations.
4. Finally, ADCs are used to translate the analog result to digital. Since they are power and area hungry, it is not feasible to allocate one ADC to each column. Therefore, several columns have to share an ADC, which means we need to have analog multiplexers. The control signals of multiplexers are provided by *Columns Select* (CS) nano-instruction. In addition, in order to activate ADCs whenever the data is ready in their inputs, new nano-instruction called *Do Read* (DoR) is used.

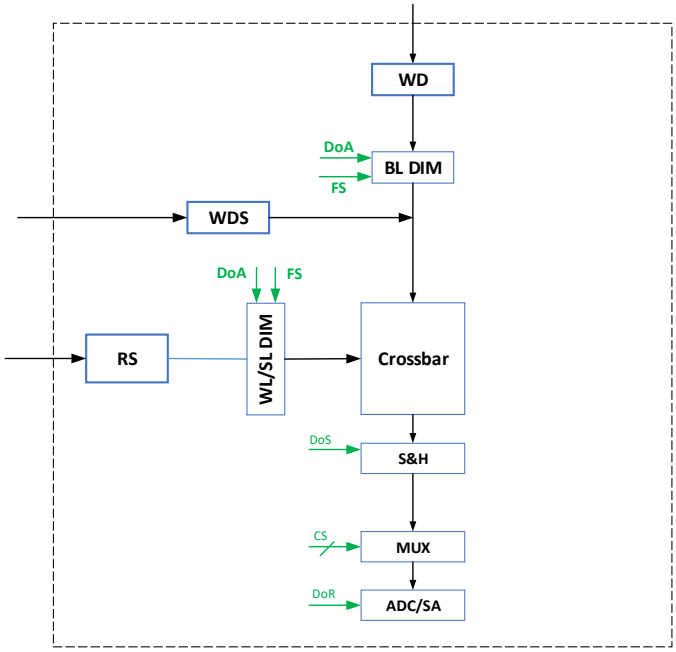

Figure 1: Simplified architecture of CIM tile

For more details about the architecture, nano-instructions, and the operations which are supported by the compiler, please read Deliverable 4.6.

2.3 Host and CIM communication

Our architecture is based on Parallel Ultra Low Power (PULP) cluster that efficiently integrates one or many CIM Macro Blocks with conventional processing cores and shared data memory. The description of the interface and communication protocol between the CIM Macro Block and the rest of processing chains in the conventional processing units was given in Deliverable 3.2. A high view of the high level architecture (Host + CIM-P) is shown in Figure 2.

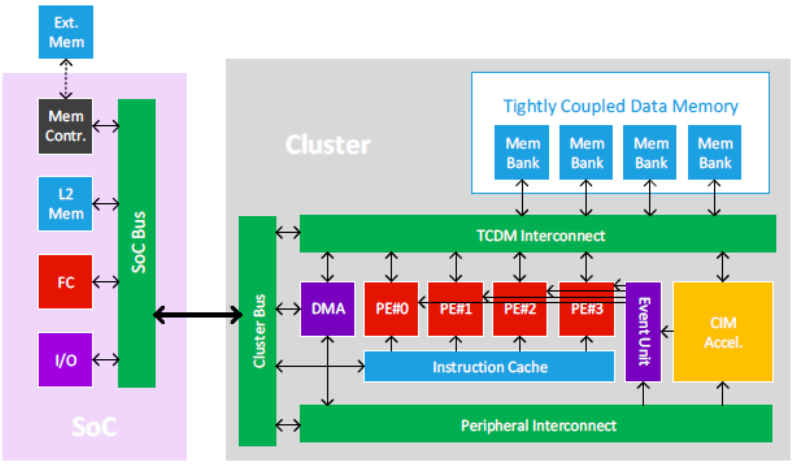

Figure 2: High-level architecture of PULP in which CIM-P is embedded

3. Overview on Target Applications

3.1 Database and encryption applications (TUD-IBM)

3.1.1 Introduction to Database applications

There are many applications which need to work with large sets of data. Large enterprises, e.g. Google, Amazon, Facebook, as well as banks employ large databases to store information. Usually, in database systems, data indexed in rows, columns, and series of tables to make data query efficient. Figure 3 presents the general structure of a database system. The systems consist of several parts which are databases, database management systems (DBMS), users, and applications. As can be observe, there are different types of database that each of which are suitable for the specific type of applications. The most well know types of database are:

1. Relational databases: present data to the user as a collection of tables that each table has a set of rows and columns
2. Object-oriented databases: information is stored in object format same as object-oriented programming
3. Distributed databases: data either may be stored on multiple computers located in the same location or distributed over a network of interconnect.
4. Hierarchical databases: data along with information related to its groups of parent and child is stored.
5. Graph databases: A graph database stores data in terms of entities(nodes) and the relationships(edge) between entities.

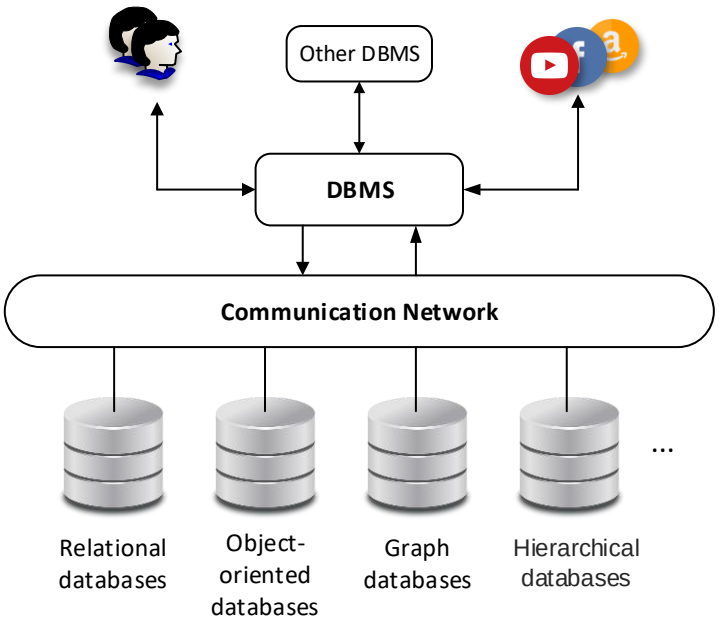

Figure 3: Structure of a database system

Table 1: An example of relational database

	A	B	C	D	E	F	G	H
Far	1	0	1	0	0	0	1	1
Near	0	1	0	1	1	1	0	0
Large	1	0	0	0	0	0	0	0
Medium	0	1	0	1	1	1	0	0
Small	0	0	1	0	0	0	1	1
New	1	0	0	1	0	0	0	0
Old	0	1	1	0	1	1	1	1

Table 1 shows an example of a relational database where 8 entries represent the information of newly discovered stars (e.g., A, B, C, etc.). Each entry (star) has 7 characteristics (e.g., Far, Near, etc.). By using zero's and one's, one can show whether a feature belongs to an entry or not. An example of a query can be a request from a user like: *"find all the stars which are newly discovered and have far distance?"*. The system should be able to process this request and find the answer in a certain amount of time by performing bitwise AND operation between first and sixth rows (i.e., row "Far" and row "New").

The most important element of the data-base system is DBMS; it refers to the comprehensive software programs used to modify, retrieve, delete, and manage all the data stored in the databases. In another word, DBMS acts as an interface between users or applications and database in order to perform users/applications commands on data ensuring that it remains organized and accessible. Usually, DBMS can manage four important things: a) data, b) the database engine which is responsible to access, modified or delete data, c) the structure of the database, and e) administrative tasks including policy changes, performance monitoring, and backup. Using administration tasks, it can limit the data for the end users or programs that can be accessed in order to satisfy high-security requirements. Additionally, DBMS can hide the physical structure of the database or even where the data is stored, which helps users to have no concern about any changes in the

structure of the database. Some examples of DBMS are MySQL, Microsoft SQL server, etc. Using DBMS to manage data has some advantages as mentioned next:

- Data can be accessed and shared by multiple users while DBMS takes care of data integrity
- It removes data redundancy since central storage of data which is accessible by multiple users is provided.
- It provides backup to recover from crashes and errors.
- Monitor all the users and programs activities
- A category can be added or removed easily without wasting memory or disturbing database

In order to perform these services, DBMS has to use CPU and memory to perform some processes which can be considered as an overhead of this management system especially when the number of queries per time increased and they have to receive a response at a limited time. Although many valuable works were performed to cope with this challenge, still more researches need to be conducted to provide higher performance for database systems as the number of applications and users that use these systems, as well as the data size, are continuously increasing.

3.1.2 Introduction to Encryption applications

Encryption is the process of converting information to an unreadable or meaningless form, which can only be decoded, read, and understood by persons or devices for whom the information is intended. To make the information encrypted, typically some mathematical operations need to be applied to the raw information. In order to achieve a higher level of security, the concept of key-based encryption was proposed. By using a key, only people who hold it can get access to the information. Key is a series of numbers that can be generated by an algorithm, digital fingerprint, etc. used for decoding and encoding of the information.

Generally speaking, key-based encryption can be used in two ways:

1) Symmetric encryption in which a single key used to encrypt and decrypt the information. In this method, the sender decodes the data using its own key and sends the key inside the information or in a separate communication to the receiver. Then, the recipient uses this key to encode the message. Although this method is fast and reliable, keeping the key secret is not possible in some situation. In this case, we have to use the second way.
2) Asymmetrical encryption: it is also known as public key cryptography and uses two keys to encrypt a plain text. Secret keys are exchanged over the Internet or a large network. It ensures that malicious persons do not misuse the keys. It is important to note that anyone with a secret key can decrypt the message and this is why asymmetrical encryption uses two related keys to boosting security. A public key is made freely available to anyone who might want to send you a message. The second private key is kept a secret so that you can only know. Since the recipient is the only person the private key, no one can decrypt the information.

Nowadays, encryption becomes pervasive and used in many applications. Next, just a few simple examples are given.

- Device locking: All the mobile devices and their information are protected by locking method. Whenever the screen is locked, the phone’s data is encrypted until the user

unlocks the phone. This combination of cryptographic passcode and data encryption improve the security of the device.

- Authentication: Usually, in order to access to the systems such as mailbox servers, websites, networks, etc. logging or signing process is required. Therefore, username and password need to be sent through the network to prove authenticity. If no encryption used for these data, any attack to the system can steal the information.
- Virtual private network (VPN): VPN provides this facility to the user to connect to their local network from a distant place via the internet by creating an encrypted communication channel. VPN encrypts the packet of data that need to be transferred between central and remote location by using symmetric cryptography.
- Secure web browsing: Browsers provide encrypted connections to the websites using a protocol called TLS. Each time that the user wants to connect to a website, it verifies based on asymmetric cryptography if the website is the one that user needs or not.

As the size of data which needs to be transferred through the network has been increasing, the encryption and decryption processes needs to be done faster in order to keep the at least the same level of performance. Therefore, to support this heavily work, we have to improve the performance of current processors to achieve this goal by applying creative approaches.

In Section 4, we will discuss why database and encryption applications have the potential to benefit from CIM architectures. We will also show how these two applications can be run on CIM architectures, and demonstrate how these applications can be run at lowest energy consumption an higher performance.

3.2 Signal processing applications (IMEC-IBM)

3.2.1 Compressed sensing and recovery

Reconstruction of a sparse high-dimensional signal from low-dimensional noisy measurements, for example received by sensor arrays, is used in many application fields, including radio interferometry for astronomical investigations, and magnetic resonance imaging (MRI), ultrasound imaging, and positron emission tomography for medical applications. Such reconstruction can also be applied for devices performing audio restoration or imaging, such as a mobile phone camera sensor. In the latter, one can significantly reduce the acquisition energy per image, or equivalently increase the image frame rate, by capturing only few measurements (e.g. 10%) instead of the whole image, while still being able to recover the original image accurately.

In Deliverable 1.1, we presented an approach to perform compressed sensing and recovery of sparse signals using CIM. It exploits the use of crossbar arrays of resistive memory devices to perform the matrix-vector multiplications associated with both the compression and recovery tasks. In this way, as shown in Deliverable 1.1, we expect to significantly reduce the memory and computing resources needed to solve the problem as well as its computational complexity, at the cost of potentially reducing solution accuracy.

In this deliverable, we present a concrete application of this concept to the compression and reconstruction of image signals. Since images are not naturally sparse, additional steps are needed in order to perform the compressed sensing and associated recovery. We therefore expand the approach presented in Deliverable 1.1 in order to be able to process image signals. We also present a block-based compression approach that can be used to compress and recover large images with CIM tiles of limited size.

3.2.2 Streaming signal processing applications

The next generation of image and video processing kernels often exhibit a mix of regular and irregular (or data-dependent) memory accesses. Moreover, they require data access which goes beyond the immediate local neighbors, and these data do not directly fit in the local register-files, so they need to be accessed from SRAM caches or scratchpad memories. This limits the efficient mapping of these kernels on modern GPUs. We have also presented the 'Guided Image Filtering' application as a representative illustration of this problem formulation. We have also provided a first hint on how to address this problem more efficiently in CIM style architectures. In this case, the CIM architecture is of CIM-P type according to the taxonomy presented in Section 2.1. We are actively continuing the work on this application but we only have very preliminary results so we will report the consolidated results in Deliverable1.3.

3.3 Machine-learning applications (ARM-ETH-IBM)

3.3.1 Deep Neural Networks applications

Deep Neural Networks (DNN) applications are increasingly being deployed in always-ON IoT devices. However, the limited resources in tiny microcontroller units (MCUs) limit the deployment of the required Machine Learning (ML) models. Therefore, Computation-In-Memory based on NVM crossbars are the most promising alternative to traditional architectures. Its high integration density, low power consumption and massively-parallel computation capabilities, will lead to orders of magnitude potential improvement regarding energy and computing efficiency.

As described in the Deliverable 1.1, inference systems at the very edge are particularly suited for this type of an architecture: always ON applications as the one shown in Figure 4; it can be mapped into small/medium size crossbars and that only require limited (3-8 bits) precision. Examples include audio, image or health-signal processing for event detection and classification.

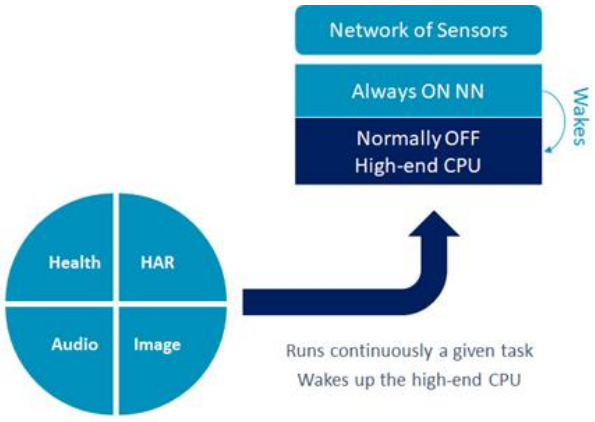

Figure 4: Always-On neural network

The deployment of systems like the above described in CIM crossbars requires highly tuned periphery from the circuit/system designers. Section 6 describe a novel mechanism to reduce this full-custom design requirement for this kind of IoT sensory applications.

3.3.2 Hyperdimensional computing

In Deliverable 1.1, we presented the application of CIM to hyperdimensional (HD) computing. HD computing (HDC) is an emerging computing framework that takes inspiration from attributes of neuronal circuits such as hyperdimensionality, fully distributed holographic representation, and (pseudo)randomness. When employed for machine learning tasks such as learning and classification, HDC involves manipulation and comparison of large patterns within memory. Moreover, a key attribute of HDC is its robustness to the imperfections associated with the computational substrates on which it is implemented. It is therefore particularly amenable to emerging non-von Neumann paradigms such as CIM, where the physical attributes of nanoscale memristive devices are exploited to perform computation in place.

Since Deliverable 1.1, significant progress has been made in designing a complete HD computing system using CIM, and hardware-friendly innovations have been developed in order to obtain the best tradeoff between accuracy and hardware complexity of the system. Since this work is still ongoing, we plan to present the complete implementation in the final Deliverable 1.3 of WP1.

4. Database and Encryption Applications

As already discussed in Section 3.1, DBMS is responsible for managing databases and perform users' queries on it. In order to achieve that, DBMS requires CPU and memory to execute some processes. For the large database and a complex query, CPU and memory limit the number of queries per unit of time which can be processed. Therefore, in order to service more queries per time, we need to improve the performance of the CPU, but performance improvement of current computing architecture is gradually saturating due to the three well-known bottlenecks: memory wall, power wall, and instruction level parallelism (ILP). In order to address this problem, novel architectures using emerging technology are suggested. As we discussed before, based on memristor device, we propose a computation in memory (CIM) architecture to boost the performance of computers.

In the following subsections, we compare traditional and CIM-based architectures by considering database and encryption applications. Two benchmarks which are XOR encryption and QUERY SELECTED database kernels are selected for analytical and simulation-based comparison. They are the memory intensive part of encryption and database applications. They contain a bulk bit-wise operation that can be performed on conventional and CIM-based architectures.

- The XOR encryption kernel performs an XOR operation on a string sequence and a predefined key. The size of the string sequence represents the problem size. In the case that the string sequence size is larger than the data width of a single XOR instruction in either architectures, a loop with multiple iterations has to be used.
- For the database application, query-06 of the TPC-H benchmark will be performed, which includes 22 queries. The query-06 performs compare instructions and checks if the requested data is available in the database or not. Therefore, an XOR operation followed by OR operation should be used since several tables need to be checked.

As we mentioned, the memory intensive parts of these two applications require logical operations. The CIM architecture supports vector-vector bit-wise logical operations. This means with little changes on the algorithm of these two applications, we can execute the logic operations on the CIM accelerator with the help of a general-purpose processor. In other words, parts of the applications which need massive logical operations will be executed on the CIM accelerator and the rest which is not supported by CIM is performed on CPU. We will also demonstrate that even when applications only partially execute on the CIM, significant improvements in terms of power and performance can be achieved.

The rest of this section is organized as follows. First, and for the completeness, we provide an overview of the architectures similar to what we did in Deliverable 1.1, as these form a basis for our analytical evaluation model. Subsequently, we apply the analytical model to the database application and present the achieved results. We will demonstrate, in which situation, why, and how much improvement can be achieved using CIM architecture. The simulation for both benchmarks will be presented in the future.

4.1 Traditional approach

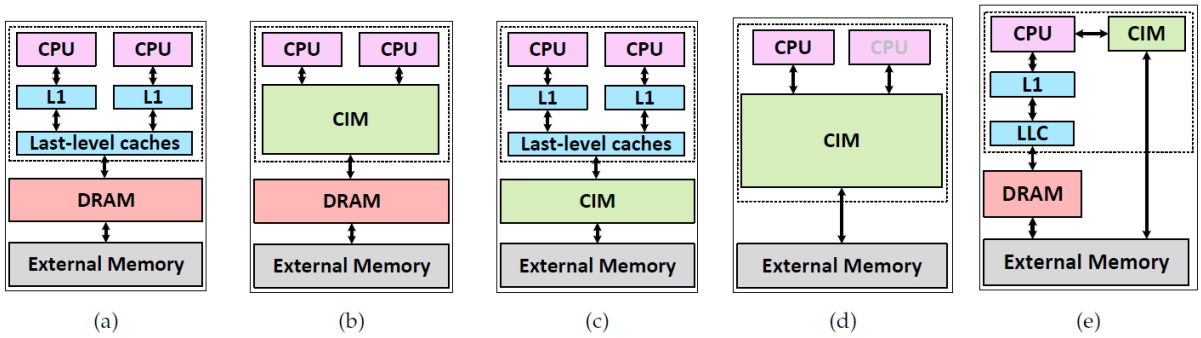

Figure 5: Conventional (a) and potential resistive architectures

Figure 5(a), depicts the traditional computer architecture where each CPU has a private and shared level of caches. In the case the required data was not found in the cache, it should be provided by memory controller from DRAM or external memory. Accessing the DRAM and external memory are time-consuming and incurs performance cost. In this architecture, obviously all parts of our benchmarks including logical and arithmetic operations will be executed on the CPUs. Due to the small caches size and depending on the behavior of applications, caches miss rate and DRAM miss (page fault) rate have huge performance overhead especially for big problem size and bad data-locality.

4.1.1 Analytical model:

In order to have analytical evaluation and comparison in terms of performance and energy between this architecture and the CIM-based one, we need to provide a model for each of them which can be used for different applications.

For the conventional architecture, we will assume Intel Xeon multi-core with the following properties:

- $np = 4$, which is the number of cores; each has a frequency of 2.5GHz
- Each core contains:

  - An ALU that is capable of executing non-memory instructions (logical, arithmetic, conditional instructions) in *one cycle*,
  - A two-level cache (L1 of 32KB and L2 of 256KB) with:
    - L1 Cache: access latencies of *one cycle,* and a miss rate of mr_{L1}
    - L2 Cache: access latencies of *two cycles*, and a miss rate of mr_{L2}.
- The cores share a main DRAM memory of 4GB with an access latency of 175 cycles (165 cycles for communication and 10 cycles for retrieving the results).
- The page fault rate (DRAM memory) is $pf_{dr} = 0.1\%$ with a penalty of 800 cycles.

In addition and in order to better characterize the application for the evaluation we are aiming at, we consider the problem size and percentage of logical instructions; note that these can be computed within the memory for CIM-based architecture as we will discuss in subsection 4.2. We define the following notations:

- n_l : to denote the percentage of logical operation (instructions)
- n_r : to denote the percentage of remaining instructions
- m_l : to denote the percentage of the corresponding memory access for n_l
- m_r: to denote the percentage of the corresponding memory access for n_r

Let's define $m = m_r + m_l$ and $m_p = \frac{m}{n_p}$, then Clock per instruction (CPI) of conventional architecture can be calculated as follows:

$$CPI_{conv} = \begin{cases} ((n\_l + n\_r).1) \,/\, n_p \; + & t_{Computation} \\ m_p.(1 - mr_{L1}).1 + & t_{L1} \\ m_p.mr_{L1}.(1 - mr_{L2}).2 + & t_{L2} \\ m.mr_{L1}.mr_{L2}.(1 - pf_{dr}).175 + & t_{DRAM-hit} \\ m.mr_{L1}.mr_{L2}.pf_{dr}.800 & t_{DRAM-p} \end{cases}$$

The CPI for conventional architecture consists of computation time (needs for ALU) as well as L1, L2, DRAM and external memory access latency. Note that, for conventional architecture, there is no difference between logical and other instructions.

4.2 CIM-based approach

Memristor is one the promising technologies due to its great scalability, high integration density, and its near-zero standby power. In this section, we discuss the CIM-based approach and the characterization of the architecture in which we are going to execute the memory intensive parts of our benchmarks. Note that using this approach, only part of the programs/benchmarks will be executed by CIM accelerator; the remaining parts are executed by the host processor; see for example Figure 5(e).

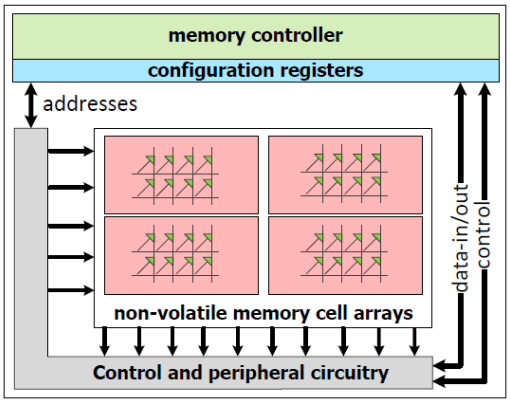

Figure 6: CIM core

Figure 6 presents an overview of CIM core which is used in our analytical and simulation parts. CIM

core consists of a memory controller, configuration registers, address decoder, and non-volatile memory cell array. The controller receives in-memory instructions from the processor. Subsequently, it activates a required row of non-volatile memory along with peripheral circuits to perform the instruction. We should notice that the CIM core used in this work only support logical operations (AND-OR-XOR) and each part of the applications which needs other operations has to be executed on the host processor.

4.2.1 Potential resistive architectures

Four possible ways to embed CIM core into the conventional architecture are presented in Figure 5.

- In Figure 5(b), CIM replaces one or more cache levels. Therefore, the in-memory operations are performed in the cache and prevent moving data toward processor, which helps to improve the performance of the system. In addition, CIM can provide large cache size due to the small footprint of the resistive devices. However, some modifications are required for both the cache controller and processor instruction set. Moreover, since caches are more frequently accessed, the endurance of the resistive devices would be a big challenge for us.
- CIM core also can be replaced by DRAM as you can see in Figure 5(c). Accessing to the DRAM is not as high as caches, which has a better impact on the lifetime of CIM core. Moreover, despite DRAM, CIM core provides non-volatile memory with high density. However, a lot of effort still required to tackle some challenges such as endurance limitation, and high memory management complexity.
- Figure 5(d) shows that CIM replaced the entire memory hierarchy. Despite the performance and endurance issues which were discussed before, this architecture could have a huge potential in terms of energy and area efficiency.
- Finally, Figure 5(e) shows the CIM core used as an accelerator. CIM just execute huge data sets that just need logical operations. Since an accelerator is often read-favored, there would be a less impact on endurance, but what is the advantage of CIM core in comparison with a conventional accelerator such as FPGA? We use CIM core to reduce the impact of data movement between CPU and traditional accelerator since a large amount of data can be stored on it despite FPGAs or GPUs. When we employ CIM as an accelerator, pragma insertion has to be used to indicate that this part of the function should be executed on the CIM. Then, this part will be translated to the in-memory instructions with pre-defined opcode in order to determine for the processor that these instructions should not be executed by itself. In addition, to have coherency between CIM and DRAM, we assume that there is no shared data between them. The CIM core is initialized with data from external memory and it needs to be performed only once. *We will evaluate the design of* Figure 5(e*) and compare it to the conventional architecture in the next section.*

Figure 7 depicts how an application source code which is written for a conventional architecture can be modified for the CIM-based architecture. As mentioned before, memory intensive parts of an application which contain logical operations is mapped to the CIM core. To force the execution of this part in the CIM, pragma's are added as shown in code 2 of Figure 7. Then the compiler translates these parts of the code to the instructions which are understandable to the CIM core. Then, the non-accelerated part of the program is executed on the host processor, while the CIM instructions are dispatched from the host to the CIM core. Moreover, the CIM core instructions are executed under a *lock/unlock scheme* to maintain data *coherency*. Since query-06 needs lots of compare (XOR) instructions without

many changes, these parts will be executed on the CIM core. However, complete query execution may require other operations that must be performed on the host. The results next will show that even by executing some parts of the application on the CIM, a remarkable gain power and performance can be realized.

```
1 ...
2 int main(int argc, char* argv[]) {
3     ...
4     for (size_t i=0; i<1000; i++) {
5         c[i] = a[i] | b[i];
6     }
7     ...
8 }
```

Code 1: Source Code for Conventional Architecture

```
1  ...
2  #include "../hmc.hpp"
3  ...
4  int main(int argc, char* argv[]) {
5      ...
6      MCPROF_START();
7      for (size_t i=0; i<1000; i++) {
8          c[i] = _cim_btw_or(&a[i], b[i]);
9      }
10     MCPROF_STOP();
11     ...
12 }
```

Code 2: Source Code for CIMX Architecture

Figure 7: Modifying an application source code for CIM based approach

4.2.2 Analytical evaluation

We already developed an analytical model for the evaluation of an application running on a conventional architecture in section 4.1. Next we will do that for CIM architecture. We assume that the CIM architecture contains both a host processor and a CIM core (see Figure 6). The following assumption are made for this architecture:

- Host processor has the same characteristics as an individual core used for the conventional architecture, and contains the following:
  - An ALU that executes non-logical instructions in *one cycle*.
  - 32KB L1 cache and 256KB L2 cache.
  - As parts of the application are offloaded to CIM core, the miss rate of other parts would be changed. To model this change, we use the square root rule of the miss rate; this defines how the cache miss rate changes for different cache sizes. However, in our models, we keep the cache sizes for both architectures the same. Instead, we assume that the same relation holds for different problem sizes (due to offloading parts of the application to CIM core). Note that when the problem size decreases, it has a similar effect of increasing the cache.
- 1GB DRAM.
- A CIM core with $n_a = 1,048,576$ parallel memory arrays (each with a size of 512 columns x 512 rows) with total capacity equal to 32GB and an area of nearly equal to 3GB DRAM.
- It is assumed that the CIM core is large enough to store all the data needed for the accelerator Unidirectional communication between the processor and CIM core is assumed to be equal to L1 access latency (i.e., 1 cycle)
- A logical instruction takes 9:3 ns on CIM core based on the Voltage Sense Amplifier (VSA)integrated in the periphery of the CIM core (see Table 2); this is equivalent to 20 CPU cycle.

In a similar way as we did for conventional architecture, The CPI for CIM-based can be derived. This is given next

$$CPI_{conv} = \begin{cases} n_r.1 + & t_{Computation-others} \\ \frac{n_l}{n_a}.4 + & t_{Computation-bitwise} \\ 1 + & t_{CPU-CIM} \\ \frac{n_l}{n_a}.4 + & t_{CIM-write} \\ m_r.(1-m_r) + & t_{L1} \\ m_r.mr_{l1}.(1-mr_{l2}).2 & t_{L1} \\ m_r.mr_{l1}.mr_{l2}.175 & t_{DRAM-hit} \end{cases}$$

Table 2 summarizes all the technology parameters and input data used for the evaluation of the two analytical models. The results of the evaluation is given in the next section.

Table 2: Evaluation parameters

Component	Parameters	
CMOS Technology		
32-bit logic gate	Delay ($cycles$)	1
	Dynamic power (uW)	33.69
	Static power (uW)	7.09
	Area (um^2)	32.65
32KB L1 cache	Delay ($cycles$)	1
	Dynamic power (uW)	0.24
	Static power (uW)	0.012
	Area (um^2)	14.2
256 KB L2 cache	Delay ($cycles$)	2
	Dynamic power (uW)	0.63
	Static power (uW)	0.108
	Area (um^2)	75
1GB DRAM	Delay ($cycles$)	175
	Dynamic power (uW)	12.86
	Static power (uW)	12
	Area (um^2)	88.98
4 GB DRAM	Delay ($cycles$)	175
	Dynamic power (uW)	51.4
	Static power (uW)	48
	Area (um^2)	355
Memristor technology		
Scouting logic operation	Delay ($cycles$)	4
	Power (uW)	8.6(OR)- 6 (AND)- 11 (XOR)
	Area (um^2)	22.3
Memristor	Write Delay ($cycle$)	1
	Write energy (fj)	1
	Static energy (J)	0
	Area (nm^2)	150
Nonvolatile memory	#col_per_row	512
	#row_per_array	512
	#array	1,048,576

4.3 Comparison and discussion

4.3.1 Analytical results

We investigate the potential improvement that CIM can realize as compared with conventional architecture by assuming the same problem size (PS), while exploring the impact of L1 and L2 cache miss rates, and the percentage of logical instructions accelerated by CIM.

Figure 8 shows the performance metric of the conventional architecture (red planes) with respect to CIMX architecture (green planes). The results of conventional architecture are normalized to those of the CIMX; note that the results of the CIMX architecture are presented by the green unit planes. The blue areas on the red planes are explained later.

- Overall, the performance speed-up of CIMX architecture increases for larger percentages of the instructions executed on CIM core (maximum 1.5x for 30% to more than 35x for 90%). This can be clearly observed as the gap between the red and green planes increases.
- CIMX architecture outperforms the conventional architecture for most of the L1 and L2 miss rates. High cache miss rates result in a longer memory access latency in the conventional architecture while CIMX architecture does not suffer much from this as computing takes place within the memory (i.e., CIM core).
- However, CIMX architecture may not achieve a speed-up for low miss rates (e.g., less than 20%). For example, when the miss rates of L1 and L2 are around 10%, a specific amount of instructions (e.g., more than 60%) have to be accelerated to realize a higher performance on CIMX architecture. Note that the miss rate of some big data applications is reported around 6-15%.

We also explore database applications which have an L1 and L2 miss rate ranging from 2 to 4%, and 40 to 90%, respectively. The performance of CIMX in these ranges are marked in blue in Figure 8 and Figure 9.

- For these particular miss rates, database applications can be executed up to 15x faster on CIMX architecture depending on the accelerated percentages. Meanwhile, CIMX architecture consumes 5x to 60x less energy than conventional architecture.
- This results in one order of magnitude improvement in terms of performance. This exploration quickly shows that database applications are a suitable candidate to be accelerated on CIMX architecture.

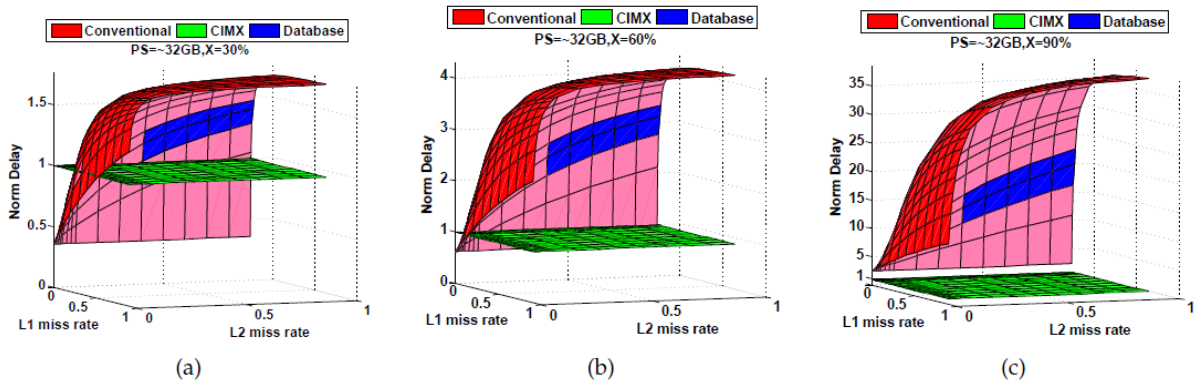

Figure 8: Analytical results of the performance (delay) metric for the conventional and CIM-based architecture

Figure 9 shows the energy metrics for both architectures. The figure reveals the following:

- Overall, similar trends are observed with respect to the percentage of accelerated instructions. However, the energy consumption of CIMX architecture is always lower, irrespective of the cache miss rates.
- In case 30% of the instructions are accelerated on CIM core, the conventional architecture consumes 1.5x to 7x less energy. This grows up to 140x in case 90% of the instructions are accelerated on CIM core

Note that the high energy consumption of conventional architecture can be partly attributed to the data movement back and forth between the CPU and memory. In addition, both cache and DRAM, suffer from a much higher leakage current as compared with the non-volatile technology.

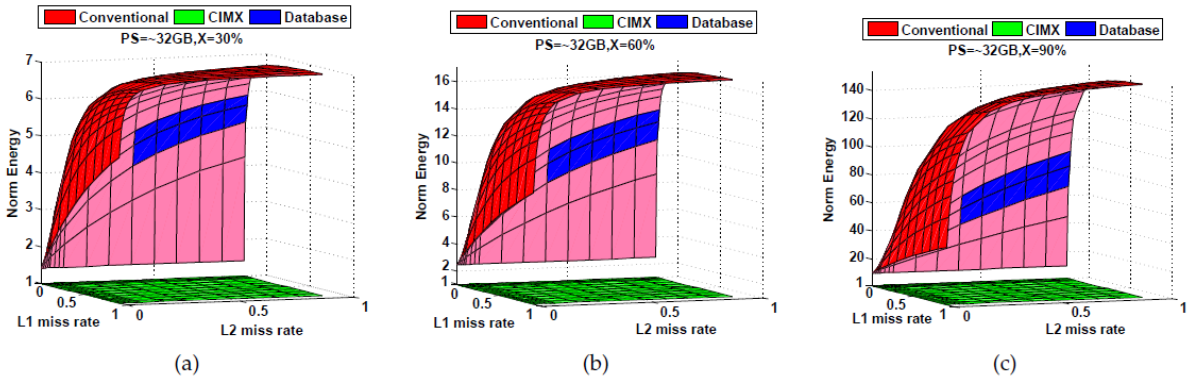

Figure 9 Analytical result of the energy metric for the conventional and CIM-base architecture

Overall, CIM architecture obtains much improvement in performance and energy on the database application. However, the accurate number depends on the application characteristics. These improvements are mainly due to the reduction in data movement and use of non-volatile technology.

5. Compressed sensing and recovery

In this section, we will describe a framework that can be used to perform compressed sensing and recovery of image signals with CIM. The implementation follows the general idea of performing compressed sensing with approximate message passing (AMP) with CIM that has been presented in Deliverable 1.1. This idea is adapted here for the specific case of image signals, which are involved in applications such as mobile phone camera sensors or MRI.

5.1 Traditional approach

Compressive imaging refers to performing compressed sensing on image signals. The entries of the input signal $x_0 \in \mathbb{R}^N$ thus represent the pixel intensities of an image. The goal is to acquire this image with $M \ll N$ measurements, and being able to reconstruct it accurately. The measurements can be modeled as

$$y = Ax_0 + w$$

where $A \in \mathbb{R}^{MxN}$ is a known measurement matrix, $x_0 \in \mathbb{R}^N$ is the unrolled image signal vector, $y \in \mathbb{R}^M$ is the measurement data vector and $w \in \mathbb{R}^M$ represents the measurement noise.

A general methodology for compressive imaging with AMP was recently introduced by Metzler et al. [C. A. Metzler, A. Maleki, and R. G. Baraniuk, "From denoising to compressed sensing," IEEE Transactions on Information Theory, vol. 62, no. 9, pp. 5117–5144, 2016]. They developed an extension of the AMP algorithm that employs a denoiser within its iterations. The proposed algorithm is written as

$$x^{t+1} = D_{\hat{\tau}_t}(A^* z^t + x^t),$$

$$z^t = y - Ax^t + \frac{1}{M} z^{t-1} \mathrm{div} D_{\hat{\tau}_{t-1}}(A^* z^{t-1} + x^{t-1}),$$

$$\hat{\tau}_t^{\,2} = \|z^t\|_2^2 / M,$$

where $x^t \in \mathbb{R}^N$ is the current estimate of x_0 at iteration t, $z^t \in \mathbb{R}^M$ is the current residual, A^* is the transpose of A, and $x^0 = 0$. D_τ denotes a denoiser which takes as input a signal plus Gaussian noise, and an estimate of the standard deviation of that noise τ. $\mathrm{div} D_\tau(x) = \sum_{n=1}^{N} \frac{\partial D_\tau(x)_n}{\partial x_n}$ denotes the divergence of the denoiser, where $D_\tau(x)_n$ is the n-th element of $D_\tau(x)$ and x_n is the n-th element of x.

The denoiser D_τ may take multiple forms depending on the type of image considered. Here, we experimented with two common denoisers:

1. **Wavelet thresholding**: It transforms the signal into a wavelet basis, thresholds the coefficients, and then inverts the transform. If W denotes the wavelet transform, this denoiser is defined as $D_{\hat{\tau}_t}(x) = W^{-1}\eta_t(Wx)$. We used the soft-threshold function

$$\eta_t(x) = \begin{cases} x - \tau_t, & x \geq \tau_t \\ x + \tau_t, & x \leq -\tau_t \\ 0, & \text{otherwise} \end{cases}$$

   and 2D Haar wavelet transform. The divergence of this denoiser can be calculated explicitly and yields $\mathrm{div} D_{\hat{\tau}_{t-1}}(A^* z^{t-1} + x^{t-1}) = \|\eta_{t-1}(W(A^* z^{t-1} + x^{t-1}))\|_0$, which is the number of non-zero elements of the thresholded sparsified estimate.

2. **BM3D (block matching 3D collaborative filtering)**: It can be considered a combination of non-local means (averaging weighted neighboring pixels) and wavelet thresholding. The term $\mathrm{div} D_{\hat{\tau}_{t-1}}(A^* z^{t-1} + x^{t-1})$ cannot be calculated explicitly and thus is estimated using the Monte-Carlo procedure described in Metzler et al. The divergence is estimated with $\mathrm{div} D_\tau(x) \approx \frac{b^*}{\varepsilon}[D_\tau(x + \varepsilon b) - D_\tau(x)]$ for small ε and vector b with entries i.i.d. $N(0,1)$. BM3D performs much better on images than wavelet thresholding because images are not exactly sparse in the wavelet domain.

5.2 CIM-based approach

As explained in D1.1, compressed sensing using CIM relies on the encoding of the elements of A as conductance values of memristive devices organized in a crossbar array, as depicted in Figure 10. The compressed measurements y are acquired by applying x_0 as voltages to the crossbar rows via digital-to-analog conversion, and obtaining y through analog-to-digital conversion of the resulting output currents at columns. The positive and negative elements

of A can be coded on separate devices together with a subtraction circuit, whereas negative vector elements can be applied as negative voltages.

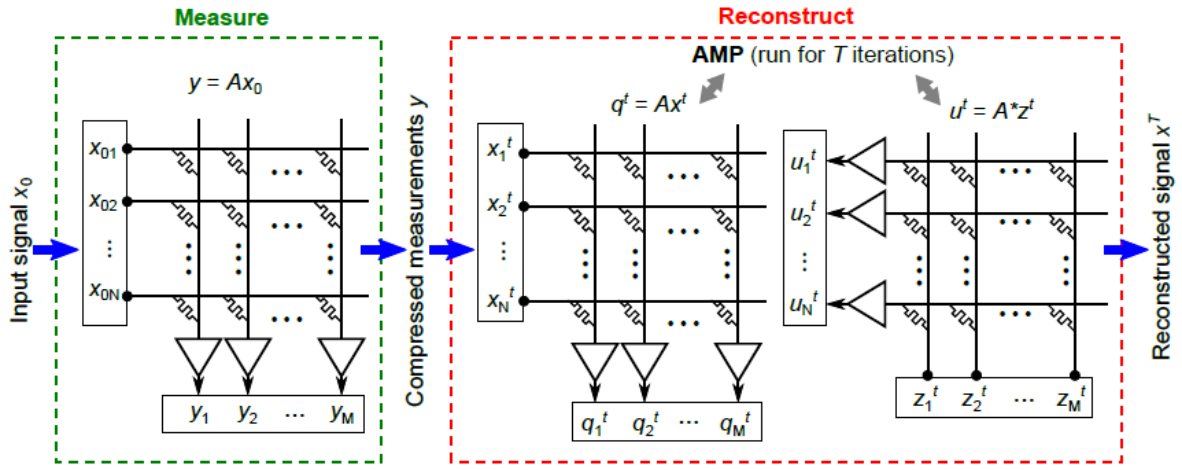

Figure 10: Proposed CIM implementation of compressed sensing with AMP recovery. A $\boldsymbol{N \times M}$ memristive crossbar encoding the measurement matrix A is used to acquire the compressed sensing measurements and to realize the matrix-vector computations $\boldsymbol{q^t = Ax^t}$ and $\boldsymbol{u^t = A^* z^t}$ of the AMP recovery algorithm. The matrix A is programmed only once in the crossbar and the same crossbar is used for the measurements and reconstruction.

Once the matrix A is programmed in the crossbar array and the measurements y are obtained, the algorithm described in Section 5.1 can be implemented in a dedicated processing unit, while the computations of $q^t = Ax^t$ and $u^t = A^* z^t$ are performed using the (same) crossbar array.

Figure 11: 128x128 pixels "house" image used for the compressive imaging experiments.

We tested this algorithm using the 128x128 pixels "house" image displayed in Figure 11 as signal x_0. The length of x_0 in this experiment is $N = 16384$. For such a large value of N, it is not possible to code all elements of a $M \times N$ Gaussian matrix in a reasonably sized CIM array, which may have maximum dimensions of up to 1024x1024 due to circuit and fabrication constrains. To overcome this difficulty, we propose a block-based compression approach whereby a small measurement matrix H of size $M_s \times N_s$ is used, with $N_s = 256$. We perform measurements on consecutive 16x16 pixels blocks using the same measurement matrix, H. In order to obtain uncorrelated measurements and ensure the convergence of AMP, we perform a (fixed) random permutation P of the pixel intensities before doing the measurements. The matrix A can be thus written as

$$A = \text{blkdiag}(H)P$$

$$A^* = P^* \text{blkdiag}(H)^*$$

where

$$blkdiag(H) = \begin{bmatrix} H & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & H \end{bmatrix}$$

is a $M \times N$ matrix with N/N_s main diagonal blocks matrices H (we assume that N is a multiple of N_s and $M_s/N_s = M/N$). The entries of H are i.i.d. Gaussian with 0 mean and variance $1/M_s$.

5.3 Comparison and discussion

We tested the block-based approach presented above with $M = N/2$ and compared with an implementation where a full $M \times N$ Gaussian matrix A is used. We performed noiseless measurements of the 128x128 pixels "house" image x_0, used wavelet thresholding as denoiser, and both methods were implemented in double-precision floating point. The evolution of the normalized mean square error (NMSE) $\|x^t - x_0\|_2^2 / \|x_0\|_2^2$ is shown in Figure 12 for the two methods. The convergence and final NMSE of the block-based approach are almost identical as for the full Gaussian matrix A. The peak signal-to-noise ratio (PSNR) at the last AMP iteration are 32.51 and 32.79 for the block-based approach and full Gaussian matrix, respectively, both of which are similar to the value of 31.86 reported in Metzler et al. for the same image, sampling rate, and denoiser (although Metzler et al. used Daubechies 4 wavelets as the sparsifying basis). It shows that the block-based implementation can be a viable solution to implement compressing imaging with images that are larger than the size that could be supported in a typical CIM.

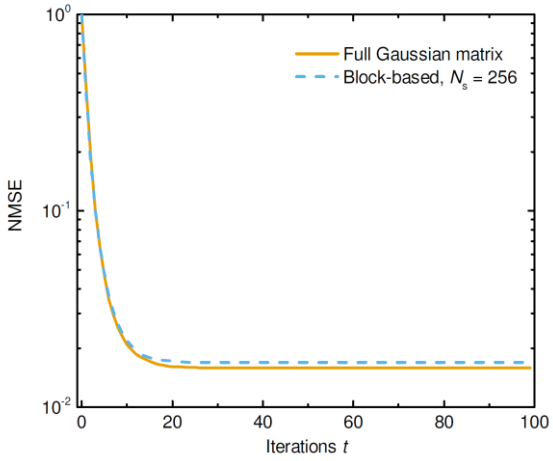

Figure 12: Evolution of normalized mean square error for the block-based approach with $N_s = 256$ and a full Gaussian measurement matrix A. We used $M = N/2$ with wavelet thresholding denoiser in double-precision floating point

Note however that this block-based approach cannot be generalized to all types of images. For example, consider an image x_0 where the pixel intensities are equal for the whole image except for one pixel. In this case, the measurements for all blocks except the one which contains the different pixel will be identical and thus correlated, which will prevent accurate reconstruction of x_0. In this study we are mostly concerned in evaluating the performance of

the CIM implementation on a typical signal-processing image with different denoisers, and the block-based approach was proposed as a means to achieve this given the limited size of the CIM. Extensions of this approach to generalize it to any type of image should be investigated through thorough numerical and/or theoretical evaluations, but are out of the scope of this work.

6. Deep Neural Networks applications

In this section we describe a HW-SW Co-design framework for analog DNN accelerators for IoT applications using NVM crossbars. This work follows the subject described in the previous deliverable, D1.1, and propose an algorithmic solution to help mapping NN into NVM crossbars, allowing reconfigurability between different applications or application versions (updated NN weights and architectures).

We propose the first training method that limits globally the set of weights/activations, improving HW blocks re-usability. This quantization algorithm obtains uniform scaling across the DNN layers independently of their characteristics, removing the need of per-layer full-custom design while reducing the peripheral HW. We validate our idea with Cifar10 CNN applications by mapping to crossbars using 4-bit devices.

The section is structured as follows: first, we summarize the traditional approach and introduce the motivation behind the developed work. Later, we describe the proposed algorithm to deploy IoT NN in CIM architectures. Finally, we present a summary with the carried out experiments and achieved results.

6.1 Traditional Approach and Motivation

Always-ON inference at the very edge, whether targeting video, image processing or a bio-signal recognition, always faces three main problems: limitations on the size of the deployed DNN, the computational load of the algorithm, and the energy consumed in the process.

The trade-off between NN size, performance and energy consumption leads to algorithm (pruning, quantization), system (digital accelerators, vectorized instructions) and technology solutions like the one targeted on MNEMOSENE. For both digital and emerging analog accelerator approaches, the precision of the operands and operations involved in the DNN determines the accuracy, latency and energy consumption of the inference operation. In particular, for CIM architectures the precision of the DACs and ADCs involved in the operation computation greatly influences the total area and power consumption.

Consequently, the quantization of both weights and activations is a critical step on the design of the accelerated system. Building up on this idea, the rest of the section briefly reviews the current quantization techniques applied to (digital) ML systems and later presents the challenges to be addressed when mapping ML algorithms to NVM tiles.

6.1.1 Brief Review of Quantization Methods

There is extensive research on the application of different quantization schemes for DNN architectures. Attending at the stage, we can identify schemes for quantizing at inference-time, training and post-training (re-train for a short number of steps an already pre-trained floating point NN). Both the characteristics of the NN and desired precision determine the stage at which apply the quantization, and, if required if the process should be gradual.

Attending at the tensor quantization itself, different methods as uniform affine, uniform symmetric, stochastic or non-uniform methods can be applied.

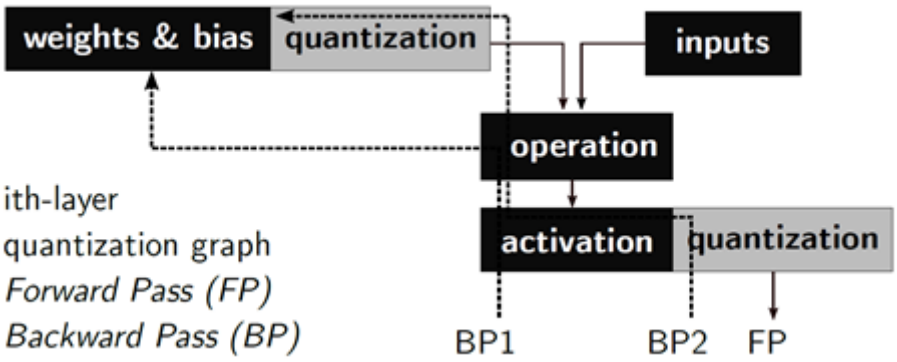

Figure 13: Simplified version of a quantized graph. Forward passes are quantized following the desired method. During Back-Propagation stages (dotted) the gradient computation may or may not be consider quantization blocks.

Finally, different quantization aware training methods can be implemented. Figure 13 presents a simplified training graph. While in forward passes (FP) both weights and activations are always quantized, the computation of the new weights using the computed gradient during back propagation (BP) stages may or may not use the quantization blocks. Schemes like the one used in TensorFlow, and introduced by Google's, use the standard floating point operation to compute the gradient on the BP. On the other hand, schemes like UNIQ or IBM's PAC introduce noise on the gradient computation.

6.1.2 NVM Crossbar For CIM Challenges

Machine Learning applications, and more specifically DNN, heavily rely on vector-matrix multiplication operations --also called multiplication-accumulation or MACs. Being MAC a very basic but high cost operation, the implementation of an analog, low power, massively-parallel MACs acceleration block is highly desired. Resistive crossbars composed of NVM elements --PCM, RRAM, MRAM-- can immediately compute MAC in constant time with a very promising energy efficiency improvements.

As depicted in Figure 14, the set of accurately programmed conductances in the NVM devices conforms the matrix G={G_ab*}, a=[1,N], b=[1,M].* If we encode our input vector as voltages *V = {V_a}*, the current flowing through each one of the bitlines *I = {I_b}* corresponds to the accumulation of the partial products *I_b = sum(V_a G_a)*.

Despite many efforts have been devoted to design NVM based accelerators, most works presented in literature rely on HW external to the chip to assist the crossbar as supporting periphery.

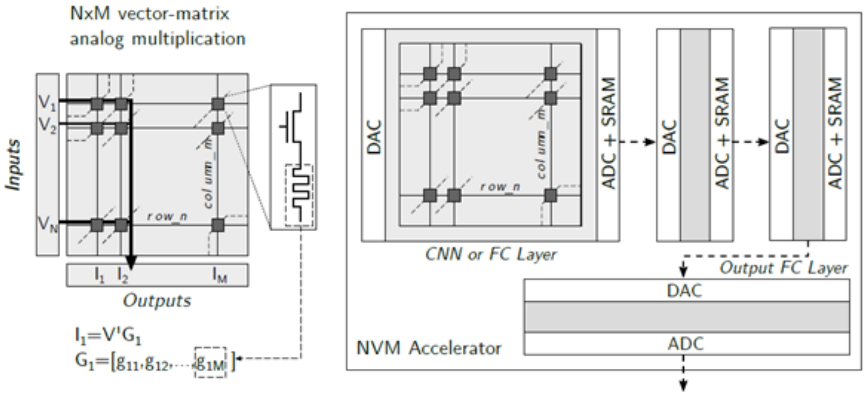

Figure 14: Principle behind the NVM based vector-matrix multiplication and basic NVM accelerator architecture. Bitline currents naturally accumulate the individual products of the input voltages and the NVM conductances and later digitized. In very deep NN crossbars are interconnected digitally.

6.1.3 NVM Technology Challenges

Resistive switching NVM are still immature and face challenging problems. Limited endurance after many writes is one of the main problems that remains unsolved. Though in-crossbar training methods have been proposed, their applicability to real products is still unclear due to this reduced lifetime.

However, always-ON inference applications, performing only analog read operations, would not suffer from this problem. Second, variability and crossbar-related errors heavily affect RRAM-CMOS hybrid circuits. However, architectures as DNNs are naturally robust against the noise that these problems may cause. Moreover, this defects can be taken into account at training time getting around device faults.

6.1.4 Challenges Related To Precision

Though 6-bit and 8-bit NVM devices have been demonstrated, and depending on the technology, variability or analog noise may compromise the encoding of more than 2 to 4 bits per cell, and thus limiting the precision of the analog multiplication.

Moreover, ADC precision is the main factor behind the total area and power consumption, and so lower-precision ADCs are desired.

To the best of our knowledge, every offline learning work present in the literature that trains the DNN externally to the NVM crossbar dynamically scale each DNN layer to the available set of conductances in which we can program the device.

This process is independent of the input, weight and activation value ranges of the layers, and as we are dealing with real voltage/current signals, this complicates the underlying HW periphery, and limits the system reconfigurability.

Consider the layers *A* and *B* of a NN described in Figure 15a. The number of inputs *n(X_A),* neurons *n(Y_A)* differs from those in layer *B*. Similarly, their ranges *[x_A0, x_A1], [y_A0, y_A]* will differ from the respective ones in layer *B*. And more importantly, the weight matrices *W_A* and *W_B* differ on their ranges *[w_A0, w_A1], [w_B0, x_B1].* However, both matrices *W_A* and *W_B* need to be mapped to the same available set of conductances the devices can be programmed in, *G*.

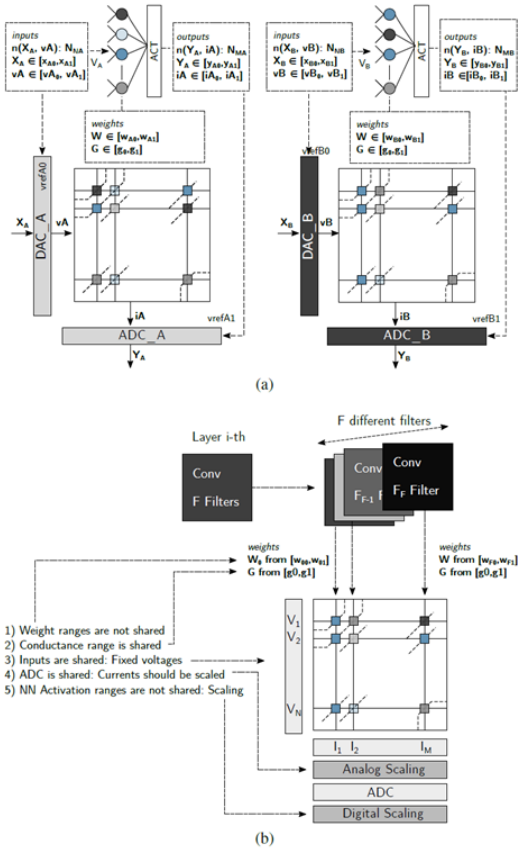

Figure 15: Different scaling across DNN layers imposes full-custom blocks per stage. The number of elements in a layer, and the range of considered inputs, weights and activations, determine the currents flowing through the bitlines. Different voltage/current signals require per-layer full-custom designed periphery.

If across layers the i-th weight matrix requires to be translated to the conductances range *[g_0, g_1],* the periphery generating the required voltage amplitudes *v_ith* and sensing the output currents *i_ith* needs to be scaled accordingly, and therefore be different. Full-custom blocks require higher design time, and limits the deployment of different NN in the same HW. Moreover, should the NN weights be updated varying voltage/current ranges, DACs/ADCs would require additional calibration processes.

To take fully advantage of the crossbar, the deployment of convolutional layers requires mapping different filters of the same layer to different columns in the tile. As described in Figure 15b, per-channel quantization methods lead to different weight/activation ranges. As voltage inputs and ADC elements are shared across the filters, analog and digital scaling stages would be required. This leads to additional area, power consumption, and higher design times.

6.2 Proposed Hard-Constrained Quantized Training

The quantization of both weights and activations is a critical step on the design of the accelerated system defining the system accuracy, area and power consumption. To avoid per-layer scaling and thus enabling system reconfigurability while reducing the area/power resources, a HW-SW co-design stage is required at training time.

To solve this problem we have developed a framework to aid mapping the DNN to the NVM hardware at training time. The main idea behind it is the use of hard-constrains when

computing forward and back-propagation passes. These constraints, related to the HW capabilities, impose the precision used on the quantization of each layer, but more importantly, guarantee that the weight, bias and activation values that each layer can have are shared across layers in the NN.

This methodology allows, after the training is finished, to map each hidden layer *L_i* to uniform HW blocks, using:

1. a single ADC design performing act()
2. a single DAC design performing to_v()
3. a single weight mapping function f()
4. a global set of activation values Y_g = [y_0, y_1]
5. a global set of input values X_g = [x_0, x_1]
6. a global set of weight values W_g = [w_0, w_1],

and being the crossbar behavior defined by

$$\begin{aligned} i_i &= \sum v_{ik} g_{ikj} \\ v_{ik} &= to\_v(x_{ik}) \\ g_{ikj} &= f(w_{ikj}) \\ y_{ij} &= act(i_{ij}). \end{aligned}$$

To achieve the desired behavior we need to ensure at training time that the following equations are met for each hidden layer *L_i* present in the NN:

$$\begin{aligned} Y_i &= \{y_{ij}\}, y_{ij} \in [y_0, y_1] \\ X_i &= \{x_{ik}\}, x_{ik} \in [x_0, x_1] \\ W_i &= \{w_{ikj}\}, w_{ikj} \in [w_0, w_1]. \end{aligned}$$

In most cases, but more commonly in classification problems the output activation does not match the hidden layers activation. Therefore for the DNN to learn the output layer should be quantized using an independent set of values *Y_o, X_o, W_o* that may or not match *Y_g, X_g, W_g*. Consequently, the output layer is the only layer that once mapped to the crossbar requires full-custom periphery.

6.2.1 HW Aware Graph Definition

As described in Figure 16, we introduce the global variable control blocks in the training graph, which manage the definition, updating and later propagation of the global variables. A global variable is a variable used to compute a global set of values *VAR_g* composed of the previously introduced *Y_g, W_g, X_g,* or others.

Each global variable can be defined as a constant --fixing the value of the corresponding global set in *VAR_g*-- or dynamically controlled using the related **global variable control**. If fixed, a design space exploration is required in order to find the best set of global variable hyperparameters for the given problem. On the contrary, we propose the use of a *Differentiable Architecture (DA)* [1] to automatically find the best set of global variable values using the back-propagation as depicted in the algorithm in Figure 17.

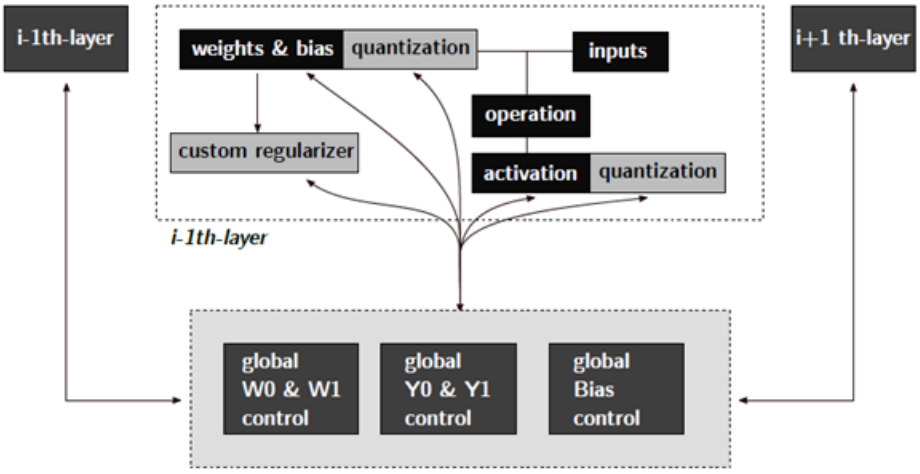

Figure 16: Simplified version of the proposed quantized graph for smart training. We define a control that automatically handles the global variables involve in the quantization process, achieving uniform quantized weight and activation values across layers

Result: Set of *global variable* values
initialization;
while *not converged* **do**
  1. Update weights W;
  2. Compute *global variables* in VAR_g;
  3. Update layer quantization parameters;
end

Figure 17 Quantized Training aided by Differentiable Architecture Search[1]

We propose defining the global variables as a function of each layer characteristics --mean, max, min, deviations, etc. If complying with DA requirements, the global control elements automatically update the related variables descending through the gradient computed in the back-propagation stage.

6.2.2 Supporting Libraries

Located at MNEMOSENE software repository}, a set of open libraries give support to the proposed methodology within the proposed framework. The libraries are based on standard TensorFlow running with Keras syntax. The experiments shown in this work uses the following quantization scheme:

- Customizable delayed quantization: Start the quantization at a given quant_delay
- Uniform tensor quantization defined by the values given by the global variables present in *VAR_g*
- Uniform precision across layers

However, the libraries have been written modularly, and therefore the final quantization block (modules represented in light grey in Figure 16) can be directly replaced with a reference to a different function. With this scheme non-uniform or stochastic quantization schemes can be incorporated.

Similarly, the quantization stage is dynamically activated/deactivated using a global variable with could be easily substituted to support incremental approaches like *Alpha-Blending*.

And finally, though the results shown in the next section focus on obtaining uniform layers to take advantage of re-usability and reconfigurability, additional global variables can be incorporated and globally orchestrated through their corresponding global variable control elements to provide multiple-precision schemes across layers following HAQ [2] scheme.

The exploration of this design space can be achieve either manually or assited using AutoML techniques [2].

6.3 Experiments and Results

We have evaluated the presented methodology using Cifar10 classification application. Cifar10 comprises the classification of 28x28 sized images into 10 different categories.

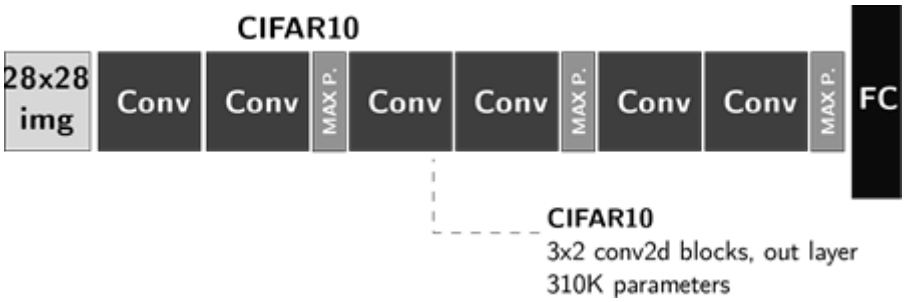

Figure 18: Example of Deep CNN Network for Cifar10

Figure 18 describes the architecture: Cifar10 problem represents a good example of always-ON medium sized DNNs, including multiple convolutional layers and 310K parameters.

After performing a quantization hyperparameter design exploration we conducted the quantized training of the use case NN using both the standard TensorFlow and the proposed approach. The code to reproduce the experiment results can be found at MNEMOSENE repository. It is to be noted that TensorFlow's scheme does not quantize the bias. This means that, when mapped to the crossbars, additional quantization studies would be needed.

Deep convolutional NN also take advantage of the proposed solution. Figure 19 shows the evolution of the DNN learning through the training process. When quantized with 4-bit (weights and activations) our solution gives accuracies only 1% away of the state of the art. Moreover, and as described in Table 3, our solution provides a significant reduction in the number of full-custom circuit modules involved in the algorithm-to-HW mapping.

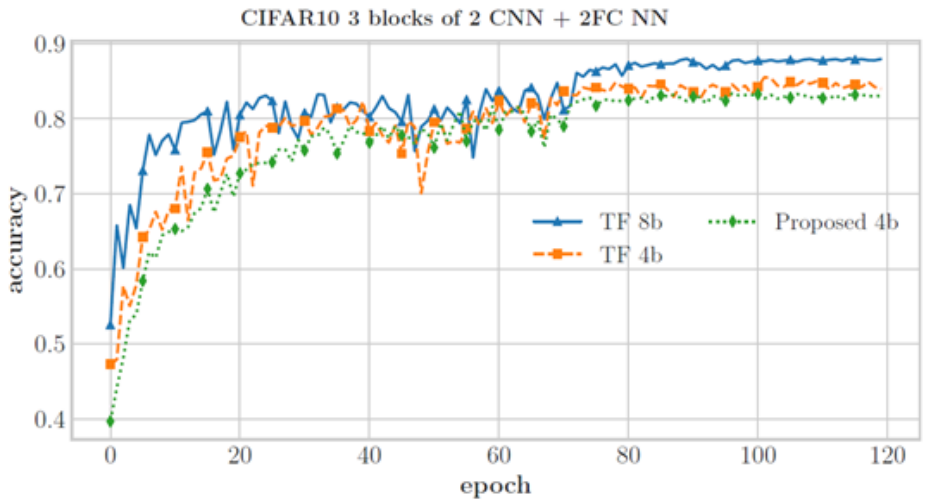

Figure 19: Cifar10 DNN. Comparison between 4-bit quantized training with TensorFlow and the proposed solution. 8-bit is shown as a baseline.

Table 3 Proposed 4b quantization scheme against TensorFlow framework.

CIFAR10 DNN QUANTIZATION SCHEMES COMPARISON

Scheme	Accuracy	# different weights	Uniform HW
TensorFlow, 8-bit	88.10%	1372	No
TensorFlow, 4-bit	84.43%	91	No
Proposed, 4-bit	83.7%	16	Yes