micromachines

Article

New Logic-In-Memory Paradigms: An Architectural and Technological Perspective

Giulia Santoro *, Giovanna Turvani and Mariagrazia Graziano

Dipartimento di Elettronica e Telecomunicazioni, Politecnico di Torino; Corso Castelfidardo 39, 10129 Torino, Italy; giovanna.turvani@polito.it (G.T.); mariagrazia.graziano@polito.it (M.G.)
* Correspondence: giulia.santoro@polito.it

Received: 30 April 2019; Accepted: 22 May 2019; Published: 31 May 2019

Abstract: Processing systems are in continuous evolution thanks to the constant technological advancement and architectural progress. Over the years, computing systems have become more and more powerful, providing support for applications, such as Machine Learning, that require high computational power. However, the growing complexity of modern computing units and applications has had a strong impact on power consumption. In addition, the memory plays a key role on the overall power consumption of the system, especially when considering data-intensive applications. These applications, in fact, require a lot of data movement between the memory and the computing unit. The consequence is twofold: Memory accesses are expensive in terms of energy and a lot of time is wasted in accessing the memory, rather than processing, because of the performance gap that exists between memories and processing units. This gap is known as the memory wall or the von Neumann bottleneck and is due to the different rate of progress between complementary metal–oxide semiconductor (CMOS) technology and memories. However, CMOS scaling is also reaching a limit where it would not be possible to make further progress. This work addresses all these problems from an architectural and technological point of view by: (1) Proposing a novel Configurable Logic-in-Memory Architecture that exploits the in-memory computing paradigm to reduce the memory wall problem while also providing high performance thanks to its flexibility and parallelism; (2) exploring a non-CMOS technology as possible candidate technology for the Logic-in-Memory paradigm.

Keywords: in-memory computing; logic-in-memory; non-von Neumann architecture; configurable logic-in-memory architecture; memory wall; convolutional neural networks; emerging technologies; perpendicular Nano Magnetic Logic (pNML)

1. Introduction

The von Neumann paradigm is the foundation of all modern computing systems. This paradigm is based on the exchange of data between a Central Processing Unit (CPU) and a memory. In particular, the CPU executes instructions on data that it retrieves from the memory, and writes back results in the memory. This data exchange mechanism is exacerbated when dealing with applications that require the manipulation of large data quantities (i.e., data-intensive applications). While through the years CPUs have become more and more powerful thanks to complementary metal–oxide semiconductor (CMOS) technology scaling, memories have not improved at the same rate, with the bandwidth being the main limitation. As a consequence, memories are not able to provide data as fast as CPUs are able to compute them. This problem is known as *von Neumann bottleneck* or *memory wall* and it limits the performance of systems based on the von Neumann architectural model as a lot of time is spent in retrieving data from the memory rather than computing them. This side effect is particularly visible when dealing with memory bound algorithms. Another critical consequence of the physical separation between the processing unit and the memory is related to the energy spent in performing memory accesses. In fact, especially for data-intensive applications, the large quantity of memory accesses required has a big impact on the overall power consumption. The very well known Moore's law, according to which the number of transistors in an integrated circuit doubles every two years, has been obeyed for decades, but the growth rate predicted by Moore is now slowing down because of the limitations that technological scaling is facing. In fact, as foretold in the 2013 International Technology Roadmap for Semiconductors (ITRS) [1], CMOS scaling is reaching a boundary where further progresses will be impeded by physical, technological and economical limitations.

The drawbacks related to the von Neumann computing model and to the CMOS technology scaling are the main factors that drive this research. On the one side, the in-memory computational paradigm is explored as an alternative to the von Neumann one. The aim is to go beyond the conventional separation between computation and storage by integrating simple logic directly inside the memory cell. We refer to this approach as Logic-in-Memory (LiM). Its key benefits are mainly: (1) Bringing the computation directly inside the memory allows one to exploit the full internal bandwidth, mitigating the memory wall problem; (2) data are computed directly inside the memory without the need to move them between the computing and the storage units, drastically reducing the amount of memory accesses and the associated energy consumption and latency. On the other side, from a technological point of view, a non-CMOS technology, namely perpendicular Nano Magnetic Logic (pNML), is considered as a possible alternative to CMOS for implementing in-memory computing architectures as it intrinsically provides non volatility and computing capabilities in the same device.

The rest of this paper is organized as follows: Section 2 presents a taxonomy of the main in-memory computing approaches, based on how the memory is used for data computation; following the proposed taxonomy, we classify the main works found in literature. In Section 3 we present the main concepts and ideas behind the Configurable Logic-in-Memory Architecture (CLiMA) that is presented here for the first time. Section 4 describes an adaptation of CLiMA for quantized Convolutional Neural Networks that in Section 5 is compared to a non in-memory architecture and Section 6 describes the adaptation for pNML.

2. State of the Art

The state of the art on in-memory computing is vast. The works found in literature differentiate from each other mainly for the role that the memory has in computing data. Starting from this observation, a taxonomy for classifying previous works was defined. According to this taxonomy the in-memory computing approaches can be divided in four main categories, as represented in Figure 1.

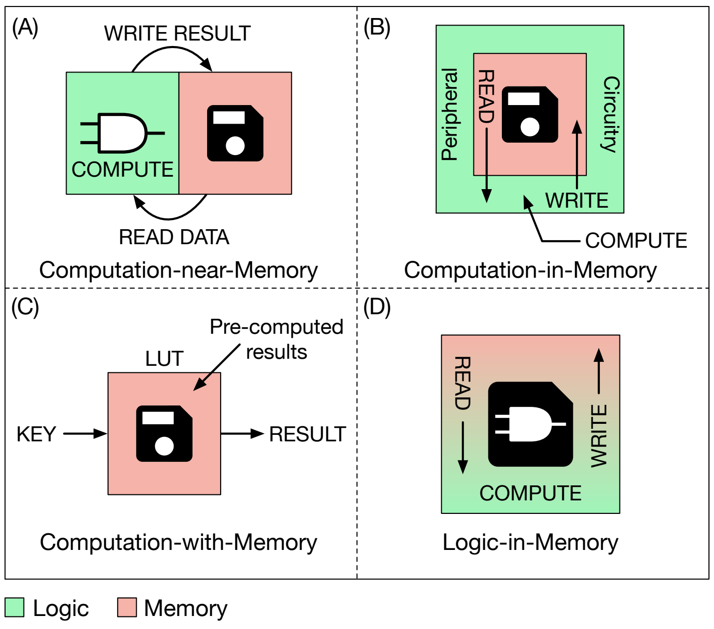

Figure 1. Depending on how the memory is used for computing data, four main in-memory computing approaches can be defined. (**A**) Computation-near-Memory (CnM): 3D-integration technologies allow one to bring computation and storage closer together by reducing the length of the interconnections. Logic and storage are still two separate entities. (**B**) Computation-in-Memory (CiM): The standard memory structure is not modified, while data computation is performed in the peripheral circuitry. (**C**) Computation-with-Memory (CwM): Memory is used as a Look Up Table to retrieve pre-computed results. (**D**) Logic-in-Memory (LiM): Data computation is performed directly inside the memory by adding simple logic in each memory cell.

The four main approaches are described in the following.

(A) Computation-near-Memory (CnM, Figure 1A): Thanks to the 3D Stacked Integrated Circuit technology (3D-SIC) [2], computation and storage are brought closer together, from which the name CnM, by stacking the two units one on top of the other. This technique has a two-fold advantage: Reducing the length of the interconnections and widening the memory bandwidth. However, this approach cannot be considered as true in-memory computing, since computation and storage are still two separate entities, but more as an evolution of conventional architectures based on the von Neumann model. Works belonging to this category are [3–8].

(B) Computation-in-Memory (CiM, Figure 1B): The structure of the memory array is not modified, while its intrinsic analog functionality is exploited to perform computation. In particular, in-memory computation is achieved by reading data from the memory which is then sensed by sense amplifiers (SAs). SAs are specifically modified in order to support the computation of a few simple logic operations (AND, OR, . . .). The result is then written back in the memory array. Decoders are also adapted in order to read more than one data from the array and execute row-wise (between data on different rows) or column-wise (between data on different columns) operations. Works belonging to this class are [9–14] and they all use a resistive non-volatile memory technology (RRAM). The approach followed in [15] is the same but here authors use a commodity volatile memory (DRAM, Dynamic Random Access Memory).

(C) Computation-with-Memory (CwM, Figure 1C): This approach uses memory as a Content Addressable Memory (CAM) to retrieve pre-computed results by means of a Look Up Table (LUT). The working principle of this kind of computation is that any Boolean function involving two or more inputs can be encoded in a memory by storing its truth table. In particular, input combinations are stored in a LUT, while results are stored in a CAM. Then the LUT is accessed through an input combination and an address is retrieved. These addresses are used to access the CAM and obtain the final result. Works that follows this approach are [16–20].

(D) Logic-in-Memory (LiM, Figure 1D): In this case logic is directly integrated inside the memory cell. Differently from the other three approaches, here data are computed locally without the need to move them outside the array (towards a close computing unit as in a CnM approach or towards the peripheral circuitry as in a CiM approach). Internal readings are performed in order to execute operations on data stored in different cells, by exploiting inter-cells connections. Internal writings are executed to locally save the result of the operation. There are a few works belonging to this category, such as [21–24].

3. Configurable Logic-In-Memory Architecture (CLiMA): Main Ideas

Our approach to in-memory computing, while mainly targeting the Logic-in-Memory concept, is not limited to it and also exploits the other approaches when required.

The novelties that we introduce with respect to existing works are manifold:

- The idea of an architecture that exploits various approaches to in-memory computing in order to adapt to different requirements and applications (Section 3);
- Configurability, hence flexibility, at different levels:
  - The basic block of CLiMA is a 1-bit Configurable LiM (CLiM) cell that can be programmed to perform different logic and arithmetic operations (Section 4.4);
  - More 1-bit CLiM cells can be grouped together to from a multi-bit CLiM cell that supports more complex operations such as bit-wise logic operations, multi-bit addition/subtraction, multiplication, shifts (Sections 3 and 4.4);
- A data flow for Convolutional Neural Networks workload and an inter-cells connection fabric specifically optimized to minimize memory accesses outside CLiMA, to maximize data-reuse inside the CLiM array and to support high parallelism (Sections 4.3–4.5);
- A pNML-based design of the 1-bit and multi-bit CLiM cells and a small version of the CLiM array (Section 6).

We demonstrate the effectiveness of our approach by comparing CLiMA to a non in-memory Deep Learning Accelerator, showing promising results in terms of performance and a significant reduction of external memory accesses, which are the main limitations of the von Neumann bottleneck. The innovations presented in this work will be thoroughly explained and highlighted in the following sections.

3.1. Overview

Figure 2 depicts the conceptual structure, in its most generic form, of the proposed in-memory computing architecture called CLiMA, Configurable Logic-in-Memory Architecture.

The key point in the definition of CLiMA is the flexibility. In fact, the idea is to conceive an architecture that well adapts to various applications that can benefit from in-memory computing in general and this means providing flexibility on different levels. In fact, applications differ for:

- Type of operations (logic, arithmetic);
- Complexity of operations (e.g., a logic function with respect to division);
- Data movement.

These parameters have an influence on the hardware requirements of the architecture. Depending on the type of operations and on their complexity, some of them can be executed directly in memory while others cannot. For this reason, as shown in Figure 2, CLiMA is conceived as a heterogeneous architecture composed of an in-memory (LiM and/or CiM) computing unit, the CLiM arrays, and a near-memory (CnM) computing unit. Operations that can be executed in-memory are dispatched to CLiM arrays, while the ones that cannot be executed in memory are assigned to the CnM unit. Each CLiM array is composed of different CLiM cells and, eventually, some extra-array (extra-row or extra-column) logic. A CLiM cell is thought as composed of a storage cell enhanced with simple logic that can be configured to perform different types of operations, from which the name Configurable Logic-in-Memory (CLiM) cell. The extra-array logic might be needed for further data processing outside the array and it can be considered as the CiM unit of CLiMA. The flexibility of CLiMA derives from its configurability (possibility of executing operations that differ for type and complexity) and from the presence of various degrees of in-memory computation (CnM, CiM, LiM).

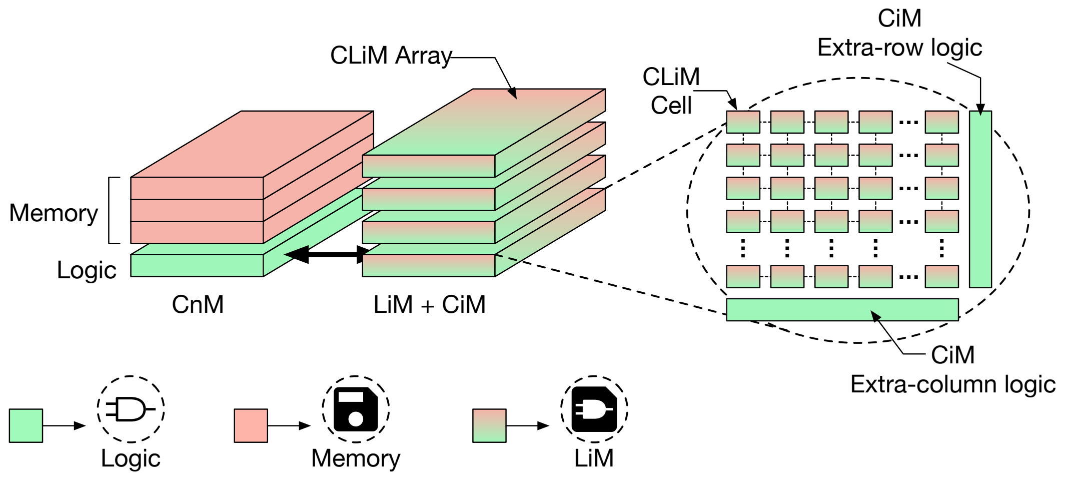

Figure 2. Conceptual structure of Configurable Logic-in-Memory Architecture (CLiMA): It can be seen as an heterogeneous unit that exploits configurability and different degrees of in-memory computation (CnM, CiM, LiM) to guarantee flexibility.

3.2. *Type of Operations and Data Movement in CLiM Array*

A more detailed view of CLiM array is shown in Figure 3.

The array is composed of CLiM cells whose reading/writing operations are controlled by bit lines (BL) and word lines (WL) as in a standard memory. Each CLiM cell is a logic-enhanced memory cell where data can be computed locally. In the example depicted in Figure 3, each CLiM cell is composed of a storage cell (MEM), a configurable logic block (CONFIG LOGIC) that can be configured to support different logic functions, and a full adder.

In addition to the local data computation inside each cell, CLiM cells are interconnected between them in order to support other kinds of operations inside the array (Figure 4):

- Intra-row computation between cells in the same row (black dashed arrow in Figure 4);
- Intra-column computation between cells in the same column (black solid arrow in Figure 4);
- Inter-row computation between two rows, an instance being an operation between a data stored in row 0 and one stored in row 1;
- Inter-column computation between two columns, an instance being an operation between a data stored in column 0 and one stored in column 1.

Intra-row connections can be exploited to implement in-memory addition. In fact, as shown in Figure 3, full adders belonging to different cells can be connected together to propagate the carry and build a Ripple Carry Adder (RCA, highlighted by the red box). Similarly, inter-row connections can be used to build an Array Multiplier (AM) by connecting two RCAs. In this way, it is possible to implement complex arithmetic functions completely in memory. The disadvantage is that RCAs and AMs are not fast arithmetic circuits, hence, applications that have a large number of additions and/or multiplications might be slowed down (especially for what concerns multiplications, since an AM is much slower than a RCA). A solution to this problem could be to delegate these operations to a fast non in-memory unit when the considered application is characterized by a very large number of arithmetic operations.

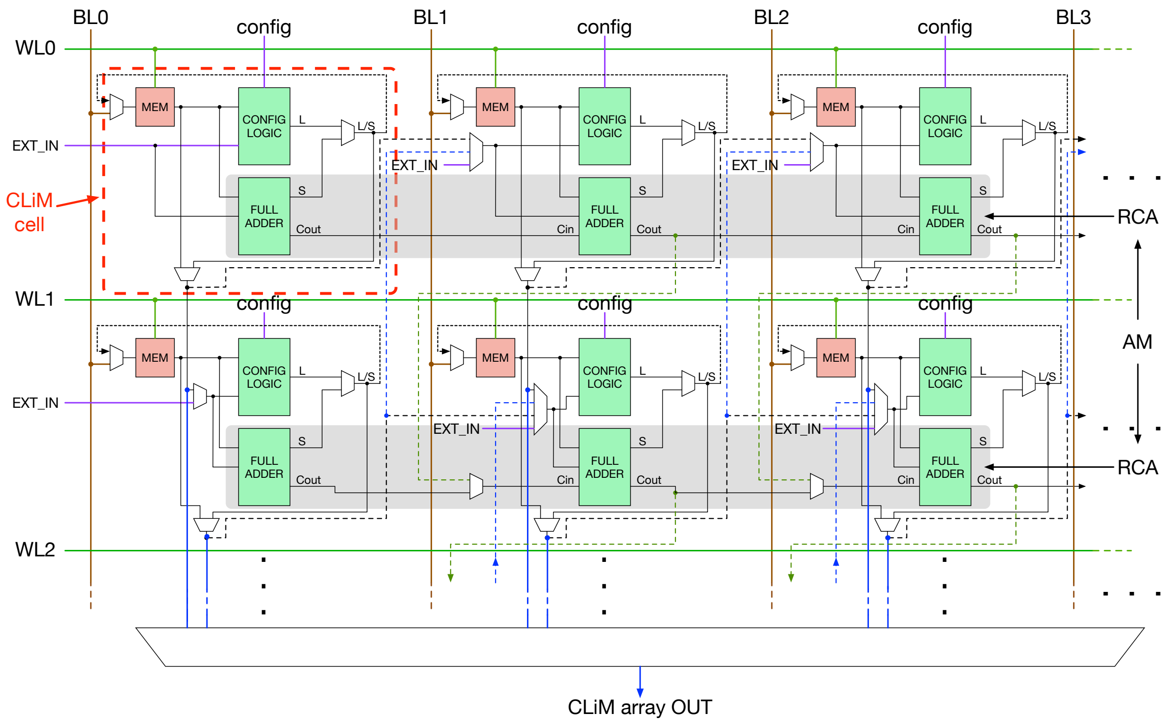

Figure 3. Detailed internal structure of the Configurable Logic-in-Memory (CLiM) array. Each CLiM cell can be represented as a logic-enhanced memory cell where data can be computed locally. By exploiting inter-cells connections it is possible to build more complex in-memory functions (e.g., a Ripple Carry Adder (RCA) or and Array Multiplier (AM)).

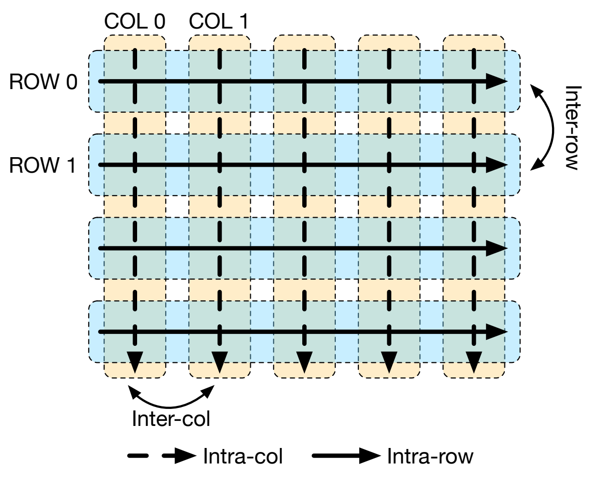

Figure 4. Possible types of data computation inside CLiM array.

4. CLiMA for Quantized Convolutional Neural Networks

On the basis of the ideas and concepts presented in Section 3, here a version of CLiMA is presented for quantized Convolutional Neural Networks. The reasons why CNNs have been chosen as target application are manifold:

- CNNs are an extremely popular application nowadays because they are a powerful method for solving many complex problems such as image recognition and classification, language processing, etc.;

- CNNs are data-intensive, hence, memory accesses represent the bottleneck;
- CNNs are computational-intensive, hence, they require hardware acceleration.

CLiMA is the ideal candidate for CNNs as it enables in-memory computation, drastically reducing the number of required memory accesses, and a high degree of parallelism, providing acceleration for time consuming applications like CNNs.

4.1. Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) [25–27] are a family of Artificial Neural Networks used for pattern recognition and classification. A CNN, as depicted in Figure 5, is composed of many 3D layers that are responsible for feature extraction and classification.

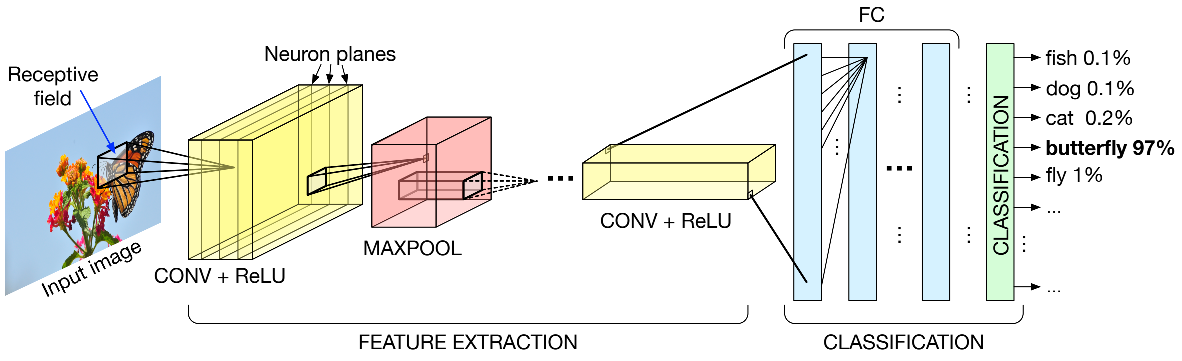

Figure 5. Convolutional Neural Networks (CNNs) are composed of different 3D layers. Each layer extracts different features from the input image.

Layers are three-dimensional as they are composed of a number of neuron planes, where each neuron analyzes a small portion of the input image, called the receptive field, extracting some key features. The feature extraction process is carried out by filtering the image with a kernel of weights (a filter), that is shared over a plane of neurons. The extraction of features by using the kernels of weights is called convolution, from which the name of the network. The output produced by the convolution operation is called the output feature map (i.e., the filtered image) and it is the input of the subsequent layer. Convolutional layers (CONV) are responsible for the extraction of features. Other type of layers are used to down-sample feature maps (e.g., maxpooling) or to introduce linear rectification (e.g., Rectifying Linear Unit (ReLU)). Fully connected (FC) layers are responsible for the actual classification.

Figure 6 shows in more detail how the convolution operation works.

The input image is usually composed of different input channels (C_{in}) with dimensions $R \times C$. The kernels used to extract features have the same number of channels C_{in} as the input image and dimensions $K \times K$, which can vary in each layer. Kernels are slid on the input feature map by a quantity called stride (S). The number of kernels (F) determines the number of channels (C_{out}) of the resulting output feature map, which has dimensions $O \times P$. The dimensions of the output feature map depend on the input image dimensions, the kernel dimensions and the stride, according to Equation (1).

$$O = \frac{R-K}{S} + 1;\ P = \frac{C-K}{S} + 1. \tag{1}$$

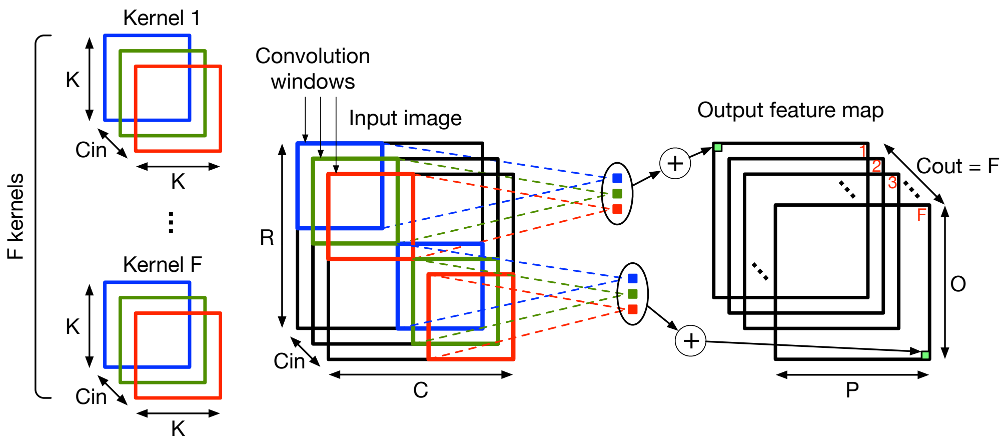

Figure 6. High-dimensional convolution operation.

CNNs are characterized by a complex structure and, over the years, network architectures have become more and more complex. The consequences of this growth are the need for very high-performance systems able to sustain such large throughput, and the increase of memory requirements because of the large number of parameters.

4.2. ShiftCNN: A Quantized CNN

Since an in-memory implementation can support only simple operations and limited precision, quantized CNNs are the perfect fit for in-memory computing architectures, since memory and computational requirements are greatly reduced in exchange for a small loss in prediction accuracy. In [28] authors propose to use power-of-two weights to eliminate the need for multiplications, which are instead transformed in simple shift operations. Moreover, according to their quantization algorithm, all weights are values of the type 2^{-n}, hence, shift operations are all arithmetic right shifts. ShiftCNN has been chosen as target application for CLiMA.

4.3. CNN Data Flow Mapping Scheme for CLiMA

In this section we present a CNN data flow mapping scheme specifically optimized for CLiMA. Differently from the commonly used unrolling technique, this mapping scheme avoids data redundancy while guaranteeing parallel computation.

The convolution operation, as highlighted in Figure 7, consists in applying a kernel of weights over the input feature map.

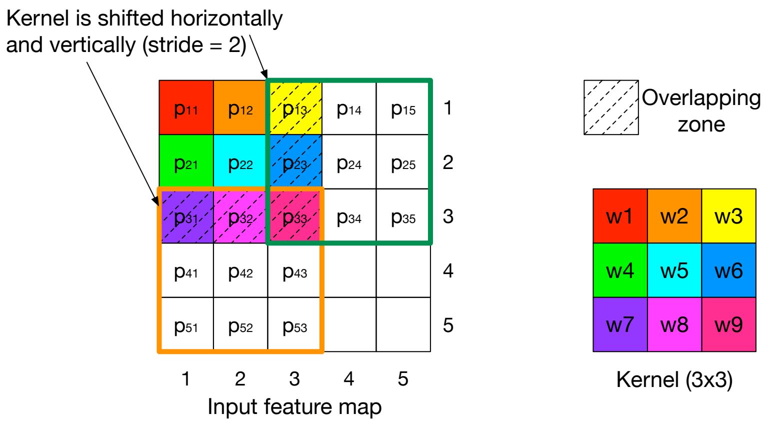

Figure 7. The kernel of weights is slid over the entire input image by a quantity called stride. The sub-region of the input image on which the kernel is applied is called convolution window. Convolution widows partially overlap.

As explained in Section 4.1, the kernel is slid horizontally and vertically by a quantity called stride. In the example in Figure 7 the stride is equal to 2. The sub-region of the input feature map on which the kernel is applied is called the convolution window. It can be seen that convolution windows partially overlap so, in order to allow parallel computation, they are unrolled and overlapping regions are replicated causing data redundancy. The impact of unrolling convolution widows is exacerbated as the size of the kernel increases and the stride decreases, since the overlapping region gets larger. The graph in Figure 8 shows how the number of input features vary when applying unrolling, for each convolutional layer of two popular CNNs, AlexNet [29] and ResNet-18 [30].

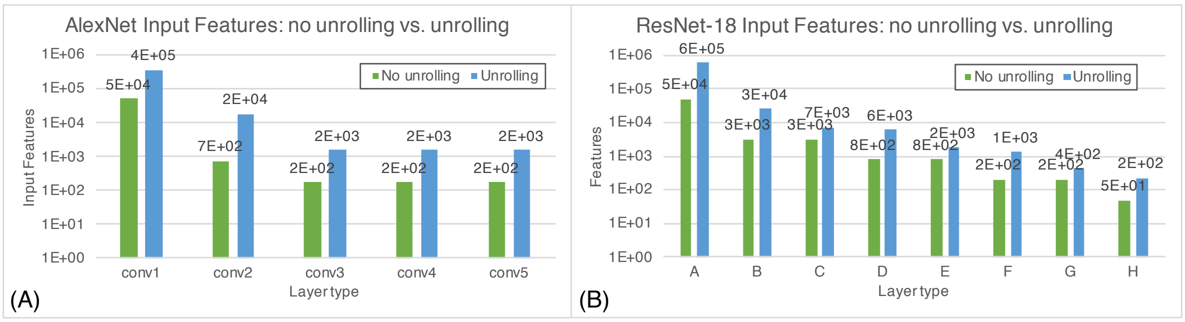

Figure 8. Data redundancy caused by unrolling in (**A**) AlexNet and (**B**) ResNet-18. Green columns represent the number of input features when applying no unrolling, blue columns represent the number of input features when applying unrolling. Input features are shown for each convolutional layer.

It can be seen that the data redundancy is not at all negligible as the number of unrolled input features (blue columns) increases of one order of magnitude with respect to the original number of features (green columns). For an architecture such as CLiMA, data redundancy is not acceptable since the storage space must be used in the most efficient way possible. For this reason, a different data flow mapping scheme is proposed. When executing convolution, not all convolution windows overlap, hence, those that do not overlap can be executed in parallel. As shown in Figure 9, the convolution operation can be divided in different steps in which only non-overlapping convolution windows are executed in parallel.

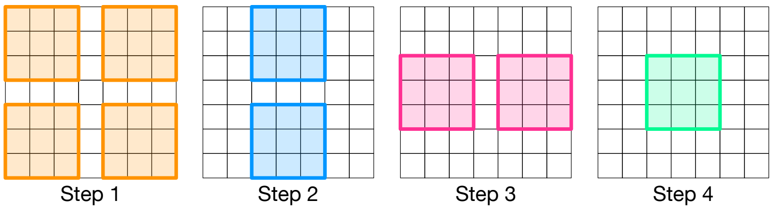

Figure 9. The convolution operation is divided in different steps. In each step only non-overlapping convolution windows are executed in parallel.

The number of steps to complete the convolution between a kernel of weights and an input feature map depends on the size of the input feature map, the kernel and on the stride. In the example in Figure 9, four steps are required to complete the convolution. The number of steps can be computed according to the following equation:

$$\#steps = \frac{tot\_conv\_windows}{parallel\_conv\_windows}. \tag{2}$$

In Equation (2), *tot_conv_windows* is the total number of convolution windows while *parallel_conv_windows* is the number of non-overlapping convolution windows that can be executed in parallel. This number can be calculated as:

$$parallel\_conv\_windows = \left(\frac{C}{K + (S-1)}\right)^2, \; K > 1. \quad (3)$$

Equation (3) is valid for kernels with dimensions larger than one ($K > 1$). When the kernel has size 1×1 the number of non-overlapping convolution windows is equal to the number of total windows. It is clear that the advantage of this parallel non-overlapping data flow scheme is to avoid data redundancy while still guaranteeing parallel computation. This scheme can be naturally mapped on CLiMA by assigning a pixel of the input feature map to each cell of the array. Weights are instead properly distributed and shifted over the array (Section 4.5).

4.4. CLiM Array Structure

Figure 10 depicts the architecture of CLiMA for quantized CNNs.

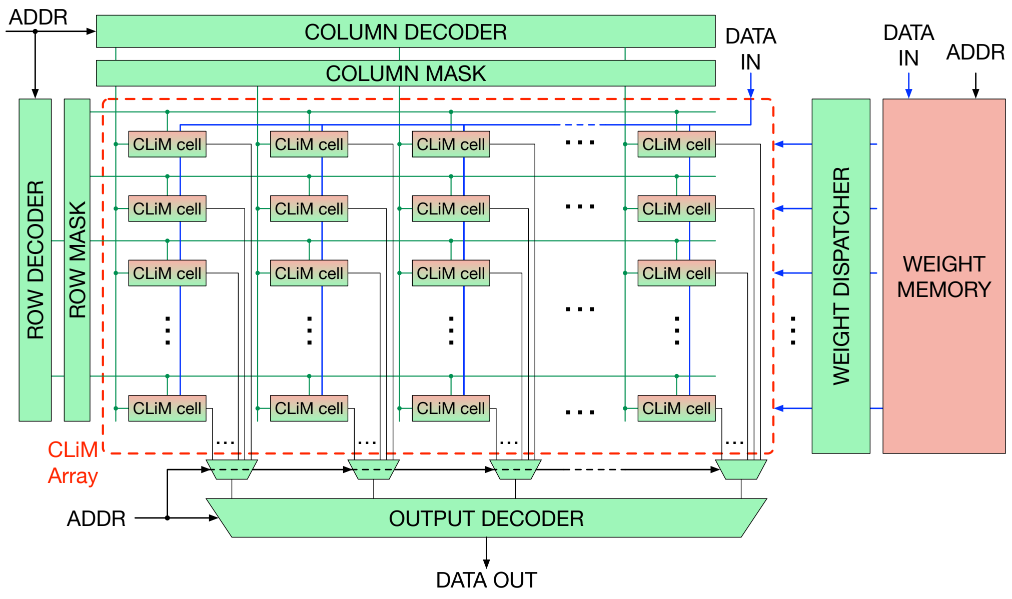

Figure 10. Architecture of CLiMA for quantized CNNs.

The main core of CLiMA is the array of CLiM cells. Each CLiM cell has both storage and computation capabilities. Modified row and column decoders are used to control the data flow inside the array. Weights are read from a weight memory, external to the array, and dispatched through a weight dispatching mechanism. More details on decoders and the weight dispatcher will be given in Section 4.5. The internal structure of the CLiM cell is shown in Figure 11.

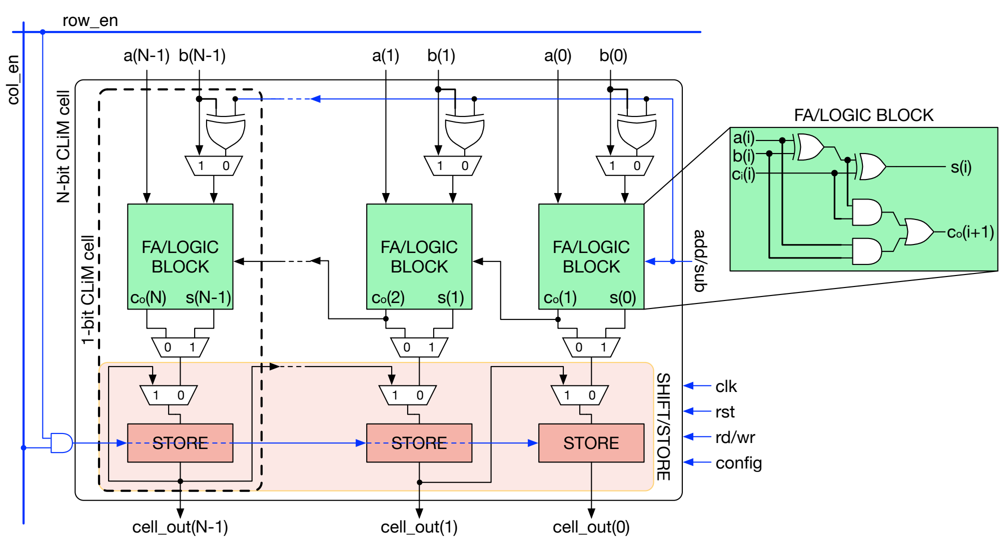

Figure 11. Internal structure of the CLiM cell. Many 1-bit CLiM cells are properly interconnected, exploiting inter-cell connections, to build a more complex N-bit CLiM cell.

It can be seen that many 1-bit CLiM cells are properly interconnected, by exploiting inter-cell connections, to create a more complex N-bit CLiM cell. Each 1-bit cell is composed of a configurable computational block, a storage cell and other simple logic. The computational block is a Full Adder (FA) that can also be used to perform logic operations by fixing one or more of the FA inputs to logic 0 or 1, as shown in Table 1.

Table 1. Logic operations that can be performed with a Full Adder by fixing one or more of the inputs. In this case A, B and C_{in} are the three inputs while S and C_{out} are the output (sum and output carry, respectively).

Fixed Input	S	C_{out}
A = 0	$B \oplus C_{in}$	$B \cdot C_{in}$
A = 1	$\overline{B \oplus C_{in}}$	$B + C_{in}$
A = 0 & B = 1	$\overline{C_{in}}$	C_{in}
A = 1 & B = 0	$\overline{C_{in}}$	C_{in}

In order to support multi-bit addition, the output carry (C_{out}) of the FA inside a 1-bit CLiM cell is connected to the input carry (C_{in}) of the adjacent 1-bit cell. By exploiting inter-cell connections it is possible to build an in-memory Ripple Carry Adder (RCA). In addition, storage cells are interconnected in a chain-like manner in order to implement a multi-bit storage block that can also work as a shift register. Only right shifts are supported in the case represented in Figure 11 since, as explained in Section 4.2, ShiftCNN requires only those. Nonetheless, with very simple modifications left shifts can also be handled. Moreover, for the sake of clarity, Figure 11 does not show the presence of redundant storage blocks (one for each 1-bit cell, in addition to the one that is also used as the shift register). The redundant storage block is used to retain partial results that will be reused for further elaboration.

The architecture depicted in Figure 10 does not show the interconnections between CLiM cells. These interconnections have been specifically designed to support CNN-like data flow inside the array. A detailed scheme of the interconnection fabric inside the CLiM array is shown in Figure 12.

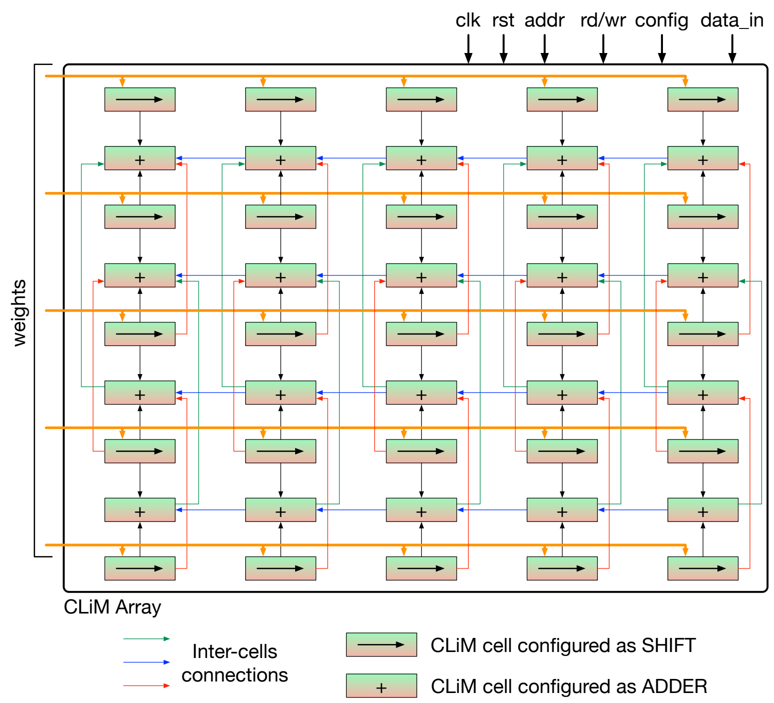

Figure 12. Interconnection fabric inside the CLiM array.

Furthermore, rows of CLiM cells are alternatively configured as shift registers (even rows) and adders (odd rows). The idea is to store pixels of the input feature map inside shift cells where they are also locally shifted according to the value of the correspondent weight. Then the shifted pixels are accumulated in the cells configured as adders. Figure 13 clarifies how convolution is managed inside the array.

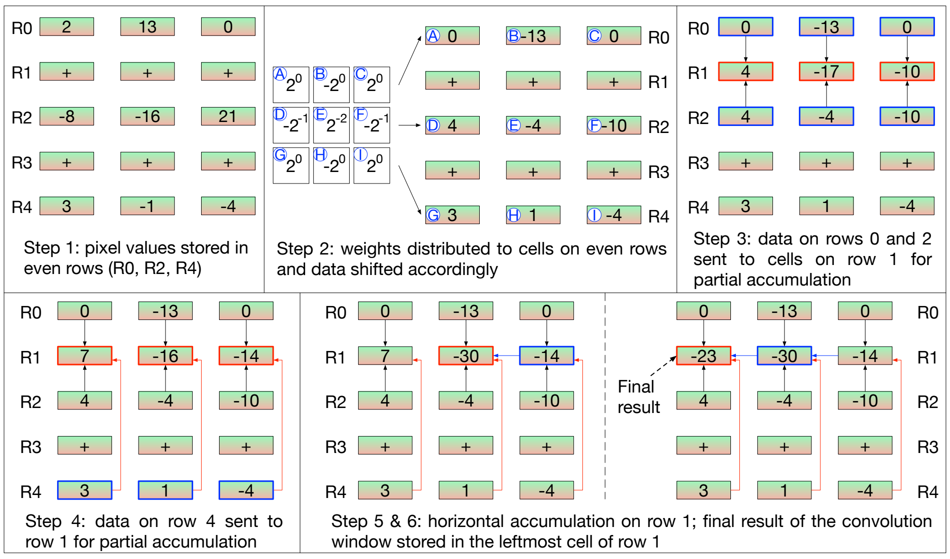
Step 1: pixel values stored in even rows (R0, R2, R4)
Step 2: weights distributed to cells on even rows and data shifted accordingly
Step 3: data on rows 0 and 2 sent to cells on row 1 for partial accumulation
Step 4: data on row 4 sent to row 1 for partial accumulation
Step 5 & 6: horizontal accumulation on row 1; final result of the convolution window stored in the leftmost cell of row 1
Final result

Figure 13. Management of convolution computation inside the CLiM array.

In particular, the computation of a 3×3 convolution window is shown as example. The interconnection fabric has been designed to be flexible, hence, it can support any kernel size.

4.5. Weight Dispatching Mechanism

In order to support the parallel non-overlapping data flow scheme shown in Figure 9, weights must be properly dispatched to the cells inside the CLiM array. In order to do so, the combined action of the weight dispatcher and row/column decoders is exploited. Row/column decoders are modified in order to activate multiple adjacent rows/columns. A starting and an ending address are provided to decoders that will consequently activate all rows/columns comprises between the starting and the ending address. Since, as it can be notice from Figure 9, parallel convolution windows might not be adjacent, row/column masks are used to disable those rows or columns comprised between the starting and ending address which must remain inactive. The weight dispatcher is used to properly shuffle weights over the array.

As highlighted in Figure 14A, the window shifting process is obtained by controlling which cells are active and which are not, step after step. At the same time, weights are properly shuffled, as shown in Figure 14B, so that they are distributed to the correct cells.

The weight dispatching mechanism has been optimized for 3 × 3 kernels since they are the most common ones. Nonetheless, other kernel sizes can be also supported.

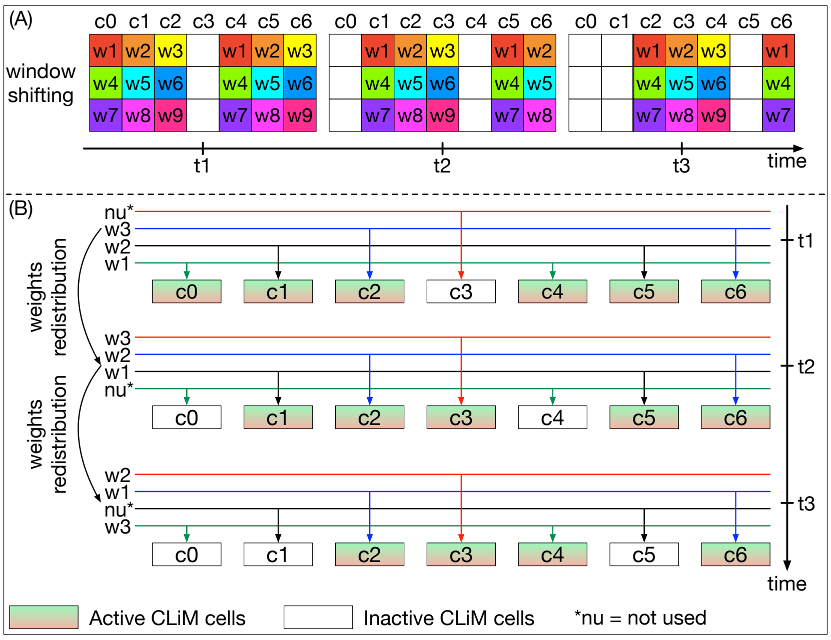

Figure 14. (**A**) Convolution windows are shifted over the array by properly activating/inactivating rows and columns. (**B**) The weight dispatcher properly distributes weights inside the CLiM array in order to reproduce the convolution window shifting process.

4.6. Data Reuse Possibilities

One of the main reasons for exploiting a Logic-in-Memory architecture such as CLiMA for Convolutional Neural Networks is the possibility of reusing data already stored and computed inside the array for further processing, without any need to move it outside.

The possibilities for data reuse in CLiMA are summarized in Figure 15 and explained in the following.

- Filters are reused across input feature maps according to the sliding window process (Figure 15A).
- Input feature maps are reused by different filters (Figure 15A).
- Partial results are reused for further processing (cross-channel accumulation) to obtain the final output feature maps (Figure 15B).

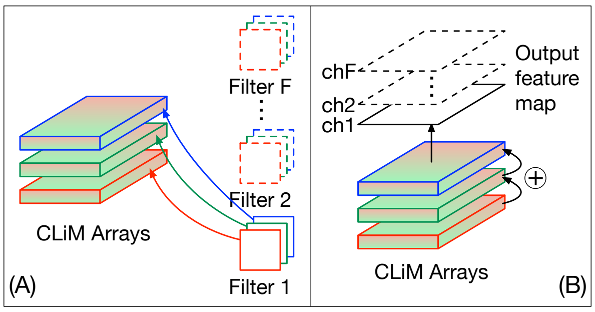

Figure 15. Data reuse in CLiMA. (**A**) Filters are reused across input feature maps according to the sliding window process. Input feature maps are also reused by different filters. (**B**) Partial results are reused for further processing to obtain the final output feature maps.

5. Results and Discussion

Before bounding it to any technology, CLiMA was modelled by using a fully parametric VHDL (VHSIC Hardware Description Language) code that was validated by means of extensive simulations and by comparing the obtained results to those obtained from an analogous model developed in MATLAB. Moreover, in order to prove the effectiveness of the CLiMA computational model, it has been compared to a conventional (non in-memory) Deep Learning Processor presented in [31,32].

An analytic computational model of CLiMA was defined. This model takes into account the following parameters:

- Convolutional layer parameters including input feature map dimensions (R, C), kernel dimensions (K), stride (S) and output feature map dimensions (O, P);
- The number of parallel non overlapping convolution windows;
- The number of execution cycles needed to complete a convolution window.

The total number of convolution widows in a layer depends on the size of the output feature map, that is given by the following equation:

$$O = P = \frac{R-K}{S} + 1. \tag{4}$$

We are assuming that input and output feature maps and kernels are square, hence, they have the same width and height ($R = C, O = P$). The total number of convolution widows, CW_{tot}, is then equal to:

$$CW_{tot} = O \cdot P = O^2 = P^2. \tag{5}$$

The number of non overlapping convolution windows, CW_{non-ov}, is given by the following expression:

$$CW_{non-ov} = \left(\frac{R}{K+(S-1)}\right)^2. \tag{6}$$

According to the data flow mapping scheme presented in Section 4.3, a certain number of steps is needed to complete a convolution operation. This number, C_{steps}, is equal to the upper bound of the ratio between the total number of convolution windows CW_{tot} and the number of parallel non-overlapping ones CW_{non-ov}:

$$C_{steps} = \left\lceil \frac{CW_{tot}}{CW_{non-ov}} \right\rceil. \tag{7}$$

The number of execution cycles, C_{cycles}, needed to complete a full convolution operation on a layer is given by the product between the number of cycles to execute a single convolution window, CW_{cycles}, and C_{steps}:

$$C_{cycles} = CW_{cycles} \cdot C_{steps}. \tag{8}$$

CW_{cycles} depends on the size of the convolution window that, in turn, depends on the size of the kernel. Moreover, by taking into account how a convolution window is mapped and executed inside CLiMA, the term CW_{cycles} can be calculated as following:

$$CW_{cycles} = 8 + 1 + \left(\frac{K-1}{2}\right) + (K-1). \tag{9}$$

In Equation (9) the following factors are taken into account:

- The number of cycles to execute shift operations; in CLiMA data are shifted 1 bit at a time. Since weights are 8-bit long, in the worst case scenario eight cycles are needed to complete the operation;
- The number of cycles to execute accumulations:
  - One cycle for partial accumulation of data couples (Figure 13, step 3); this term does not depend on the size of the kernel because these accumulations can always be done in parallel;
  - $(K-1)/2$ cycles for partial accumulation of non-adjacent data (Figure 13, step 4); this term depends on the size of the kernel, in fact, as the convolution window dimension changes the number of non-adjacent data to accumulate changes as well;
  - $K-1$ cycles to perform final horizontal accumulations (Figure 13, steps 5 and 6); similarly to the previous term, also this one depends on the size of the kernel.

Equations (7) and (9) can be substituted in Equation (8) to obtain the total number of cycles required to execute a full convolution operation of a layer.

This simple but effective computational model was used to extract results and carry out comparisons between CLiMA and the Deep Learning Processor, by considering AlexNet and ResNet-18. The Deep Learning Processor is composed of a number of Processing Elements (PEs) that are capable of performing different types of operations including Multiply-Accumulate (MAC) ones. PEs work in parallel and each of them has a throughput of 1 MAC per cycle. Assuming that each PE executes a convolution window, it takes $K \times K$ cycles to complete a single convolution window. For what concerns CLiMA, the assumption is that a certain number of non-overlapping convolution windows is executed in parallel inside the array. In order to perform comparisons, four different scenarios were considered. The difference between these scenarios is the parallelism that, for the Deep Learning Processor, is referred to the number of parallel PEs, while for CLiMA, it is referred to the number of parallel non-overlapping windows. Figures 16 and 17 report the average number of clock cycles needed to perform a complete convolution in different parallelism scenarios for AlexNet and ResNet, respectively.

The average number of clock cycles is simply calculated by averaging the number of clock cycles needed to complete the convolution of each layer in the considered CNN. In both graphs, the parallelism level is reported on the x axis, while the average number of clock cycles is shown in the y axis. It can be clearly seen that, for both the CNNs and for all the parallelism scenarios, CLiMA outperforms the Deep Learning Accelerator. In the AlexNet case, the average cycles are reduced by 78% percent in the worst parallelism scenarios (only 10 PEs or non-overlapping convolution windows). The percentage reduction slightly decreases as the parallelism increases, reaching -70% in the best parallelism scenario (60 PEs or non-overlapping convolution windows). For what concerns ResNet, the trend shown in Figure 17 is similar to the AlexNet one, except that the difference between the average cycles of CLiMA with respect to the Deep Learning Accelerator is smaller. In fact, it ranges from -49% in the worst parallelism scenario to -45% in the best.

For both the CNNs, CLiMA provides a reduction in terms of average cycles needed to complete the convolution in all the layers of the network that is higher when the parallelism level is smaller, as compared to the Deep Learning Accelerator, further proving the effectiveness of the CLiMA computational model. The reduction difference between AlexNet and ResNet-18 depends on the characteristics of the two networks (i.e., layers and kernels dimensions, number of channels etc.).

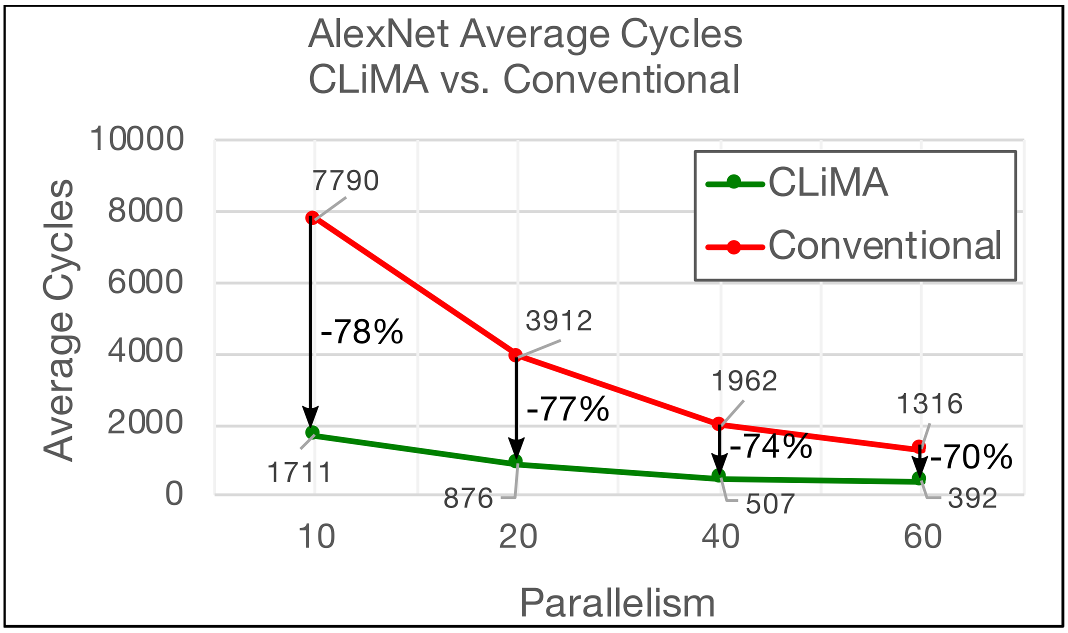

Figure 16. Average cycles needed to execute AlexNet in different scenarios: CLiMA vs. Conventional.

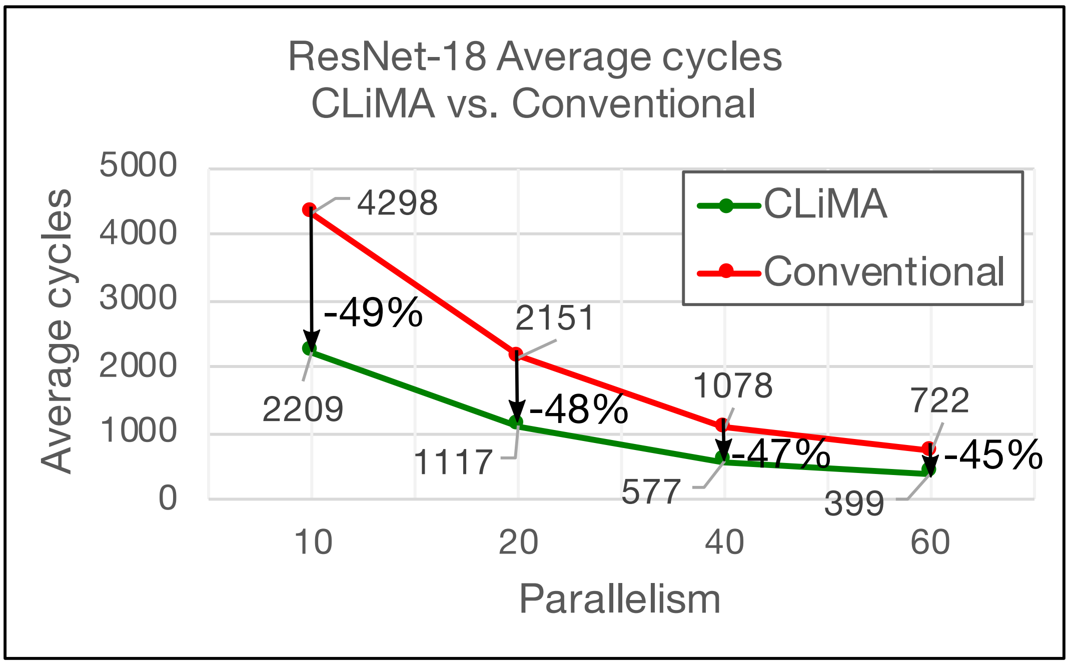

Figure 17. Average cycles needed to execute ResNet-18 in different scenarios: CLiMA vs. Conventional.

The VHDL code used to describe CLiMA was synthesized in order to get an estimation of the maximum working frequency at which the architecture can run. The technology used for the synthesis is the same used for the Deep Learning Accelerator and it is a commercial 28 nm FDSOI (Fully Depleted Silicon-on-Insulator). For both architectures a parallelism of 10 has been chosen and the maximum reachable working frequency, in both cases, is approximately 1.8 GHz. The working frequency was used to compute the execution time required by CLiMA and the Deep Learning Processor to run ALexNet and ResNet-18 when the parallelism is 10. Results are reported in Table 2.

Table 2. Performance estimation of CLiMA with respect to the Deep Learning Accelerator for AlexNet and ResNet-18 when the parallelism is 10. For both architectures the working frequency is 1.8 GHz.

CNN Type	Architecture	Average Cycles	T_{exec} (µs)
AlexNet	CLiMA	1711	0.95
	DL Acc.	7790	43.2
ResNet-18	CLiMA	2209	1.2
	DL Acc.	42,939	24

When comparing the two architectures, since the working frequency is the same, whereas the number of average cycles required by CLiMA is much lower than what the Deep Learning Accelerator requires, the resulting execution time needed to complete the convolution of Alexnet and ResNet-18 is, respectively, 45× and 20× lower for CLiMA with respect to the Deep Learning Accelerator.

The main figure of comparison between the two architectures is related to the number of memory accesses. In fact, we want to demonstrate that not only is the CLiMA computational paradigm effective in terms of execution acceleration thanks to its intrinsic massive parallelism, but it is also effective in reducing the data exchange between the processing unit and the memory. When considering CLiMA, as shown in Figure 10, we can identify the computing core that is the CLiM array and an external memory that is the weight memory. This memory is accessed to retrieve the weights that are reused over all the convolution windows inside a feature map, therefore, requiring only $K \times K$ read operations. We are assuming that the input features are already stored inside each CLiM cell of the array, neglecting the write operations required to load them for the first time as this is an initialization operation that cannot be avoided. Once the convolution operation is completed, the final results, which are then reused for cross-channel accumulation, are already stored inside the CLiM array, hence, no further external write or read operation is needed.

When considering, instead, the Deep Learning Accelerator, both input features and weights are continuously read from an input buffer and passed to the execution unit that performs MAC operations and then writes results into an output buffer. Therefore, the number of read/write operations to input/output buffers, when considering all convolution windows in a layer, is:

- $2 \times (K \times K) \times tot\_conv\_windows$ read accesses to the input buffer to retrieve input features and weights;
- $O \times P$ write accesses to the output buffer to store the convolution results.

As for CLiMA, we are not considering that input features and weights must be loaded from an external memory into the input buffer because it is an unavoidable operation.

Figures 18 and 19 show the comparison in terms of memory accesses between CLiMA and the Deep Learning Accelerator for AlexNet and ResNet-18, respectively. It can be clearly noticed that the in-memory computational model and the data reuse possibilities offered by CLiMA make it possible to drastically reduce the number of memory accesses with respect to a non-in-memory conventional paradigm, such as the one used in the Deep Learning Processor.

In general, comparing CLiMA to other architectures (either in-memory or conventional ones) is not easy because of architectural, technological and computational differences. As a result, the risk is that the comparison might be unfair. In addition, most of the time, papers lack of details about how the proposed architectures manage the computation or how there are no common comparison figures. This makes comparisons even more difficult and, for this reason, CLiMA was only compared to a conventional architecture (the Deep Learning Processor) about which we had sufficient details to be able to extract some useful data.

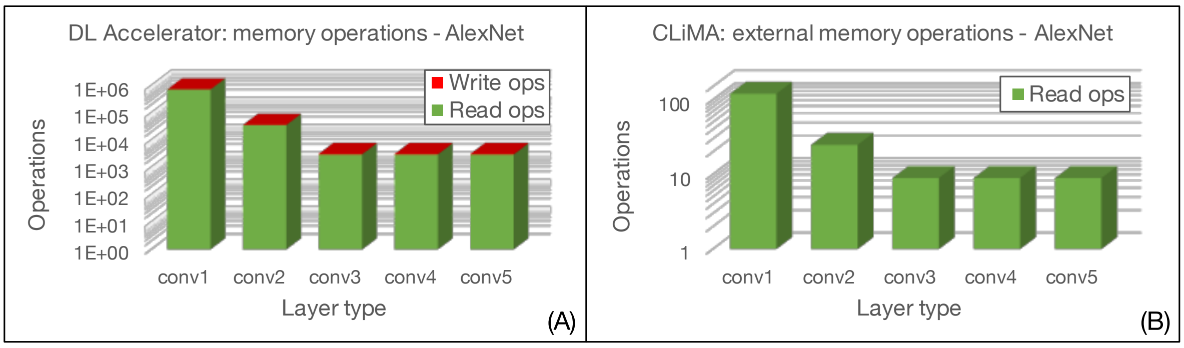

Figure 18. Memory access evaluation for AlexNet in (**A**) Deep Learning Accelerator and (**B**) CLiMA.

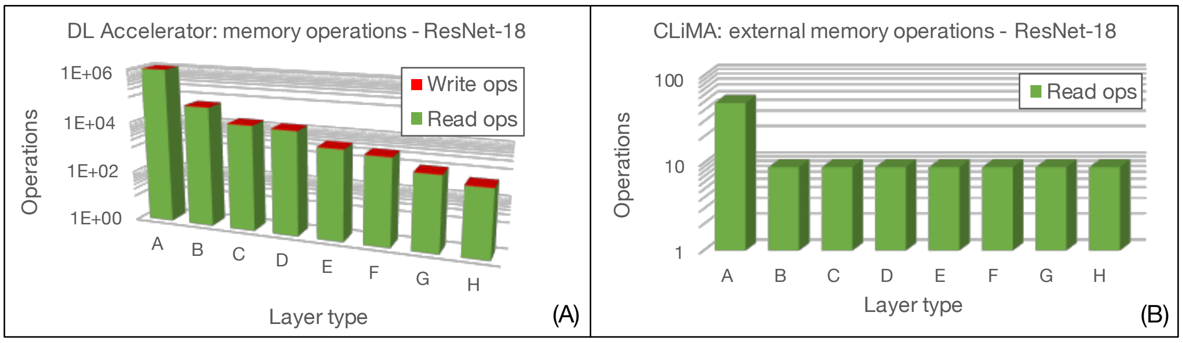

Figure 19. Memory access evaluation for ResNet-18 in (**A**) Deep Learning Accelerator and (**B**) CLiMA.

6. Beyond CMOS: A pNML Implementation

Perpendicular Nano Magnetic Logic (pNML) [33] is considered one of the most promising alternative technologies to CMOS [34] and it is perfect for in-memory computation as it intrinsically provides both non-volatility and computing capabilities [35,36]. In addition, pNML offers 3D integrability and low power consumption, all characteristics that make this technology ideal for overcoming the issues related to von Neumann architectures and CMOS scaling.

6.1. pNML Basics

pNML is based on the nanomagnet, a small (∼tens of nanometers) single domain multi-layer Co/Pt device that has bi-stable behavior. This means that, because of the perpendicular (from which the name perpendicular NML) magnetization anisotropy, it can be only in two stable magnetization states that depend on the direction of the magnetization. These states can be used to encode logic '0' and logic '1', as shown in Figure 20A.

Signal propagation in pNML depends on the magneto-static field-coupling interactions between nanomagnets [37]. In order to propagate the information in a specific direction, the magnetic properties of a small region of the nanomagnet are modified through Focused Ion Beam (FIB) irradiation [38]. The irradiated region is called the Artificial Nucleation Center (ANC). As shown in Figure 20B, neighboring pNML cells couple in a parallel or anti-parallel way, depending on their relative position, favoring signal propagation in a direction that depends on the position of the ANC. The ANC is the point where the nucleation of a domain wall starts and eventually propagates inside the magnetic device (Figure 20C). ANCs can also be obtained by changing the shape and thickness of the nanomagnet [39] (Figure 20E). The propagation of information inside pNML circuits is obtained thanks to an external magnetic field (sinusoidal as shown in Figure 20D) that is applied globally to the circuit [40]. This external magnetic field has the same function of the clock signal in CMOS circuits. Thanks to the combined action of ANCs and the clocking field, information propagation can be correctly controlled in pNML circuits. The elementary pNML blocks with which any logic circuit can be built are the inverter (Figure 20E), the notch (Figure 20F) and the minority voter

(Figure 20G and 3D version in Figure 20H). The notch works as a barrier, blocking the signal propagation unless a short depinning magnetic field is applied [41]. This block can be used to create memory elements [42,43]. Moreover, pNML technology allows one to build 3D structures by stacking different layers of nanomagnets [44–47]. Previous works such as [42,48–52] already explore the potentialities of NanoMagnetic Logic architectures (3D and non), but none of them propose a complete Logic-in-Memory design, which is instead presented in the following.

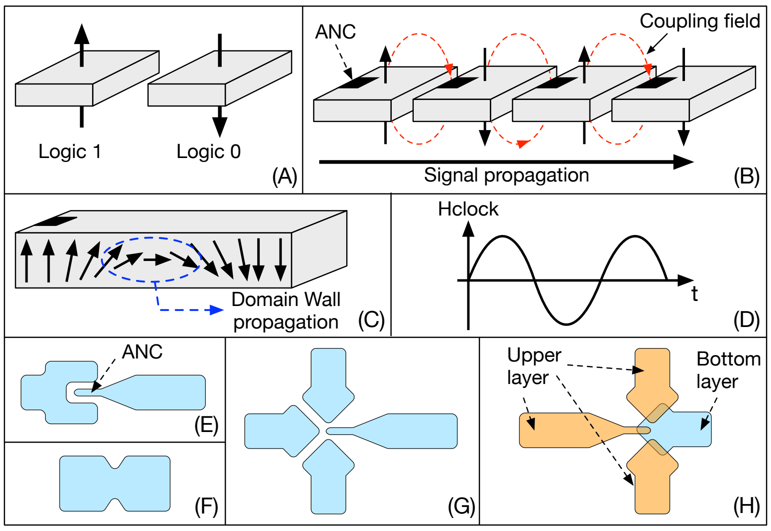

Figure 20. pNML basics. (**A**) The magnetization direction encodes logic '0' and '1'. (**B**) The Artificial Nucleation Center (ANC) guarantees correct signal propagation in a perpendicular Nano Magnetic Logic (pNML) chain of magnets. (**C**) Domain wall propagation inside the nanomagnet causes the switch of the magnetization direction. (**D**) Global out-of-plane magnetic field used as clocking mechanism. (**E**) Inverter. (**F**) Notch. (**G**) Minority voter. (**H**) 3D minority voter.

6.2. pNML-Based CLiM Array

Figure 21 depicts a small pNML-based version of the CLiM array described in Section 4.4.

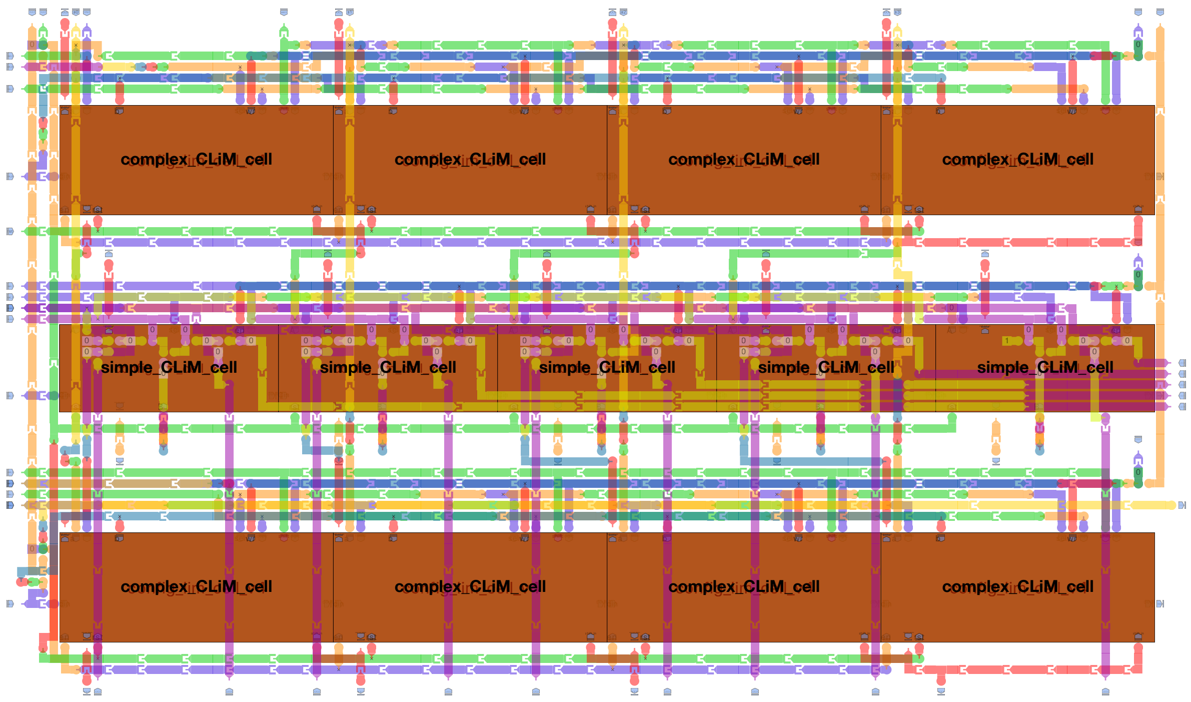

Figure 21. Small pNML-based version of the CLiM array.

The design of the array was done by using MagCAD (https://topolinano.polito.it) [53], a CAD for emerging technologies developed at the VLSI Laboratory (research group in the Department of Electronics and Telecommunication Engineering of Politecnico di Torino). MagCAD allows one to design pNML structures thanks to an intuitive and simple GUI (Graphical User Interface) in which elementary blocks can be combined together to form more complex structures and 3D designs. Starting from the designed structure, MagCAD allows the extraction of the VHDL description of the circuit, that is based on a compact VHDL model [35] of pNML devices. The generated VHDL can be used to simulate (using a common HDL simulator) and verify the functionality of the circuit [54–56]. The complexity of the pNML-based CLiM array depends on the complexity of the interconnections between CLiM cells, as it can be noticed from Section 4.4. This strongly limits the size of the design that can be implemented by hand, without any support for the routing. The design in Figure 21 uses nine layers of nanomagnets. There are two types of cells used for the pNML array, one called complex (Figure 22) and the other the simple (Figure 23) CLiM cell. Both are based on the structure shown in Figure 11, the only difference between them being that the simple CLiM cell does not support shift operations and does not have the redundant storage block. The simple CLiM cell can be used in the odd rows of the array that perform only accumulations.

Both the cells have four layers of magnets. Based on the dimensions of the nanomagnets, that in these designs are 30 × 50 nm, the area occupied by the complex cell is 22.5 µm, while the simple cell occupies 14.4 µm. The area of the CLiM array is 582 µm and interconnections occupy a big portion of it because of their complexity.

Even though in the designs here presented we have used relatively large nanomagnets (30 × 50 nm), pNML can be easily scaled to improve compactness. The designs could be also improved in order to reduce the impact of interconnections on the overall area occupation, however, as already said, the lack of an automatic and optimized routing tool makes it challenging. Nonetheless, the non-volatile nature of the technology and the total absence of current flow and leakage sources makes it an ideal beyond-CMOS technology for in-memory computing.

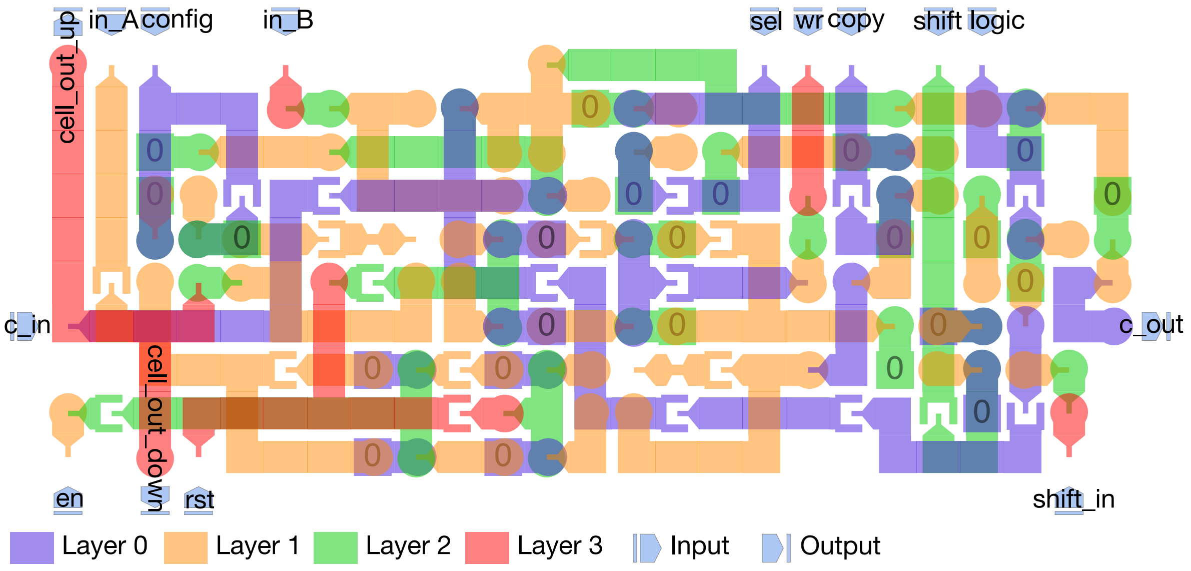

Figure 22. Complex pNML cell.

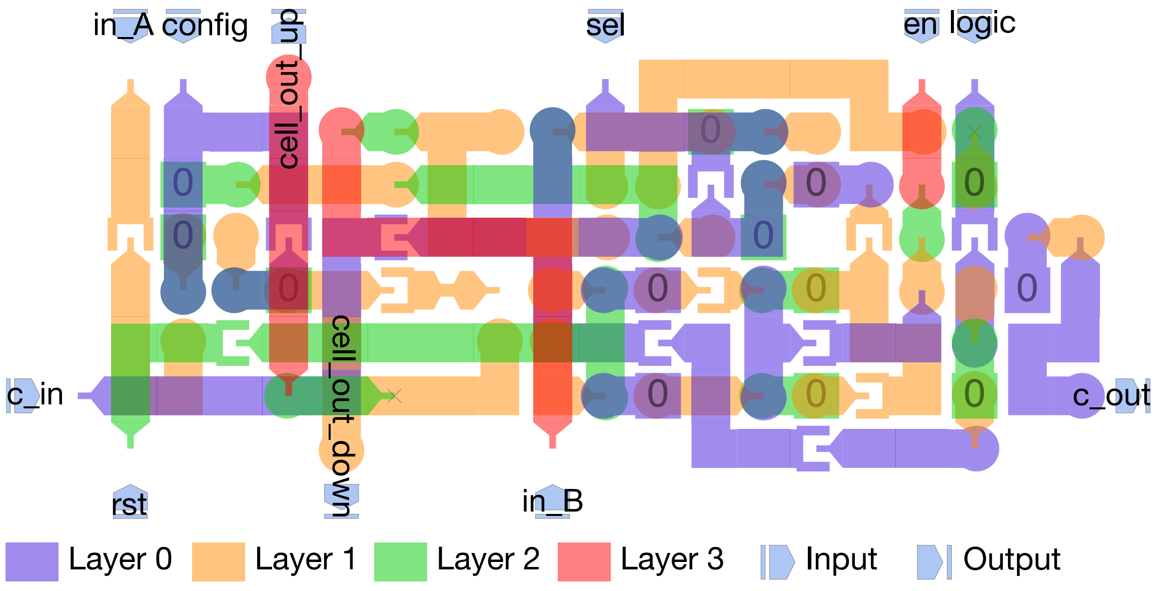

Figure 23. Simple pNML cell.

7. Conclusions

The Configurable Logic-in-Memory architecture that we have presented has strong points and issues that are worth being analyzed. Regarding its advantages, CLiMA provides:

- In-memory computation: Data are processed directly inside the memory, drastically reducing the need of data movement and favoring their reusing for further computation;
- Parallelism: The array is intrinsically highly parallel and perfect for accelerating compute and data intensive applications;
- Flexibility: The configurability of the cells and the possibility of exploiting inter-cells connections to build complex in-memory functions make CLiMA adaptable to different applications.

Regarding its limitations, mainly two can be identified:

- Not all data-flows can be supported in an array-like structure because moving data from any source to any destination is not easy and would require a very complex (but flexible) network of interconnections;
- The control of data movement between cells is complex and must be managed carefully in order to avoid cells receiving/sending wrong data from/to wrong cells.

To conclude, the Logic-in-Memory paradigm seems to be a promising alternative to the von Neumann one. We have defined a novel Configurable Logic-in-Memory Architecture that relies on in-memory computation, flexibility and parallelism to tackle the memory bottleneck problem while also providing high performance.

Author Contributions: Conceptualization, G.S.; methodology, G.S., M.G. and G.T.; investigation, G.S.; data curation, G.S.; writing—original draft preparation, G.S.; writing—review and editing, G.S., G.T. and M.G.; supervision, M.G.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. 2013 International Technology Roadmap for Semiconductors (ITRS). Available online: http://www.itrs2.net/2013-itrs.html (accessed on 31 May 2013).
2. 2009 International Technology Roadmap for Semiconductors (ITRS). Available online: https://www.semiconductors.org/wp-content/uploads/2018/09/Interconnect.pdf (accessed on 31 May 2009).

3. Kim, D.H.; Athikulwongse, K.; Healy, M.B.; Hossain, M.M.; Jung, M.; Khorosh, I.; Kumar, G.; Lee, Y.; Lewis, D.L.; Lin, T.; et al. Design and Analysis of 3D-MAPS (3D Massively Parallel Processor with Stacked Memory). *IEEE Trans. Comput.* **2015**, *64*, 112–125. [CrossRef]
4. Zhu, Q.; Akin, B.; Sumbul, H.E.; Sadi, F.; Hoe, J.C.; Pileggi, L.; Franchetti, F. A 3D-Stacked Logic-in-Memory Accelerator for Application-Specific Data Intensive Computing. In Proceedings of the 2013 IEEE International 3D Systems Integration Conference (3DIC), San Francisco, CA, USA, 2–4 October 2013; pp. 1–7. [CrossRef]
5. Ahn, J.; Hong, S.; Yoo, S.; Mutlu, O.; Choi, K. A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing. In Proceedings of the 2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA), Portland, OR, USA, 13–17 June 2015; pp. 105–117. [CrossRef]
6. Zhang, D.; Jayasena, N.; Lyashevsky, A.; Greathouse, J.L.; Xu, L.; Ignatowski, M. TOP-PIM: Throughput-oriented Programmable Processing in Memory. In Proceedings of the 23rd International Symposium on High-performance Parallel and Distributed ComputingVancouver, BC, Canada, 23–27 June 2014; pp. 85–98. [CrossRef]
7. Xie, C.; Song, S.L.; Wang, J.; Zhang, W.; Fu, X. Processing-in-Memory Enabled Graphics Processors for 3D Rendering. In Proceedings of the 2017 IEEE International Symposium on High Performance Computer Architecture (HPCA), Austin, TX, USA, 4–8 February 2017; pp. 637–648. [CrossRef]
8. Tang, Y.; Wang, Y.; Li, H.; Li, X. ApproxPIM: Exploiting realistic 3D-stacked DRAM for energy-efficient processing in-memory. In Proceedings of the 2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC), Chiba, Japan, 16–19 January 2017; pp. 396–401. [CrossRef]
9. Angizi, S.; He, Z.; Fan, D. PIMA-Logic: A Novel Processing-in-Memory Architecture for Highly Flexible and Energy-Efficient Logic Computation. In Proceedings of the 2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC), San Francisco, CA, USA, 24–29 June 2018; pp. 1–6. [CrossRef]
10. 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 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA), Seoul, Korea, 18–22 June 2016; pp. 27–39. [CrossRef]
11. Han, L.; Shen, Z.; Shao, Z.; Huang, H.H.; Li, T. A novel ReRAM-based processing-in-memory architecture for graph computing. In Proceedings of the 2017 IEEE 6th Non-Volatile Memory Systems and Applications Symposium (NVMSA), Taiwan, China, 16–18 August 2017; pp. 1–6. [CrossRef]
12. Gaillardon, P.; Amarú, L.; Siemon, A.; Linn, E.; Waser, R.; Chattopadhyay, A.; De Micheli, G. The Programmable Logic-in-Memory (PLiM) computer. In Proceedings of the 2016 Design, Automation Test in Europe Conference Exhibition (DATE), Dresden, Germany, 14–18 March 2016; pp. 427–432.
13. Li, S.; Xu, C.; Zou, Q.; Zhao, J.; Lu, Y.; Xie, Y. Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories. In Proceedings of the 2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC), Austin, TX, USA, 5–9 June 2016; pp. 1–6. [CrossRef]
14. Papandroulidakis, G.; Vourkas, I.; Abusleme, A.; Sirakoulis, G.C.; Rubio, A. Crossbar-Based Memristive Logic-in-Memory Architecture. *IEEE Trans. Nanotechnol.* **2017**, *16*, 491–501. [CrossRef]
15. 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 for Bulk Bitwise Operations Using Commodity DRAM Technology. In Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture, Cambridge, MA, USA, 14–18 October 2017; pp. 273–287. [CrossRef]
16. Huangfu, W.; Li, S.; Hu, X.; Xie, Y. RADAR: A 3D-ReRAM based DNA Alignment Accelerator Architecture. In Proceedings of the 2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC), San Francisco, CA, USA, 24–28 June 2018; pp. 1–6. [CrossRef]
17. Kaplan, R.; Yavits, L.; Ginosar, R.; Weiser, U. A Resistive CAM Processing-in-Storage Architecture for DNA Sequence Alignment. *IEEE Micro* **2017**, *37*, 20–28. [CrossRef]
18. Yavits, L.; Kvatinsky, S.; Morad, A.; Ginosar, R. Resistive Associative Processor. *IEEE Comput. Archit. Lett.* **2015**, *14*, 148–151. [CrossRef]
19. Imani, M.; Rosing, T. CAP: Configurable resistive associative processor for near-data computing. In Proceedings of the 2017 18th International Symposium on Quality Electronic Design (ISQED), Santa Clara, CA, USA, 14–15 March 2017; pp. 346–352. [CrossRef]

20. Imani, M.; Gupta, S.; Arredondo, A.; Rosing, T. Efficient query processing in crossbar memory. In Proceedings of the 2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED), Taiwan, China, 24–26 July 2017; pp. 1–6. [CrossRef]
21. Matsunaga, S.; Hayakawa, J.; Ikeda, S.; Miura, K.; Hasegawa, H.; Endoh, T.; Ohno, H.; Hanyu, T. Fabrication of a Nonvolatile Full Adder Based on Logic-in-Memory Architecture Using Magnetic Tunnel Junctions. *Appl. Phys. Express* **2008**, *1*, 091301. [CrossRef]
22. Jarollahi, H.; Onizawa, N.; Gripon, V.; Sakimura, N.; Sugibayashi, T.; Endoh, T.; Ohno, H.; Hanyu, T.; Gross, W.J. A Nonvolatile Associative Memory-Based Context-Driven Search Engine Using 90 nm CMOS/MTJ-Hybrid Logic-in-Memory Architecture. *IEEE J. Emerg. Sel. Top. Circuits Syst.* **2014**, *4*, 460–474. [CrossRef]
23. Yang, K.; Karam, R.; Bhunia, S. Interleaved logic-in-memory architecture for energy-efficient fine-grained data processing. In Proceedings of the 2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS), Boston, MA, USA, 6–9 August 2017; pp. 409–412. [CrossRef]
24. Cofano, M.; Vacca, M.; Santoro, G.; Causapruno, G.; Turvani, G.; Graziano, M. Exploiting the Logic-In-Memory paradigm for speeding-up data-intensive algorithms. *Integration* **2019**. [CrossRef]
25. LeCun, Y.; Bengio, Y. *The Handbook of Brain Theory and Neural Networks*; Chapter Convolutional Networks for Images, Speech, and Time Series; MIT Press: Cambridge, MA, USA, 1998; pp. 255–258.
26. LeCun, Y.; Bengio, Y.; Hinton, G. Deep learning. *Nature* **2015**, *521*, 436–444. [CrossRef] [PubMed]
27. LeCun, Y.; Kavukcuoglu, K.; Farabet, C. Convolutional networks and applications in vision. In Proceedings of the 2010 IEEE International Symposium on Circuits and Systems, Paris, France, 30 May–2 June 2010; pp. 253–256.
28. Gudovskiy, D.A.; Rigazio, L. ShiftCNN: Generalized Low-Precision Architecture for Inference of Convolutional Neural Networks. *arXiv* **2017**, arXiv:1706.02393.
29. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. ImageNet Classification with Deep Convolutional Neural Networks. In Proceedings of the 25th International Conference on Neural Information Processing Systems—Volume 1; Curran Associates Inc.: Lake Tahoe, NV, USA, 2012; pp. 1097–1105.
30. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep Residual Learning for Image Recognition. *arXiv* **2015**, arXiv:1512.03385.
31. Santoro, G.; Casu, M.R.; Peluso, V.; Calimera, A.; Alioto, M. Energy-performance design exploration of a low-power microprogrammed deep-learning accelerator. In Proceedings of the 2018 Design, Automation Test in Europe Conference Exhibition (DATE), Dresden, Germany, 19–23 March 2018; pp. 1151–1154. [CrossRef]
32. Santoro, G.; Casu, M.R.; Peluso, V.; Calimera, A.; Alioto, M. Design-Space Exploration of Pareto-Optimal Architectures for Deep Learning with DVFS. In Proceedings of the 2018 IEEE International Symposium on Circuits and Systems (ISCAS), Florence, Italy, 27–30 May 2018; pp. 1–5. [CrossRef]
33. Becherer, M.; Csaba, G.; Porod, W.; Emling, R.; Lugli, P.; Schmitt-Landsiedel, D. Magnetic Ordering of Focused-Ion-Beam Structured Cobalt-Platinum Dots for Field-Coupled Computing. *IEEE Trans. Nanotechnol.* **2008**, *7*, 316–320. [CrossRef]
34. Nikonov, D.E.; Young, I.A. Benchmarking of Beyond-CMOS Exploratory Devices for Logic Integrated Circuits. *IEEE J. Explor. Solid-State Comput. Devices Circuits* **2015**, *1*, 3–11. [CrossRef]
35. Cairo, F.; Turvani, G.; Riente, F.; Vacca, M.; Gamm, S.B.V.; Becherer, M.; Graziano, M.; Zamboni, M. Out-of-plane NML modeling and architectural exploration. In Proceedings of the 2015 IEEE 15th International Conference on Nanotechnology (IEEE-NANO), Rome, Italy, 27–30 July 2015; pp. 1037–1040. [CrossRef]
36. Causapruno, G.; Riente, F.; Turvani, G.; Vacca, M.; Roch, M.R.; Zamboni, M.; Graziano, M. Reconfigurable Systolic Array: From Architecture to Physical Design for NML. *IEEE Trans. Very Large Scale Integr. (VLSI) Systems* **2016**. [CrossRef]
37. Chiolerio, A.; Allia, P.; Graziano, M. Magnetic dipolar coupling and collective effects for binary information codification in cost-effective logic devices. *J. Magn. Magn. Mater.* **2012**, *324*, 3006–3012. [CrossRef]
38. Breitkreutz, S.; Kiermaier, J.; Ju, X.; Csaba, G.; Schmitt-Landsiedel, D.; Becherer, M. Nanomagnetic Logic: Demonstration of directed signal flow for field-coupled computing devices. In Proceedings of the European Solid-State Device Research Conference (ESSDERC), Helsinki, Finland, 12–16 September 2011; pp. 323–326. [CrossRef]

39. Kimling, J.; Gerhardt, T.; Kobs, A.; Vogel, A.; Wintz, S.; Im, M.Y.; Fischer, P.; Peter Oepen, H.; Merkt, U.; Meier, G. Tuning of the nucleation field in nanowires with perpendicular magnetic anisotropy. *J. Appl. Phys.* **2013**, *113*, 163902. [CrossRef]
40. Becherer, M.; Kiermaier, J.; Breitkreutz, S.; Eichwald, I.; Žiemys, G.; Csaba, G.; Schmitt-Landsiedel, D. Towards on-chip clocking of perpendicular Nanomagnetic Logic. *Solid-State Electron.* **2014**, *102*, 46–51. [CrossRef]
41. Goertz, J.J.W.; Ziemys, G.; Eichwald, I.; Becherer, M.; Swagten, H.J.M.; Breitkreutz-v. Gamm, S. Domain wall depinning from notches using combined in- and out-of-plane magnetic fields. *AIP Adv.* **2016**, *6*, 056407. [CrossRef]
42. Ferrara, A.; Garlando, U.; Gnoli, L.; Santoro, G.; Zamboni, M. 3D design of a pNML random access memory. In Proceedings of the 2017 13th Conference on Ph.D. Research in Microelectronics and Electronics (PRIME), Giardini Naxos, Italy, 12–15 June 2017; pp. 5–8. [CrossRef]
43. Riente, F.; Ziemys, G.; Mattersdorfer, C.; Boche, S.; Turvani, G.; Raberg, W.; Luber, S.; Breitkreutz-v Gamm, S. Controlled data storage for non-volatile memory cells embedded in nano magnetic logic. *AIP Adv.* **2017**, *7*, 055910. [CrossRef]
44. Becherer, M.; Gamm, S.B.V.; Eichwald, I.; Žiemys, G.; Kiermaier, J.; Csaba, G.; Schmitt-Landsiedel, D. A monolithic 3D integrated nanomagnetic co-processing unit. *Solid-State Electron.* **2016**, *115*, 74–80. [CrossRef]
45. Eichwald, I.; Kiermaier, J.; Breitkreutz, S.; Wu, J.; Csaba, G.; Schmitt-Landsiedel, D.; Becherer, M. Towards a Signal Crossing in Double-Layer Nanomagnetic Logic. *IEEE Trans. Magn.* **2013**, *49*, 4468–4471. [CrossRef]
46. Eichwald, I.; Breitkreutz, S.; Kiermaier, J.; Csaba, G.; Schmitt-Landsiedel, D.; Becherer, M. Signal crossing in perpendicular nanomagnetic logic. *J. Appl. Phys.* **2014**, *115*, 17E510. [CrossRef]
47. Eichwald, I.; Breitkreutz, S.; Ziemys, G.; Csaba, G.; Porod, W.; Becherer, M. Majority logic gate for 3D magnetic computing. *Nanotechnology* **2014**, *25*, 335202. [CrossRef] [PubMed]
48. Cofano, M.; Santoro, G.; Vacca, M.; Pala, D.; Causapruno, G.; Cairo, F.; Riente, F.; Turvani, G.; Roch, M.R.; Graziano, M.; et al. Logic-in-Memory: A Nano Magnet Logic Implementation. In Proceedings of the 2015 IEEE Computer Society Annual Symposium on VLSI, Montpellier, France, 8–10 July 2015; pp. 286–291. [CrossRef]
49. Riente, F.; Ziemys, G.; Turvani, G.; Schmitt-Landsiedel, D.; Gamm, S.B.; Graziano, M. Towards Logic-In-Memory circuits using 3D-integrated Nanomagnetic logic. In Proceedings of the 2016 IEEE International Conference on Rebooting Computing (ICRC), San Diego, CA, USA, 17–19 October 2016; pp. 1–8. [CrossRef]
50. Garlando, U.; Riente, F.; Turvani, G.; Ferrara, A.; Santoro, G.; Vacca, M.; Graziano, M. Architectural exploration of perpendicular Nano Magnetic Logic based circuits. *Integration* **2018**, *63*, 275–282. [CrossRef]
51. Santoro, G.; Vacca, M.; Bollo, M.; Riente, F.; Graziano, M.; Zamboni, M. Exploration of multilayer field-coupled nanomagnetic circuits. *Microelectron. J.* **2018**, *79*, 46–56. [CrossRef]
52. Vacca, M.; Graziano, M.; Wang, J.; Cairo, F.; Causapruno, G.; Urgese, G.; Biroli, A.; Zamboni, M. *NanoMagnet Logic: An Architectural Level Overview*; LNCS, Lecture Notes in Computer Science (including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics); Springer: Berlin/Heidelberg, Germany, 2014; Volume 8280, pp. 223–256.
53. Riente, F.; Garlando, U.; Turvani, G.; Vacca, M.; Roch, M.R.; Graziano, M. MagCAD: A Tool for the Design of 3D Magnetic Circuits. *IEEE J. Explor. Solid-State Comput. Devices Circuits* **2017**, *3*, 65–73. [CrossRef]
54. Turvani, G.; Riente, F.; Graziano, M.; Zamboni, M. A quantitative approach to testing in Quantum dot Cellular Automata: NanoMagnet Logic case. In Proceedings of the 2014 10th Conference on Ph.D. Research in Microelectronics and Electronics (PRIME), Grenoble, France, 30 June–3 July 2014; pp. 1–4. [CrossRef]
55. Turvani, G.; Tohti, A.; Bollo, M.; Riente, F.; Vacca, M.; Graziano, M.; Zamboni, M. Physical design and testing of Nano Magnetic architectures. In Proceedings of the 2014 9th IEEE International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS), Santorini, Greece, 6–8 May 2014; pp. 1–6. [CrossRef]
56. Turvani, G.; Riente, F.; Cairo, F.; Vacca, M.; Garlando, U.; Zamboni, M.; Graziano, M. Efficient and reliable fault analysis methodology for nanomagnetic circuits. *Int. J. Circuit Theory Appl.* **2016**, *45*, 660–680. [CrossRef]