CIM-SIM: Computation In Memory SIMulator

Ali BanaGozar, Kanishkan Vadivel, Sander Stuijk, Henk Corporaal
Eindhoven University of Technology
Eindhoven, The Netherlands
{a.banagozar,k.vadivel,s.stuijk,h.corporaal}@tue.nl

Stephan Wong, Muath Abu Lebdeh, Jintao Yu, Said Hamdioui
Delft University of Technology
Delft, The Netherlands
{j.s.s.m.wong,m.f.m.abulebdeh,j.yu-1,s.hamdioui}@tudelft.nl

ABSTRACT

Computation-in-memory reverses the trend in von-Neumann processors by bringing the computation closer to the data, to even within the memory array, as opposed to introducing new memory hierarchies to keep (frequently used) data closer to a central processing unit (CPU). In recent years, new non-volatile memory (NVM) technologies, e.g., memristor, PCM, etc., have proven that they can function as memories and perform computations on the stored data as well. In particular, when they are combined with a modest set of (digital) peripheral modules, a wider range of operations can be supported, e.g., vector matrix multiply and Boolean logic. In this paper, we are introducing the CIM-SIM, an open source simulator written in SystemC, which is capable of simulating the functional behaviour of such architectures. The architecture includes the definition of a set of technology-agnostic nano-instructions.

CCS CONCEPTS

• **Hardware → Application specific instruction set processors**; **Emerging architectures**; *Memory and dense storage*; • **Computing methodologies → Simulation tools**.

KEYWORDS

Simulator, Non-Von Neumann Architecture, Non-Volatile Memory, Computation In Memory, Memristor

ACM Reference Format:
Ali BanaGozar, Kanishkan Vadivel, Sander Stuijk, Henk Corporaal and Stephan Wong, Muath Abu Lebdeh, Jintao Yu, Said Hamdioui. 2019. CIM-SIM: Computation In Memory SIMulator. In *22nd International Workshop on Software and Compilers for Embedded Systems (SCOPES '19), May 27–28, 2019, Sankt Goar, Germany*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3323439.3323989

1 INTRODUCTION

The nearing end of Moore's law urges researchers to devise brand new processors. It could be achieved making substantial changes at different levels ranging from devices to architectures [14, 15]. One promising new paradigm encompassed the use of NVMs [4, 8–10] to perform both compute and storage of data at the same site. The current state-of-art mainly focuses on the design of memory cells/arrays and its directly related peripheral circuitry while the technology is still being developed. This approach is also categorized as CIM-P (Compute-In-Memory with Periphery) [1]. High-level applications kernels are envisioned to execute on such CIM-P tiles without a clear method to control and schedule the operations within the memory array and periphery. In this paper, we propose our first version of an instruction set architecture (ISA) to fill this gap by achieving this goal. In addition, we introduce a simulator, called CIM-SIM, that is capable of executing nano-instructions of the ISA. The proposed nano-instructions are technology-agnostic and modular by design, i.e., capable of supporting different sets of peripheral modules. CIM-SIM offers researchers an opportunity to investigate not only different architecture designs for NVM-based computation platforms, but corresponding high level tools (e.g. compilers) as well. Additionally, its modular design makes it easy for circuit and device level researches to modify the connections, and to add or remove components based on their own designs. Being an Instruction Set Archiecture (ISA), CIM-SIM fetches a sequence of nano-instructions which are derived from breaking down a kernel. In Section 4, all the nano-instructions are defined, and, using an example, it is explained how one can break a kernel into a sequence of nano-instructions.

The remainder of this paper is organized as follows: in Section 2 related work on NVM computation platforms is introduced. From Section 3 till Section 5 it is illustrated how the CIM-SIM is developed, what the instructions are and how one can initialize the simulator. In Section 6 a couple of potential kernels and applications that can be mapped to such an architecture are presented. Lastly, the paper concludes in Section 7.

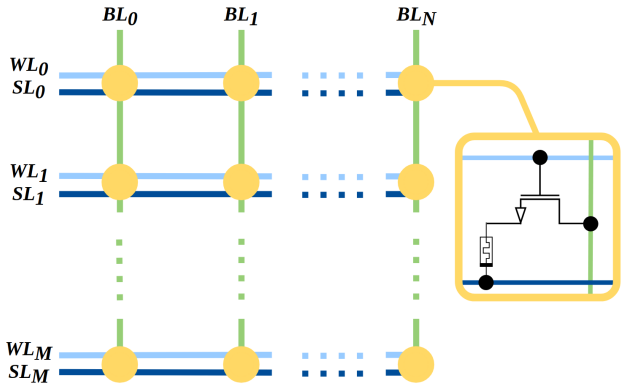

Figure 1: Non-Volatile Memory Crossbar

SCOPES '19, May 27–28, 2019, Sankt Goar, Germany

ACM ISBN 978-1-4503-6762-2/19/05…$15.00
https://doi.org/10.1145/3323439.3323989

2 NON-VOLATILE MEMORY BASED COMPUTATION PLATFORMS

After the first memristor was built in 2008, quite a few research groups have tried to exploit it for different purposes in various manners. The most attractive, and the most popular approach, nevertheless, is to organize them in a crossbar, with a memristor (in series with an access transistor) at every cross-point (Fig. 1.) Being surrounded by some analog circuitries, a crossbar could be enabled to serve both as memory unit, and as processing element at the same physical position, which makes them favourable for non-Von Neumann architectures [4, 8–10]. Although the platform itself has been investigated quite extensively, the ways that it could be adopted by existing architectures (e.g. as an accelerator) or how it could be exploited as a stand-alone processor is not comprehensively explored. In [6, 12, 13], different approaches have been taken to utilize memristor-based platforms in an architecture that is only suitable for Neural Networks (NNs). In [6], authors have proposed to process in ReRAM-based main memory to accelerate NN computation, bringing the memory unit and processor elements physically closer to each other. Shafiee, et al. [13] have proposed a tiled architecture, again for accelerating NN, where every tile comprises an eDRAM Buffer, memristor crossbars, analog circuitries, input and output registers, and several units with special functionalities. [11] proposes an architecture which uses NVM based main memory to perform bulk bit-wise operations. Finally, [17] proposes a simulator, called MNSIM, for memristor-based neuromorphic systems. All the proposed architectures, nonetheless, just make use of some of memristor potentials, and none proposes a comprehensive architecture which could exploit them to their full potential. CIM-SIM, on the other hand, not only supports different kernels, but it can easily be modified to meet the requirements of potential kernels that may be mapped to an NVM-crossbar in future, as well. As an example, for some kernels the WLs in the crossbar must be vertical. The new arrangement can be supported by the simulator, modifying connections between blocks.

3 OUR NANO-ARCHITECTURE

Our simulator, which is written in SystemC, mimics the functional behaviour of a possible NVM-based platform, which could process different kernels such as Vector Matrix Multiply (VMM), and bitwise Boolean logic. A general organization of a CIM-P tile is depicted in Fig. 2. The calculator represents the analog circuitry of the NVM crossbar (X-bar) (Fig. 1.) In addition, the calculator contains DIMs (Digital Input Modulator), and A/Ds (Analog/Digital converter) to interface with the digital (nano-)controller and associated registers. Moreover, the sample-and-hold (S&H) circuitry allows for the separation of calculations within the X-bar and the readout circuitry, i.e., the A/D blocks. In the digital periphery (outside the calculator), several registers are being used to control the calculator. They are:

- WD (Write Data) register is used to temporary store the data that is to be loaded into the crossbar.
- WDS (Write Data Select) register serves as a mask to protect devices which are not intended to be modified from being overwritten. This accelerates writing process, decreases energy consumption, and alleviates low endurance of NVMs, avoiding unnecessary writes.

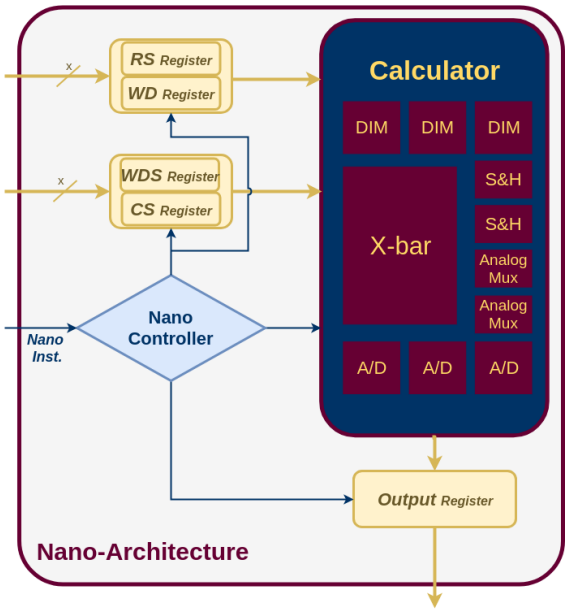

Figure 2: Nano-architecture Block Diagram

- RS (Row Select) register is used to select which row or rows should be activated. E.g., only one row needs to be activated when writing data, but several rows needs to be activated for other operations. Additionally, it holds the input values to be processed on the crossbar.
- CS (Column Select) register is used to select which columns from the crossbar should be read and converted to digital data. In this manner, non-relevant columns can be skipped or interleaving of data stored in the crossbar can be supported. Moreover, when multiple columns need to share a single A/D-block, the CS register can be used to control this.
- Output register is being used to temporarily store the digital data before it is sent to external devices, e.g., a host CPU, other memories, or other CIM-P tiles.

The nano-controller can send specific control signals to the calculator to drive specific actions within the calculator. For example, the sense amplifiers (within the A/D blocks) can be controlled perform different Boolean operations. These control signals are collectively referred to as FS (Function Select) signals.

4 (NANO-)INSTRUCTION SET

Before defining the instructions for our nano-architecture, we observe that the control of the CIM-P tile can be divided into several distinct phases. We envision that in the future these phases can

Table 1: Parameters

Symbol	Definition
x	Data Bus Width (Byte)
S_{reg}	Size of RS or WD register (Byte)
S_{WDS}	Size of WDS register (Byte)

Algorithm 1: Load Class

Input: $Start_{address}$
Output: Null
*__reg is__ RS_{reg} **or** WD_{reg};
for *(*$i = 0;\ i < S_{reg};\ i = i + x$*)* **do**
    $reg[i] \leftarrow \$(Start_{address} + i)$;

be overlapped to allow for pipelining purposes. The phases (with corresponding instructions) are:

- **Load:** Instructions in this phase fetch data from a higher level memory and includes the following two instructions: *WD (Write Data)*, and *RS (Row Select)*. Instructions get the starting address of the data in higher level memory as input and fill the respective register accordingly (Algorithm 1.)
- **Configuration:** Instructions in this phase are used to configure specific parts of nano-architecture which can operate in different fashions depending on the operation. The *WDS* instruction sets the pattern of the columns to be selected in the *WDS register*. In our implementation, based on our use case, we assume the pattern is $value_1$ number of consecutive columns starting from $value_0$. In a similar fashion, *CS* determines the column to be selected among the columns which share one A/D controlling select line of the analog multiplexers ($value_0$). The operation to be performed is decided by the *FS* instruction *Tag* ($value_0$) (Algorithm 2.)
- **Compute:** In this phase, we specify a single instruction: *DoA (Do Array)*. DoA triggers the DIMs to steer the data for the operation specified by *FS*. Data is written, or processed by the array and if any result is produced, it will be held in S&H.
- **Read:** In this phase, we specify a single instruction: *DoR (Do Read)*. Columns selected by *CS* are read, and converted to corresponding digital values by the A/Ds.

Finally, it is expected that the digital peripheral circuits can be clocked faster than a single operation on the (analog) crossbar. This means that several clock ticks can occur while an operation is performed in the calculator. This is another motivation for the proposed phases and instructions, as it allows us to schedule them in a flexible manner, i.e., configure the periphery for the next operation before actually issuing the DoA instruction.

Algorithm 2: Configuration Class

Input: $value_0, value_1, [..., value_n]$
Output: Null
if *(Instruction = WDS)* **then**
    for *(*$i = 0;\ i < S_{WDS};\ i = i + 1$*)* **do**
        if $(value_0 =< i < value_0 + value_1)$ **then**
            $WDS_{reg}[i] \leftarrow 1$;
        else
            $WDS_{reg}[i] \leftarrow 0$;
else if *(Instruction = CS)* **then**
    $CS_{reg} \leftarrow value_0$;
else if *(Instruction = FS)* **then**
    $FS_{signal} \leftarrow value_0$;

Algorithm 3: Programming Crossbar

```
RS    $address1;
WD    $address2;
WDS   value0, value1;
FS    WR;
DoA;
```

5 CROSSBAR INITIALIZATION

To utilize a NVM crossbar, first of all, it should be programmed. Basically, to program a NVM to a certain state (conductance), a voltage higher than the threshold voltage should be applied across the device. There are serious challenges to program a device whether it is in a crossbar or not, though, which are out of the scope of this paper [2, 16]. Here we will just explain the sequence of instructions that should be issued to set up the analog and digital peripheries for writing memristors in a crossbar. Since a voltage should be applied across a memristor to program it, both the WD register and the RS register should be filled with correct values. Additionally, the WDS register needs to be set in a way that certain columns which are not intended to be modified are disconnected. Then, the nano-architecture should be configured via FS instruction. Having them all set, and when the nano controller receives the *DoA* instruction, it could issue the corresponding signal and certain parts of the crossbar selected by registers are programmed (Algorithm 3.)

6 POTENTIAL KERNELS

In this section two popular kernels which we already have run on the nano-architecture are explained. We have chosen these kernels since, they are the most promising kernels that could be mapped on NVM-crossbar based platforms. These , however, are just examples to clarify how CIM-SIM could be used; one, could modify the simulator to be suitable to run other kernels.

6.1 Vector Matrix Multiply

As neural networks show promising results in various fields they are attracting quite some attention. Consequently, quite intensive efforts has been put to develop suitable platforms and algorithms for implementation of the respective kernels, e.g. VMM [3, 5, 8, 13]. Considering that, one of the kernels that we have implemented in our simulator is VMM. To do a VMM, depending on the size of the vector, the rows which are not intended to take part in calculation will not be activated at all. Thus, the respective $RS_{register}$ will hold

Algorithm 4: Vector Matrix Multiplication

```
RS    $address;
FS    VMM;
DoA;
CS    value0;
DoR;
```

Algorithm 5: Bulk Bit-wise Boolean Logic (AND)

```
1 RS   $address;
2 FS   AND;
3 DoA;
4 CS   value_0;
5 DoR;
```

zero, making WL low, which deactivates the whole row. Rows being configured, FS instruction will determine the functionality, which in this case is VMM. Issuing DoA, CS, and DoR data are processed and sent out to the output register (Algorithm 4.)

6.2 Bulk Bit-wise Boolean Logic

Performing bit-wise Boolean logic is essential in various applications such as in queries [7]. Some NVM-crossbar based platforms have been proposed to implement the kernel [18]. In this case all the data to be processed is in the memristor crossbar itself. The data desired to be processed is selected, applying a read voltage to corresponding rows, according to [18], which is non-zero value for the selected rows and zeros for others. This means the data to be fetched via instruction *RS $address* only could have two non-zero values. In this example we have decided to have an AND function, so the FS tag is *AND*. (Algorithm 5.)

7 CONCLUSION

In this paper, we presented the CIM-SIM, a new functional accurate simulator for NVM-crossbar-based computing platforms. It can be used to analyze the functional behaviour of kernels like VMM or bulk bit-wise Boolean logic using a sequence of nano-instructions. Various extensions to the simulator are envisioned. First, we intend to extend the simulator to incorporate various timings in both the analog crossbar (defined by technology trends) and the digital controller. This can be used to perform a trade-off analysis between the digital and analog circuits. For example, matching the clock of the digital circuit to the delay of the analog array will allow for power/energy savings. Second, different NVM technologies will provide and different applications will need different operations within the CIM-P tile. Using our CIM-SIM, these (matching and non-matching) provisions and requirements can be investigated. Third, the defined instructions allow for an exploration in the automatic code generation, e.g., building a new compiler, for current and future NVM architectures as well as the rescheduling of the same instructions to enhance performance. The code generation and scheduling will greatly depend on many technology factors and, again, our CIM-SIM can be used for this investigation. Finally, we intend to complete our CIM-SIM with interface to other existing simulators, e.g., GEM5. This will allow for a more in-depth investigation into hardware/software co-design techniques combining traditional processors with NVM-based accelerators.

REFERENCES

[1] Muath Abu Lebdeh, Uljana Reinsalu, Hoang Anh Du Nguyen, Stephan Wong, and Said Hamdioui. 2019. Memristive Device Based Circuits for Computation-in-Memory Architectures. In *2019 IEEE International Symposium on Circuits and Systems (ISCAS), Accepted (Paper ID: 5782769).*

[2] Fabien Alibart, Ligang Gao, Brian D Hoskins, and Dmitri B Strukov. 2012. High precision tuning of state for memristive devices by adaptable variation-tolerant algorithm. *Nanotechnology* 23, 7 (2012), 075201. https://doi.org/10.1088/0957-4484/23/7/075201

[3] Md Zahangir Alom, Tarek M Taha, Christopher Yakopcic, Stefan Westberg, Mahmudul Hasan, Brian C Van Esesn, Abdul A S Awwal, and Vijayan K Asari. 2018. The history began from alexnet: A comprehensive survey on deep learning approaches. *arXiv preprint arXiv:1803.01164* (2018).

[4] Irem Boybat, Manuel Le Gallo, SR Nandakumar, Timoleon Moraitis, Thomas Parnell, Tomas Tuma, Bipin Rajendran, Yusuf Leblebici, Abu Sebastian, and Evangelos Eleftheriou. 2018. Neuromorphic computing with multi-memristive synapses. *Nature communications* 9, 1 (2018), 2514. https://doi.org/10.1038/s41467-018-04933-y

[5] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, and Olivier Temam. 2014. DaDianNao: A Machine-Learning Supercomputer. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-47)*. IEEE Computer Society, Washington, DC, USA, 609–622. https://doi.org/10.1109/MICRO.2014.58

[6] Ping Chi, Shuangchen Li, Cong Xu, Tao Zhang, Jishen Zhao, Yongpan Liu, Yu Wang, and Yuan Xie. 2016. PRIME: A Novel Processing-in-memory Architecture for Neural Network Computation in ReRAM-based Main Memory. In *Proceedings of the 43rd International Symposium on Computer Architecture (ISCA '16)*. IEEE Press, Piscataway, NJ, USA, 27–39. https://doi.org/10.1109/ISCA.2016.13

[7] Jerry Chou, Mark Howison, Brian Austin, Kesheng Wu, Ji Qiang, E. Wes Bethel, Arie Shoshani, Oliver Rübel, Prabhat, and Rob D. Ryne. 2011. Parallel Index and Query for Large Scale Data Analysis. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC '11)*. ACM, New York, NY, USA, Article 30, 11 pages. https://doi.org/10.1145/2063384.2063424

[8] Miao Hu, John Paul Strachan, Zhiyong Li, R Stanley, et al. 2016. Dot-product engine as computing memory to accelerate machine learning algorithms. In *2016 17th International Symposium on Quality Electronic Design (ISQED)*. IEEE, 374–379. https://doi.org/10.1109/ISQED.2016.7479230

[9] Manuel Le Gallo, Abu Sebastian, Roland Mathis, Matteo Manica, Heiner Giefers, Tomas Tuma, Costas Bekas, Alessandro Curioni, and Evangelos Eleftheriou. 2018. Mixed-precision in-memory computing. *Nature Electronics* 1, 4 (2018), 246. https://doi.org/10.1038/s41928-018-0054-8

[10] Can Li, Miao Hu, Yunning Li, Hao Jiang, Ning Ge, Eric Montgomery, Jiaming Zhang, Wenhao Song, Noraica Dávila, Catherine E Graves, et al. 2018. Analogue signal and image processing with large memristor crossbars. *Nature Electronics* 1, 1 (2018), 52. https://doi.org/10.1038/s41928-017-0002-z

[11] Shuangchen Li, Cong Xu, Qiaosha Zou, Jishen Zhao, Yu Lu, and Yuan Xie. 2016. Pinatubo: A Processing-in-memory Architecture for Bulk Bitwise Operations in Emerging Non-volatile Memories. In *Proceedings of the 53rd Annual Design Automation Conference (DAC '16)*. ACM, New York, NY, USA, Article 173, 6 pages. https://doi.org/10.1145/2897937.2898064

[12] Shaoli Liu, Zidong Du, Jinhua Tao, Dong Han, Tao Luo, Yuan Xie, Yunji Chen, and Tianshi Chen. 2016. Cambricon: An Instruction Set Architecture for Neural Networks. In *Proceedings of the 43rd International Symposium on Computer Architecture (ISCA '16)*. IEEE Press, Piscataway, NJ, USA, 393–405. https://doi.org/10.1109/ISCA.2016.42

[13] Ali Shafiee, Anirban Nag, Naveen Muralimanohar, Rajeev Balasubramonian, John Paul Strachan, Miao Hu, R. Stanley Williams, and Vivek Srikumar. 2016. ISAAC: A Convolutional Neural Network Accelerator with In-situ Analog Arithmetic in Crossbars. In *Proceedings of the 43rd International Symposium on Computer Architecture (ISCA '16)*. IEEE Press, Piscataway, NJ, USA, 14–26. https://doi.org/10.1109/ISCA.2016.12

[14] John M Shalf and Robert Leland. 2015. Computing beyond Moore's Law. *Computer* 48, 12 (2015), 14–23. https://doi.org/10.1109/MC.2015.374

[15] R Stanley Williams. 2017. What's Next? [The end of Moore's law]. *Computing in Science Engineering* 19, 2 (2017), 7–13. https://doi.org/10.1109/MCSE.2017.31

[16] H-S Philip Wong, Heng-Yuan Lee, Shimeng Yu, Yu-Sheng Chen, Yi Wu, Pang-Shiu Chen, Byoungil Lee, Frederick T Chen, and Ming-Jinn Tsai. 2012. Metal–oxide RRAM. *Proc. IEEE* 100, 6 (2012), 1951–1970.

[17] Lixue Xia, Boxun Li, Tianqi Tang, Peng Gu, Pai-Yu Chen, Shimeng Yu, Yu Cao, Yu Wang, Yuan Xie, and Huazhong Yang. 2018. MNSIM: Simulation Platform for Memristor-Based Neuromorphic Computing System. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37, 5, 1009–1022. https://doi.org/10.1109/TCAD.2017.2729466

[18] Lei Xie, Hoang Anh Du Nguyen, Jintao Yu, Ali Kaichouhi, Mottaqiallah Taouil, Mohammad AlFailakawi, and Said Hamdioui. 2017. Scouting logic: A novel memristor-based logic design for resistive computing. In *2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 176–181. https://doi.org/10.1109/ISVLSI.2017.39