CIM-SIM

Citation for published version (APA):

DOI:
10.1145/3323439.3323989

Document status and date:
Published: 27/05/2019

Document Version:
Accepted manuscript including changes made at the peer-review stage

Please check the document version of this publication:

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher’s website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.

Download date: 08. Jan. 2020
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:

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.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

SCOPES ’19, May 27–28, 2019, Sankt Goar, Germany
© 2019 Association for Computing Machinery.
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.

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>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>Data Bus Width (Byte)</td>
</tr>
<tr>
<td>S_{reg}</td>
<td>Size of RS or WD register (Byte)</td>
</tr>
<tr>
<td>S_{WDS}</td>
<td>Size of WDS register (Byte)</td>
</tr>
</tbody>
</table>
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 \( \text{value}_1 \) number of consecutive columns starting from \( \text{value}_0 \). 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 (\( \text{value}_0 \)). The operation to be performed is decided by the *FS* instruction (Tag (\( \text{value}_0 \))) (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.

### 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 peripherals 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 have 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 1: Load Class**

<table>
<thead>
<tr>
<th>Input: ( \text{Start}_{\text{address}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output: Null</td>
</tr>
<tr>
<td>*reg is RS( _\text{Reg} ) or WD( _\text{Reg} );</td>
</tr>
<tr>
<td>\textbf{1 for} (( i = 0; i &lt; S_{\text{Reg}}; i = i + x )) \textbf{do}</td>
</tr>
<tr>
<td>\textbf{2 } *\text{reg}[i] \leftarrow $ (\text{Start}_{\text{address}} + i) );</td>
</tr>
</tbody>
</table>

**Algorithm 2: Configuration Class**

<table>
<thead>
<tr>
<th>Input: ( \text{value}_0, \text{value}_1, [...], \text{value}_n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output: Null</td>
</tr>
<tr>
<td>\textbf{1 if} (\text{Instruction} = \text{WDS}) \textbf{then}</td>
</tr>
<tr>
<td>\textbf{2 for} (( i = 0; i &lt; S_{\text{WDS}}; i = i + 1 )) \textbf{do}</td>
</tr>
<tr>
<td>\textbf{3 if} (\text{value}_0 = &lt; i &lt; \text{value}_0 + \text{value}_1) \textbf{then}</td>
</tr>
<tr>
<td>\textbf{4 } \text{WDS}_{\text{Reg}}[i] \leftarrow 1 );</td>
</tr>
<tr>
<td>\textbf{5 else}</td>
</tr>
<tr>
<td>\textbf{6 } \text{WDS}_{\text{Reg}}[i] \leftarrow 0 );</td>
</tr>
<tr>
<td>\textbf{7 else if} (\text{Instruction} = \text{CS}) \textbf{then}</td>
</tr>
<tr>
<td>\textbf{8 } \text{CS}_{\text{Reg}} \leftarrow \text{value}_0;</td>
</tr>
<tr>
<td>\textbf{9 else if} (\text{Instruction} = \text{FS}) \textbf{then}</td>
</tr>
<tr>
<td>\textbf{10 } \text{FS}_{\text{signal}} \leftarrow \text{value}_0;</td>
</tr>
</tbody>
</table>

**Algorithm 3: Programming Crossbar**

| 1 RS $\text{address}_1$; |
| 2 WD $\text{address}_2$; |
| 3 WDS \text{value}_0, \text{value}_1; |
| 4 FS \text{WR}; |
| 5 DoA; |

**Algorithm 4: Vector Matrix Multiplication**

| 1 RS $\text{address}_1$; |
| 2 FS \text{VMM}; |
| 3 DoA; |
| 4 CS \text{value}_0; |
| 5 DoR; |
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 $RS\ saddress$ 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


