RdaCIM: A Read-Disturb-Aware Computation-In-Memory Simulator

Zhengtao Huang

RdaCIM: A Read-Disturb-Aware Computation-In-Memory Simulator

THESIS

submitted in partial fulfilment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER AND EMBEDDED SYSTEMS ENGINEERING

by

Zhengtao Huang

Computer Engineering Group
Department of Quantum and Computer Engineering
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands

RdaCIM: A Read-Disturb-Aware Computation-In-Memory Simulator

Author: Zhengtao Huang
Student id: 5833469

Abstract

Memristor-based Computation-In-Memory (CIM) architectures are a genre of emerging computing designs, and they have the potential to provide a power-efficient computational power for artificial intelligence (AI). However, current memristor-based CIM designs face the challenges from non-idealities, such as read disturb. The mitigation of non-idealities in CIM architectures is an area of active research.

Simulation tools are important design tools for CIM. Conventionally, SPICE simulations are used for CIM architectures. Modern high-level simulation frameworks for CIM are faster when compared to SPICE, and therefore it is desirable to investigate the feasibility of non-ideality analysis, such as read disturb analysis, in high-level simulations. However, current high-level simulators do not include a model for read disturb, and they are not suitable for read disturb analysis.

To fill this gap, this thesis presents RdaCIM, a read-disturb-aware CIM simulator. RdaCIM is high-level simulation tool that exploits parallelism provided by multi-core CPUs and AVX instructions. More importantly, RdaCIM involves the non-trivial non-ideality of read disturb into the simulations, which makes it possible for the user to perform read disturb related investigations on the simulator.

Experiments have been done with RdaCIM to show the feasibility of read disturb analysis on this tool. The effectiveness of a rewriting scheme as a countermeasure to read disturb is verified on RdaCIM. Furthermore, an effort to reduce the overhead of rewriting by dynamic voltage adjustment is presented and verified with RdaCIM. Performance benchmarks have been done to elaborate the benefits of the parallelised implementation of the simulation tool.

Thesis Committee:

Prof. Dr. Georgi Gaydadjiev, Faculty EEMCS, TU Delft
Dr. Anteneh Gebregiorgis, Faculty EEMCS, TU Delft
Dr. Sebastian Feld, Faculty EEMCS, TU Delft
Dr. Theofilos Spyrou, Faculty EEMCS, TU Delft

Preface

This thesis project marks the final stage of my master's study at TU Delft, which has been an invaluable experience for my personal growth. Before I started the thesis project, I barely knew anything about semiconductor devices, and my knowledge on analogue circuit design was only at a basic level, but I decided to take the challenge and to step out of my comfort zone. The challenges I encountered at TU Delft have provided an opportunity for me to improve my learning techniques and problem solving skills. More importantly, I got a chance to build up the mental strength in face of difficulties. These are important experiences that will help me to approach my goals in life.

I would like to thank Dr. Anteneh Gebregiorgis and Prof. Georgi Gaydadjiev for their care and guidance during my thesis project. I would like to thank everyone I know from the department of Quantum and Computer Engineering at TU Delft. I would also like to thank my friends who supported me and kept me company. I am deeply grateful for my parents who have been supporting my education and my dream.

Zhengtao Huang
The Hague, the Netherlands
October 20, 2025

Contents

List of Figures

List of Tables

Acronyms

ADC Analogue-Digital Converter. 8, 20–22, 30

AI Artificial Intelligence. 1, 10, 38

API Application Programming Interface. 15, 16

AVX Advanced Vector Extensions. 22, 33–35, 38

CIM Computation-in-Memory. 1–3, 6–8, 10, 15, 16, 22, 24, 37–39

CMOS Complementary Metal-Oxide-Semiconductor. 1

CPU Central Processing Unit. 22, 34, 38

DAC Digital-Analogue Converter. 7, 20–22, 27, 29, 30, 32

GPU Graphics Processing Unit. 15, 17, 38

HRS High Resistance State. 5, 6, 10, 11, 16, 20, 23, 30, 38, 39

IO Input/Output. 8

KCL Kirchhoff's Current Law. 8

LRS Low Resistance State. 5, 10, 11, 16, 20, 23, 30, 38, 39

LUT Look-up Table. 16

MLC Multi-Level Cell. 6

PCM Phase-Change Memory. 17

RRAM Resistive Random-Access Memory. 1, 5, 6, 9, 17, 30, 37

SIMD Single-Instruction-Multiple-Data. 2, 22, 34, 38

SLC Single-Level Cell. 6

SPICE Simulation Program with Integrated Circuit Emphasis. 1, 2, 15, 16, 19, 37

Chapter 1

Introduction

1.1 Motivation

The growing potentials of Artificial Intelligence (AI) have created an increasing demand for computational power. The Complementary Metal-Oxide-Semiconductor (CMOS) technology and the Von-Neumann architecture are the cornerstones of modern computing systems, and the advances in CMOS technology and the development of sophisticated computer architectures have been supporting the growing need for computing power by AI applications. However, Further shrinkage of CMOS technology nodes and the scaling of multi-core systems would encounter diminishing returns in terms of performance-power ratio [7][26]. The ever-growing demands for performance and power efficiency fuels the research in emerging computing paradigms [10]. Memristor-based Computation-in-Memory (CIM) designs have shown their potential in the area of edge AI applications [9]. The research into memristor-based CIM architectures is aimed to tackle the limitations of the CMOS technology and the conventional Von-Neumann architectures by adopting emerging non-volatile memory devices and a memory-centric design approach [36].

The reliability of memristor-based computing is an area of active research. CIM architectures based on emerging memristive devices, such as Resistive Random-Access Memory (RRAM), are prone to non-ideal effects that lead to reduced computational accuracy [36][8]. One of the major non-idealities is the read disturb, which introduces unwanted changes to the memory state during read operations [27].

Simulators are important tools for circuit design [19], and they are important design tools for developing reliable CIM systems. In the area of CIM architectures, simulation tools are available at multiple design levels [17][13]. Traditionally, the research into the non-idealities in CIM designs often requires simulations at a lower design level, and the Simulation Program with Integrated Circuit Emphasis (SPICE) [18] is the norm for the investigations into non-idealities such as read disturb. However, device and circuit level simulations such as SPICE are relatively slower when compared to simulations at higher design levels, which limits the scale of the circuit under simulation. Therefore, it is desirable to investigate the feasibility of non-ideality analysis on simulation tools that are not based on SPICE.

1.2 Problem Statement and Research Questions

The non-idealities in CIM architectures affect their computational accuracy. While investigations into non-idealities are conventionally conducted on SPICE-based simulations, these conventional simulation tools are slow and hard to scale up. Recently developed high-level simulation tools for CIM are parallelisable and more flexible, which allows faster simulations and a better user experience. They incorporate models of non-ideal effects in their designs to produce simulation results that are more resemblant of the reality. Although most of current high-level simulators involve non-idealities such as device variability and the non-infinite ON/OFF ratio in their implementations [13], few of them has an emphasis on read disturb in their design, which makes it difficult to perform analysis of the impacts of read disturb on these platforms.

Due to the limitations of current high-level simulation tools in terms of the modelling of non-idealities, a new simulation tool is needed to verify the feasibility of high-level simulations for the analysis of read disturb.

Considering the challenges in current simulation tools for CIM, the following research questions are proposed to navigate the investigation:

1. Is it feasible to perform analyses of read disturb on a CIM simulation tool that is not based on SPICE?

2. What are the benefits brought by a parallelisable implementation of simulation tool in terms of speed and scalability?

1.3 Contributions

Considering the limitations of conventional SPICE simulation tools in terms of speed and flexibility, and the limitations of current high-level simulation tools in terms of read disturb analysis, this work attempts to address the gap by presenting RdaCIM (Read-Disturb-Aware Computation-In-Memory Simulator). RdaCIM is a high-level simulator that incorporates modelling of read disturb without SPICE-based simulations, and the simulations on RdaCIM are parallelised with Single-Instruction-Multiple-Data (SIMD) instructions and multi-threading, which allows for simulations at larger scales within a reasonable execution time.

In order to verify the feasibility of read disturb analysis without the use of SPICE-based simulation tools, a cell rewriting scheme and a dynamic voltage adjustment scheme are deployed in simulations on RdaCIM. An analysis of the dynamic voltage adjustment scheme is performed, and the frequencies of necessary cell rewritings are recorded.

Performance benchmarking is done to demonstrate the benefits brought by parallelisation to the CIM simulation tool, and it shows the importance of a parallelisable implementation of simulation tool in face of simulations at large scales.

The contributions of this work can be summarised into the aspects listed below.

1. The implementation of the RdaCIM simulator, a parallelised high-level simulator for memristor-based computation-in-memory accelerators with considerations of read disturb.
2. An investigation into a dynamic voltage adjustment scheme that reduces the overhead of cell rewriting as a counter-measure against read disturb.
3. The verification of the feasibility of non-ideality analysis on a parallelised high-level simulation tool.
4. A performance analysis of the proposed simulation tool under different levels of parallelism.

1.4 Thesis Structure

The remaining parts of the thesis are structured into five chapters. In chapter 2, the background of this work is given, which includes an overview of CIM, memristors, non-idealities of CIM, read disturb and its mitigations, and current simulation tools for CIM. In chapter 3, an overview of several modern CIM simulation tools are given. In chapter 4, the details of the implementation of RdaCIM are explained. In chapter 5, results from experiments regarding the verification of the dynamic voltage adjustment scheme are discussed. The performance benchmarking of RdaCIM is reported. Chapter 6 concludes the thesis by answering the research questions, discussing the limitations of this work, and providing an outlook for potential work in the future.

Chapter 2

Background

2.1 Memristors and CIM

The concept of memristors was first introduced by Leon Chua in 1971 [6]. It was defined as an electrical device whose resistance is decided by the history of the current running through the device. The discussion around memristors remained theoretical until Strukov et al. made the first memristive device in 2008 [30]. Though there has been a debate on whether such semiconductor devices are so-called "real memristor devices" [34], the term "memristor" has become an umbrella term for devices with a programmable resistance.

RRAM is an example of a memristive device, as its resistance can be manipulated by applying an external electric field [35]. Figure 2.1 illustrates the conditions of RRAM cells in Low Resistance State (LRS) and High Resistance State (HRS).

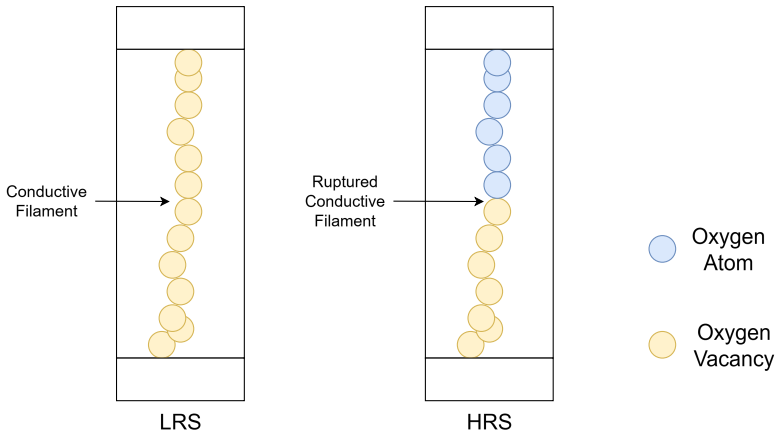

Figure 2.1: Conductive filaments in RRAM cells

The oxygen ions from the metal oxide in an RRAM device may migrate under an external voltage, and the oxygen vacancies left in the oxide would form conductive filaments within the cell, which leads to an LRS state, as illustrated in Figure 2.2.

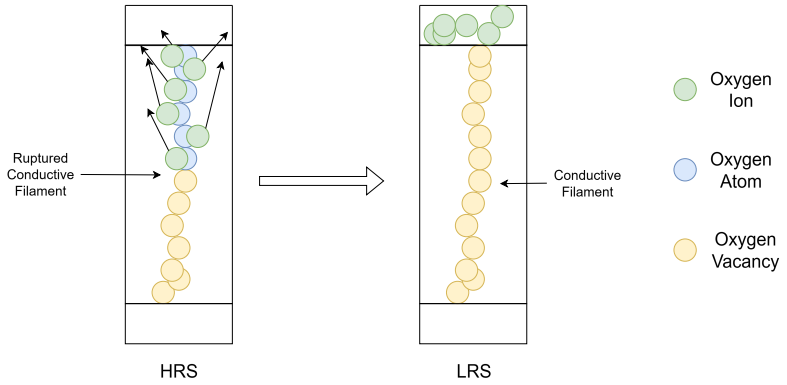

Figure 2.2: Switching from HRS to LRS in RRAM cells

When applied a voltage with a reversed polarity, the oxygen ions may migrate back to the vacancies and destroy the conductive filaments, which leads to an HRS state, as illustrated in Figure 2.3.

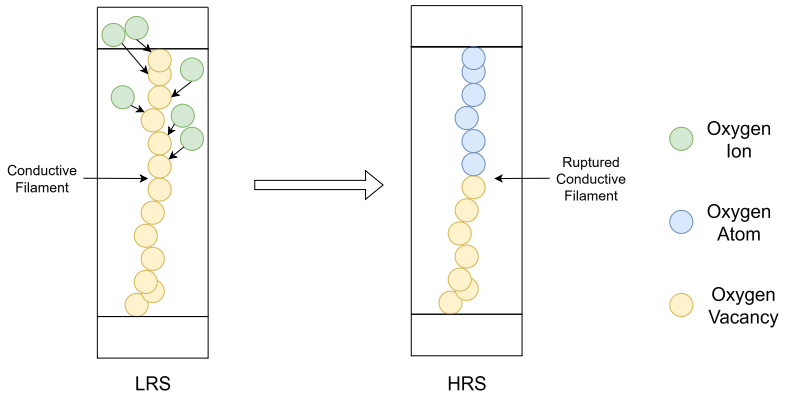

Figure 2.3: Switching from LRS to HRS in RRAM cells

In this way, the resistance of an RRAM cell can be programmed, and the information is stored as the resistance states of the cells. It is worth noting that an RRAM cell may be programmed into more than two states. A Single-Level Cell (SLC) is programmed into at most two states, while a Multi-Level Cell (MLC) allows more levels of resistance states. The data stored in a RRAM cell can be read by applying a read voltage that is smaller than the programming voltage, and the resulting current differs according to the resistance state of the RRAM cell. Though writing to and reading from the cells require applying voltages on them, the retention of data does not require an external energy supply, and therefore RRAM is a form of non-volatile memory.

Memristor devices such as RRAM are the building blocks for a genre of CIM architectures for AI [36] [9].

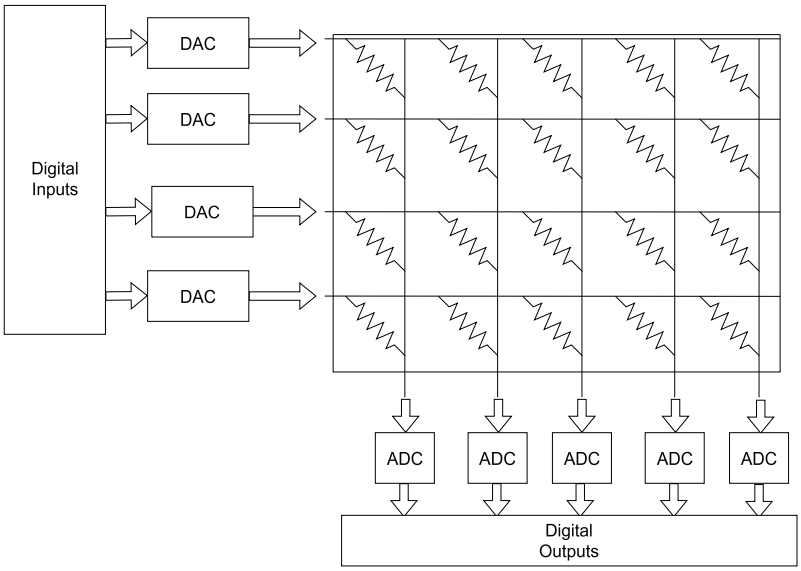

Figure 2.4: A CIM crossbar array

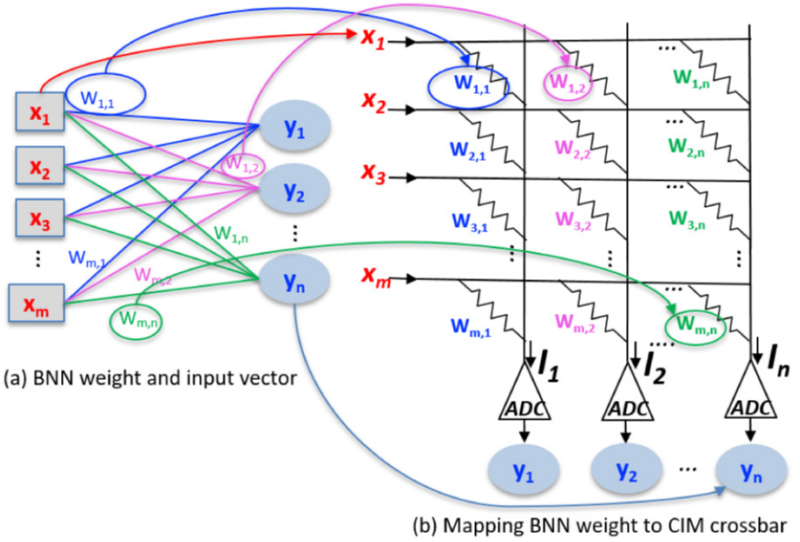

Figure 2.5: Mapping of a neural network onto a CIM crossbar array. Source: [9], ©2023 The Authors. Published by Elsevier Ltd. Under the terms of the Creative Commons Attribution-NonCommercial-No Derivatives License (CC BY NC ND).

Figure 2.4 illustrates a crossbar array of a mixed-signal CIM architecture, and Figure 2.5 illustrates the mapping of a neural network onto a the crossbar array. The weights of a neural network are programmed into the the memristor cells in the array as resistance states, and the digital inputs are converted into voltage levels by a Digital-Analogue Converter (DAC). The input voltages are applied on the memristor cells, and the resulted current from each cells

would be the product of multiplying the input voltage and the conductance of the cells. Due to Kirchhoff's Current Law (KCL), the total amount of current flowing through a column would be the sum of the current flowing through each of the cells in the column. In this way, each column of the array calculates a weighted sum in analogue, and the weighted sums are converted back to the digital domain through an Analogue-Digital Converter (ADC). This implementation allows a large scale of parallelism while avoiding a large Input/Output (IO) footprint.

2.2 Challenges for Memrister-based CIM

The benefits of memristor-based mixed-signal CIM implementation are not without costs. Designs of this genre can be affected by non-idealities in the memristor cells or the circuit, and these non-idealities lead to a reduced accuracy of computation [8][32]. Examples of major non-idealities caused by non-ideal memristor cells include variation [20], non-zero G_{min} [4], conductance drift [2], and read disturb [27].

2.2.1 Variation

When writing to a cell, the actual conductance resulted from the write voltage may deviate from the desired value, and the resulted conductance may vary from device to device and from cycle to cycle.

Device-to-Device Variation

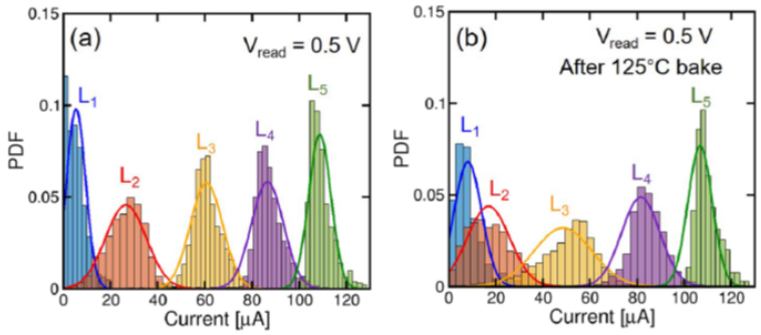

Figure 2.6: Distributions of read current obtained from cells programmed to five resistance states in an RRAM array. (a) Distributions after programming. (b) Distributions after annealing at 125°C for one hour. Source: [20], ©2021 IEEE.

The device-to-device variation is the effect that different memristor cells in the array may exhibit different switching behaviours when applied the same writing pulses of weiting voltage. It leads to differences in the resistance of the cells that are programmed to the same resistance state.

Figure 2.6 illustrates the distributions of read currents measured by Pedretti et al. [20] from different cells programmed to five resistance states in an RRAM array. The resistances of the cells are modelled with the Gaussian distribution by Pedretti et al. As is shown in the figure, the distribution of the resistance of a cell in a certain state may overlap with the resistance of another cell in a different state.

Cycle-to-cycle Variation

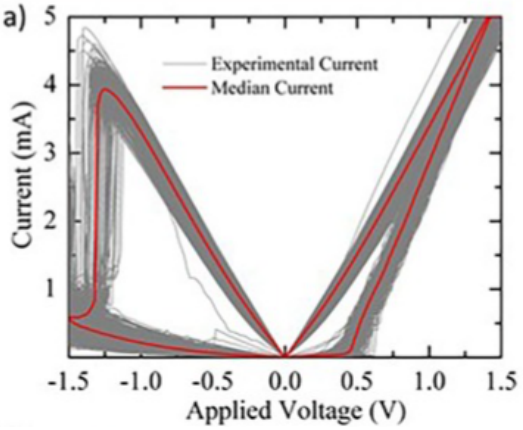

Figure 2.7: Experimental I-V curves: 450 cycles and median curve. Source: [25], ©2021 The Authors. Published by Elsevier Ltd. Under the terms of the Creative Commons CC-BY license.

The cycle-to-cycle variation is the effect that the same memristor cell may not reach the same resistance every time when it is programmed under the same writing voltage pulse. In order to ensure an accurate computation, write-read-verify schemes are often in place to compensate for the effects of variations. In Figure 2.7, the current-voltage relation of an RRAM device measured by Salvador et al. [25] is illustrated . It can be observed that current-voltage relation varies for each cycle of voltage sweep.

2.2.2 Non-zero G_{min}

The lowest possible conductance (G_{min}) of the cell is not zero, which means that the cell would allow current flowing through when applied a voltage even if the cell is programmed as a zero weight [4] [3]. Figure 2.8 illustrates an example of an error caused by non-zero minimum conductance of the cells. Ideally, multiplication with a zero weight should produce a zero product, but non-zero G_{min} may cause inaccurate results in this scenario.

2.2.3 Conductance Drift

The conductance of a memristor cell may not always stay the same after being programmed. Time and temperature may lead to a change in the conductance of the cells. In Figure 2.9,

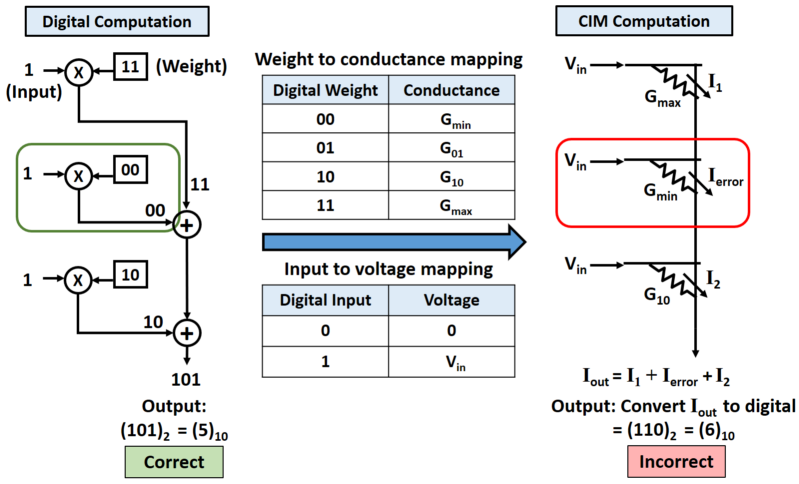

Figure 2.8: An example of error due to non-zero minimum conductance. Source: [3], ©2023 IEEE.

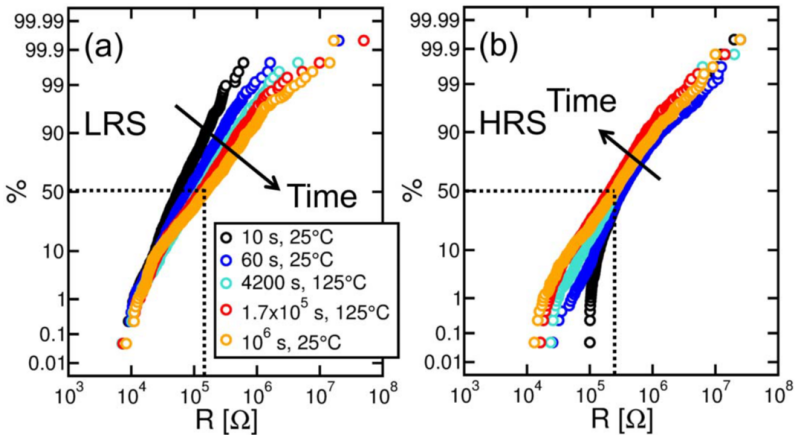

Figure 2.9: Distributions of resistance of cells in an RRAM array at different time and temperature. Source: [1], ©2015 IEEE.

changes in the distributions of resistance of RRAM cells measured by Ambrogio et al. [1] suggest that the resistance of LRS cells gradually increases, and the resistance of HRS cells gradually decreases as a result of increasing time.

2.2.4 Read Disturb

Read disturb is the effect that voltages applied on the memristor cells during read operations may introduce unwanted changes in the states of cells. For memristor-based CIM implementations of AI inference engines, the weights of the neurons are stored in the memristor cells as resistance states. During operations, the input voltages are applied to the cells every

read cycle, and these external voltages may lead to undesired changes in the resistance of the cells and a reduced accuracy.

Effects of Voltage-Induced Conductance Change

In the study by Li et al. [15], the changes in the resistance of the cells under pulses of voltages are measured. In the results shown in Figure 2.10, the cells are programmed to HRS states initially, and after applying a number of voltage impulses, the resistance of the cells tends to decrease and in some cases become indistinguishable with LRS.

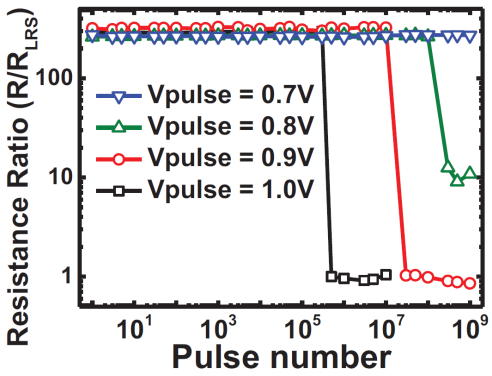

Figure 2.10: HRS disturbance under consecutive pulses of various voltages with fixed 100 ns pulse width. Source: [15], ©2014 IEEE.

In a different set of results shown in Figure 2.11, the cells are programmed to LRS initially and applied a number of consecutive voltage pulses that have a reversed polarity than their programming voltage. The resistance of the cells tend to increase after a number of pulses.

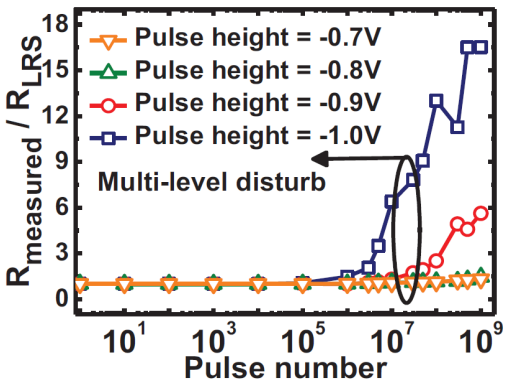

Figure 2.11: LRS disturbance under consecutive pulses of various voltages with fixed 100 ns pulse width. Source: [15], ©2014 IEEE.

Impacts of Read Disturb on Memristor-based AI Implementations

Researchers have investigated the effects of read disturb on memristor-based CIM accelerators for AI. In a study by Shim et al. [27], the effects of read disturb on the accuracy of neural networks deployed on CIM architectures are evaluated. The changes in the inference accuracy of a VGG-8 [29] neural network with the CIFAR-10 dataset is illustrated in Figure 2.12. It can be observed that the inference accuracy of the network drops significantly after the cycle of read operations reaches a certain threshold, and that a higher read voltage leads to a sooner deterioration of accuracy.

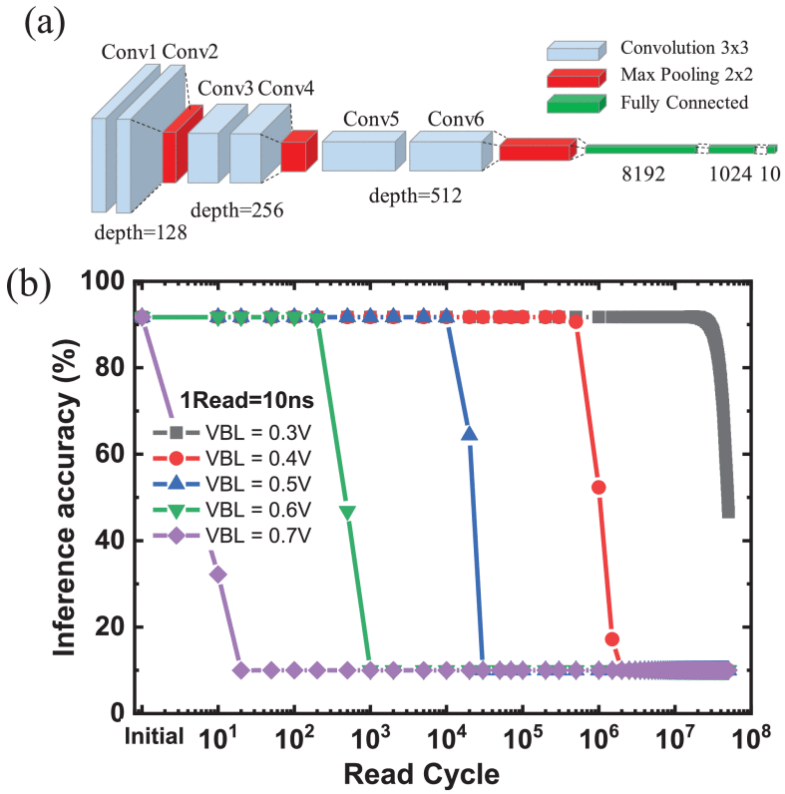

Figure 2.12: (a) The VGG-8 network architecture. (b) Impacts of read disturb on inference accuracy with CIFAR-10 data set. Source: [27], ©2020 IEEE.

Mitigations to Read Disturb

Since read disturb introduces errors to memristor-based CIM systems, countermeasures have to be in place to sustain the accuracy of computation. Periodic rewriting of the cells is often in place as a countermeasure to read disturb [28][16]. However, the period and the scope of rewriting have an impact on the performance and the power consumption of the design. A higher frequency of rewriting brings higher overheads.

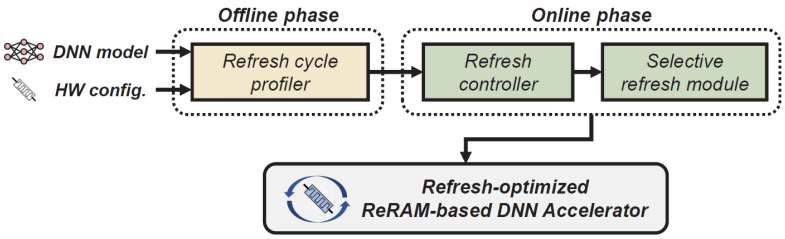

Figure 2.13: An overview of the framework proposed by Shin et al. Source: [28], ©2022 Association for Computing Machinery.

In order to minimise the overhead of rewriting, Shin et al. [28] proposed a framework that divides the cells into different groups and sets the cycles for rewrites according to the conductance states of the cells and the input voltages applied on the cells. An overview of the framework is shown in Figure 2.13.

Chapter 3

Related Work

While conventional simulation tools for CIM are based on SPICE, researchers have presented modern simulation frameworks that are capable of performing simulations at a higher level. These tools are developed with modern software engineering techniques, and they provide APIs in modern programming languages [13]. MemTorch [12], IBM analog hardware acceleration kit [24], and NeuroSim [5] are examples of modern CIM simulation tools. These simulation tools are parallelisable, and they benefit from a Graphics Processing Unit (GPU). They can be integrated with Pytorch or Tensorflow to perform simulations with neural networks. Some of these tools have an analogue layer that incorporates models of memristor devices. These models of memristor add non-idealities into the process of simulation, which makes it possible to produce results that are resemblant to reality. This chapter mainly discusses recent simulation tools in terms of their capabilities of non-ideality modelling.

3.1 MemTorch

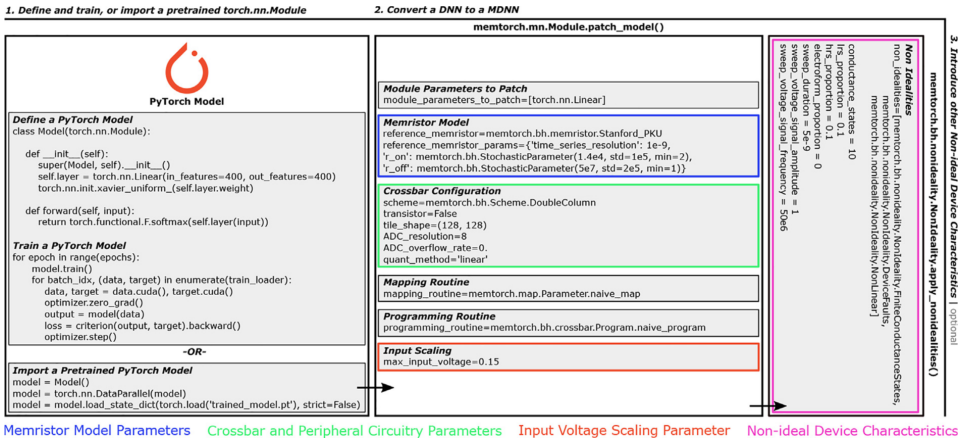

Figure 3.1: An illustration of an example of workflow on MemTorch Source: [12], ©2022 Elsevier B.V.

MemTorch [11][12] proposed by Lammie et al. is an example of modern simulation tool that does not rely on SPICE-based simulations. While MemTorch is developed in C++, CUDA, and Python, a Python interface is provided in the framework. Low level operations in the framework are hidden under abstractions and the use of APIs. The framework can be integrated with Pytorch or Tensorflow, which makes it easier to deploy the neural networks onto the simulator. An illustration of the workflow on MemTorch is shown in Figure 3.1.

The modelling of memristive devices and their non-idealities is an important part of modern CIM simulation frameworks. A module of MemTorch introduces four non-idealities related to memristive devices into the process of simulation, and they are variability, finite number of discrete conductance states, stuck-at faults, and non-linearity of the device.

Device-to-device and cycle-to-cycle variabilities are modelled stochastically in MemTorch. Finite number of conductance states is simulated by quantisation of the resistance states. Some devices are chosen randomly to be stuck at HRS or LRS in order to simulate stuck-at faults. The non-linearity of I/V characteristics of devices are modelled with a Look-up Table (LUT).

However, MemTorch assumes negligible change in the conductance of the devices during read cycles by default, which makes it infeasible for read disturb analysis.

3.2 IBM Analogue Hardware Acceleration Kit

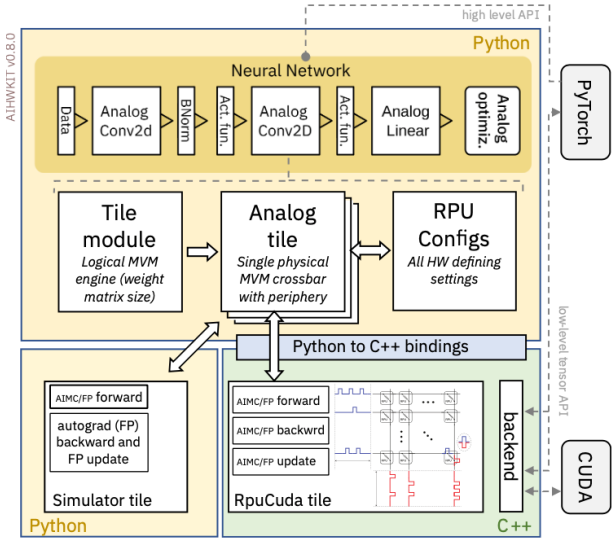

Figure 3.2: An illustration of the code structure of IBM Analogue Hardware Acceleration Kit. Source: [24], ©2021 IBM Research.

The IBM Analogue Hardware Acceleration Kit [24] is a simulation toolkit targeting neural networks deployed on analogue crossbar arrays, which is the case of memristor-based CIM. The simulation toolkit is interoperable with PyTorch, and the simulations benefit from

CUDA implementations and GPUs. The analogue layer of the simulation toolkit introduces the analogue properties into the simulations, which brings non-idealities into consideration. The code structure of the toolkit is illustrated in Figure 3.2.

By default, the IBM Analogue Hardware Acceleration Kit adopts models for Phase-Change Memory (PCM) devices and metal-oxide RRAM devices. PCM devices are another genre of memristor cells with programmable conductance [14]. For the two genres of memristor devices, variations during programming, conductance drift after programming, and read noise are modelled with statistical models. It is worth noting that read noise is not the same as read disturb, as the read noise is an instantaneous fluctuation rather than a change to the conductance states of the cells.

3.3 DNN+NeuroSim V2.0

The DNN+NeuroSim V2.0 [22] is developed on top of DNN+NeuroSim [21] and NeuroSim [5]. The simulation framework has two part, the python wrapper and the NeuroSim core. When compared to MemTorch and the IBM Analogue Hardware Acceleration Kit, the uniqueness of DNN+NeuroSim V2.0 is that the NeuroSim core provides an estimation of the hardware in terms of area, latency, and energy consumption. The structure of the simulator is illustrated in Figure 3.3. A data retention model is utilised to predict the effect of conductance drift. Nonlinearity, cycle-to-cycle variation, and device-to-device variation are also modelled.

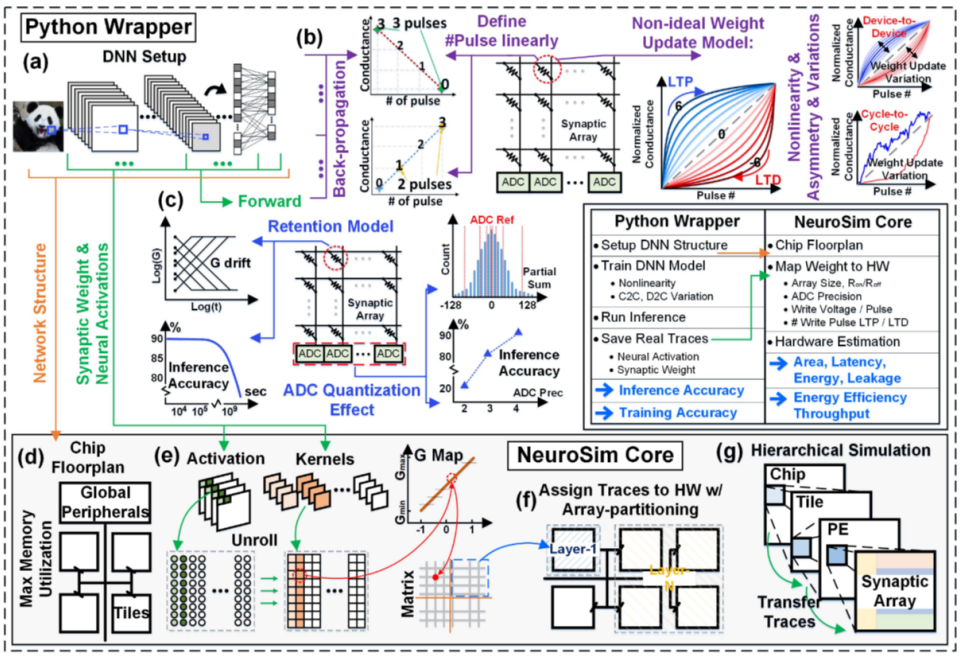

Figure 3.3: Framework structure of DNN+NeuroSim V2.0. Source: [22], ©2021, IEEE.

3.4 Comparison of Non-Ideality Modelling Support

The support for non-ideality modelling in MemTorch, the IBM Analogue Hardware Acceleration Kit, and the DNN+NeuroSim V2.0 is summarised and listed in Table 3.1.

	MemTorch [12]	**IBM toolkit** [24]	**DNN+NeuroSim V2.0** [22]
Variability	Supported	Supported	Supported
Stuck-at Faults	Supported	Not Supported	Not Supported
Read Disturb	Not Supported	Not Supported	Not Supported
Conductance Drift	Not Supported	Supported	Supported
Random Telegraph Noise	Not Supported	Supported	Not Supported

Table 3.1: Comparison of non-ideality modelling support

It can be observed that the simulation frameworks have varying degrees of support for non-ideality modelling. However, none of them incorporates read disturb in the design. Therefore, a simulation framework that involves the modelling of read disturb is needed in order to verify the feasibility of read disturb analysis on high-level simulation frameworks.

Chapter 4

RdaCIM: Read-Disturb-Aware Computation-In-Memory Simulator

4.1 Overview

The proposed RdaCIM simulator is a simulation tool for memristor-based in-memory computation. The uniqueness of RdaCIM is that it incorporates the effects of read disturb into the process of simulation. As mentioned in chapter 2, read disturb is a non-trivial non-ideality of memristor cells that has a negative impact on the computational accuracy, and it often requires proactive measures to mitigate. Periodic cell rewriting is a common countermeasure to read disturb, but it introduces overhead into the system. The incorporation of read disturb and its countermeasures makes the simulation results more resemblant to the reality, and this new feature provides a possibility for the users to perform analyses of read disturb and its countermeasures on a high-level simulation platform that is not based on SPICE.

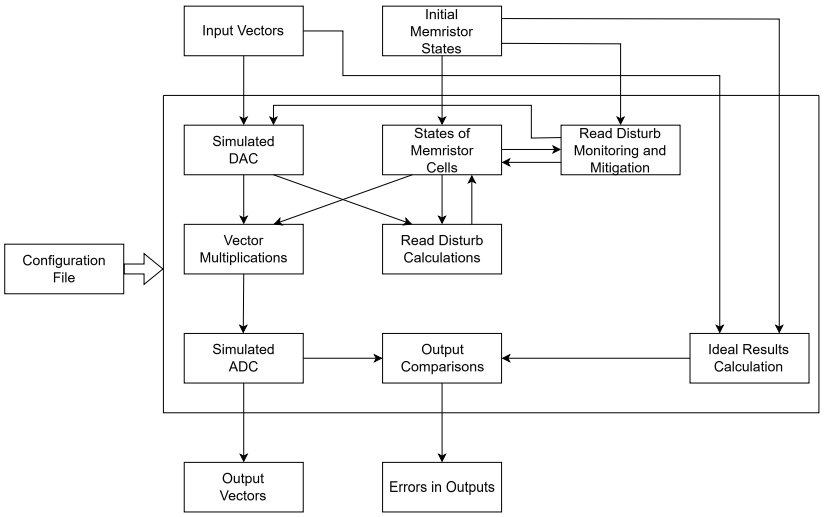

Figure 4.1: Components of RdaCIM and the data flow

The RdaCIM simulator is implemented in C, and it is made up of several modules. An overview of the software is illustrated in Figure 4.1. The simulator reads from three files that contain the input vectors, initial states of the memristor cells, and the configured parameters for the simulation. The input vectors go through a simulated DAC, and the integer elements of the vectors are converted into voltages as floating-point numbers. The voltages are multiplied with the conductance values of the memristor cells, and the sums of the products go through a simulated ADC and are converted back into integers. During this process, the conductance values of the memristor cells may deviate according to the history of reads and input voltages, which resembles the read disturb effect. The mitigation schemes targeting read disturb is also implemented. The conductance of the cells is monitored during each cycle of simulation, if a certain level of degradation is detected, the software would rewrite the cells with their initial conductance values or to adjust the output voltages of the DAC. The output vectors are compared with ideal outputs calculated from the input vectors and initial states of memristor cells in software.

In the rest of this chapter, different aspects of the implementation of RdaCIM will be discussed in detail.

4.2 Software Design of RdaCIM

4.2.1 Configuration File

As illustrated in Figure 4.1, RdaCIM accepts a configuration file that contains the parameters for the simulation. The TOML [23] format is chosen for RdaCIM, and the library `tomlc17` [33] is used to parse the TOML files. An example of a configuration file is illustrated in Listing 1.

The configuration file is parsed when RdaCIM starts, and the software begins to load the information of the input voltages and the initial states of the cells from other files. The dimensions of the array, the conductances of the cells in HRS and LRS, and the parameters of the DAC and ADC are set according to the configuration file.

4.2.2 Crossbar Array

In a CIM architecture, the crossbar array is where the data is stored and where the computation takes place. The crossbar array consists of memristor cells and the peripheries. In the software design of RdaCIM, a data structure is designed to keep a record of the states of the memristor cells, and two functions are used to simulate the behaviours of DACs and ADCs.

Data structures are an important aspect of the implementation of RdaCIM. The information of memristor cells is stored in C structures, as illustrated in Listing 2. The field `conductance` keeps a record of the current conductance of the memristor cell, and it is involved in the vector multiplications that produce the output arrays. The other three fields facilitate the simulation of the read disturb effect, which is further discussed in section 4.3. The memory space for an array of these structures is allocated on the heap by calling the malloc() function. Before the start of a simulation, the array is initialised with the data that is read from the file that contains the initial states of the cells.

Listing 1 An Example of TOML Configuration File

```
[file]
input_volts = "input_voltages.txt"
input_cells = "input_cells.txt"
output_sums = "sum_per_column.txt"
output_errors = "error_per_column.txt"

[array_size]
num_col = 300
num_row = 100

[rram_states]
resistance_high = 100.0
resistance_low = 2.0

[dac]
in_bits = 1
min_out = 0.0
max_out = 0.4

[adc]
out_bits = 10
min_in = 0.0
max_in = 204.6
initial_offset = 0.0
```

Listing 2 C Structure for a Memristor Cell

```
struct MemristorCell
{
    float initial_conductance;
    float conductance;
    float prev_voltage;
    unsigned num_read;
}
```

4.2.3 Simulation of Periphery

The DAC converts the integers from an input array into voltages represented in floating-point numbers, and the ADC converts the results of vector multiplications from currents into integers.

The maximum possible input of the DAC is decided by the number of input bits configured for the DAC, as shown in Equation 4.1, where n is the number of input bits of the

DAC.

$$Input_{max} = 2^n - 1 \tag{4.1}$$

The relation of the inputs and outputs of the DAC is shown in Equation 4.2, where $Output_{max}$ and $Output_{min}$ are the maximum and minimum outputs of the DAC, and $Input_{max}$ is its maximum possible input.

$$Output = Input \times \frac{Output_{max} - Output_{min}}{Input_{max}} + Output_{min} \tag{4.2}$$

The maximum possible output of the ADC is decided by the number of output bits configured for the ADC, as shown in Equation 4.3, where n is the number of output bits of the ADC.

$$Output_{max} = 2^n - 1 \tag{4.3}$$

The relation of the inputs and outputs of the ADC is shown in Equation 4.4, where $Input_{max}$ and $Input_{min}$ are the maximum and minimum inputs of the ADC, and $Output_{max}$ is its maximum possible input. The $Offset$ is a constant that can be configured before the simulation.

$$Output = \lfloor \frac{Input - Input_{min}}{Input_{max} - Input_{min}} \times Output_{max} + Offset \rfloor \tag{4.4}$$

4.2.4 Implementations of Vector Multiplications

The vector multiplications are at the core of the simulation. During each cycle of simulation, an input vector is multiplied with all the columns of the array. The integer elements of the vector are first converted into voltages through the DAC function, and the floating-point voltage values are multiplied with the conductance values stored in the array. The products of the vector multiplications, or weighted sums, correspond to the currents in the CIM hardware. The theoretical currents are converted into integers by the ADC function, and the results are stored and outputted.

Parallelism with OpenMP and AVX

The vector multiplications contribute significantly to the execution time of the simulation. Therefore, a parallelised implementation is desirable for faster simulations, and it allows simulations at larger scales within a reasonable runtime. A more detailed performance analysis is reported in section 5.3.

Within each cycle of simulation, the vector multiplications are accelerated with OpenMP and Advanced Vector Extensions (AVX) intrinsics, which parallelises the programme with multi-threading and SIMD instructions. The operations between each column of the array and the input vector are distributed among different Central Processing Unit (CPU) cores, and the operations between each element of the input vector and each element within each column are done in groups with AVX instructions, as illustrated in Figure 4.2.

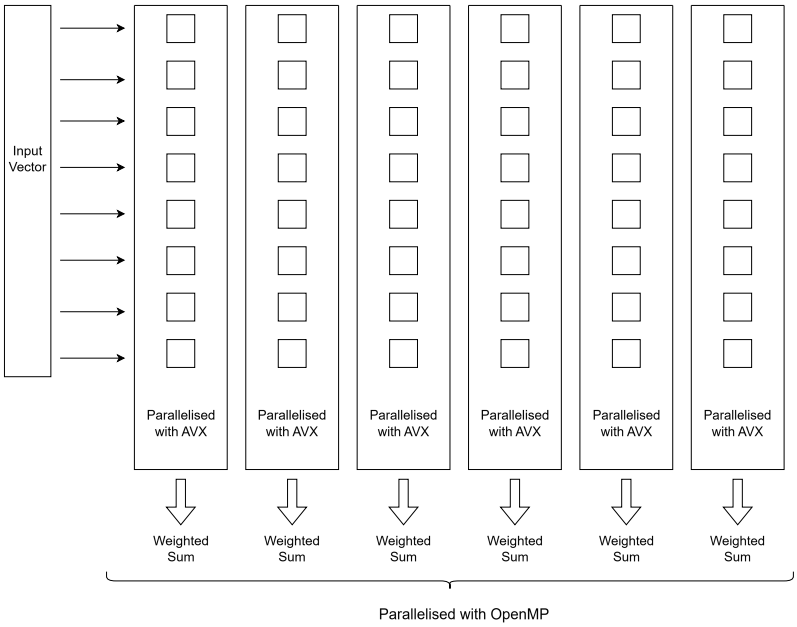

Figure 4.2: Parallelised vector multiplications

Appendix A contains a detailed implementation of a sequential version as the baseline. Details of a column-based parallelisation using OpenMP is in Appendix B. In Appendix C is an implementation that utilises both OpenMP and AVX intrinsics, as illustrated in Figure 4.2.

4.3 Read Disturb Simulation in RdaCIM

4.3.1 Incorporation of Read Disturb into RdaCIM

Modelling of Read Disturb

The consideration of read disturb is a unique feature of RdaCIM. In order to incorporate the effect of read disturb into the simulation, a fitting model proposed by Su et al. [31] is adopted in RdaCIM.

It is worth noting that the read impulses do not have the same impact on both LRS and HRS states, because the read voltages always have the same polarity, and the polarity is either the same as the set voltage or the reset voltage. In the work by Su et al. [31], the read disturb under the LRS state is modelled. In this case, it is assumed that the read voltages would affect the cells in LRS states, and their conductances would tend to decrease under the influences of read voltages.

The fitting model developed by Su et al. [31] can be divided into two stages, as illustrated in Equation 4.5. In the first stage, the conductance remains stable at its initial value G_0. After the number of reads N reaches a threshold N_T, the model evolves into the second

Initial V_{read}	0.3
G_0	500
$\frac{n_0}{c_2}$	24.0
t_ω	1000
t_0	0.1
s	0.25
c_1	0.017
α	0.07
k_B	$8.617333262 \times 10^{-5}$
T	300

Table 4.1: Example parameters for the read disturb model

stage where the conductance follows an inverse power-law.

Equation 4.7 describes the threshold for the number of reads beyond which the conductance starts to change, where t_ω is the length of the read pulse, t_0 is the point where the extrapolations of the traces of the conductance in the second stage intersect, $\frac{n_0}{c_2}$ and s are parameters derived from measurements and fitting.

The parameter p is described in Equation 4.6, where the c_1 and α are constants derived from measurements, and k_BT is the product of the Boltzmann constant and temperature.

$$G(N) = \begin{cases} G_0, & \text{for } N < N_T(V_{\text{read}}, G_0), \\ G_0 \cdot N_T^p \cdot N^{-p}, & \text{for } N \geq N_T(V_{\text{read}}, G_0). \end{cases} \tag{4.5}$$

$$p = c_1 \cdot \exp\left(\frac{\alpha |V_{\text{read}}|}{k_B T}\right) \tag{4.6}$$

$$N_T = \frac{t_0}{t_\omega} \cdot \left(\frac{n_0}{c_2}\right)^{\frac{1}{p}} \cdot G_0^{\frac{s}{(1-s)p}} \tag{4.7}$$

An example of the modelled behaviour of read disturb is illustrated in Figure 4.3. The parameters used in the example are listed in Table 4.1. The example fitting parameters are the same as the ones published by Su et al. [31], except for $\frac{n_0}{c_2}$ in Equation 4.7, which is not provided in their publication. The value of $\frac{n_0}{c_2}$ is estimated by comparing the reproduced curves in MATLAB and the curves published by Su et al.

Considerations for Variable Read Voltages

In the work by Su et al. [31], the read voltage is assumed to be the same for all read operations. However, memristor-based CIM systems may have different voltage levels as inputs, and the read voltages applied to the memristor cells can vary.

An assumption about the behaviour of read disturb under multiple voltage levels is made, and it is described by $G'(N,V)$ in Equation 4.8. Suppose V_{read} at $N = 0$ is V_0, and

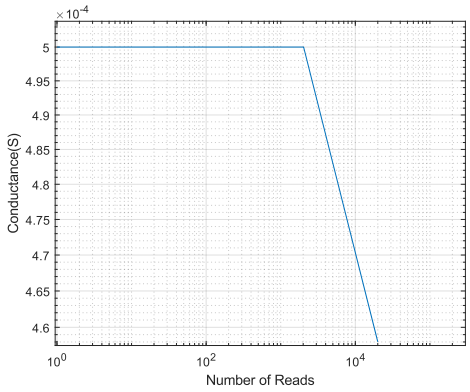

Figure 4.3: Modelled read disturb under example conditions

V_{read} at N and $N+1$ are V_N and V_{N+1} respectively.

$$\begin{cases} G'(0,V_0) = G(0,V_0), \\ G'(N+1,V_{N+1}) = G'(N,V_N) - G(N,V_{N+1}) + G(N+1,V_{N+1}). \end{cases} \tag{4.8}$$

An example of the behaviour of read disturb under this assumption is illustrated in Figure 4.4. It is under the parameters listed in Table 4.1 with a change in V_{read} from $0.3V$ to $0.25V$ at the 10000th cycle.

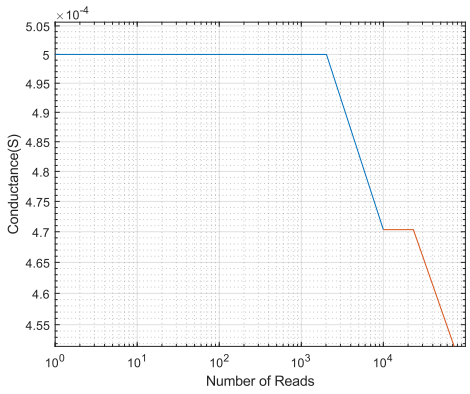

Figure 4.4: Modelled read disturb under example conditions with a change in V_{read} at 10000th cycle

4.3.2 Read Disturb Mitigation in RdaCIM

Rewriting

Since read disturb introduces unwanted changes to the states of the memristor cells, it is necessary to implement a countermeasure. Periodic rewriting is often in place to sustain

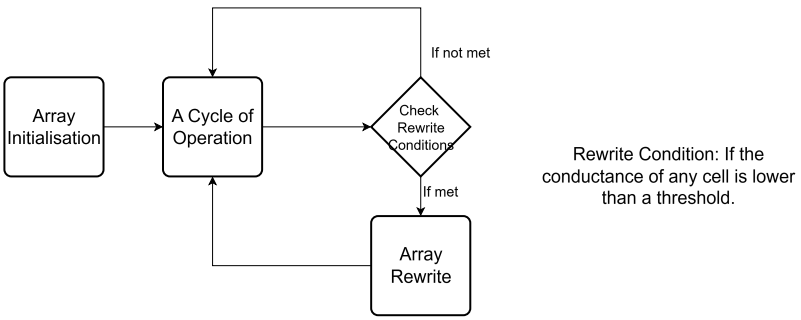

Figure 4.5: Rewriting the array according to the conditions of the cells.

the accuracy of computation, and therefore, a rewriting scheme is included into RdaCIM. After each cycle of simulation, the conductances of the cells in the array are checked against their initial values. In case that a cell is found to have a conductance that is lower than a threshold, an array rewrite will be initiated, as illustrated in Figure 4.5.

The rewrite threshold for a cell is defined to be a proportion of the initial conductance of the cell, which is determined by a rewrite factor, as in Equation 4.9.

$$Rewrite\ Threshold = Rewrite\ Factor \times Initial\ Conductance \tag{4.9}$$

The choice of the rewrite factor has an impact on the frequency of array rewrite and the accuracy of the computation. The choice of the rewrite factor is discussed in chapter 5.

Dynamic Voltage Adjustment

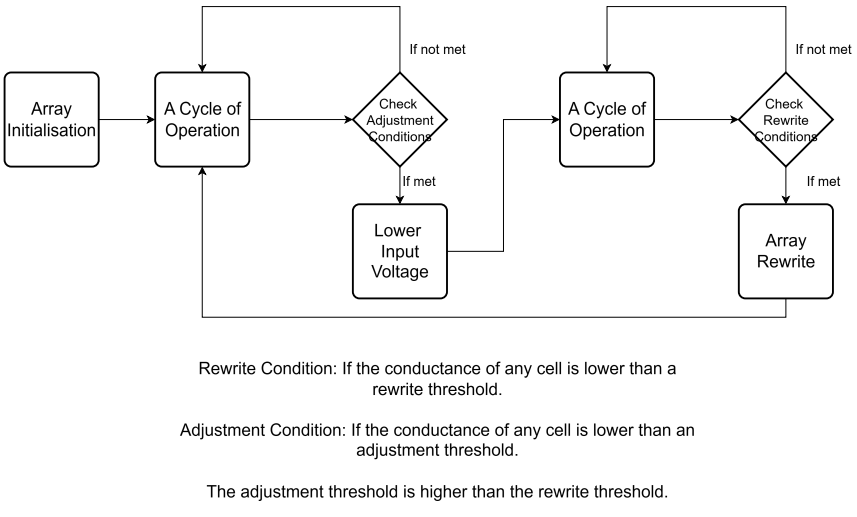

Figure 4.6: Adjusting the input voltage before rewriting the array according to the conditions of the cells.

Since a higher input voltage leads to a faster degradation of the states of the memristor cells, a design with a flexible input voltage is introduced to RdaCIM in order to verify its effect in reducing the frequency of array rewrites. The output voltages of the DAC are lowered in case that the conductance of any cell in the array is lower than an adjustment threshold that is higher than the rewrite threshold. The procedure is illustrated in Figure 4.6.

Similar to the rewrite threshold, the adjustment threshold for a cell is defined to be a proportion of the initial conductance of the cell, which is determined by a adjustment factor, as described in Equation 4.10.

$$Adjustment\ Threshold = Adjustment\ Factor \times Initial\ Conductance \tag{4.10}$$

The details of the verification of the dynamic voltage adjustment design are discussed in chapter 5.

Chapter 5

Experiments and Evaluations

5.1 Experiment Setup

5.1.1 Hardware and Operating System

The experiments are conducted on an HP ZBook Power G10 laptop running Ubuntu 24.04.2 LTS. The details of the specifications is listed in Table 5.1.

Laptop Model	HP ZBook Power G10
Processor	Intel Core i7-13700H
Memory	32 Gigabytes
Operating System	Ubuntu 24.04.2 LTS
Linux Kernel Version	6.14.0-24-generic
GCC Version	13.3.0

Table 5.1: Specifications of hardware and operating system

5.1.2 Parameters for the Read Disturb Model

As discussed in section 4.3, a fitting model proposed by Su et al. [31] is adopted to predict the effects of read disturb. Therefore, the model resembles the behaviour of the memristor cells from which the fitting parameters are extracted. The fitting parameters used in RdaCIM are the same as the ones published by Su et al. [31], except for $\frac{n_0}{c_2}$ in Equation 4.7, which is not provided in their publication. The value of $\frac{n_0}{c_2}$ used in the experiments with RdaCIM is estimated by comparing the reproduced curves in MATLAB and the curves published by Su et al. The values of the constants and parameters used in Equation 4.6 and Equation 4.7 are listed in Table 5.2. Note that the read voltage is not a constant in the experiments, as it is determined by the DAC settings.

$\frac{n_0}{c_2}$	24.0
G_0	500
t_ω	1000
t_0	0.1
s	0.25
c_1	0.017
α	0.07
k_B	$8.617333262 \times 10^{-5}$
T	300

Table 5.2: Parameters and constants for the read disturb model

5.1.3 Parameters for the DAC, ADC and the RRAM cells

The parameters for the periphery and the RRAM cells are listed in Table 5.3. Note that in this setting, the number of rows in the array should be less than 1023, as it is restricted by the number of output bits of the ADC. The maximum ADC input current is calculated from the maximum DAC output, the maximum number of rows, and the resistance of LRS.

HRS Resistance ($k\Omega$)	100.0
LRS Resistance ($k\Omega$)	2.0
Number of DAC bits	1
Minimum DAC output (V)	0.0
Maximum DAC output (V)	0.3
Number of ADC output bits	10
Minimum ADC input (mA)	0.0
Maximum ADC input (mA)	153.45

Table 5.3: Parameters for the DAC, ADC and the RRAM cells

The initial states of the RRAM cells are set to be randomly HRS or LRS. The digital inputs for every row and every cycle are set to be randomly 1 or 0, which means that the DAC outputs are randomly 0V or 0.3V.

5.2 Read Disturb Mitigation Results

An important part of this work is to verify the feasibility of read disturb analysis in high-level simulation tools. As mentioned in subsection 4.3.2, the rewriting and the dynamic voltage adjustment schemes are implemented in RdaCIM.

5.2.1 Verification of Simulation Results

Since the non-ideal effects are incorporated in RdaCIM, the results produced from simulations may deviate from their ideal values. As mentioned in chapter 4, a result verification

module is utilised to analyse the difference between ideal outputs and simulated outputs. An example of a comparison is illustrated in Figure 5.1. The software produces a set of ideal

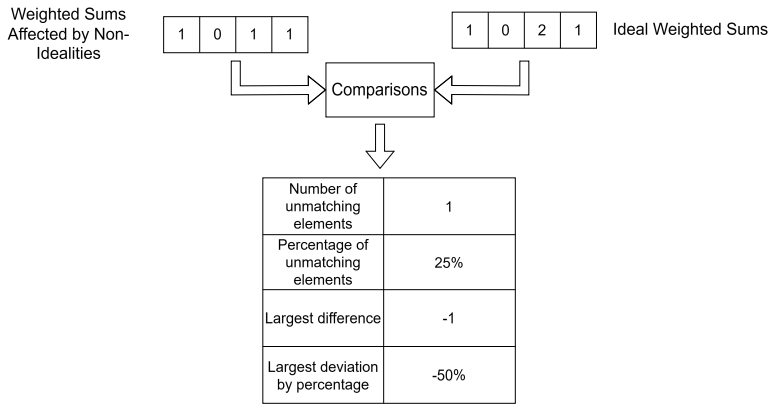

Figure 5.1: An example of a comparison between ideal and simulated results.

results by performing vector multiplications between the input vectors and the initial states of the cells digitally without involving the impacts from non-idealities. Every weighted sums generated from simulations is compared with its counterpart in the ideal output vector. The number of unmatching results, the percentage of unmatching results, and the largest difference found in the unmatching results are reported.

5.2.2 Rewriting

Rewrite Threshold	**% of Non-ideal Outputs**	**Number of Rewrites**
0 (no rewrite)	27.779099 %	0
0.98 $\times$ initial conductance	2.274100 %	1
0.985 $\times$ initial conductance	0.206833 %	1
0.99 $\times$ initial conductance	0.003967 %	1
0.991 $\times$ initial conductance	0.000933 %	2
0.992 $\times$ initial conductance	0.000167 %	2
0.9925 $\times$ initial conductance	0.000067 %	2
0.993 $\times$ initial conductance	0 %	2
0.9935 $\times$ initial conductance	0 %	2
0.994 $\times$ initial conductance	0 %	2

Table 5.4: Number of non-ideal outputs from a 300×100 array per 10000 cycles at different rewrite thresholds

The rewriting scheme as illustrated in Figure 4.5 is crucial to maintain the computational accuracy in the event of read disturb. Choosing a suitable rewrite threshold is the key to minimise the rewrite frequency while sustaining the accuracy. A set of experiments have

been done in order to find a suitable value for the threshold. The measurements are done with a 300×100 array under 10000 cycles of simulation. The number of non-ideal outputs and the number of array rewrites are recorded.

The results in Table 5.4 suggests that the rewriting scheme is an effective countermeasure against read disturb, as the percentage of non-ideal results dropped from more than 27% to zero. The minimum rewrite threshold should be 0.993 times of the initial conductance of the cells, in order to prevent non-ideal outputs.

5.2.3 Dynamic Voltage Adjustment

The dynamic voltage adjustment scheme as illustrated in Figure 4.6 is developed on top of the rewriting scheme. It is aimed to reduce the number of rewrites by adjusting the input voltages applied on the cells in the crossbar array.

A set of measurements are done with and without the dynamic voltage adjustment scheme. For the case of dynamic adjustment scheme, the maximum DAC output voltage would change from 0.3V to 0.264V when the conductance of any of the cells is below 0.995 times of its initial conductance. For both cases, the threshold for array rewrites is set at 0.9935 times of the initial conductance of the cell.

Nr. of Cycles	**Nr. of Rewrites** Voltage fixed at 0.3V	**Nr. of Rewrites** Voltage Adjustment	**Decrease in Nr. of Rewrites**
10000	2	0	100%
20000	4	0	100%
30000	6	1	83.3%
40000	8	1	87.5%
50000	10	2	80.0%
60000	12	2	83.3%
70000	15	3	80.0%
80000	17	3	82.4%
90000	19	4	78.9%
100000	21	4	81.0%

Table 5.5: Number array rewrites per different number of cycles of simulation on a 300×100 array with and without dynamic voltage adjustment

The results are illustrated in Table 5.5. The rewriting scheme with dynamic voltage adjustment requires 78.9% to 87.5% less frequent rewrites than the scheme without dynamic voltage adjustment does, and the scheme eliminates the need for rewriting when the number of cycles is less than 20000. Therefore, it is safe to conclude that a dynamic voltage adjustment scheme reduces the overhead of array rewrites significantly.

Note that as the number of cycles increases, the number of non-ideal outputs does not necessarily remain zero. While the percentage remains low, a small number of non-ideal weighted sums are observed. Since the DAC inputs are distributed randomly, it is suspected that when the number of cycles increases, it would be more likely to have edge cases where

the input vector contains predominantly 1s. In these cases, the states of the cells would degrade faster, and the threshold found in subsection 5.2.2 may not be sufficient to ensure an error-free output in these edge cases.

5.3 Performance Benchmarking

The execution time is an important factor for simulations, especially when the scale of the simulation is large. A faster simulation tool allows simulations at larger scales within a reasonable runtime. Therefore, it is desirable to perform a performance benchmarking on RdaCIM. The benchmarks are done on an environment with a specification listed in Table 5.1.

5.3.1 Baseline

As mentioned in section 4.2.4, OpenMP and AVX are utilised to parallelise the software. Before discussing the parallelised versions of RdaCIM, a benchmarking of the sequential version as the baseline is useful to elaborate the growth in the execution time with respect to the scale of the simulation.

Two sets of benchmarks on the sequential version have been done. In one of them, the number of cycles in simulations is kept at 1000, and the size of the array ranges from 30×10 to 30000×1000. The results are illustrated in Table 5.6.

In the second set of benchmarks, the size of the array is kept at 300×100, and the number of cycles of simulations ranges from 10 to 100000. The results are illustrated in Table 5.7.

The two sets of benchmarks have shown that the execution time of simulations grow linearly with respect to the number of cycles or the size of the array. Since a reasonable execution time can be a factor that limits the scale of the simulation, an acceleration of RdaCIM helps to scale up the simulations.

Array Size (col $\times$ row)	**Time (second) per 1000 cycles**
30×10	0.015616
30×100	0.157012
300×10	0.139347
300×100	1.537799
3000×100	15.037787
300×1000	16.471950
3000×1000	172.211717
30000×1000	1802.509634

Table 5.6: Execution time per 1000 cycles of simulations of different sizes of arrays in baseline version

Number of Cycles	Time (second)
10	0.015516
100	0.152087
1000	1.515688
10000	15.129828
100000	152.350899

Table 5.7: Execution time of different amount of cycles of simulations with a 300 × 100 array in baseline version

5.3.2 Acceleration by OpenMP and AVX Intrinsics

In order to reduce the execution time of simulations on RdaCIM, multi-threading and SIMD instructions are used to fully utilise the computing power provided by the CPU of the hardware platform. The implementation of RdaCIM is made multi-threaded with the OpenMP library, and the AVX intrinsics are used to achieve SIMD operations.

Array Size (col × row)	Time - Baseline	Time - OpenMP	Speedup
30 × 10	0.015616s	0.021145s	0.7385×
30 × 100	0.157012s	0.058557s	2.6814×
300 × 10	0.139347s	0.048311s	2.8844×
300 × 100	1.537799s	0.315113s	4.8802×
3000 × 100	15.037787s	3.171794s	4.7411×
300 × 1000	16.471950s	3.173227s	5.1909×
3000 × 1000	172.211717s	31.267505s	5.5077×
30000 × 1000	1802.509634s	236.795041s	7.6121×

Table 5.8: Comparison of baseline and OpenMP version - Execution time per 1000 cycles of simulations of different sizes of arrays

Array Size (col × row)	Time - Baseline	Time - OpenMP+AVX	Speedup
30 × 10	0.015616s	0.027045s	0.5774×
30 × 100	0.157012s	0.043428s	3.6155×
300 × 10	0.139347s	0.040451s	3.4445×
300 × 100	1.537799s	0.218156s	7.0491×
3000 × 100	15.037787s	2.112029s	7.1201×
300 × 1000	16.471950s	2.213611s	7.4412×
3000 × 1000	172.211717s	20.532790s	8.3872×
30000 × 1000	1802.509634s	185.242496s	9.7305×

Table 5.9: Comparison of baseline and OpenMP+AVX version - Execution time per 1000 cycles of simulations of different sizes of arrays

Two sets of benchmarking have been done to demonstrate the improvement achieved

by parallelisation. The first set of benchmarking is done on a version of RdaCIM that is parallelised by OpenMP only, and the second set of benchmarking involves the version of RdaCIM that combines OpenMP and AVX implementations. During both sets of benchmarking, the simulation runs for 1000 cycles, and the size of the arrays under simulation range from 30×10 to 30000×1000. The results are illustrated in Table 5.8 and Table 5.9.

It can be observed that the two parallelised versions achieve higher speed-ups towards the baseline when the size of array is larger, and starting from an array size of 30×100, the two parallelised versions are faster than the baseline. For simulations with an array as large as 30000×1000, the speed-up achieved by the OpenMP version is more than 7 times, and combining OpenMP and AVX achieves a speed-up more than 9 times. However, the two parallelised versions are slower than the baseline when the array size is 30×10. It can be suspected when the array size is small enough, the overhead brought by thread management, synchronisation, and extra data movement would cancel out the benefits of the parallelisation.

Chapter 6

Conclusions and Future Work

6.1 Conclusion and Answers to Research Questions

This work presents RdaCIM, a high-level simulation tool for CIM that incorporates the effects of read disturb as a part of the simulation. Compared to conventional SPICE-based simulation tools, RdaCIM offers a faster and more flexible simulation experience. It also fills the gap that other high-level CIM simulation frameworks do not include models for read disturb by default.

Research Question 1: Is it feasible to perform analyses of read disturb on a CIM simulation tool that is not based on SPICE?

RdaCIM utilises an analytical model that is derived from real devices for the simulation of the effects brought by read disturb. Based on this functionality, several experiments have been done to verify the feasibility of read disturb analysis on a high-level simulation framework that is not based on SPICE. The experiments done with RdaCIM have suggested the effectiveness of threshold-based periodic rewriting as a countermeasure to read disturb. The results in subsection 5.2.2 have shown that a rewriting scheme with a rewrite threshold compensates for the non-ideal effects brought by read disturb. A further investigation into the relation between the dynamic voltage adjustment scheme and the frequency of rewrites has been done with RdaCIM. The measurements shown in subsection 5.2.3 have suggested the dynamic voltage adjustment scheme brings a 78.9% to 87.5% decrease in the frequency of rewrites, and therefore it is proven that the scheme is an effective method to reduce the overhead brought by periodic rewriting of RRAM cells. These experiments and the results have proven the feasibility of read disturb analysis on a high-level simulation tools without the usage of SPICE.

Research Question 2: What are the benefits brought by a parallelisable implementation of simulation tool in terms of speed and scalability?

An performance benchmark on RdaCIM has shown that CIM simulation tools benefit from parallelised implementations. By combining different levels of parallelisation brought by

OpenMP and AVX, a speed-up up to 9.7305 times has been observed. The acceleration of the simulation tool makes it possible to perform simulations at a larger scale within a reasonable time frame. Therefore, introducing parallelism into CIM simulation tools improves the user experience and expands the use cases.

6.2 Limitations of RdaCIM

When compared to other modern simulation frameworks for memrister-based CIM accelerators, RdaCIM lacks a simple interface with modern neural network frameworks, which makes it more complex to perform simulations for AI applications.

RdaCIM exploits the parallelism provided by multi-core CPUs and SIMD instructions, but a support for GPU acceleration is expected to provide a larger speed-up for simulations at large scales.

Considering the fact that the analytical model for read disturb is implemented in C as a part of RdaCIM, it may not be easy to replace or update the model. Software engineering techniques can be explored to facilitate portable implementations of models for non-idealities.

6.3 Future Work

6.3.1 Future Work Related to Simulation Tools

Considering the limitations mentioned in section 6.2, potential future work includes the implementation of interfaces with neural network frameworks, GPU acceleration, and portable models for non-idealities. However, it is also desirable to investigate other aspects related to this work.

RdaCIM includes a model for read disturb, but it is not the only non-idealities introduced into the design. Since the resistances of the memristor cells are set to finite numbers, the simulations are also affected by finite LRS/HRS ratio and non-zero G_{min}. it can be desirable to investigate the interactions of these non-idealities under inputs with different distributions.

Based on the work with RdaCIM, it is also desirable to extend other simulation frameworks, such as MemTorch [12] and the IBM Analogue Hardware Acceleration Kit [24], with a model for read disturb.

Acceleration for AI is not the only potential application for memristor-based CIM architectures. Future work may include exploring the use cases of RdaCIM in the fields of scientific computation, signal processing, and genome sequencing.

Though the CIM architecture addresses some of the limitations of the conventional Von-Neumann architecture, the Von-Neumann architecture still plays an essential role in general-purpose computing. A computing system with a CIM design is likely to involve both a CIM accelerator and a general-purpose processor. Therefore, future work may include combining RdaCIM with a simulator for general-purpose processors.

6.3.2 Future Work Related to Non-Idealities in CIM

Most of the investigations into non-idealities of CIM and their mitigation methods have been done in an empirical approach. However, it can be desirable to investigate the feasibility of a theoretical approach in the research. Suppose the non-idealities involved in CIM systems can be divided into two categories. The first category includes the non-idealities that are predictable, which means that their effects and onsets can be predicted by factors that are observable by the developer of the architecture. Read disturb is an example of the non-idealities in the first category, as its effects can be predicted by the read voltages and the number of reads. The second category includes the non-idealities whose effects or onsets are not predictable by the developers of the architecture, such as cycle-to-cycle variations, which is caused by the stochastic nature of the switching of HRS and LRS. The mitigation methods to the second category of non-idealities mainly involves redundancy, whether in time or in area. For example, the write-verify schemes targeting variations involves performing a higher number of write operations to the cells, and they can be seen as a form of redundancy. For the second category of non-idealities, it may be desirable to investigate quantitatively the relation between the number of redundancy and the upper bound of computational accuracy of the system.

Bibliography

[1] Stefano Ambrogio, Simone Balatti, Zhong Qiang Wang, Yu-Sheng Chen, Heng-Yuan Lee, Frederick T. Chen, and Daniele Ielmini. Data retention statistics and modelling in hfo2 resistive switching memories. In *2015 IEEE International Reliability Physics Symposium*, pages MY.7.1–MY.7.6, April 2015. doi: 10.1109/IRPS.2015.7112810.

[2] Andrea Baroni, Artem Glukhov, Eduardo Pérez, Christian Wenger, Daniele Ielmini, Piero Olivo, and Cristian Zambelli. Low conductance state drift characterization and mitigation in resistive switching memories (rram) for artificial neural networks. *IEEE Transactions on Device and Materials Reliability*, 22(3):340–347, Sep. 2022. ISSN 1558-2574. doi: 10.1109/TDMR.2022.3182133.

[3] Rajendra Bishnoi, Sumit Diware, Anteneh Gebregiorgis, Simon Thomann, Sara Mannaa, Bastien Deveautour, Cédric Marchand, Alberto Bosio, Damien Deleruyelle, Ian O'Connor, Hussam Amrouch, and Said Hamdioui. Energy-efficient computation-in-memory architecture using emerging technologies. In *2023 International Conference on Microelectronics (ICM)*, pages 325–334, Dec 2023. doi: 10.1109/ICM60448.2023.10378889.

[4] Pai-Yu Chen and Shimeng Yu. Technological benchmark of analog synaptic devices for neuroinspired architectures. *IEEE Design Test*, 36(3):31–38, June 2019. ISSN 2168-2364. doi: 10.1109/MDAT.2018.2890229.

[5] Pai-Yu Chen, Xiaochen Peng, and Shimeng Yu. Neurosim: A circuit-level macro model for benchmarking neuro-inspired architectures in online learning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(12):3067–3080, Dec 2018. ISSN 1937-4151. doi: 10.1109/TCAD.2018.2789723.

[6] L. Chua. Memristor-the missing circuit element. *IEEE Transactions on Circuit Theory*, 18(5):507–519, Sep. 1971. ISSN 2374-9555. doi: 10.1109/TCT.1971.1083337.

[7] Hadi Esmaeilzadeh, Emily Blem, Renee St. Amant, Karthikeyan Sankaralingam, and Doug Burger. Dark silicon and the end of multicore scaling. In *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ISCA '11, page

365–376, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450304726. doi: 10.1145/2000064.2000108.

[8] Anteneh Gebregiorgis, Abhairaj Singh, Sumit Diware, Rajendra Bishnoi, and Said Hamdioui. Dealing with non-idealities in memristor based computation-in-memory designs. In *2022 IFIP/IEEE 30th International Conference on Very Large Scale Integration (VLSI-SoC)*, pages 1–6, Oct 2022. doi: 10.1109/VLSI-SoC54400.2022.9939618.

[9] Anteneh Gebregiorgis, Abhairaj Singh, Amirreza Yousefzadeh, Dirk Wouters, Rajendra Bishnoi, Francky Catthoor, and Said Hamdioui. Tutorial on memristor-based computing for smart edge applications. *Memories - Materials, Devices, Circuits and Systems*, 4:100025, 2023. ISSN 2773-0646. doi: https://doi.org/10.1016/j.memori.2023.100025. URL https://www.sciencedirect.com/science/article/pii/S2773064623000026.

[10] Said Hamdioui, Shahar Kvatinsky, Gert Cauwenberghs, Lei Xie, Nimrod Wald, Siddharth Joshi, Hesham Mostafa Elsayed, Henk Corporaal, and Koen Bertels. Memristor for computing: Myth or reality? In *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, pages 722–731, March 2017. doi: 10.23919/DATE.2017.7927083.

[11] Corey Lammie and Mostafa Rahimi Azghadi. Memtorch: A simulation framework for deep memristive cross-bar architectures. In *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, Oct 2020. doi: 10.1109/ISCAS45731.2020.9180810.

[12] Corey Lammie, Wei Xiang, Bernabé Linares-Barranco, and Mostafa Rahimi Azghadi. Memtorch: An open-source simulation framework for memristive deep learning systems. *Neurocomputing*, 485:124–133, 2022. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2022.02.043. URL https://www.sciencedirect.com/science/article/pii/S0925231222002053.

[13] Corey Lammie, Wei Xiang, and Mostafa Rahimi Azghadi. Modeling and simulating in-memory memristive deep learning systems: An overview of current efforts. *Array*, 13:100116, 2022. ISSN 2590-0056. doi: https://doi.org/10.1016/j.array.2021.100116. URL https://www.sciencedirect.com/science/article/pii/S2590005621000540.

[14] Manuel Le Gallo and Abu Sebastian. An overview of phase-change memory device physics. *Journal of Physics D: Applied Physics*, 53(21):213002, mar 2020. doi: 10.1088/1361-6463/ab7794. URL https://dx.doi.org/10.1088/1361-6463/ab7794.

[15] Haitong Li, Hong-Yu Chen, Zhe Chen, Bing Chen, Rui Liu, Gang Qiu, Peng Huang, Feifei Zhang, Zizhen Jiang, Bin Gao, Lifeng Liu, Xiaoyan Liu, Shimeng Yu, H.-S. Philip Wong, and Jinfeng Kang. Write disturb analyses on half-selected cells of

cross-point rram arrays. In *2014 IEEE International Reliability Physics Symposium*, pages MY.3.1–MY.3.4, June 2014. doi: 10.1109/IRPS.2014.6861158.

[16] Jilan Lin, Cheng-Da Wen, Xing Hu, Tianqi Tang, Ing-Chao Lin, Yu Wang, and Yuan Xie. Rescuing rram-based computing from static and dynamic faults. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 40(10):2049–2062, Oct 2021. ISSN 1937-4151. doi: 10.1109/TCAD.2020.3037316.

[17] John Taylor Maurer, Ahmed Mamdouh Mohamed Ahmed, Parsa Khorrami, Sabrina Hassan Moon, and Dayane Alfenas Reis. A survey on computing-in-memory (cim) and emerging nonvolatile memory (nvm) simulators. *Chips*, 4(2), 2025. ISSN 2674-0729. doi: 10.3390/chips4020019. URL https://www.mdpi.com/2674-0729/4/2/19.

[18] Laurence W. Nagel and D.O. Pederson. Spice (simulation program with integrated circuit emphasis). Technical Report UCB/ERL M382, Apr 1973. URL http://www2.eecs.berkeley.edu/Pubs/TechRpts/1973/22871.html.

[19] D. Pederson. A historical review of circuit simulation. *IEEE Transactions on Circuits and Systems*, 31(1):103–111, January 1984. ISSN 1558-1276. doi: 10.1109/TCS.1984.1085422.

[20] Giacomo Pedretti, Elia Ambrosi, and Daniele Ielmini. Conductance variations and their impact on the precision of in-memory computing with resistive switching memory (rram). In *2021 IEEE International Reliability Physics Symposium (IRPS)*, pages 1–8, March 2021. doi: 10.1109/IRPS46558.2021.9405130.

[21] Xiaochen Peng, Shanshi Huang, Yandong Luo, Xiaoyu Sun, and Shimeng Yu. Dnn+neurosim: An end-to-end benchmarking framework for compute-in-memory accelerators with versatile device technologies. In *2019 IEEE International Electron Devices Meeting (IEDM)*, pages 32.5.1–32.5.4, Dec 2019. doi: 10.1109/IEDM19573.2019.8993491.

[22] Xiaochen Peng, Shanshi Huang, Hongwu Jiang, Anni Lu, and Shimeng Yu. Dnn+neurosim v2.0: An end-to-end benchmarking framework for compute-in-memory accelerators for on-chip training. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 40(11):2306–2319, Nov 2021. ISSN 1937-4151. doi: 10.1109/TCAD.2020.3043731.

[23] Tom Preston-Werner. Toml: Tom's obvious minimal language. URL https://toml.io.

[24] Malte J. Rasch, Diego Moreda, Tayfun Gokmen, Manuel Le Gallo, Fabio Carta, Cindy Goldberg, Kaoutar El Maghraoui, Abu Sebastian, and Vijay Narayanan. A flexible and fast pytorch toolkit for simulating training and inference on analog crossbar arrays. In *2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pages 1–4, June 2021. doi: 10.1109/AICAS51828.2021.9458494.

[25] E. Salvador, M.B. Gonzalez, F. Campabadal, J. Martin-Martinez, R. Rodriguez, and E. Miranda. Spice modeling of cycle-to-cycle variability in rram devices. *Solid-State Electronics*, 185:108040, 2021. ISSN 0038-1101. doi: https://doi.org/10.1016/j.sse.2021.108040. URL https://www.sciencedirect.com/science/article/pii/S003811012100085X.

[26] John Shalf. The future of computing beyond moore's law. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 378: 20190061, 2020. doi: 10.1098/rsta.2019.0061.

[27] Wonbo Shim, Yandong Luo, Jae-Sun Seo, and Shimeng Yu. Investigation of read disturb and bipolar read scheme on multilevel rram-based deep learning inference engine. *IEEE Transactions on Electron Devices*, 67(6):2318–2323, June 2020. ISSN 1557-9646. doi: 10.1109/TED.2020.2985013.

[28] Hyein Shin, Myeonggu Kang, and Lee-Sup Kim. Re2fresh: A framework for mitigating read disturbance in reram-based dnn accelerators. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '22, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392174. doi: 10.1145/3508352.3549345. URL https://doi.org/10.1145/3508352.3549345.

[29] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015. URL https://arxiv.org/abs/1409.1556.

[30] Dmitri B Strukov, Gregory S Snider, Duncan R Stewart, and R Stanley Williams. The missing memristor found. *Nature*, 453(7191):80–83, 2008. ISSN 1476-4687. doi: 10.1038/nature06932.

[31] Po-Cheng Su, Cheng-Min Jiang, Yu-Jia Chen, Chih-Chieh Wang, Kai-Shin Li, Chao-Cheng Lin, and Tahui Wang. Analytical modeling of read-induced set-state conductance change in a hafnium-oxide resistive switching device. *IEEE Transactions on Electron Devices*, 67(1):113–117, Jan 2020. ISSN 1557-9646. doi: 10.1109/TED.2019.2953781.

[32] Xiaoyu Sun and Shimeng Yu. Impact of non-ideal characteristics of resistive synaptic devices on implementing convolutional neural networks. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 9(3):570–579, Sep. 2019. ISSN 2156-3365. doi: 10.1109/JETCAS.2019.2933148.

[33] CK Tan. tomlc17. URL https://github.com/cktan/tomlc17.

[34] Sascha Vongehr and Xiangkang Meng. The missing memristor has not been found. *Scientific Reports*, 5(1):11657, 2015. ISSN 2045-2322. doi: 10.1038/srep11657.

[35] 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. Metal–oxide rram.

Proceedings of the IEEE, 100(6):1951–1970, June 2012. ISSN 1558-2256. doi: 10.1109/JPROC.2012.2190369.

[36] Shimeng Yu, Wonbo Shim, Xiaochen Peng, and Yandong Luo. Rram for compute-in-memory: From inference to training. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 68(7):2753–2765, July 2021. ISSN 1558-0806. doi: 10.1109/TCSI .2021.3072200.

Appendix A

Sequential Implementation

```
void crossbar_kcl(const int num_col, const int num_row,
                  const int* digital_inputs, int* arr_col_sum,
                  struct MemristorCell* ptr_arr_cell,
                  enum voltage_mode voltage_mode)
{
    for (int i = 0; i < num_col; i++)
    {
        float temp_current = 0;
        for (int j = 0; j < num_row; j++)
        {
            temp_current = temp_current +
                dac(digital_inputs[j], voltage_mode) *
                ptr_arr_cell[j * num_col + i].conductance * 1000;
            read_disturb(
                dac(digital_inputs[j],
                voltage_mode),
                &ptr_arr_cell[j * num_col + i]);
        }
        arr_col_sum[i] = adc(temp_current, voltage_mode);
    }
}
```

Appendix B

Implementation with OpenMP

```
void crossbar_kcl(const int num_col, const int num_row,
                  const int* digital_inputs, int* arr_col_sum,
                  struct MemristorCell* ptr_arr_cell,
                  enum voltage_mode voltage_mode)
{
    omp_set_num_threads(omp_get_num_procs());
    int i;
    int j;
#pragma omp parallel for schedule(static) default(none) \
    shared(ptr_arr_cell, voltage_mode, num_col, \
    num_row, digital_inputs, arr_col_sum)\
    private(j)
    for (i = 0; i < num_col; i++)
    {
        float temp_current = 0;
        for (j = 0; j < num_row; j++)
        {
            temp_current += dac(digital_inputs[j], voltage_mode) *
                ptr_arr_cell[i * num_row + j].conductance * 1000;
            read_disturb(dac(digital_inputs[j], voltage_mode),
                &ptr_arr_cell[i * num_row + j]);
        }
        arr_col_sum[i] = adc(temp_current, voltage_mode);
    }
}
```

Appendix C

Implementation with OpenMP and AVX

```
    void crossbar_kcl(const int num_col, const int num_row,
                      const int* digital_inputs, int* arr_col_sum,
                      struct MemristorCell* ptr_arr_cell,
                      enum voltage_mode voltage_mode)
{
    omp_set_num_threads(omp_get_num_procs());
    int i;
    int j;
    __m256 constant_vec = _mm256_set1_ps(1000);
    __m256 dac_current_vec;
    __m256 cell_conductance_vec;
    __m256 temp_current_sum_vec;
#pragma omp parallel for schedule(static) default(none) \
     shared(ptr_arr_cell, voltage_mode, num_col, num_row, \
     digital_inputs, arr_col_sum, constant_vec) \
     private(j, temp_current_sum_vec, dac_current_vec, \
     cell_conductance_vec)
    for (i = 0; i < num_col; i++)
    {
        float temp_current = 0;
        temp_current_sum_vec = _mm256_set1_ps(0);
        for (j = 0; j < num_row; j = j + 8)
        {
            if (num_row - j >= 8)
            {
                dac_current_vec = _mm256_set_ps(
                    dac(digital_inputs[j], voltage_mode),
                    dac(digital_inputs[j + 1], voltage_mode),
                    dac(digital_inputs[j + 2], voltage_mode),
```

```
        dac(digital_inputs[j + 3], voltage_mode),
        dac(digital_inputs[j + 4], voltage_mode),
        dac(digital_inputs[j + 5], voltage_mode),
        dac(digital_inputs[j + 6], voltage_mode),
        dac(digital_inputs[j + 7], voltage_mode));
    cell_conductance_vec = _mm256_set_ps(
        ptr_arr_cell[i * num_row + j].conductance,
        ptr_arr_cell[i * num_row + j + 1].conductance,
        ptr_arr_cell[i * num_row + j + 2].conductance,
        ptr_arr_cell[i * num_row + j + 3].conductance,
        ptr_arr_cell[i * num_row + j + 4].conductance,
        ptr_arr_cell[i * num_row + j + 5].conductance,
        ptr_arr_cell[i * num_row + j + 6].conductance,
        ptr_arr_cell[i * num_row + j + 7].conductance);
    cell_conductance_vec =
        _mm256_mul_ps(cell_conductance_vec, constant_vec);
    temp_current_sum_vec = _mm256_fmadd_ps(
        dac_current_vec,
        cell_conductance_vec,
        temp_current_sum_vec);
    read_disturb(
        ((float*)&dac_current_vec)[0],
        &ptr_arr_cell[i * num_row + j]);
    read_disturb(
        ((float*)&dac_current_vec)[1],
        &ptr_arr_cell[i * num_row + j + 1]);
    read_disturb(
        ((float*)&dac_current_vec)[2],
        &ptr_arr_cell[i * num_row + j + 2]);
    read_disturb(
        ((float*)&dac_current_vec)[3],
        &ptr_arr_cell[i * num_row + j + 3]);
    read_disturb(
        ((float*)&dac_current_vec)[4],
        &ptr_arr_cell[i * num_row + j + 4]);
    read_disturb(
        ((float*)&dac_current_vec)[5],
        &ptr_arr_cell[i * num_row + j + 5]);
    read_disturb(
        ((float*)&dac_current_vec)[6],
        &ptr_arr_cell[i * num_row + j + 6]);
    read_disturb(
        ((float*)&dac_current_vec)[7],
        &ptr_arr_cell[i * num_row + j + 7]);
```

```
}
else if (num_row - j == 7)
{
    dac_current_vec = _mm256_set_ps(
        dac(digital_inputs[j], voltage_mode),
        dac(digital_inputs[j + 1], voltage_mode),
        dac(digital_inputs[j + 2], voltage_mode),
        dac(digital_inputs[j + 3], voltage_mode),
        dac(digital_inputs[j + 4], voltage_mode),
        dac(digital_inputs[j + 5], voltage_mode),
        dac(digital_inputs[j + 6], voltage_mode),
        0.0f);
    cell_conductance_vec = _mm256_set_ps(
        ptr_arr_cell[i * num_row + j].conductance,
        ptr_arr_cell[i * num_row + j + 1].conductance,
        ptr_arr_cell[i * num_row + j + 2].conductance,
        ptr_arr_cell[i * num_row + j + 3].conductance,
        ptr_arr_cell[i * num_row + j + 4].conductance,
        ptr_arr_cell[i * num_row + j + 5].conductance,
        ptr_arr_cell[i * num_row + j + 6].conductance,
        0.0f);
    cell_conductance_vec =
        _mm256_mul_ps(cell_conductance_vec, constant_vec);
    temp_current_sum_vec = _mm256_fmadd_ps(
        dac_current_vec,
        cell_conductance_vec,
        temp_current_sum_vec);
    read_disturb(
        ((float*)&dac_current_vec)[0],
        &ptr_arr_cell[i * num_row + j]);
    read_disturb(
        ((float*)&dac_current_vec)[1],
        &ptr_arr_cell[i * num_row + j + 1]);
    read_disturb(
        ((float*)&dac_current_vec)[2],
        &ptr_arr_cell[i * num_row + j + 2]);
    read_disturb(
        ((float*)&dac_current_vec)[3],
        &ptr_arr_cell[i * num_row + j + 3]);
    read_disturb(
        ((float*)&dac_current_vec)[4],
        &ptr_arr_cell[i * num_row + j + 4]);
    read_disturb(
        ((float*)&dac_current_vec)[5],
```

```
            &ptr_arr_cell[i * num_row + j + 5]);
        read_disturb(
            ((float*)&dac_current_vec)[6],
            &ptr_arr_cell[i * num_row + j + 6]);
    }
    else if (num_row - j == 6)
    {
        ...
    }
    else if (num_row - j == 5)
    {
        ...
    }
    else if (num_row - j == 4)
    {
        ...
    }
    else if (num_row - j == 3)
    {
        ...
    }
    else if (num_row - j == 2)
    {
        ...
    }
    else
    {
        dac_current_vec = _mm256_set_ps(
            dac(digital_inputs[j], voltage_mode),
            0.0f,
            0.0f,
            0.0f,
            0.0f,
            0.0f,
            0.0f,
            0.0f);
        cell_conductance_vec = _mm256_set_ps(
            ptr_arr_cell[i * num_row + j].conductance,
            0.0f,
            0.0f,
            0.0f,
            0.0f,
            0.0f,
            0.0f,
```

```
                    0.0f);
                cell_conductance_vec =
                    _mm256_mul_ps(cell_conductance_vec, constant_vec);
                temp_current_sum_vec = _mm256_fmadd_ps(
                    dac_current_vec,
                    cell_conductance_vec,
                    temp_current_sum_vec);
                read_disturb(
                    ((float*)&dac_current_vec)[0],
                    &ptr_arr_cell[i * num_row + j]);
            }
        }
        temp_current = ((float*)&temp_current_sum_vec)[0] +
                       ((float*)&temp_current_sum_vec)[1] +
                       ((float*)&temp_current_sum_vec)[2] +
                       ((float*)&temp_current_sum_vec)[3] +
                       ((float*)&temp_current_sum_vec)[4] +
                       ((float*)&temp_current_sum_vec)[5] +
                       ((float*)&temp_current_sum_vec)[6] +
                       ((float*)&temp_current_sum_vec)[7];
        arr_col_sum[i] = adc(temp_current, voltage_mode);
    }
}
```