Computation-in-Memory based on Memristive Devices

Du Nguyen, Hoang Anh

DOI
10.4233/uuid:ba02810b-e380-4c88-a4ed-d6bd2598ab2f

Publication date
2019

Document Version
Final published version

Citation (APA)
Du Nguyen, H. A. (2019). *Computation-in-Memory based on Memristive Devices*. [Dissertation (TU Delft), Delft University of Technology]. https://doi.org/10.4233/uuid:ba02810b-e380-4c88-a4ed-d6bd2598ab2f

COMPUTATION-IN-MEMORY BASED ON MEMRISTIVE DEVICES

COMPUTATION-IN-MEMORY BASED ON MEMRISTIVE DEVICES

Dissertation

for the purpose of obtaining the degree of doctor
at Delft University of Technology,
by the authority of the Rector Magnificus, prof. dr. ir. T.H.J.J. van der Hagen,
chair of the Board for Doctorates,
to be defended publicly on Friday, 13 September 2019 at 10:00 o'clock

by

Hoang Anh DU NGUYEN

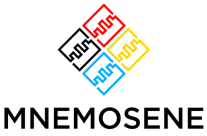
Master of Science in Computer Engineering,
Delft University of Technology, the Netherlands
born in Danang, Vietnam.

This dissertation has been approved by the

promotor: Prof. Dr. Ir. S. Hamdioui
copromotor: Dr. Ir. M. Taouil

Composition of the doctoral committee:

Rector Magnificus,	chairman
Prof. Dr. Ir. S. Hamdioui	Delft University of Technology, promotor
Dr. Ir. M. Taouil	Delft University of Technology, copromotor
Independent members:	
Prof. Dr. Ir. K.L.M. Bertels	Delft University of Technology
Prof. Dr. H. Corporaal	Eindhoven University of Technology
Prof. Dr. F. Catthoor	Katholieke Universiteit Leuven, Belgium
Prof. Dr. M. B. Tahoori	Karlsruhe Institute of Technology, Germany
Dr. R. V. Joshi	T. J. Watson Research Center, IBM, USA
Reserved members:	
Dr. Ir. P. F. A. Van Mieghem	Delft University of Technology



Keywords: Computer architecture, resistive computing, Computation-in-Memory

Cover designed by: Van Sanh Le

ISBN 978-94-6384-060-6

An electronic version of this dissertation is available at
https://doi.org/10.4233/uuid:ba02810b-e380-4c88-a4ed-d6bd2598ab2f

Dedicated to

my father for the motivation to start this journey,
and Nhi An for the courage to end this journey.

CONTENTS

SUMMARY

In-memory computing is a promising computing paradigm due to its capability to alleviate the memory bottleneck. It has even higher potential when implemented using memristive devices or memristors with various beneficial characteristics such as nonvolatility, high scalability, near-zero standby power consumption, high density, and CMOS compatibility. Exploring in-memory computing architectures in the combination with memristor technology is still in its infancy phase. Therefore, it faces challenges with respect to the development of the devices, circuits, architectures, compilers and applications.

This thesis focuses on exploring and developing in-memory computing in terms of architectures (including classification, limited schemes of instruction set, micro-architecture, communication and controller, as well as automation and simulator), and circuits (including logic synthesis flow and interconnect network schemes).

In-Memory architecture classification and survey - We first investigate the state-of-the-art of in-memory computing and propose a classification to have an overview on both existing and unexplored architectures. The classification is based on three main criteria: computation location (i.e., where the results are produced), memory technology (i.e., the memory technology is used), and computation parallelism (i.e., the maximum parallelism level can be exploited). Based on the computation, four main classes are derived: Computation-inside-Memory Array (CIM-A) which produces results *inside* the memory and within the memory *array*, Computation-in-Memory Peripheral (CIM-P) which produces results *inside* the memory and within the *peripheries*, Computation-outside-Memory Near (COM-N) which produces results *outside* the memory and *near* the memory core, and Computation-outside-Memory Far (COM-F) which which produces results *outside* the memory and *far* from the memory core, respectively. Subsequently, we review and compare the four classes and existing architectures quantitatively. The proposed classification and survey show not only the architectures that were explored in details in this dissertation, but also potential architectures that can be explored in the future.

Architecture Level- We propose two architectures representing CIM-A and CIM-P class from the above classification. For CIM-A class, we first propose a concept of integrating computation and memory into one physical device, specifically memristive or memristor devices. This concept has potentials to alleviate the memory wall or memory bottleneck in particular, and the architecture and technology wall in general. We demonstrate the potentials of this concept using a health care application and mathematical application. Thereafter, we use this concept to perform a parallel addition using only the crossbar array which also stores the operands of the function. We show the preliminary result of this parallel adder in comparison with conventional architectures such as

multicore and GPU architecture. Subsequently, we implement the above parallel adder while taking into consideration its controller, communication and interconnect network schemes. Two implementations using two distinct logic designs are compared with a multicore architecture. The results show that the two implementations outperform multicore architecture at least two orders of magnitude in terms of performance, energy, and area combined metrics. In addition, the controller and communication pose relatively large overheads in this architectures. Even though many aspects of this architecture in particular and CIM-A class in general were explored, there are still many questions regarding to the integration between CMOS controller and memristor crossbars, the trade-off between isolation and parallelism, as well as the generation of complex and efficient functional units as building blocks, etc.

For CIM-P class, we first show the potentials of Computation-in-Memory (CIM) core (i.e., a memristor crossbar with capability to perform logical operations using peripheral circuitry). Thereafter, we propose different architectures that can integrate an arithmetic CIM core into different position of the memory hierarchy (i.e., computational cache, main memory or accelerator). Thereafter, we select the architecture that use CIM core as accelerator due to the current state of memristive devices and in-memory computing architecture. Subsequently, we investigate the potentials of this architecture using an analytical model. Finally, we build a simulation platform to port applications executing on the proposed architecture. With this, we verify our assumptions on the analytical model and explore potential applications for the proposed architecture. Both the analytical and simulation results show that the proposed architecture outperforms the conventional architecture at least one order of magnitude in terms of performance and energy. It is worth to notice that the architecture as well as simulation platform are in their infancy stage, and more efforts are required to fully utilize the architecture's potentials on big data applications.

Circuit Level - As it is essential to build basic blocks for in-memory computing architectures, we propose logic synthesis automation tools and interconnect networks to realize digital complex function on memristor crossbar. First we propose a generic synthesis framework to map a digital arithmetic function described in hardware description language (HDL) on memristor circuits. We demonstrate this framework using two case studies of 2-bit counter and 8-bit adder. As this framework is a preliminary result, efforts are still required to automate the framework and explore more complex functions. Thereafter, we propose different interconnect network schemes that can be used in a memristive circuit/design. Using a case study of parallel adder (based on CIM-A class architecture), we demonstrate three schemes including direct scheme using only *copy* operation, indirect scheme using CMOS circuits (i.e., controller), and hybrid scheme which combines the direct and indirect scheme. The results show that hybrid scheme provides the highest performance and lowest energy consumption, hence, should be considered to be used in CIM architecture. It is worth to emphasize that the proposed solution are roughly evaluated and more detail implementations are required to realize these solutions in certain designs/systems.

SAMENVATTING

Gegevensverwerking-in-geheugen is een veelbelovend computerparadigma vanwege de mogelijkheid om het geheugenknelpunt te verlichten. Het heeft een nog hogere potentie wanneer het wordt geïmplementeerd met behulp van geheugenresistieve elementen of geheugenweerstanden met verscheidene voordelige kenmerken zoals niet-vluchtigheid, hoge schaalbaarheid, bijna nul standby-stroomverbruik, hoge dichtheid en CMOS-compatibiliteit. Het verkennen van gegevensverwerking-in-geheugenarchitecturen in combinatie met geheugenweerstandstechnologie bevindt zich nog in de kinderschoenen. Daarom staat het voor uitdagingen met betrekking tot de ontwikkeling van de elementen, schakelingen, architecturen, compilers en applicaties.

Dit proefschrift richt zich op het verkennen en ontwikkelen van gegevensverwerking-in-geheugen op het gebied van architecturen (waaronder classificatie, beperkte schema's van instructieset, micro-architecturen, communicatie en controllers, evenals automatisering en simulators), en schakelingen (waaronder logische-synthesestappenplan en verbindingsnetwerkschema's).

Gegevensverwerking-in-geheugenarchitecturenclassificatie en –overzicht - We onderzoeken eerst de state-of-the-art van gegevensverwarking-in-geheugen en stellen een classificatie voor om een overzicht te hebben van zowel bestaande als niet eerder onderzochte architecturen. De classificatie is gebaseerd op drie hoofdcriteria: berekeningslocatie (d.w.z. waar de resultaten worden geproduceerd), geheugentechnologie (d.w.z. de geheugentechnologie die wordt gebruikt) en berekeningsparallellisme (d.w.z. het maximale parallellisme dat kan worden benut). Op basis van berekening worden vier hoofdklassen afgeleid: Gegevensverwerking-in-Geheugen Array (GiG-A) die resultaten produceert in het geheugen en in de geheugenarray, Gegevensverwerking-in-Geheugen Perifeer (GiG-P) die resultaten produceert in het geheugen en binnen de periferie, Gegevensverwerking-buiten-Geheugen Dichtbij (GbG-D) die resultaten buiten het geheugen en in de buurt van het geheugen produceert, en Gegevensverwerking-buiten-Geheugen Ver (GbG-V) die resultaten buiten het geheugen produceert en ver van het geheugen, respectievelijk. Vervolgens bekijken en vergelijken we de vier klassen en bestaande architecturen kwantitatief. De voorgedragen classificatie en overzicht tonen niet alleen de architecturen die in detail in dit proefschrift zijn onderzocht, maar ook potentiële architecturen die in de toekomst kunnen worden onderzocht.

Architectuurniveau - We stellen twee architecturen voor die de GiG-A- en GiG-P-klasses vertegenwoordigen uit de bovenstaande classificatie. Voor de GiG-A-klasse stellen we eerst een concept voor voor het integreren van zowel berekening als geheugen in één fysiek apparaat, in het bijzonder geheugenresistieve- of geheugenweerstandselementen. Dit concept heeft de potentie om de geheugenmuur of het geheugenknelpunt in

het bijzonder, en de architectuur- en technologiemuur in het algemeen, te verlichten. We demonstreren de mogelijkheden van dit concept met behulp van een toepassing voor de gezondheidszorg en een wiskundige toepassing. Daarna gebruiken we dit concept om een parallelle optelling uit te voeren met alleen de kruisschakelingsarray waarin ook de operanden van de functie worden opgeslagen. We vergelijken het voorlopige resultaat van deze parallelle met conventionele architecturen zoals multicore- en GPU-architecturen. Vervolgens implementeren we de bovengenoemde parallelle opteller, rekening houdend met de controller, communicatie en verbindingsnetwerkschema's. Twee implementaties met twee verschillende logische ontwerpen worden vergeleken met een multicore-architectuur. De resultaten laten zien dat de twee implementaties beter presteren dan een multicore-architectuur op het gebied van prestaties, energie en combinaties van prestatiepunten. Bovendien vormen de controller en communicatie relatief grote overheadkosten in deze architecturen. Hoewel veel aspecten van deze architectuur in het bijzonder voor de CIM-A-klasse ook en in het algemeen werden onderzocht, zijn er nog steeds veel vragen over de integratie tussen CMOS-controller en geheugenweerstandskruisschakelingen, de wisselwerking tussen isolatie en parallellisme, evenals het genereren van complexe en efficiënte functionele eenheden als bouwstenen, enz. Voor de GiG-P-klasse tonen we eerst de mogelijkheden van de Gegevensverwerking-in-Geheugen (GiG)-kern (d.w.z. een geheugenweerstandskruisschakeling met de mogelijkheid om logische bewerkingen uit te voeren met behulp van perifere schakelingen). Daarna stellen we verschillende architecturen voor die een rekenkundige GiG-kern kunnen integreren in verschillende posities van de geheugenhiërarchie (d.w.z. computercache, hoofdgeheugen of versneller). Daarna selecteren we de architectuur die de GiG-kern als versneller gebruikt vanwege de huidige status van geheugenresistieve elementen en GiG-architectuur. Vervolgens onderzoeken we de potentie van deze architectuur met behulp van een analytisch model. Tot slot bouwen we een simulatieplatform om applicaties te porten die op de voorgestelde architectuur worden uitgevoerd. Hiermee verifiëren we onze veronderstellingen over het analytische model en verkennen we mogelijke toepassingen voor de voorgedragen architectuur. Zowel de analytische als de simulatieresultaten tonen aan dat de voorgestelde architectuur beter presteert dan de conventionele architectuur in termen van prestaties en energie met ten minste één orde van grootte. Het is de moeite waard om op te merken dat zowel de architectuur als het simulatieplatform in de kinderschoenen staan en dat er meer werk verricht moet worden om de mogelijkheden van de architectuur voor big data-applicaties volledig te kunnen benutten.

Schakelingsniveau - Omdat het essentieel is om basisblokken te bouwen voor GiG-architecturen, stellen we automatiseringsmiddelen voor logische synthese voor en verbinden we netwerken om de digitale complexe functie op de geheugenweerstandskruisschakeling te realiseren. Eerst stellen we een generiek syntheseplan voor om een digitale rekenkundige functie in hardware description language (HDL) op geheugenweerstandsschakelingen te implementeren. We demonstreren dit stappenplan met behulp van twee casestudy's, een 2-bits teller en een 8-bits opteller. Omdat dit stappenplan een voorlopig resultaat is, zijn er nog inspanningen nodig om het stappenplan te automatiseren en complexere functies te verkennen. Daarna stellen we verschillende ver-

bindingsnetwerkschema's voor die kunnen worden gebruikt in een geheugenresistieve schakeling/ontwerp. Aan de hand van een case study van een parallelle opteller (gebaseerd op de architectuur van de GiG-A-klasse), demonstreren we drie schema's inclusief een direct schema met alleen een kopieerbewerking, een indirect schema met CMOS-schakelingen (d.w.z. een controller) en een hybride schema dat het directe en indirecte schema combineert . De resultaten laten zien dat een hybride schema de hoogste prestaties en het laagste energieverbruik oplevert en daarom moet worden beschouwd als degene om te gebruiken in de GiG-architectuur. Het is de moeite waard om te benadrukken dat de voorgestelde oplossing ruwweg wordt geëvalueerd en dat meer gedetailleerde implementaties nodig zijn om deze oplossingen in bepaalde ontwerpen/systemen te realiseren.

ACKNOWLEDGEMENTS

Growing up in an Eastern culture, I believe that fate leads me to the things that I deserve. Therefore, first, I am thankful for all the opportunities that showed me the way, all the difficulties that diverted my way, all the good that encouraged me, and all the bad that changed me. Furthermore, having a chance to meet and interact with all the people mentioned below is, to my belief, my fortunate fate.

I would like to acknowledge my supervisors Prof. dr. ir. Said Hamdioui, Dr. ir. Mottaqiallah Taouil, and Prof. dr. ir. Koen Bertels. First, I would like to thank Prof. dr. ir. Said Hamdioui, my promotor. To be honest, being your student is challenging, but rewarding. Despite your unmanageable schedule, you always insisted on having our weekly meeting, spending time on brainstorm sessions and correcting my papers thoroughly. Said, thank you for investing efforts in me, both from a research and personal point of view. Moreover, I would like to thank Dr. ir. Mottaqiallah Taouil, my daily supervisor and former office-mate. You are first of all a nice colleague that is always available to discuss existing and non-existing problems, to correct my messy code, to teach me how to write proper code and English, as well as to suffer with me all the 'Friday' paper deadlines. With all your contributions and energy, I am glad that you have been officially my supervisor. Motta, thank you for your time, availability and patience in me, both in research and non-research related matters. Lastly, I would like to acknowledge Prof. dr. ir. Koen Bertels, my former promotor. Koen, thank you for being my first promotor. I remember your encouragements both at the coffee corner and during my evaluation meetings; that gave me courage to continue this path every time it beated me down. In addition, I would like to thank Koen as the previous head of the QCE department for his inspiring talks, generous lady activities, exciting social events, and great team spirit. Different from other PhD lives elsewhere, I am lucky to have a quite open connecting working environment, to share my PhD burdens and to enjoy life "a bit" during my PhD.

I would like to thank my memristor team members, officially, Lei, Jintao, Muath, Uljana and unofficially, Imran, Razvan, Adib, Berna for all the constructive discussions and feedback. We have been gone through our first days with very little knowledge of memristors, to lively discussions on every new idea we had. It has been a long memorable journey to me. With ups and downs, we shared our troubles and joys. I cannot say that I enjoyed all of it, but I am sure the sufferings were less thanks to your presence along the way.

I would like to thank specially my Brazilian research friends Prof. Luigi Carro, Dr. Marco Antonio Zanata Alves, and Paulo Cesar Santos for many long inspiring discussions and support during my hardest time of working on the SiNUCA simulator. Your unconditional supports mean a lot to me and my work.

I would like to thank all of my co-authors Francky Catthoor, Dietmar Fey, Barbareschi Mario, Bosio Alberto, Traiola Marcello, Vatajelu Elena Ioana for all the effort invested in our research. I also would like to thank the committee members for accepting their role, reading this dissertation and providing useful feedback. A special thanks to Daniel for helping me translating the thesis summary, as well as to Lingling and Nga for being my paranymphs.

I would like to thank the supporting staffs of QCE for handling my day-to-day work smoothly. Lidwina and Joyce, thank you for taking care of the paper work, collaborating with me during colloquium organization, and inventing exciting lady activities. Erik and Eef, thank you for the always-available servers, website and work stations.

I would like to thank all my colleagues for their creativity in both scientific and social activities; they are my source of inspiration and joy. Imran, Miki, Pascal, and Guilherme, thank you for jointly organizing the colloquia with me; it is my pleasure to work with you guys. Mafalda, Joost, and Leon, thank you for various interesting social events; we had a great time, especially that time when I tried my first tequila. Anthony, Roel, Berna, Adib, Lei, Razvan, Jintao, Motta, Mohammad, Innocent, Abdullah, Muath, and Abduqader, thank you for being my great office mates; in my moody days, you have been patiently listening to my complaints and responsibly sharing all the challenges with me. Shanshan, Mahroo, Carmina, Hale, Misa, and Lingling, thank you for giving me helpful breaks from my research during the lady activities; it has been enjoyable and powerful to be with you girls. Daniel, Jintao, Troya, Guilherme, Haji, Moritz, and Muath, thank you for good talks and enjoyable lunch time; I wish you guys to have Nature papers soon! Nicoleta, Xiang, Lizhou, Yande, Baozhou, Nauman, Hamid, Cuong, George, and Joost, thank you for the "gezellig" atmosphere in our lab and social events; it is my pleasure to know and talk to you guys.

I am sure that I could not get through the five tough years without my Vietnamese Community in Delft (VCiD). I would like to thank anh Nghi, anh Chi, chi Phuong, anh Bach Duong, anh Hung, anh Thang, and anh Hieu for building a warm and strong community. I would like to thank the members of the online group of Vietnamese ladies in Delft (Hoi chi em) for sharing experiences, plants and good dishes. Vinh - Minh - Dau, chi Huong - anh Dung - be Ngoc, Nhat Anh - Nga, Phan Anh - Ninh - Bo, anh Canh - chi Kim Anh - Ben, Son - Linh - Sumo - Sumi, Dao Tung - Nhung - Mai, and anh Hieu - chi Trang - Gau, Vinh - Diem - Cherry, anh Thao - chi Thao - Nu - Na, anh Phuc - Thao Nguyen, Trang Phan - Eric, chi Lan, Thao Nguyen, Ha, Tinh - Tue, Tin - Huong, Thu, chi Tran, anh Duoc, Thanh Vo, Thien (Alex), Viet, and Vi thank you for all the gatherings and making me feel like I always have a big family to look after me here.

Along the way of gaining a degree, I always had remote supports from my angels (in Vietnamese "quý nhân") to not only overcome difficulties, but also choose a right direction and exploit the best of any situations. I would like to send a special thank to anh Cuong - chi Anh - nha Tom Xiu for many exact advice at the right moment, for your care and con-

cern when seeing anything unusual on my facebook. I would like to also send a special thank to prof. James Peckol for being my remote supervisor and mentor in persuading me to trust my guts and carry on until now. A special thank also to Nghia for always being there, encouraging me, and listening to my problems. Another special thanks to my "USA aids" Trang Le - Tho - Anthony and "Australia aids" Thien throughout the journey. Getting to know and keeping in touch with all of you for the whole time are my blessings.

Last but not least, I would like to express my deepest thank to my big and small family. Mom and Dad, thank you for raising me as an independent girl, always supporting me, and providing me the best condition to pursue this long path. My dear grandmother-in-law and parents-in-law, thank you for always understanding and supporting me through the journey that is completely strange to you. I would like to also thank my sister - Hoang Phuong for taking care of my parents while I am away from home. And a special thank to my sister-in-law - chi Hien and her family for taking care of my parents-in-law while my husband joined me in the Netherlands years ago. Certainly, anh - my best friend and long life partner, thank you for always standing by me, sharing my sorrows and joys and problems, cooking comforting food when I was "always" upset, and never losing hope in me or my PhD. I would like to thank fate, God, or any super power that brings you to my life.

Hoang Anh Du Nguyen Delft, 17 October, 2018

1

INTRODUCTION

Nowadays, most human activities rely on computing systems such as embedded computers, personal computers and servers, in order to communicate, process and store information. Conventional computing systems are based on a load-store architecture that intrinsically suffers from three well-known walls: the memory wall, the power wall and the instruction-level parallelism wall. In the last several decades, the computer performance has been mainly driven by improvements in the technology. However, CMOS technology is reaching its physical -if not economical- limits. Therefore, today's computing systems face challenges in meeting the ever-increasing requirements. In order to solve this problem, novel architectures coupled with emerging technologies are under research as a complement or alternative for future computing systems. In this chapter, we first introduce the motivation behind Computation-in-Memory (CIM) architecture using memristive devices; it is a novel architecture that performs computation inside the resistive memory. Subsequently, we present the opportunities and challenges to develop such an architecture. Thereafter, we briefly describe the research directions of this dissertation, followed by its main contributions. Finally, we outline the remainder of this dissertation.

1

1.1. MOTIVATION

Today's big data and embedded applications have impacted many aspects of human life from health-care to network security [1–3]. These applications do not only require a huge storage and computing capacity, but also high energy efficiency. Therefore, it is essential to build faster, more energy efficient and compact computing systems.

Computing system's performance has been driven by technology scaling for the last several decades [4]. Unfortunately, technology scaling has gradually come to an end and suffers from a lot of problems [5, 6]. These problems can be summarized with the following three walls [7]:

- **Reliability wall** occurs as technology scaling is reaching its physical limits [6], which leads to a reduced life time and increased failure rate [8].
- **Leakage wall** occurs because static power becomes dominant due to the usage of volatile CMOS technology and decreasing threshold voltages [9]; this makes the static power become dominant in the total power consumption.
- **Cost wall** occurs due to the complexity in fabricating and testing new devices; this reduces the economical benefits when commercializing these new devices [10].

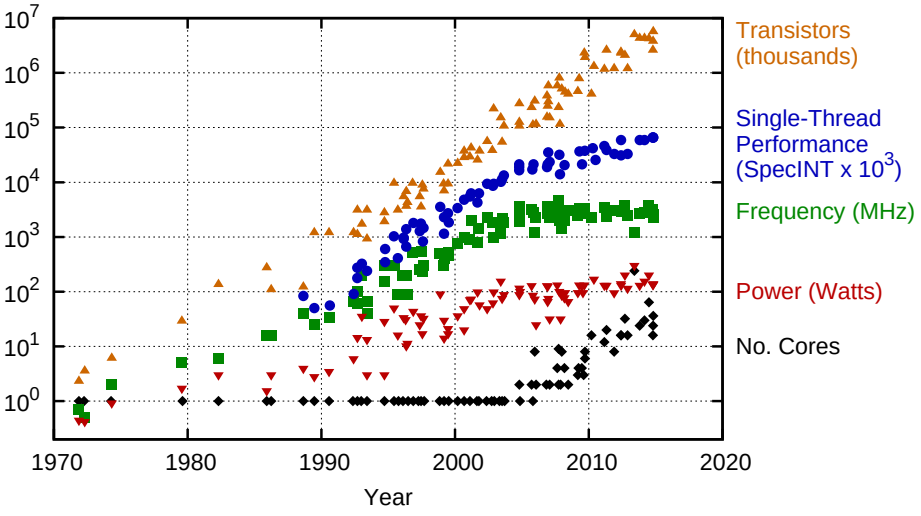


Figure 1.1: Trends of Microprocessors in the Last 40 Years [11, 12]

Meanwhile, existing computing systems are also facing three famous architecture walls [13] that are inherent to von Neumann architectures [14], as shown in Fig. 1.1:

- **Memory wall** occurs due to the different processor and memory speed. As a result, data cannot be efficiently fed to processors through long latency and limited off-chip bandwidth interconnections, especially for multicore processors [5, 15, 16].

- **Power wall** occurs due to the heat generated by high frequency processors, which leads to dark silicon and a saturated computing performance [17, 18].
- **Instruction-level parallelism (ILP) wall** occurs due to the increasing difficulty to extract sufficient parallelism for multicore processors; this leads to a saturating performance as adding more resources will not result in increased performance [17].

All above-mentioned walls have posed difficulties in improving the performance of existing architectures based on existing technologies. Therefore, novel architectures as well technologies are required to address those problems. In-memory computing is a novel computing paradigm that has the potentials to improve the architecture performance for specific applications by integrating processing and storage units in the same physical location using resistive devices [19, 20]. Resistive technology [19, 21, 22] including various resistive devices; each has potentials to be used as a complementary technology to CMOS due to its scalability, high density, nonvolatility, zero leakage power and CMOS compatibility [23–26]. Therefore, in-memory computing based on resistive technology is promising to build high performance and energy efficient computer systems.

1.2. OPPORTUNITIES AND CHALLENGES

This section discusses the opportunities and challenges of developing in-memory computing architectures using resistive devices. Fig. 1.2 shows different aspects that need to be explored in order to implement the new in-memory computing architectures, including device, logic, architecture, compiler and application. Each aspect is discussed next.

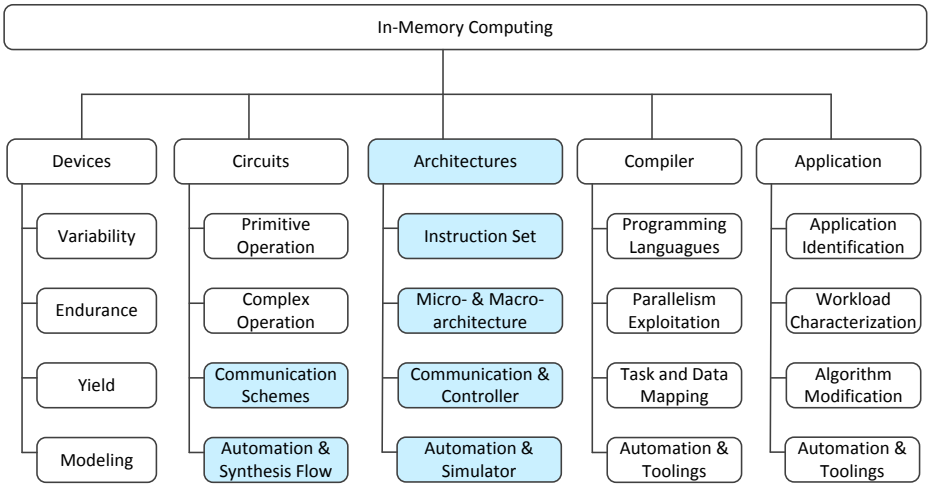


Figure 1.2: Overview of In-Memory Computing

Devices: In-memory computing can be implemented using various technologies ranging from conventional charge-based memories such as DRAM/SRAM/Flash [27–29]

or emerging non charge-based memories [30]. The non charge-based memories can be further divided into different types based on their physical mechanism: resistive [30, 31], "magnetic" memories [30, 32, 33], molecular memories [34–37] or mechanical memories [38, 39], etc. Resistive memories store the data as a resistance value; it includes Resistive RAM (RRAM) [31], phase change memory (PCM) [40], etc. The resistance in RRAM is determined by the presence or absence of a conductive filament between its two electrodes [30], while the resistance in PCM relies on a change between amorphous and crystalline phases [41, 42]. Magnetic memories, such as Magnetic RAM (MRAM), store the data using the magnetization direction of the free layer with respect to the hard or reference layer; it includes, for example, conventional magnetic RAM [43] and STT-MRAM [44, 45]. The resistive and magnetic memories are organized in crossbars with cells placed at each junction. The other types of memories, (i.e., molecular memories, mechanical memories) have not been shown to be useful for computing yet. It is worth mentioning that each of these memory technologies has its own characteristics (read/write latency, endurance, capacity, etc.). Among them, resistive memories can be used effectively for both memories and computation with high scalability, high integration density, and near-zero standby power, etc. [46–48]. Several prototypes of up to 32GB resistive memory have been reported recently [49–51] as shown in Fig. 1.3. However, it also faces challenges in terms of high dynamic write power, endurance, variability, cost and inefficient device modeling [46, 47, 52].

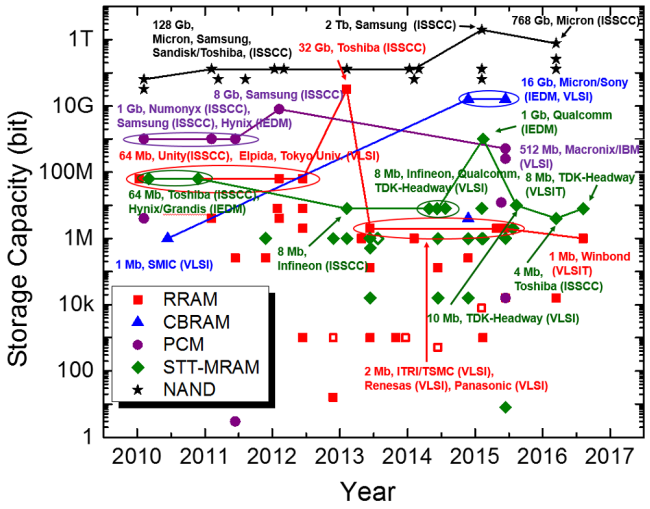


Figure 1.3: Trends of Emerging Device Technologies [53]

Circuits: One important aspect of architectures is logic and circuit design. Efficient logic and circuit design is required to build architectures with a high scalability, high performance and low energy consumption. In-memory computing can perform computations using only resistive cells or hybrid circuits where the resistive cells are used

together with peripheral circuits. Resistive logic circuits that enable in-memory computing in particular have been summarized in recent surveys [54, 55]. However, circuit designs for in-memory computing is still in an early stage; hence, there is a lack of efficient circuit design for logic and arithmetic operations, a proper instruction set, an appropriate interconnect network schemes, as well as a synthesis flow to automate the design process.

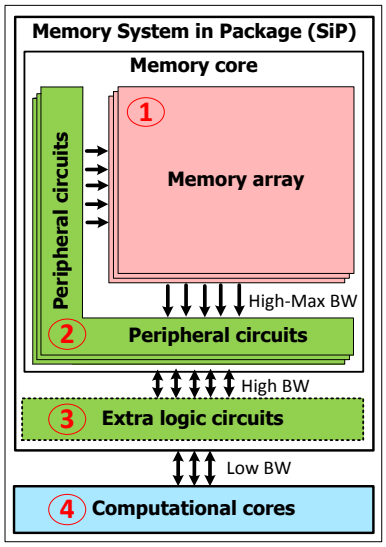


Figure 1.4: Memory-centric Computing

Architectures: The idea of performing computations inside the memory has been proposed since 1970; the authors refer to it as Logic-in-Memory (LIM) [56] and use it for caches. Thereafter, similar concepts were developed using different technologies ranging from embedded DRAM to 3D-stacked DRAM. Different names were used to illustrate the same basic concept such as Processing-in-Memory (PIM) [57, 58], Near-Memory-Computing (NMC) [59], and recently Computation-in-Memory (CIM) [20]. Typically, these architectures perform parts of the operations within or near the memory arrays with the objective to reduce the amount of data movement. By reducing the memory bottleneck, the performance can be improve dramatically; e.g., at least 10x [20, 60] for CIM.

The above architectures can be classified into two groups based on the computation location which is defined as where the results are produced; this includes Computation-in-Memory (CIM) and Computation-Out-Memory (COM). Each of these two groups can be classified further as shown in Fig. 1.4: CIM-A where the computation result is produced inside the memory **a**rray; CIM-P where the computation result is produced in the **p**eripheral circuits of the memory core; COM-N where the computation result is produced in the logic layers located inside the memory system **n**ear the memory core; and COM-F where the computation result is produced outside the memory system **f**ar from

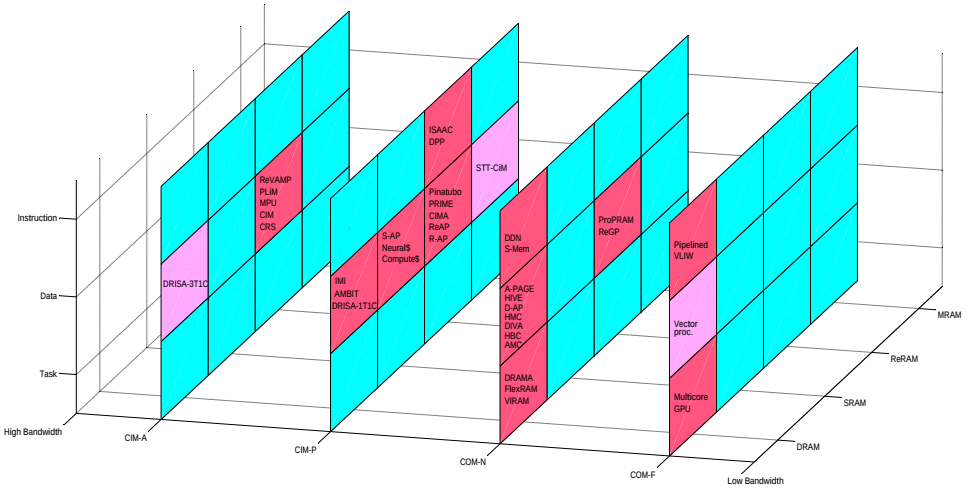


Figure 1.5: Memory-Centric Computing Classification

the memory core, i.e., processors. In order to further differentiate between architectures, the memory technology and computation parallelism are also considered. Examples of memory technologies include conventional charge-based (DRAM, SRAM, Flash, etc.) and emerging non-charge-based memory (resistive RAM, magnetic RAM, molecular memory, etc.). The computation parallelism includes task-level, data-level, and instruction-level. The range of all possible sub-classes using these three criteria and their associated architectures are shown in Fig. 1.5.
As there are many architectures, it is essential to explore various architectures to identify their potentials for memory-intensive applications. In addition, a lot of effort is still required to develop CIM instruction sets, micro- and macro- architectures as well as finding appropriate applications. Moreover, other fundamental components of the architecture still need exploration such as the interconnect network, communication, and controller. Furthermore, performance trade-offs between the architectures is still unknown and must be further explored. Therefore, it is essential to develop an analytical model and a simulation framework to investigate these aspects.

Compilers: As in-memory computing is still in its fancy stage, limited work on programming languages and compilers has been proposed [61, 62]. These articles proposed a programming language based on a domain-specific language and a compiler using arithmetic skeleton as a template to mapping an algorithm onto memristor crossbar, respectively. Further work is required to explore the potentials of in-memory computing, especially in terms of efficient programming to extract parallelism from applications, minimize internal communication between the components of the architecture, and map data intelligently into memory for efficient computation.

Applications: Some applications have been evaluated roughly using in-memory computing with positive results [20, 57, 63]. These applications include big data, memory

intensive problems in some specific fields such as database manipulation, image processing, bio-sequencing, etc. However, it is essential to find and explore appropriate applications that could efficiently make use of CIM architecture and exploit its intrinsic properties. Note that the considered in-memory computing architecture can perform only limited number of specific operations efficiently. Hence, it is worth to explore various applications which have a high percentage of these operations. For example, some previous work has proposed quite a lot of applications consisting of a large number of logical operations such as database processing, graph processing, security encryption, and bio-sequencing [64–69]. In addition, exploring applications requires an automated simulation framework as well as novel algorithms that can be optimized for specific architectures. Hence, it is essential to explore potential algorithms, characterize and tune these algorithms to exploit the potential in-memory architectures.

1.3. RESEARCH TOPICS

Many challenges described in Section 1.2 still need to be addressed. The research carried out in this thesis focuses mostly on the investigation of new architectures that exploit emerging non-volatile memory technology (i.e., resistive RAM). In addition to architectures (including a limited instruction set, micro-architecture, communication and controller, and automation and simulator), it covers a part of circuit design (i.e., logic synthesis and interconnect network) as shown by the colored boxes in Fig. 1.2.

- **Exploration of In-Memory Computing Architectures:** As in-memory computing is emerging due to new available memory technologies, it is essential to understand the concept of Computation-in-Memory (CIM), define the space of in-memory computing and classify it. In this thesis, we first explore the scope of in-memory computing in terms of devices, circuits and architectures. Thereafter, we identify classification metrics to determine the complete space of in-memory computing and define the complete space of existing and possible future architectures.

- **Architecture Level:** Based on the above explored space, we select two architectures to be further explore, analyze their pros and cons as well as their potentials in dealing with data-intensive applications. For each architecture, we implement different case study and investigate the instruction set, communication and controller as well as the interconnect overhead of these implementation to show their potentials and limitations. We also investigate an analytical model and simulator to explore the performance of this architecture for different applications.

- **Circuit Level:** In order to build the above architecture, it is essential to design different primitive functions. Hence, a logic synthesis framework is required to accelerate the design process. In this thesis, we investigate a synthesis framework that synthesizes logic functions using resistive circuits; thereafter, these circuits can be used in the in-memory computing architectures to perform operations within the memory. In order to build the above architecture, an interconnect network is required to connect multiple primitive functions and blocks. Therefore, we also explore various interconnect networks and communication schemes.

1

1.4. CONTRIBUTIONS

The contributions of this dissertation are directly related to the research topics presented in the previous section.

1.4.1. EXPLORATION OF IN-MEMORY COMPUTING ARCHITECTURES

We study the existing in-memory architectures and propose a classification that includes both the conventional and future emerging architectures. With respect to this research topic, the main contributions are as follows:

- A classification of memory-centric computing architectures that is based on three metrics [70]: computation location, memory technology and computation parallelism. The computation location indicates where the computations are performed (e.g., near or far from the memory) and provides an insight regarding the severeness of the memory wall. The memory technology, which provides characteristics of the memory, can enable new computer architectures (e.g., resistive computing). The computation parallelisms specifies the type of parallelism that can be exploited in an architecture (e.g. task level parallelism). With these distinct metrics, the classification shows four main classes based on the first metric: Computation-in-Memory Array (CIM-A), Computation-in-Memory Peripheral (CIM-P), Computation-out-Memory Near(COM-N), Computation-out-Memory Far (COM-F).

- A survey of existing memory-centric computing architectures [71]; it reviews more than 30 architectures in the context of the four above-mentioned classes. In addition, we present a qualitative comparison of the four main classes, and the pros and cons of the existing architectures.

1.4.2. ARCHITECTURE LEVEL

We investigate the feasibility and evaluate the performance of two in-memory architectures. Based on the above classification, we focus on two in-memory architectures: CIM which is a CIM-A architecture and CIMX which is a CIM-P architecture. With respect to this research topic, the main contributions are as follows:

1. Computation-in-Memory (CIM) architecture

   - A CIM architecture that interweaves computation and storage into a physical non-volatile memory crossbar [20]. The memory crossbar consists of memristive devices placed at each horizontal and vertical nanowire junction. A control and communication block applies voltages to these horizontal and vertical nanowire to perform useful operations.
   - A CIM Parallel Adder that maps a mathematical function (i.e., parallel addition) on the memristor crossbar [72, 73]. This mapping is evaluated using a simplified analytical model and compared against two conventional architectures (i.e., multicore and GPU). The potential performance, energy and area of this mapping shows approximately an improvement of two orders of magnitude with respect to the other two architectures. We propose

two implementations based on two distinct memristive logic designs (i.e., Boolean and implication logic) to realize the above CIM parallel adder. Both implementations consist of a memristor crossbar, controller and communication network. They are evaluated using an analytical model and compared against a multicore architecture. The results show that the two implementations outperform the multicore architecture with two orders of magnitude in terms of performance, energy and area.

2. Computation-in-Memory Accelerator (CIMX)

   - A CIM core design [74] based on scouting logic [75]. Moreover, the potential characteristics and applications of CIM core is roughly explored. In addition, the CIM core's performance is analyzed in comparison with conventional multicore.
   - A CIMX architecture using CIM core [76]. Several different architectures using CIMX at different memory hierarchy are proposed, as well as their pros and cons are discussed. Based on the discussion, we selected a promising architecture where a CIM core accelerator is added to a conventional architecture. An analytical model is also proposed to estimate the performance of the proposed architecture. The results are compared against the conventional part of the architecture, i.e. the conventional architecture without the CIM core. In order to estimate the performance based on applications, we propose a simulation framework to explore appropriate applications that can benefit from the proposed architecture. The simulation framework is used for both the conventional and proposed in-memory architectures. Both analytical and simulation results show that the proposed architecture obtains at least one order of magnitude improvements in terms of performance and energy.

1.4.3. CIRCUIT LEVEL

At circuit level, we propose automation tools to generate basic functional units and interconnect network schemes to connect these basic functional units. With respect to this research topic, the main contribution is as follows:

- A synthesis framework [77]; it uses the memristive design methods to map a logic circuit described in HDL to memristor circuits (i.e., including both memristor crossbar and discrete memristors). Thereafter, we validate the framework using two case studies: a 2-bit counter and 8-bit adder.

- Three different interconnect network schemes to support communication between functional components within or between resistive crossbars [78]. The first scheme utilizes the primitive *copy* operation [79] to perform communication directly inside the memristor crossbar. The second scheme uses the CMOS circuits (i.e., controller) outside the memristor crossbar to perform communication by reading out a value from the source memristor and writing this value back to the destination

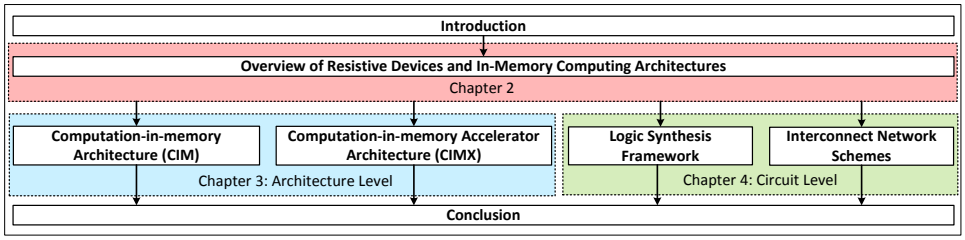


Figure 1.6: Thesis Outline

memristor. The third scheme combines the two above schemes and is called a hybrid scheme. We evaluate the proposed schemes using the CIM parallel adder case study.

1.5. THESIS ORGANIZATION

The remainder of this thesis is illustrated in Fig. 1.6 and described next.

Chapter 2 presents the overview of resistive devices and state-of-the-art of in-memory computing. First, it presents the background of resistive devices (e.g., history and working principles), and their applications for memories, logic designs and computing architectures. Thereafter, it discusses the contributions of this dissertation with respect to the classification and survey of in-memory computing architectures.

Chapter 3 discusses the contributions of this dissertation with respect to the architecture level. First, it discusses the CIM architecture that perform parallel addition as a case study. Thereafter, it discusses the CIM accelerator (CIMX) architecture, its associated analytical model and simulation framework.

Chapter 4 discusses the contributions of this dissertation with respect to the logic level. It first proposes a generic synthesis framework that can exploit different memristive logic design methodologies. Thereafter, it shows two case studies using the Boolean logic circuit design method [80]. Thereafter, it shows interconnect network schemes to provide communication between function components inside the in-memory architecture.

Chapter 5 concludes this dissertation and shows possible future research directions.

2

OVERVIEW AND CLASSIFICATION

This chapter discusses the fundamentals of the memristive device and in-memory computing architecture. In terms of memristive device background, it discusses memristive devices, circuits and architectures. First, it briefly presents the history, working principles, and major properties of memristive devices. Thereafter, it discusses the potential applications of memristive devices in the following domains: non-volatile memory, logic design, and computing architecture. In terms of in-memory computing architectures, it presents a classification of memory-centric computing architectures and a survey based on this classification. First, it proposes a classification based on three metrics: computation location, memory technology and computation parallelism. The classification shows a complete space exploration of memory-centric architectures including existing and potential future architectures; therefore, it shows the position of in-memory computing in the whole architecture space. Thereafter, it reviews existing architectures quantitatively, compares among four main classes based on computation location, as well as discusses their pros and cons.

The content of this chapter is based on the following research article:

1. **H.A. Du Nguyen**, J. Yu, L. Xie, M. Taouil, S. Hamdioui, D. Fey, *Memristive Devices for Computing: Beyond CMOS and Beyond von Neumann*, IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC), Abu Dhabi, UAE, October, 2017, pp. 1-10
2. **H.A. Du Nguyen**, J. Yu, M. Abu Lebdeh, M. Taouil, S. Hamdioui, F. Catthoor, *A Classification of In-Memory Computing*, under review.
3. **H.A. Du Nguyen**, J. Yu, M. Abu Lebdeh, M. Taouil, S. Hamdioui, F. Catthoor, *A Survey of In-Memory Computing*, to be submitted.

2.1. PROBLEM STATEMENT

Today's computer architectures suffer from many challenges both at technology level and architecture level. As a consequence, existing architectures are unable to deal with emerging big data applications. At the technology level, many emerging technologies are currently explored to find a substitution for CMOS devices. Among them, memristive devices show many promising characteristics such as nonvolatility, zero static power consumption, small footprint, and CMOS compatibility. At architecture level, many architectures are proposed to solve the limitations of von-Neumann architectures in terms of memory bottleneck, power constraint and instruction-level parallelism exploitation. Therefore, two questions are still open: (1) what are the potential capabilities of memristive devices for a new non von-Neuman architecture, (2) what is the potential space to be explored in memory-centric computing architectures? This chapter focuses on these two questions.

Exploration of memristive device potentials: it is essential to comprehensively explore the potential of memristive devices in building logic functions, memories, arithmetic operations, and novel computer architectures. Especially, the unique properties of memristor devices are investigated to be applied in the concept of neuromorphic and emerging computation-in-memory architecture.

Exploration of in-memory computing architectures: it is essential to comprehensively explore the complete space of computing architectures using the memory-centric approach. First, the memory-centric computing architectures are classified so that a complete space can be explored. Second, the existing architectures are placed into this classification; with this overview, potential architectures are identified and further explored.

2.2. MAIN CONTRIBUTIONS

The main contributions in the above aspects are as follows.

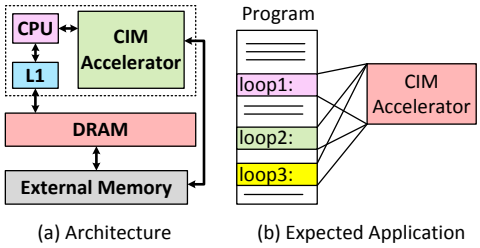


Figure 2.1: CIM-based Architecture

Exploration of memristive device potentials [54]: Memristive device, better known as memristor, is the fourth fundamental two-terminal element, next to the resistor, capacitor, and inductor. It was initially proposed in 1971 by the circuit theorist Leon Chua [81]. Memristive device became renowned in 2008 when the first physical memristor device was fabricated by HP Lab [21]. Memristive devices can be used for logic functions,

memories, arithmetic operations, and novel computer architectures. As I mainly contributes to the novel computer architectures, specifically the computation-in-memory architecture, the corresponding contribution is explained as follows.

In terms of novel computer architectures, two emerging resistive computing architecture including Computation-in-Memory (CIM) and neuromorphic processing are described. The CIM-based architecture consists of a conventional processor, caches, CIM accelerator, main memory DRAM and external memory (as shown in Fig. 2.1(a)). CIM accelerator is beneficial to a program as shown in Fig. 2.1(b). In this case, multiple invoked loops work on the same large datasets; obviously the data should be initialized on the CIM accelerator. Each time the loop is invoked, the processor sends a request to the CIM accelerator; the latter, performs the requested operations and returns the results to the processor. Examples of such applications are database applications, where multiple queries (each consisting of large loops) are applied to a fixed database. These queries are used to look for specific data patterns in the database.

Exploration of in-memory computing architectures [70, 71]: We present a memory-centric computing classification based on three metrics: computation location, memory technology, and computation parallelism [70]. The computation location indicates where the computation results are produced (e.g., near or far from the memory) and provides an insight regarding the severeness of the memory wall. The memory technology is a fundamental component in enabling new computer architectures (e.g., resistive computing). The computation parallelisms specifies the type of parallelism that can be exploited in an architecture (e.g., task level parallelism). With these distinct metrics, the classification covers *all* computing architectures in general and memory-centric computing in specific. Among them, in-memory computing architectures play a major role. Next, we will explain the classification metrics in detail.

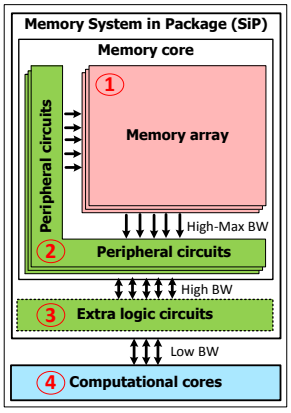


Figure 2.2: Memory-centric Computing

A computer architecture or system consists of one or more memories and computational units as shown in Fig. 2.2. The memories is the main storage unit; it can include only memory core with memory arrays and its supporting peripheral circuits, or mem-

ory core with extra logic circuits, which is called memory System-in-Packages (SiP). The computations is performed traditionally using computation cores, however, they can also be performed using extra logic circuits, peripheral circuits and memory array of the memory SiP.

In case computations take place *inside* the memory core, depending on *where* the result of the computation is produced, an architecture can be placed into two classes:

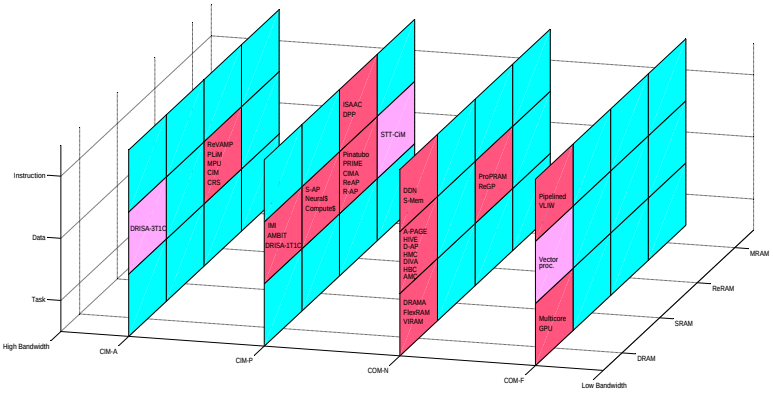


Figure 2.3: Memory-centric Computing Classification

- *CIM-Array (CIM-A):* In CIM-A, the computing result is produced within the array. Examples of architectures that fit within this class are PLiM [82], ReVAMP [83], MPU [84], CIM device [85], etc. The CIM-A core typically requires a significant redesign of the memory array to support computing, as conventional memory cell layout and their embedding in the bit and wordline structure may not allow them to be used for computation-in-memory. In addition, modifications in the periphery are sometimes needed to support the changes in the cell changes. Therefore, CIM-A architectures can be sub-divided into two groups: (1) basic CIM-A where only changes inside the memory array are required, and (2) hybrid CIM-A where in addition to major changes in the memory array also minimal to medium changes are required in the peripheral circuit.

- *CIM-Periphery (CIM-P):* In a CIM-P, the computing result is produced within the peripheral circuitry. Examples of proposal architectures that fit in this class are PRIME [86], Pinatubo[64], CIM-Accelerator [54], etc. This architecture typically focuses on special circuits in the peripheral circuit to realize e.g., bit-wise logic operations [64, 75], matrix-vector multiplication exploiting Ohm's law [87], etc. Even though the computational results are produced in the peripheral circuits for CIM-P, the memory array could be a significant component in the computations. For example, when multiple rows are activated simultaneously in the array, different logic [64, 75] and arithmetic operations [84, 88] can be realized in the periphery. As the peripheral circuits are modified, the currents/voltages applied to the memory array are typically different than in the conventional memory. Hence, similarly

as to the CIM-A sub-classes, the CIM-P architectures are also further divided into two groups: (1) basic CIM-P where only changes inside the peripheral is required, which means the current levels should not be affected, and (2) hybrid CIM-P where the majority of the changes take place in the peripheral circuit and minimal to medium changes in the memory array.

For computations take place *outside* the memory core, computations take either place in the extra logic circuits inside the memory SiP (3) or in the traditional computational cores (4) such as CPU, FPGA, etc. In case of the former, the computations take place near the memory core and the architecture is referred to as Computation-Outside-Memory Near (COM-N). In case of the latter, the architecture is referred to as Computation-Outside-Memory Far (COM-F).

The existing architectures are classified based on the above discussed metrics; the result is shown in Fig. 2.3. The classification contains 48 categories. Some categories, the ones located in *red* planes, show that a *lot* of work has been done for that particular class. For the categories in the *pink* planes, a *moderate number* of work has been done. To our best knowledge, no architectures exists in the *blue* planes; these fields are currently unexplored as they received no attention yet from the research community or non-existing due to current restrictions of the technology. We also present a survey on the existing architectures and evaluates these architectures quantitatively [71].

Embedded Tutorial

2

Memristive Devices for Computing: Beyond CMOS and Beyond von Neumann

H.A. Du Nguyen*, Jintao Yu*, Lei Xie*, Mottaqiallah Taouil*, Said Hamdioui*, Dietmar Fey†
*Computer Engineering, Delft University of Technology, Delft, the Netherlands
S.Hamdioui@tudelft.nl
† Computer Architecture, University of Erlangen-Nrnberg, Erlangen, Germany
dietmar.fey@informatik.uni-erlangen.de

***Abstract*—Traditional CMOS technology and its continuous down-scaling have been the driving force to improve performance of existing computer architectures. Today, however, both technology and computer architectures are facing challenges that make them incapable of delivering the growing computing performance requirement at pre-defined constraints. This forces the exploration of both novel architectures and technologies; not only to maintain the economic profit of technology scaling, but also to enable the computing architecture solutions for big-data and data-intensive applications. This paper discusses the emerging memristive device as a complement (or an alternative) to CMOS devices and shows how such devices enable novel computing paradigms that will solve the challenges of today's architectures for certain applications. The paper covers not only the potential of memristor devices in enabling novel memory technologies, logic design styles, and arithmetic operations, but also their potential in enabling in-memory computing and neuromorphic computing.**

I. Introduction

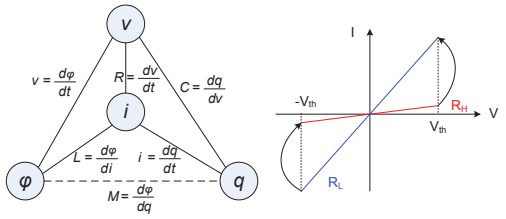
Today's and emerging applications including internet-of-things (IoT) and big data applications are extremely demanding in terms of storage and computing performance. Such world-changing applications will not only impact all aspects of our daily life, but also change a lot in the IC and computer manufacture industry. Emerging applications require computing performance which was typical of supercomputers a few years ago, but with constraints on size, power consumption and guaranteed response time which are typical of the embedded applications [1,2]. Both current device technologies and computer architectures are encountering significant challenges that make them incapable of providing the required functionalities and properties.

Nanoscale CMOS technology is facing three walls [2]: (1) the reliability wall as technology scaling leads to increased failure rate and reduced device lifetime [2], (2) the leakage wall as static power dominates and might be even larger than dynamic power at more advanced technology nodes (due to volatile technology and decreasing supply voltage) [3]; (3) the cost wall as the cost per transistor via pure geometric scaling of process technology is plateauing [4]. These walls have led to the slowdown of the CMOS scaling. On top of that, today's computer architectures are facing the three well-known walls [5]: (1) the memory wall due to the growing gap between processor and memory speeds, and the limited memory bandwidth making the memory access as the killer of performance and energy consumption for data-intensive applications; e.g. big-data; (2) the Instruction Level parallelism (ILP) wall due to the complexity of extracting sufficient parallelism to keep all cores running; (3) the power wall as the practical power limit for cooling is reached, which leads to no further increase of CPU clock frequency. In order for computing systems to continue delivering required performance and sustaining profits for the near future, alternative computing architectures have to be explored in the light of emerging device technologies. Resistive computing, neuromorphic computing and quantum computing are some candidates for the next-generation computing paradigms, while memristor devices, quantum dots, spin-wave devices are couple of emerging device technologies [6]. Among these technologies, memristor is a promising candidate to complement and/or replace traditional CMOS (at least for some applications) due to many advantages such as near-zero standby power, high device scalability, high integration density, and CMOS process compatibility [7,8]. Therefore, it provides significant potential to implement high density memories [9–11], different logic design styles [12–16], and consequently enabling new computing paradigms [17–21].

This paper will comprehensively explore the potential of memristors in building logic functions, memories, arithmetic operations, and novel computer architectures. Section I briefly describes the history and characteristics of memristive devices. Section II and III overview the logic design styles and non-volatile memories based on memristive devices, respectively. Section IV shows how the unique properties of memristor devices enable the concept of neuromorphic and emerging computation-in-memory architecture. Section V highlights the major challenges for memristive device based computing, followed by a conclusion of this paper.

II. Memristive Devices: What Are They?

Memristive device, better known as memristor, is the fourth fundamental two-terminal element, next to the resistor, capacitor, and inductor. It was initially proposed in 1971 by the circuit theorist Leon Chua [22]. He noticed that there was still a missing relationship between flux and charge as shown by the dashed line in Fig. 1(a). Theoretically, a memristive device is a



(a) The four fundamental elements (b) Pinched hysteresis loop

Fig. 1: Stateful Logic

passive element that maintains a relationship between the time integrals of current and voltage across a two-terminal element, while considering the internal state variable of the device. Hence, a memristor can be expressed either by a function of charge q or flux ϕ. An important fingerprint of a memristor is the pinched hysteresis loop current-voltage characteristic as illustrated in Fig. 1(b). It shows that memristive devices have two stable states: high R_{H} and low R_{L} resistive states. When the voltage across the memristive device is greater than the absolute value of its threshold voltage (i.e., V_{th}), then it switches from one resistive state to another. Secondly, it has the ability to remember its history (i.e., the internal state).

After a silent period of more than 30 years, memristive device became renowned in 2008 when the first physical memristor device was fabricated by HP Lab [23]. HP built a metal-insulator-metal device using a titanium oxide as a insulator sandwiched by two metal electrodes. They successfully identified the memristive behaviour over its two-terminal node as described by Leon Chua. The device tunes its resistance by controlling positive charged oxygen vacancies in the insulator layer by applying different voltages. After the first memristive device was manufactured, many memristor devices based on different type of materials have been proposed such as HfO_x, TaO_x, SiO_x [7,8].

III. MEMRISTIVE DEVICES FOR LOGIC

This section first classifies existing memristor-based logic design styles. Thereafter, it briefly describes examples of each class. Finally, it qualitatively compares them.

A. Classification

Multiple logic design styles have been proposed [12–16,24–27]. We divide them into several classes using the following criteria:

- **Input Data Representation** indicates whether the input data is represented by a voltage or resistance.
- **Output Data Representation** indicates whether the output data is represented by a voltage or resistance.
- **Processing Elements** indicates whether the data is *processed* based on memristors only or by using a hybrid cmos/memristor combination. Obviously the *control* of the memristors is always done using CMOS circuits.

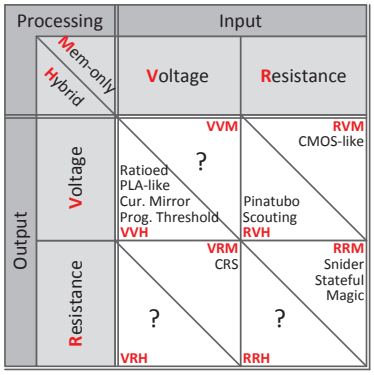


Fig. 2: Classification of Memristor-Based Logic Design Styles.

Fig. 2 shows the classification result; there are eight classes in total. Each class is named based on the input and output representation signals, and the processing element. For instance, scouting logic is located in the RVH class where R indicates the input data representation, V the output data representation and H hybrid CMOS/memristor processing. The classification clearly shows that the existing logic designs fit in five defined classes, and that three classes are potentially not explored yet.

- **VVH**: Memristor ratioed logic [24], PLA-like [12], current mirror based threshold logic [13], and programmable threshold logic [25] belong to this class. They use a voltage to represent both input and output data and CMOS gates (e.g., inverter [12,13,24] and D Flip-Flop [25]) as a threshold function (and inverter). The memristors are used as either configuration switches [12,24] or input weights [13,25].
- **RVH**: Pinatubo [28] and Scouting logic [27] are the work published in this class. They use a resistance to represent the input data and a voltage to represent the output data. Both logic styles perform logic operations by modifying memory read operations.
- **RVM**: CMOS-like logic [26] is the only existing work in this class. It uses a resistance to represent the input data and a voltage to represent the output data. It replaces MOSFETs in the pull-up and -down network of the conventional CMOS logic with memristors.
- **VRM**: Complementary Resistive Switching (CRS) logic [14] is the only published work in this class. It uses a voltage to represent the input data and a resistance to represent the output data. CRS logic performs logic operations by modifying memory write operations. In addition, You et al. extended the existing CRS logic gates with other Boolean logic gates which requires also fewer execution steps [29].
- **RRM** Snider [15] and stateful [16] logic belong to this class. They use a resistance to represent both the input and output data. They perform logic operations by using memristors as voltage dividers which conditionally switch

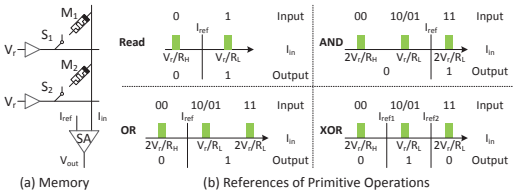


Fig. 3: Scouting Logic

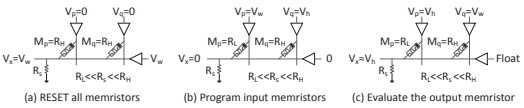


Fig. 4: Stateful Logic

the output memristors. Lehtonen et al. [30] extended stateful logic to support more types of logic operations (e.g., AND-IMP and OR-IMP). Kvatinsky et al. [31] and Xie et al. [32] extended Snider logic to support more types of logic operations (e.g., AND and OR).

In the remainder of this section, the working principle of two logic design styles will be given as examples since they are the most popular candidates to implement resistive computing systems. Finally, a comparison between the state-of-the-art will be provided.

B. RVH: Scouting Logic

As Pinatubo and scouting logic share the same idea, we use scouting logic as an example using different circuit implementations. Scouting logic [27] supports the AND, OR and XOR logic operations. Scouting logic uses resistances R_H and R_L to represent its logic inputs 0 and 1, respectively; it uses voltages V_{dd} and *GND* to represent its logic output 1 and 0, respectively.

Scouting logic is inspired by memory read operations. Typically when a cell is read, say Memristor M_1 of Fig. 3(a), a read voltage V_r is applied to its row and the switch S_1 is activated. Subsequently, a current I_{in} will flow through the bit line to the input of the sense amplifier (SA). This current is compared to the reference current I_{ref}. If I_{in} is greater than I_{ref} (i.e., when M_1 is R_L state), the output of the SA changes to V_{dd} (logic 1). Similarly, when M_1 is R_H state, I_{in}<I_{ref} and subsequently the output changes to logic 0. For proper operations, I_{ref} should be fixed between high and low currents of Fig. 3(b). Instead of reading a single memristor at a time, scouting logic activates the two inputs of the gate simultaneously (e.g., M_1 and M_2 in Fig. 3(a)). As a result, the input current to the sense amplifier is determined by the equivalent input resistance ($M_1//M_2$). This resistance results in three possible values: $\frac{R_L}{2}$, $\frac{R_H}{2}$ and $R_L//R_H \approx R_L$. Hence, the input current I_{in} can have only three values. By changing the value of I_{ref} different gates can be realized.

For example, to implement an OR gate I_{ref} should be set between $\frac{2V_r}{R_H}$ and $\frac{V_r}{R_L}$ as depicted in Fig. 3(b)). When the inputs are $p = 0$ and $q = 1$, the input current I_{in} to the sense amplifier is around $\frac{V_r}{R_L}$. As $\frac{2V_r}{R_H}$<I_{ref}<$\frac{V_r}{R_L}$, $I_{in} > I_{ref}$ and the output voltage V_{out} is V_{dd}. The AND and XOR operations work in a similar way. Note that the XOR gate needs two references which is not shown in Fig. 3(a). More details on the sense amplifier can be found in [27].

C. RRM: Stateful Logic

Stateful logic [16] supports material implication (IMP) as primitive logic operation. The IMP operation is denoted by Eq. 1.

$$\text{IMP: } q' = p \rightarrow q = \bar{p} + q \tag{1}$$

Here p and q are inputs while q' is the output. Stateful logic uses R_H and R_L represent logic 0 and 1, respectively; both for the inputs and outputs. An IMP gate consists of two memristors (i.e., M_p and M_q) and a resistor R_s ($R_L \ll R_s \ll R_H$). M_p is only used for the input p while M_q is used both for the input q and output q'. To perform the operation, control voltages V_h and V_w are applied to M_p and M_q, respectively; the control voltages typically satisfy the relationship: $0<V_h=\frac{V_w}{2}<V_{th}<V_w<2V_{th}$.

To illustrate the working principle of stateful logic, an example of an IMP gate is given for the inputs $p = 1$ and $q = 0$, as shown in Fig. 4. It consists of three steps. First, all the memristors are reset to R_H by applying voltages $V_p = V_q =$*GND* and $V_x = V_w$ (see Fig. 4(a)). Second, M_p is programmed to R_L ($p = 1$) by applying voltages $V_p = V_w$, $V_q = V_h$ and $V_x = 0$ (see Fig. 4(b)). V_h is used to prevent M_q from undesired switching. Finally, the IMP gate is evaluated by applying voltages $V_p = V_h$, $V_q = V_w$ and keeping the row floating (see Fig. 4(c)). Therefore, $V_x \approx V_h$ ($R_L \ll R_s \ll R_H$) and the voltage across M_q is $V_q - V_x \approx V_w - V_h < V_{th}$. As a result, M_q stays in R_H. More details and the latest progress can be found in [16,30,33].

D. Comparison

We use the following metrics to qualitatively compare the existing memristor logic design styles.

- **Array Compatibility** indicates whether the logic style is compatible with normal 1R and/or 1T1R memory arrays or not.
- **CMOS Controller Requirement** indicates whether the logic style needs a CMOS circuit to control it or not.
- **Nonvolatility** indicates whether the logic style can store the data when it is powered off or not.
- **Area** indicates how area-efficient the logic style is to perform operations.
- **Speed** indicates how fast the logic style is to perform operations.
- **Energy Consumption** indicates how energy-efficient the logic style is to perform operations.
- **Scalability** indicates how well the logic style can be scaled to implement more complex circuits.

TABLE I: Comparison Between Existing Logic Styles

Style	Class	Array	Control	NV	Speed	Area	Energy	Scalability	Robustness
Memristor ratioed logic	VVH	No	No	No	+	++	++	++	+
PLA-like memristor logic	VVH	No	No	No	+	++	++	++	++
Current mirror threshold logic	VVH	No	No	No	+	++	++	++	++
Programmable threshold logic	VVH	No	No	No	+	++	++	++	++
Pinatubo / Scouting logic	RVH	Yes	Yes	Yes	+	+	++	+	+
CMOS-like logic	RVM	No	Yes	Yes	-	-	-	-	+
CRS logic	VRM	Yes	Yes	Yes	-	-	-	-	-
Snider logic	RRM	Yes	Yes	Yes	-	-	-	-	-
Stateful logic	RRM	Yes	Yes	Yes	-	-	-	-	-

- **Robustness** indicates how robust the logic style is to be resilient against the unrelaible CMOS and memristor technology.

Table I shows the comparison result. We can draw the following conclusions with respect to the metrics.

- **Array Compatibility**: Design styles of RVH, VRM and RRM are compatible with memory arrays. CMOS like memristor logic is not compatible with memory arrays due to its irregular topology. Design styles of VVH are not compatible with 1R/1T1R array as they need to add CMOS inverters or D flip-flops to memory arrays. Note that array compatibility is an important requirement to implement resistive computing systems.
- **CMOS Controller Requirement**: The logic styles of VVH do not need additional CMOS control units as their inputs and outputs are voltage based. In contrast, other logic styles need to transduce the data between voltages and resistances, and also need the controller to control each step during execution. Note that several logic design styles require multiple execution steps.
- **Nonvolatility**: Only the design styles of VVH are volatile, as both their inputs and outputs are represented by voltages. In contrast, other logic styles have their input and/or output represented by resistances, and thus are nonvolatile.
- **Speed**: The design styles of VVH and RVH are faster as they can finish logic operations in a single step. In contrast, other logic design styles are slower as they need multiple steps.
- **Area**: Design styles of VVH require smaller area as they do not need CMOS controllers. In contrast, other design styles require larger area as they need CMOS controllers. In addition, Pinatubo/Scouting logic needs a simpler controller as it only needs a single step instead of multiple [27].
- **Energy Consumption**: Three main factors impact on the energy consumption; they are controller necessity, nonvolatiltiy and speed. Design styles of VVH do not need CMOS controllers and they are fast, and hence they are likely not to consume a lot energy to perform logic operations. Design styles of RVH are nonvolatile and fast, and hence they are likely to consume less energy to perform logic operations. In contrast, the other design styles possibly need more energy as they need complex controllers and longer time to perform logic operations.
- **Scalability**: Controller necessity impacts on the scalability. Design styles of VVH are the easiest to scale up as they do not need CMOS controllers. Design styles of RVH are easier to scale as they need a simpler controller. In contrast, the other design styles are hard to scale up as they need complex controllers.
- **Robustness**: Controller necessity impacts on the robustness as many transistors are involved by controllers. In addition, design styles are more reliable if the memristors do not need to switch during logic operations. This is because memristor devices suffer from cycle-to-cycle variation [2]. Except memristor ratioed logic, design styles of VVH are likely to be most robust as they need no CMOS controllers and memristor switching. Design styles of RVH and RVM are more reliable than other styles as they do not need to switch memristors during logic operations.

Overall, in order to implement the resistive computing architectures, design styles of RVM , VRM, and RRM are very suitable due to their array compatibility. Among them, scouting logic is the most promising candidate due to its good performance in the remaining aspects. In addition, the design styles of VVH and RVM are possible alternatives to replace CMOS logic.

IV. MEMRISTIVE DEVICES FOR MEMORIES

Many non-volatile memory elements have been proposed such as phase-change-memories (PCMs), spin-torque-transfer magnetic RAMs (STT-MRAMs), and resistive RAMs (ReRAMs). A very good introduction into the the topic of memristive memory and the ReRAM technology is given in the first two chapters [34], [35], in the book *Resistive Switching*, edited by Ielmini and Waser [36].

Each of these device classes shows a more or less different technology and working principle causing different benefits and drawbacks what itself leads to different appropriate use scenarios of these devices. In the following we will briefly

show an overview of these memristive devices used as memories.

PCMs are based on the use of calcogenide materials which can be switched between an amorphous and a crystalline state. This is realised by heating up a conductive rod reaching through the calcogenide material with a high write current. The two states show different behaviours in their electric resistance when the current is flowing through such a device. If the calcogenide is crystalline, the whole device is in the low resistance state (LRS), in contrast it is in the high resistance state (HRS) if the calcogenide is amorphous. Furthermore, it is also possible to adjust intermediate states which are located between the two extremes, the LRS and the HRS. This possibility leads us to the first benefit of such PCM devices, namely its feasible multi-level cell operation. Additionally, PCMs offer a quite mature technology and show a good compatibility (MLC) with CMOS. PCMs have more than 10^9 an endurance comparable to ReRAMs which have the best endurance of current non-volatile memristive devices. The endurance corresponds to the maximum number of possible switching cycles up to the moment, in which the device does not work anymore. On the other side there are some challenges in the controlling of the switching process. This refers to the necessary high write circuits, a 10x slower switching speed than ReRAMs due to the slow crystalline process, and the resistance drift in the amorphous state that has to be compensated on circuit level.

STT-RAMs are based on a parallel and anti-parallel configuration of a stack of ferromagnetic layers forming a magnetic tunnel junction (MTJ) structure. The magnetization at the terminals of the MTJ stack is on one side fixed, therefore this side is denoted as a fixed layer, whereas on the opposite side the so-called free layer is located, which can be switched between two magnetization directions. If both layers are in parallel to each other, the electrons with opposite orientation spin-polarized can pass with a high probability through the stack. Therefore in this case the device is in HRS. In contrast, if the two layers are polarized anti-parallel to each other the probability that an electron can pass both layers is low, since the electron will always meet a layer with different polarization to its own one independent in which direction the electron is spin-polarized. Therefore in this case the device is in a HRS. The benefits of such technology is the fast switching and its relatively mature technology even if it is a challenge to make it compatible with CMOS because the MTJ stack can consist of more than ten layers of not easy to handle ferromagnetic materials, e.g. CoFeB or MgO. Nevertheless, due to its low energy efficient features STT-MRAM technology is strongly discussed to use them in last-level caches.

The ReRAM technology can be subdivided in three different approaches which all exploit nanoionic switching mechanisms. These three approaches are typified either as electrochemical memory (ECM), valence change memory (VCM) (see Fig. 5), or thermochemical memory (TCM) which are using different ionic mechanisms to generate different resistances. In TCMs and ECMs a so-called filamentary structure is used to build up

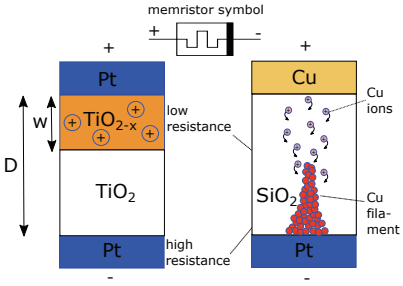


Fig. 5: ECM (left) and VCM (right) ReRAMS (ECM).

and down small metallic bridges by redoxation and oxidation processes in ionized material layers consisting of e.g. TiO_2 or HfO_2 which are entangled between two metal plates as terminals. TCMs are unipolar, i.e. the same voltage is applied to the poles and a filament with low resistance characteristic is growing from both sides. In contrast to that, in ECM two opposite voltages are applied to the terminals which are normally composed of different metals. By this bipolar control mechanisms voltage and reversed voltage signals are used to build up the metallic filament by a redox transitions and to dissolve it again by launching local oxidation processes.

In VCMs not only a filament but also a complete metallic layer or an area interface is built up and dissolved by the exchange of ions. VCMs are also bipolar devices and they correspond to the technique that was used in the memristors of Hewlett Packard [37]. Due to the focus on ion motion as underlying switching process much more localized structures in the nanometer range sized cells, e.g. 10x10 nm[1] or even less, can be realized offering good scalability. A large HRS / LRS ratio makes the interfacing to resistance evaluating CMOS circuits easier. ReRAMs are further characterized by fast switching in the ns range, but even $100\ ps$ have been demonstrated. This characteristic is given due to the small distances the ions have to move and the high electrical field forces that occur in the nanoscale active region causing a so-called Joule heating what for its part further increases the ion mobility. A further advantage is the good compatibility of ReRAMs with CMOS manufacturing processes even if 3D integration of PCMs is a little bit easier since PCMs need only a unipolar selection device compared to the bipolar ReRAMs switching. The endurance, which is may be the most important feature for memristive elements concerning their use in computing circuits either as memory or as switching element, is reported very different in literature. One can find values of 10^6 cycles up to more than 10^{12} cycles. The power comsumption is in the pf range, which makes ReRAMs a good candidate for an use in embedded applications. For example, in 2013 Panasonic is the first semiconductor manufacturer who integrated ReRAM into their microcontroller for storing firmware [38].

V. MEMRISTIVE DEVICES FOR ARITHMETIC

This section lays a focus on how memristive devices can be used to realise new computing concepts for arithmetic circuits. In particular, the presented concepts will exploit qualitative benefits of memristive devices that can not be so easily realised with pure SRAM or DRAM memory cells. This will be on the digital side the multi-level cell (MLC) capability which can be used for new ternary computing concepts like e.g. ternary adders. Ternary content-addressable memory (TCAM) is another architecture in which memristive devices are used for ternary computing schemes [39]. However, TCAM does not exploit the MLC feature. It uses two memristors for a storing a logical 1 and 0, and an additional third memristor for the realisation of the *don't care* state, which is essential for a CAM. Therefore, the rest of this section will only focus on the ternary computing.

Actually, ternary computing schemes for arithmetic operations like addition, subtraction, multiplication and division are long known. First mathematical investigations go back to the 17^{th} century. Two newer ground breaking work was done by Avizienis [40] and Parhami [41]. The first work shown that carry-free additions can be realised by using so-called signed-digit numbers to a base $r \geq 3$. It means that numbers are not presented in the usual sign magnitude presentation, like e.g. in one's or two's complement, but each digit can also have negative values. Then, independent of the operand's word length an addition can be carried out in $O(1)$ instead of $O(N)$ or $O(log(n))$ which is unavoidable if pure binary numbers and at the same time a reasonable number of integrated Boolean gates is utilized. In 1988 Parhami [41] presented a solution in which also a base $r = 2$ can be used to make the realisation with digital electronics possible.

The question remains why such ternary concepts were not used in the last decades in integrated microprocessors if their benefits concerning the run time of arithmetic operations are obvious. One answer to that question is that no CMOS compatible storage device was available that could store three states. This had led to a situation that the complete register files, the caches and even the data segments of the main memory had to be doubled by two SRAM or DRAM cells to store three states. With the emerging of MLC-capable memristive devices this situation has changed. The idea to use MLC memristive devices for ternary adders was first published in [42]. The first technical solution using MLC ReRAMs for redundant arithmetic operations is shown in [43]. The clear qualitative benefit over SRAM and DRAM memory for ternary arithmetic can be exploited in two directions. First, MLC based memristive devices can be used as ternary memory in digital CMOS circuits or, second, in pure in-memory computing circuits in which a well-directed state transfer between the three states in one memristive device is induced according to the compute rules of ternary computing.

The way how a ternary addition works is explained by means of the example shown in the Table II. The basic idea to avoid a carry transfer over more than two digits is that in step 1 only 0, -1, or -2 is used for the result of a digit in the intermediate sum z; whereas in the so-called transfer vector t only 0 or a positive 1 is used. This avoids a further generation of a carry. The rules of math have to be observed, i.e. $0 + 1$ yields -1 for z_i and 1 for t_{i+1}. The second step is necessary to get rid of the -2 in digit z_1. Now, -2 turns to -1 in digit t'_2. In general, by repeatedly applying the rules of addition, only 0s and 1s are generated in the vector z' while only 0s and -1s in vector t'. Then, no situation can occur that two positive 1s or two negative 1s will meet at the same digit position and no carry can occur. Therefore the addition requires exactly three steps independent of the operands' word length.

TABLE II: Addition of Two Ternary Numbers.

x =	(0	0	-1	0)$_2$	= $(-2)_{10}$
+y =	(0	1	-1	0)$_2$	= $(+2)_{10}$
step 1:					
	0	-1	-2	0	= z
0	1	0	0	0	= t
step 2:					
0	1	1	0	0	= z'
0	-1	-1	0	0	= t'
step 3:					
0	0	0	0	0	= $s = 0$

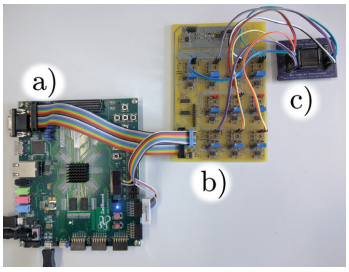


Fig. 6: Prototyping Platform for Memristive Ternary Adder.

Details about the complete Boolean logic for the ternary compute steps and an extensive comparison with other possible ternary representations concerning a solution with MLC capable memristive devices can be found in [44].

The memristive ternary adder was realized as a first prototype using discrete electronic devices (Fig. 6) consisting of an FPGA board (a) that implements the Boolean logic for the ternary adder, a device from BioInspired Inc. (c) to provide the memristors, and an interface card (b) designed by our own which realises the communication between the FPGA and the memristor device via ADC and DAC functions. More details in the set up can be found in [45].

Fig. 7 shows a measurement of the memristor device used as ternary storage. There are two sets of five measurement curves to see. Each curve shows the current running through the memristor by reading the memristor with an applied low voltage after we wrote a memristor in subsequently increased

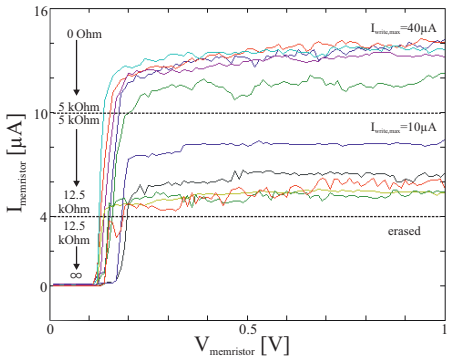


Fig. 7: Determine the Resistance of a Memristor.

```
Datei  Bearbeiten  Ansicht  Suchen  Terminal  Hilfe
Memristor test environment
------------------------------
1) Read all memristors
2) Write register 0
3) Write register 1
4) Add SD
5) Add SD R0+R1->R1
------------------------------
6) All =0
7) All =1
8) All =-1
9) Show threshold values

|MEMR    |VALUE   |ADCLEVEL|
+--------+--------+--------+
|       0|       1|   16674|
|       1|       1|   24492|
|       2|       1|   15681|
|       3|       0|   65535|
|       4|      -1|   61994|
|       5|      -1|   49575|
|       6|       0|   65535|
|       7|       0|   65535|

Register 0
+----+----+----+----+----+
| 00 | 00 |  1 |  1 |  1 |
+----+----+----+----+----+

Register 1
+----+----+----+----+----+
|  0 |  0 | -1 | -1 |  0 |
+----+----+----+----+----+
```

Fig. 8: Testing Program for a Memristive Ternary Adder.

voltage steps of $\delta V = 0.5\ V$. This was done for two different compliant current limits of $10\ \mu A$ and $40\ \mu A$. Even though the five curves for a certain current compliance level scatters widely, we still could clearly distinguish the three desired states.

Fig. 8 shows a screen display of the program that controls carried out experiments. Multiple of the in all 20 memristor cells in the memristor device (Fig. 6c) were addressed as ternary registers. Their content was read in the FPGA. Then, the new ternary result was calculated there and written back to the ternary memristor device.

VI. Memristive Devices for Resistive Computing

Resistive computing enabled by memristive technology has introduced new opportunities to renovate existing computing paradigms for embedded and low power computing [46,47],

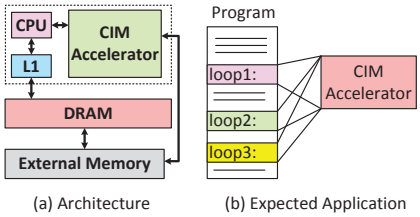


Fig. 9: CIM-based Architecture

in-memory computing [18,21,48] as well as neuromorphic computing [49,50].

The rest of this section will describe the Computation-in-Memory (CIM) and neuromorphic processing as examples of the emerging resistive computing.

A. Computation-in-Memory

One potential in-memory computing architecture is CIM, which was introduced in [18] based on the concept of integrating computation and storage units in a dense memristor crossbar. CIM is implemented as an accelerator (for specific applications) and integrated into conventional architectures to improve overall computer performance [2,18].

The CIM-based architecture consists of a conventional processor, caches, CIM accelerator, main memory DRAM and external memory (as shown in Fig. 9(a)). Similarly as in conventional architectures, the processor fetches, decodes and executes a big data program. However, in conventional architectures, the intensive memory accesses consume (relative to ALU instructions) an enormous amount of energy and significantly degrade the overall performance due to frequent cache misses. As compared to an ALU operation, loading a word from the on-chip SRAM (50x) and off-chip DRAM (6400x) cost much more energy [51,52]. Eliminating this communication will impact the overall performance significantly, especially for data-intensive applications. In order to reduce the data transfers between caches and memories, the CIM accelerator will execute the data-intensive parts of the program locally within the CIM accelerator. Note that the CIM accelerator can perform parallel operations locally on the data stored in the non-volatile memory, hence the memory bottleneck can be significantly reduced. Therefore, the CIM architecture achieves significant improvements in both performance and energy consumption. The performance can be further improved if appropriate applications are mapped on the CIM accelerator.

The potential applications that benefit from CIM accelerator are (big data) applications where communication between processor and memory results in a low performance and high energy consumption. In case the CIM accelerator's capacity is large enough to store the application data, a high level of parallelism can be exploited. In addition, a higher performance can be achieved when different operations are applied to the same data, i.e., data that is not changing frequently; this also

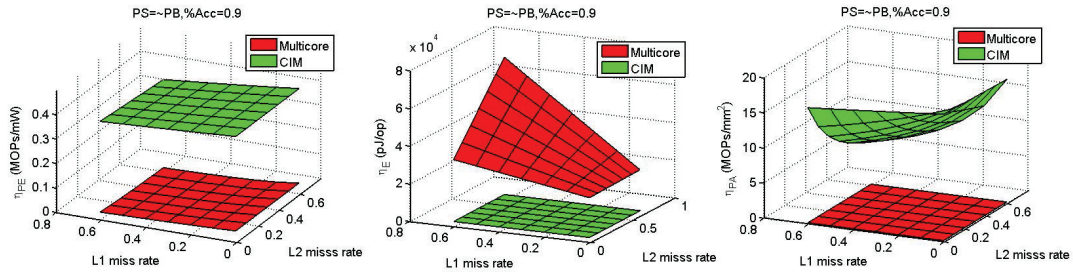


Fig. 10: Evaluation Results for CIM and Multicore Architectures

benefits the endurance of the non-volatile memory of the CIM accelerator. Last but not least, if the processor provides appropriate instructions to the CIM accelerator ahead of the normal execution time, the CIM accelerator already can start performing its operations while the CPU is simultaneously executing other operations, resulting in overall performance improvement.

Fig. 9(b) illustrates a program that could be executed efficiently on the CIM accelerator. In this case, multiple invoked loops work on the same large datasets; obviously the data should be initialized on the CIM accelerator. Each time the loop is invoked, the processor sends a request to the CIM accelerator; the latter, performs the requested operations and returns the results to the processor. Examples of such applications are database applications, where multiple queries (each consisting of large loops) are applied to a fixed database. These queries are used to look for specific data patterns in the database.

To illustrate how the CIM-based computer architecture advances the state-of-the-art, its estimated performance will be compared to a multicore-based architecture. The assumptions for the multicore architecture and CIM-based architecture are similar to those in [53]; the multicore architecture consists of 4 cores (ALU only), two levels of caches (32KB L1 and 256KB L2) and 4GB DRAM; the CIM-based architecture consists of one core (ALU only), two levels of caches (32KB L1 and 256KB L2), 2GB DRAM, and a CIM accelerator with a dedicated computing unit and a 2GB memory. The non-accelerated part is executed by the ALU of the conventional processor and the accelerated part by CIM's dedicated computing unit. The memory operations are modeled based on cache miss rates and DRAM access time, similarly as provided in [18,53]. Three metrics are used for the evaluation: (1) performance energy efficiency η_{PE} (defined by MOPs/mW), (2) energy efficiency η_E (defined by pJ/op), and (3) performance area efficiency η_{PA} (defined by MOPs/mm^2).

Fig. 10 shows the results of the evaluation metrics for both architectures. It assumes that 90% of the instructions can be accelerated on CIM. Both architectures execute petabyte problem size. As the organization of the CIM-based architecture preserves the conventional part of a multicore architecture (i.e., CPU, caches, DRAM and external memory), only 10x improvement is obtained with respect to the performance-energy efficiency. However, the CIM-based architecture achieves four orders of magnitude energy efficiency improvement in comparison with the multicore architecture. Furthermore, the proposed architecture is 15x area-efficient than the multicore architecture. In comparison with state-of-the-art, the proposed architecture is capable of realizing significant improvements, despite the high switching latency and low endurance of memristor technology. The improvements are the result of a significant reduction of cache and DRAM accesses and the usage of non-volatile memory. The reduction of memory accesses leads to a lower latency and lower energy consumption, while the non-volatile memory reduces the static power to practically zero.

B. Neuromorphic Processing

Memristive devices related to neuromorphic processing concern the weights that have to be used in artificial neural networks. They are quasi naturally mapped onto different stored resistance levels. Moreover, the neuromorphic processing systems use memristive devices as not only processing elements but also storing elements, which is not possible using CMOS memories. This finally leads to more energy-efficient and smaller circuits as compared to current CMOS solutions.

Actually, the proposals found in literature to use memristive devices for analogue processing came much earlier and have also a much higher number than the proposals for digital solutions. The idea, e.g. for memristive spike-time-dependent plasticity (STDP) networks [54], [55] is to mimic directly the functional behaviour of a neuron. In STDP networks the strength of a link to a cell is determined by the time correlation of incoming signals to a neuron along that link and the output spikes. The shorter the input pulses occur compared to the output spike, the stronger the input links to the neuron are weighted. In contrast, the longer the input signals lay behind the output spike, the weaker the link is adjusted. This process of strengthening or weakening the weight shall be directly mapped onto memristors by increasing or decreasing their resistance depending which voltage polarity is applied to the terminals of a two-terminal memristive device. This direct mapping of an STDP network to an analogue equivalent of the

biological cells by an artificial memristor based neuron cells shall emerge new extreme low-energy neuromorphic circuits.

An example how memristive devices are used for neuromorphic processing is a chip based on PCM technology realized by IBM. The chip comprises 64k cells, consisting of 256 axons by 256 dendrites and was demonstrated in 2015 [56]. The update of the artificial synaptic weights uses STDP as an in-situ learning function. Besides this memristor based STDP networks, there are lots of other proposals, e.g. [57], for neural networks to be realised with memristor based crossbar and mesh architectures for cognitive detection and vision applications.

VII. Opportunities and Challenges

A. Opportunities

Memristive devices provide significant opportunities with respect to the following aspects.

- Memristive devices are a promising alternative for the leakage wall that limits CMOS technology scaling.They are non-volatile and require low energy consumption; hence, they can be used to produce memory and logic circuits that have practically zero static power [58–60].
- Memristive devices could also help solving the computer architecture walls. Memristive devices are capable of both storing and computing, which enables new computing paradigms [18,52,61]. Furthermore, massive parallelism can be achieved as the high density of memristive devices facilitate more functional units within the same area [20,53]. In addition, the high energy consumption of today's computers can also be reduced due to the low dynamic energy and zero static energy consumption [53].
- The above benefits (non-volatility, low energy consumption and scalability) provide solutions for emerging applications such as embedded and low power systems for IoT [62], big data and health care applications [18].

B. Challenges

Despite the above opportunities, memristive devices are facing several challenges.

- Memristive devices are still in their infancy stage; hence, there is a lack of libraries with well-optimized memristive designs as well as mature automation tools to speed-up the exploration process.
- Memristive devices suffer from limited endurance which is currently around (10^{12}) [8]. This is insufficient for general purpose computing; therefore, a new computing paradigm or logic style is required to deal with this limited endurance (e.g., high-radix computing [63]). In addition, researchers strongly believe that the endurance will substantially increase and reach 10^{16} [64]
- Memristive devices also face reliability challenges in both manufacturing and design phase. Nonvolatile memory manufacturing has to deal with nonuniform resistance profile across the crossbar array, resistance drift, inherent device-to-device and cycle-to-cycle variations as well as yield issues [2]. Furthermore, the intrinsic variation of memristive devices make it difficult to design robust logic circuits [2,65].
- The integration of memristive devices with CMOS is still an open research question. It seems that this kind of integration is possible as presented in [66]; however, details of stacking memristive layers on top of CMOS are still in research. Moreover, there is limited work that investigates the impact of the CMOS controller on an entire memristive system [53]. Further investigation requires not only the optimization of the CMOS part, but also its efficient use to control the crossbars while maintaining the system scalability.

VIII. Conclusion

Memristive devices provide many opportunities; not only to enable next-generation non-volatile memories (with fast access speed, up to 1TB storage capacity, and multi-level capability), but also to enable novel alternative computing architectures required for emerging applications, such as energy-efficient neuromorphic systems and computation-in-memory architectures. However, there are still many challenges ahead that need to be addressed.

References

[1] EU-Commission, "Next generation computing roadmap." European Union, 2014.

[2] S. Hamdioui *et al.*, "Memristor for computing: Myth or reality?" in *DATE*. IEEE, 2017.

[3] B. Hoefflinger, *Chips 2020: a guide to the future of nanoelectronics*. Springer Science & Business Media, 2012.

[4] H. Jones, "Whitepaper: Semiconductor industry from 2015 to 2025." International Business Strategies, 2015.

[5] J. L. Hennessy *et al.*, *Computer architecture: a quantitative approach*. Elsevier, 2011.

[6] ITRS, "Beyond cmos white paper." ITRS, 2014.

[7] R. Waser *et al.*, "Redox-based resistive switching memories–nanoionic mechanisms, prospects, and challenges," *Advanced Materials*, pp. 2632–2663, 2009.

[8] J. J. Yang *et al.*, "Memristive devices for computing," *Nature nanotechnology*, vol. 8, pp. 13–24, 2013.

[9] R. Fackenthal *et al.*, "A 16gb reram with 200mb/s write and 1gb/s read in 27nm technology," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014 IEEE International*. IEEE, 2014, pp. 338–339.

[10] M.-F. Chang *et al.*, "Challenges and circuit techniques for energy-efficient on-chip nonvolatile memory using memristive devices," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 5, pp. 183–193, 2015.

[11] A. Kawahara *et al.*, "An 8 mb multi-layered cross-point reram macro with 443 mb/s write throughput," *IEEE Journal of Solid-State Circuits*, vol. 48, pp. 178–185, 2013.

[12] J. Borghetti *et al.*, "A hybrid nanomemristor/transistor logic circuit capable of self-programming," *Proceedings of the National Academy of Sciences*, vol. 106, pp. 1699–1703, 2009.

[13] G. S. Rose *et al.*, "Leveraging memristive systems in the construction of digital logic circuits," *Proceedings of the IEEE*, vol. 100, pp. 2033–2049, 2012.

[14] E. Linn *et al.*, "Beyond von neumann–logic operations in passive crossbar arrays alongside memory operations," *Nanotechnology*, vol. 23, p. 305205, 2012.

[15] G. Snider, "Computing with hysteretic resistor crossbars," *Applied Physics A: Materials Science & Processing*, vol. 80, pp. 1165–1172, 2005.

[16] J. Borghetti *et al.*, "Memristive switches enable stateful logic operations via material implication," *Nature*, vol. 464, pp. 873–876, 2010.

[17] Q. Guo *et al.*, "Ac-dimm: associative computing with stt-mram," *ACM SIGARCH Computer Architecture News*, vol. 41, pp. 189–200, 2013.

[18] S. Hamdioui *et al.*, "Memristor based computation-in-memory architecture for data-intensive applications," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*. EDA Consortium, 2015, pp. 1718–1725.

[19] H. A. Du Nguyen *et al.*, "Computation-in-memory based parallel adder," in *NANOARCH*. IEEE, 2015.

[20] A. Haron *et al.*, "Parallel matrix multiplication on memristor-based computation-in-memory architecture," in *High Performance Computing & Simulation (HPCS), 2016 International Conference on*. IEEE, 2016, pp. 759–766.

[21] P.-E. Gaillardon *et al.*, "The programmable logic-in-memory (plim) computer," in *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2016, pp. 427–432.

[22] L. Chua, "Memristor-the missing circuit element," *IEEE Transactions on circuit theory*, vol. 18, pp. 507–519, 1971.

[23] D. B. Strukov *et al.*, "The missing memristor found," *nature*, vol. 453, p. 80, 2008.

[24] S. Kvatinsky *et al.*, "Mrl-memristor ratioed logic," in *2012 13th International Workshop on Cellular Nanoscale Networks and their Applications*. IEEE, 2012, pp. 1–6.

[25] L. Gao *et al.*, "Programmable cmos/memristor threshold logic," *IEEE Transactions on Nanotechnology*, vol. 12, pp. 115–119, 2013.

[26] I. Vourkas *et al.*, "A novel design and modeling paradigm for memristor-based crossbar circuits," *IEEE Transactions on Nanotechnology*, vol. 11, pp. 1151–1159, 2012.

[27] L. Xie *et al.*, "Scouting logic: A novel memristor-based logic design for resistive computing," in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2017, pp. 335–340.

[28] S. Li *et al.*, "Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories," in *DAC*. IEEE, 2016.

[29] T. You *et al.*, "Exploiting memristive bifeo3 bilayer structures for compact sequential logics," *Advanced Functional Materials*, vol. 24, pp. 3357–3365, 2014.

[30] E. Lehtonen *et al.*, "Memristive stateful logic," in *Memristor Networks*. Springer, 2014, pp. 603–623.

[31] S. Kvatinsky *et al.*, "Magic–memristor-aided logic," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 61, pp. 895–899, 2014.

[32] L. Xie *et al.*, "Boolean logic gate exploration for memristor crossbar," in *2016 International Conference on Design and Technology of Integrated Systems in Nanoscale Era (DTIS)*. IEEE, 2016, pp. 1–6.

[33] S. Kvatinsky *et al.*, "Memristor-based material implication (imply) logic: design principles and methodologies," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, pp. 2054–2066, 2014.

[34] R. Waser *et al.*, *Introduction to Nanoionic Elements for Information Technology*. Wiley-VCH Verlag GmbH & Co. KGaA, 2016, pp. 1–30. [Online]. Available: http://dx.doi.org/10.1002/9783527680870.ch1

[35] E. Linn *et al.*, *ReRAM Cells in the Framework of Two-Terminal Devices*. Wiley-VCH Verlag GmbH & Co. KGaA, 2016, pp. 31–48. [Online]. Available: http://dx.doi.org/10.1002/9783527680870.ch2

[36] D. Ielmini *et al.*, *Resistive switching: from fundamentals of nanoionic redox processes to memristive device applications*. John Wiley & Sons, 2015.

[37] R. Williams, "How we found the missing memristor," *IEEE Spectr.*, vol. 45, pp. 28–35, Dec. 2008. [Online]. Available: http://dx.doi.org/10.1109/MSPEC.2008.4687366

[38] Y. Hayakawa *et al.*, "Highly reliable taox reram with centralized filament for 28-nm embedded application," in *2015 Symposium on VLSI Circuits (VLSI Circuits)*, June 2015, pp. T14–T15.

[39] P. Junsangsri *et al.*, "A memristor-based TCAM (Ternary Content Addressable Memory) cell," in *2014 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, Jul. 2014, pp. 1–6.

[40] A. Avizienis, "Signed-digit numbe representations for fast parallel arithmetic," *Electronic Computers, IRE Transactions on*, vol. EC-10, pp. 389–400, Sept 1961.

[41] B. Parhami, "Carry-free addition of recoded binary signed-digit numbers," *Computers, IEEE Transactions on*, vol. 37, pp. 1470–1476, 1988.

[42] D. Fey, "Using the multi-bit feature of memristors for register files in signed-digit arithmetic units," *Semiconductor Science and Technology*, vol. 29, p. 104008, 2014. [Online]. Available: http://stacks.iop.org/0268-1242/29/i=10/a=104008

[43] W. Kim *et al.*, "Multistate Memristive Tantalum Oxide Devices for Ternary Arithmetic," *Scientific Reports*, vol. 6, Dec. 2016. [Online]. Available: http://www.nature.com/articles/srep36652

[44] D. Fey *et al.*, "Using memristor technology for multi-value registers in signed-digit arithmetic circuits," in *Proceedings of the Second International Symposium on Memory Systems*, ser. MEMSYS '16. New York, NY, USA: ACM, 2016, pp. 442–454.

[45] D. Wust *et al.*, "Prototyping memristors in digital system with an fpga-based testing environment," in *to appear in Proc. IEEE/ 27^{th} International Symposium on Power, and Timing, Modeling, Optimization and simulation (PATMOS)*, Sep. 2017, p. 7 pages.

[46] C. J. Xue, "Redesigning software and systems for nonvolatile processors on self-powered devices," in *Smart Sensors at the IoT Frontier*. Springer, 2017, pp. 107–123.

[47] F. Su *et al.*, "Design of nonvolatile processors and applications," in *Very Large Scale Integration (VLSI-SoC), 2016 IFIP/IEEE International Conference on*. IEEE, 2016, pp. 1–6.

[48] L. Yavits *et al.*, "Resistive associative processor," *CAL*, 2015.

[49] H. Mostafa *et al.*, "Beyond spike-timing dependent plasticity in memristor crossbar arrays," in *Circuits and Systems (ISCAS), 2016 IEEE International Symposium on*. IEEE, 2016, pp. 926–929.

[50] F. Alibart *et al.*, "Pattern classification by memristive crossbar circuits using ex situ and in situ training," *Nature communications*, vol. 4, p. 2072, 2013.

[51] A. Danowitz *et al.*, "Cpu db: recording microprocessor history," *Communications of the ACM*, vol. 55, pp. 55–63, 2012.

[52] A. Pedram *et al.*, "Dark memory and accelerator-rich system optimization in the dark silicon era," *IEEE Design & Test*, vol. 34, pp. 39–50, 2017.

[53] H. A. Du Nguyen *et al.*, "On the implementation of computation-in-memory parallel adder," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2017.

[54] G. S. Snider, "Spike-timing-dependent learning in memristive nanodevices," in *2008 IEEE International Symposium on Nanoscale Architectures, NANOARCH 2008, Anaheim, CA, USA, June 12-13, 2008*, 2008, pp. 85–92. [Online]. Available: https://doi.org/10.1109/NANOARCH.2008.4585796

[55] T. Serrano-Gotarredona *et al.*, "Stdp and stdp variations with memristors for spiking neuromorphic learning systems," *Frontiers in Neuroscience*, vol. 7, p. 15, 2013. [Online]. Available: http://journal.frontiersin.org/article/10.3389/fnins.2013.00002

[56] S. Kim *et al.*, "Nvm neuromorphic core with 64k-cell (256-by-256) phase change memory synaptic array with on-chip neuron circuits for continuous in-situ learning," in *2015 IEEE International Electron Devices Meeting (IEDM)*, Dec 2015, pp. 17.1.1–17.1.4.

[57] C. K. K. Lim *et al.*, "Computing Image and Motion with 3-D Memristive Grids," in *Memristor Networks*, A. Adamatzky *et al.*, Eds. Springer International Publishing, 2014, pp. 553–558.

[58] Crossbar Inc., "Rethink data storage with 3d reram." [Online]. Available: https://www.crossbar-inc.com/en/

[59] Intel Corporation, "Revolutionizing memory and storage." [Online]. Available: https://www.intel.com/content/www/us/en/architecture-and-technology/intel-optane-technology.html

[60] S. C. Bartling *et al.*, "An 8mhz 75μa/mhz zero-leakage non-volatile logic-based cortex-m0 mcu soc exhibiting 100vdd=0v with lt;400ns wakeup and sleep transitions," in *2013 IEEE International Solid-State Circuits Conference Digest of Technical Papers*, Feb 2013, pp. 432–433.

[61] M. Barbareschi *et al.*, "Memristive devices: Technology, design automation and computing frontiers," in *Design & Technology of Integrated Systems In Nanoscale Era (DTIS), 2017 12th International Conference on*. IEEE, 2017, pp. 1–8.

[62] K. Ma *et al.*, "Nonvolatile processor architecture exploration for energy-harvesting applications," *IEEE Micro*, vol. 35, pp. 32–40, 2015.

[63] B. Parhami, *Computer arithmetic*. Oxford university press, 1999, vol. 20, no. 00.

[64] M.-F. Chang *et al.*, "Endurance-aware circuit designs of nonvolatile logic and nonvolatile sram using resistive memory (memristor) device," in *Design Automation Conference (ASP-DAC), 2012 17th Asia and South Pacific*. IEEE, 2012, pp. 329–334.

[65] L. Xie *et al.*, "On the robustness of memristor based logic gates," in *Design and Diagnostics of Electronic Circuits & Systems (DDECS), 2017 IEEE 20th International Symposium on*. IEEE, 2017, pp. 158–163.

[66] Q. Xia *et al.*, "Memristor- cmos hybrid integrated circuits for reconfigurable logic," *Nano letters*, vol. 9, pp. 3640–3645, 2009.

2

A Classification of Memory-Centric Computing

Hoang Anh Du Nguyen
Delft University of Technology
H.A.DuNguyen@tudelft.nl

Jintao Yu
Delft University of Technology
J.Yu-1@tudelft.nl

Muath Abu Lebdeh
Delft University of Technology
M.F.M.AbuLebdeh@tudelft.nl

Mottaqiallah Taouil
Delft University of Technology
M.Taouil@tudelft.nl

Said Hamdioui
Delft University of Technology
S.Hamdioui@tudelft.nl

Francky Catthoor
Inter-university Micro-Electronics Center (IMEC)
Francky.Catthoor@imec.be

ABSTRACT

Technological and architectural improvements have been constantly required to sustain the demand of faster and cheaper computers. However, CMOS down-scaling is suffering from three technology walls: leakage wall, reliability wall and cost wall. On top of that, performance increase due to architectural improvements is also gradually saturating due to three well-known architecture walls: memory wall, power wall and instruction level parallelism (ILP) wall. Hence, a lot of research is focusing on proposing and developing new technologies and architectures. In this paper, we present a comprehensive classification of memory-centric computing architectures; it is based on three metrics: computation location, level of parallelism and used memory technology. The classification does not only provide an overview of existing architectures with their pros and cons, but also unify the terminology that uniquely identifies these architecture, and highlight the potential future architectures that can be further explored. Hence, it sets up a direction for future research in the field.

ACM Reference Format:
Hoang Anh Du Nguyen, Jintao Yu, Muath Abu Lebdeh, Mottaqiallah Taouil, Said Hamdioui, and Francky Catthoor. 2019. A Classification of Memory-Centric Computing. In *Proceedings of Journal (in review) (Manuscript)*. ACM, New York, NY, USA, 17 pages. https://doi.org/

1 INTRODUCTION

For several decades, technology scaling has provided a 43% performance gain for each successive node and cheaper computers as a result of a higher operating frequency and lower cost per transistor, respectively [14, 51]. On top of that, smart architectural improvements such as pipelining and cache hierarchies increased the computer performance up to 50% every two years [46]. However, CMOS scaling suffers from three main walls: leakage wall, reliability wall and cost wall [42], while computer architectures also face three walls: memory wall, power wall and instruction level parallelism (ILP) wall [96]. In order to address these walls, novel technologies and architecture are under research to improve the performance [51]. As a result, an enormous amount of computer architectures has been proposed recently. Therefore, a complete classification of these architectures is needed; not only to have a useful way of describing and comparing them, but also to have a clear view about what is explored and what not yet.

Manuscript, In review, 2019
2019. ACM ISBN ...$
https://doi.org/

Limited work has addressed this problem. Most of the well-known classifications separate the processors from the memory. Therefore, these classifications often are processor-centric based architectures, such as Flynn's [34], Skillicorn's [113] and Shami-Hemani's classification [108]. Although these classifications work well for processor-centric architectures proposed in the past decades, they are not applicable to the emerging memory-centric architectures. Other small-scale surveys mostly target a specific type of computer architectures such as vector processors, automata processors or processing-in-memory architectures [22, 55, 66, 104, 109, 118, 121, 122]. These surveys only discuss a limited part of the computer architecture classification, and in addition, do not contain the complete space of both conventional processor-centric architectures and memory-centric architectures. Therefore, these surveys often make no distinction between processing inside and near the memory. This leads to a confusion in terminology (e.g., processing-in-memory, logic-in-memory, in-memory computing, near-memory computing, etc.). For example, Hybrid Memory Cube is considered to be near-memory-computing [97], however, it is also referred to as processor-in-memory [3]. Some recent classifications and reviews did mention those architectures in the context of technology development [90, 131]. However, these papers mostly targeted the technological feasibility instead of the characteristics and variants of such computer architectures. In addition to the above, there are some architecture-related papers that briefly discussed the features of emerging architectures [85, 87, 102]. However, they are incomplete, focus mostly on relatively narrow aspects and only classify the architectures based on applications [85] and logic design methods [87, 102]. In short, there is still a lack of systematic and complete classification that focuses on memory-centric computing or computer architectures in general. This is exactly the target of this paper.

This paper presents a comprehensive classification of memory-centric computing, and discusses both conventional and emerging computing architectures. The classification is based on three metrics: computation location, memory technology and computation parallelism. The computation location indicates where the computations are performed (e.g., near or far from the memory) and provides an insight regarding the severeness of the memory wall. The memory technology, which provides characteristics of the memory, can enable new computer architectures (e.g., resistive computing). The computation parallelisms specifies the type of parallelism that can be exploited in an architecture (e.g. task level parallelism). With these distinct metrics, the classification covers *all* computing architectures in general and memory-centric computing in specific. Note

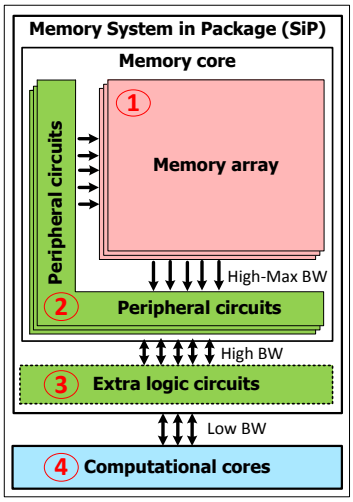


Figure 1: Computer Architecture

however that it does not make previous proposed classifications obsolete, as they typically target specific sub-classes. In short, the contributions of this paper are the following:

- Unify the terminology for computer architectures such that it is applicable to all computing paradigms including conventional, in-memory and near-memory computing.
- Propose a complete classification that includes both existing and emerging architectures.
- Explain one representative architecture of each sub-class in detail.
- Discuss and evaluate the main advantages and disadvantages of the different classes and selected architectures.
- Highlight the whole space of memory-centric computing, including the non-explored architectures.

The rest of this paper is structured as follows. Section 2 shows the metrics used in the classification, briefly introduces the four classes, and provides a quantitative comparison among them. Sections 3, 4, and 5 present the characteristics of the three memory-centric computing classes; the fourth class contains the traditional von Neuman architectures and is out-of-the-scope of this paper. Section 6 discusses the pros and cons of this classification and compares its with existing ones. Finally, section 7 concludes this paper.

2 CRITERIA AND CLASSIFICATION

In this section, we first present the set of metrics we use to classify computer architectures. Thereafter, we show our classification and map the existing architectures on it. Finally, we compare the classes qualitatively based on their most important metric.

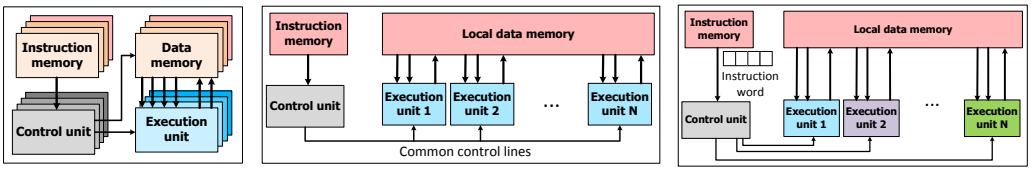
2.1 Classification Metrics

We propose several metrics to classify computer architectures based on the computing resources and memory. A computer architecture or system consists of (one or more) memories and (one or more) computational units as shown in Fig. 1. The memories can reside in a core (i.e., memory core) or System-in-Packages (SiP). A memory core consists of one or more cell arrays (used for storage) and peripheral circuits (used to access the memory cells). Note that register files and caches are not considered as storage here, as they are optimized for speed with relatively small capacity and temporary storage [46]. Hence, the long term storage of data takes place in the higher layers such as main memory and solid-state disks. Traditionally, the computing takes place in the computational cores. However, recently architectures with computing power in the memory have been proposed [43, 95, 97]. In case the memory contains additional logic circuits such as in Hybrid Memory Cubes (HMC) [97], we speak of a System in Package (SiP). With an increasing distance from the main memory array, the available bandwidth (specified by BW in Fig. 1) reduces; note that the bandwidth here is related to the memory bottleneck and will be discussed further in Section 2.3. Based on these definitions, the following metrics are used to classify computer architectures: computation location, memory technology and computation parallelism; they are discussed next.

Computation location: it indicates ***where*** *the result of the computation is produced.* A computation is defined here as a primitive logic function (e.g., logical operations) or arithmetic operation (e.g., addition, multiplication). Fig. 1 indicates the four possibilities where a computation result can be produced; they can be identified by four circled numbers. If the result is produced *within* the memory core, (i.e., the computing takes places within one of the memories), then the computer architecture is referred to as *Computation-Inside-Memory (CIM).* If the result is produced outside the memory core, then the architecture is referred to as *Computation-Outside-Memory (COM).* Both CIM and COM can be further sub-classified.

It is worth stressing that **CIM** architectures perform computations *within* the memory core. As already mentioned, the memory consists of a *memory array* and the *peripheral circuits*. More specifically, depending on *where* the result of the computation is produced, CIM architectures can be divided into two basic sub-classes. These sub-classes can be combined into many hybrid combinations. We will now describe this large space by focusing first on the two extreme sides of this space:

- *CIM-Array (CIM-A):* In CIM-A, the computing result is produced within the array. Note that this is different from a standard write operation. Typical examples of CIM-A architectures use memristive logic designs such as MAGIC and imply [63, 69]. CIM-A architectures require always a redesign of cells to support such logic design, as the conventional memory cell dimensions and their embedding in the bit- and wordline structure do not allow them to be used for logic. A memory cell is namely heavily optimized in terms of processing stack and layout; hence, any changes in the array access require a completely new cell design and characterization process as the material stack of a memory array is specifically optimized for specific control voltages, current,



(a) Task Parallelism **(b) Data Parallelism** **(c) Instruction Parallelism**

Figure 2: Three Types of Computation Parallelism

etc. In addition, modifications in the periphery are sometimes needed to support the changes in the cell changes. Therefore, CIM-A architectures can be sub-divided into two groups: (1) basic CIM-A where only changes inside the memory array are required, and (2) hybrid CIM-A where in addition to major changes in the memory array also minimal to medium changes are required in the peripheral circuit. An example of basic CIM-A is an architecture that performs computations using implication logic [73]. In this logic style, only one memory row is activated at a time, and a number of columns (bits) are read out through sense amplifiers. Hence, due to the same usage as in normal memory, the peripheral circuits do not require any modifications. An example of hybrid CIM-A is an architecture that performs computations using MAGIC [69]; in this case, multiple memory rows are written simultaneously; due to the high write currents modifications are required to the cell and medium changes in the peripheral circuits are needed to activate the multiple rows.

- *CIM-Periphery (CIM-P):* In CIM-P, the computing result is produced within the peripheral circuitry. Typical examples of CIM-P architectures contains logical operations and vector-matrix multiplications [20, 77, 130]. CIM-P architectures typically contain dedicated peripheral circuits such as DACs and/or ADCs [36, 107], and customized sense amplifiers [77, 130]. Note that more radical changes in the peripheral circuit can be made in the future (e.g., changing in control voltages leads to radical changes in voltage drivers and sense amplifiers, or including a full functional processor inside memory banks). Even though the computational results are produced in the peripheral circuits for CIM-P, the memory array is a substantial component in the computations. As the peripheral circuits are modified, the currents/voltages applied to the memory array are typically different than in the conventional memory. Hence, similarly as to the CIM-A sub-classes, the CIM-P architectures are also further divided into two groups: (1) basic CIM-P where only changes inside the peripheral is required, which means the current levels should not be affected, and (2) hybrid CIM-P where the majority of the changes take place in the peripheral circuit and minimal to medium changes in the memory array. An example of basic CIM-P is Pinatubo logic [77]. Pinatubo activates two or more (but not many) rows of a memory array simultaneously during read operations for computations; in addition to a customized sense amplifier to perform the logic operation, this architecture also requires modifications in the address decoder to activate several rows. Note however, that modifications in the cell/array are not required as the total read current is still small. An example of hybrid CIM-P is ISAAC [107]. ISAAC activates all rows of a memory array at the same time during read operations to perform a matrix vector product using an ADC read out circuit. This architecture accumulates currents in the bitline that impose higher electrical loading in the memory array; hence, not only the periphery circuit is heavily modified but also the cell requires changes due to the high bit-line current.

The difference between CIM-A and CIM-P classes is the location of producing results. The results of CIM-A architectures are produced inside the memory array, which may sometimes require read-out operations to obtain the results for further calculations; instead, in CIM-P the results are obtained directly after the operations and may sometimes need an additional step to write the results back to memory. In order to perform computations, both sub-classes impact the design of the memory core. However, in many/most cases both the cell and the peripheral circuitry require changes, i.e., they are hybrids. In case these changes affect mostly the cell, we speak of hybrid CIM-A, otherwise hybrid CIM-P.

In **COM classes**, computations take either place in the extra logic circuits inside the memory SiP (3) or in the traditional computational cores (4) such as CPU, FPGA, etc. In case of the former, the computations take place near the memory core and the architecture is referred to as Computation-Outside-Memory Near (COM-N). In case of the latter, the architecture is referred to as Computation-Outside-Memory Far (COM-F). Note that the bandwidth is still high for COM-N as compared to COM-F, but lower than CIM-A and CIM-P.

Note that architectures where the computation takes place in difference places (e.g., array and peripheral) are called composite architectures. Hence, they are compositions of the leaf nodes in our classification tree. In addition, an architecture could have multiple primitive functions, each with a different computation location. Also these architectures are considered to be composite.

In addition to the computation location, which specifies where the results are produced, it is possible to further divide the classes using the computation method; it specifies how the computation is performed. For example, CIM-A often uses memristor-based computations such as IMPLY [13, 111], Snider [114], MAGIC [69]. CIM-P often uses current summations such as Scouting logic [130], Ambit [106] and Pinatubo [77]. However, this metric is not included

to the classification for two reasons: (i) it is strongly coupled to the computational location, and (ii) it makes the classification too complex and hence loses its simplicity. Nevertheless, including such a metric can be complementary to our work. A further sub-classification based on this metric can be based on existing work in other classifications as shown in [25, 102].

Memory technology: it indicates which technology is used to implement the memory array. The technologies are either conventional charge-based memories such as DRAM/SRAM [82, 83, 89] or emerging non charge-based memories [103]. The non charge-based memories can be further divided into different types based on their physical mechanism: resistive [71, 103], "magnetic" memories [9, 18, 103], molecular memories [39, 74, 75, 99] or mechanical memories [15, 40]. Resistive memories store the data as a resistance value; it includes Resistive RAM (RRAM) [125], phase change memory (PCM) [71], etc. The resistance in RRAM is determined by the presence or absence of a conductive filament between its two electrodes [103], while the resistance in PCM relies on a change between amorphous and crystalline phases [101, 126]. Magnetic memories, such as Magnetic RAM (MRAM), store the data using the magnetization direction of the free layer with respect to the hard or reference layer; it includes, for example, conventional magnetic RAM [135] and STT-MRAM [35, 48]. The resistive and magnetic memories are organized in crossbars with cells placed at each junction. The other types of memories, (i.e., molecular memories, mechanical memories) have not been shown to be useful for computing yet; hence, they are not discussed further in this classification. It is worth mentioning that each of these memory technologies has its own characteristics (read/write latency, endurance, capacity, etc.) and therefore are deployed at different levels in the memory hierarchy [103]. Therefore, the memory technology does not only dictate which CIM operations are technology-wise feasible, but also where in the memory hierarchy they take place.

Computation parallelism: it indicates the level of parallelism that can be exploited in a computer system; i.e., task, data, and/or instruction level parallelism, as shown in Fig. 2. An architecture supports task level parallelism when it contains multiple independent control units and multiple data memories, as shown in Fig. 2a. The independent control units can be used to execute multiple threads or instruction sequences from the same application concurrently; examples are multithreading [29, 120] and multicore systems [38]. In data parallelism, a single control unit is used to apply the *same* instruction concurrently on a collection of data elements, as shown in Fig. 2b; note that all execution units share the same control signals. The data elements can be processed using constant sizes (e.g., vector and array processor [27, 32]) or varying sub-word sizes (e.g., SWAR (SIMD Within A Register) processor [33, 98]). In instruction level parallelism, a single control unit is used to execute *various* instructions concurrently, as shown in Fig. 2c; hence, the execution units have different control signals. A further distinction can be made based on intra-instruction (e.g., pipe-lined processor [119]), inter-instruction (e.g., VLIW processor [127]) parallelism, or a combination of them (e.g., speculative processor [84]).

The three above-mentioned criteria (i.e., computation location, memory technology and computation parallelism) are dependent on each other. The computation location has a big impact on the feasibility of the memory technology and the computation parallelism. For example, realizing ILP in CIM-A is quite difficult or realizing COM-N with SRAM is not economically feasible. Also the parallelism is not fully independent from the computation location (e.g, CIM/COM) and memory technology. For example, data parallelism is often applied straightforward in CIM-A and CIM-P [26, 36]; however it is difficult to realize ILP in CIM-A, while it is much easier in COM-N and COM-F due to the intrinsic pipeline stages in conventional processors. The computation parallelism is also affected by the technology as the technology poses restrictions on the endurance. For example, exploiting ILP in CIM-A architectures demands a high endurance as more writes are required to store immediate stages and hence, are not attractive for emerging memories like RRAM and PCM with endurance limitations.

2.2 Classification

We classify the existing architectures based on the above discussed metrics; the result is shown in Fig. 3. The references of the abbreviated architectures are listed in Table 1.

The classification contains 48 categories. Some categories, the ones located in *red* planes, show that a *lot* of work has been done for that particular class. For the categories in the *pink* planes, a *moderate number* of work has been done. To our best knowledge, no architectures exists in the *blue* planes; these fields are currently unexplored as they received no attention yet from the research community or non-existing due to current restrictions of the technology; these blue planes are not further discussed in this paper due to two main reasons: (1) scope of the paper; the technical and economical feasibility of these planes requires an intensive discussion and is by itself a new contribution and (2) space limitations. Later on, we will discuss several architectures from the red and pink planes.

The developments in memory-centric computing are shown in the timelime of Fig. 4; this shows the trend of computing moving from COM-F to COM-N, CIM-A and CIM-P. In the figure, a larger circle indicates that more work has been proposed in that year. Note that the conventional architectures in COM-F are not memory-centric and hence, are left out. The concept of merging computation and memory was introduced back in 1970 [116]. This concept became popular around 1997 in COM-N architectures and was further developed until 2002. These COM-N architectures, such as VIRAM [67] (initially named IRAM), DIVA [23] or FlexRAM [60], never commercialized due to the limitations of embedded DRAM technology (i.e., costly fabrication process, and inefficient speed and memory capacity trade-off [52, 61, 62]). After that, a long silent period in academia community was observed from 2002 to 2010. Meanwhile, industrial efforts have been invested to deploy large eDRAM in commercial COM-N systems such as POWER7 processor [58], PlayStation2 [6] and Intel's top-class CPUs [68]. From 2012 to 2016, new commercial COM-N architectures based on novel 3D stacking technology have been proposed such as Hybrid Memory Cube (HMC) [47] and High Bandwidth Memory (HBM) [80]. In the last several years, with the emerging of resistive technology, CIM-A and CIM-P architectures started to become popular.

Note that many of the architectures are hybrid and/or composite which means that they can map into multiple classes. In order to

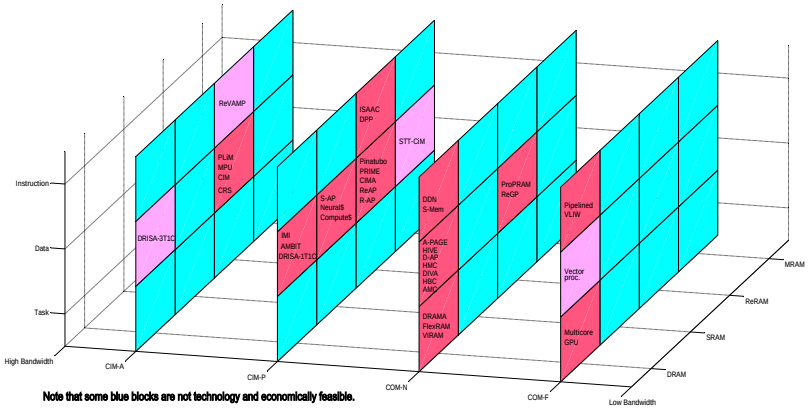


Figure 3: Memory-centric Computing Classification

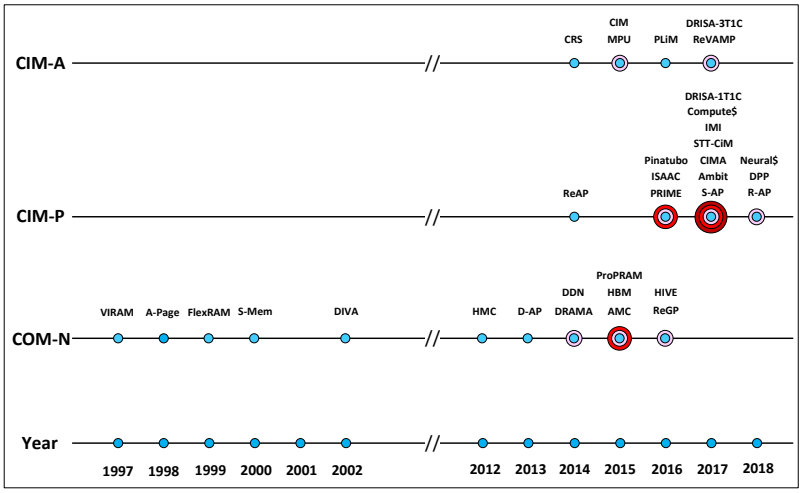


Figure 4: Memory-centric Computing Timeline

Table 1: Abbreviation List

Abbreviation	Reference
DRISA-3T1C	[76]
ReVAMP	[8]
PLiM	[37]
MPU	[49]
CIM	[26, 43]
CRS	[111]
ISAAC	[107]
DPP	[36]
IMI	[31]
AMBIT	[106]
DRISA-1T1C	[76]
S-AP	[117]
Neural$	[28]
Compute$	[1]
Pinatubo	[77]
PRIME	[20]
CIMA	[25]
ReAP	[132]
R-AP	[133]
STT-CiM	[53]
DDN	[19]
S-Mem	[81]
A-PAGE	[94]
HIVE	[3]
D-AP	[92]
DIVA	[23]
HMC	[47]
AMC	[91]
HBM	[80]
DRAMA	[30]
FlexRAM	[60]
VIRAM	[67]
ProPRAM	[124]
ReGP	[88]
Pipelined	[46, 119]
VLIW	[79, 127]
Vector Proc.	[21, 32, 98]
Multicore	[50]
GPU	[64, 93]

simplify Figure 3, the architectures are classified based on their dominant features. For example, DPP exploits both ILP and DLP; however, DPP focuses more on performing various parallel operations using multiple functional units, while it also processing a whole row/column inside the memory; hence, the dominant feature of DPP parallelism is selected as ILP.

2.3 Qualitative Evaluation

In this subsection, we briefly compare the different computing types qualitatively using the most important classification metric, i.e., the computation location. This metric dictates the type of data movements, computation requirements, available bandwidth, memory design efforts, endurance requirement and maturity. With respect to the computation requirements, we discuss whether the architectures require a specific data alignment and whether they have the capability to realize complex functions. With respect to the memory design efforts, we discuss the modifications that are required for the cells, array, peripheral circuit and controller. With respect to maturity, we do not only mean the readiness of the memory technology, but also the available programming and software support,

and current status (i.e., simulations, prototype or fabrication) for such architectures. The results are shown in Table 2 and 3; their content will be discussed next.

Data movement outside the memory core indicates whether the data will remain in the memory core during computing or transferred to computational units outside it. It affects the memory bottleneck due to latency and the energy consumption of data transfers. Both CIM-A and CIM-P architecture have a relatively low amount of data movement outside the memory core, as the processing occurs inside the memory core. Therefore, they have the potential to alleviate the memory bottleneck. Instead of moving data from the memory to the computational cores, in these architectures the instructions are moved and directly applied to the memory; these instructions typically operate on a large data set, hence a high level of parallelism can be obtained. Note however that the current state of the art typically allows limited functions to be implemented in these architectures. Therefore, complex functions would still require data movements to the computational cores outside the memory. For COM-N and COM-F architectures, data is first read from the memory. Thereafter, they are typically stored in registers before being fed to the processing units. The amount parallelism is limited here due to constraints in the bandwidth, number of available registers and processing units.

Computation requirements include data alignment and the ability to implement complex functions efficiently. Data alignment is required for all architectures. However, CIM-A and CIM-P classes perform computations directly on the data residing inside the memory, and hence, the robustness and performance are impacted more by data misalignment. Note that performing a data alignment cannot be handled by the host processors in in-memory computing architectures due to a far communication distance, while adding additional logic inside the memory core to handle this is also not trivial. It requires an area overhead to temporary store operands and do the alignment with CMOS logic. For other classes, the impact of data alignment is less severe; nevertheless, data misalignment can cause a performance degradation in other classes as well.

As the primitive operations in CIM-A and CIM-P are limited, architectures in these classes face challenges in computing complex functions such as arithmetic operations with integer or floating point numbers. As a result, a lot of primitive operations are required to realize such complex functions, if even possible. For example, a multi-bit addition in CIM requires multiple single-bit addition as primitive operations and communication operations between these single-bit additions [26].On top of that, each primitive step that involves a write operation in a memristor based CIM architecture suffers from a high latency due to its high write time. In addition, current CIM-P architectures require a high cost to implement a diverse set of arithmetic operations as their efficiency today is mainly limited to bitwise logical operations and matrix vector multiplications. Moreover, providing complex functionality using peripheral circuits in CIM-P is difficult, due to limited available area on the memory core. Note that despite these drawbacks, the performance can be still high when sufficient parallelism is exploited, e.g., by operating on multiple crossbars in parallel. Furthermore, data doesn't have to be transferred to the main processor and hence, the energy and performance can be improved. In COM-N and COM-F, computations are performed by CMOS circuits which contain mature, optimized and if needed, dedicated functional units. However, the main bottleneck comes from the many additional data transfers through the memory hierarchy.

Available bandwidth specifies how much data can be transferred between the computational and storage units. This metric is important as it affects the amount of parallelism that can be exploited. The available bandwidth is considered as similarly as bandwidth specification of multiple level in the memory hierarchy; hence it includes four ranges: max (TBs), high (10 GBs), medium (GBs) and low (MBs) [12]. Note that these terms are used for nowadays' memory technology as the exact bandwidth values are subject to change with new or different technologies. CIM-A architectures may exploit the maximum bandwidth, as operations happen inside the memory array. CIM-P architectures have a bandwidth range from high to max, depending on the complexity of the peripheral circuitry. Note that the peripheral circuits can be complex, e.g., when large customized sense amplifiers are used. Therefore, the placement of such sense amplifiers may be limited due to area constraints. In such cases, still a relative high bandwidth can be realized. For COM-N, the bandwidth is bounded by on-chip interconnections between the memory core and extra logic circuits; for example, in Hybrid Memory Cube [97] the bandwidth is limited by the number of TSVs and available registers. This bandwidth for TSV is considered high in comparison with COM-F, where the bandwidth is even lower due to off-chip interconnections [128].

The four classes (i.e. CIM-A, CIM-P, etc.) also require the **memory design efforts** to obtain a functional memory. In some cases, it is very difficult (or may be even practically impossible) to modify the cells, array, periphery and controller. CIM-A architectures require a redesign of the cell in order to make the computing feasible. Re-characterizing the cell requires a huge effort and induces a huge cost. Other classes, except for hybrid CIM-P, do not require this modification due to the usage of standard memory cells. In terms of changes in the periphery, CIM-P architecture require complex read-out circuits as the output value of two or more accessed cells may end up in multiple levels. Moreover, complex peripheral circuits (i.e., ADC, DAC) limit the scalability when they exhibit internal bottlenecks and could also dominate the area of the memory core when the memory sizes are small. Hence, CIM-P is mainly useful for larger sizes. Other classes, except for hybrid CIM-A, can utilize existing optimized read-out circuits, and hence do not require modifications in the periphery. In terms of memory controller, the complexity reduces from high to low for CIM-A, CIM-P, COM-N and COM-F, respectively. CIM-A architectures require a complex controller as it is responsible for both controlling the crossbar (consisting of a large number of states, each controlling different voltage drivers) and handling data transfer (which involves the usage of buffers/registers to store temporary values). CIM-P architectures have relatively simpler controllers as the computations are constructed in a similar manner as for conventional memory (read/write) operations. The difference is that they typically have to deal with more in-memory operations. COM-N and COM-F architectures utilize the memory in a conventional way, and hence, standard memory controllers

	Data Movement outside memory core	Computation requirements		Available bandwith	Memory design efforts		
		Data Alignment	Complex function		Cells & array	Periphery	Controller
CIM-A	No	Yes	High latency	Max	High	Low/medium	High
CIM-P	No	Yes	High cost	High-Max	Low/medium	High	Medium
COM-N	Yes	NR	Low cost	High	Low	Low	Low
COM-F	Yes	NR	Low cost	Low	Low	Low	Low

NR: Not Required

Table 2: Comparison among Architecture Classes in terms of Data Movement, Computation Requirements, Available Bandwidth and Memory Design Efforts

	Endurance requirement	Maturity			Applications
		Software support	Technology	Development	
CIM-A	High	Emerging		Simulation	Data Intensive - Computational Complexity
CIM-P	Medium	Emerging		Simulation	Data Intensive - Bitwise operations
COM-N	Medium	Commercialized		Fabricated	General-purpose and Application-specific
COM-F	Low	Common Practise		Fabricated	General-purpose

Table 3: Comparison among Architecture Classes in terms of Endurance Requirement, Maturity and Applications

can be used.

Endurance requirement specifies how many write operations can be performed before the memory of the architecture starts to fail. A memory that needs a higher number of writes will have a lower lifetime when both have technology-wise the same endurance. Three ranges can be specified for the architectures: a high endurance requirement (i.e. much higher than DRAM endurance 10^{15} [134]); a medium endurance requirement approximately equal to DRAM endurance; and a low endurance requirement much less than the DRAM endurance. CIM-A has in general a high endurance requirement due to the need of multiple write steps to perform simple Boolean functions. CIM-P has a lower endurance requirement as operations are performed during read operations [130]. Nevertheless, results have to be still written back to the memory in order to perform complex functions. As CIM-A and CIM-P architectures are typically based on emerging devices such as memristors, their endurance could be a potential issue. Similarly to CIM-P, COM-N architectures operate closely to the memory and have to write back the results to the main memory due to the absence/limited number of registers and caches. In contrast, COM-F architectures have a much lower endurance requirement as computations are performed using CMOS and the results of the operations are rarely written back to the main memory due to the usage of registers and caches.

Maturity does not only refer to how feasible/reliable the memory technology is, but also how much software support exists and the current development status of the architectures in the classes. As CIM-A and CIM-P are relatively new concepts and typically resistive based, a lot of work has still to be done to realize these architectures both from a hardware and software point of view. Resistive memories and non-volatile memories in general are typically under research development. For example, the limited endurance puts a constraint on the amount of computations that can be performed in resistive based CIM-A architectures. On the software side, programming languages, compilers, simulators still need to be developed in general for these architectures. In COM-N class, several architectures have been prototyped in the industry and therefore, they are more mature than CIM-A and CIM-P. Architectures in COM-N have also more software support as they are equipped with tool chains that allow product development on these architectures; for example, Micron's automata processor is already commercialized and is programmed in Automata Network Markup Language (ANML) [92]. COM-F architectures are today's conventional von Neumann architectures. They have the highest maturity from both technological point and software support. With respect to the development status, CIM-A and CIM-P architectures mostly are verified using simulations, either cycle-accurate simulations [4, 11] or circuit verification simulation (i.e., HSpice). COM-N and COM-F architectures are further developed; they have been demonstrated

in prototypes and commercial products [97, 100]. In general, COM architectures are more mature than CIM architectures.

Applications that run effectively on the architectures are also described in Table 3. In general, CIM architectures can be more efficient than COM architectures for certain data intensive applications as they are less affected by the memory bottleneck. For CIM-A architectures and several CIM-P architectures (e.g., Pinatubo, CIMA, STT-CiM), there are currently limited types of operations can be efficiently performed on these architectures; hence, limited range of applications can be mapped on these architectures. For example, CIM-A architectures focus more on arithmetic operations such as matrix multiplication [45], parallel addition [26]. CIM-P architectures focus on bulk bitwise applications such as database processing, graph processing, image processing, security and biosequencing application [2, 7, 44, 77, 105, 106]. COM-N architectures are used for both general-purpose and application-specific. A limited number of COM-N architectures are considered as general purpose computers such as FlexRAM [59] and SM [81]. Other COM-N architectures targets specific applications such as vector processing (e.g., VIRAM [56], DIVA [24], AMC [91], etc.), automata processing (D-AP [92]), and neural computation (DDN [19]). COM-F architectures are mostly designed for general-purpose applications.

3 COMPUTATION-IN-MEMORY - ARRAY (CIM-A)

The CIM-A class contains mostly resistive computing architectures that use memristive-based logic circuits [25] to perform computations and resistive RAM (RRAM) as memory technology. The resistive logic circuits may implement different design styles such as stateful logic [73], IMPLY [70], MAGIC [69], CRS-based logic [111], etc. These design styles can be further classified, as presented in [102]. In addition to resistive computing, computations can be performed using DRAM cells as demonstrated in [76] which will be explained later.

Few architectures have been proposed in this class; they are Complementary Resistive Switch (CRS) [111], Computation-in-Memory (CIM) [26, 41, 43], Memristive Memory Processing Unit (MPU) [49], Programmable Logic-in-Memory Computer (PLiM) [37], ReRAM based VLIW architecture (ReVAMP) [8], A DRAM-based Reconfigurable In-Situ Accelerator with a 3T1C cell design (DRISA-3T1C) [76]. Most of the architectures, except for REVAMP, have similar organizations. They contain a memristor crossbar (except for DRISA-3T1C) that is used for both storage and computation and a controller that applies the voltages to the memory array. Each architecture uses a different logic style and controller; for example, CRS, MPU, and PLiM use CRS-based logic, Memristive-Aid loGIC (MAGIC), and majority logic, respectively, while CIM can use any logic styles. ReVAMP uses a different architecture and integrates the resistive memory in a pipelined processor in which the memory replaces both the cache and register file. It optimizes traditional pipelined processors by combining the execution, memory and write-back in a single stage. DRISA-3T1C contains a DRAM memory array and performs NOR instructions by reading two rows simultaneously and writing the results back via the sense amplifier to another row. During the read, the capacitances of the accessed cells will discharge the bitline via a transistor when one or both cell values are high; only when both capacitance values are zero the bitline remains high. As examples, we only describe the CRS and ReVAMP architectures next in more detail; they are the latest proposed architectures that represent a basic CIM-A and hybrid CIM-A architectures, respectively. Due to page limitations, only one representative figure is used to describe each architecture.

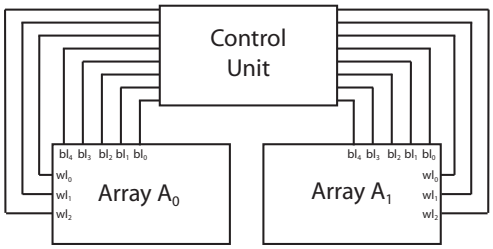


Figure 5: Complementary Resistive Switch-logic Crossbar Array (CRS) [111]

3.1 Basic CIM-A architecture

Complementary Resistive Switch (CRS) architecture was proposed in 2014 by A. Siemon, et al., from RWTH Aachen University [111]. It is a memristor based architecture that exploits data level parallelism using implication logic. The architecture consists of multiple crossbars and a control unit (as shown in Fig. 5 [111]). The crossbar stores and performs logic operations using CRS cells; a CRS cell consists of two resistive switches or resistive RAMs. The control unit distributes signals to the intended addresses (wordlines and bitlines) to perform operations on the crossbars.

The crossbar is controlled by a sequence of operations including: write-in (WI), read-out (RO), write-back (WB), and compute (CP). Before the operations can be performed, the crossbar part used for computation is once entirely reset to a logic value 0. The WI operation writes a logic value into a memristor. The RO operation reads a logic value from a cell; the logic output value is determined by the sense amplifier. The RO operation is destructive and changes the value of the memristor to logic value 1. The task of the WB operation is to recover the destroyed value. Finally, the CP instructions are used to execute the implication logic gates [78, 111]. The data transfer between CRS cells is carried out through the control unit using a RO and WB operations; in other words, the control unit reads a value of the source CRS cell and writes this value into the destination cell.

In addition to the general characteristic of CIM-A described in Table 2 and 3, CRS has the following advantages:

- It is less impacted by the sneak path currents due to the usage of CRS cells. The cell's resistance is always equivalent to high resistance, hence, sneak path currents are eliminated. However, variations in resistances will make such paths practically unavoidable unless a 1T2R cell is used.
- CRS logic requires fewer cells to perform computations than Fast Boolean Logic (FBL) [129].

However, it also has the following limitations:

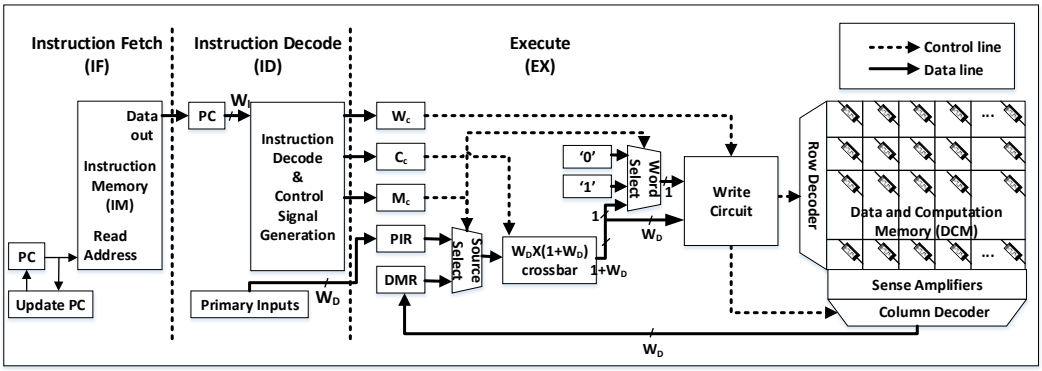


Figure 6: ReRAM based VLIW architecture (ReVAMP) [8]

- The latency of the primitive functions varies and requires read-out instructions to determine the voltages that have to be applied.
- The RO operation is destructive, hence, a WB operation is required after each RO operation, which increases the latency and energy of computations.
- The data tranfer method is indirect as it is based on the read-out and write-back scheme. As all cells have high resistance, direct copying of cells in the crossbar is not applicable.
- The control unit imposes a high overhead as it is responsible for both controlling the crossbar (requiring a large number of states) and transferring data (which involves the usage of buffers/registers to store temporary values).
- The architecture requires additional compiling techniques and tools to convert conventional Boolean logic functions to implication logic.

This architecture was only evaluated at circuit level using adders. Therefore, it is hard to make general conclusions on the performance and the applicability of this architecture.

3.2 Hybrid CIM-A architecture

ReVAMP was proposed in 2017 by D. Bhattacharjee, et al., from Nanyang Technological University [8]. It is a memristor based architecture that exploits data parallelism using majority logic. The architecture consists of an Instruction Fetch (IF), Instruction Decode (ID), and Execute (EX) stage (as shown in Fig. 6 [8]). The IF block fetches instructions from the Instruction Memory using the program counter (PC) as address, and puts the resulting instruction in the Instruction Register (IR). The ID block decodes the instruction and generates control signals which are placed in the control registers of the EX block. The EX stage finally executes the instruction.

The IF and ID stages are similar to those of the traditional five-pipelined RISC architecture. The IF stage includes an Instruction Memory (IM) and a Program Counter (PC). The ID stage contains registers (IR and Primary Inputs), and an Instruction Decode and Control Signal Generation. The EX stage consists of several registers (i.e., Data Memory Register (DMR), Primary Input Register (PIR), Mux control (M_c) register, Control (C_c) register, Wordline (W_c) register), as well as a crossbar interconnect, wordline select multiplexer, data Source Select multiplexer, and a Write circuit to control the crossbar that stores data. Once an instruction is fetched and decoded in IF and ID, respectively, the control registers in EX are filled with suitable values. These values control the multiplexers that are responsible for applying the right control signals to the crossbar. Depending on the operation, primary inputs from PIR or data retrieved from the crossbar stored in DMR can be used for the next operation. The crossbar interconnect permutes the inputs and control signals (indicated by C_c) to generate the voltages that need to be applied to the memory crossbar. The Write circuit applies these voltages to the targeted wordline address (indicated by W_c).

In addition to the general characteristic of CIM-A described in Table 2 and 3, ReVAMP has the following advantages:

- The data transfer may include direct (within the crossbar based on copying resistance values) and indirect (based on read-out/write-back) schemes.
- The crossbar is based on only one device per cell, resulting in a more compact architecture as compared with other architectures which make use of two devices per cell (i.e., Complementary Resistive Switch CRS [111]).
- The architecture does not suffer destructive reads as is the case for CRS architecture [111], hence the write energy might be less due to the absence of a write back operation.

However, it also has the following limitations:

- The latency of majority primitive functions varies depending on the functional complexity; in addition, before any operations are applied to the cells, these cells first have to be read-out in order to determine the appropriate control voltages.
- The architecture has to deal with sneak path currents. Possible solutions as mentioned above.

- The EX stage is complex as it integrates both the control signals for memory and computations. Therefore, it is not easy to pipeline this architecture, as the EX stage will consume more time than the other stages; i.e., the stages IF, ID, and EX are not balanced.
- The architecture requires additional compiling techniques and tools to convert conventional Boolean logic functions to majority logic gates.

The architecture is simulated and evaluated using EPFL benchmarks [5] and compared against PLiM [37], which is based on a resistive memory with the same logic style.

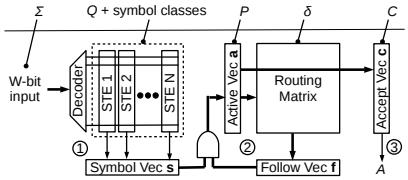


Figure 7: General Architecture for Automata Processor [133]

CIM-P architectures, respectively.

4 COMPUTATION-IN-MEMORY - PERIPHERY (CIM-P)

The CIM-P class consists of architectures which perform computations during read-out operations (i.e., two or more word lines are activated simultaneously) using special peripheral circuitry. Such operations are typically analog in nature. As there are less restrictions on the functionality of the cell, various memory technologies can be used in this category such as DRAM, SRAM and non-volatile memory technologies.

A medium number of architectures have been proposed in this class: Resistive Associative Processor (ReAP) [132], A Processing-in-Memory Architecture for Neural Network Computation in ReRAM-based Main Memory (PRIME) [20], A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic (ISAAC) [107], In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology (Ambit) [106], A Processing-in-Memory Architecture for Bulk Bitwise Operations (Pinatubo) [77], In-Memory Intelligence (IMI) [31], Compute Caches (Compute$) [1], A DRAM-based Reconfigurable In-Situ Accelerator with 1T1C design (DRISA-1T1C) [76], Computation-in-Memory Accelerator (CIMA) [25], Computing in Memory Spin-Transfer Torque Magnetic RAM (STT-CiM) [53], Cache Automaton (S-AP) [117], Neural Cache (Neural$) [28], RRAM Automata Processor (R-AP) [133], Data Parallel Processor (DPP) [36].

These architectures fundamentally perform computations in the same way by activating multiple rows simultaneously in the memory and using generally specialized sense amplifiers and/or ADC converters to get the results. ReAP performs computations by implementing Content-addressable-Memory (CAM) operations using look-up-tables (LUTs). PRIME and ISAAC perform vector-matrix multiplications for neural applications. Ambit, IMI, Compute$, DRISA-1T1C, Pinatubo, CIMA, STT-CIM, Neural$ and DPP perform computations using customized sense-amplifiers only; Ambit, IMI and DRISA-1T1C use DRAM, Compute$ and Neural$ uses SRAM, while the rest is based on non-volatile memory. These architectures can only perform logical operations except for IMI, DRISA-1T1C, Neural$ and DPP which also perform more complex functions by having additional logic inside the peripheral circuits. S-AP and R-AP implement inner product operations in automata processors. S-AP is implemented using SRAM technology while R-AP uses non-volatile memory. As examples, we only describe the R-AP and DPP architectures next in more details; they are the latest proposed architectures that represent basic CIM-P and hybrid

4.1 Basic CIM-P architecture

R-AP was proposed in 2018 by J. Yu, et al., from Delft University of Technology [133]. The architecture targets an automata processor which exploits data level parallelism by performing computations using state machines. An automata processor contains two main components: the State Transition Elements (STEs) and the routing matrix; the STE stores the accepting states, while the routing matrix stores the state transitions as shown in Fig. 7 [133]. The automata processor accepts one input symbol at a time, generates next active states and decides whether a complete input string is accepted or not.

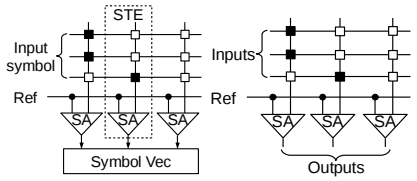
The architecture consists of STEs and a routing matrix which are implemented using RRAM technology. Each RRAM column corresponds to an STE which stores the accepting states in RRAM cells, as shown in Fig. 8a [133]. The input symbol is fed to all the STEs simultaneously. The sense amplifiers collect a dot-product results of a vector-matrix multiplication. The output of the STE together with the routing matrix are used to determine the next active states, as shown in Fig. 8b [133]; this process is carried on until all input symbols are processed. In case the one or more final active states are part of the acceptance states, it means that the input string has been matched with the corresponding pattern of the acceptance state. Note that data transfer inside the automata processor is carried out using the routing matrix.

In addition to the general characteristic of CIM-P described in Table 2 and 3, R-AP has the following advantages:

- The architecture is used as a read-favoured accelerator, which has a positive impact on the endurance due to infrequent use [16, 123]. Only when the automata changes, the STEs and routing matrix have to be updated.
- Automata processing can be used to perform both logical and arithmetic operations in general.
- Data can be transferred using both direct and indirect schemes.
- The architecture uses non-volatile memory, hence consumes low energy and has a small footprint.
- The automata processing techniques and tooling are quite mature, hence it is feasible to explore many applications using automata processing.

However, it also has the following limitations:

- The modified peripheral circuitry (row drivers) might pose high overhead in the memory system.



(a) Used as STEs (b) Used as routers

Figure 8: Resistive RAM Automata Processor (R-AP) [133]

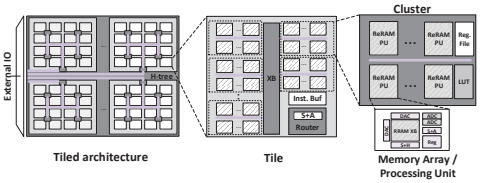


Figure 9: Data Parallel Processor (DPP) [36]

- The architecture requires additional compiling techniques and tools to perform conventional Boolean logic functions using automata processing.

The architecture has been validated using circuit level simulations and evaluated against S-AP [117].

4.2 Hybrid CIM-P architecture

DPP was proposed in 2018 by D. Fujiki, et al. from University of Michigan [36]. DPP is a RRAM-based architecture that exploits instruction and data level parallelism by performing computations using a combination of RRAM-based dot-product operations and LUTs. The architecture consists of multiple RRAM tiles connected as an H-tree; each tile has multiple clusters and some logic units (as shown in Fig. 9 [36]). Tiles and clusters form a SIMD-like processor that performs the parallel operations. The architecture is considered as a general purposed architecture as it can perform all primitive functions such as logical, arithmetic, shift and copy operations.

In addition to clusters, each tile has several units to support the computations including instruction buffer, Shift and Add (S+A), and router. Each cluster additionally has one or more computational units; they are Shift and Add (S+A), Sample and Hold (S+H), DAC and ADC, a LUT and register file (as shown in the right part of Fig. 9). While reading from the high latency RRAM, other units are simultaneously used for processing. Therefore, the S+H is used to read data (in the form of a current) from the RRAM array and temporarily store it. Once that data is needed, it is fed to an ADC to convert the analog value to a digital value. The S+A is used to perform carry propagation in a multiple-bit addition. DAC is used to apply a digital value to the RRAM array with an appropriate control voltage. Some complex functions that cannot be realized with these units are performed using LUTs and register file in each cluster. Data transfer can be performed by enabling two memory rows for direct copy operations, or using the buffers and read-out operations for indirect copy operations.

In addition the general characteristic of CIM-P described in Table 2 and 3, DPP has the following advantages:

- Computations include both logical operations and simple arithmetic operations (i.e., addition, multiplication).
- Data can be transferred using both direct and indirect schemes.
- The architecture uses non-volatile memory, hence consumes low energy and has a small footprint.
- This architecture is claimed to be general purpose, hence it can exploits existing instruction set, compiling techniques and tools, as well as applications.

However, it also has the following limitations:

- The architecture uses non-volatile memory as main memory, which may impact the life time due to limited endurance [16, 123].
- As the sense amplifies are complex, a trade-off between area and bandwidth has to be made.

The architecture potential was simulated and evaluated against CPU Intel Xeon E5-2697 using a subset of PARSEC benchmarks [10] and against GPU NVIDIA Titan XP using Rodinia benchmarks [17].

5 COMPUTATION-OUT-MEMORY - NEAR (COM-N)

The COM-N class consists of architectures that perform computation using additional logic units outside the memory core but inside the memory SiP. These architectures were proposed in the past and evolved through different memory technologies ranging from conventional DRAM, embedded DRAM to emerging memory technologies such as RRAM.

Many architectures have been proposed in this class: Vector Intelligent RAM (VIRAM) [56, 65, 67, 95], Active Page(A-Page) [94], Advance Intelligent Memory System (FlexRAM) [60], Modular Reconfigurable Smart Memories (S-Mem) [81], Data-intensive Architecture (DIVA) [23, 24], Hybrid Memory Cube (HMC) [54, 97], Active Memory Cube (AMC) [91], Micron Automata Processor (D-AP) [92], A machine-learning supercomputer (DDN) [19], An Architecture for Accelerated Processing Near Memory (DRAMA) [30], High-Bandwidth Memory (HBM) [57, 72, 80, 115], A Near Data Computing Architecture using Non-Volatile Memory (ProPRAM) [124], Resistive GP-SIMD (ReGP) [88], HMC Instruction Large Vector Extensions (HIVE) [3].

These architectures mainly differ as a consequence of using different technologies. VIRAM, FlexRAM, SM, DIVA, and DRAMA are based on embedded DRAM technology and try to integrate processing units near the main memory; FlexRAM integrates multiple single-core processors with caches, SM a reconfigurable processor, VIRAM and DIVA a vector processor, and DRAMA a Coarse-grain reconfigurable accelerators. A-Page is based on reconfigurable DRAM architecture which integrates conventional DRAM into an FPGA; it implements the reconfigurable logic near the main DRAM memory. HMC, AMC, HBM, and HIVE are based on 3D-stacked DRAM; HMC and HBM support general computing, while AMC

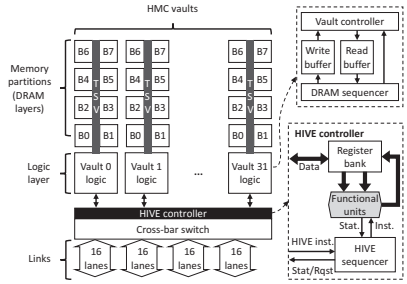


Figure 10: HMC Module with HMC Instruction Large Vector Extensions (HIVE) [3]

and HIVE are optimized for VLIW and vector processing, respectively. DaDianNao, D-AP, ReGP, and ProPRAM utilize logic units that are located near the memory. DaDianNao and D-AP implements a neural network and an automata processor, respectively with very simple logic units inside the conventional DRAM. ReGP integrates a simplified SIMD processor near non-volatile memory. ProPRAM utilizes existing logic units near the non-volatile memory to perform simple computations such as addition and logical operations. As an example, we only describe the HIVE and ReGP architectures next in more details; They are the most recent architectures proposed in COM-N class.

HIVE was proposed in 2016 by M. A. Z. Alves, et al., from Federal University of Rio Grande do Sul [3]. HIVE is a Hybrid Memory Cube (HMC) [54, 97] based architecture that performs large vector operations inside the logic die of a HMC. The architecture consists of a host processor and a HMC module that is extended with a HIVE, as shown in Fig. 10 [3]. The host processor, not shown in the figure, is a pipelined-like architecture with six stages; it fetches, decodes, renames, dispatches, executes and commits a sequence of instruction. If an instruction fragment has to be executed using in-memory instructions, the processor diverts the instruction fragment to the HMC module. HMC module executes the fragment and returns the result back to the processor.

HMC module consists of multiple DRAM layers, logic vaults, HIVE controller, a crossbar switch and multiple-lane links to host processor (as shown in the left side of Fig. 10). The data is stored in multiple DRAM layers and retrieved by the HIVE. The HIVE controller contains a register bank, functional units and a HIVE sequencer (as shown in the bottom right of Fig. 10). The logic vaults contains a vault controller, write and read buffer, and a DRAM sequencer (as shown in the top right of Fig. 10). Once the HIVE sequencer receives an instruction, it locks the involved memory address space; if the memory has already been locked, the requested instruction returns a fail status to processor; otherwise, a memory synchronization occurs by flushing related cache data into DRAM. The logic vaults and HIVE subsequently execute the instructions by reading data to read buffers and register bank, performing operations using functional units, and (optional) storing into memory using write buffers. The operations in HIVE are based on vector operations that operate on 8KB of data at a time executed by the 32 logic vaults and HIVE functional units. As the amount of data is large, a DRAM sequencer and HIVE sequencer schedule these operation accordingly. The results can be collected in register banks and sent back to the host processor through the crossbar switch and links.

In addition to the general characteristic of COM-N described in Table 2 and 3, HIVE comes with the following advantages:

- The parallelism is high due to vector processing on 8KB of data.
- The architecture uses HMC which is mature, commercialized and has some advantages such as high performance, high bandwidth, low power, high density [54, 97].

However, it also has the following limitation:

- The architecture has a complex HMC module which has a control, communication and programming overhead.

The architecture is simulated and evaluated using some integer (vector search and memory reset/set operations) and floating-point (vector sum, matrix stencil, and matrix multiplication) kernels against three baseline platforms; both HIVE and baseline platforms are based on the Intel Atom processor. Like HIVE, the three baseline platforms have also additional processing capacities; for the baseline platforms they are as follows: 1) HMC instructions using HMC 2.0 memory [47] (HMC+HMC), 2) 128-bit SSE instructions with DDR-3 1333 modules (SSE+DDR) and 3) 128-bit SSE instructions with HMC 2.0 (SSE+HMC).

ReGP was proposed in 2016 by A. Morad, et al., from Technion-Israel Institute of Technology [88]. ReGP is a RRAM memory based architecture that exploits data parallelism by attaching a SIMD-like processing unit to the resistive memory, as shown in Figure 11 [88]. The architecture consists of a sequential processor (which is a conventional processor), its L1 and LLC cache, shared memory array and SIMD processor. The sequential processor executes traditional code and controls the SIMD processor in a master-slave mode. The SIMD processor executes parallel instructions on the data stored in the shared memory array.

The SIMD processor contains multiple processing units (PUs), a sequencer and a Network on Chip (NoC) with reduction tree. Each PU contains registers, a single bit full-adder and a function generator to perform arithmetic and logical operations. The sequencer receives instructions from the sequential processor and assigns them to PUs. The PUs load data from the shared memory array and perform the requested operations. If required, the NoC and reduction trees are used to perform more complex functions.

In addition to the general characteristic of COM-N described in Table 2 and 3, ReGP comes with the following advantages:

- The parallelism is high due to multiple parallel processing units.
- The architecture uses non-volatile memory, hence consumes a low amount of energy and has a small footprint.
- The architecture can reuse compilers, programming languages and tools from SIMD architectures.

However, it also has the following limitation:

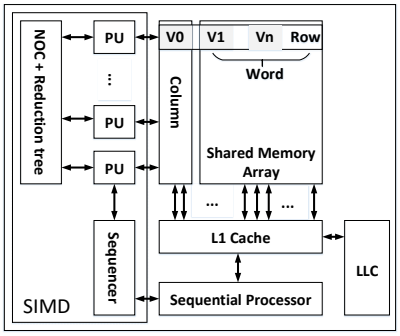


Figure 11: Resistive GP-SIMD (ReGP) [88]

- The operations within the processing units are simple; complex functions such as floating point operations can cause a high overhead.

The architecture is simulated and evaluated against CMOS GP-SIMD [87] using a benchmark for dense matrix multiplications [86].

6 DISCUSSION

This section aims to first evaluate the completeness of the proposed classification. Thereafter, we compare it with existing work in the field.

6.1 Completeness

The proposed classification presented in Fig. 3 is complete and comprehensive. These points can as follows be proven: (i) theoretically, due to the exploration of all the possible classes derived from the classification metrics, and (ii) practically, by mapping all existing memory-centric architectures on the classification.

Theoretically, the classification contains four main classes derived from the "computation location" (first metric); both inside and outside, approximately close or distant from the memory core. Moreover, the second metric consists of both charge-based and non-charge-based memories. Finally, the parallelism metric ranges from instruction, to data and task levels. Each metric is in it self complete, and therefore, the entire classification is complete. The classification does not only contain the existing solutions, but also highlights the potential future solutions that can be further explored (e.g., classes in blue spaces in Fig. 3). Note that hybrid architectures are also covered in this classification. For example, a conventional architecture (COM-F) with accelerator in CIM-P class (e.g., ReAP, ISAAC, CIMA) is considered a hybrid architecture, i.e., a COM-F/CIM-P hybrid.

Practically, it contains an overview of the most existing computer architectures and places them in perspective. In addition, the classification can be used to illustrate the past and future trends (see Fig. 4). Moreover, it clearly depicts a shift from conventional processor-centric architectures towards memory-centric architectures based on emerging technologies (3D stacking, RRAM, etc.).

6.2 Related work

Comparison with traditional/processor-centric architecture classifications: conventional classifications like Flynn's [34], Skillicorn's [113] and Shami-Hemani's [108] classification are quite comprehensive and were considered complete at the time they were published. However, these classifications focus on processor-centric architectures and, hence they can only be used to classify conventional architectures (i.e., architectures in COM-F class). Aside from the above mentioned classifications, some publications on COM-N class have presented intensive architectural reviews [22, 109, 112]. However, they have a restricted focus on near-memory-processing architectures based on 2D, 2.5D, and 3D-stacked DRAM. Signh's classification [112] is the most recent work that provides a review of near-memory computing architectures, i.e. COM-N architectures. It classifies architectures mainly based on the memory hierarchy and processing type (e.g., programmable unit, fixed functional unit and reconfigurable unit). Moreover, it evaluates the architectures based on multiple characteristics of memory, processing, evaluation tools, interoperability, and application domains. However, the classification is not easy to use as the metrics are not systematic. Furthermore, it is not clear if the classification is complete and if it covers all ranges of near-computing architectures. Last but not least, in comparison with the aforementioned classifications, our proposed classification goes one step further to cover both conventional and emerging architectures by having the additional classes CIM-A and CIM-P. Moreover, the proposed classification is so broad that several of its classes are not explored yet. New architectures in these unexplored areas can be easily added to the classification. In addition, our proposed classification uses three selective metrics which create distinctive and easy-to-use terminologies, classes and sub-classes.

Comparison with recent/emerging architecture classifications: recent surveys and classifications for emerging architectures have been proposed by Mittal [85] and Reuben [102]. Mittal's classification only tries to link architectures with their applications. Specifically, the classification discusses three unconventional architectures: processing-in-memory, machine learning and neural network based architectures using RRAM. They mostly focus on applications containing dot-product operations in the RRAM crossbar. This classification is not complete, as RRAM in particular and emerging memory technology in general can also be used to implement other functions such as bitwise logic operations [77, 130], arithmetic operations using implication logic [69, 110], Boolean logic[114, 129], etc. Reuben's classification classifies existing resistive logic design methods into three classes: in-memory, near-memory and out-of-memory computing. The near-memory class has three sub-classes without identifiers (e.g., they are based on how data moves out of the memory array; this includes data movements (1) between consecutive logic levels, (2) for computing each Sum-of-Products, (3) for computing each logic gate). This classification, however, tries to redefine the terminologies without defining clear generic metrics for each class. Instead, each class uses different criteria to distinguish between their sub-classes. Therefore, it is not a systematic and comprehensive classification, which makes it difficult to use in identifying and exploring architectures. Moreover, it is difficult

to judge if the classification is complete. Furthermore, the classification focuses only on resistive memories, while other emerging memory technologies are also promising. Overall, both Mittal and Reuben classifications are not complete and comprehensive enough to classify all architectures.

6.3 Future directions and challenges

This section mainly highlights future trends in memory-centric computing. We observe that computation in memory is one of the solutions to alleviate the memory bottleneck for certain applications. Both the bandwidth between the core and main memory as well as the overall latency can be significantly improved by computation-in-memory architectures. In addition, implementing computation-in-memory in main memory (using DRAM, memristive devices) is more feasible than using on-chip memory (SRAM) due to technology limitations and economical feasibility. Furthermore, the two main directions that are currently explored are CIM-A and CIM-P, in which CIM-P is more feasible than CIM-A due to the complex underlining memory technology. Last but not least, computation-in-memory (CIM-A and CIM-P) do not make conventional architectures obsolete; in fact, multicore architectures with caches are relevant for applications with high data locality, while computation-in-memory architectures can be only used efficiently for certain specific applications.

It is worth mentioning that the focus of this paper is proposing a unified terminology and classification instead of presenting a survey. In our future work we will present a survey that intensively discusses all architectures.

7 CONCLUSION

In this paper, we have proposed a classification using three metrics: computational location, memory technology and level of parallelism. We have used the most important metric, i.e. computational location, to describe and evaluate the four main classes (and the selected architectures therein). The work shows that architectures do not only require to be memory bottleneck free, but also energy and area efficient. In order to accomplish that, the architectures must be implemented with the right technologies. The relationship and dependency between the architecture and technologies becomes stronger for memory-centric computing architectures. This work also showed that new architectures typically emerge after new technology developments (e.g., introduction of 3D stacking and RRAM). Our classification unifies the prior work and aims to provide a comprehensive and unique terminology for memory-centric computing architectures. Finally, the classification does not only present an overview of existing architectures, but also predicts the potential of future architecture variants, including hybrid architectures that may combine different strengths of the different classes.

ACKNOWLEDGMENTS

The results presented in this paper have been obtained in the framework of the project "Computation-in-memory architecture based on resistive devices" (MNEMOSENE), which has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 780215.

REFERENCES

[1] Shaizeen Aga, Supreet Jeloka, Arun Subramaniyan, Satish Narayanasamy, David Blaauw, and Reetuparna Das. 2017. Compute caches. In *High Performance Computer Architecture (HPCA), 2017 IEEE International Symposium on*. IEEE, 481–492.

[2] Junwhan Ahn, Sungpack Hong, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. 2016. A scalable processing-in-memory accelerator for parallel graph processing. *ACM SIGARCH Computer Architecture News* 43, 3 (2016), 105–117.

[3] Marco AZ Alves, Matthias Diener, Paulo C Santos, and Luigi Carro. 2016. Large vector extensions inside the HMC. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2016*. IEEE, 1249–1254.

[4] Marco Antonio Zanata Alves, Carlos Villavieja, Matthias Diener, Francis Birck Moreira, and Philippe Olivier Alexandre Navaux. 2015. SiNUCA: A Validated Micro-Architecture Simulator.. In *HPCC/CSS/ICESS*. 605–610.

[5] Luca Amarú, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. 2015. The EPFL combinational benchmark suite. In *Proceedings of the 24th International Workshop on Logic & Synthesis (IWLS)*.

[6] John Barth, Don Plass, Erik Nelson, Charlie Hwang, Gregory Fredeman, Michael Sperling, Abraham Mathews, Toshiaki Kirihata, William R Reohr, Kavita Nair, et al. 2010. A 45 nm SOI embedded DRAM macro for the POWERâĎć processor 32 MByte on-chip L3 cache. *IEEE Journal of Solid-State Circuits* 46, 1 (2010), 64–75.

[7] Gary Benson, Yozen Hernandez, and Joshua Loving. 2013. A bit-parallel, general integer-scoring sequence alignment algorithm. In *Annual Symposium on Combinatorial Pattern Matching*. Springer, 50–61.

[8] Debjyoti Bhattacharjee, Rajeswari Devadoss, and Anupam Chattopadhyay. 2017. ReVAMP: ReRAM based VLIW architecture for in-memory computing. In *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 782–787.

[9] Sabpreet Bhatti, Rachid Sbiaa, Atsufumi Hirohata, Hideo Ohno, Shunsuke Fukami, and SN Piramanayagam. 2017. Spintronics based random access memory: a review. *Materials Today* (2017).

[10] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. 2008. The PARSEC benchmark suite: Characterization and architectural implications. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*. ACM, 72–81.

[11] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, Somayeh Sardashti, et al. 2011. The gem5 simulator. *ACM SIGARCH Computer Architecture News* 39, 2 (2011), 1–7.

[12] Evgeny Bolotin, David Nellans, Oreste Villa, Mike O'Connor, Alex Ramirez, and Stephen W Keckler. 2015. Designing efficient heterogeneous memory architectures. *IEEE Micro* 35, 4 (2015), 60–68.

[13] Julien Borghetti, Gregory S Snider, Philip J Kuekes, J Joshua Yang, Duncan R Stewart, and R Stanley Williams. 2010. Memristive switches enable stateful logic operations via material implication. *Nature* 464, 7290 (2010), 873–876.

[14] S. Borkar. 1999. Design challenges of technology scaling. *Micro, IEEE* 19, 4 (Jul 1999), 23–29. https://doi.org/10.1109/40.782564

[15] Rafmag Cabrera, Emmanuelle Merced, and Nelson Sepúlveda. 2013. A microelectro-mechanical memory based on the structural phase transition of VO2. *physica status solidi (a)* 210, 9 (2013), 1704–1711.

[16] Meng-Fan Chang, Ching-Hao Chuang, Min-Ping Chen, Lai-Fu Chen, Hiroyuki Yamauchi, Pi-Feng Chiu, and Shyh-Shyuan Sheu. 2012. Endurance-aware circuit designs of nonvolatile logic and nonvolatile SRAM using resistive memory (memristor) device. In *Design Automation Conference (ASP-DAC), 2012 17th Asia and South Pacific*. IEEE, 329–334.

[17] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W Sheaffer, Sang-Ha Lee, and Kevin Skadron. 2009. Rodinia: A benchmark suite for heterogeneous computing. In *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*. Ieee, 44–54.

[18] E Chen, D Apalkov, Z Diao, A Driskill-Smith, D Druist, D Lottis, V Nikitin, X Tang, S Watts, S Wang, et al. 2010. Advances and future prospects of spin-transfer torque random access memory. *IEEE Transactions on Magnetics* 46, 6 (2010), 1873–1878.

[19] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, et al. 2014. Dadiannao: A machine-learning supercomputer. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 609–622.

[20] Ping Chi, Shuangchen Li, Cong Xu, Tao Zhang, Jishen Zhao, Yongpan Liu, Yu Wang, and Yuan Xie. 2016. PRIME: a novel processing-in-memory architecture for neural network computation in ReRAM-based main memory. In *ACM SIGARCH Computer Architecture News*, Vol. 44. IEEE Press, 27–39.

[21] Gianni Conte, Stefano Tommesani, and Francesco Zanichelli. 2000. The long and winding road to high-performance image processing with MMX/SSE. In *Computer Architectures for Machine Perception, 2000. Proceedings. Fifth IEEE International Workshop on*. IEEE, 302–310.

[22] Joao Paulo C. de Lima, Paulo Cesar Santos, Marco A. Z. Alves, Antonio C. S. Beck, and Luigi Carro. 2018. Design Space Exploration for PIM architectures in 3D-stacked memories. In *Computer Frontier*. ACM, 295–308.

[23] Jaffrey Draper, J Tim Barrett, Jeff Sondeen, Sumit Mediratta, Chang Woo Kang, Ihn Kim, and Gokhan Daglikoca. 2005. A prototype processing-in-memory (PIM) chip for the data-intensive architecture (DIVA) system. *Journal of VLSI signal processing systems for signal, image and video technology* 40, 1 (2005), 73–84.

[24] Jeff Draper, Jacqueline Chame, Mary Hall, Craig Steele, Tim Barrett, Jeff LaCoss, John Granacki, Jaewook Shin, Chun Chen, Chang Woo Kang, et al. 2002. The architecture of the DIVA processing-in-memory chip. In *Proceedings of the 16th international conference on Supercomputing*. ACM, 14–25.

[25] HA Du Nguyen, Jintao Yu, Lei Xie, Mottaqiallah Taouil, Said Hamdioui, and Dietmar Fey. 2017. Memristive devices for computing: Beyond CMOS and beyond von Neumann. In *Very Large Scale Integration (VLSI-SoC), 2017 IFIP/IEEE International Conference on*. IEEE, 1–10.

[26] Hoang Anh Du Nguyen, Lei Xie, Mottaqiallah Taouil, Razvan Nane, Said Hamdioui, and Koen Bertels. 2017. On the implementation of computation-in-memory parallel adder. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 25, 8 (2017), 2206–2219.

[27] P Dudek and SJ Carey. 2006. General-purpose 128/spl times/128 SIMD processor array with integrated image sensor. *Electronics Letters* 42, 12 (2006), 678–679.

[28] Charles Eckert, Xiaowei Wang, Jingcheng Wang, Arun Subramaniyan, Ravi Iyer, Dennis Sylvester, David Blaauw, and Reetuparna Das. 2018. Neural Cache: Bit-Serial In-Cache Acceleration of Deep Neural Networks. *arXiv preprint arXiv:1805.03718* (2018).

[29] Susan J Eggers, Joel S Emer, Henry M Levy, Jack L Lo, Rebecca L Stamm, and Dean M Tullsen. 1997. Simultaneous multithreading: A platform for next-generation processors. *IEEE micro* 17, 5 (1997), 12–19.

[30] Amin Farmahini-Farahani, Jung Ho Ahn, Katherine Morrow, and Nam Sung Kim. 2015. DRAMA: An architecture for accelerated processing near memory. *IEEE Computer Architecture Letters* 14, 1 (2015), 26–29.

[31] Tim Finkbeiner, Glen Hush, Troy Larsen, Perry Lea, John Leidel, and Troy Manning. 2017. In-Memory Intelligence. *IEEE Micro* 37, 4 (2017), 30–38.

[32] Nadeem Firasta, Mark Buxton, Paula Jinbo, Kaveh Nasri, and Shihjong Kuo. 2008. Intel AVX: New frontiers in performance improvements and energy efficiency. *Intel white paper* 19 (2008), 20.

[33] Randall James Fisher. 2003. General-purpose SIMD within a register: Parallel processing on consumer microprocessors. (2003).

[34] M. Flynn. 1966. Very high-speed computing systems. *Proc. IEEE* 54, 12 (Dec 1966), 1901–1909. https://doi.org/10.1109/PROC.1966.5273

[35] GD Fuchs, NC Emley, IN Krivorotov, PM Braganca, EM Ryan, SI Kiselev, JC Sankey, DC Ralph, RA Buhrman, and JA Katine. 2004. Spin-transfer effects in nanoscale magnetic tunnel junctions. *Applied Physics Letters* 85, 7 (2004), 1205–1207.

[36] Daichi Fujiki, Scott Mahlke, and Reetuparna Das. 2018. In-Memory Data Parallel Processor. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 1–14.

[37] Pierre-Emmanuel Gaillardon, Luca Amar, Anne Siemon, Eike Linn, Rainer Waser, Anupam Chattopadhyay, and Giovanni De Micheli. 2016. The programmable logic-in-memory (PLiM) computer. In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 427–432.

[38] Simcha Gochman, Avi Mendelson, Alon Naveh, and Efraim Rotem. 2006. Introduction to Intel Core Duo Processor Architecture. *Intel Technology Journal* 10, 2 (2006).

[39] Jonathan E Green, Jang Wook Choi, Akram Boukai, Yuri Bunimovich, Ezekiel Johnston-Halperin, Erica DeIonno, Yi Luo, Bonnie A Sheriff, Ke Xu, Young Shik Shin, et al. 2007. A 160-kilobit molecular electronic memory patterned at 10 11 bits per square centimetre. *Nature* 445, 7126 (2007), 414.

[40] Beat Halg. 1990. On a micro-electro-mechanical nonvolatile memory cell. *IEEE Transactions on Electron Devices* 37, 10 (1990), 2230–2236.

[41] Said Hamdioui, Koenraad Laurent Maria Bertels, and Mottaqiallah Taouil. 2017. Computing device for âĂIJbig dataâĂİ applications using memristors. US Patent 9,824,753.

[42] Said Hamdioui, Shahar Kvatinsky, Gert Cauwenberghs, Lei Xie, Nimrod Wald, Siddharth Joshi, Hesham Mostafa Elsayed, Henk Corporaal, and Koen Bertels. 2017. Memristor for computing: Myth or reality?. In *Proceedings of the Conference on Design, Automation & Test in Europe*. European Design and Automation Association, 722–731.

[43] Said Hamdioui, Lei Xie, Hoang Anh Du Nguyen, Mottaqiallah Taouil, Koen Bertels, Henk Corporaal, Hailong Jiao, Francky Catthoor, Dirk Wouters, Linn Eike, et al. 2015. Memristor based computation-in-memory architecture for data-intensive applications. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*. EDA Consortium, 1718–1725.

[44] JongWook Han, Choon-Sik Park, Dae-Hyun Ryu, and Eun-Soo Kim. 1999. Optical image encryption based on XOR operations. *Optical Engineering* 38, 1 (1999), 47–55.

[45] Adib Haron, Jintao Yu, Razvan Nane, Mottaqiallah Taouil, Said Hamdioui, and Koen Bertels. 2016. Parallel matrix multiplication on memristor-based computation-in-memory architecture. In *High Performance Computing & Simulation (HPCS), 2016 International Conference on*. IEEE, 759–766.

[46] John L Hennessy and David A Patterson. 2011. *Computer architecture: a quantitative approach*. Elsevier.

[47] HMC. 2018. Hybrid Memory Cube Specification 2.1. http://hybridmemorycube.org/

[48] M Hosomi, H Yamagishi, T Yamamoto, K Bessho, Y Higo, K Yamane, H Yamada, M Shoji, H Hachino, C Fukumoto, et al. 2005. A novel nonvolatile memory with spin torque transfer magnetization switching: Spin-RAM. In *Electron Devices Meeting, 2005. IEDM Technical Digest. IEEE International*. IEEE, 459–462.

[49] Rotem Ben Hur and Shahar Kvatinsky. 2016. Memristive memory processing unit (MPU) controller for in-memory processing. In *Science of Electrical Engineering (ICSEE), IEEE International Conference on the*. IEEE, 1–5.

[50] IBM. 2014. Power 4 - The First Multi-Core, 1GHz Processor.

[51] ITRS. 2010. ITRS ERD report. http://www.itrs.net

[52] Subramanian S Iyer and Howard L Kalter. 1999. Embedded DRAM technology: opportunities and challenges. *IEEE spectrum* 36, 4 (1999), 56–64.

[53] Shubham Jain, Ashish Ranjan, Kaushik Roy, and Anand Raghunathan. 2017. Computing in memory with spin-transfer torque magnetic RAM. *arXiv preprint arXiv:1703.02118* (2017).

[54] Joe Jeddeloh and Brent Keeth. 2012. Hybrid memory cube new DRAM architecture increases density and performance. In *VLSI Technology (VLSIT), 2012 Symposium on*. IEEE, 87–88.

[55] Zhang Jianwu, Zhao Danying, et al. 2008. Survey on microprocessor architecture and development trends. In *Communication Technology, 2008. ICCT 2008. 11th IEEE International Conference on*. IEEE, 297–300.

[56] David Judd, Katherine Yelick, Christoforos Kozyrakis, David Martin, and David Patterson. 2001. Exploiting on-chip memory bandwidth in the VIRAM compiler. In *Intelligent Memory Systems*. Springer, 122–134.

[57] Hongshin Jun, Jinhee Cho, Kangseol Lee, Ho-Young Son, Kwiwook Kim, Hanho Jin, and Keith Kim. 2017. HBM (High Bandwidth Memory) DRAM Technology and Architecture. In *Memory Workshop (IMW), 2017 IEEE International*. IEEE, 1–4.

[58] Ron Kalla, Balaram Sinharoy, William J Starke, and Michael Floyd. 2010. Power7: IBM's next-generation server processor. *IEEE micro* 30, 2 (2010), 7–15.

[59] Yi Kang, Wei Huang, Seung-Moon Yoo, D. Keen, Zhenzhou Ge, V. Lam, P. Pattnaik, and J. Torrellas. [n. d.]. FlexRAM: Toward an advanced Intelligent Memory system. In *2012 IEEE 30th International Conference on Computer Design (ICCD)* (2012). 5–14. https://doi.org/10.1109/ICCD.2012.6378608

[60] Yi Kang, Wei Huang, Seung-Moon Yoo, Diana Keen, Zhenzhou Ge, Vinh Lam, Pratap Pattnaik, and Josep Torrellas. 2012. FlexRAM: Toward an advanced intelligent memory system. In *Computer Design (ICCD), 2012 IEEE 30th International Conference on*. IEEE, 5–14.

[61] Doris Keitel-Schulz and Norbert Wehn. 1998. Issues in embedded DRAM development and applications. In *Proceedings of the 11th international symposium on System synthesis*. IEEE Computer Society, 23–31.

[62] Doris Keitel-Schulz and Norbert Wehn. 2001. Embedded DRAM development: Technology, physical design, and application issues. *IEEE Design & Test of Computers* 18, 3 (2001), 7–15.

[63] Kyosun Kim, Sangho Shin, and Sung-Mo Kang. 2011. Stateful logic pipeline architecture. In *2011 IEEE International Symposium of Circuits and Systems (ISCAS)*. IEEE, 2497–2500.

[64] David Kirk et al. 2007. NVIDIA CUDA software and GPU parallel computing architecture. In *ISMM*, Vol. 7. 103–104.

[65] Christoforos Kozyrakis. 2002. *Scalable vector media-processors for embedded systems*. Technical Report. CALIFORNIA UNIV BERKELEY COMPUTER SCIENCE DIV.

[66] Christoforos Kozyrakis and David Patterson. 2002. Vector vs. superscalar and VLIW architectures for embedded multimedia benchmarks. In *Proceedings of the 35th annual ACM/IEEE international symposium on Microarchitecture*. IEEE Computer Society Press, 283–293.

[67] Christoforos E Kozyrakis, Stylianos Perissakis, David Patterson, Thomas Anderson, Krste Asanovic, Neal Cardwell, Richard Fromm, Jason Golbus, Benjamin Gribstad, Kimberly Keeton, et al. 1997. Scalable processors in the billion-transistor era: IRAM. *Computer* 30, 9 (1997), 75–78.

[68] Nasser Kurd, Muntaquim Chowdhury, Edward Burton, Thomas P Thomas, Christopher Mozak, Brent Boswell, Praveen Mosalikanti, Mark Neidengard, Anant Deval, Ashish Khanna, et al. 2014. Haswell: A family of IA 22 nm processors. *IEEE Journal of Solid-State Circuits* 50, 1 (2014), 49–58.

[69] Shahar Kvatinsky, Dmitry Belousov, Slavik Liman, Guy Satat, Nimrod Wald, Eby G Friedman, Avinoam Kolodny, and Uri C Weiser. 2014. MAGIC–Memristor-aided logic. *IEEE Transactions on Circuits and Systems II: Express Briefs* 61, 11 (2014), 895–899.

[70] Shahar Kvatinsky, Guy Satat, Nimrod Wald, Eby G Friedman, Avinoam Kolodny, and Uri C Weiser. 2014. Memristor-based material implication (IMPLY) logic:

design principles and methodologies. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 22, 10 (2014), 2054–2066.

[71] Benjamin C Lee, Engin Ipek, Onur Mutlu, and Doug Burger. 2010. Phase change memory architecture and the quest for scalability. *Commun. ACM* 53, 7 (2010), 99–106.

[72] Jong Chern Lee, Jihwan Kim, Kyung Whan Kim, Young Jun Ku, Dae Suk Kim, Chunseok Jeong, Tae Sik Yun, Hongjung Kim, Ho Sung Cho, Yeon Ok Kim, et al. 2016. 18.3 A 1.2 V 64Gb 8-channel 256GB/s HBM DRAM with peripheral-base-die architecture and small-swing technique on heavy load interface. In *Solid-State Circuits Conference (ISSCC), 2016 IEEE International*. IEEE, 318–319.

[73] Eero Lehtonen, Jussi H Poikonen, and Mika Laiho. 2014. Memristive stateful logic. In *Memristor Networks*. Springer, 603–623.

[74] Chao Li, Wendy Fan, Bo Lei, Daihua Zhang, Song Han, Tao Tang, Xiaolei Liu, Zuqin Liu, Sylvia Asano, Meyya Meyyappan, et al. 2004. Multilevel memory based on molecular devices. *Applied Physics Letters* 84, 11 (2004), 1949–1951.

[75] Chao Li, Daihua Zhang, Xiaolei Liu, Song Han, Tao Tang, Chongwu Zhou, Wendy Fan, Jessica Koehne, Jie Han, Meyya Meyyappan, et al. 2003. Fabrication approach for molecular memory arrays. *Applied physics letters* 82, 4 (2003), 645–647.

[76] Shuangchen Li, Dimin Niu, Krishna T Malladi, Hongzhong Zheng, Bob Brennan, and Yuan Xie. 2017. Drisa: A dram-based reconfigurable in-situ accelerator. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 288–301.

[77] Shuangchen Li, Cong Xu, Qiaosha Zou, Jishen Zhao, Yu Lu, and Yuan Xie. 2016. Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories. In *DAC*. IEEE.

[78] E Linn, R Rosezin, S Tappertzhofen, R Waser, et al. 2012. Beyond von Neumann–logic operations in passive crossbar arrays alongside memory operations. *Nanotechnology* 23, 30 (2012), 305205.

[79] Andrea Lodi, Mario Toma, Fabio Campi, Andrea Cappelli, Roberto Canegallo, and Roberto Guerrieri. 2003. A VLIW processor with reconfigurable instruction set for embedded applications. *IEEE Journal of solid-state circuits* 38, 11 (2003), 1876–1886.

[80] Joe Macri. 2015. AMD's next generation GPU and high bandwidth memory architecture: FURY. In *Hot Chips 27 Symposium (HCS), 2015 IEEE*. IEEE, 1–26.

[81] Ken Mai, Tim Paaske, Nuwan Jayasena, Ron Ho, William J Dally, and Mark Horowitz. 2000. Smart memories: A modular reconfigurable architecture. *ACM SIGARCH Computer Architecture News* 28, 2 (2000), 161–171.

[82] Ariel Maislos et al. 2011. A new era in embedded flash memory. *Flash memory summit* (2011).

[83] Jack A Mandelman, Robert H Dennard, Gary B Bronner, John K DeBrosse, Rama Divakaruni, Yujun Li, and Carl J Radens. 2002. Challenges and future directions for the scaling of dynamic random-access memory (DRAM). *IBM Journal of Research and Development* 46, 2.3 (2002), 187–212.

[84] Pedro Marcuello, Antonio González, and Jordi Tubella. 1998. Speculative multithreaded processors. In *Proceedings of the 12th international conference on Supercomputing*. ACM, 77–84.

[85] Sparsh Mittal. 2018. A Survey of ReRAM-Based Architectures for Processing-In-Memory and Neural Networks. *Machine Learning and Knowledge Extraction* 1, 1 (2018). https://doi.org/10.3390/make1010005

[86] Amir Morad, Leonid Yavits, and Ran Ginosar. 2014. Efficient dense and sparse Matrix multiplication on GP-SIMD. In *Power and Timing Modeling, Optimization and Simulation (PATMOS), 2014 24th International Workshop on*. IEEE, 1–8.

[87] Amir Morad, Leonid Yavits, and Ran Ginosar. 2015. GP-SIMD processing-in-memory. *ACM Transactions on Architecture and Code Optimization (TACO)* 11, 4 (2015), 53.

[88] Amir Morad, Leonid Yavits, Shahar Kvatinsky, and Ran Ginosar. 2016. Resistive GP-SIMD processing-in-memory. *ACM Transactions on Architecture and Code Optimization (TACO)* 12, 4 (2016), 57.

[89] Onur Mutlu. 2013. Memory scaling: A systems architecture perspective. In *Memory Workshop (IMW), 2013 5th IEEE International*. IEEE, 21–25.

[90] Ravi Nair. 2015. Evolution of memory architecture. *Proc. IEEE* 103, 8 (2015), 1331–1345.

[91] Ravi Nair, Samuel F Antao, Carlo Bertolli, Pradip Bose, Jose R Brunheroto, Tong Chen, C-Y Cher, Carlos HA Costa, Jun Doi, Constantinos Evangelinos, et al. 2015. Active memory cube: A processing-in-memory architecture for exascale systems. *IBM Journal of Research and Development* 59, 2/3 (2015), 17–1.

[92] H Noyes et al. 2014. MicronâĂŹs automata processor architecture: Reconfigurable and massively parallel automata processing. In *Proc. of Fifth International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies*.

[93] NVIDIA. 2012. Tesla K20X GPU Accelerator Board Specification.

[94] Mark Oskin, Frederic T Chong, and Timothy Sherwood. 1998. *Active pages: A computation model for intelligent memory*. Vol. 26. IEEE Computer Society.

[95] David Patterson, Thomas Anderson, Neal Cardwell, Richard Fromm, Kimberly Keeton, Christoforos Kozyrakis, Randi Thomas, and Katherine Yelick. 1997. A case for intelligent RAM. *IEEE micro* 17, 2 (1997), 34–44.

[96] David A Patterson. 2006. Future of computer architecture. In *Berkeley EECS Annual Research Symposium (BEARS), College of Engineering, UC Berkeley, US.*

[97] J Thomas Pawlowski. 2011. Hybrid memory cube (HMC). In *Hot Chips 23 Symposium (HCS), 2011 IEEE*. IEEE, 1–24.

[98] Alex Peleg and Uri Weiser. 1996. MMX technology extension to the Intel architecture. *IEEE micro* 16, 4 (1996), 42–50.

[99] M Radosavljević, M Freitag, KV Thadani, and AT Johnson. 2002. Nonvolatile molecular memory elements based on ambipolar nanotube field effect transistors. *Nano Letters* 2, 7 (2002), 761–764.

[100] RM Ramanathan. 2006. Intel® Multi-Core Processors. *Making the Move to Quad-Core and Beyond* (2006).

[101] Simone Raoux, Feng Xiong, Matthias Wuttig, and Eric Pop. 2014. Phase change materials and phase change memory. *MRS bulletin* 39, 8 (2014), 703–710.

[102] John Reuben, Rotem Ben-Hur, Nimrod Wald, Nishil Talati, Ameer Haj Ali, Pierre-Emmanuel Gaillardon, and Shahar Kvatinsky. 2017. Memristive logic: A framework for evaluation and comparison. In *Power and Timing Modeling, Optimization and Simulation (PATMOS), 2017 27th International Symposium on*. IEEE, 1–8.

[103] Gurtej S Sandhu. 2013. Emerging memories technology landscape. In *Non-Volatile Memory Technology Symposium (NVMTS), 2013 13th*. IEEE, 1–5.

[104] Karthikeyan Sankaralingam, Ramadass Nagarajan, Haiming Liu, Changkyu Kim, Jaehyuk Huh, Doug Burger, Stephen W Keckler, and Charles R Moore. 2003. Exploiting ILP, TLP, and DLP with the polymorphous TRIPS architecture. In *ACM SIGARCH Computer Architecture News*, Vol. 31. ACM, 422–433.

[105] Vivek Seshadri, Kevin Hsieh, Amirali Boroum, Donghyuk Lee, Michael A Kozuch, Onur Mutlu, Phillip B Gibbons, and Todd C Mowry. 2015. Fast bulk bitwise AND and OR in DRAM. *IEEE Computer Architecture Letters* 14, 2 (2015), 127–131.

[106] Vivek Seshadri, Donghyuk Lee, Thomas Mullins, Hasan Hassan, Amirali Boroumand, Jeremie Kim, Michael A Kozuch, Onur Mutlu, Phillip B Gibbons, and Todd C Mowry. 2017. Ambit: In-memory accelerator for bulk bitwise operations using commodity DRAM technology. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 273–287.

[107] Ali Shafiee, Anirban Nag, Naveen Muralimanohar, Rajeev Balasubramonian, John Paul Strachan, Miao Hu, R Stanley Williams, and Vivek Srikumar. 2016. ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. *ACM SIGARCH Computer Architecture News* 44, 3 (2016), 14–26.

[108] M.A. Shami and A. Hemani. 2012. Classification of Massively Parallel Computer Architectures. In *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2012 IEEE 26th International*. 344–351. https://doi.org/10.1109/IPDPSW.2012.42

[109] Patrick Siegl, Rainer Buchty, and Mladen Berekovic. 2016. Data-centric computing frontiers: A survey on processing-in-memory. In *Proceedings of the Second International Symposium on Memory Systems*. ACM, 295–308.

[110] A Siemon, S Menzel, A Chattopadhyay, R Waser, and E Linn. 2015. In-memory adder functionality in 1S1R arrays. In *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 1338–1341.

[111] Anne Siemon, Stephan Menzel, Rainer Waser, and Eike Linn. 2015. A complementary resistive switch-based crossbar array adder. *IEEE journal on emerging and selected topics in circuits and systems* 5, 1 (2015), 64–74.

[112] Gagandeep Singh, Lorenzo Chelini, Stefano Corda, Ahsan Javed Awan, Sander Stuijk, Roel Jordans, Henk Corporaal, and Albert-Jan Boonstra. 2018. A Review of Near-Memory Computing Architectures: Opportunities and Challenges. In *Proceedings of the 21st Euromicro Conference on Digital System Design (DSD)*.

[113] D.B. Skillicorn. 1988. A taxonomy for computer architectures. *Computer* 21, 11 (Nov 1988), 46–57. https://doi.org/10.1109/2.86786

[114] G Snider. 2005. Computing with hysteretic resistor crossbars. *Applied Physics A: Materials Science & Processing* 80, 6 (2005), 1165–1172.

[115] Kyomin Sohn, Won-Joo Yun, Reum Oh, Chi-Sung Oh, Seong-Young Seo, Min-Sang Park, Dong-Hak Shin, Won-Chang Jung, Sang-Hoon Shin, Je-Min Ryu, et al. 2017. A 1.2 V 20 nm 307 GB/s HBM DRAM with at-speed wafer-level IO test scheme and adaptive refresh considering temperature distribution. *IEEE Journal of Solid-State Circuits* 52, 1 (2017), 250–260.

[116] Harold S Stone. 1970. A logic-in-memory computer. *IEEE Trans. Comput.* 100, 1 (1970), 73–78.

[117] Arun Subramaniyan, Jingcheng Wang, Ezhil R. M. Balasubramanian, David Blaauw, Dennis Sylvester, and Reetuparna Das. 2017. Cache Automaton. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-50 '17)*. ACM, New York, NY, USA, 259–272. https://doi.org/10.1145/3123939.3123986

[118] Jinwoo Suh, Eun-Gyu Kim, Stephen P Crago, Lakshmi Srinivasan, and Matthew C French. 2003. A performance analysis of PIM, stream processing, and tiled processing on memory-intensive signal processing kernels. In *ACM SIGARCH Computer Architecture News*, Vol. 31. ACM, 410–421.

[119] Mark R Thistle and Burton J Smith. 1988. A processor architecture for Horizon. In *Supercomputing'88.[Vol. 1]., Proceedings*. IEEE, 35–41.

[120] Dean M Tullsen, Susan J Eggers, and Henry M Levy. 1995. Simultaneous multithreading: Maximizing on-chip parallelism. In *ACM SIGARCH Computer Architecture News*, Vol. 23. ACM, 392–403.

[121] Mario Vestias and Horácio Neto. 2014. Trends of CPU, GPU and FPGA for high-performance computing. In *Field Programmable Logic and Applications (FPL), 2014 24th International Conference on*. IEEE, 1–6.

[122] Borui Wang, Martin Torres, Dong Li, Jishen Zhao, and Florin Rusu. 2016. Performance Implications of Processing-in-Memory Designs on Data-Intensive Applications. In *Parallel Processing Workshops (ICPPW), 2016 45th International Conference on*. IEEE, 115–122.

[123] Jue Wang, Xiangyu Dong, Yuan Xie, and Norman P Jouppi. 2014. Endurance-aware cache line management for non-volatile caches. *ACM Transactions on Architecture and Code Optimization (TACO)* 11, 1 (2014), 4.

[124] Ying Wang, Yinhe Han, Lei Zhang, Huawei Li, and Xiaowei Li. 2015. ProPRAM: exploiting the transparent logic resources in non-volatile memory for near data computing. In *Proceedings of the 52nd Annual Design Automation Conference*. ACM, 47.

[125] Rainer Waser. 2012. Redox-based resistive switching memories. *Journal of nanoscience and nanotechnology* 12, 10 (2012), 7628–7640.

[126] Rainer Waser and Masakazu Aono. 2007. Nanoionics-based resistive switching memories. *Nature materials* 6, 11 (2007), 833.

[127] Stephan Wong, Thijs Van As, and Geoffrey Brown. 2008. ρ-VEX: A reconfigurable and extensible softcore VLIW processor. In *ICECE Technology, 2008. FPT 2008. International Conference on*. IEEE, 369–372.

[128] Wm A Wulf and Sally A McKee. 1995. Hitting the memory wall: implications of the obvious. *ACM SIGARCH computer architecture news* 23, 1 (1995), 20–24.

[129] Lei Xie, Hoang Anh Du Nguyen, Mottaqiallah Taouil, and Koen Bertels Said Hamdioui. 2015. Fast boolean logic mapped on memristor crossbar. In *Computer Design (ICCD), 2015 33rd IEEE International Conference on*. IEEE, 335–342.

[130] Lei Xie, Hoang Anh Du Nguyen, Jintao Yu, Ali Kaichouhi, Mottaqiallah Taouil, Mohammad AlFailakawi, and Said Hamdioui. 2017. Scouting Logic: A Novel Memristor-Based Logic Design for Resistive Computing. In *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 335–340.

[131] J Joshua Yang, Dmitri B Strukov, and Duncan R Stewart. 2013. Memristive devices for computing. *Nature nanotechnology* 8, 1 (2013), 13–24.

[132] Leonid Yavits, Shahar Kvatinsky, Amir Morad, and Ran Ginosar. 2015. Resistive Associative Processor. *CAL* (2015).

[133] Jintao Yu, Lei Xie, Mottaqiallah Taouil, and Said Hamdioui. 2018. Memristive Devices for Computation-In-Memory. In *Design, Automation and Test in Europe DATE*.

[134] Shimeng Yu and Pai-Yu Chen. 2016. Emerging memory technologies: Recent trends and prospects. *IEEE Solid-State Circuits Magazine* 8, 2 (2016), 43–56.

[135] Jian-Gang Zhu. 2008. Magnetoresistive random access memory: The path to competitiveness and scalability. *Proc. IEEE* 96, 11 (2008), 1786–1798.

MANUSCRIPT 1

A Survey on Memory-Centric Computer Architectures

Hoang Anh Du Nguyen, Jintao Yu, Muath Abu Lebdeh *Student Member, IEEE,* Mottaqiallah Taouil, *Member, IEEE,* Said Hamdioui, *Senior Member, IEEE,* Francky Catthoor, *Fellow, IEEE,*

Abstract—Faster and cheaper computers have been constantly demanding technological and architectural improvements. However, current technology is suffering from three technology walls: leakage wall, reliability wall and cost wall. Meanwhile, existing architecture performance are also saturating due to three well-known architecture walls: memory wall, power wall and instruction level parallelism (ILP) wall. Hence, a lot of novel technologies and architectures have been introduced and developed intensively. Our previous work has presented a comprehensive classification and broad overview on memory-centric computer architectures. In this paper, we aims to discuss all the memory-centric architectures thoroughly and evaluates their advantages and disadvantages.

Index Terms—Computation-in-Memory, Computer Architectures, Resistive Computing, Classification.

1 INTRODUCTION

FOR several decades, CMOS down-scaling and architecture improvements have doubled computer performance following Moore law [1], [2], [3]. However, existing technology suffers from three main walls: leakage wall, reliability wall and cost wall [4], while computer architectures also face three walls: memory wall, power wall and instruction level parallelism (ILP) wall [5]. In order to address these walls, a lot of novel technologies and architecture are under research to improve the performance [1]. As a result, an enormous amount of computer architectures has been proposed so far.

Since the first von Neuman architecture in 1982, computer architectures have been evolved to various complex organizations including pipelined, superscalar, multicore, etc. [3], [6], [7]. In an effort to further improve their performance, a concept of integrating memory and processing units, so-called Logic-in-Memory (LIM), was invented in 1970 [8]; however, it was only applied to cache/on-chip memory and was soon abandoned due to its ineffectiveness. Since 2000, big data and embedded applications have been on the rise and required a new computer system with not only higher performance but also energy efficient. In order to fulfill these requirements, several architectures were explored with the concept of LIM applied for main memories, and regarded as Processing-In-Memory (PIM) architectures (i.e., FlexRAM [9], DIVA [10], intelligent RAM [11], etc). However, the PIM architectures were also shortly dismissed due to limitation of embedded DRAM technology [12], [13], [14]. Since 2008, emerging non-volatile memroy technology (i.e., memristor) has revived the concept of Processing-In-Memory under the new name In-Memory Computing [15] or Computation-In-Memory [16]. The novel architectures together with new technology promise a lot of potentials in terms of performance, energy and area [16], [17]. Some examples can be listed as Computation-In-Memory (CIM) [18], ReVAMP [19], Pinatubo [20], etc. All these architectures have common and distinct features, and was addressed inconsistently in the community. This leads to confusion in differentiate among architectures and exploring novel potential architectures.

In our previous work, we have presented a classification of architectures that gathers and classifies existing architectures, and also show unexplored architectures []. Based on the classification, this paper provides an intensive survey on existing memory-centric architectures. The contributions of this paper are the following:

- Presenting main characteristics and working principles of existing memory-centric architectures.
- Discussing and evaluating main advantages and disadvantages of each architecture.
- Comparing among different existing architectures and outlook the characteristics of the future architectures.

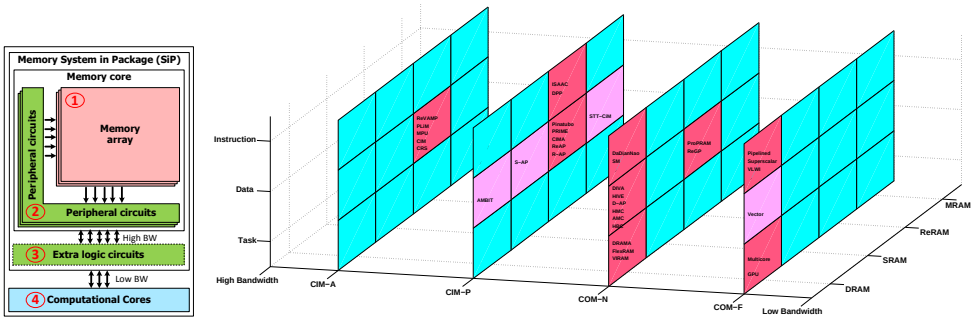
The rest of this paper is structured as follows. Section 2 summarizes the metrics used in the classification, the overview of different classes with their existing architectures. Section 3 and 4 present two main categories in the proposed classification including CIM-A and CIM-P, respectively; each section describes and evaluate the architectures qualitatively. Section 5 compares existing architectures and discusses the future architecture requirements. Finally, section 6 concludes this paper. Due to the page limits, we also include Appendix A and B that describe and evaluate the architectures in the other two less-focused categories: COM-N and COM-F, respectively; note that the comparison in section 5 still includes these two categories.

2 CRITERIA AND CLASSIFICATION

In this section, we first summarize a set of metrics used in the classification. Thereafter, we show our classification of

- *The authors are with the Department of Computer Engineering, Delft University of Technology, the Netherlands. E-mail: H.A.DuNguyen@tudelft.nl*

Manuscript received ...; revised ...



(a) Architecture (b) Classification

Fig. 1: Computer Architecture and Proposed Classification

computer architectures and map the existing architectures on it.

2.1 Classification Metrics

The classification includes several metrics to classify computer architectures based on the its computing resources and memory. A computer architecture or system consists of one or more memories and computational units as shown in Fig. 1a. The memories is the main storage unit; it can include only memory core with memory arrays and its supporting peripheral circuits, or memory core with extra logic circuits, which is called memory System-in-Packages (SiP). The computations is performed traditionally using computation cores, however, they can also be performed using extra logic circuits, peripheral circuits and memory array of the memory SiP. A computer system can be classified based on three metrics:

- **Computation position:** it defines where the result of the computation is produced. A computation includes a primitive logic function (e.g., logical operations) or arithmetic operation (e.g., addition, multiplication). The four possibilities of computation position can be seen at the four circles in Fig. 1a. If the result is produced within the memory core, the computer architecture is referred to as Computation-In-Memory (CIM); otherwise, the result is produced outside the memory core and the architecture is referred to as Computation-Out-Memory (COM). Both CIM and COM can be further sub-classified.
  - **CIM:** Computations in CIM either takes place in the memory array (1) or peripheral circuit (2) which are referred to as Computation-In-Memory Array (CIM-A) and Computation-In-Memory Peripheral (CIM-P), respectively. Both sub-classes impact the design of the memory, either the redesign of cells to support computing within the crossbar, or special circuits in the peripheral circuit such as customized sense amplifiers [20], [21], or the usage of ADCs [22].
  - **COM:** Computation in COM take either place in the extra logic circuits inside the memory SiP (3) or in the traditional computational cores (4) which are referred to as Computation-Out-Memory Near (COM-N) and Computation-Out-Memory Far (COM-F), respectively.

  Note that hybrid architectures are considered when a primitive function can generate its results at multiple computation positions, or an architecture has multiple primitive functions, each with a different computation position.

- **Memory technology:** it defined which technology is used to implement the memory array. The technologies includes conventional charge-based memories such as DRAM/SRAM/Flash [23], [24], [25] and emerging non charge-based memories [26]. The non charge-based memories include different types distinguished by their physical mechanism; these includes resistive [26], [27], "magnetic" memories [26], [28], [29], molecular memories [30], [31], [32], [33], mechanical memories [34], [35], and other types of memories, (i.e., molecular memories, mechanical memories) that can currently not be used for computing and are not discussed further in this classification.

- **Computation parallelism:** it defines the level of parallelism that can be exploited in a computer system; i.e., task, data, and/or instruction level parallelism. In task level parallelism, a system has multiple independent control units and data memories; examples include multithreading [36], [37] and multicore systems [7]. In data parallelism, a system has a single control unit is used to apply the same instruction concurrently on a collection of data elements; examples includes data elements with constant sizes (e.g., vector and array processor [38], [39]), and varying sub-word sizes (e.g., SWAR (SIMD Within A Register) processor [40], [41]). In instruction level parallelism, a system has a single control unit is

used to execute various instructions concurrently; examples include intra-instruction (e.g., pipe-lined processor [42] and inter-instruction (e.g., VLIW processor [43]) parallelism. Additionally, they can be combined together as in speculative processor [44].

2.2 Classification

Based on the above discussed metrics, 48 classes can be differentiated; among them, 13 classes are occupied by the existing architectures which are located in red planes and pink planes. The red planes demonstrate that a lot of work has been done for that particular class. The pink planes demonstrate a moderate number of work has been done. The blue planes demonstrate either unexplored classes due to no attention yet of the research community, or non-existing due to current restrictions of the memory technology. In the following sections, we will discuss all the architectures in the red and pink planes.

3 COMPUTATION-IN-MEMORY - ARRAY (CIM-A)

The CIM-A class contains mostly resistive computing architectures that use memristive-based logic circuits [45] to perform computations and resistive RAM (RRAM) as memory technology. Few architectures have been proposed in this category. Table 1 shows a brief comparison among the architectures which will be explained in each subsection.

On one hand, all these architectures have several common advantages:

- Low memory bottleneck due to computing inside the memory.
- High data parallelism due to the possibility of performing concurrent operations inside the crossbars.
- Low leakage and small footprint due to the usage of non-volatile memory technology.

On the other hand, they all share several limitations:

- High computing latency per access due to the high latency of writing memristors and the need of multiple write steps to perform Boolean functions. Note that despite a high computing latency, the performance can be still high when sufficient parallelism is exploited.
- Higher endurance requirement due to the need of multiple write steps to perform Boolean functions.
- Requires redesigning of the cell in order to make the computing feasible.

The following subsections discuss the details and characteristics of each architecture.

3.1 CRS: Complementary Resistive Switch Architecture

Complementary Resistive Switch (CRS) architecture was proposed in 2014 by A. Siemon, et al., from RWTH Aachen University [46]. It is a memristor based architecture that exploits data level parallelism using implication logic. The architecture consists of multiple crossbars and a control unit (as shown in Fig. 2 [46]). The crossbar stores and performs logic operations using CRS cells; a CRS cell consists of two resistive switches or resistive RAMs. The control unit distributes signals to the intended addresses (wordlines and bitlines) to perform operations on the crossbars.

The crossbar is controlled by a sequence of operations including: write-in (WI), read-out (RO), write-back (WB), and compute (CP). Before the operations can be performed, the crossbar part used for computation is once entirely reset to a logic value 0. The WI operation writes a logic value into a memristor. The RO operation reads a logic value from a cell; the logic output value is determined by the sense amplifier. The RO operation is destructive and changes the value of the memristor to logic value 1. The task of the WB operation is to recover the destroyed value. Finally, the CP instructions are used to execute the implication logic gates [46], [47]. The data transfer between CRS cells is carried out through the control unit using a RO and WB operations; in other words, the control unit reads a value of the source CRS cell and writes this value into the destination cell.

In addition to the general advantages of CIM-A architectures, CRS has the following advantages:

- It is less impacted by the sneak path currents due to the usage of CRS cells. The cell's resistance is always equivalent to high resistance, hence, sneak path currents are eliminated. However, variations in resistances will make such paths practically unavoidable unless a 1T2R cell is used.
- CRS logic requires fewer cells to perform computations than FBL.

However, it also has the following limitations:

- The latency of the primitive functions varies and requires read-out instructions to determine the voltages that have to be applied.
- The RO operation is destructive, hence, a WB operation is required after each RO operation, which increases the latency and energy of computations.
- The data tranfer method is indirect as it is based on the read-out and write-back scheme. As all cells have high resistance, direct copying of cells in the crossbar is not applicable.
- The control unit imposes a high overhead as it is responsible for both controlling the crossbar (requiring a large number of states) and transferring data (which involves the usage of buffers/registers to store temporary values).
- The area of CRS cells is larger than those based on a single memristor cells.
- The architecture requires additional compiling techniques and tools to convert conventional Boolean logic functions to implication logic.

This architecture was only evaluated at circuit level using adders. Therefore, it is hard to make general conclusions on the performance and the applicability of this architecture.

3.2 CIM: Computation-in-Memory

CIM was proposed in 2015 by H.A. Du Nguyen, et al., from Delft University of Technology [16], [17], [48]. It is a memristor based architecture that exploits data level parallelism by using any memristor logic style; the authors

	Hierarchy level	Computations		Memory Technology	Overheads			Sneak path current	Destructive read	Required read-out*	Copy scheme	Evaluation	
		Logic style	Available functions		Cells	Periphery	Controller					Simulator	App.
CRS	Accelerator	CRS	Logical, +	NVM	CRS	Conv.	Complex	No	Yes	Yes	Indirect	Hspice	No
CIM	Accelerator	Varied	Logical, +, x	NVM	RRAM	Conv.	Varied	Yes	No	Varied	Both	Analytical	Parallel adder and multiplier
MPU	Main memory	MAGIC	Logical, +, x	NVM	MAGIC	Conv.	Simple	Yes	No	No	Both	Analytical	Image processing
PLiM	Main memory	Majority	Majority gates	NVM	RRAM	Conv.	Complex	Yes	No	Yes	Both	Analytical	Encryption
ReVAMP	Main memory	Majority	Majority gates	NVM	RRAM	conv.	Complex	Yes	No	Yes	Both	Analytical	EPFL benchmarks
DRISA-3T1C	Accelerator	DRAM	Boolean	DRAM	3T1C	Modif.	Simple	No	No	No	Both	CACTI-3DD, in-house	CNN

+: n-bit addition
x: n-bit multiplication
NVM: Non-volatile memory
Conv.: Conventional
Modif.: Modified
(*): Required read-out during computations
App.: Applications and benchmarks

TABLE 1: Comparison among Architectures of CIM-A Classes

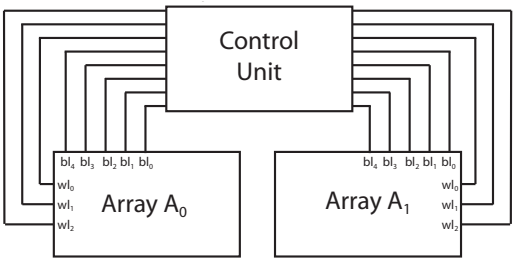


Fig. 2: Complementary Resistive Switch-logic Crossbar Array (CRS) [46]

have showed the potential of this architecture using Fast Boolean Logic (FBL) [49] and implication logic [46]. The architecture consists of a memristor crossbar, and a control and communication block as shown in Fig. 3 [17]. The memristor crossbar stores data and performs computations. The control and communication block applies appropriate voltages to the memristor crossbar.

The architecture uses state machines stored in the *control and communication block* to compute and transfer data in the crossbar. Once triggered, the state machine applies an appropriate sequence of control voltages to the rows and columns of the memristor crossbar. Depending on the memristor types, the data transfer occurs directly inside the crossbar (for single RRAM cells) or indirectly through the *control and communication block* outside the crossbar (for CRS cells) using the CRS read-out/write-back scheme.

In addition to the general advantages of CIM-A architectures, CIM comes with the following advantage:

- The architecture can accommodate any type of memristor logic design due to the flexibility of the control and communication block.
- In case FBL is used, the latency of primitive functions (i.e., addition, multiplication) is a constant number.
- The data transfer using both direct and indirect schemes has been intensively explored in [50], [51].
- The control block of FBL is less complex than the control block of implication logic due to a fixed number of write steps and a simpler control voltage scheme [49].
- In case single RRAM cell is used, the write energy and area of a RRAM cell is smaller in comparison with a CRS cell.

However, it also has the following limitations:

- The architecture has to deal with sneak path currents in case a single RRAM cell (0T1R) is used as multiple rows and columns are activated simultaneously. Possible solutions to alleviate the problem consist of isolating each FBL circuit, or the usage of a transistor-memristor (1T1R) structure to actively control each memristor using a transistor [52], [53], or isolated/half select voltages [54], [55].
- In case FBL is used, typically a lot of cells are required due to LUT based computing.
- In case CRS cells are used, the same drawbacks of CRS architecture applies, i.e., a larger cell area, the control unit imposes a high overhead as the controllers are responsible for both controlling the crossbar and transferring data, and it requires additional compiling techniques and tools to convert conventional Boolean logic functions to implication logic .

The potentials of the architecture are demonstrated using a case study of a binary-tree based parallel adder and multiplier [48], [56]; the architecture is compared with a conventional multicore architecture. CIM architecture achieves at least one order of magnitude improvements in terms of delay, energy and area.

3.3 MPU: Memristive Memory Processing Unit

MPU was proposed in 2016 by R. Ben Hur, et al., from Technion-Israel Institue of Technology [57]. It is a memristor based architecture that exploits data level parallelism using Memristive-Aid loGIC (MAGIC) [58]. The architecture consists of a conventional processor, MPU controller and a memristive memory as shown in Fig. 4. The processor contains a control unit, an arithmetic and logic unit, and a memory controller. The MPU controller includes a Processor-In/Out block to interface to the conventional CPU, control blocks to execute specific commands (arithmetic, set/reset,

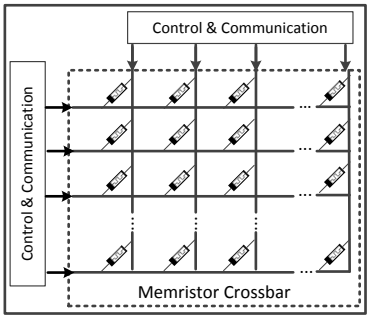


Fig. 3: The Computation-In-Memory Architecture (CIM) [48]

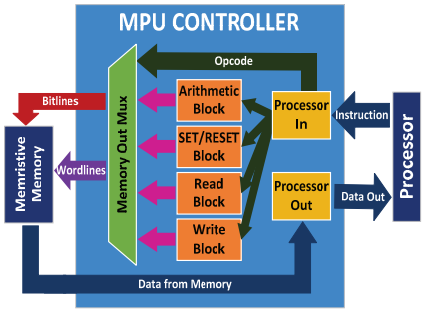


Fig. 4: The Memristor Memory Processing Unit (MPU) [57]

read and write block) and a output multiplexer to apply appropriate voltages to specific rows/columns of the memristive memory. The conventional processor sends an instruction to the memristive memory using its own memory controller and the MPU controller. The memory controller of the processor recognizes the memristive memory instructions in a similar manner as conventional memory operations, while the MPU controller determines whether to treat the memristive memory as a storage element or a processing element. Based on that, the MPU controller applies read/write signals or a sequence of signals to perform logical or arithmetic operations.

The MPU controller uses the Processor-In unit to divert the instructions to specific blocks (such as arithmetic, read and write blocks) responsible for the execution of those operations. Each block determines the appropriate voltages that have to be applied to the memristive memory. The set/reset, read and write block have a latency of 1 cycle, while the arithmetic block requires multiple cycles to execute a vector operations using MAGIC logic [58]. Data movements in the crossbar are performed directly using *copy* and *INV* (double negation) operations.

In addition to the general advantages of CIM-A architectures, MPU comes with the following advantages:

- The latency of MAGIC primitive functions is fixed.
- The data transfer may include direct (based on copying) and indirect (based on read-out/write-back) schemes.
- The MPU controller is simpler than for the CRS architecture, as each operation consists of a fixed number of steps while less control voltage values are used.
- The write energy of a single MAGIC cell is smaller in comparison to those of a CRS cell.
- MAGIC requires in comparison to FBL fewer cells to perform computations.

However, it also has the following limitations:

- The architecture has to deal with sneak path currents. Possible solutions are mentioned in Section 3.2.
- The control voltages used in MAGIC have to satisfy the constraint $2V_{reset}<V_w<V_{set}$, where V_{set} is the minimum voltage required to switch a memristor from R_H to R_L, and V_{eset} is the minimum voltage required to switch a memristor from R_L to R_H; in other words, it requires that the memristors have a higher V_{set} than V_{reset}, leading to an unbalanced hysteresis loop. This limits the types of memristors that can be used for MAGIC.
- The architecture requires additional compiling techniques and tools to convert conventional Boolean logic functions to MAGIC gates.

The potential of the architecture is demonstrated by performing a logical bit-wise OR operation of two 8-bit vectors in 20 steps. The latest research has shown that MAGIC can be used for several arithmetic operations such as addition, multiplication, etc. [59], [60], [61].

3.4 PLiM: Programmable Logic-in-Memory Computer

PLiM was proposed in 2016 by P. Gaillardona, et al., from EPFL [62]. It is a memristor based architecture that exploits data parallelism using majority logic [63]. The architecture consists of a resistive memory organized in banks and a Logic-in-Memory (LiM) controller block as shown in Fig. 5. The memory is a memristive crossbar that stores both the instructions and data. The LiM controller is composed of a number of registers and a finite state machine (FSM). The controller functions as a simple processor; it fetches instructions from the memory array, decodes and executes the operation inside the memory.

The LiM controller operates in two modes: conventional memory read/write mode and in-memory instruction mode. In the read/write mode, the FSM is deactivated, and the memory array is read or written in the same manner as a standard memory. In the in-memory instruction mode, FSM is activated, and an instruction is performed using majority logic gates inside the memory. Once the FSM is enabled, the following operations are performed. First, the FSM resets all registers in the LiM controller. Second, an instruction is read from the address in the program counter (PC) and decoded to obtain the addresses of the two operands and output; the addresses of the two operands are stored in registers @A and @B, while the output address is stored in register @Z. Third, the value of the two operands are read using the addresses in registers @A and @B; the obtained values are stored in registers @A and @B, respectively. Fourth, depending on the logic values (0 or 1) of the operands, appropriate voltages

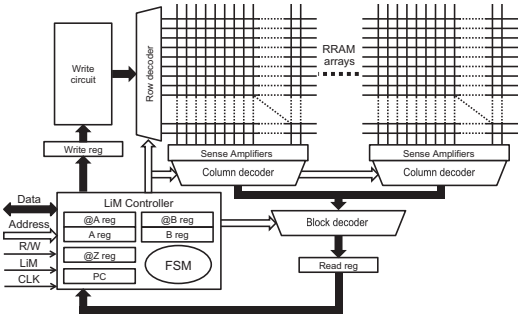


Fig. 5: The Programmable Logic-in-Memory Computer (PLiM) [62]

are applied to the crossbar at address @Z to perform a majority logic gate. Finally, the PC is incremented by one.

In addition to the general advantages of CIM-A architectures, PLiM has the following advantages:

- The data transfer may include both direct and indirect schemes.
- The write energy and area of a memristor cell is smaller as compared to a CRS cell.

However, it also has the following limitations:

- The latency of majority primitive functions varies depending on the functional complexity and some read-outs are required to determine the voltage values to be applied.
- The architecture has to deal with sneak path currents. Possible solutions as mentioned in Section 3.2.
- The LiM controller is complex as it has to determine the control voltage values based on the operands' values.
- The architecture requires additional compiling techniques and tools to convert conventional Boolean logic functions to majority logic gates.

The architecture is evaluated with a PRESENT Block Cipher algorithm [64], which encrypts a 64-bit plain text with an 80 or 128-bit key. The algorithm is compiled into a sequence of majority logic gates and executed on PLiM. Unfortunately, the results show that PLiM's performance is almost a factor of two slower than a 180nm FPGA implementation [64].

3.5 ReVAMP: ReRAM based VLIW architecture

ReVAMP was proposed in 2017 by D. Bhattacharjee, et al. from Nanyang Technological University [19]. It is a memristor based architecture that exploits data parallelism using majority logic. The architecture consists of an Instruction Fetch (IF), Instruction Decode (ID), and Execute (EX) stage. The IF block fetches instructions from the Instruction Memory using the program counter (PC) as address, and puts the resulting instruction in the Instruction Register (IR). The ID block decodes the instruction and generates control signals which are placed in the control registers of the EX block. The EX stage finally executes the instruction.

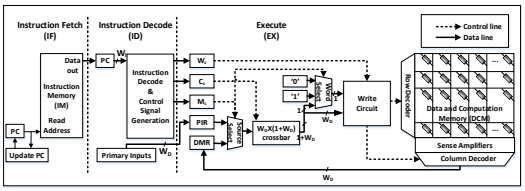


Fig. 6: ReRAM based VLIW architecture (ReVAMP) [19]

The IF and ID stages are similar to those of the traditional five-pipelined RISC architecture. The IF stage includes an Instruction Memory (IM) and a Program Counter (PC). The ID stage contains registers (IR and Primary Inputs), and an Instruction Decode and Control Signal Generation. The EX stage consists of several registers (i.e., Data Memory Register (DMR), Primary Input Register (PIR), Mux control (M_c) register, Control (C_c) register, Wordline (W_c) register), as well as a crossbar interconnect, wordline select multiplexer, data Source Select multiplexer, and a Write circuit to control the crossbar that stores data. Once an instruction is fetched and decoded in IF and ID, respectively, the control registers in EX are filled with suitable values. These values control the multiplexers that are responsible for applying the right control signals to the crossbar. Depending on the operation, primary inputs from PIR or data retrieved from the crossbar stored in DMR can be used for the next operation. The crossbar interconnect permutes the inputs and control signals (indicated by C_c) to generate the voltages that need to be applied to the memory crossbar. The Write circuit applies these voltages to the targeted wordline address (indicated by W_c).

In addition to the general advantages of CIM-A architectures, ReVAMP has the following advantages:

- The data transfer may include direct (within the crossbar based on copying resistance values) and indirect (based on read-out/write-back) schemes.
- The crossbar is based on only one device per cell, resulting in a more compact architecture as compared with other architectures which make use of two devices per cell (i.e., Complementary Resistive Switch CRS [46]).

However, it also has the following limitations:

- The latency of majority primitive functions varies depending on the functional complexity; in addition, before any operations are applied to the cells, these cells first have to be read-out in order to determine the appropriate control voltages.
- The architecture has to deal with sneak path currents. Possible solutions to alleviate the problem consist of isolating each tile/crossbar, or using a transistor-memristor (1T1R) structure to actively control each memristor using a transistor [52], [53], or using isolated/half select voltages [54], [55], [65].
- The EX stage is complex as it integrates both the control signals for memory and computations. Therefore, it is not easy to pipeline this architecture, as

the EX stage will consume more time than the other stages; i.e., the stages IF, ID, and EX are not balanced.
- The architecture requires additional compiling techniques and tools to convert conventional Boolean logic functions to majority logic gates.

The architecture is simulated and evaluated using EPFL benchmarks [66] and compared against PLiM [62], which is based on a resistive memory with the same logic style.

3.6 DRISA-3T1C: A DRAM-based Reconfigurable In-Situ Accelerator with 3T1C design

DRISA-3T1C was proposed in 2016 by S. Li, et al. from University of California [67]. It is a DRAM based architecture that exploits data parallelism by performing NOR gate inside DRAM cells [63]. The architecture consists of a DRAM memory organized in a hierarchy of banks, sub-arrays and mat; each levels are controlled by their corresponding controllers as shown in Fig. 7(a),(b) and (c). The banks are connected through global bus (gbus) while communication among subarrays are carried out using bank buffers (bBuf). The mats perform both data storage and computations.

The memory mats consist of cell regions for both data and computations, and peripheral circuits including calc-SA, intra/inter-lance shifter (SHF) and lane forwarding unit (FWD). The cell regions contains multiple DRAM cells which consists of three transistors connected to form a NOR gate and one capacitor to store the data value. In order to perform computations, two DRAM cells (Rs and Rt) to be activated simultaneously and one DRAM cell (Rr) to store the computation result (as shown in Fig. 7). Read voltages are applied to the sources DRAM cells (Rs and Rt) through the wordline (rWL) while write voltage is applied to the result DRAM cell (Rr). The voltage collected by the sense amplifier (SA) is used to control the transistor in the Rr DRAM cell. Due to the NOR organization of these transistors, a NOR operation is realized and produce results in DRAM cells. The SA (also called calc-SA) cooperates with extra logic circuitry such as SHF and FWD to perform complex functions such as addition, copy and inner product.

In addition to the general advantages of CIM-A architectures, DRISA-3T1C has the following advantages:

- The latency of NOR primitive functions is fixed.
- The data transfer may include both direct and indirect schemes.
- The architecture does not suffer destructive read as in the case of CRS architecture [46], hence the write energy might be less due to the absence of write-after-read.
- The controller is simpler than for the CRS architecture, as each operation consists of a fixed number of steps while fewer control voltage values are used.
- The architecture uses DRAM technology, which has several benefits such as: high maturity and endurance, no sneak path currents, and the accessibility to optimized architectures, technology and tools.

However, it also has the following limitations:

- The latency of complex functions varies depending on the functional complexity as each function needs to be converted into multiple NOR gates.
- The architecture uses DRAM technology which suffers from a low performance, high energy consumption, large footprint and is difficult to scale down.

The architecture is simulated and evaluated against GPU TITAN X [68] using four CNN applications including 8-layer AlexNet [69], 16-layer VGG-16 [70], 19-layer VGG-19, and 152-layer ResNet-152 [71].

4 COMPUTATION-IN-MEMORY - PERIPHERALS (CIM-P)

The CIM-P class consists of architectures which perform computations during read-out operations (i.e. 2 or more word lines are activated simultaneously) using special peripheral circuitry. Such operations are typically analog in nature. As there are less restrictions on the functionality of the cell, various memory technologies can be used in this category such as DRAM, SRAM and non-volatile memory technologies. A medium number of architectures have been proposed in this category. Table 2 shows a brief comparison among the architectures which will be explained in each subsection.

On one hand, these architectures have several common advantages:

- Low memory bottleneck as the results are produced in the peripheral circuitry which is connected directly to memory array.
- High parallelism due to the the possibility of performing multiple concurrent operations.
- High performance as computations are performed in a single read step.
- Relatively simple controllers as the operations are constructed in a similar manner as for conventional memory (read/write) operations.
- Higher compatibility with available memory technologies, because redesigning cells would induces a huge cost for the vendors.
- Lower endurance requirement as operations are based on reading instead of writing [21].

On the other hand, they all share the following limitations:

- Overhead to align data; note that each operations requires the data to be aligned in the memory. Therefore, if the operands are not located in the same crossbar, data transfer operations are required.
- Additional write overhead when the results have to be stored back in to the memory. Note that the outputs are produced as voltages in the peripheral circuit, and therefore, if the results have to be stored back in the memory extra write operations would be necessary.
- Complex peripheral circuits (as they have to be modified) limit the scalability and could also dominate the area of the memory core.
- The level of parallelism is determined by the peripheral circuit. More parallelism leads to larger and complex peripheral circuits.
- Hard to implement arithmetic operations and as of today mainly limited to bitwise logical operations.

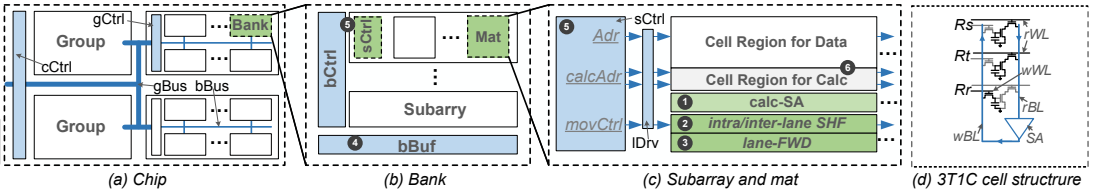


Fig. 7: A DRAM-based Reconfigurable In-Situ Accelerator (DRISA) [67]

- As the equivalent voltage or current of two or more accessed cells may end up in multiple levels, complex sense amplifiers are required.

The following subsections discuss the details of each architecture.

4.1 ReAP: Resistive Associative Processor

ReAP was proposed in 2014 by Y. Leonid, et al. from Technion-Israel Institute of Technology [72]. ReAP is a RRAM based architecture that exploits data parallelism using LUTs implemented with Contend Addressable Memories (CAMs) to perform computations. The architecture consists of a crossbar of resistive CAM cells and peripheral circuits including sense amplifiers and registers (as shown in Fig. 8). A CAM cell consists of two resistive RAM cells which store the true and complement value of a single bit. Multiple CAM cells are used to create a look-up-table (LUTs); together they implement a specific function (e.g. a 1-bit full-adder). In case ReAP is active, it performs a *compare* the inputs (stored in register *KEY* with the LUT words and produce in case a match occurs the corresponding outputs in *TAG* registers.

The *compare* operation is performed in a similar manner as in conventional CAMs. First the Match/Word line is precharged. Thereafter, the values in *KEY* are applied to the bitlines depending on the *MASK* value; if a bit is masked, it is kept floating. If the *KEY* matches the content on a particular wordline, the TAG will generate the value '1' at the output, otherwise '0'. For example, in case a key bit is one, both the true (i.e., low resistance) and complement value (i.e., high resistance) will keep the floating word line high in case a one is stored. In case the cell holds a zero, i.e., the true memristor has a high resistance and complement a low resistance value, the complement path will discharge the Match/Word line. Similar conclusions can be drawn in case the key bit is 0. In order to execute a more complex function, LUTs can be reconfigured. In such cases the output of the LUT is fed back to input of the same LUT but with a different configuration. Another option is to implement the function using multiple LUTs.

In addition to the general advantages of CIM-P architectures, ReAP has the following advantages:

- The architecture is used as an accelerator, which has a positive impact on the endurance due to infrequent use [73], [74]. In contrast, some CIM-P architectures are used as main memory and they require a much higher endurance.

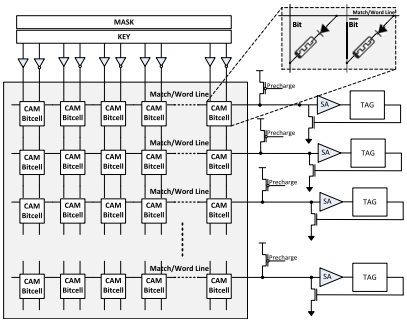


Fig. 8: Resistive Associative Processor (ReAP) [72]

- Computing based on LUTs is quite mature (e.g., in FPGA's) and can benefit from existing techniques and tools.
- The architecture uses non-volatile memory, hence consumes a low amount of energy and has a small footprint.

However, it also has the following limitations:

- Computations using LUTs can be inefficient if the number of inputs per LUT is large. If multiple smaller LUTs are used the latency becomes higher.
- The data transfer consists of an indirect read-out scheme.
- The architecture has to deal with sneak path currents. Possible solutions are mentioned in Section 3.2.
- The write operations of this architecture may suffer from high energy consumption, as two memristors are written per CAM cell.
- The architecture might not exploit the full memory bandwidth, as it is challenging to fit all sense amplifiers into the memory core.
- The architecture requires additional compiling techniques and tools to convert conventional Boolean logic functions to CAM based LUTs.

The architecture was evaluated analytically using several benchmarks [75] such as N-pairs Black-Scholes option pricing, N-point Fast Fourier Transform, and Dense Matrix Multiplications. They compared the results of these benchmarks on ReAP with two other platforms: a CMOS equivalent of ReAP denoted by CMOS-AP and GTX480 GPU.

	Hierarchy level	Computations		Memory Technology	Overheads			Sneak path current	Destructive read	Required read-out*	Copy scheme	Evaluation	
		Logic style	Available functions		Cells	Periphery	Controller					Simulator	App.
ReAP	Accelerator	CAM	LUT-based	NVM	RRAM	Modif.	Simple	Yes	No	No	Both	Analytical	Arithmetic
PRIME	Main memory	NN.	MM.	NVM	RRAM	Modif.	Simple	Yes	No	No	Both	Analytical	Arithmetic
ISAAC	Accelerator	NN.	MM.	NVM	Memristor	Modif.	Simple	Yes	No	No	Both	Analytical	CNNs&DNNs
Ambit	Accelerator	Bool.	Logical	DRAM	1T1C	Modif.	Simple	No	No	No	Both	Rambus	Bitwise
Pinatubo	Main memory	Bool.	Logical	NVM	Memristor	Modif.	Simple	Yes	No	No	Both	In-house	Bitwise
CIMA	Accelerator	Bool.	Logical	NVM	Memristor	Modif.	Simple	Yes	No	No	Both	Analytical	Bitwise
STT-CiM	Accelerator	Bool.	Logical,+,	NVM	STT-MRAM	Modif.	Simple	Yes	No	No	Both	STT-CiM Sim.	(1)
S-AP	Accelerator	Bool.	Logical,+	SRAM	8T	Modif.	Simple	No	No	No	Both	VASim	ANMLZoo &Regex
DPP	Accelerator	Bool.	Logical,+,*	NVM	RRAM	Modif.	Simple	Yes	No	No	Both	TensorFlow CACTI	PARSEC
R-AP	Accelerator	MM.	Logical,+,*	NVM	RRAM	Modif.	Simple	No	No	No	Both	Hspice	No
DRISA-1T1C	Accelerator	AND, OR, NOT	Logical, +, inner product	DRAM	1T1C	Modif.	Simple	No	No	No	Both	CACTI-3DD, in-house	CNN
IMI	Accelerator	AND, XOR	Logical,+	DRAM	2T2C	Modif.	Simple	No	No	No	Both	PISA-CAKE	(2)
Compute$	Cache	AND, NOR, XOR	Logical,+,*, copy, search	SRAM	6T	Modif.	Simple	No	No	No	Both	SniperSim	(3)
Neural$	Cache	NN.	MM.	SRAM	8T	Modif.	Simple	No	No	No	Both	Prototyped	Inception v3

+: n-bit addition
x: n-bit multiplication
NVM: Non-volatile memory
NN.: Neural network
Conv.: Conventional
Modif.: Modified
MM.: matrix multiplication
Bool.: Boolean
(*): Required read-out during computations
App.: Applications and benchmarks
Analytical: Analytical model
STT-MRAM: spin-transfer torque magnetic RAM
STT-CiM Sim.: STT-CiM device to architecture evaluation framework
(1): string matching, text processing, low-level graphics, data compression, bio-informatic, image processing and cryptography
(2): Gaussian Blur, Alpha Blend, CSV Parsing, ShA1, and Image Fusion
(3): WordCount, StringMatch, DB-BitMap, BMM, and checkpointing

TABLE 2: Comparison among Architectures of CIM-P Classes

4.2 PRIME: A Processing-in-Memory Architecture for Neural Network Computation in ReRAM-based Main Memory

PRIME was proposed in 2016 by C. Pinga, et al., from University of California [76]. PRIME is a resistive RAM based architecture that exploits data level parallelism to perform computations for neural networks (i.e., weighted vector-matrix multiplication) using high-precision multi-level sense amplifiers and some extra logic circuits. The architecture consists of a CPU and multiple RRAM banks; each RRAM bank contains multiple memory crossbars (mem subarrays), full function (FF) and buffer subarrays, as shown in Fig. 9. The CPU sends instructions to the resistive RAM banks; an instruction is either a memory operation (read/write) or a neural network computation. The memory bank performs the request without blocking the CPU, i.e., the CPU continues executing (different) instructions simultaneously. The results are returned to the CPU for further processing.

In the resistive RAM banks, the memory crossbars store data in multiple mats, while the FF and buffer subarrays serve for computation. Special subarray structures are used to enable both neural network computations and memory operations (blue blocks in Fig. 10) feasible. The neural network computations are mainly performed in the FF subarray, while the buffer subarray stores temporary data that needs to be processed; this enables a parallel execution between CPU and FF subarrays. Neural network computations are performed using a vector matrix multiplication between a weighted matrix stored in the FF subarray and a vector stored in the buffer subarray. Additional logic gates such as subtraction and sigmoid units are used to compute negative weights and sigmoid activation functions before the results are sensed by the multi-level sense amplifiers. In order to communicate between the memories, a controller can be use to apply appropriate voltages to the crossbar to move data directly inside it, or use read-out and write-back schemes.

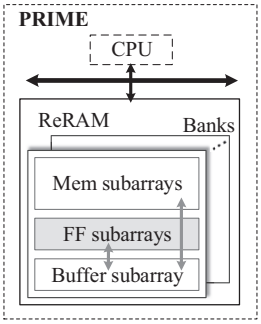


Fig. 9: A Processing-in-Memory Architecture for Neural Network Computation (PRIME) [76]

In addition to the general advantages of CIM-P architectures, PRIME has the following advantages:

- The computations for neural networks are quite mature and can benefit from existing neural network techniques and tools.
- The computations for neural networks do not require a high precision; hence, they are more resilient against device variations.
- Data can be transferred in the crossbar using both direct and indirect schemes.
- The architecture uses non-volatile memory, hence consumes a low energy and has a small footprint.

However, it also has the following limitations:

- The architecture uses non-volatile memory as main memory, which may impact the life time due to limited endurance [73], [74].

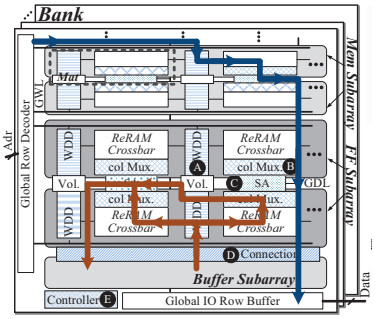


Fig. 10: PRIME's Bank architecture [76]

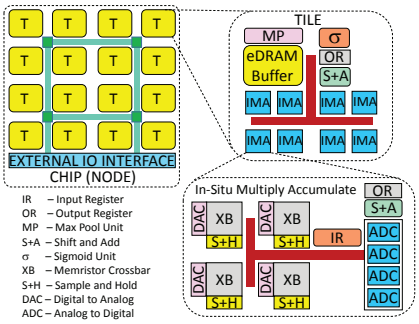


Fig. 11: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic (ISAAC) [78]

- The architecture has to deal with sneak path currents. Possible solutions are mentioned in Section 3.2.
- As the sense amplifies are complex, a trade-off between area and bandwidth has to be made.
- In case general purpose computing is desired, the architecture requires additional compiling techniques and tools to perform conventional Boolean logic functions using neural network computations.

The architecture is synthesized using TSMC CMOS library 60nm and modeled using NVSIM, CACTI-3D and CACTI-IO. It is evaluated using MlBench benchmarks [77] and compared against a CPU-only solution.

4.3 ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic

ISAAC was proposed in 2016 by S. Ali, et al., from University of Utah [78]. ISAAC is a memristor based architecture that performs dot-product computations using the memristor crossbar and CMOS peripheral circuitry to exploit instruction level parallelism. The architecture consists of multiple tiles connected through an on-chip concentrated mesh and an I/O interface, as shown in the left part of Fig. 11. The architecture is only used during the inference phase of machine learning applications, i.e., the phase after training; the inference phase consists of dot product operations to compute convolutions, shift and add operations and sigmoid operations. ISAAC processes inputs from the I/O interface in multiple tiles. After processing, the outputs are communicated through the I/O interface to the outside world or a different ISAAC chip.

Each tile of ISAAC contains multiple In-Situ Multiply Accumulate (IMA) units that are connected through a bus, an eDRAM buffer, output register (OR), and computation units (max-pool, sigmoid and Shift-and-Add (S+A)). Each IMA contains multiple memristor arrays with their DAC and Sample-and-Hold (S+H) units, an Input and Output Register (IR, OR), Shift-and-Add (S+A) and multiple ADC units. Inputs from the I/O interface are delivered to the memristor arrays and are used to perform a dot product computation with the weights that are already stored in the memristor array. The results thereafter go through the S+H units (to temporarily store data before feeding them to ADCs) and S+A units (to accumulate data) if applicable.

Finally, if multiple inputs are fetched, a pipeline is created using IR and OR of the IMAs and tiles. In order to transfer data within a single memory array, a controller can be used to apply appropriate voltages to move data directly inside the memory crossbar, or use read-out and write-back schemes.

In addition to the general advantages of CIM-P architectures, ISAAC has the following advantages:

- The architecture is used as an accelerator, which has a positive impact on the endurance due to infrequent use [73], [74]. In contrast, some CIM-P architectures are used as main memory and therefore require a much higher endurance.
- The computations for neural networks are quite mature and can benefit from existing neural network techniques and tools.
- The computations for neural networks do not require a high precision; hence, they are more resilient against device variation.
- Data can be transferred in the crossbar using both direct and indirect schemes.
- The architecture uses non-volatile memory, hence consumes low energy and has a small footprint.

However, it also has the following limitation:

- The architecture has to deal with sneak path currents. Possible solutions are mentioned in Section 3.2.
- The architecture might suffer from a high overhead due to the need of ADC and DAC converters.
- As the sense amplifies are complex, a trade-off between area and bandwidth has to be made.
- In case general purpose computing is desired, the architecture requires additional compiling techniques and tools to perform conventional Boolean logic functions using neural network computations.

The architecture is evaluated analytically and compared against DaDianNao architecture (which is an ASIC design with embedded DRAM) using a suite of CNN [70], [79] and DNN workloads [80], [81].

4.4 Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology

Ambit was proposed in 2017 by V. Seshadria, et al., from Carnegie Mellon University [82]. Ambit is a DRAM based architecture that performs in-memory instructions using modified peripheral circuits to exploit data level parallelism. This architecture can be plugged into a computer system as an accelerator in a similar manner as a GPU. The architecture consists of an Ambit controller and a 3D-stacked DRAM memory with modified sense amplifiers as shown in Fig. 12. After receiving an instruction from the host processor, Ambit determines whether a normal memory operation or an in-memory instruction should be performed. After performing the required operations, the results are transferred back to the host processor for further processing.

Depending on the type of operation, the Ambit controller activates single or multiple rows in the DRAM memory. The currents are summed up based on the values stored in the DRAM cells, and converted into a digital value using the modified sense amplifiers. To transfer data, Ambit enables *row copy* (RowClone [83]) operations to directly move data inside DRAM memory. Moreover, an indirect scheme can be used a well by having the Ambit controller performing *read* and *write* operations.

In addition to the general advantages of CIM-P architectures, Ambit has the following advantages:

- Data can be transferred in the memory using both direct and indirect schemes.
- The architecture uses DRAM technology, which has several benefits such as: high maturity and endurance, no sneak path currents, and the accessibility to optimized architectures, technology and tools.
- As a DRAM cell is larger than a memristor based cell, it is easier to fit the modified sense amplifiers in the peripheral circuit.

However, it also has the following limitations:

- Computations are currently limited to logical operations. More research is required to map complex functions on the architecture.
- The architecture uses DRAM technology which suffers from a low performance, high energy consumption, large footprint and is difficult to scale down.

The architecture is simulated by Rambus simulator and evaluated against the implementations on multicore Intel Skylake CPU [84], NVIDIA GeForce GTX 745 GPU [85] and HMC 2.0 [86] using logical vector operations and bitmap index application [87], [88].

4.5 Pinatubo: A Processing-in-Memory Architecture for Bulk Bitwise Operations

Pinatubo was proposed in 2016 by S. Li, et al., from University of California [20]. Pinatubo is a non-volatile memory based architecture that exploits data level parallelism by performing bulk bitwise operations using modified sense amplifiers. The architecture consists of a processor with caches, a non-volatile main memory, and modified sense amplifiers (as shown in Fig. 13). The processor sends in-memory instructions to the main memory and also handles

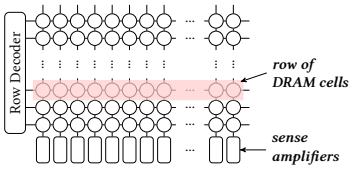


Fig. 12: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology (Ambit) [82]

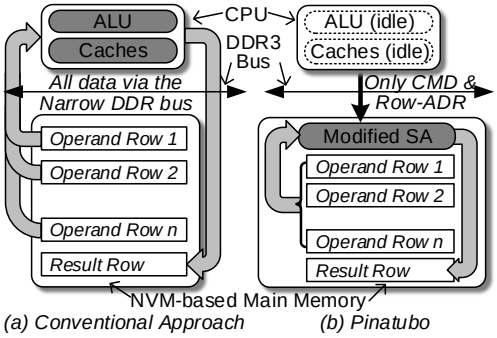


Fig. 13: The Processing-in-Memory Architecture for Bulk Bitwise Operations (Pinatubo) [20]

the operations that cannot be performed on the main memory. After an instruction is sent to the main memory, single or multiple rows of the memory are activated simultaneously depending on the type of instructions (i.e., normal read or in-memory instructions). The modified sense amplifiers thereafter perform a read-out operation to produce the results which can be a normal read or a bitwise vector operations. In cases needed, the results are transferred back to the processor for further processing.

The main memory architecture is shown in Fig. 14; it consists of multiple banks which are further divided into mats. Note that the modified sense amplifiers can only perform bitwise vector operations on data residing in the same mat. For operations where the data resides in different mats whether on the same bank or not, extra logic gates (e.g., AND, OR) are used to perform the operations. Communication can be performed by enabling two memory rows for direct copy operations, or using the buffers and read-out operations for indirect data transfer.

In addition to the general advantages of CIM-P architectures, Pinatubo has the following advantages:

- Data can be transferred in the memory using both direct and indirect schemes.
- The architecture uses non-volatile memory, hence consumes low energy and has a small footprint.

However, it also has the following limitations:

- The architecture uses non-volatile memory as main memory, which may impact the life time due to limited endurance [73], [74].

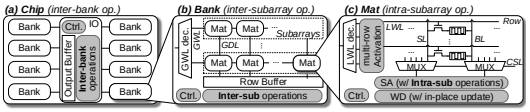


Fig. 14: Main memory of Pinatubo [20]

- Computations currently include only logical operations. More research is required to map complex functions on the architecture.
- The architecture has to deal with sneak path currents. Possible solutions are mentioned in Section 3.2.
- As the sense amplifies are complex, a trade-off between area and bandwidth has to be made.
- Efforts are required to use this architecture with a host processor. For example, the instruction set of the host processor has to be adapted and additional software support is needed to maximally exploit the performance.

The architecture is simulated using an in-house cycle accurate simulator modified from Sniper [89] and evaluated using three applications: vector OR operations, bitmap-based BFS for graph processing [90] and bitmap-based database using Fastbit [91].

4.6 CIMA: Computation-in-Memory Accelerator

CIMA was proposed in 2017 by H.A. Du Nguyen, et al., from Delft University of Technology [45]. CIMA is a resistive based accelerator that exploits data level parallelism by performing computations with custom sense amplifiers. The architecture consists of a conventional processor, caches, CIM accelerator, main memory DRAM and external memory (as shown in Fig. 15). The processor fetches, decodes and executes non-intensive memory parts of an application and off-loads the memory intensive parts to the CIM accelerator. Similarly as in Pinatubo, the CIM accelerator performs operations by activating one or multiple wordlines. The difference with Pinatubo however is that CIMA is used as an accelerator and has a more efficient sense amplifiers. In case needed, the results are transferred back to the processor for further processing.

The CIM accelerator consists of a CIM controller, peripheral circuits (including decoder, voltage driver, and sense amplifiers), and a memristor crossbar. The CIM controller receives instructions from the processor and performs operations by sensing the current of two or more activated rows of the crossbar. Based on the type of operation, the customized sense amplifiers compute the output based on this current. CIMA uses scouting logic [21]; currently, it can only perform bitwise logical operations. However, more operations such as addition, vector-matrix multiplication and matrix-matrix multiplication have demonstrated to be feasible [92]. The data transfer in the memory can be performed by enabling two memory rows using direct copy operations, or using indirect read-out and write-back operations.

In addition to the general advantages of CIM-P architectures, CIMA has the following advantages:

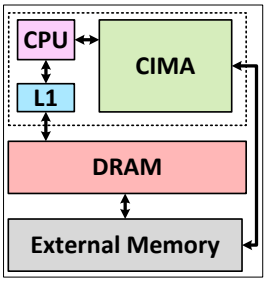


Fig. 15: Computation-in-Memory Accelerator (CIMA) [45]

- The architecture is used as an accelerator, which has a positive impact on the endurance due to infrequent use [73], [74].
- Data can be transferred in the memory using both direct and indirect schemes.
- The architecture uses non-volatile memory, hence consumes low energy and has a small footprint.

However, it also has the following limitations:

- Computations currently include only logical operations. More research is required to map complex functions on the architecture.
- The architecture has to deal with sneak path currents. Possible solutions are mentioned in Section 3.2.
- As the sense amplifies are complex, a trade-off between area and bandwidth has to be made.
- Additional software support (i.e., profiling and extracting memory intensive kernels) is required to maximally exploit the accelerator performance.

The architecture is evaluated against a theoretical model of a conventional multicore architecture.

4.7 STT-CiM: Computing in Memory Spin-Transfer Torque Magnetic RAM

STT-CiM was proposed in 2017 by S. Jain, et al., from Purdue University [93]. STT-CiM is a Spin-Transfer Torque Magnetic RAM based architecture that that exploits data level parallelism by performing computations using both modified sense amplifiers and some additional CMOS gates. The architecture consists of a conventional architecture with a STT-MRAM used as scratch-pad memory. This scratch-pad memory is equipped with the capability to perform in-memory instructions. These instructions are sent from the main processor.

The STT-CiM contains a CiM decoder, an array of memory cells, enhanced address decoder and modified sensing circuitry to perform computations, as shown in Fig. 16. Based on the in-memory instruction, the enhanced address decoder activates one (for normal read) or multiple rows (for computations) of the memory array. The CiM decoder determines simultaneously the reference currents of the sense amplifiers. For example, in case an addition is executed, the set of logic gates for addition is enabled. The results are captured by the modified sense amplifiers. Data transfer can be performed by enabling two memory rows for direct copy

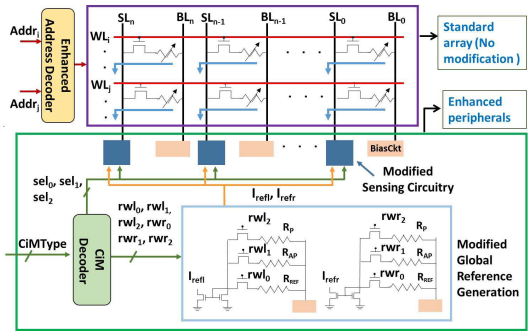


Fig. 16: Computing in Memory Spin-Transfer Torque Magnetic RAM (STT-CiM) [93]

operations, or using the buffers and read-out operations for indirect copy operations.

In addition to the general advantages of CIM-P architectures, STT-CiM has the following advantages:

- The architecture is used as an accelerator (i.e., scratch-pad memory), which has a positive impact on the endurance due to infrequent use [73], [74].
- Computations currently include both logical operations and addition.
- The data transfer may include both direct and indirect schemes.
- The architecture uses non-volatile memory, hence consumes low energy and has a small footprint.

However, it also has the following limitations:

- The architecture has to deal with sneak path currents. Possible solutions are mentioned in Section 3.2.
- As the sense amplifies are complex, a trade-off between area and bandwidth has to be made.
- Additional software support (i.e., profiling and extracting memory intensive kernels) is required to maximally exploit the accelerator performance.

The architecture is evaluated using the STT-CiM device-to-architecture evaluation framework [93] and a set of benchmarks including string matching, text processing, low-level graphics, data compression, bio-informatic, image processing and cryptography.

4.8 S-AP: Cache Automaton

S-AP was proposed in 2017 by A. Subramaniyan, et al., from University of Michigan [94]. The architecture targets an automata processor which exploits data level parallelism by performing computations using state machines. An automata processor contains two main components: the State Transition Elements (STEs) and the routing matrix; the STE stores the accepting states, while the routing matrix stores the state transitions as shown in Fig. 17. The automata processor accepts one input symbol at a time, generates next active states and decides whether a complete input string is accepted or not.

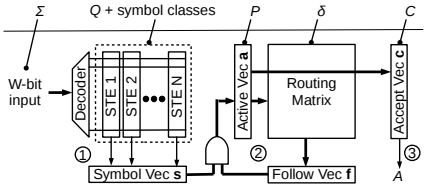


Fig. 17: General Architecture for Automata Processor [95]

The architecture consists of STEs and a routing matrix which are implemented using SRAM technology. Each SRAM column corresponds to an STE which stores the accepting states in SRAM cells. The input symbol is fed to all the STEs simultaneously. The sense amplifiers collect a dot-product results of a vector-matrix multiplication. The output of the STE together with the routing matrix are used to determine the next active states; this process is carried on until all input symbols are processed. In case the one or more final active states are part of the acceptance states, it means that the input string has been matched with the corresponding pattern of the acceptance state. Note that data transfer inside the automata processor is carried out using the routing matrix.

In addition to the general advantages of CIM-P architectures, S-AP has the following advantages:

- Computations may include logical and arithmetic operations using automata processing.
- Data can be transferred using both direct and indirect schemes.
- The architecture uses SRAM technology, which has several benefits such as maturity, high endurance, no sneak path currents, and may benefit for the the existing optimizing techniques and tools.
- As an SRAM cell is relatively large as compared to a memristor/DRAM cell, it is easier to fit the modified sense amplifiers in the peripheral circuit.
- The automata processing techniques and tooling are quite mature, hence it is feasible to explore many applications using automata processing.

However, it also has the following limitations:

- The architecture uses SRAM technology which suffers from high energy consumption, low scalability and large footprint.
- The architecture requires additional compiling techniques and tools to perform conventional Boolean logic functions using automata processing.

The S-AP is simulated using VASim [96] and evaluated against DRAM-AP and x86 CPU using ANMLZoo [96] and the Regex [97] benchmark suites.

4.9 DPP: Data Parallel Processor

DPP was proposed in 2018 by D. Fujiki, et al. from University of Michigan [22]. DPP is a RRAM-based architecture that exploits instruction and data level parallelism by performing computations using a combination of RRAM-based dot-product operations and LUTs. The architecture

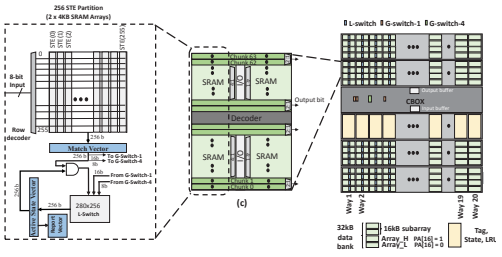


Fig. 18: SRAM Automata Processor(S-AP) [94]

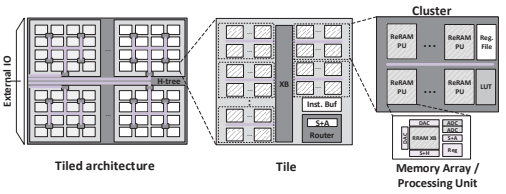


Fig. 19: Data Parallel Processor (DPP) [22]

consists of multiple RRAM tiles connected as an H-tree; each tile has multiple clusters and some logic units (as shown in Fig. 19). Tiles and clusters form a SIMD-like processor that performs the parallel operations. The architecture is considered as a general purposed architecture as it can perform all primitive functions such as logical, arithmetic, shift and copy operations.

In addition to clusters, each tile has several units to support the computations including instruction buffer, Shift and Add (S+A), and router. Each cluster additionally has one or more computational units; they are Shift and Add (S+A), Sample and Hold (S+H), DAC and ADC, a LUT and register file (as shown in the right part of Fig. 19). While reading from the high latency RRAM, other units are simultaneously used for processing. Therefore, the S+H is used to read data (in the form of a current) from the RRAM array and temporarily store it. Once that data is needed, it is fed to an ADC to convert the analog value to a digital value. The S+A is used to perform carry propagation in a multiple-bit addition. DAC is used to apply a digital value to the RRAM array with an appropriate control voltage. Some complex functions that cannot be realized with these units are performed using LUTs and register file in each cluster. Data transfer can be performed by enabling two memory rows for direct copy operations, or using the buffers and read-out operations for indirect copy operations.

In addition to the general advantages of CIM-P architectures, DPP has the following advantages:

- Computations include both logical operations and simple arithmetic operations (i.e., addition, multiplication).
- Data can be transferred using both direct and indirect schemes.
- The architecture uses non-volatile memory, hence consumes low energy and has a small footprint.
- This architecture is claimed to be general purpose, hence it can exploits existing instruction set, compiling techniques and tools, as well as applications.

However, it also has the following limitations:

- The architecture uses non-volatile memory as main memory, which may impact the life time due to limited endurance [73], [74].
- The architecture has to deal with sneak path currents. Possible solutions as mentioned in Section 3.
- As the sense amplifies are complex, a trade-off between area and bandwidth has to be made.

The architecture potential was simulated and evaluated against CPU Intel Xeon E5-2697 using a subset of PARSEC benchmarks [98] and against GPU NVIDIA Titan XP using Rodinia benchmarks [99].

4.10 R-AP: Resistive RAM Automata Processor

R-AP was proposed in 2018 by J. Yu, et al. from Delft University of Technology [95]. R-AP is an automata processor that exploits data level parallelism by performing computations similarly as mentioned in Section 4.8. The working principle of R-AP is similar to the S-AP. In contrast to S-AP, R-AP uses RRAM based STEs and routing matrice, as shown in Fig. 20.

In addition to the general advantages of CIM-P architectures, R-AP has the following advantages:

- The architecture is used as a read-favoured accelerator, which has a positive impact on the endurance due to infrequent use [73], [74].
- Automata processing can be used to perform both logical and arithmetic operations in general.
- Data can be transferred using both direct and indirect schemes.
  item The architecture uses non-volatile memory, hence consumes low energy and has a small footprint.
- The automata processing techniques and tooling are quite mature, hence it is feasible to explore many applications using automata processing.

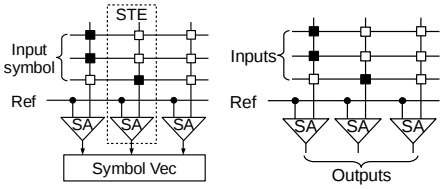
However, it also has the following limitations:

- The architecture has to deal with sneak path currents. Possible solutions are mentioned in Section 3.2.
- As the sense amplifies are complex, a trade-off between area and bandwidth has to be made.
- The architecture requires additional compiling techniques and tools to perform conventional Boolean logic functions using automata processing.

The architecture has been validated using circuit level simulations and evaluated against S-AP.

4.11 DRISA-1T1C: A DRAM-based Reconfigurable In-Situ Accelerator with 1T1C design

DRISA-1T1C was proposed in 2016 by S. Li, et al. from University of California [67]. It is a DRAM based architecture that exploits data parallelism by performing NOR gate inside DRAM cells [63]. The architecture consists of a DRAM memory organized in a hierarchy of banks, sub-arrays and



(a) Used as STEs (b) Used as routers

Fig. 20: Resistive RAM Automata Processor (R-AP) [95]

mat; each levels are controlled by their corresponding controllers as shown in Fig. 21(a),(b) and (c). The banks are connected through global bus (gbus) while communication among subarrays are carried out using bank buffers (bBuf). The mats perform both data storage and computations.

The memory mats consist of cell regions for both data and computations, and peripheral circuits including calc-SA, intra/inter-lance shifter (SHF) and lane forwarding unit (FWD). The cell regions contains multiple standard DRAM cells. In order to perform computations, complex sense amplifiers are required. Logical operations including AND and OR are performed using this complex sense amplifiers similarly as in the case of Ambit [82]. Other function such as NOT are performed using extra logic gates and a latch; in this scheme, one operands row is read out and stored in a latch, thereafter, another operands row is read out and fed into a logic gate together with the previous operand rows in the latch. The SA (also called calc-SA) cooperates with extra logic circuitry such as SHF and FWD to perform complex functions such as addition, copy and inner product.

In addition to the general advantages of CIM-P architectures, DRISA-1T1C has the following advantages:

- The architecture is used as an accelerator (i.e., scratch-pad memory), which has a positive impact on the endurance due to infrequent use [73], [74].
- The data transfer may include both direct and indirect schemes.
- The architecture uses DRAM technology, which has several benefits such as: high maturity and endurance, no sneak path currents, and the accessibility to optimized architectures, technology and tools.
- As a DRAM cell is larger than a memristor based cell, it is easier to fit the modified sense amplifiers in the peripheral circuit.

However, it also has the following limitations:

- As the peripheral circuits are complex, a trade-off between area and bandwidth has to be made.
- The architecture suffers destructive read, hence an extra copy of the operand rows are required.
- The architecture uses DRAM technology which suffers from a low performance, high energy consumption, large footprint and is difficult to scale down.

The architecture is simulated and evaluated against GPU TITAN X [68] using four CNN applications including 8-layer AlexNet [69], 16-layer VGG-16 [70], 19-layer VGG-19, and 152-layer ResNet-152 [71].

4.12 IMI: In-Memory Intelligence

IMI was proposed in 2017 by T. Finkbeiner, et al., from Micron Technology [100]. It is a DRAM based architecture that exploits data parallelism by performing computations using bit-serial computing elements attached to the DRAM's sense amplifiers. The architecture consists of a host system on chip, double data rate (DDR) interface, regular DRAM and in-memory-computation DRAM arrays with accumulator as shown in Fig. 22. The host system on chip communicates with both regular and in-memory-computation DRAM through the DDR interface. In-memory computations are performed inside of the in-memory-computation DRAM.

The in-memory-computation DRAM includes multiple banks, each with a scalar bank processing control unit (BPCU). Each bank contains multiple subarrays. Each subarray comprises of 2T2C cells and computing elements that is capable of performing XOR and AND operations. Complex functions are based on these operations. The scalar BPCU controls both the conventional memory operations and in-memory computation in vector parallel form.

In addition to the general advantages of CIM-P architectures, IMI has the following advantages:

- The data transfer may include both direct and indirect schemes.
- The architecture uses DRAM technology, which has several benefits such as: high maturity and endurance, no sneak path currents, and the accessibility to optimized architectures, technology and tools.
- As a DRAM cell is larger than a memristor based cell, it is easier to fit the modified sense amplifiers in the peripheral circuit.

However, it also has the following limitations:

- As the peripheral circuits are complex, a trade-off between area and bandwidth has to be made.
- The architecture uses DRAM technology which suffers from a low performance, high energy consumption, large footprint and is difficult to scale down.

The architecture is simulated using a custom-built simulator called PISA-CAKE and evaluated against GPUs and AMR processors using several benchmarks such as Gaussian Blur, Alpha Blend, CSV Parsing, ShA1, and Image Fusion [100].

4.13 Compute$: Compute caches

Compute$ was proposed in 2017 by S. Aga, et al., from University of Michigan [101]. It is a SRAM based architecture that exploits data parallelism by performing computations using two differential sense amplifiers [102]. The architecture consists of multiple processors with their own L1, L2 and L3 caches; each cache is a SRAM memory with in-memoy computation capability as shown in Fig. 23(a). The applications are mapped to exploit the data locality in L2 caches. The processors and cache hierarchy are designed to aware of this data locality and in-memory computations.

Each cache contains a cache controller, and multiple subarrays organized in an H-tree form as shown in Fig. 23(b). The cache controller is extended from the conventional

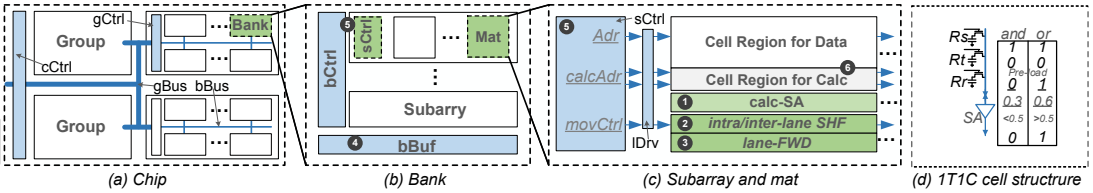


Fig. 21: A DRAM-based Reconfigurable In-Situ Accelerator (DRISA) [67]

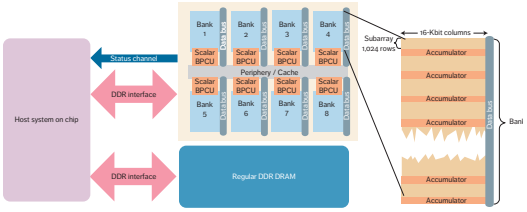


Fig. 22: In-memory Intellgence overview [100]

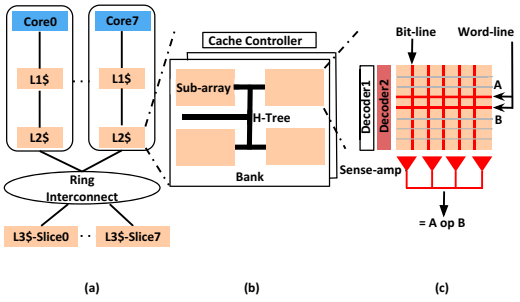


Fig. 23: Compute$ overview [101]

cache controller to manage the parallel executions of multiple instructions in multiple banks. The cache controller also determine which cache level to execute a function to ensure the data locality. Computations are performed by enabling two rows of a subarrays simultaneously and activing the functions of a sense amplifier. The sense amplifiers are designed to execute AND, NOR, XOR, copy, search, comparision and carryless multiplication [101].

In addition to the general advantages of CIM-P architectures, Compute$ has the following advantages:

- The data transfer may include both direct and indirect schemes.
- The architecture uses SRAM technology, which has several benefits such as maturity, high endurance, no sneak path currents, and may benefit for the the existing optimizing techniques and tools.
- As an SRAM cell is relatively large as compared to a memristor/DRAM cell, it is easier to fit the modified sense amplifiers in the peripheral circuit.

However, it also has the following limitations:

Fig. 24: Neural$ overview [103]

- As the peripheral circuits are complex, a trade-off between area and bandwidth has to be made.
- The architecture uses SRAM technology which suffers from high energy consumption, low scalability and large footprint.

The architecture is simulated using SniperSim [89] and evaluated using multiple benchmarks including WordCount, StringMatch, DB-BitMap, BMM, and checkpointing [101].

4.14 Neural$: Bit-serial in-cache acceleration of deep neural networks

Neural$ was proposed in 2018 by C. Eckert, et al., from University of Michigan and Intel Corporation [103]. It is a SRAM based architecture that exploits data parallelism by performing computations for neural network using modified peripheral circuits. The architecture includes a multicore processor with multiple cache slices as shown in the leftmost part of Fig. 24. Each cache slide contains multiple banks controlled by a CBOX. Each bank comprised of multiple subarrays with their own peripheral circuits to support in-memory computation.

The subarrays stores data inside the SRAM cells and performs computations using two conventional sense amplifiers and logic gates. The sense amplifiers read out two rows of data simultaneously and feed the results to a set of logic gates. With these logic units, the architecture is capable of performing complex function such as addition, multiplication and reduction.

In addition to the general advantages of CIM-P architectures, Neural$ has the following advantages:

- The data transfer may include both direct and indirect schemes.

- The architecture uses SRAM technology, which has several benefits such as maturity, high endurance, no sneak path currents, and may benefit for the the existing optimizing techniques and tools.
- As an SRAM cell is relatively large as compared to a memristor/DRAM cell, it is easier to fit the modified sense amplifiers in the peripheral circuit.

However, it also has the following limitations:

- As the peripheral circuits are complex, a trade-off between area and bandwidth has to be made.
- The architecture uses SRAM technology which suffers from high energy consumption, low scalability and large footprint.
- In case general purpose computing is desired, the architecture requires additional compiling techniques and tools to perform conventional Boolean logic functions using neural network computations.

The architecture is demonstrated as a prototype and evaluated against multicore CPU Xeon E5 [104] and GPU Titan Xp [68] for the Inception v3 model [105].

5 DISCUSSION

In this section, we try to analytically evaluate all architectures in four aspects: memory array, peripheral circuits, controllers, interconnect network, and applications. For the prior four aspects, we consider its usage, complexity and performance impact as they are components of the architecture. For the application, we evaluate the potential ranges of application for each group of architectures.

In terms of **memory array**, we first consider how the memory array is used in each category. CIM-A and CIM-P use memory array as temporary and long-term storage, and computing unit, while COM-N and COM-P use memory array only for storage. In general, CIM-A and CIM-P architectures exploit more functionality of the memory array. Therefore, more requirements are applied for memory array such as higher endurance, lower write voltage and energy, and isolation among computing unit to ensure robustness and parallelism. Meanwhile, COM-N and COM-P architectures requires less from memory array which are specifically designed to optimize capacity and read speed. Second, we consider how much complexity is required to adapt the memory array to the new method to perform computation. In general, only the architectures in CIM-A category require modifications inside the memory array. CIM-P class might require some modification in cell pitch to fit their peripheral circuit, however, these changes are considered as a performance trade-off rather than in the case of CIM-A category. COM-N and COM-F requires no change in the memory array, however, using emerging memory such as RRAM for storage might suffer from low endurance, and high write latency and energy; this will impact the performance of the architectures. In particular, CRS and MPU require more modification than other CIM-A architectures due to the change in their cell structure; note that CRS requires a cell that combines two memristors while MPU requires a cell that has asymmetric voltages to switch from high to low resistance and vice versa. Third, we consider the performance impact of the memory array. As the memory array is used as multiple purposes for CIM-A and CIM-P architectures, it increases overall performance as a results of no memory bottleneck, reduced idle time and reduced static power consumption. This is not the case for COM-N and COM-P architectures where memory bottleneck exists and reduces the overall performance and energy.

In terms of **peripheral circuits**, we first consider the peripheral circuit usage in each category. Except for CIM-P where peripheral circuit plays an crucial role in computation, other classes uses peripheral circuits mainly for reading/writing data from/to memory. In general, CIM-P architectures perform part of computation using their peripheral circuit, hence it requires a more functionality from the peripheral circuit than the architectures in other classes. For example. the peripheral circuit is required to perform also parts of logical or arithmetic operations, as in the case of most CIM-P architectures such as Pinatubo, STT-CiM, DPP. For other architectures, the peripheral circuit performs conventional memory operations, which do not require more functionality. Second, we consider the complexity of the peripheral circuit. In general, CIM-P class requires more complex peripheral circuit while other classes require conventional peripheral circuit. Complex peripheral circuit can be either (i) modifying the peripheral circuit itself (i.e., row decoder or sense amplifiers), or (ii) adding more logic gates after the peripheral circuit to perform complex functions. The first options include modifications that enables multiple rows concurrently, reads multiple-level values at the sense amplifiers, and combines the aforementioned. In particular, all architectures in CIM-P class requires concurrent multiple row enabling; some require only limited number of rows enabling to perform logical operations such as Ambit, Pinatubo, CIMA, STT-CiM, and DPP while others require all the rows of a crossbar enabled to perform matrix multiplication operations. In addition, PRIME, ISAAC, and DPP demand also a read circuit that can differentiate multiple levels (i.e., ADC) while other architectures in CIM-P class only requires read circuit to differentiate between 0 and 1; note that the presence of ADC increases dramatically the complexity of the peripheral circuit. Third, we consider the performance impact of peripheral circuit to overall architecture performance. As most architectures in CIM-P class depends on their peripheral circuit to perform computation, the number of peripheral circuit elements determines the maximum parallelism that can be obtained. However, as the peripheral circuit of some architectures are more complex than other, there is a performance-area trade-off in locating appropriate amount of peripheral circuit. Therefore, CIM-P class architectures such as PRIME, ISAAC, and DPP achieves lower parallelism than other CIM-P class architectures. In addition, the architectures which use non-volatile memory can fit fewer of these complex peripheral circuits than the architectures which use conventional larger foot-print memory technology (i.e., DRAM); hence, Ambit has positive impact on performance than other CIM-P architectures. For other classes, as peripheral circuit play less important role, it does not impact their parallelism.

In terms of **controller**, we first consider the controller usage in each architecture; note that controller here are considered as in-memory computation controller. In general, COM-N and COM-F architectures do not require this

controller as a conventional memory controller is sufficient. CIM-A and CIM-P architectures which perform (parts of) computation require a controller as memory is a passive device while performing computation requires multiple activating signals at different timing; hence the controller for these architectures controls both memory operations, computations and possibly communication [50]; therefore, more bottleneck are expected for the controller of CIM-A and CIM-P architectures. Second, we consider the controller complexity for each architecture. In general, the controller for CIM-A architectures is more complex than that of CIM-P architectures as computation in CIM-A architectures normally requires multiple steps with various control signals; whereas CIM-P class requires normally a single step and fixed control (read) signals. In addition, some architectures require also communication happening within controllers; this increases the complexity of the controller. For example, CRS architecture has a relatively more complex controller than other CIM-A architecture because of its communication can only carried out using controller [48]. Furthermore, architectures using DRAM technology has a more mature, optimized controller than emerging memory such as STT-MRAM or RRAM; hence, Ambit's controller is less complex than other CIM-P architecture. Third, we consider the performance impact of the controller. In general, a complex controller reduces overall performance as it increases the length of the critical path, which reduces architecture's frequency. Therefore, CIM-A architecture's controller is less efficient than that of CIM-P architecture. In particular, CRS's controller is considered as negative impact on overall performance while Ambit's controller is considered as positive impact on overall performance.

In terms of **interconnect network**, we first consider the interconnect network usage in each architecture. In general, conventional interconnect network can be used for COM-N and COM-F architectures, while novel interconnect network schemes still need to be explored for CIM-A and CIM-P architectures [50], [51]. These novel schemes currently include (1) only using memristor crossbar, (2) using CMOS circuits (controller and peripheral circuits) , and (3) using both memristor crossbar and CMOS circuits. Most architectures in CIM-A and CIM-P class can use all three schemes, except some architectures with unconventional memory cell usage such as CRS and ReAP. Second, we consider the interconnect network complexity for each architecture. In general, the interconnect network for CIM-A architectures is more complicated, as it happens inside the memory crossbar together with computations, which might suffer from controller complexity and lack of isolation. Interconnect network of CIM-P architectures are more flexible as results are produced outside the memory crossbar, and can be communicated outside of memory crossbar using conventional mature interconnect network. Third, we consider the impact of the interconnect network on the architecture's performance. In general, reducing communication can increase overall performance, especially when (i) interconnect network basic operation is not efficient, and (ii) the number of required communication is huge. Therefore, interconnect network in CIM-A architectures have higher impact on overall performance than on CIM-P architectures. Furthermore, flexibility to use multiple schemes can help reduce the negative impact on system performance; hence, CRS and ReAP have higher negative impact on performance than other architectures.

In terms of **application**, we look at the range of applications that can be used for each architecture. In general, CIM architectures can be more efficient than COM architectures for data intensive applications due to their capabilities in reducing memory bottleneck. For CIM-A architectures and some CIM-P architectures (e.g., Pinatubo, CIMA, STT-CiM), there are currently limited types of operations can be efficiently performed on these architectures; hence, limited range of applications can be mapped on these architectures; this includes bulk bitwise applications such as database processing, graph processing, image processing, security and biosequencing application [20], [82], [106], [107], [108], [109]. In addition, only several architectures are considered as general purpose computers such as ReAP, DPP, FlexRAM, SM, COM-F architectures, etc. Other architectures targets specific applications such as vector processing (e.g., PLiM, ReVAMP, VIRAM, DIVA, AMC, etc.), automata processing (S-AP, D-AP, and R-AP), and neural computation (PRIME and ISAAC).

6 Conclusion

In this paper, we have proposed a classification including four groups of computer architectures. Moreover, nearly 30 selected architectures were presented and evaluated quantitatively. The work shows that a potential architecture does not only require to be memory bottleneck free, but also energy and area efficient. In order to accomplish that, it is the joint effort of both architectural improvement and technology development. The relationship between the two are closer and closer in later proposed architectures. This work also shows that architectures are not changing dramatically, but gradually with small changes and technology developments. Indeed, technology enables the feasibility and potential of the same architecture in the past. In general, the classification does not only present an overview of existing architectures, but also predicts the potential of future architecture variants.

Acknowledgments

The authors would like to thank Xie Lei and Razvan Nane for a lot of intensive and useful discussions. The results presented in this paper have been obtained in the framework of the project Computation-in-memory architecture based on resistive devices (MNEMOSENE), which has received funding from the European Unions Horizon 2020 research and innovation programme under grant agreement No 780215.

References

[1] ITRS. (2010) ITRS ERD report. [Online]. Available: http://www.itrs.net

[2] S. Borkar, "Design challenges of technology scaling," *Micro, IEEE*, vol. 19, no. 4, pp. 23–29, Jul 1999.

[3] J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach*. Elsevier, 2011.

[4] S. Hamdioui, S. Kvatinsky, G. Cauwenberghs, L. Xie, N. Wald, S. Joshi, H. M. Elsayed, H. Corporaal, and K. Bertels, "Memristor for computing: Myth or reality?" in *Proceedings of the Conference on Design, Automation & Test in Europe*. European Design and Automation Association, 2017, pp. 722–731.

[5] D. A. Patterson, "Future of computer architecture," in *Berkeley EECS Annual Research Symposium (BEARS), College of Engineering, UC Berkeley, US*, 2006.

[6] M. Johnson and M. Johnson, *Superscalar microprocessor design*. prentice Hall Englewood Cliffs, New Jersey, 1991, vol. 77.

[7] S. Gochman, A. Mendelson, A. Naveh, and E. Rotem, "Introduction to intel core duo processor architecture." *Intel Technology Journal*, vol. 10, no. 2, 2006.

[8] H. S. Stone, "A logic-in-memory computer," *IEEE Transactions on Computers*, vol. C-19, no. 1, pp. 73–78, 1970.

[9] Y. Kang, W. Huang, S.-M. Yoo, D. Keen, Z. Ge, V. Lam, P. Pattnaik, and J. Torrellas, "Flexram: Toward an advanced intelligent memory system," in *Computer Design (ICCD), 2012 IEEE 30th International Conference on*. IEEE, 2012, pp. 5–14.

[10] J. Draper, J. T. Barrett, J. Sondeen, S. Mediratta, C. W. Kang, I. Kim, and G. Daglikoca, "A prototype processing-in-memory (pim) chip for the data-intensive architecture (diva) system," *Journal of VLSI signal processing systems for signal, image and video technology*, vol. 40, no. 1, pp. 73–84, 2005.

[11] C. E. Kozyrakis, S. Perissakis, D. Patterson, T. Anderson, K. Asanovic, N. Cardwell, R. Fromm, J. Golbus, B. Gribstad, K. Keeton *et al.*, "Scalable processors in the billion-transistor era: Iram," *Computer*, vol. 30, no. 9, pp. 75–78, 1997.

[12] S. S. Iyer and H. L. Kalter, "Embedded dram technology: opportunities and challenges," *IEEE spectrum*, vol. 36, no. 4, pp. 56–64, 1999.

[13] D. Keitel-Schulz and N. Wehn, "Issues in embedded dram development and applications," in *Proceedings of the 11th international symposium on System synthesis*. IEEE Computer Society, 1998, pp. 23–31.

[14] ——, "Embedded dram development: Technology, physical design, and application issues," *IEEE Design & Test of Computers*, vol. 18, no. 3, pp. 7–15, 2001.

[15] M. Di Ventra and Y. V. Pershin, "Memcomputing: a computing paradigm to store and process information on the same physical platform," *Nature Physics*, vol. 9, no. 4, pp. 200–202, Apr. 2013, arXiv:1211.4487 [cond-mat, q-bio].

[16] S. Hamdioui, L. Xie, H. A. D. Nguyen, M. Taouil, K. Bertels, H. Corporaal, H. Jiao, F. Catthoor, D. Wouters, L. Eike *et al.*, "Memristor based computation-in-memory architecture for data-intensive applications," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*. EDA Consortium, 2015, pp. 1718–1725.

[17] S. Hamdioui, K. L. M. Bertels, and M. Taouil, "Computing device for big data applications using memristors," Nov. 21 2017, uS Patent 9,824,753.

[18] H. A. Du Nguyen, L. Xie, M. Taouil, R. Nane, S. Hamdioui, and K. Bertels, "On the implementation of computation-in-memory parallel adder," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2017.

[19] D. Bhattacharjee, R. Devadoss, and A. Chattopadhyay, "Revamp: Reram based vliw architecture for in-memory computing," in *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2017, pp. 782–787.

[20] S. Li, C. Xu, Q. Zou, J. Zhao, Y. Lu, and Y. Xie, "Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories," in *Design Automation Conference (DAC), 2016 53nd ACM/EDAC/IEEE*. IEEE, 2016, pp. 1–6.

[21] L. Xie, H. A. D. Nguyen, J. Yu, A. Kaichouhi, M. Taouil, M. AlFailakawi, and S. Hamdioui, "Scouting logic: A novel memristor-based logic design for resistive computing," in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2017, pp. 335–340.

[22] D. Fujiki, S. Mahlke, and R. Das, "In-memory data parallel processor," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 2018, pp. 1–14.

[23] O. Mutlu, "Memory scaling: A systems architecture perspective," in *Memory Workshop (IMW), 2013 5th IEEE International*. IEEE, 2013, pp. 21–25.

[24] J. A. Mandelman, R. H. Dennard, G. B. Bronner, J. K. DeBrosse, R. Divakaruni, Y. Li, and C. J. Radens, "Challenges and future directions for the scaling of dynamic random-access memory (dram)," *IBM Journal of Research and Development*, vol. 46, no. 2.3, pp. 187–212, 2002.

[25] A. Maislos *et al.*, "A new era in embedded flash memory," *Flash memory summit*, 2011.

[26] G. S. Sandhu, "Emerging memories technology landscape," in *Non-Volatile Memory Technology Symposium (NVMTS), 2013 13th*. IEEE, 2013, pp. 1–5.

[27] R. Waser, "Redox-based resistive switching memories," *Journal of nanoscience and nanotechnology*, vol. 12, no. 10, pp. 7628–7640, 2012.

[28] S. Bhatti, R. Sbiaa, A. Hirohata, H. Ohno, S. Fukami, and S. Piramanayagam, "Spintronics based random access memory: a review," *Materials Today*, 2017.

[29] E. Chen, D. Apalkov, Z. Diao, A. Driskill-Smith, D. Druist, D. Lottis, V. Nikitin, X. Tang, S. Watts, S. Wang *et al.*, "Advances and future prospects of spin-transfer torque random access memory," *IEEE Transactions on Magnetics*, vol. 46, no. 6, pp. 1873–1878, 2010.

[30] J. E. Green, J. W. Choi, A. Boukai, Y. Bunimovich, E. Johnston-Halperin, E. DeIonno, Y. Luo, B. A. Sheriff, K. Xu, Y. S. Shin *et al.*, "A 160-kilobit molecular electronic memory patterned at 10 11 bits per square centimetre," *Nature*, vol. 445, no. 7126, p. 414, 2007.

[31] M. Radosavljević, M. Freitag, K. Thadani, and A. Johnson, "Nonvolatile molecular memory elements based on ambipolar nanotube field effect transistors," *Nano Letters*, vol. 2, no. 7, pp. 761–764, 2002.

[32] C. Li, D. Zhang, X. Liu, S. Han, T. Tang, C. Zhou, W. Fan, J. Koehne, J. Han, M. Meyyappan *et al.*, "Fabrication approach for molecular memory arrays," *Applied physics letters*, vol. 82, no. 4, pp. 645–647, 2003.

[33] C. Li, W. Fan, B. Lei, D. Zhang, S. Han, T. Tang, X. Liu, Z. Liu, S. Asano, M. Meyyappan *et al.*, "Multilevel memory based on molecular devices," *Applied Physics Letters*, vol. 84, no. 11, pp. 1949–1951, 2004.

[34] B. Halg, "On a micro-electro-mechanical nonvolatile memory cell," *IEEE Transactions on Electron Devices*, vol. 37, no. 10, pp. 2230–2236, 1990.

[35] R. Cabrera, E. Merced, and N. Sepúlveda, "A micro-electro-mechanical memory based on the structural phase transition of vo2," *physica status solidi (a)*, vol. 210, no. 9, pp. 1704–1711, 2013.

[36] D. M. Tullsen, S. J. Eggers, and H. M. Levy, "Simultaneous multithreading: Maximizing on-chip parallelism," in *ACM SIGARCH Computer Architecture News*, vol. 23, no. 2. ACM, 1995, pp. 392–403.

[37] S. J. Eggers, J. S. Emer, H. M. Levy, J. L. Lo, R. L. Stamm, and D. M. Tullsen, "Simultaneous multithreading: A platform for next-generation processors," *IEEE micro*, vol. 17, no. 5, pp. 12–19, 1997.

[38] N. Firasta, M. Buxton, P. Jinbo, K. Nasri, and S. Kuo, "Intel avx: New frontiers in performance improvements and energy efficiency," *Intel white paper*, vol. 19, p. 20, 2008.

[39] P. Dudek and S. Carey, "General-purpose 128/spl times/128 simd processor array with integrated image sensor," *Electronics Letters*, vol. 42, no. 12, pp. 678–679, 2006.

[40] A. Peleg and U. Weiser, "Mmx technology extension to the intel architecture," *IEEE micro*, vol. 16, no. 4, pp. 42–50, 1996.

[41] R. J. Fisher, "General-purpose simd within a register: Parallel processing on consumer microprocessors," 2003.

[42] M. R. Thistle and B. J. Smith, "A processor architecture for horizon," in *Supercomputing'88.[Vol. 1]., Proceedings.* IEEE, 1988, pp. 35–41.

[43] S. Wong, T. Van As, and G. Brown, "ρ-vex: A reconfigurable and extensible softcore vliw processor," in *ICECE Technology, 2008. FPT 2008. International Conference on*. IEEE, 2008, pp. 369–372.

[44] P. Marcuello, A. González, and J. Tubella, "Speculative multithreaded processors," in *Proceedings of the 12th international conference on Supercomputing*. ACM, 1998, pp. 77–84.

[45] H. Du Nguyen, J. Yu, L. Xie, M. Taouil, S. Hamdioui, and D. Fey, "Memristive devices for computing: Beyond cmos and beyond von neumann," in *Very Large Scale Integration (VLSI-SoC), 2017 IFIP/IEEE International Conference on*. IEEE, 2017, pp. 1–10.

[46] A. Siemon, S. Menzel, R. Waser, and E. Linn, "A complementary resistive switch-based crossbar array adder," *IEEE journal on emerging and selected topics in circuits and systems*, vol. 5, no. 1, pp. 64–74, 2015.

[47] E. Linn, R. Rosezin, S. Tappertzhofen, R. Waser *et al.*, "Beyond von neumann–logic operations in passive crossbar arrays alongside memory operations," *Nanotechnology*, vol. 23, no. 30, p. 305205, 2012.

[48] H. A. Du Nguyen, L. Xie, M. Taouil, R. Nane, S. Hamdioui, and K. Bertels, "On the implementation of computation-in-memory

parallel adder," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 8, pp. 2206–2219, 2017.

[49] L. Xie, H. A. D. Nguyen, M. Taouil, and K. Bertels Said Hamdioui, "Fast boolean logic mapped on memristor crossbar," in *Computer Design (ICCD), 2015 33rd IEEE International Conference on*. IEEE, 2015, pp. 335–342.

[50] H. Du Nguyen, L. Xie, J. Yu, M. Taouil, and S. Hamdioui, "Interconnect networks for resistive computing architectures," in *Design & Technology of Integrated Systems In Nanoscale Era (DTIS), 2017 12th International Conference on*. IEEE, 2017, pp. 1–6.

[51] L. Xie, H. A. Du Nguyen, M. Taouil, S. Hamdioui, and K. Bertels, "Interconnect networks for memristor crossbar," in *Nanoscale Architectures (NANOARCH), 2015 IEEE/ACM International Symposium on*. IEEE, 2015, pp. 124–129.

[52] P. Yao, H. Wu, B. Gao, G. Zhang, and H. Qian, "The effect of variation on neuromorphic network based on 1t1r memristor array," in *Non-Volatile Memory Technology Symposium (NVMTS), 2015 15th*. IEEE, 2015, pp. 1–3.

[53] E. J. Merced-Grafals, N. Dávila, N. Ge, R. S. Williams, and J. P. Strachan, "Repeatable, accurate, and high speed multi-level programming of memristor 1t1r arrays for power efficient analog computing applications," *Nanotechnology*, vol. 27, no. 36, p. 365202, 2016.

[54] J. Borghetti, G. S. Snider, P. J. Kuekes, J. J. Yang, D. R. Stewart, and R. S. Williams, "Memristive switches enable stateful logic operations via material implication," *Nature*, vol. 464, no. 7290, pp. 873–876, 2010.

[55] L. Xie, H. A. Du Nguyen, M. Taouil, S. Hamdioui, and K. Bertels, "Boolean logic gate exploration for memristor crossbar," in *2016 International Conference on Design and Technology of Integrated Systems in Nanoscale Era (DTIS)*. IEEE, 2016, pp. 1–6.

[56] A. Haron, J. Yu, R. Nane, M. Taouil, S. Hamdioui, and K. Bertels, "Parallel matrix multiplication on memristor-based computation-in-memory architecture," in *High Performance Computing & Simulation (HPCS), 2016 International Conference on*. IEEE, 2016, pp. 759–766.

[57] R. B. Hur and S. Kvatinsky, "Memristive memory processing unit (mpu) controller for in-memory processing," in *Science of Electrical Engineering (ICSEE), IEEE International Conference on the*. IEEE, 2016, pp. 1–5.

[58] S. Kvatinsky, D. Belousov, S. Liman, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Magic–memristor-aided logic," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 61, no. 11, pp. 895–899, 2014.

[59] S. Kvatinsky, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Memristor-based material implication (imply) logic: design principles and methodologies," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 10, pp. 2054–2066, 2014.

[60] A. Haj-Ali, R. Ben-Hur, N. Wald, and S. Kvatinsky, "Efficient algorithms for in-memory fixed point multiplication using magic," in *Circuits and Systems (ISCAS), 2018 IEEE International Symposium on*. IEEE, 2018, pp. 1–5.

[61] M. Imani, S. Gupta, and T. Rosing, "Ultra-efficient processing in-memory for data intensive applications," in *Proceedings of the 54th Annual Design Automation Conference 2017*. ACM, 2017, p. 6.

[62] P.-E. Gaillardon, L. Amar, A. Siemon, E. Linn, R. Waser, A. Chattopadhyay, and G. De Micheli, "The programmable logic-in-memory (plim) computer," in *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2016, pp. 427–432.

[63] S. Shirinzadeh, M. Soeken, P.-E. Gaillardon, and R. Drechsler, "Fast logic synthesis for rram-based in-memory computing using majority-inverter graphs," in *Proceedings of the 2016 Conference on Design, Automation & Test in Europe*. EDA Consortium, 2016, pp. 948–953.

[64] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. Robshaw, Y. Seurin, and C. Vikkelsoe, "Present: An ultra-lightweight block cipher," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2007, pp. 450–466.

[65] I. Vourkas, D. Stathis, G. C. Sirakoulis, and S. Hamdioui, "Alternative architectures toward reliable memristive crossbar memories," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 1, pp. 206–217, 2016.

[66] L. Amarú, P.-E. Gaillardon, and G. De Micheli, "The epfl combinational benchmark suite," in *Proceedings of the 24th International Workshop on Logic & Synthesis (IWLS)*, no. EPFL-CONF-207551, 2015.

[67] S. Li, D. Niu, K. T. Malladi, H. Zheng, B. Brennan, and Y. Xie, "Drisa: A dram-based reconfigurable in-situ accelerator," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2017, pp. 288–301.

[68] Titan x pascal. [Online]. Available: https://www.nvidia.com/en-us/geforce/products/10series/titan-x-pascal/

[69] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[70] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[71] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[72] L. Yavits, S. Kvatinsky, A. Morad, and R. Ginosar, "Resistive associative processor," *CAL*, 2015.

[73] M.-F. Chang, C.-H. Chuang, M.-P. Chen, L.-F. Chen, H. Yamauchi, P.-F. Chiu, and S.-S. Sheu, "Endurance-aware circuit designs of nonvolatile logic and nonvolatile sram using resistive memory (memristor) device," in *Design Automation Conference (ASP-DAC), 2012 17th Asia and South Pacific*. IEEE, 2012, pp. 329–334.

[74] J. Wang, X. Dong, Y. Xie, and N. P. Jouppi, "Endurance-aware cache line management for non-volatile caches," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 11, no. 1, p. 4, 2014.

[75] E. S. Chung, P. A. Milder, J. C. Hoe, and K. Mai, "Single-chip heterogeneous computing: Does the future include custom logic, fpgas, and gpgpus?" in *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 2010, pp. 225–236.

[76] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "Prime: a novel processing-in-memory architecture for neural network computation in reram-based main memory," in *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3. IEEE Press, 2016, pp. 27–39.

[77] F. Leisch and E. Dimitriadou, "Machine learning benchmark problems," 2010.

[78] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 14–26, 2016.

[79] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[80] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "Deepface: Closing the gap to human-level performance in face verification," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 1701–1708.

[81] T. Iakymchuk, A. Rosado-Muñoz, J. F. Guerrero-Martínez, M. Bataller-Mompeán, and J. V. Francés-Víllora, "Simplified spiking neural network architecture and stdp learning algorithm applied to image classification," *EURASIP Journal on Image and Video Processing*, vol. 2015, no. 1, p. 4, 2015.

[82] V. Seshadri, D. Lee, T. Mullins, H. Hassan, A. Boroumand, J. Kim, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry, "Ambit: In-memory accelerator for bulk bitwise operations using commodity dram technology," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2017, pp. 273–287.

[83] V. Seshadri, Y. Kim, C. Fallin, D. Lee, R. Ausavarungnirun, G. Pekhimenko, Y. Luo, O. Mutlu, P. B. Gibbons, M. A. Kozuch *et al.*, "Rowclone: fast and energy-efficient in-dram bulk data copy and initialization," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2013, pp. 185–197.

[84] 6th generation intel core processor family datasheet. [Online]. Available: http://www.intel.com/content/www/us/en/processors/core/desktop-6th-gen-core-family-datasheet-vol-1.html

[85] Geforce gtx 745. [Online]. Available: http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-745-oem/specifications

[86] Hybrid memory cube specivcation 2.0. [Online]. Available: http://www.hybridmemorycube.org/files/SiteDownloads/HMC-30G-VSR_HMCC_Specification_Rev2.0_Public.pdf

[87] K. Stockinger and K. Wu, "Bitmap indices for data warehouses," in *Data Warehousing and Mining: Concepts, Methodologies, Tools, and Applications*. IGI Global, 2008, pp. 1590–1605.

[88] C.-Y. Chan and Y. E. Ioannidis, "Bitmap index design and evaluation," in *ACM SIGMOD Record*, vol. 27, no. 2. ACM, 1998, pp. 355–366.

[89] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sniper: exploring the level of abstraction for scalable and accurate parallel multi-core simulation," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2011, p. 52.

[90] S. Beamer, K. Asanović, and D. Patterson, "Direction-optimizing breadth-first search," *Scientific Programming*, vol. 21, no. 3-4, pp. 137–148, 2013.

[91] K. Wu, "Fastbit: an efficient indexing technology for accelerating data-intensive science," in *Journal of Physics: Conference Series*, vol. 16, no. 1. IOP Publishing, 2005, p. 556.

[92] S. Mittal, "A survey of reram-based architectures for processing-in-memory and neural networks," *Machine Learning and Knowledge Extraction*, vol. 1, no. 1, 2018. [Online]. Available: http://www.mdpi.com/2504-4990/1/1/5

[93] S. Jain, A. Ranjan, K. Roy, and A. Raghunathan, "Computing in memory with spin-transfer torque magnetic ram," *arXiv preprint arXiv:1703.02118*, 2017.

[94] A. Subramaniyan, J. Wang, E. R. M. Balasubramanian, D. Blaauw, D. Sylvester, and R. Das, "Cache automaton," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-50 '17. New York, NY, USA: ACM, 2017, pp. 259–272.

[95] J. Yu, H. A. Du Nguyen, M. Taouil, and S. Hamdioui, "Memristor devices for computation-in-memory," in *Proceedings of the 2018 Design, Automation & Test in Europe Conference & Exhibition*. EDA Consortium, 2018, pp. 1718–1725.

[96] J. Wadden, V. Dang, N. Brunelle, T. Tracy II, D. Guo, E. Sadredini, K. Wang, C. Bo, G. Robins, M. Stan *et al.*, "Anmlzoo: a benchmark suite for exploring bottlenecks in automata processing engines and architectures," in *Workload Characterization (IISWC), 2016 IEEE International Symposium on*. IEEE, 2016, pp. 1–12.

[97] M. Becchi, M. Franklin, and P. Crowley, "A workload for evaluating deep packet inspection architectures," in *Workload Characterization, 2008. IISWC 2008. IEEE International Symposium on*. IEEE, 2008, pp. 79–89.

[98] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: Characterization and architectural implications," in *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*. ACM, 2008, pp. 72–81.

[99] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*. Ieee, 2009, pp. 44–54.

[100] T. Finkbeiner, G. Hush, T. Larsen, P. Lea, J. Leidel, and T. Manning, "In-memory intelligence," *IEEE Micro*, vol. 37, no. 4, pp. 30–38, 2017.

[101] S. Aga, S. Jeloka, A. Subramaniyan, S. Narayanasamy, D. Blaauw, and R. Das, "Compute caches," in *High Performance Computer Architecture (HPCA), 2017 IEEE International Symposium on*. IEEE, 2017, pp. 481–492.

[102] S. Jeloka, N. B. Akesh, D. Sylvester, and D. Blaauw, "A 28 nm configurable memory (tcam/bcam/sram) using push-rule 6t bit cell enabling logic-in-memory," *IEEE Journal of Solid-State Circuits*, vol. 51, no. 4, pp. 1009–1021, 2016.

[103] C. Eckert, X. Wang, J. Wang, A. Subramaniyan, R. Iyer, D. Sylvester, D. Blaauw, and R. Das, "Neural cache: Bit-serial in-cache acceleration of deep neural networks," *arXiv preprint arXiv:1805.03718*, 2018.

[104] A. Sodani, R. Gramunt, J. Corbal, H.-S. Kim, K. Vinod, S. Chinthamani, S. Hutsell, R. Agarwal, and Y.-C. Liu, "Knights landing: Second-generation intel xeon phi product," *Ieee micro*, vol. 36, no. 2, pp. 34–46, 2016.

[105] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.

[106] V. Seshadri, K. Hsieh, A. Boroum, D. Lee, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry, "Fast bulk bitwise and and or in dram," *IEEE Computer Architecture Letters*, vol. 14, no. 2, pp. 127–131, 2015.

[107] J. Ahn, S. Hong, S. Yoo, O. Mutlu, and K. Choi, "A scalable processing-in-memory accelerator for parallel graph processing," *ACM SIGARCH Computer Architecture News*, vol. 43, no. 3, pp. 105–117, 2016.

[108] G. Benson, Y. Hernandez, and J. Loving, "A bit-parallel, general integer-scoring sequence alignment algorithm," in *Annual Symposium on Combinatorial Pattern Matching*. Springer, 2013, pp. 50–61.

[109] J. Han, C.-S. Park, D.-H. Ryu, and E.-S. Kim, "Optical image encryption based on xor operations," *Optical Engineering*, vol. 38, no. 1, pp. 47–55, 1999.

[110] D. Judd, K. Yelick, C. Kozyrakis, D. Martin, and D. Patterson, "Exploiting on-chip memory bandwidth in the viram compiler," in *Intelligent Memory Systems*. Springer, 2001, pp. 122–134.

[111] D. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Keeton, C. Kozyrakis, R. Thomas, and K. Yelick, "A case for intelligent ram," *IEEE micro*, vol. 17, no. 2, pp. 34–44, 1997.

[112] C. Kozyrakis, "Scalable vector media-processors for embedded systems," CALIFORNIA UNIV BERKELEY COMPUTER SCIENCE DIV, Tech. Rep., 2002.

[113] J. A. Poovey, T. M. Conte, M. Levy, and S. Gal-On, "A benchmark characterization of the eembc benchmark suite," *IEEE micro*, vol. 29, no. 5, 2009.

[114] M. Oskin, F. T. Chong, and T. Sherwood, *Active pages: A computation model for intelligent memory*. IEEE Computer Society, 1998, vol. 26, no. 3.

[115] D. Burger and T. M. Austin, "The simplescalar tool set, version 2.0," *ACM SIGARCH computer architecture news*, vol. 25, no. 3, pp. 13–25, 1997.

[116] K. Mai, T. Paaske, N. Jayasena, R. Ho, W. J. Dally, and M. Horowitz, "Smart memories: A modular reconfigurable architecture," *ACM SIGARCH Computer Architecture News*, vol. 28, no. 2, pp. 161–171, 2000.

[117] U. J. Kapasi, W. J. Dally, S. Rixner, J. D. Owens, and B. Khailany, "The imagine stream processor," in *null*. IEEE, 2002, p. 282.

[118] W. Wulf, E. Cohen, W. Corwin, A. Jones, R. Levin, C. Pierson, and F. Pollack, "Hydra: The kernel of a multiprocessor operating system," *Communications of the ACM*, vol. 17, no. 6, pp. 337–345, 1974.

[119] J. Draper, J. Chame, M. Hall, C. Steele, T. Barrett, J. LaCoss, J. Granacki, J. Shin, C. Chen, C. W. Kang *et al.*, "The architecture of the diva processing-in-memory chip," in *Proceedings of the 16th international conference on Supercomputing*. ACM, 2002, pp. 14–25.

[120] D. G. Elliott, W. M. Snelgrove, and M. Stumm, "Computational ram: A memory-simd hybrid and its application to dsp," in *Custom Integrated Circuits Conference, 1992., Proceedings of the IEEE 1992*. IEEE, 1992, pp. 30–6.

[121] J. Jeddeloh and B. Keeth, "Hybrid memory cube new dram architecture increases density and performance," in *VLSI Technology (VLSIT), 2012 Symposium on*. IEEE, 2012, pp. 87–88.

[122] J. T. Pawlowski, "Hybrid memory cube (hmc)," in *Hot Chips 23 Symposium (HCS), 2011 IEEE*. IEEE, 2011, pp. 1–24.

[123] R. Nair, S. F. Antao, C. Bertolli, P. Bose, J. R. Brunheroto, T. Chen, C.-Y. Cher, C. H. Costa, J. Doi, C. Evangelinos *et al.*, "Active memory cube: A processing-in-memory architecture for exascale systems," *IBM Journal of Research and Development*, vol. 59, no. 2/3, pp. 17–1, 2015.

[124] P. Bohrer, J. Peterson, M. Elnozahy, R. Rajamony, A. Gheith, R. Rockhold, C. Lefurgy, H. Shafi, T. Nakra, R. Simpson *et al.*, "Mambo: a full system simulator for the powerpc architecture," *ACM SIGMETRICS Performance Evaluation Review*, vol. 31, no. 4, pp. 8–12, 2004.

[125] J. J. Dongarra, P. Luszczek, and A. Petitet, "The linpack benchmark: past, present and future," *Concurrency and Computation: practice and experience*, vol. 15, no. 9, pp. 803–820, 2003.

[126] Daxpy subroutine, linear algebra package, lapack 3.5.0. [Online]. Available: http://www.netlib.org/lapack/explore-html/d9/dcd/daxpy_8f.html

[127] M. A. Alves, M. Diener, P. C. Santos, and L. Carro, "Large vector extensions inside the hmc," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2016*. IEEE, 2016, pp. 1249–1254.

[128] M. A. Z. Alves, C. Villavieja, M. Diener, F. B. Moreira, and P. O. A. Navaux, "Sinuca: A validated micro-architecture simulator." in *HPCC/CSS/ICESS*, 2015, pp. 605–610.

[129] H. Noyes *et al.*, "Microns automata processor architecture: Reconfigurable and massively parallel automata processing," in *Proc. of Fifth International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies*, 2014.

[130] K. Angstadt, J. Wadden, V. Dang, T. Xie, D. Kramp, W. Weimer, M. Stan, and K. Skadron, "Mncart: An open-source, multi-architecture automata-processing research and execution ecosystem," *IEEE Computer Architecture Letters*, vol. 17, no. 1, pp. 84–87, 2018.

[131] N. A. Almubarak, A. Alshammeri, and I. Ahmad, "Automata processor architecture and applications: A survey," *International Journal of Grid and Distributed Computing*, vol. 9, no. 4, pp. 53–66, 2016.

[132] I. Roy, "Algorithmic techniques for the micron automata processor," Ph.D. dissertation, Georgia Institute of Technology, 2015.

[133] A. V. Aho and M. J. Corasick, "Efficient string matching: an aid to bibliographic search," *Communications of the ACM*, vol. 18, no. 6, pp. 333–340, 1975.

[134] M. Becchi, C. Wiseman, and P. Crowley, "Evaluating regular expression matching engines on network and general purpose processors," in *Proceedings of the 5th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*. ACM, 2009, pp. 30–39.

[135] M. Góngora-Blandón and M. Vargas-Lombardo, "State of the art for string analysis and pattern search using cpu and gpu based programming," *Journal of information security*, vol. 3, no. 04, p. 314, 2012.

[136] Y. Zu, M. Yang, Z. Xu, L. Wang, X. Tian, K. Peng, and Q. Dong, "Gpu-based nfa implementation for memory efficient high speed regular expression matching," in *ACM SIGPLAN Notices*, vol. 47, no. 8. ACM, 2012, pp. 129–140.

[137] G. Vasiliadis, M. Polychronakis, and S. Ioannidis, "Parallelization and characterization of pattern matching using gpus," in *Workload Characterization (IISWC), 2011 IEEE International Symposium on*. IEEE, 2011, pp. 216–225.

[138] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun *et al.*, "Dadiannao: A machine-learning supercomputer," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 2014, pp. 609–622.

[139] W. J. Dally and B. P. Towles, *Principles and practices of interconnection networks*. Elsevier, 2004.

[140] A. B. Kahng, B. Li, L.-S. Peh, and K. Samadi, "Orion 2.0: A power-area simulator for interconnection networks," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 1, pp. 191–196, 2012.

[141] NVIDIA, "Tesla k20x gpu accelerator board specification," 2012.

[142] A. Farmahini-Farahani, J. H. Ahn, K. Morrow, and N. S. Kim, "Drama: An architecture for accelerated processing near memory," *IEEE Computer Architecture Letters*, vol. 14, no. 1, pp. 26–29, 2015.

[143] C. Weis, N. Wehn, L. Igor, and L. Benini, "Design space exploration for 3d-stacked drams," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011*. IEEE, 2011, pp. 1–6.

[144] G. H. Loh, "3d-stacked memory architectures for multi-core processors," in *ACM SIGARCH computer architecture news*, vol. 36, no. 3. IEEE Computer Society, 2008, pp. 453–464.

[145] Q. Zhu, B. Akin, H. E. Sumbul, F. Sadi, J. C. Hoe, L. Pileggi, and F. Franchetti, "A 3d-stacked logic-in-memory accelerator for application-specific data intensive computing," in *3D Systems Integration Conference (3DIC), 2013 IEEE International*. IEEE, 2013, pp. 1–7.

[146] S. K. Venkata, I. Ahn, D. Jeon, A. Gupta, C. Louie, S. Garcia, S. Belongie, and M. B. Taylor, "Sd-vbs: The san diego vision benchmark suite," in *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*. IEEE, 2009, pp. 55–64.

[147] J. A. Stratton, C. Rodrigues, I.-J. Sung, N. Obeid, L.-W. Chang, N. Anssari, G. D. Liu, and W.-m. W. Hwu, "Parboil: A revised benchmark suite for scientific and commercial throughput computing," *Center for Reliable and High-Performance Computing*, vol. 127, 2012.

[148] J. C. Lee, J. Kim, K. W. Kim, Y. J. Ku, D. S. Kim, C. Jeong, T. S. Yun, H. Kim, H. S. Cho, Y. O. Kim *et al.*, "18.3 a 1.2 v 64gb 8-channel 256gb/s hbm dram with peripheral-base-die architecture and small-swing technique on heavy load interface," in *Solid-State Circuits Conference (ISSCC), 2016 IEEE International*. IEEE, 2016, pp. 318–319.

[149] K. Sohn, W.-J. Yun, R. Oh, C.-S. Oh, S.-Y. Seo, M.-S. Park, D.-H. Shin, W.-C. Jung, S.-H. Shin, J.-M. Ryu *et al.*, "A 1.2 v 20 nm 307 gb/s hbm dram with at-speed wafer-level io test scheme and adaptive refresh considering temperature distribution," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 250–260, 2017.

[150] J. Macri, "Amd's next generation gpu and high bandwidth memory architecture: Fury," in *Hot Chips 27 Symposium (HCS), 2015 IEEE*. IEEE, 2015, pp. 1–26.

[151] H. Jun, J. Cho, K. Lee, H.-Y. Son, K. Kim, H. Jin, and K. Kim, "Hbm (high bandwidth memory) dram technology and architecture," in *Memory Workshop (IMW), 2017 IEEE International*. IEEE, 2017, pp. 1–4.

[152] Y. Wang, Y. Han, L. Zhang, H. Li, and X. Li, "Propram: exploiting the transparent logic resources in non-volatile memory for near data computing," in *Proceedings of the 52nd Annual Design Automation Conference*. ACM, 2015, p. 47.

[153] R. Ubal, B. Jang, P. Mistry, D. Schaa, and D. Kaeli, "Multi2sim: a simulation framework for cpu-gpu computing," in *Parallel Architectures and Compilation Techniques (PACT), 2012 21st International Conference on*. IEEE, 2012, pp. 335–344.

[154] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, "Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 7, pp. 994–1007, 2012.

[155] R. N. Horspool, "Practical fast searching in strings," *Software: Practice and Experience*, vol. 10, no. 6, pp. 501–506, 1980.

[156] P. Svärd, B. Hudzia, J. Tordsson, and E. Elmroth, "Evaluation of delta compression techniques for efficient live migration of large virtual machines," *ACM Sigplan Notices*, vol. 46, no. 7, pp. 111–120, 2011.

[157] Darpa intrusion detection data sets. [Online]. Available: http://www.ll.mit.edu/mission/

[158] Pizza and chili repository. [Online]. Available: http://pizzachili.dcc.uchile.cl/texts.html

[159] Opencv library. [Online]. Available: http://code.opencv.org

[160] A. Morad, L. Yavits, S. Kvatinsky, and R. Ginosar, "Resistive gp-simd processing-in-memory," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 12, no. 4, p. 57, 2016.

[161] A. Morad, L. Yavits, and R. Ginosar, "Gp-simd processing-in-memory," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 11, no. 4, p. 53, 2015.

[162] ——, "Efficient dense and sparse matrix multiplication on gp-simd," in *Power and Timing Modeling, Optimization and Simulation (PATMOS), 2014 24th International Workshop on*. IEEE, 2014, pp. 1–8.

[163] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *Computer Architecture (ISCA), 2011 38th Annual International Symposium on*. IEEE, 2011, pp. 365–376.

[164] A. Pedram, S. Richardson, M. Horowitz, S. Galal, and S. Kvatinsky, "Dark memory and accelerator-rich system optimization in the dark silicon era," *IEEE Design & Test*, vol. 34, no. 2, pp. 39–50, 2017.

[165] S. Palacharla, N. P. Jouppi, and J. E. Smith, *Complexity-effective superscalar processors*. ACM, 1997, vol. 25, no. 2.

[166] J. E. Smith and G. S. Sohi, "The microarchitecture of superscalar processors," *Proceedings of the IEEE*, vol. 83, no. 12, pp. 1609–1624, 1995.

[167] G. Conte, S. Tommesani, and F. Zanichelli, "The long and winding road to high-performance image processing with mmx/sse," in *Computer Architectures for Machine Perception, 2000. Proceedings. Fifth IEEE International Workshop on*. IEEE, 2000, pp. 302–310.

[168] A. Lodi, M. Toma, F. Campi, A. Cappelli, R. Canegallo, and R. Guerrieri, "A vliw processor with reconfigurable instruction set for embedded applications," *IEEE Journal of solid-state circuits*, vol. 38, no. 11, pp. 1876–1886, 2003.

Hoang Anh Du Nguyen (S'15) received the M.Sc. degrees in Computer Engineering from the Delft University of Technology, Delft, The Netherlands where she is currently a PhD candidate at the Computer Engineering Laboratory. Her current research interests include Computation-in-Memory architecture for big-data, memristor based systems and synthesis automation.

Jintao Yu (S'17) received the M.Sc degree in China in 2013. He is currently a pursuing the Ph.D. degree with the Computer Engineering Laboratory, Delft University of Technology. His current research interests include memristor-based computing systems and domain-specific languages.

Muath Abu Lebdeh received his BSc degree in electrical and electronic engineering, and MSc degree in electrical and computer engineering from Khalifa University in 2015 and 2017, respectively. Currently, he is a PhD student at Delft University of Technology. His research interests include emerging NVM and CIM circuits and architectures.

Mottaqiallah Taouil (S'10—M'15) received the M.Sc. and Ph.D. degrees (both with Hons.) in computer engineering from the Delft University of Technology, Delft, The Netherlands. He is currently an Assistant Professor in the Dependable Nano-Computing Group, Delft University of Technology. His current research interests include hardware security, test & reliability, and resistive computing.

Said Hamdioui (M'99—SM'11) received the M.S.E.E. and Ph.D degrees (Hons.) from the Delft University of Technology. He is currently Chair Professor on Dependable and Emerging Computer Technologies and Head of the Computer Engineering Laboratory of the Delft University of Technology (TUDelft), the Netherlands. Prior to joining TUDelft, he worked for Intel Corporation (Califorina, USA), Philips Semiconductors R&D (Crolles, France) and for Philips/ NXP Semiconductors (Nijmegen, The Netherlands). His current research focuses on two domains: Dependable CMOS nano-computing (including Reliability, Testability, Hardware Security) and emerging technologies and computing paradigms (including 3D stacked ICs, memristors for logic and storage, in-memory-computing). He holds two patents and has authored one book, contributed to two, and co-authored over 190 conference and journal papers. Prof. Hamdioui is a member of the Association for European NanoElectronics Activities / ENIAC Scientific Commitee Council, and Associate Editor of the IEEE Transactions on VLSI Systems, and he serves on the editorial board of IEEE Design and Test and the Journal of Electronic Testing: Theory and Applications. He has delivered various keynote speeches, distinguished lecturers, and invited presentations and tutorial at major international forum/conferences/schools and at leading semiconductor companies.

Francky Catthoor (F'05) received the engineering degree and a Ph.D. in electrical engineering from the Katholieke Universiteit Leuven, Belgium in 1982 and 1987 respectively. Between 1987 and 2000, he has headed several research domains in the area of high-level and system synthesis techniques and architectural methodologies. Since 2000 he is also strongly involved in other activities at IMEC including related application and deep submicron technology aspects, biomedical imaging and sensor nodes, and smart photo-voltaic modules, all at the Inter-university Micro-Electronics Center (IMEC), Heverlee, Belgium. Currently he is an IMEC fellow. He is part-time full professor at the EE department of the K.U.Leuven. In 1986 he received the Young Scientist Award from the Marconi International Fellowship Council. He has been associate editor for several IEEE and ACM journals, like Trans. on VLSI Signal Procsesing, Trans. on Multi-media, and ACM TODAES. He was the program chair of several conferences including ISSS'97 and SIPS'01. He has been elected an IEEE fellow in 2005.

APPENDIX A
COMPUTATION-OUT-MEMORY - NEAR (COM-N)

The COM-N class consists of architectures that perform computation using additional logic units outside the memory core but inside the memory SiP. These architectures were proposed in the past and evolved through different memory technologies ranging from conventional DRAM, embedded DRAM to emerging memory technologies such as RRAM. A large number of architectures have been proposed in this category. Table 3 shows a brief comparison among the architectures which will be explained in each subsection.

On one hand, these architectures have several common advantages:

- Reduced memory bottleneck compared to COM-F as the computations take place close to the memory core and therefore can benefit from the on-chip memory bandwidth.
- A wide variety of high performance functions can be implemented as the computing takes place with mature CMOS technology.
- These architectures do not suffer from endurance requirements, especially as traditional mature memories can be used.

On the other hand, they all share the following limitations:

- The amount of parallelism is either limited by the bandwidth or the available resources, which puts much more bandwidth restrictions than what can be achieved in the CIM-A/-P class. An area trade-off has to be made between registers and the type and amount of computing resources.
- There is an additional write overhead when the results have to be stored back in to the memory. Note that the outputs are produced outside the memory core, and therefore, extra write operations would be necessary in such cases.
- Relative complex memory controllers are needed as these architectures have to support COM-N instructions within the memory SiP and maintain the memory coherency with the rest of the memory hierarchy.
- Efforts are still required to modify instruction sets, compilers and tools to support in-memory instructions.

The following subsections discuss the details of each architecture.

A.1 VIRAM: Vector Intelligent RAM

VIRAM [11], [110], [111], [112] was proposed in 1997 by J. Gebis, et al., from University of California. VIRAM is an embedded DRAM based architecture that exploits data parallelism by performing computations using a vector processor near the embedded DRAM. The architecture consists of a host processor and its caches, a floating point unit (FPU), a DMA, and an embedded DRAM memory with a vector processor as a co-processor, as shown in Fig. 25. A specialized compiler generates two instruction sequences, one for the memory-intensive and one for the non memory-intensive part [110]. The memory-intensive part is executed on the vector processor with a fast on-chip bandwidth connection to the embedded DRAM. The non-memory intensive part is executed on the host processor with a slow off-chip bandwidth connection to the embedded DRAM. Note that for the non-memory intensive part the embedded DRAM plays the role of a conventional memory.

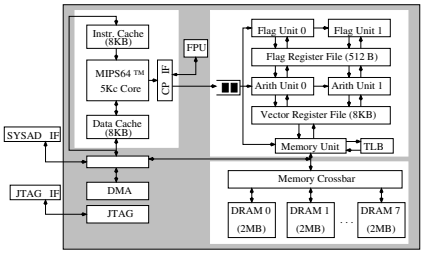


Fig. 25: Vector Intelligent RAM (VIRAM) [110]

The vector processor contains multiple arithmetic units, flag register files, a vector register file with 8 KB capacity and a memory control unit. All these components are used during the execution of multiple instructions. Synchronization is carried out using the flag register file that maintains the status of each arithmetic unit.

In addition to the general advantages of COM-N architectures, VIRAM comes with the following advantages:

- High parallelism due to vector processing on 8KB of data.
- The architecture uses embedded DRAM which is mature and has some advantages such as high performance, high bandwidth, low power [12], [14].

However, it also has the following limitations:

- The embedded DRAM has some limitations such as a high fabrication cost, low scalability, and trade-off between performance and capacity [12], [13], [14].

The architecture was demonstrated as a prototype and evaluated against the superscalar processor PowerPC MPC7455 and the VLIW processor VelociTI TMS320C6203 using EEMBC media benchmarks [113].

A.2 A-page: Active pages

A-page [114] was proposed in 1998 by M. Oskin, et al., from University of California. A-page is an DRAM based architecture that exploits data parallelism by performing computations using a reconfigurable processor near the DRAM sub-arrays. The architecture consists of a host processor and a DRAM memory with reconfigurable logic embedded in each subarray (so-called RADram), as shown in Fig. 26. To use A-page, computations of an application is partitioned to be executed on the host processor or memory. The host processor dispatches the instructions executed on memory to RADram. The RADram locks the data used by the instructions and performs the computations. Thereafter, the results is returned to the host processor. Multiple RADram subarrays and the host processor communicates through an inter-page scheme to ensure the synchronization.

	Hierarchy level	Computations		Memory Technology	Overheads			Sneak path current	Destructive read	Required read-out*	Copy scheme	Evaluation	
		Logic style	Processor type		Cells	Periphery	Controller					Simulator	App.
VIRAM	Main memory	Bool.	Vector	eDRAM	-	Conv.	Medium	No	No	No	Indirect	Prototype	EEMBC
A-page	Main memory	Bool.	Reconfigurable	RADRAM	-	Conv.	Medium	No	No	No	Indirect	SimpleScalar	(1)
FlexRAM	Main memory	Bool.	CPU	eDRAM	-	Conv.	Medium	No	No	No	Indirect	Prototype	(2)
S-Mem	Main memory	Bool.	Flexible	eDRAM	-	Conv.	Medium	No	No	No	Indirect	Imagine	(3)
DIVA	Main memory	Bool.	Vector	eDRAM	-	Conv.	Medium	No	No	No	Indirect	Prototype	(4)
AMC	Main memory	Bool.	Vector	HMC	-	Conv.	Complex	No	No	No	Indirect	Mambo	DGEMM, DAXPY
HIVE	Main memory	Bool.	Vector	HMC	-	Conv.	Complex	No	No	No	Indirect	SiNUCA	(5)
DaDianNao	Accelerator	NN.	NFU	eDRAM	-	Conv.	Simple	No	No	No	Indirect	Booksim2.0	CNN, DNN
DRAMA	Main memory	Bool.	CGRA	3D-DRAM	-	Conv.	Complex	No	No	No	Indirect	Gem5	San Diego Vision, Parboil
HBM	Main memory	Bool.	-	3D-DRAM	-	Conv.	Complex	No	No	No	Indirect	Product	No
ProPRAM	Main memory	Bool.	Logical,+, *	NVM	PCM, STT-MRAM	Modif.	Medium	No	No	No	Indirect	Multi2Sim, NVSIM	PUMA
ProPRAM	Main memory	Bool.	Logical,+, *	NVM	memristor	Modif.	Medium	No	No	No	Indirect	Analytical	MM.

+: n-bit addition
x: n-bit multiplication
NVM: Non-volatile memory
NN.: Neural network
Conv.: Conventional
Modif.: Modified
Bool.: Boolean
NFU: Neural Functional Unit
(*): Required read-out during computations
App.: Applications and benchmarks
MM.: Matrix multiplication benchmarks
(1): C++ STL array template, database query, image processing, largest common subsequence algorithm, MMX primitives, and sparse-matrix multiply
(2): data mining, protein pattern matching, TCP-D, MPEG-2 motion estimation
(3): 1024-point FFT, a 13-tap FIR filter, a 7x7 convolution, and an 8x8 DCT
(4): scientific computing, databases and image processing
(5): vector search, memory reset/set operations, vector sum,matrix stencil, matrix multiplication kernels
(6): vector search, memory reset/set operations, vector sum,matrix stencil, matrix multiplication kernels

TABLE 3: Comparison among Architectures of COM-N Classes

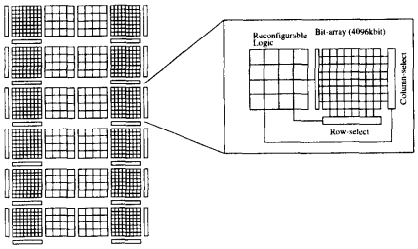


Fig. 26: Active pages (A-page) [114]

The RADram includes multiple subarrays. Each subarray has its own array, address decoders and 256 logic elements which is reconfigurable logic based on FPGAs.

In addition to the general advantages of COM-N architectures, VIRAM comes with the following advantages:

- Reconfigurability due to the reconfigurable logic.
- High parallelism due to the presence of logic in each subarray.

However, it also has the following limitations:

- The architecture uses DRAM technology which suffers from a low performance, high energy consumption, large footprint and is difficult to scale down.

The architecture concept was simulated using cycle-accurate simulator (SimpleScalar v2.0 [115]). The reconfigurable logic are implemented using FPGA. The performance of A-page was evaluated using multiple processor and memory-centric applications such as C++ STL array template, database query, image processing, largest common subsequence algorithm, MMX primitives, and sparse-matrix multiply [114].

A.3 FlexRAM: Advance Intelligent Memory System

FlexRAM was proposed in 1999 by Y. Kang, et al., from University of Illinois Urbana-Champaign [9]. FlexRAM is an embedded DRAM based architecture that exploits task level parallelism by performing computations on simplified conventional processors near the main memory. The architecture consists of a host processor (P.Host), L1/L2 caches, multiple FlexRAMs and plain DRAM, as shown in Fig.27. It works in two modes: as a conventional computer and as a near-memory computer. In the conventional mode, the FlexRAM is transparent and its memory can be accessed by P.Host as conventional DRAM. In the PIM mode, the P.Host can benefit from the FlexRAM by distributing instructions to multiple FlexRAMs. FlexRAMs are used as accelerators that execute multiple instructions on multiple data. After a FlexRAM finishes a task, it signals P.Host through a polling scheme.

A FlexRAM contains a P.Mem processor, its caches and 64 P.Array cores interleaved with 64 MByte DRAM memory. The P.Mem processor is a superscalar processor with floating-point support, with its own instruction and data cache. The P.Array core is a simple RISC processor equipped with 1 MByte DRAM memory. P.Array cores can also access the 1 MByte DRAM memory of its left and right neighbour. The instructions are stored on an instruction cache with four input ports which is shared among every 4 P.Array cores. A P.Mem communicates with P.Array cores through reserved registers. P.Mem can also broadcast a 32-bit word to all P.Arrays by setting a broadcast flag in all P.Arrays. In addition, it controls the communication between two P.arrays. After completing a task, a P.Array sends a notification to P.Mem through a P.Mem's Notify register.

In addition to the general advantages of COM-N architectures, FlexRAM comes with the following advantages:

- The architecture uses embedded DRAM which is mature and has some advantages such as high performance, high bandwidth, low power [12], [14].

However, it also has the following limitations:

- The parallelism is limited due to the complex hierarchy of logic units within the memory SiP.

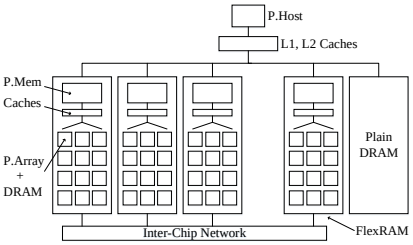


Fig. 27: Advance Intelligent Memory System (FlexRAM) [9]

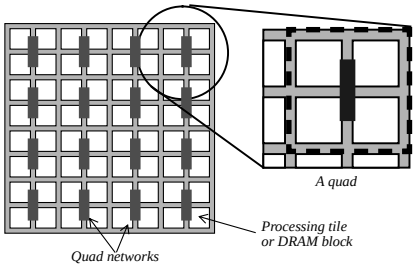


Fig. 28: Modular Reconfigurable Smart Memories (SM) [116]

- The data transfer within and among FlexRAM still requires improvements as it is complex to synchronized among multiple computation units, and a lot of traffic among computation units can limit its scalability and performance.
- The architecture uses embedded DRAM which has some limitations such as penalty trade-offs between performance and capacity, high fabrication cost, low scalability [12], [13], [14].

An architecture prototype was fabricated in a 505 mm^2 chip in 0.18um MLD technology running at a frequency of 400MHz. Its performance was measured by executing various algorithms (data mining, protein pattern matching, TCP-D, MPEG-2 motion estimation [9]) on two modes: the conventional and PIM-based computer.

A.4 S-Mem: Modular Reconfigurable Smart Memories

S-Mem [116] was proposed by K. Mai, et al., from Stanford University in 2000. SM is an embedded DRAM based architecture which has the capability of being reconfigured into various architectures; hence it can exploit various levels of parallelism such as instruction or data level parallelism based on the configuration. The architecture can be configured into different types of architectures ranging from application-specific to general purposed architectures. The architecture consists of quads connected with a mesh; each quad consists of 4 tiles. The interconnection inside and outside a quad consists of a high-speed interconnect network based on crossbars (as shown in Fig. 28). Depending on the target architecture, the tiles and their interconnect networks are configured accordingly.

Each tile in SM contains either a processor or memory. A processor tile consists of a processor engine, crossbar interconnect, quad interface and 16 8KB SRAMs (as shown in Fig. 29). A memory tile consist of a 2-4MB embedded DRAM, which has the same dimensions as a processor tile. Both the processor and memory tiles have some configuration flexibility. A processor tile can be configured for different architectures with a reconfigurable instruction format. A processor engine also contains multiple floating point and integer units. The SRAMs can be configured either as caches, scratpad memories or register files.

In addition to the general advantages of COM-N architectures, SM comes with the following advantages:

- The architecture is reconfigurable, which is flexible for application-specific application.

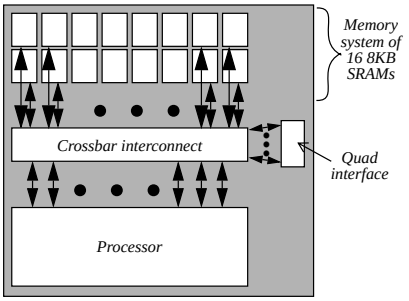


Fig. 29: Smart Memories Tile [116]

- The architecture uses embedded DRAM which is mature and has some advantages such as high performance, high bandwidth, low power [12], [14].

However, it also has the following limitations:

- The parallelism is limited due to the complex hierarchy of logic units within the memory SiP.
- Data transfer within and between process engines still requires further improvements, as it is complex to synchronize multiple computation units. In addition, a lot of traffic between them can limit its scalability and performance.
- The architecture uses embedded DRAM which has some limitations such as penalty trade-offs between performance and capacity, high fabrication cost, low scalability [12], [13], [14].

Two processors are mapped, implemented and evaluated using this architecture: the Imagine streaming processor [117] and Hydra multiprocessor [118]. They are simulated using Imagine simulator [117] and evaluated based on the set of applications including 1024-point FFT, a 13-tap FIR filter, a 7x7 convolution, and an 8x8 DCT [116].

A.5 DIVA: Data-intensive Architecture

DIVA was proposed in 2002 by J. Draper, et al., from USC Information Sciences Institute [10], [119] The architecture is based on a intelligent RAM [111], computational RAM [120] or embedded DRAM [14] and exploits data level parallelism by performing computations using vector processors. The

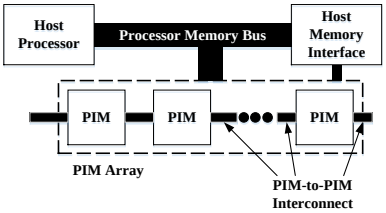


Fig. 30: Data-intensive Architecture (DIVA) [10]

architecture consists of a host processor, host memory interface and multiple in-memory computing blocks as co-processors (as shown in Fig. 30); the set of in-memory computing blocks (denoted as PIMs) can be accessed as a conventional memory or smart-memory co-processor. Through the host memory interface, the host processor is responsible for distributing workloads to PIMs, managing memory and switching context between different user programs; a PIM-to-PIM interconnect provides high bandwidth links among PIMs.

A PIM architecture contains a host interface, a PIM Routing Component (PiRC) and several PIM nodes. Through the host interface, memory accesses and computation packed parcels are transferred to PIMs. The PiRC routes parcels among PIM nodes. A PIM consists of processing logic units, several megabytes of memory, PBUF and a memory port. The processing unit in a PIM includes a scalar processor and a special unit called At-the-Sense-Amps Processor (ASAP). The scalar processor is a single-issue, in-order execution, 32-bit processor with a floating-point unit. ASAP, also referred to as Wide World Unit is used for 256-bit wide operations on data objects stored in a local memory row. PBUF in each PIM is served as local memory.

In addition to the general advantages of COM-N architectures, DIVA comes with the following advantages:

- The parallelism is high due to vector processing of multiple 256-bit operations concurrently.
- The architecture uses embedded DRAM which is mature and has some advantages such as high performance, high bandwidth, low power [12], [14].

However, it also has the following limitations:

- The architecture has a complex processor design which requires overhead in controlling, communication and programming.
- The architecture uses embedded DRAM which has some limitations such as penalty trade-offs between performance and capacity, high fabrication cost, low scalability [12], [13], [14].

A DIVA chip prototype was fabricated in TSMC 0.18um technology. Its performance was measured using a set of benchmark applications that includes scientific computing, databases and image processing.

A.6 HMC: Hybrid Memory Cube

HMC was introduced in 2012 by J. Jeddeloh, et al., from Micron [121], [122]. HMC is a 3D DRAM based architecture

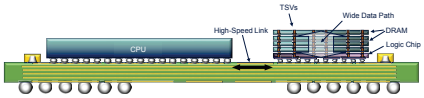


Fig. 31: Hybrid Memory Cube (HMC) [122]

that exploits data level parallelism by performing near memory computations inside the additional logic layer of the memory SiP. The architecture consists of a host CPU, a DRAM chip that contains a logic chip and multiple stacked DRAM dies based on TSVs, and a high-speed link between CPU and DRAM (as shown in Fig. 31). The logic layer controls the DRAM as its slaves, communicates with the host CPU and is able to perform near memory computations. There are two architectures that are based on HMC. They are explained next.

A.6.1 AMC: Active Memory Cube

AMC was introduced in 2015 by R. Nair, et al., from IMB [123]. AMC is an HMC based architecture that performs vector instructions in the logic die of the HMC. The architecture consists of a host processor, system network and multiple AMCs that contain a base logic layer and multiple DRAM layers (as shown in Fig. 32). The host processor communicates to AMCs using the system network that is capable of transferring data with a bandwidth of 256GB/s for read and write operations. This bandwidth is split in 8 lanes (i.e., 32 GB/s) where each lane is connected to an AMC. Each AMC has an 8GB capacity with an internal bandwidth of 320GB/s. The architecture comes with a compiler that analyzes applications, prepares sets of code that are distributed to the AMCs and also generates code for the interaction and communication between AMCs. The host processor handles the interaction code while each AMC executes the code containing useful computations.

Each AMC contains multiple AMC lanes, vault controllers and an interconnect network. Each AMC lane includes an instruction buffer, registers, control units, a load-store unit to perform conventional memory operations, and arithmetic units to perform either vector operations or long-instruction-word operations. The vault controller stores data that is loaded from DRAM layers and performs automic operations on the data before moving it forward to the host processor or AMC lanes. Furthermore, the vault controller is also responsible for maintaining the coherence between AMC lanes.

In addition to the general advantages of COM-N architectures, AMC comes with the following advantages:

- The parallelism is high due to multiple and concurrent processing lanes.
- The architecture uses HMC which is mature and already commercialized and in addition has some advantages such as a high performance, high bandwidth, low power, and high density [121], [122].

However, it also has the following limitation:

- The architecture has a complex processor design which has a control, communication and programming overhead.

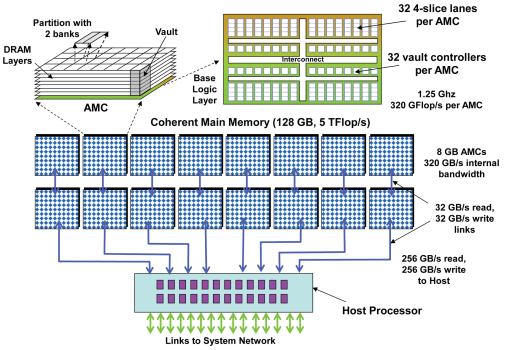


Fig. 32: Active Memory Cube (AMC) [123]

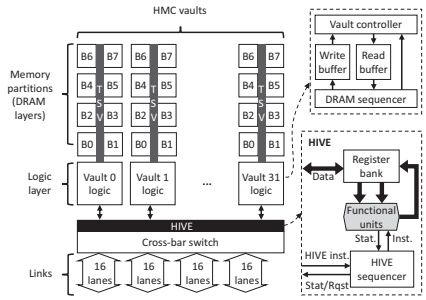


Fig. 33: HMC Instruction Large Vector Extensions (HIVE) [127]

The architecture was simulated using Mambo simulator [124] and evaluated using two computation kernels for supercomputers: DGEMM [125] and DAXPY [126].

A.6.2 HIVE: HMC Instruction Large Vector Extensions

HIVE was proposed in 2016 by M. Alves, et al., from Federal University of Rio Grande do Sul [127]. HIVE is a Hybrid Memory Cube (HMC) [121], [122] based architecture that performs large vector operations inside the logic die of a HMC. The architecture consists of a host processor and a HMC module that is extended with a HIVE, as shown in Fig. 33. The host processor, not shown in the figure, is a pipelined-like architecture with six stages; it fetches, decodes, renames, dispatches, executes and commits a sequence of instruction. If an instruction fragment has to be executed using in-memory instructions, the processor diverts the instruction fragment to the HMC module. HMC module executes the fragment and returns the result back to the processor.

HMC module consists of multiple DRAM layers, logic vaults, HIVE controller, a crossbar switch and multiple-lane links to host processor (as shown in the left side of Fig. 33). The data is stored in multiple DRAM layers and retrieved by the HIVE. The HIVE controller contains a register bank, functional units and a HIVE sequencer (as shown in the bottom right of Fig. 33). The logic vaults contains a vault controller, write and read buffer, and a DRAM sequencer (as shown in the top right of Fig. 33). Once the HIVE sequencer receives an instruction, it locks the involved memory address space; if the memory has already been locked, the requested instruction returns a fail status to processor; otherwise, a memory synchronization occurs by flushing related cache data into DRAM. The logic vaults and HIVE subsequently execute the instructions by reading data to read buffers and register bank, performing operations using functional units, and (optional) storing into memory using write buffers. The operations in HIVE are based on vector operations that operate on 8KB of data at a time executed by the 32 logic vaults and HIVE functional units. As the amount of data is large, a DRAM sequencer and HIVE sequencer schedule these operation accordingly. The results can be collected in register banks and sent back to the host processor through the crossbar switch and links.

In addition to the general advantages of COM-N architectures, HIVE comes with the following advantages:

- The parallelism is high due to vector processing on 8KB of data.
- The architecture uses HMC which is mature, commercialized and has some advantages such as high performance, high bandwidth, low power, high density [121], [122].

However, it also has the following limitation:

- The architecture has a complex HMC module which has a control, communication and programming overhead.

The architecture is simulated using SiNUCA [128] and evaluated using some integer (vector search and memory reset/set operations) and floating-point (vector sum, matrix stencil, and matrix multiplication) kernels against three baseline platforms; both HIVE and baseline platforms are based on the Intel Atom processor. Like HIVE, the three baseline platforms have also additional processing capacities; for the baseline platforms they are as follows: 1) HMC instructions using HMC 2.0 memory [86] (HMC+HMC), 2) 128-bit SSE instructions with DDR-3 1333 modules (SSE+DDR) and 3) 128-bit SSE instructions with HMC 2.0 (SSE+HMC).

A.7 D-AP: Micron Automata Processor

D-AP was proposed in 2013 by P. Dlugosch, et al., from Micron Technology [129]. D-AP is an automata processor that exploits instruction level parallelism based on the same concept as mentioned in Section 4.8. D-AP consists of multiple STEs and a routing matrix (as shown in Fig. 34). The STEs are implemented using DRAM technology. The routing matrix contains multiple programmable switches, buffers, routing lines and cross-point connections.

Each DRAM column corresponds to an STE which stores for each state the input symbols it accepts. One input symbol is fed each cycle to all the DRAM columns simultaneously and as a result possible next states based on the current input symbol are returned. Simultaneously, the routing matrix is used to calculate the possible next states from the current active states. By AND-ing the resuls of both

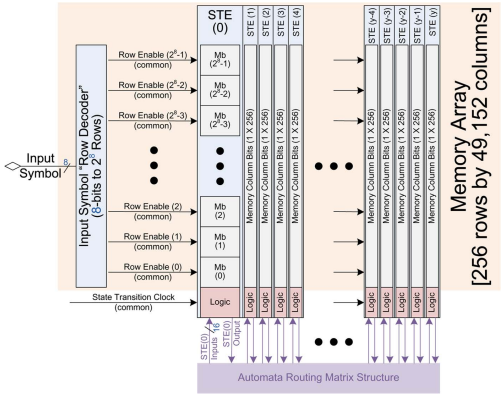


Fig. 34: DRAM Automata Processor (D-AP) [129]

operations, the actual next states can be determined. The program terminates when all input symbols are processed.

In addition to the general advantages of COM-N architectures, D-AP has the following advantages:

- The architecture uses DRAM technology, which has several benefits such as a high maturity, high endurance, no sneak path currents, and may benefit from existing optimizing techniques and tools.
- The automata processing techniques are quite mature and several tools exist already [130], [131], [132], hence it is feasible to explore many applications using automata processing.

However, it also has the following limitations:

- The architecture uses DRAM technology which suffers from a low performance, high energy consumption, low scalability and large footprint.
- The architecture requires additional compiling techniques and tools to perform conventional Boolean logic functions as they will have to be mapped to automata's.

The architecture is fabricated and evaluated and compared against hardware implementations of automata processing [133], [134], [135], [136], [137].

A.8 DaDianNao: A Machine-Learning Supercomputer

DaDianNao was proposed in 2014 by Y. Chen, et al., from Chinese Academy of Sciences [138]. DaDianNao is eDRAM based architecture that exploits instruction level parallelism by performing neural computations using a Neural Functional Unit (NFU). The architecture consists of multiple nodes; each node has its own interconnection to other nodes, multiple tiles and eDRAM router that connects these tiles (as shown in the left part of Fig. 35). DaDianNao is an improved version of DianNao; it exploits the interconnect network to reduce data movements.

A DaDianNao tile contains four eDRAM banks and an NFU which can perform multiplications, additions, and transfer functions of linear interpolation depending on the

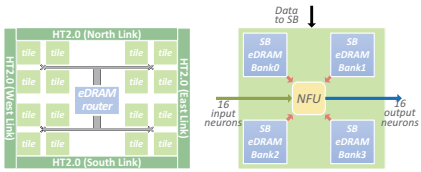


Fig. 35: A Machine-learning Supercomputer (DaDianNao) [138]

type of the neural network. The eDRAM banks store input neurons which are broadcasted through a fat tree interconnect network and output neurons which are computed using NFU. After computation, the values of the output neurons are collected by the center eDRAM bank of DaDianNao.

In addition to the general advantages of COM-N architectures, DaDianNao has the following advantages:

- Computations for neural networks are quite mature and they do not require a high precision (therefore, they are resilient against device variation), and can benefit from existing neural network techniques and tools.
- The architecture uses embedded DRAM which is mature and has some advantages such as a high performance, high bandwidth, low power [12], [14].

However, it also has the following limitation:

- The architecture uses embedded DRAM which has some limitations such as penalty trade-offs between performance and capacity, high fabrication cost, low scalability [12], [13], [14].
- The architecture requires additional compiling techniques and tools to perform conventional Boolean logic functions as they have to be mapped to neural networks.

The architecture is simulated using Booksim2.0 [139], [140] and evaluated by implementing state-of-the-art machine-learning algorithms (CNNs and DNNs) and is compared against GPU NVIDIA K20 [141].

A.9 DRAMA: DRAM-Accelerator

DRAMA was proposed in 2014 by A. Farmahini-Farahania, et al., from University of Wisconsin-Madison [142]. DRAMA is based on 3D-stacked DRAM technology that exploits task level parallelism by computating with coarse-grain reconfigurable accelerators (CGRAs). The architecture includes a host processor, an accelerator-enabled DRAM rank (AEDR), and a DRAM DIMM interface (as shown in Fig. 36). The host processor sends the configuration data, address generation parameters for the data to be processed, and other kernel parameters to AEDR before triggering the kernel execution on AEDR. The host processor communicates with the AEDR through DRAM DIMM.

An AEDR includes CGRAs stacked on top of DRAM devices. The CGRAs operate on data stored in the DRAM, communicate with the DRAM through TSVs and execute kernels. A CGRA contains a grid of 32-bit functional

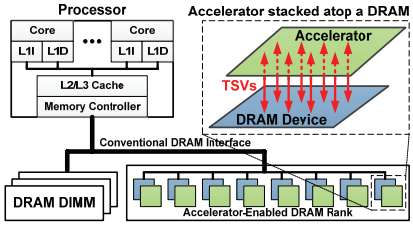


Fig. 36: An Architecture for Accelerated Processing Near Memory (DRAMA) [142]

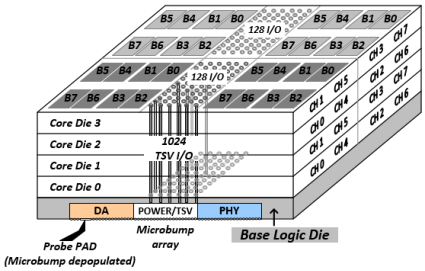


Fig. 37: High Bandwidth Memory (HBM) [151]

units (FUs), small distributed storages, configurable routing switches and a memory controller. The FUs perform computational operations. The distributed storages are used to store immediate results. The configurable routing switches set the connections up between FUs and storages. They are configured at run-time by the configuration from the host processor. The memory controller is used to issue memory requests to a DRAM devices.

In addition to the general advantages of COM-N architectures, DRAMA comes with the following advantages:

- The parallelism is high due to multiple parallel cores.
- The architecture uses 3D-stacked DRAM which has some advantages such as a high performance, high bandwidth, and high scalability [143], [144].

However, it also has the following limitations:

- The architecture has a complex accelerator design which has control, communication and programming overhead.
- The architecture uses 3D-stacked DRAM which has a high fabrication cost and high power consumption [145].

The architecture is simulated with gem5 simulator using San Diego Vision suite and Parboil benchmarks suites [146], [147] to verify its functional timing and execution. The power is estimated by McPAT model. Components in the architecture such as 3D stacking DRAM with TSVs and CGRAs are synthesized with 32 and 40nm to estimate their area.

A.10 HBM: High-Bandwidth Memory

HBM was proposed in 2015 by H. Jun, et al., from SK Hynix [148], [149], [150], [151]. HBM is a heterogeneous 3D stacked DRAM platform that exploits data level parallelism by performing near-memory computations using additional logic circuits in the memory SiP. The architecture consists of one logic die and multiple DRAM dies which are connected through TSVs and microbumps (as shown in Fig. 37). The logic die performs computations on the data that is stored in the DRAM dies and tries to avoid data movements to the host processors.

In addition to the general advantages of COM-N architectures, HBM comes with the following advantages:

- The parallelism is high due to multiple parallel cores.
- The architecture uses 3D-stacked DRAM which has some advantages such as a high performance, high bandwidth, high scalability, and low power consumption [143], [144], [150].

However, it also has the following limitations:

- The architecture has a complex 3D stacked DRAM module which has a control, communication and programming overhead.
- The architecture uses 3D-stacked DRAM which has a high fabrication cost and high power consumption [145].

The first generation of HBM increases the memory bandwidth dramatically up to 100GB/s per stack and reduces the power consumption up to 60% [150]. Later generations of HBM such as HBM2 and HBM3 realize even bandwidth improvements up to 400GB/s using new error correction techniques [151].

A.11 ProPRAM: A Near Data Computing Architecture based on Non-Volatile Memory

ProPRAM was proposed in 2015 by Y. Wang, et al., from Chinese Academy of Sciences [152]. ProPRAM is a non-volatile memory (i.e., PCM) based architecture that exploits data level parallelism by performing computations using computational resources in the peripheral circuitry. The architecture consists of a host processor with a modified instruction set and a non-volatile memory module with its peripheral circuits transparent to the host processor. The host processor sends instructions to the non-volatile memory module in a similar manner as normal read/write instruction but with a modified instruction set. The non-volatile memory receives these instructions and reuses the computational resources in the memory module to perform the computations on the data stored in the non-volatile memory.

The non-volatile memory module contains a matrix of memory cells, a conventional peripheral circuit consisting of sense amplifiers, word drivers, row buffers, col decoders, and a specific peripheral circuit consisting of components related to non-volatile memory instructions such as Data-comparison Write (DCW), DCW Inversion (DCWI) and Flip-n-Write (as shown in Fig. 38). The non-volatile memory related peripheral circuit is capable of performing comparisons, additions, and logic operations. Therefore, an appropriate addressable scheme can be used to direct operands

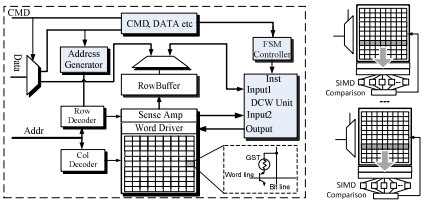


Fig. 38: A Near Data Computing Architecture based on Non-Volatile Memory (ProPRAM) [152]

(data from non-volatile memory) to these computational units. For example, the DCW unit in Fig. 38 is able to compute different operations using the first input from the row buffer and the second input from the sense amplifiers; the output is stored back into the memory.

In addition to the general advantages of COM-N architectures, ProPRAM comes with the following advantages:

- The parallelism is high due to multiple parallel processing units.
- The architecture exploits existing computational resources which reduces area and power consumption.
- The architecture uses non-volatile memory, hence consumes low energy and has a small footprint.

However, it also has the following limitations:

- The parallelism is low due to limited available computing resources in the memory peripheral circuitry.
- The architecture has programming and compiling overhead to make the computation units transparent to the host processor.

The architecture is simulated using Multi2Sim [153] and NVSIM [154]; it is evaluated using PUMA benchmark suite [155], [156], [157], [158], [159] and has been compared against a multicore computing system.

A.12 ReGP: Resistive GP-SIMD

ReGP was proposed in 2016 by A. Morad, et al., from Technion-Israel Institute of Technology [160]. ReGP is a RRAM memory based architecture that exploits data parallelism by attaching a SIMD-like processing unit to the resistive memory, as shown in Figure 39. The architecture consists of a sequential or conventional processor, its L1 and LLC cache, shared memory array and SIMD processor. The sequential processor executes traditional code and controls the SIMD processor in a master-slave mode. The SIMD processor executes parallel instructions on the data stored in the shared memory array.

The SIMD processor contains multiple processing units (PUs), a sequencer and a Network on Chip (NoC) with reduction tree. Each PU contains registers, a single bit full-adder and a function generator to perform arithmetic and logical operations. The sequencer receives instructions from the sequential processor and assigns them to PUs. The PUs load data from the shared memory array and perform the requested operations. If required, the NoC and reduction trees are used to perform more complex functions.

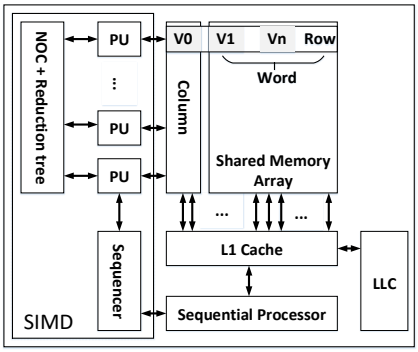


Fig. 39: Resistive GP-SIMD (ReGP) [160]

In addition to the general advantages of COM-N architectures, ReGP comes with the following advantages:

- The parallelism is high due to multiple parallel processing units.
- The architecture uses non-volatile memory, hence consumes low amount of energy and small footprint.
- The architecture can reuse compilers, programming languages and tools from SIMD architectures.

However, it also has the following limitation:

- The operations within the processing units are simple; complex functions such as floating point operations can cause a high overhead.

The architecture is evaluated analytically against CMOS GP-SIMD [161] using the dense matrix multiplication benchmark [162].

APPENDIX B
COMPUTATION-OUT-MEMORY - FAR (COM-F)

COM-F class includes all conventional architectures which perform computations based on a processor-centric model. The architectures in this class consists of one or multiple computational cores, possibly with caches (including L1, L2 and L3) or other small storage units (e.g., register files) and a separate main memory (normally DRAM). These architectures typically have a lot of data movements from the main memory to the computational cores and vice versa. In addition, a complex coherence scheme is required to keep the coherency among different levels of caches and main memory, especially when each processor comes with its own cache. These architectures have several common advantages:

- High potential parallelism due to the possibility of performing multiple concurrent operations. Architectures in this category could exploit instruction, data and task level parallelism and hybrid combinations of them. However, this potential is only reachable when the number of data accesses is low compared to the number of computations, i.e., when the memory bandwidth is high enough.

- A wide variety of high performance functions can be implemented as the computing takes place with mature CMOS technology.
- Applications, compiling, programming languages and tools are very mature and a lot of commercial products exist.
- These architectures do not suffer from endurance requirements, especially as traditional mature memories can be used.

On the other hand, they all share the following limitations:

- High memory bottleneck as data has to travel through a long memory hierarchy and the results are produced far from the memory core. Moreover, its hard to hide the main memory latency.
- The maximum amount of parallelism is either limited by the bandwidth or the available resources. An area trade-off has to be made between registers, caches and computing resources.
- Additional write overhead when the results have to be stored back in to the memory. Note that the outputs are produced outside the memory core, and therefore, extra write operations would be necessary in such cases.
- Relatively complex memory controllers are needed to maintain data coherency, especially for multicores.
- High power consumption of SRAM based caches and cores leads to dark silicon. This limits the amount of concurrent operations that can take place [163], [164].

In this class, multiple architectures have been proposed. In the most basic form (i.e., a single or scalar processor), such architectures typically include one or more arithmetic logic units (ALUs, floating point units, etc.), register files and controlling units (program counters, muxes, etc.) [3]. Later on, these scalar processes have been improved by including multiple pipeline stages referred to as pipelined processor [3], [42]. The datapath of a pipelined processor is split into multiple stages; each stage executes a part of the instruction such as instruction fetch, decode, execution, write-back etc. [3]. Thereafter, multiple methods were introduced to improve the parallelism further. First, each stage of a pipelined processor is duplicated to form a superscalar processor. Such processors could execute multiple instructions simultaneously [6], [165], [166]. Second, ALUs can be duplicated to deal with multiple data streams, the so called vector processor [38], [40], [167]. Third, the ALUs can be duplicated and integrated with other functional units to perform multiple different instructions concurrently on a large data stream; such a processors is called Very Long Instruction Word (VLIW) processor [43], [168]. Finally, multiple single-core processors are integrated together in multicore processors; they can perform multiple independent tasks simultaneously [7].

3

ARCHITECTURE LEVEL

This chapter presents two architectures: Computation-in-Memory (CIM) and Computation-in-Memory Accelerator (CIMX). The CIM architecture represents a CIM-A class which produces computation results inside the memory array. First, it motivates the proposed architecture to address the challenges of computer architectures in big data era. Thereafter, it shows how to map a parallel adder into the memory array, and demonstrates the feasibility of the above parallel adder in terms of memory array mapping, controller and interconnect network using two different memristive logic designs. The CIMX architecture represents a CIM-P class which produces computation results using the peripheral circuits. First, it discusses the proposed architecture, where the CIM core is used as an accelerator as it relatively needs a lower endurance and has a reasonable operation diversity. Subsequently, it proposes an analytical model and a simulation framework to estimates the performance of both CIMX architecture and a conventional architecture (i.e., multicore architecture) for several applications.

The content of this chapter is based on the following research articles:

1. S. Hamdioui, L. Xie, **H.A. Du Nguyen**, M. Taouil, K.L.M. Bertels, "Memristor based computation-in-memory architecture for data-intensive applications." Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition (DATE), Grenoble, France, 2015.
2. **H. A. Du Nguyen**, L. Xie, M. Taouil, R. Nane, S. Hamdioui and K. Bertels, "Computation-in-memory based parallel adder," Proceedings of the 2015 IEEEACM International Symposium on Nanoscale Architectures (NANOARCH), Boston, USA, 2015.
3. **H. A. Du Nguyen**; L. Xie; M. Taouil; R. Nane; S. Hamdioui; K. Bertels, "On the Implementation of Computation-in-Memory Parallel Adder," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems , vol.25, no.8.
4. J. Yu, **H. A. Du Nguyen**, L. Xie, M. Taouil, S. Hamdioui, "Memristive devices for computation-in-memory," Design, Automation & Test in Europe Conference & Exhibition, Dresden, Germany, 2018.
5. **H. A. Du Nguyen**, J. Yu, M. Abu Lebdeh, I. Ashraf, M. Taouil, S. Hamdioui, "A Computation-In-Memory Accelerator based on Resistive Devices," The international symposium on Memory Systems (MEMSYS), Washington DC, USA, 2019.

3.1. PROBLEM STATEMENT

Based on the classification proposed in Chapter 2, two architectures belongs to CIM-A and CIM-P class were investigated. Due to the novelty of these architectures, there are still many questions to be investigated. For CIM-A architectures which employ memristor crossbar to perform computations, it is essential to investigate the following aspects: (1) the mapping of a complex function on memristor crossbar, (2) the performance improvements of this computing scheme, and (3) the overhead of controller, interconnect network, etc. For CIM-P architectures which employ peripheral circuits to perform computations, it is essential to investigate the following aspects: (1) the appropriate architecture and memory level (of the memory hierarchy) to implement this computing scheme, (2) the methods to evaluate and simulate this computing scheme, (3) the performance improvements of this computing scheme. This chapter focuses on these issues.

CIM-A architectures: First, a CIM-A architecture performing computation within the memory array is proposed. In addition, the potentials of this architecture for big data applications is investigated using two study cases of health care and mathematics applications. Thereafter, a more detailed CIM architecture is studied with a mapping of a complex function (i.e., parallel addition) on the memristor crossbar. Finally, it is essential to investigate the performance improvements and overheads of this computing scheme using the parallel addition as a study case.

CIM-P architectures: First, one or more computing core are proposed to employ the CIM-P computing method of performing computations using the peripheral circuits. In addition, an interesting question is whether the CIM-P cores should be used as a memory hierarchy (i.e., caches, main memory) or accelerator. Finally, it is essential to build an analytical model and simulation platform to explore the potentials of the proposed architectures based on the CIM-P computing cores.

3.2. MAIN CONTRIBUTIONS

The main contributions in the above aspects are as follows.

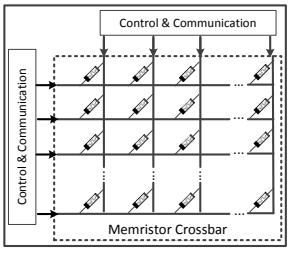
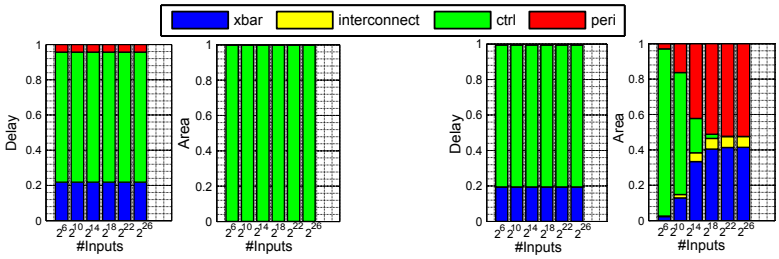


Figure 3.1: CIM Architecture

CIM-A architectures [20, 72, 73]: We first introduce memristor background and its potential applications; thereafter we propose the Computation-in-Memory architecture as a CIM-A architecture (as shown in Fig. 3.1). The Computation-in-Memory architecture is proposed as an integration of computation and storage in the same physical location;

this architecture offers an opportunities to solve big data applications efficiently. The architecture is realized using non-volatile memristive devices, instead of CMOS devices. Therefore, it has the potentials to reduce both memory wall and energy consumption with orders of magnitude.



(a) CRS Implementation (b) FBLC Implementation

Figure 3.2: Delay and Area Analysis of Memristor Implementation's Components

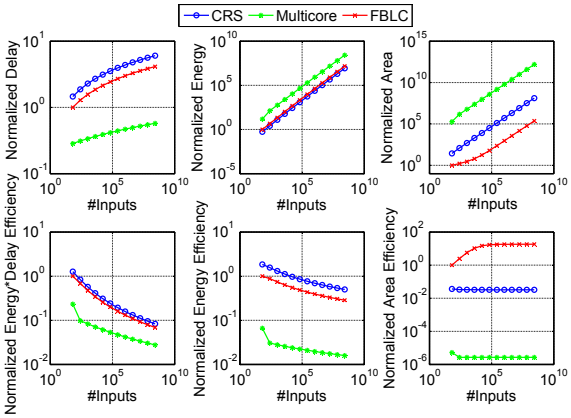


Figure 3.3: Performance Results of CRS, FBLC and Multicore Implementation

We further investigate the details of CIM architecture by mapping a parallel addition on the memristor crossbar. Thereafter, we study the performance and overheads of the parallel addition mapping. Two implementations, one based on Boolean logic (FBLC implementation) and the other on implication logic (CRS implementation), are investigated in terms of their impact on the total cost. This cost includes the the delay, energy and area of the crossbar (xbar), interconnect network (interconnect), peripheral circuit (peri) and CMOS controller (ctrl). The results show that the implementation of both designs is feasible. In fact, the area of the CRS implementation is dominated by the controller; hence, it is less scalable as shown in Fig. 3.2(a). On the other hand, the FBLC implementation is more scalable as the crossbar and peripheral circuits dominate the overall cost, as shown in Fig. 3.2(b). In comparison with conventional architectures, the memristor implementations outperform a similar multicore implementation by nearly

two orders of magnitude in terms of Energy Delay Efficiency, Energy Efficiency and Area Efficiency, as shown in Fig. 3.3.

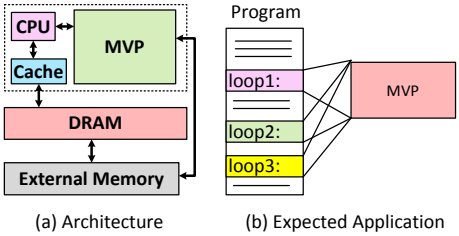


Figure 3.4: Memristive Vector Processor architecture

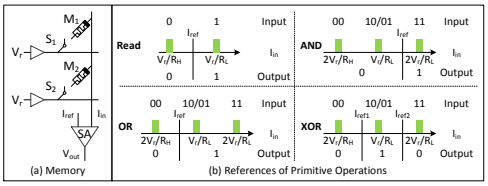


Figure 3.5: Scouting logic [75]

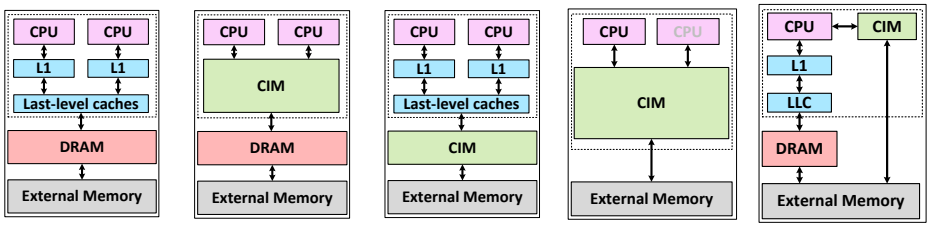


Figure 3.6: Conventional (a) and potential resistive architectures: CIMC (b), CIMM (c), CIMU (d), CIMX (e)

CIM-P architectures [76, 88]: We first propose two CIM-P cores which are Automata Processor and Memristive Vector Processor (MVP). In this content, I mainly contribute to the MVP as shown in Fig. 3.4(a). The MVP is based on the scouting logic that is capable of performing logical operations while reading two memory rows simultaneously (as shown in Fig. 3.5) [75]. The MVP is beneficial in a program illustrated in Fig. 2.1(b). The program consists of multiple loops processing a dataset that is preloaded and mapped on MVP. Each time a loop is called, the processor sends a (macro)-instruction to MVP; the instruction is locally decoded and executed. The result is returned to the processor. This feature occurs in multiple applications such as database management [89], DNA sequencing [90–92], and graph processing [93].

We propose four architectures that are feasible to employ this MVP which is also called as Computation-in-Memory (CIM) core. The four architectures use CIM core either in the memory hierarchy (i.e., caches, main memory), as an accelerator, or as a universal memory (as shown in Fig. 3.6b, 3.6c, 3.6d, 3.6e, respectively). The architecture with CIM core as accelerator (so-called CIMX) is selected due to its requirements of relatively a lower endurance and reasonable operation diversity. Thereafter, we also build an analytical model and simulation platform to investigate the potentials of this accelerator. The architecture shows that CIMX architecture outperforms conventional architectures with at least one order of magnitude improvements in terms of delay and energy while consuming less chip area (as shown in Fig. 3.7). In addition, three benchmarks were selected to execute on the simulation platforms. CIMX achieves nearly one order of magnitude improvement in delay and energy with a smaller footprint for the three simulated benchmarks. These improvements are mainly due to reduction in data movement and use of non-volatile technology.

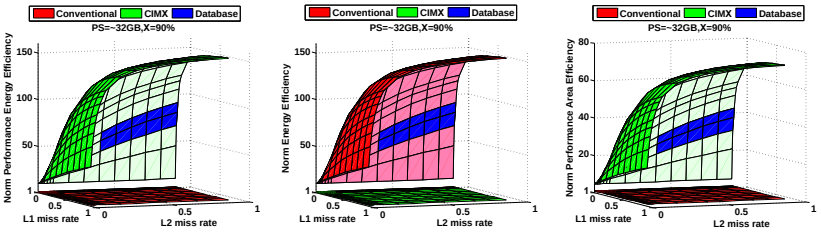


Figure 3.7: Analytical results of the performance energy, energy, and performance area efficiency metrics

Memristor Based Computation-in-Memory Architecture for Data-Intensive Applications

[1]Said Hamdioui [1]Lei Xie [1]Hoang Anh Du Nguyen [1]Mottaqiallah Taouil [1]Koen Bertels [2]Henk Corporaal [2]Hailong Jiao

[1]Computer Engineering
Delft University of Technology
Delft, the Netherlands
First_Letter_First_Name.LastName@tudelft.nl

[2]Electronic Systems group
Eindhoven University of Technology
Eindhoven, The Netherlands
{h.corporaal, H.Jiao}@tue.nl

3

[3]Francky Catthoor [3]Dirk Wouters

[3]IMEC, Kapeldreef 75, B-3001
Leuven, Belgium
<catthoor@imec.be>

[4]Linn Eike

[4]RWTH Aachen University
Aachen, Germany
linn@IWE.RWTH-Aachen.de

[5]Jan van Lunteren

[5]IBM Research Laboratory
Zurich, Switzerland
jvl@zurich.ibm.com

***Abstract*—One of the most critical challenges for today's and future data-intensive and big-data problems is *data storage and analysis.* This paper first highlights some challenges of the new born Big Data paradigm and shows that the increase of the data size has already surpassed the capabilities of today's computation architectures suffering from the limited bandwidth, programmability overhead, energy inefficiency, and limited scalability. Thereafter, the paper introduces a new *memristor-based architecture* for data-intensive applications. The potential of such an architecture in solving data-intensive problems is illustrated by showing its capability to increase the computation efficiency, solving the communication bottleneck, reducing the leakage currents, etc. Finally, the paper discusses why memristor technology is very suitable for the realization of such an architecture; using memristors to implement dual functions (storage and logic) is illustrated.**

I. INTRODUCTION

Today's applications are becoming extremely data intensive; healthcare, social media, large scientific/engineering experiments, and security are just couple of examples. As the speed of information growth exceeds Moore's Law, since the beginning of this new century, excessive data is posing major challenges [1] and a new scientific paradigm is born: data-intensive scientific discovery, also known as *Big Data problems*. The primary goal is to analyse and increase the understanding of both data and processes in order to extract the highly useful information hidden in the huge volume of data, which in turn can be used to increase e.g., the productivity. Storing and analysing such data is posing major challenges as the data volume already surpassed the capability of today's computers which suffer from e.g., communication and memory-access bottlenecks due to limited bandwidth [2-lahiri, 3-somavat]. For instance, a transfer of 1 petabyes data at a rate of 1000MB/second will take 12.5 days! Memory size and memory access do not only kill the performance, but also severely impact energy/power consumption [2, 3 ,4]. In addition, CMOS technology used to implement today's architectures contributes to such consumption due to high leakage currents; not to mention other challenges the technology is facing such as limited scalability, reduced reliability [30-34], etc. In conclusion, today's CMOS based architectures are not able to provide the computation capability needed for data-intensive applications. *New* architectures based on *new* technologies are urgently required.

This paper discusses a new architecture, *Computation-In-Memory* (CIM Architecture), for specific data-intensive applications; it is based on the *integration* of *storage* and *computation* in the *same* physical location (crossbar topology) and the use of *non-volatile resistive-switching technology (memristive devices or memristors in short)* [30, 38, 39, 94] instead of CMOS technology.

The rest of the paper is organized as follows. Section II highlights the Big Data problem and shows how the conventional computers based on CMOS technology are incapable to deal with such problems; and motivates the need for a new architecture. Section III discusses CIM architecture, including its concept and its potential; the section puts that in perspective by taking couple of application examples and comparing the performance of CIM architecture with the state-of-the-art. Section III shows why memristor is the key enabler for CIM architecture by illustrating how the device, in crossbar architecture, can perform a dual function (storage and computation). Section IV concludes the paper.

II. DATA-INTENSIVE APPLICATIONS VS CMOS COMPUTERS

A. Big Data and Data Intensive Applications

No one can deny the fact that a large number of fields and sectors, ranging from economics and business activities to public administration, from national security to many scientific research areas, involve data-intensive applications, hence, dealing with Big Data problems. Big Data is extremely valuable to generate productivity in businesses and evolutionary breakthroughs in scientific disciplines, which give us a lot of opportunities to make great progress in many fields [1]. The primary goal is to increase the understanding of processes in order to extract so much potential and highly useful values hidden in the huge volumes of data, and therefore, it comes with many challenges, such as data capture, *data storage, data analysis,* and data visualization.

978-3-9815370-4-8/DATE15/©2015 EDAA

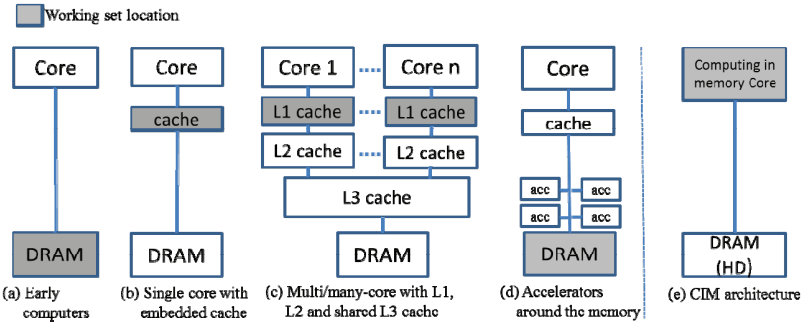


Figure 1: Classification of computing systems based on working set location

Performing data analysis within economically affordable time and energy is the *pillar* to solve big data problems

B. *Today's Computers*

The computing systems, developed since the introduction of stored program computers by John von Neumann in the forties [5], can be classified based on the location of the so-called "*working set*" (loosely defined as the collection of information referenced by a program during its execution) into four classes (a) to (d) as shown in Figure 1. In the early computers (typically before the 80s), the working set was contained in main memory. Due to the gap between the core (CPU) speed and the memory, caches were introduced to reduce the gap and increase the overall performance, where the caches have become the location of the working set. Today's computing systems for data-intensive applications are still based on Von Neumann (VN) architectures and still rely on many parallel (mini-)cores with a shared SRAM cache (parallel CPUs, GPUs, SIMD-VLIWs, vector processors); see Figure 1(c). Clusters of cores can be replicated many times, each having their own L1 cache, but it is far from realistic to assume a distributed reasonable sized L1 cache in every mini-core; too much area and leakage power overhead is incurred in that case. Such solutions suffer from major limitations such as a decreased performance acceleration per core [6], increased power consumption [7, 8], and limited system scalability [6, 9]. These are mainly caused by the processor-memory bottleneck [10, 11]. As current data-intensive applications require huge data transfers back and forth between processors and memories through load/store instructions [12], the maximal performance cannot be extracted as the processors will have many idle moments while waiting for data [10-14]. Computation, which is the main activity of a system, by far consumes less energy and chip area, and has lower execution time compared to communication and memory access (e.g., L1 cache), especially for data intensive applications [15]. The energy consumption of the cache accesses and communication makes up easily 70% to 90% [2,3,4]; not to mention the rest of the memory hierarchy. In addition, programmability in conventional processors also comes at a substantial energy cost: for example, [4] reports that executing a *multiply* instruction on a simple in-order core in 45nm technology consumes about 70 pJ, whereas the actual operation itself consumes less than 4 pJ. The overhead is due to instruction fetching and decoding and other control.

Triggered by these issues, the design of high-performance computing systems is starting to move away from a conventional computation-centric model towards a more data-centric approach. The latter concept intends to improve performance and power efficiency through reduction of data movement by performing the actual processing closer to where the data resides in the memory system. Several alternative architectures are proposed that fall into this category. One alternative is called "*Processor-in-memory*" as shown in Figure 1(d); additional processing units (accelerators) are put around one or more memories which are the working set location; examples are FlexRAM [16], DIVA [17], TeraSys [18], EXECUBE [19], HTMT [20], Computational RAM [21], DSP-RAM [22], Smart memories-based architecture [23], Gilgamesh [24], Continuum computer architecture [24], and MICRON's architecture for automata processing [26]. The second alternative architecture is called **"***Memory-in-processor*", which is an extension of what is shown in Figure 1(c), where extra addressable memories are put close to the cores; examples are Data Arithmetic SRAM [27] and Connection machine [28]. The third is called "*In memory computing/database*" (mainly for database management), which primarily relies on the storage of the complete database working set in the main memory of dedicated servers rather than relying on complicated relational databases operating on comparatively slow disk drives [29].

C. *CMOS Technology*

Today's computers are manufactured using the traditional CMOS technology, which is reaching the inherent physical limits due to down-scaling. Technology nodes far below 20nm are presumably only practical for limited applications due to multiple challenges [30-34,], such as high static power consumption, reduced performance gain, reduced reliability, complex manufacturing process leading to low yield and complex testing process, and extremely costly masks.

Many novel nano-devices and materials are under investigation to replace the CMOS technology in next IC generations. Among the emerging devices, such as graphene transistor [35], nanotube [36], tunnel field-effect transistor (TFET) [37], etc., memristor [38, 39] is a promising candidate.

Its advantages are CMOS process compatibility [40], lower cost, zero standby power [41], nanosecond switching speed [42], great scalability and high density [43], and non-volatile nature [44, 45]. It offers a high OFF/ON resistance ratio [46] and it is promising to have a good endurance and retention time [47]. More importantly, the memristor is a *two-terminal resistive-switching device* that can be used to *build both storage and information processing units* [48, 49, 50].

D. *The Need of New Architecture*

The speed at which data is growing has already surpassed the capabilities of today's computation architectures suffering from communication bottleneck (due to limited bandwidth), energy inefficiency (due to CMOS technology), and programmability overhead. Therefore, there is a need for a new architecture using new device technology, being able to (a) eliminate the communication bottleneck and support massive parallelism to increase the overall performance, (b) reduce the energy inefficiency to improve the computation efficiency. This can be done by taking the data-centric computing concept much further by integrating the processing units and the memory in the same physical location and therefore moving the working set into the core as shown in Figure 1 (e).

III. CIM ARCHITECTURE- BEYOND VON NEUMANN

This section presents first the concept of the CIM architecture as an alternative of today's architectures. Thereafter the potential of the architecture will be illustrated by selecting couple of data-intensive applications and making an analysis of different performance metrics and comparing the results with the state-of-the art. Finally, major open questions related to the implementation of the CIM architectures will be highlighted.

A. *CIM Architecture Concept*

To tackle the big data computation problems and solve the todays computers bottlenecks, we propose a memristor-based architecture paradigm where both the computation and the storage take place at the same physical location (the crossbar array). The approach intends to provide solutions based computing-in-memory architectures using non-volatile devices. Figure 2 shows the traditional versus the proposed CIM architecture; note that in CIM architecture the storage and computation are integrated together in a very dense crossbar array where memristors are injected at each junction of the crossbar (top electrode and bottom electrode). The communication and control from/to the crossbar can be realized using CMOS technology. CIM architecture addresses important challenges and has huge potentials which go substantially beyond the current state-of-the-art.

- *Tightly integrated computation-in-memory crossbar architecture supporting massive parallelism*: as the storage and computation are integrated together, the communication bottleneck is significantly reduced. In addition, because the memristor technology is highly scalable (~5nm [30]), huge crossbar architectures allowing massive parallelism are feasible.
- *An architecture with practically zero leakage:* Today's architectures heavily rely on SRAMs as caches. These are required to have a very fast R/W access, leading to increasingly high leakage with technology scaling. Hence, the memristor crossbar architecture solves also the leakage bottleneck, at least in the memory.
- *Significant performance improvement at lower energy and area:* Given the nature of the architecture (supporting massive parallelism), the non-volatile technology in the crossbar, and the small feature size of the memristor, the architecture has the potential of improving the overall performance at extremely low power consumption and smaller area. The next section illustrates this potential for two different applications.

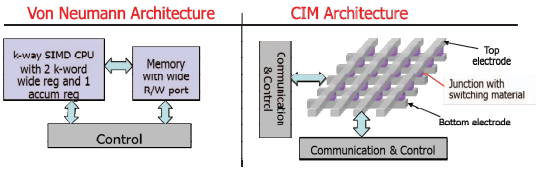


Figure 2: Traditional versus proposed architecture

B. *CIM Architecture Potential*

To illustrate how CIM architecture significantly advances the state-of-the-art, the performance of CIM and the conventional architectures for two applications will be estimated.

1) Healthcare: using genomics in diagnosing/treating diseases: the continuously dropping price of DNA sequencing has shifted the challenge from acquiring genetic information to the actual processing and analysis of this information [51]. Despite its computational simplicity, the huge amount of genetic data (hundreds of GBs per experiment) that needs to be processed makes the analysis rather time consuming and even not practical due to communication/memory access bottleneck. A practical solution used today for comparing two DNA sequences is based on the creation of a sorted index of the reference DNA that can be used to identify the location of matches and mismatches in another sequence rapidly. This approach, however, results in eliminating available data locality in the reference and causing huge number of cache misses with high memory access penalty and high energy cost. To quantify this effect, we assume we have 200 GB of DNA data to be compared to a healthy reference of 3GB for 50% coverages [51] and evaluate the DNA sorted-index sequencing algorithm on both the conventional architecture and CIM architecture; see Figure 2. The conventional architecture is assumed to scalable multi-core architecture, consisting of a number of clusters, each with 32 cores. Table 1 presents further made assumptions for both architectures. Three metrics are used for the evaluation: (a) the energy-delay product per operations, (b) the computation efficiency defined as the number (#) of operations per required energy, and (c) performance (#operations) per area;

Table 1: Assumptions made for conventional and CIM architectures

Assumption for conventional architecture	Assumptions for CIM architecture

Generic assumptions
• FinFET 22nm multi-core implementation
o Gate delay = 14 ps [53, 54]
o Area per gate: 0.248 μm^2 [30]
o Power consumption per gate: 175 nW [54]
o Leakage power: (a) Leakage power consumption per gate: 42,83 nW [30], (b) Leakage duration: cycle time – delay per gate
o Operating frequency: 1 GHz
• The architecture consist of a certain number of clusters of processing units, each cluster shares an 8kB L1 cache.
For Healthcare example
• Typically, the DNA reference sequence must be covered 50 times by short reads. The length of the short reads are assumed to be 100 characters.
• 200GB of DNA data is compared to a healthy reference of 3GB.
• Number of short reads no_short_reads = coverage * 3 * giga/ short_read_len; coverage=50, short_read_len=100
• Number of comparisons no_comparsions = 4*no_short_reads, for each A, C, G, T nucleotides
• Number of clusters is 18750, each contains 32 comparators.
o Limited with the state-of-the-art chip area
• Each cluster shares 8 kB Cache (per cluster)
o Area: 0.0092 mm^2 [57]
o Hit ratio = 50%; Hit cycle time = 1 cycle
o Miss penalty = 165 cycle [55];
o Write cycle time = 1 cycle;
o Static power: 1/64 Watt [56]
For Mathematics example
• Fully scalable *reusing* clusters; each has 8 kB shared cache.
• Additions are performed by 32 adders per cluster:
o Adder architecture: Carry Look Ahead (CLA)
o Number of gates per adder: 208 [52]
o Number of gate delay: 18;
o Adder latency: 252ps = 18*14ps
o Energy per 32-bit adder
• Shared 8 kB Cache (per cluster): the same as for healthcare except with 98% hit rate,
	Generic assumptions
	• Memristor 5nm crossbar implementation [30]
	o Memristor write time: 200 ps [60]
	o Area per memristor: $1x10^{-4}$ um^2 [30]
	o Dynamic energy per write operation: 1 fJ [30]
	• The memory capacity of the CIM architectures is assumed to be equal to the sum of all caches for the CMOS based computer.
	For Healthcare example
	• Each comparison is performed by a comparator
	o Comparator: 2 XOR and a NAND implemented by implication logic [58]
	o Number of memristors per comparator: 13 (XOR: 5, NAND: 3)
	o Area per comparator: $1.3 * 10^{-3}$ um^2 [58]
	o Number of steps per comparator: 16 steps (Two XOR works in parallel, an XOR takes 13 steps, and an NAND takes 3 steps, step takes a memristor write time). [58]
	o Comparator latency: 3.2 ns
	o Dynamic energy per comparator: 45fJ [58]
	o Static energy per comparator: 0 fJ [30]
	• The crossbar size equals to total cache size of CMOS computer
	o Size= 18750*8kB = 1.536*10^8 memristors
	o Date hit rate = 50%, Hit cycle time = 1 cycle
	o Miss penalty = 165 cycle
	For Mathematics example
	• The crossbar is scalable to support the 10^6 adders
	• Additions are performed by memristors
	o Adder architecture: TC-adder [59]
	o Number of memristors per adder: 34 (N+2, N=32) [59]
	o Area per adder: $3.4x10^{-3}$ um^2
	o Number of steps per 32-bit addition: 133 (4N+5, N=32, each step takes a memristor write time). [59]
	o Adder latency: 16600 ps (133 * 200 ps)
	o Dynamic energy per 32-bit adder: 246 fJ (8 (operations per bit) *32 (bits) * 1 fJ [59])
	o Static energy per 32-bit adder: 0 fJ [30]
	• The memory hit rate is assumed to be 98%, remaining parameters are the same as for the healthcare example.

2) Mathematics: Here we assume 10^6 parallel addition operations and make similar assumptions for the two architectures as those done in the previous example; see Table 1 for the details.

Table 2 shows the obtained results for the two applications for conventional (Conv.) and CIM architectures; both applications clearly show that the improvements are orders of magnitude. Reducing/eliminating memory accesses, the non-volatile technology, and the high parallelism are enabling these improvements.

Table 2: Huge potential of CIM architecture

Metric	Archit.	DNA Sequencing	10^6 additions
Energy-delay/ operations	Conv.	2.0210e-06	1.5043e-18
	CIM	2.3382e-09	9.2570e-21
Computing efficiency	Conv.	4.1097e+04	6.5226e+09
	CIM	3.7037e+07	3.9063e+12
Performance area	Conv.	5.7312e+09	5.1118e+09
	CIM	5.1118e+09	4.9164e+12

C. CIM Architecture Challenges

Although we mentioned that CIM architecture targets data-intensive applications, especially applications that require massive parallelism and huge data working sets to be continuously kept in the memory, the proposed concepts can be adapted to any computation-in-memory (CIM) architecture for high computation efficiency. This architecture paradigm shift, based on memristor technology, changes the traditional system design, compiler tools, manufacturing processes, etc., to facilitate its "industrialization". In fact, it is well recognized that memristor technologies are very promising. Although understanding of its capabilities and limitations is still evolving, the technology is expected to rule the computer world, from material science (understanding and proving the properties of the materials and assuring reliability), to design methods, tools, operating systems and its potential applications. Examples of its use are replacement of RAM, flash and even disk drives, complex self-learning neural networks, advanced artificial neural brains, and many more [61]. It may play a significant role in advancing Exascale

computing, ‘computer on a chip’ capabilities, as well as driving developments in neural and analogue computing. Next section will elaborate more on memristor technology.

IV. MEMRISTOR - THE KEY ENABLER FOR CIM ARCHITECTURE IMPLEMENTATION

This section reviews first the memristor technology. Thereafter its suitability for the realization of both storage and logic functions is discussed and illustrated.

3

A. Memristor Technology

Memristors or memristive devices, also referred to as resistive memory devices, are very broad groups of memory technologies; they can be classified based on their dominant physical operating mechanism into three classes [30]: *Phase Change Memories, Electrostatic/ Electronic Effects Memories, and Redox memories.* The redox-based resistive switching devices (ReRAMs) are attracting most attention due to their excellent scaling, endurance, and retention properties [30, 95]; their physical mechanism for switching is based on reduction/oxidation (Redox)-related chemical effects. The category of “Redox RAM” encompasses a wide variety of *Metal-Insulator-Metal (MIM)* structures; the electrochemical mechanisms driving the resistance state (from high to low or vice versa) can operate in the bulk I-layer, along conducting filaments in the I-layer, and/or at the I- layer/metal contact interfaces in the MIM structure. The ReRAMs consist of three types, two bipolar and one unipolar [30,50,61]; the rest of the section will focus on the two bipolar devices and show the best available properties for both device types; these are the Valence Change Memory (VCM) and the Electrochemical metallization (ECM) devices.

For both VCM (HfO_x) and ECM (Ag-chalcogenide) devices a feature size of $F = 10$ nm was reported [62, 63]. A minimum switching time of < 200 ps was shown for TaO_x-based VCM devices [42], whereas for ECM devices (Ag-MSQ) switching times below 10 ns were realized [64]. In terms of endurance, more than 10^{12} cycles are feasible for TaO_x-based VCM cells and more than 10^{10} for Ag-GeSe ECM cells [65]. Extrapolated retention of > 10 years was, for example, shown in [66] (TaO_x-based VCM cells) and [67] (Ag-chalcogenide).

In ECM devices a conductive metallic filament (Cu or Ag) is established during switching, thus, the filament length can be considered the state variable [68]. For a memristive ECM model, both electronic and ionic currents must be considered, and the strong non-linearity of the switching kinetics must be reflected by the model. VCM modelling is even more challenging due to the versatile device physics [69]. Since simple memristor models fail to predict the correct device behaviour [39, 70], more complex empirical and physics-based models were developed recently [71, 72].

B. Memristor for Crossbar Memories

The primary driver for ReRAM research is the semiconductor industry seeking for novel energy-efficient non-volatile and highly scalable memory elements [30]. A straightforward implementation of the ReRAM array is realised using a passive crossbar architecture, resulting in the highest density [73, 74]. However, this architecture suffers from undesired paths for current called *sneak paths* [75]; due to the low resistive current paths, the maximum array is limited to small arrays [76]. To overcome this issue, three classes of solutions are proposed:

- *Selector devices*, which are separate devices in connection with the RRAM cell such as a diode or a transistor (1S1R) [77, 78].
- *Switching device modification*, where the resistive devices is modified; E.g., serially connecting of two anti-serial memristive devices (bipolar switches) resulting into a “complementary resistive switcher” (CRS) being able to block the current at low voltage irrespective of the state of the device [78], or the deployment of a high nonlinear memristive device (due to current-controlled negative differential resistance) to overcome sneak path [79].
- *Bias schemes*, where the voltage bias applied to non-accessed wordlines and bitlines are set to values different from those applied to accessed wordline and bitlines in order to minimize the sneak path current; examples are multistage reading [80] and use of AC signal instead of DC for sensing the data stored in the desired cell [81].

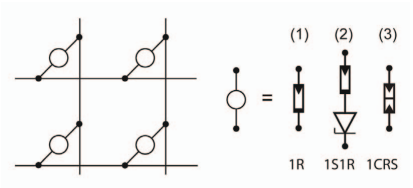


Figure 3: Illustration of complementary resistive switch

Figure 3 sketches the concept of the the crossbar array and some junction options to deal with sneak paths, while Figure 4 illustrates the *I-V* characteristic of a CRS cell which consists of two memristive ECM devices A and B. The states '0' and '1' are the logical storage states and the state 'LRS/LRS’ occurs only when reading the memory state. The internal memory states '0 'and '1' of a CRS cell are indistinguishable at low voltages because state ‘0’ as well as state ‘1’ show a high resistance. Therefore, no parasitic current sneak paths can arise. To read the stored information of a single CRS cell, a read voltage must be applied to the cell. If the CRS cell is in state '0', then it switches to state ‘ON’; if the cell is in state '1' then it remains in its state. In case conventional crossbar (with resistive current paths), reading ON state is a destructive operation, therefore, it is necessary to write back the previous state of the cell after reading it. In general, the writing of state ‘0’ requires a negative voltage ($V < V_{th,4}$) and for writing '1' a positive voltage $V > V_{th,2}$ is required.

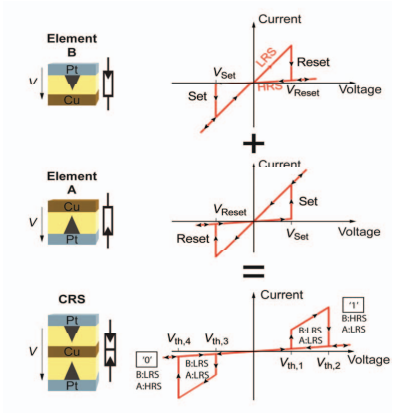


Figure 4: Left: Zoom in a passive nano-crossbar array. Right: possible cross point junctions

C. *Memristor for Logic Functions*

Memristive devices are well suited for the implementation of logic functions including: (a) programmable interconnects [82], (b) look-up tables (LUTs) [83] or content addressable memories (CAMs) [84], and (c) sequential 'stateful' logic operations [49, 58, 85].

Programmable logic arrays based on resistive switching junctions were suggested first in [82] and later also applied to FPGAs [86]. Typically, the CMOS overhead is relatively large since the array size is small. A next step was the CMOL FPGA concept [87], where a sea of elementary CMOS cells is connected to a small crossbar part-array. In this approach the elementary CMOS cells are connected via resistive switches (1S1R) enabling wired-or functionality. In general, reconfigurable on-chip wiring enables new options for memristive chip design and can also be combined with the functionalities as those described next .

Resistive memories can be either used to implement small LUTs for FPGAs (as suggested in [83]) or LUTs can be mapped to large-scale crossbar arrays [88, 89] to reduce the crossbar array overhead. Moreover, CAMs based on memristors are feasible with different flavors [90, 91]; e.g., a CRS-based CAM is recently demonstrated [84].

Memristors are also used to design (sequential) logic operations based on Boolean functions [92] or (material) implication logic (IMP) [49, 58, 85]; the latter seems to be more popular. Figure 5 uses two examples to illustrate the concept of IMP. Figure 5(a) gives a basic logic function using two memristors. Together with a load resistor R_G, the operation p IMP q is conducted as follows [49]:

1. Set device p to p ($V_P = \pm V_{Write}$)
2. Set device Q to q ($V_Q = \pm V_{Write}$)
3. $q' = p$ IMP q ($V_P = V_{COND}$ and $V_Q = V_{write}$)
4. Read q'

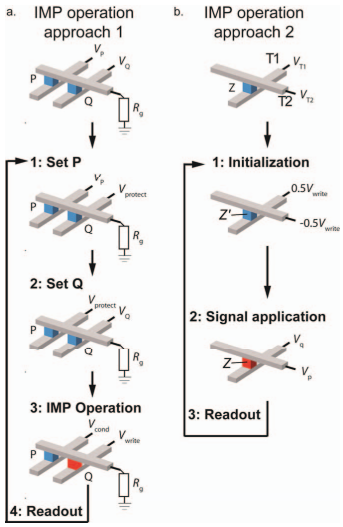


Figure 5: Two ways to implement IMP. Blue cube represents state '0' and the red cube state '1'

An alternative approach to implement p IMP q, with superior performance, is suggested in [93], as shown in Figure 5(b). The input signals Vp = ±½V_{Write} and Vq = ±½V_{Write} are applied at the terminals T1 and T2 of the memristor. The final result is stored as resistive state Z. For Z = p IMP q the following steps are performed:

1. Init device Z to '1' (VT1 = +½V_{Write}, VT2 = -½V_{Write})
2. Z' = p IMP q (VT1 = Vq, VT2 = Vp)
3. Read Z'

IMP can be used to design arithmetic operations such as adders [58, 56]; hence, it paves the path to more complex memristive in-memory-computing architectures.

V. CONCLUSION

This paper discusses data storage and analysis as one of the most critical challenges for today's and future data-intensive and big-data problems. It shows how the increase of the data size has already surpassed the capabilities of today's computation architectures suffering from the limited bandwidth, energy inefficiency and limited scalability. Thereafter, the paper proposes a new architecture based on the integration of the storage and computation in the same physical location (using a crossbar topology); the architecture is driven by non-volatile resistive-switching technology (memristors) instead of traditional CMOS technology. Therefore, it has the potential to reduce both the memory wall and energy consumption with orders of magnitude, and enables massive parallelism. Hence, significantly improving the performance and enabling the solution of big-data problems. The details and many aspects of the architecture still need to be worked out.

ACKNOWLEDGMENT

We would like to thank all people who contributed to the discussion of CIM architectures including Zaid Al-ars from delft University of Technology, Jan van Dalfson from Eindhoven University of Technology, Luca Benini from ETHZ, Shahar Kvatinsky from Technion, Lotfi Mhamdi from Leed University, Albert Cohen from INRIA, Stephan Menzel and Vikas Rana from RWTH, and Andrea Fantini from IMEC.

3

REFERENCES

[1] P. Chen and C-Y Zhang, 'Data-intensive applications, challenges, techniques and technologies: A survey on Big Data', Information Sciences, v 275, pp. 314-347, 2014

[2] K. Lahiri, A. Raghunathan, 'Power analysis of system-level on-chip communication architectures', International Conference on Hardware/Software Codesign and System Synthesis, CODES + ISSS, pp 236-241, 2014

[3] P. Somavat and V. Namboodiri. 'Energy consumption of personal computing including portable communication devices', Journal of Green Engineering, 1(4):447–475, 2011

[4] M. Horowitz, 'Computing's Energy Problem and what we can do about it', slides of the keynote at ISSCC 2014

[5] A.W. Burks, et al., 'Preliminary discussion of the logical design of an electronic computing instrument', 1946

[6] H. Esmaeilzadeh, et al., 'Dark silicon and the end of multicore scaling', in Proceedings of the 38th annual international symposium on Computer architecture, pp. 365-376, 2011

[7] S. Bampi, and R. Reis, 'Challenges and Emerging Technologies for System Integration beyond the End of the Roadmap of Nano-CMOS', 'VLSI-SoC: Technologies for Systems Integration' (Springer Berlin Heidelberg), pp. 21-33, 2011

[8] T. Yuan, 'CMOS design near the limit of scaling', IBM Journal of Research and Development, 2002, 46, (2.3), pp. 213-222

[9] X.-J. Yang et al., 'Progress and Challenges in High Performance Computer Technology', J. Comp. Sci. Tech., 2006, 21, (5), pp. 674-681

[10] S. A. McKee, 'Reflections on the Memory Wall', CF'04, pp. 162, 2004.

[11] M.V. Wilkes, 'The Memory Wall and the CMOS End-point', SIGARCH Comput. Archit. News, 1995, 23, (4), pp. 4-6

[12] R.E. Bryant, 'Data-Intensive Scalable Computing for Scientific Applications', Computing in Science & Engin., 2011, 13, (6), pp. 25-33

[13] W.A. Wulf and S.A. McKee, 'Hitting the memory wall: implications of the obvious', SIGARCH Comp. Archit. News, 23, pp. 20-24, 1995

[14] D. Patterson et al. 'A case for intelligent RAM', IEEE Micro, 1997, 17, (2), pp. 34-44

[15] C. Hernandez et al., 'Energy and Performance Efficient Thread Mapping in NoC-Based CMPs under Process Variations', in International Conference on Parallel Processing, 2011, pp. 41-50

[16] Y. Kang et al., 'FlexRAM: Toward an advanced Intelligent Memory system', in IEEE 30th International Conference on Computer Design, pp. 5-14, 2012

[17] J. Draper et al., 'The architecture of the DIVA processing-in-memory chip', In Proceedings of the 16th international conference on Supercomputing, pp. 14-25, 2002

[18] M. Gokhale et al., 'Processing in memory: the Terasys massively parallel PIM array', Computer, vol. 28, pp. 23-31, 1995

[19] P. M. Kogge, 'EXECUBE-A New Architecture for Scaleable MPPs', in International Conference on Parallel Processing, Vol. 1, 1994, pp. 77-84

[20] P. M. Kogge et al., 'PIM architectures to support petaflops level computation in the HTMT machine', in International Workshop on Innovative Architecture for Future Generation High-Performance Processors and Systems, 1999, pp. 35-44

[21] D. G. Elliott et al., 'Computational RAM: implementing processors in memory', IEEE Design Test of Computers, vol. 16, pp. 32-41, 1999

[22] Z. Wang et al., 'DSP-RAM: A logic-enhanced memory architecture for communication signal processing', in IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, 1999, pp. 475-478

[23] M. Ken et al., 'Smart Memories: a modular reconfigurable architecture', in Proceedings of the 27th International Symposium on Computer Architecture, 2000, pp. 161-171

[24] T. L. Sterling and H. P. Zima, 'Gilgamesh: A Multithreaded Processor-In-Memory Architecture for Petaflops Computing', in ACM/IEEE Conference Supercomputing, 2002, pp. 48-48

[25] T. Sterling and M. Brodowicz, 'Continuum computer architecture for nano-scale and ultra-high clock rate technologies', in Innovative Architecture for Future Generation High-Performance Processors and Systems, 2005, pp. 1-9

[26] P. Dlugosch et al., 'An Efficient and Scalable Semiconductor Architecture for Parallel Automata Processing', IEEE Transactions on Parallel and Distributed Systems, 2014, vol. 99, pp. 3088- 3098

[27] N. Venkateswaran et al., 'Memory in Processor: A Novel Design Paradigm for Supercomputing Architectures', 2003, pp. 19-26

[28] L. W. Tucker and G. G. Robertson, 'Architecture and applications of the Connection Machine', Computer, vol. 21, pp. 26-38, 1988

[29] H. Plattner, 'SanssouciDB: An In-Memory Database for Processing Enterprise Workloads', Datenbanksysteme in Büro, Technik und Wissenschaft (German Database Conference), 2011

[30] ITRS ERD report. [Online]. Available: http://www.itrs.net/Links/2010ITRS/Home2010.htm, 2010

[31] B. Hoefflinger, 'Chips 2020: A Guide to the Future of Nanoelectronics', The Frontiers Collection, Springer Berlin Heidelberg, 2012, pp. 421–427

[32] J. McPherson, 'Reliability trends with advanced CMOS scaling and the implications for design', in IEEE Custom Integrated Circuits Conference, 2007, pp. 405–412

[33] S. Borkar, 'Design perspectives on 22Nm CMOS and beyond', in Proceedings of the 46th Annual Design Automation Conference, 2009, pp. 93-94

[34] G. Gielen, et al., 'Emerging yield and reliability challenges in nanometer CMOS technologies', in Proceedings of the Conference on Design, Automation and Test in Europe, pp. 1322-1327, 2008

[35] F. Schwierz, 'Graphene transistors', Nature nanotechnology, vol. 5, no. 7, pp. 487–496, 2010

[36] G. Agnus et al., 'Two-terminal carbon nanotube programmable devices for adaptive architectures', Advanced Materials, vol. 22, no. 6, pp. 702–706, 2010

[37] K. Boucart and A. M. Ionescu, 'Double-gate tunnel fet with high-gate dielectric', IEEE Transactions on Electron Devices, vol. 54, no. 7, pp. 1725–1733, 2007

[38] L. Chua, 'Memristor-the missing circuit element', IEEE Transactions on Circuit Theory, vol. 18, no. 5, pp. 507–519, 1971

[39] D. B. Strukov et al., 'The missing memristor found', Nature, vol. 453, no. 7191, pp. 80–83, May 2008

[40] D. B. Strukov et al., 'Hybrid cmos/memristor circuits', in IEEE International Symposium on Circuits and Systems, 2010, pp. 1967–1970

[41] H. Lee et al., 'Low-power and nanosecond switching in robust hafnium oxide resistive memory with a thin ti cap', IEEE Electron Device Letters, vol. 31, no. 1, pp. 44–46, 2010

[42] A. C. Torrezan et al., 'Sub-nanosecond switching of a tantalum oxide memristor', Nanotechnology, vol. 22, no. 48, pp. 1-7, 2011

[43] M.-J. Lee et al., 'A fast, high-endurance and scalable non-volatile memory device made from asymmetric Ta2O5x/TaO2x bilayer structures', Nature Materials, vol. 10, no. 8, pp. 625–630, 2011

[44] T. Prodromakis and C. Toumazou, 'A review on memristive devices and applications', in 17th IEEE International Conference on Electronics, Circuits and Systems, 2010, pp. 934–937

[45] W. Zhao et al., 'Nanodevice-based novel computing paradigms and the neuromorphic approach', in IEEE International Symposium on Circuits and Systems, 2012, pp. 2509–2512

[46] Y. S. Chen et al., 'Highly scalable hafnium oxide memory with improvements of resistive distribution and read disturb immunity', in IEEE International Electron Devices Meeting, 2009, pp. 1–4

[47] J. J. Yang et al., 'High switching endurance in tao memristive devices', Applied Physics Letters, vol. 97, p. 232102- 232103, 2010

[48] J. J. Yang et al., 'Memristive devices for computing', Nat. Nanotechnol., vol. 8, pp. 13-24, 2013

[49] J. Borghetti et al., ''Memristive' switches enable 'stateful' logic operations via material implication', Nature, 464, pp. 873-876, 2010

[50] S. Hamdioui et al., 'Memristor based memories: Technology, design and test', IEEE 9th International Conference on Design & Technology of Integrated Systems In Nanoscale Era, pp. 1-7, 2014

[51] E. A. Worthey, 'Analysis and annotation of whole-genome or whole-exome sequencing-derived variants for clinical diagnosis', Current Protocols in Human Genetics, 2001

[52] B. Parhami, Computer arithmetic: algorithms and hardware designs. Oxford University Press, Inc., 2009

[53] A. Muttreja et al., 'CMOS logic design with independent-gate FinFETs', 25th International Conference on Computer Design, pp. 560–567, 2007

[54] C. Meinhardt et al., 'FinFET basic cells evaluation for regular layouts', in IEEE Fourth Latin American Symposium on Circuits and Systems, pp. 1–4, 2013

[55] D. Levinthal, Cycle Accounting Analysis on Intel R CoreTM2 Processors. Intel Corp, 2008

[56] C.Y. Lee et al., 'CACTI-FinFET: An integrated delay and power modeling framework for FinFET-based caches under process variations', in 48th Design Automation Conference, pp. 866–871, 2011

[57] E. Karl et al., 'A 4.6 GHz 162Mb SRAM design in 22nm tri-gate CMOS technology with integrated active V MIN-enhancing assist circuitry', in IEEE International Solid-State Circuits Conference Digest of Technical Papers, pp. 230–232, 2012

[58] S. Kvatinsky et al., 'Memristor-Based Material Implication (IMPLY) Logic: Design Principles and Methodologies', IEEE Transactions on VLSI Systems, vol. 22, no. 10, pp. 2054–2066, 2014

[59] A. Siemon et al., 'A Complementary Resistive Switch-based Crossbar Array Adder', arXiv:1410.2031, 2014

[60] J. Walker, Memristor and the future, [Online]. Available: http://www.nobeliefs.com/memristor.htm, 2010

[61] R. Waser et al., 'Redox-Based Resistive Switching Memories - Nanoionic Mechanisms, Prospects, and Challenges', Advanced Materials, vol. 21, pp. 2632-2663, 2009

[62] B. Govoreanu et al., '10x10nm2 Hf/HfOx Crossbar Resistive RAM with Excellent Performance, Reliability and Low-Energy Operation', IEEE International Electron Devices Meeting, pp. 31.6.1-31.6.4, 2011

[63] K. Terabe et al., 'Quantized conductance atomic switch', Nature, vol. 433, pp. 47-50, 2005

[64] M. Meier et al., 'A Nonvolatile Memory With Resistively Switching Methyl-Silsesquioxane', IEEE Electron Device Letters, vol. 30, pp. 8-10, 2009

[65] M. N. Kozicki et al., 'Nanoscale memory elements based on solid-state electrolytes', IEEE Transactions on Nanotechnology, vol. 4, pp. 331-338, 2005

[66] Z. Wei et al., 'Retention model for high-density ReRAM', 4th IEEE International Memory Workshop, pp. 1-4, 2012

[67] M. Kund et al., 'Conductive bridging RAM (CBRAM): an emerging non-volatile memory technology scalable to sub 20nm,' IEEE International Electron Devices meeting Technical Digest, pp. 754 – 757, 2005

[68] S. Ferch et al., 'Simulation and Comparison of two Sequential Logic-in-Memory Approaches Using a Dynamic Electrochemical Metallization Cell Model', Microelectronics Journal, vol. 45, pp. 1416-1428, 2014

[69] R. S. Williams et al., 'Physics-based memristor models', IEEE International Symposium on Circuits and Systems, pp. 217-220, 2013

[70] E. Linn et al., 'Applicability of Well-Established Memristive Models for Simulations of Resistive Switching Devices', IEEE Transactions on Circuits and Systems, vol. 61, pp. 2402 - 2410, 2014

[71] M. D. Pickett et al., 'Switching dynamics in titanium dioxide memristive devices', Journal of Applied Physics, 2009

[72] J. P. Strachan et al., 'State Dynamics and Modeling of Tantalum Oxide Memristors', IEEE Transactions on Electron Devices, vol. 60, pp. 2194-2202, 2013

[73] D.B. Strukov et al., 'Prospects for Terabit-scale Nanoelectronic Memories', Nanotechnology, vol. 16, no. 1, pp. 137–148, 2005

[74] H.D. Lee et al., 'Integration of 4F2 selector-less crossbar array 2Mb ReRAM based on transition metal oxides for high density memory applications', Symposium on VLSI Technology, pp. 151-152, 2012

[75] N. Ramaswamy, 'Challenges in Engineering RRAM technology for high density applications', IEEE International Workshop On Integrated Reliability, pp. 1-5, 2012

[76] A. Flocke et al., 'Fundamental analysis of resistive nano-crossbars for the use in hybrid Nano/CMOS-memory', 33rd European Solid-State Circuits Conference, pp. 328-331, 2007

[77] H. Manem et al., 'Design considerations for variation tolerant multilevel cmos/nano memristor memory', in Symposium on Great Lakes Symposium on VLSI, pp. 287-292, 2010

[78] E. Linn et al., 'Complementary Resistive Switches for Passive Nanocrossbar Memories', Nature Materials, vol. 9, pp. 403-406, 2010

[79] J.J. Yang et al., 'Engineering nonlinearity into memristors for passive crossbar applications', Applied Physics Letters, 2012

[80] M. A. Zidan et al., 'Memristor-based memory: The sneak paths problem and solutions', Microelectronics Journal, 44 (2), pp. 176-183, 2013

[81] M. Qureshi et al., 'AC sense technique for memristor crossbar', Electronics Letters, vol. 48, pp. 757-758, 2012

[82] M. R. Stan et al., 'Molecular electronics: from devices and interconnect to circuits and architecture', Proceedings of the IEEE, vol. 91, pp. 1940-1957, 2003

[83] M. Liu et al., 'Application of nanojunction-based RRAM to reconfigurable IC', Micro Nano Letters, vol. 3, pp. 101-105, 2008

[84] L. Nielen et al., 'An Experimental Associative Capacitive Network based on Complementary Resistive Switches for Memory-intensive Computing', 2014 IEEE Silicon Nanoelectronics Workshop, 2014

[85] E. Lehtonen et al., 'Stateful Implication Logic with Memristors', IEEE/ACM International Symposium on Nanoscale Architectures, pp. 33-36, 2009

[86] A. Dehon, 'Nanowire-Based Programmable Architectures', ACM Journal on Emerging Technologies in Computing Systems, vol. 1, pp. 109-162, 2005

[87] K. K. Likharev et al., 'CMOL: Devices, Circuits, and Architectures', Introducing Molecular Electronics, vol. 680, pp. 447-477, 2006.

[88] S. Paul et al., 'Computing with Nanoscale Memory: Model and Architecture', pp. 1-6, 2009

[89] S. Paul et al., 'A Scalable Memory-Based Reconfigurable Computing Framework for Nanoscale Crossbar', IEEE Transactions on Nanotechnology, vol. 11, pp. 451-462, 2012

[90] K. Eshraghian et al., 'Memristor MOS Content Addressable Memory (MCAM): Hybrid Architecture for Future High Performance Search Engines', IEEE Transactions on VLSI Systems, vol. 19, pp. 1407-1417, 2011

[91] S.J. Lee et al., 'Complementary Resistive Switch (CRS) Based Smart Sensor Search Engine', 8th IEEE International Conference on Intelligent Sensors, Sensor Networks and Information Processing, pp. 485 - 490, 2013

[92] G. Snider, 'Computing with hysteretic resistor crossbars', Applied Physics A, vol. 80, pp. 1165–1172, 2005

[93] E. Linn et al., 'Beyond von Neumann-logic operations in passive crossbar arrays alongside memory operations', Nanotechnology, vol. 23, pp. 305205/1-6, 2012

[94] L.O. Chua, 'Resistance switching memories are memristors', Applied Physics A, vol. 102, pp. 765-783, 2011

[95] H Aziza, et. al, 'A novel test structure for OxRRAM process variability evaluation', - Microelectronics Reliability, Vol. 53, N. 9, pp.1208-1212, 2013

Computation-In-Memory Based Parallel Adder

Hoang Anh Du Nguyen, Lei Xie, Mottaqiallah Taouil, Razvan Nane, Said Hamdioui, Koen Bertels
Laboratory of Computer Engineering, Faculty of EE, Mathematics and CS
Delft University of Technology, Mekelweg 4, 2628 CD Delft, The Netherlands
Email: H.A.DuNguyen@tudelft.nl

3

***Abstract*—Today's computing systems suffer from memory/communication bottleneck, resulting in energy and performance inefficiency. This makes them incapable to solve data-intensive applications within economically acceptable limits. Computation-In-Memory (CIM) architecture, based on the integration of storage and computation in the same physical location using non-volatile memristor technology offers a potential solution for the memory bottleneck. This paper presents a CIM based parallel adder, and shows its potentials and superiority for intensive computing and massive parallelism by comparing it with state-of-the art computing systems including multicore, GPU and FPGA architecture. The results show that CIM based parallel adder can achieve at least two orders of magnitude improvement in computational efficiency, energy efficiency and area efficiency.**

I. INTRODUCTION

In the last several decades, CMOS down-scaling has been the primary driver behind computer performance improvement [1]. However, CMOS technology is reaching its physical -if not economical- limits [2]. Down-scaling devices has led to many challenges such as leakage power [2], reliability [3], fabrication process and turnaround time [4], test complexity [4], cost for mask and design [5], and yield [6]. Furthermore, the performance gain by increasing clock speed has saturated since early 2000 [7]; today, speed-up is no longer the result of a faster clock, but rather a result of parallelization on multi-core and many-core systems. However, the number of parallel cores that can be programed and the computation efficiency that can be extracted are tending to saturate as well [8]. Obviously, all today's computing systems are mainly built on John von Neumann stored-program computer concept [9]. A major drawback of this computer design is the gap between the processing units and the main memory, the so-called memory bottleneck [7,10]. For data-intensive applications, the memory bottleneck is becoming even more severe and is putting major limitations both on performance and energy consumption. All of these motivate the need for a new architecture being able to (a) eliminate the communication bottleneck and support massive parallelism to increase the overall performance, (b) reduce the energy inefficiency to improve the computation efficiency.

Getting the memory closer the processing unit and reducing the memory bottleneck has attracted a lot of attention. In 1969, Logic-In-Memory (LIM) was originally introduced as a memory accelerator [11]; i.e., add some processing units close to main memory. In 1992, LIM concept re-appeared and named computational RAM, and typically uses the same accelerator concept where these are supposed to perform operations needed by the memory such as address translations [12]. In the late 1990s and early 2000s, Processor-In-Memory (PIM) was proposed [13] and manufactured [14]. PIM is based on splitting the main memory in different parts, each with surrounded computing units to bring the computation near to the memory; the architecture has a master CPU that takes care of the overall control. PIM concept was later used and refined for different applications; examples are EXECUBE [15], IRAM [16], FlexRAM [17], DIVA [18], Gilgamesh [19]. In 2004, Memory-In-Logic (MIL), which provides massive addressable memory on the processor, was proposed for supercomputer systems [20]. All mentioned above efforts have tried to close the gap between processor and memory speed [21]. However, as the computation and the storage are kept separately, they fundamentally use von Neumann stored-program computer concept and therefore suffer from memory bottleneck, which negatively impacts the performance [7].

This paper uses Computation-In-Memory (CIM) concept, that we have recently developed [22], to design a parallel adder and illustrate the huge potential of such an architecture for a simple case study: intensive arithmetic operations (additions). The architecture uses a revolutionary approach based on (a) the integration of storage and computation in the same physical location, and (b) non-volatile memristor technology [23]. It is worth noting that adding multiple numbers is a basic yet very representative operation in big data applications [24]. In existing architectures (e.g., multicore, GPU, and FPGA), simple operations such as adding multiple numbers already face the memory bottleneck. As the processors have to fetch huge amounts of data from memory, the intrinsic parallelism cannot be exploited fully in such architectures.

The main contributions of this paper are:

- A CIM based parallel adder for intensive computing.
- The evaluation of the proposed adder and comparison of its performance with traditional architectures including multicore, GPU and FPGA.

The proposed design achieves at least two orders of magnitudes improvements for big problems!

The rest of this paper is structured as follows. Section II briefly describes the concept of CIM architecture, and presents the CIM parallel adder. Section III provides estimations of CIM parallel adder's performance and compares it with other traditional architectures. Section IV shows our evaluation results and analysis. Finally, section V concludes this paper.

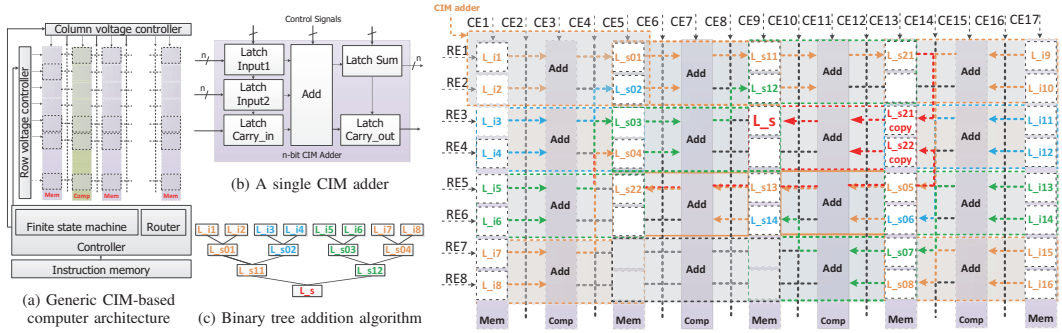
II. CIM PARALLEL ADDER

This section briefly first describes the CIM architecture. Thereafter, it presents the concept of the parallel adder. Finally, it demonstrates how to map these adders efficiently on the crossbar architecture.

A. Generic CIM Computer Architecture

The main advantage of CIM architecture over von-Neumann architectures is the tight integration of both computing

978-1-4673-7849-9/15/$31.00 ©2015 IEEE



(a) Generic CIM-based computer architecture

(b) A single CIM adder

(c) Binary tree addition algorithm

(d) CIM adders mapped onto CIM architecture

Fig. 1: CIM parallel adder

and storing operations using the same physical crossbar. Hence, massive parallelism is achieved with minimized communication.

Fig. 1a shows the main three CIM architecture components: crossbar, communication network and controller. The crossbar consists of memristors that are used to implement logic functions and/or storage. The communication network is either implemented within the crossbar or by using separate metal layers. A controller that is implemented by CMOS devices handles auxiliary operations such as distributing data and controlling signals to the crossbar.

The crossbar is specialized to perform computation and storage operations in cells organized in rows and columns. Each cell can be a computational unit (such as an adder or multiplier) or storage location (such as a memory cell). The cells in a row or column can be configured with the same or different functionality. The communication in CIM architecture has maximum flexibility. The architecture allows bi-directional communication in both horizontal and vertical direction. The controller contains a router and a finite state machine (FSM). The router provides the FSM with a communication scheme for data distribution and movements. The FSM fetches instructions from an instruction memory (e.g. hard disk), converts fetched instructions to controlling signals for the row/column voltage controller.

In this paper, our focus is to investigate the effectiveness of the crossbar with respect to computation and storage. Details on controller and communication are under further investigation.

B. A CIM-based Adder Tree

Fig. 1b shows a single CIM adder. The basic computational unit is an *n*-bit adder [25,26], which is surrounded by a number of memory cells (latches). An *n*-bit adder contains three *n*-bit latches (two for the inputs and one for the sum), a 1-bit carry-in and a 1-bit carry-out latch.

The CIM parallel adder arranges multiple CIM adders in a binary tree network. The carry-in and carry-out registers of an adder are connected properly to generate correct addition results. The binary tree network is ineffective using traditional platforms due to the difference between processor and memory fabrication. A processor coupled with a large amount of memory is unrealistic with traditional CMOS technology. Using the new features of the CIM architecture and its underlying memristor technology, the adder tree can be effectively mapped to reduce addition latency and increase resource utilization.

Fig. 1d shows a mapped binary adder tree with 16 inputs (see Fig. 1c for an 8 input example) on the CIM architecture. Each CIM adder corresponds to the adder presented in Fig 1b. Note that the output latches *sum* at each adder are reused as input latches in the next adder stage. The first column of the crossbar gets the first half of the inputs L_i. *Add* units in the second column add every two corresponding input latches L_i and store results in corresponding output latches L_s in the third column. The fourth column adds up results from output latches of the third column. Another direction of computation happens from the final column backwards to utilize as many resources as possible. In other words, a cell in CIM architecture is configured to an *add* unit or a latch. The interconnects (dotted lines) between multiple rows and columns represent communication channels among cells.

The crossbar array for N additions contains at least $\frac{N}{2} \times (log_2(N))$ CIM adders. Due to the multi-directional characteristic of the CIM-based adder, additions can operate in two direction flows (from left to right and vice versa) of the array (as shown in Fig. 1d). These bi-directional operations efficiently exploit resources in the architecture. Therefore, the architecture is designed with one additional column and half number of rows in comparison with the above-mentioned size. Hence, for N inputs the delay and array size equals $log_2(N)+1$ and $\frac{N}{4} \times (2log_2(N)+1)$ cells, respectively (each adder processes two inputs). With this design, every operation is performed on a distinct operational and storage unit; hence, there is no operation overlapping at a particular location. In addition, maximum number of adders in the architecture are used to avoid idle adders. With smart communication schemes, the architecture can be pipelined to increase overall performance.

III. ARCHITECTURAL CONSIDERATIONS

To illustrate how the CIM architecture improves the state-of-the-art, the performance of CIM is evaluated and compared

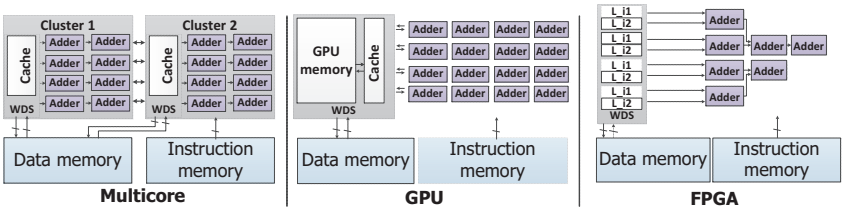


Fig. 2: Multicore vs GPU vs FPGA vs CIM parallel adders

against modern computer architectures using the addition test case. The binary tree addition algorithm (as shown in Fig. 1c) was mapped on four architectures. The following section describes the architecture models and performance metrics.

A. Architecture Models

For this estimation, we build simplified models that embed the basic characteristics of multicore, GPU and FPGA platforms. These models are referred to as multicore, GPU and FPGA architecture. To prevent an unfair comparison between the different architectures, optimistic assumptions are being made for the modern architectures (multicore, GPU and FPGA architecture) while pessimistic ones for CIM. For the multicore, GPU and FPGA architecture, an architecture with only parallel adder (not a complete processor) is used. For the CIM architecture, we use the dedicated adder tree of Fig. 1d. We assume that all modern architectures have processing units, working data sets, a proper controller and memories. Fig. 2 depicts assumed architecture for the multicore, GPU and FPGA. The CIM architecture, in contrast to the other architectures, only requires an instruction memory as the data is stored inside the crossbar. For the other architectures, a data memory is required. As this paper focuses on the computation and storage only, the cost for initial data load, controller and memories are not considered for all evaluated platforms.

The main assumptions for all the architectures are described in Table I. All of them are evaluated assuming 2^2 to 2^{21} inputs using 32-bit adders. For each architecture, the most optimistic available technology data is used. That is, multicore and GPU are based on data from 22nm technology, while the adder tree in the FPGA architecture was simulated with Virtex-7 (FFG1157 FPGA) using the 28nm technology library. For CIM architecture, assumptions are based on the ITRS 5nm memristors [28]. Though the physical memristor size has an order of magnitude benefit in comparison with other architectures, the estimation aims to show architectural impact more than technology dependence. In addition, worst-case assumptions are made for CIM architecture while the best-case for the other architectures. It is important to note that the comparison is one-by-one between CIM and other architectures. Indirect comparisons among multicore, GPU and FPGA are not relevant in this paper due to the optimistic assumptions made for each architecture.

Each processing unit is an adder that is organized depending on the architecture's characteristics. For the multicore architecture, 32 adders are grouped in a cluster (mimicking a 32-core system). Multiple clusters together form the architecture. Each adder gets its inputs from a cache. For the GPU architecture, each GPU core contains only a single adder. Hence, the GPU architecture has a large number of adders. The adders in multicore and GPU architecture are assumed to execute as many parallel tasks as possible. For FPGA architecture, the adders are connected in a binary tree network, the same one used in Fig 1c for CIM. The difference between multicore or GPU architecture versus FPGA or CIM architecture is whether the adders are organized in a binary tree network or not. Adders in FPGA and CIM architecture are organized using a binary tree addition network, while multicore and GPU architecture use as many adders as possible at every stage of computation. Note that the logarithmic addition algorithm is also used in these cases. However, the organization of adders impacts the area and power metrics of each architecture.

The working data set (WDS) differs for each architecture. Data is loaded initially from a data or program memory (e.g. hard disk, etc.) to this WDS. For multicore architecture, the WDS is an 8KB cache for each cluster. For GPU architecture, the WDS includes a 6GB GPU global memory and a 64KB global cache for the whole architecture. As data is loaded from memories, a particular hit rate, hit delay, miss rate, and miss penalty are assumed for multicore and GPU architecture. These memory characteristics are based on optimistic estimations on existing multicore and GPU architectures. In particular, the area and power data for cache is derived from data of 512KB-cache. The area and power of GPU global memory are provided by NVIDIA GPU datasheets [37]. As hit delay for a cache in multicore and GPU is assumed as fast as 1 cycle (with 1GHz clock rate), the register file is not considered in these architectures. For FPGA architecture, the WDS consists of a big register file. For CIM architecture, the WDS consists of interleaved latches with adders as described in Section II-B. Four architectures with different configurations of processing units and working data sets show the diversity of computer architectures in this estimation.

For multicore, GPU and CIM architecture, estimations were performed on assumptions listed in Table I. The FPGA implementation is generated by Vivado HLS tool [27] to ensure a good FPGA design and simulated by Xilinx ISE [27]. Due to large simulation time and limited FPGA chip area, measurements from a small-scale implementation simulation are scaled up for large-scale FGPA implementation. The calculations and measurements include only evaluated architecture components such as memory and adders.

TABLE I: Assumptions for each architecture

Arch.	Multicore	GPU		FPGA	CIM
Technology	22nm			28nm	5nm
Adder design	Rippled eight 4-bit Carry-look-ahead adder			auto-mapped on FPGA	Aachen Toggle-Cell-Adder
Area-parameter	208 CMOS gates [24]			1.47mm^2/adder [27]	34 memristors [25]
	0.248um^2/gate [28]				100nm^2
	51.6um^2/adder				3400nm^2
Delay-parameter	18 gates [24]			7.17ns/adder [27]	133 steps [25]
	9ps/gate [29,30]				200ps/step [31]
	162ps/adder				26600ps/adder
Energy-parameter	Static power + Dynamic power			0.0173W/adder [27]	Dynamic power
Leakage power consumption	$I_{leakage} * V_{dd}$ 6.15pA [28] * 0.86V				0
Dynamic power consumption	67mW/gate [29,32]				246fJ [28]
Memory design	8KB cache/cluster	64KB cache	6GB GPU global memory	Register file	Scalable memristor-based memory
Memory operating frequency	1GHz			No cache miss	No cache miss
Hit rate	0.95	0.90	0.995		
Load delay on hit	1 cycle	1 cycle	96 cycles [33,34]		
Missed penalty	165 cycles [35]	96 cycles [33,34]	165 cycles [35]		
Area	0.0092mm^2/cache [36]	0.0737mm^2/cache [36]	529mm^2/memory [37]		
Static power	0.0156W/cache [38]	0.125W/cache [38]	68W/memory [37]		
Dynamic power	25% static power	25% static power	25% static power		

Adders are designed specifically for their target architecture. Multicore and GPU architecture use a ripple carry adder with eight 4-bit Carry-Look-Ahead adders. The delay of an adder is calculated based on the gate delay and the number of required gates per adder in the longest path. Area is calculated based on the required number of CMOS gates per adder. The characteristics of a FPGA adder are extracted from ISE and shown in Table I. However, as the FPGA implementation is generated and mapped automatically by synthesis tools, the measurements of a large scale implementation are not scaled up using a single FPGA adder characteristics. Instead, an FPGA implementation of 256 adders is used to estimate larger implementations. CIM architecture uses a 32-bit Carry-Ripple-Adder based on memristor [25]. The delay of an adder is calculated by the number of steps to perform an addition. Each step corresponds to one memristor delay, which is the worst-case estimated as 200ps [31]. Area is calculated based on the required number of memristors per adder [25]. The above memristor-based adder is the fastest memristor-based adder available in literature. Meanwhile, other adder designs (multicore and GPU) are optimized for delay and area.

In order to make a realistic estimation, we investigated two cases: infinite resources and limited resources. In the first case, we assume architectures' resources is infinitely scalable. In the second case, we assume a maximum amount of resources based on existing devices that implement each architecture. In particular, the most recent finfet chip of 22nm data is used for multicore and GPU architecture while Virtex-7 chip data is used for FPGA architecture. When required resources cannot fit on a chip, we assumed multiple chips are used for multicore, GPU and FPGA architecture, respectively. Hence, extra area was counted for extra chips, which made delay and energy increase accordingly. We ignored additional delay and energy for transferring data among chips/GPUs/FPGAs. This makes their delay/area estimation optimistic. Memristor device currently has no implementation, hence we assumed maximum size of a memristor chip was $0.7mm^2$, in which 30% extra area is assumed for inter-chip communication. The chip area defined the limit of multicore architecture's area. Similar numbers for GPU and FPGA were chosen.

B. Performance Metrics

The estimation was performed and verified in Matlab. We consider three metrics: $total\_delay$ (D), $total\_energy$ (E), and $total\_area$ (A) described by Equation 1. Each parameter consists of two components: computation made by adders and communication made by local data memory access. The interconnection and controller are not considered in this estimation. Energy is calculated by delay and power. Both static power and dynamic power are considered for energy estimation. Static power is mainly caused by leakage current while dynamic power is consumed by switching activities. Computational activities related to working adders consume dynamic power while idle adders and WDS consumes mostly static power. CIM architecture is based on memristor; hence, it is claimed to consume no static power [39]. From the three above metrics, we derived three performance metrics: computation efficiency (η_C), energy efficiency (η_E) and area efficiency (η_A). Equations for these parameters are described in Equation 2.

$$\begin{aligned} D &= D_{comp} + D_{comm} \\ E &= E_{comp} + E_{mem} \\ A &= A_{comp} + A_{comm} \end{aligned} \quad (1)$$

$$\eta_C = \frac{D * E}{\#ops} \qquad \eta_E = \frac{\#ops}{E} \qquad \eta_A = \frac{\#ops}{A} \quad (2)$$

IV. RESULTS

A. Infinite Resources

Fig. 3 shows the performance of the four architectures when resources are assumed to scale up infinitely.

With respect to delay, Fig. 3 shows that CIM architecture performs slowest among four architecture. As all four architectures perform the same *n*-additions using binary tree algorithm, the delay for computation is *log(n)* stages. The differences among four architectures are memory access delay and the amount of time to perform a single addition. As architectures with adders organized in a binary tree network (FPGA and CIM architecture) benefit from fewer data loads and stores, they have lower delay in memory accessing. In

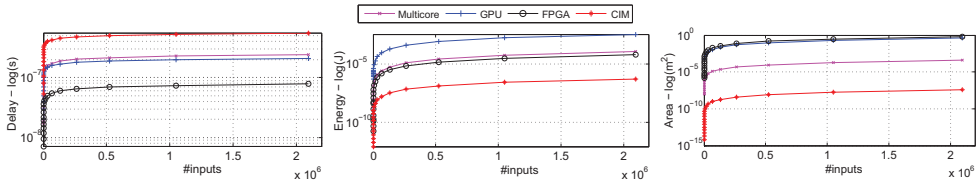


Fig. 3: Multicore vs GPU vs FPGA vs CIM parallel adders without resource constraints (intrinsic parameters)

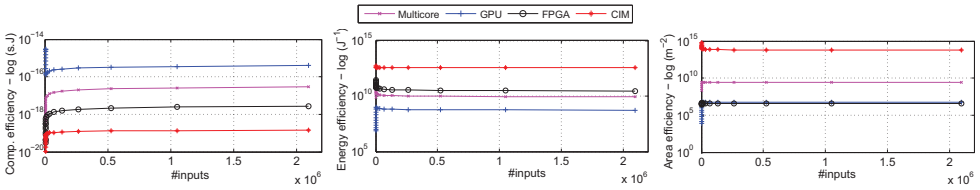


Fig. 4: Multicore vs GPU vs FPGA vs CIM parallel adders without resource constraints (derived parameters)

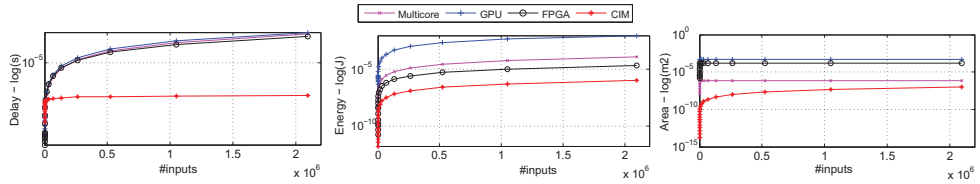


Fig. 5: Multicore vs GPU vs FPGA vs CIM parallel adders with resource constraints (intrinsic parameters)

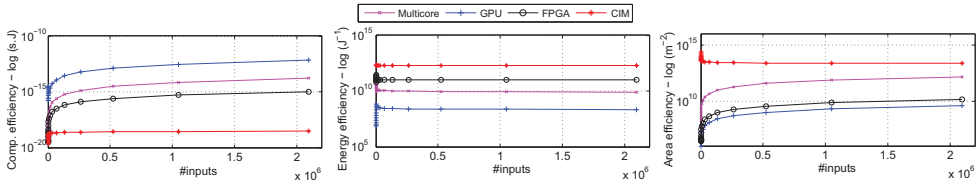


Fig. 6: Multicore vs GPU vs FPGA vs CIM parallel adders with resource constraints (derived parameters)

addition, the FPGA architecture has no cache and data is loaded from registers, while multicore architecture has the highest miss rate in comparison with other architectures with cache. However, memristor has slower switching speed [28] and large number of addition steps [25]. Hence, an addition using CIM takes much more time than FPGA, multicore and GPU architecture. CIM performs nearly four times slower than FPGA architecture while GPU and multicore architecture perform twice faster than CIM architecture.

For energy and area, CIM architecture consumes the least while GPU and multicore architecture consume much more in energy and area (as shown in Fig. 3). The high energy and area consumption was caused mostly by the cache and memory. As unlimited available resources are assumed, area and energy scale up with the input size. For multicore architecture, only a small cache (8kB) was included in each cluster. However, as the number of inputs scales up, the amount of caches in the whole architecture increases. For GPU architecture, the WDS contain 6GB of global memory and 64kB of cache in a default configuration. However, this increased amount of memories does not scale up linearly as in multicore architecture. The default amount of GPU WDS corresponds to 1536 adders. When the number of inputs requires more adders, a scale-up-ratio is applied on the default WDS size. This practically means more GPU platforms have to be used, hence more energy and area are consumed. For FPGA architecture, the same principle applies as if resources are required more than a single platform (default size is taken from Virtex-7 platform) can support, multiple platforms are used. CIM architecture has an advantage of low power consumption, no static power, and small area. The advantage reflects clearly in the results as CIM architecture achieve low energy consumption and area cost. GPU architecture consumes energy linearly with the input size while other architectures' energy scales up 2 to 4 order magnitude less than GPU architecture. GPU and FPGA architecture also consume more area in comparison with multicore and CIM a factor of five and ten order of magnitude, respectively.

Using performance metrics (η_C, η_E and η_A), we observed that although CIM architecture has the highest delay metric in comparison with other architectures (as shown in Fig. 4), it performs the best with respect to the three performance metrics. Note that for computation efficiency, the lower values are better.

B. Limited Resources

Fig. 5 shows the performance of four architectures when resources are assumed to be limited. The maximum resources are dependent on the latest available chip size for each architecture. For multicore architecture, a chip size of $700mm^2$ is used for 22nm technology. For GPU architecture, a chip size of $300mm^2$ is used with 1536 adders. For FPGA architecture, a chip size of $400mm^2$ is used corresponding to Virtex-7 platform. For CIM architecture, no fabrication data is available yet; hence we assume the area constraint for a memristor chip around $0.7mm^2$ (1000 times smaller than the multicore architecture as memristor has advantage of physical size). With these resource constraints, area constantly stays at a particular input size. If the required resources are larger than the provided resources, the architecture has to reuse resources several times to perform the same amount of additions. Hence, the delay increases, which leads to increasing static power consumption of local memories and dynamic power consumption of adders.

Fig. 6 shows that CIM performs better than other architectures in all performance metrics. There is a large gap among CIM and other architectures. For computational performance, CIM architecture performs three order magnitude better than FPGA and seven order magnitude than multicore and GPU architecture. Indeed, CIM architecture achieves even lower delay in the case of limited resources. This gain comes from the advantage of smaller devices and lower energy consumption. Even though, the delay of a single memristor and memristor-based adder are high, an efficient architecture without the WDS (e.g. caches) shows significant performance improvement.

V. CONCLUSION

In this paper, we have presented a CIM-based parallel adder and estimated its performance. Despite the simplicity of the case study, the results clearly show that CIM architecture has a huge potential and orders of magnitude improvements. This is mainly due to reducing/eliminating memory accesses, using the non-volatile technology, and exploiting the high level of parallelism. CIM architecture seems to be very promising and could enable computation of current infeasible big data applications, fuelling important societal changes.

REFERENCES

[1] H. Esmaeilzadeh *et al.*, "Dark silicon and the end of multicore scaling," *SIGARCH Comput. Archit. News*, vol. 39, pp. 365–376, 2011.

[2] S. Borkar, "Design challenges of technology scaling," *IEEE MICRO*, vol. 19, pp. 23–29, Jul 1999.

[3] J. W. McPherson, "Reliability trends with advanced CMOS scaling and the implications for design," in *IEEE CICC*, 2007, pp. 405–412.

[4] S. Borkar, "Design perspectives on 22nm CMOS and beyond," in *DAC*. ACM, 2009, p. 9394.

[5] G. Declerck, "A look into the future of nanoelectronics," in *Symposium on VLSI Technology, Digest of Technical Papers*, 2005, pp. 6–10.

[6] G. Gielen *et al.*, "Emerging yield and reliability challenges in nanometer CMOS technologies," in *DATE*. ACM, 2008, p. 13221327.

[7] S. Kaxiras, *Architecture at the End of Moore*, ser. Advances in Atom and Single Molecule Machines. Springer Berlin Heidelberg, 2013, pp. 1–10.

[8] J. W. Janneck, "Computing in the age of parallelism: Challenges and opportunities," Keynote talk, 2013.

[9] A. W. Burks *et al.*, *Preliminary discussion of the logical design of an electronic computing instrument (1946)*. Ablex Publishing Corp., 1989, pp. 39–48.

[10] S. A. McKee, "Reflections on the memory wall," in *CF*. ACM, 2004.

[11] W. H. Kautz, "Cellular logic-in-memory arrays," *IEEE Transactions on Computers*, vol. C-18, pp. 719–727, 1969.

[12] D. G. Elliott *et al.*, "Computational RAM: implementing processors in memory," *IEEE Design Test of Computers*, vol. 16, pp. 32–41, 1999.

[13] P. M. Kogge *et al.*, "Pursuing a petaflop: point designs for 100 TF computers using PIM technologies," in *Symposium on the Frontiers of Massively Parallel Computing*, 1996, pp. 88–97.

[14] D. Keitel-Schulz and N. Wehn, "Embedded DRAM development: Technology, physical design, and application issues," *IEEE Des. Test*, vol. 18, pp. 7–15, 2001.

[15] P. M. Kogge, "EXECUBE-a new architecture for scaleable mpps," in *ICPP*, vol. 1, 1994, pp. 77–84.

[16] D. Patterson *et al.*, "A case for intelligent RAM," *IEEE Micro*, vol. 17, pp. 34–44, 1997.

[17] Y. Kang *et al.*, "FlexRAM: Toward an advanced intelligent memory system," in *ICCD*, pp. 5–14.

[18] J. Draper *et al.*, "The architecture of the diva processing-in-memory chip," in *ICS*. ACM, pp. 14–25.

[19] T. L. Sterling and H. P. Zima, "Gilgamesh: A multithreaded processor-in-memory architecture for petaflops computing," in *Supercomputing Conference*, pp. 48–48.

[20] N. Venkateswaran *et al.*, "Memory in Processor: A novel design paradigm for supercomputing architectures," ser. MEDEA '03. New York, NY, USA: ACM, pp. 19–26.

[21] E. Upchurch *et al.*, "Analysis and modeling of advanced PIM architecture design tradeoffs," in *Innovative Architecture for Future Generation High-Performance Processors and Systems*, 2003, pp. 66–75.

[22] S. Hamdioui *et al.*, "Memristor based Computation-in-Memory architecture for data-intensive applications," in *DATE*, 2015.

[23] L. O. Chua, "The fourth element," *JPROC*, vol. 100, pp. 1920–1927, 2012.

[24] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*. Oxford University Press, 2000.

[25] A. Siemon *et al.*, "A complementary resistive switch-based crossbar array adder," *IEEE JETCAS*, 2014.

[26] A. A. El-Slehdar *et al.*, "Memristor based N-bits redundant binary adder," *Microelectronics Journal*, vol. 46, pp. 207–213, 2015.

[27] Xilinx, "Xilinx Design tools," 2014.

[28] "The international technology roadmap for semiconductors ITRS," 2011.

[29] C. Meinhardt and R. Reis, "FinFET basic cells evaluation for regular layouts," in *IEEE LASCAS*, 2013, pp. 1–4.

[30] A. Muttreja *et al.*, "CMOS logic design with independent-gate FinFETs," in *ICCD*, 2007, pp. 560–567.

[31] C. T. Antonio *et al.*, "Sub-nanosecond switching of a tantalum oxide memristor," *Nanotechnology*, vol. 22, p. 485203, 2011.

[32] M. Bohr and K. Mistry, "Intels revolutionary 22nm transistor technology," Tech. Rep., 2011.

[33] V. Volkov, "Better performance at lower occupancy," UC Berkeley, Tech. Rep., 2010.

[34] C. Woolley, "GPU optimization fundamentals," Tech. Rep., 2013.

[35] D. Levinthal, "Performance analysis guide for Intel core i7 processor and intel xeon 5500 processors," Tech. Rep., 2008.

[36] E. Karl *et al.*, "A 4.6ghz 162mb SRAM design in 22nm tri-gate CMOS technology with integrated active VMIN-enhancing assist circuitry," in *ISSCC*, 2012, pp. 230–232.

[37] J. Zhao *et al.*, "Optimizing GPU energy efficiency with 3d die-stacking graphics memory and reconfigurable memory interface," *ACM Trans. Archit. Code Optim.*, vol. 10, pp. 1–25, 2013.

[38] L. Chun-Yi and N. K. Jha, "CACTI-FinFET: An integrated delay and power modeling framework for FinFET-based caches under process variations," in *DAC*, pp. 866–871.

[39] X. Yuan, "Modeling, architecture, and applications for emerging memory technologies," *IEEE Design & Test of Computers*, vol. 28, pp. 44–51, 2011.

2206 IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, VOL. 25, NO. 8, AUGUST 2017

On the Implementation of Computation-in-Memory Parallel Adder

Hoang Anh Du Nguyen, Lei Xie, *Student Member, IEEE*, Mottaqiallah Taouil, Razvan Nane, Said Hamdioui, and Koen Bertels, *Member, IEEE*

3

***Abstract*— Today's computer architectures suffer from many challenges, such as the near end of CMOS downscaling, the memory/communication bottleneck, the power wall, and the programming complexity. As a consequence, these architectures become inefficient in solving big data problems or general data intensive applications. Computation-in-memory (CIM) is a novel architecture that tries to solve/alleviate the impact of these challenges using the same device (i.e., the memristor) to implement the processor and memory in the same physical crossbar. In order to analyze its feasibility in depth, this paper proposes two memristor implementations of a data intensive arithmetic application (i.e., parallel addition). To the best of our knowledge, this is the first paper that considers the cost of the entire architecture including both crossbar and its CMOS controller. The results show that CIM architecture in general and the CIM parallel adder in particular have a high scalability. CIM parallel adder achieves at least two orders of magnitude improvement in energy and area in comparison with a multicore-based parallel adder. Moreover, due to a wide variety of memristor design methods (such as Boolean logic), tradeoffs can be made between the area, delay, and energy consumption.**

***Index Terms*— Boolean logic, computation-in-memory (CIM), implication logic, parallel adder.**

I. INTRODUCTION

COMPUTER performance improvement has in the previous decades mostly been the result of CMOS downscaling [1]. In recent years, CMOS downscaling is reaching its end [2], [3] due to many challenges such as leakage power consumption [4], reliability [5], fabrication process and turnaround time [6], test complexity [6], cost for mask and design [4], and yield [7]. As a result, increasing the clock frequency is no longer possible; performance gain has to be achieved through parallelism using multicore/many cores architectures. However, these architectures suffer from an inefficient programmability and high energy consumption [1], [8] due to a gap between memory and processing unit speed, the so-called memory bottleneck [2], [9], [10]. This is the core problem of the von Neumann store-program computer concept [11] used in today's computing systems, which leads to performance and energy inefficiency, especially for data intensive applications. Note that today supercomputers are used to deal with very limited number of data intensive (or compute intensive) applications; they are expensive, power hungry, and area inefficient [12]–[14]. Hence, there is a need for a novel architecture that significantly reduces the memory bottleneck, massively supports parallelism, and is energy efficient.

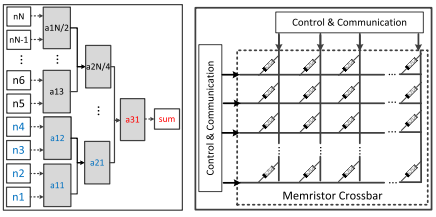
To alleviate the memory bottleneck and provide practical and efficient solutions for data intensive applications, many architectural solutions have been proposed. They can be classified into three categories. First, processor-in-memory (PIM) was introduced as an architecture that consists of a host CPU, main memory, and a number of accelerators close to the main memory to prevent intensive communication with the CPU [15]–[18]. Many implementations of this architecture have been proposed, e.g., EXECUBE [19], IRAM [20], FlexRAM [21], DIVA [22], and Gilgamesh [23]. However, the effectiveness of this architecture strongly depends on the technology to fabricate the accelerators and main memories, which is called merged-logic DRAM [24], [25]. Unfortunately, this merged technology still suffers from a high cost and low density [24], [25]. Second, near data architectures [26], [27] were proposed as a PIM architecture but using the emerging nonvolatile memory technology, either using traditional processor approach [28] or using novel neural computing approach [29]. Note that all aforementioned efforts, both at architectural and technology level, tried to close the gap between processor and memory speed [18]. However, as all of these architectures use stored-program/von Neumann concept, the computation is still carried out in a separate physical unit. Therefore, the memory bottleneck is still affecting the computer performance and energy [2], [3], [9], [10], [30]–[32]. Third, as a result of this, resistive computing architectures were introduced to alleviate the memory bottleneck using memristor technology. Although the memristor technology is not mature yet, several publications focused on the design of circuits and architectures such as programmable logic-in-memory architecture [33], resistive GP-SIMD processing-in-memory architecture [34], and computation-in-memory (CIM) computing paradigm [35]. However, these works mostly focus on the memristor part. They ignore the controlling and peripheral circuit overheads. Du Nguyen *et al.* [36] have analyzed the performance of a parallel memristor adder based on the crossbar (i.e., an adder that takes the sum of multiple inputs and produces a single output) using the CIM computing paradigm. They have shown for different metrics that such an adder outperforms state-of-the-art multicore, GPUs, and memristor implementations with two orders of magnitude. All the adder implementations were based on high-level assumptions. In this paper, we show the feasibility of implementing two different

Manuscript received July 28, 2016; revised November 17, 2016 and February 15, 2017; accepted March 18, 2017. Date of publication May 3, 2017; date of current version July 24, 2017.

The authors are with the Department of Computer Engineering, Delft University of Technology, 2628 CD Delft, The Netherlands (e-mail: h.a.dunguyen@tudelft.nl).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2017.2690571



(a) (b)

Fig. 1. CIM parallel adder. (a) Binary tree addition algorithm with eight inputs. (b) CIM architecture.

parallel adders in the memristor crossbar, i.e., one based on implication logic and one on Boolean logic.

To the best of our knowledge, this is the first publication that considers the cost of the entire system (i.e., the crossbar and CMOS controller). The delay, area and energy costs of the CMOS controller, peripheral circuits (required to read and write to the crossbar), and the crossbar itself are analyzed. In addition, we evaluate the scalability of the parallel adder for both implementations.

The rest of this paper is structured as follows. Section II briefly describes the parallel addition algorithm, CIM architecture, and how the parallel adder algorithm is mapped on the crossbar. Section III presents the state-of-the-art of memristor adders and selects the two best memristor adders. These two adders are used in Section IV to build the memristor-based parallel adder. In this section, we discuss all implementation aspects (i.e., memristor crossbar, interconnect network, peripheral, and CMOS controller). Section V first provides a performance comparison between the two memristor implementations and subsequently compares them with a multicore-based implementation; thereafter, it discusses the memristor implementation's advantages and disadvantages. Finally, Section VII concludes this paper.

II. COMPUTATION-IN-MEMORY PARALLEL ADDER

In this section, we describe the parallel addition algorithm and the CIM computing paradigm. Thereafter, we map the algorithm on CIM. Finally, we conclude with the potentials of the CIM parallel adder.

A. Parallel Addition Algorithm

Fig. 1(a) shows the parallel addition that is based on a reduction tree that computes the sum of multiple numbers. All the inputs n_1–n_N are added in the first stage in pairs, where N is the number of inputs. Subsequently, the obtained partial results are added in successive stages until the final result (sum) is obtained. The parallel addition algorithm is based on the binary reduction tree and has a $\log_2(N)$ lower bound time complexity. This complexity is realized when enough available resources are available (e.g., adders) to perform all additions in each stage concurrently.

B. Computation-In-Memory Architecture

The CIM architecture consists of three components [also see Fig. 1(b)].

1) A memristor crossbar, which consists of interweaved computation and storage units mapped on the grid where memristors are placed at horizontal and vertical nanowire junctions.
2) A control and communication unit, which applies control signals to the memristor crossbar; memristors are passive devices, and as such, they need to be controlled with conventional CMOS logic. Furthermore, this block enables communication within the memristor crossbar using nanowires and/or metal wires.
3) A peripheral circuit is required to interface between the controller and memristor crossbar. It consists of voltage drivers, sense amplifiers, and so on. Note that the peripheral circuit is not depicted in Fig. 1(b).

CIM architecture has the following major advantages:

1) Memory bottleneck reduction: The computation occurs directly inside the nonvolatile memory where data are stored; hence, the data movement between processors and memories is minimized. This alleviates or eliminates the memory bottleneck depending on applications.
2) Flexibility: The placement of computation and storage units is flexible and optimized based on application-specific requirements to enable local communication; this leads to low latency.
3) Massive parallelism: As computation and storage units are tightly coupled in the same physical memristor crossbar, the memory bottleneck is reduced. Hence, a huge amount of operations can be performed in parallel. Due to the memristor's hysteresis curve, they can be used both for logic and memory [35], [37].
4) Low area: Due to its small feature size of $4F^2$, the memristor crossbar requires a low area.
5) Low energy: As memristors have zero leakage, only dynamic energy has to be considered [38].

C. Parallel Adder

Du Nguyen *et al.* [36] mapped the parallel addition algorithm on the memristor crossbar referred to as CIM parallel adder. Due to a flexible placement, the communication between memory and logic units can be optimized by exploiting application-specific computational and storage patterns. For example, an h-tree layout is considered in the CIM matrix multiplier presented in [39]. Fig. 2 shows the organization of the parallel adder; this layout is selected due to its area efficiency.

The parallel addition algorithm based on the binary reduction tree [Fig. 1(a)] is split into two parts. The lower part of the tree with corresponding inputs n_i ($1 \leq i \leq N/2$) is used to add the first half of the inputs; their corresponding adders are mapped from left to right on the memristor crossbar [see Fig. 2]. To optimize the area and keep a regular structure, the upper part of the tree is mapped from right to left. The results of these two parts are finally added together in the middle of the memristor crossbar (see sum in Fig. 2). This mapping minimizes the communication between the adders and storage units, simplifies the interconnect network, and reduces the required area. Section IV provides more implementation details.

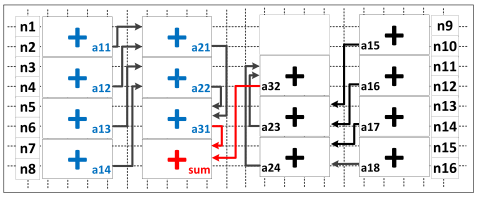


Fig. 2. CIM parallel adder with $N = 16$ inputs.

Previous results showed that the above CIM parallel adder significantly outperforms equivalent multicore, GPU, and FPGA implementations by at least two orders of magnitude in energy–delay product, energy efficiency, and area efficiency [36]. The improvement is the result of a novel architecture based on nonvolatile technology that reduces memory accesses and exploits application-specific parallelism and communication patterns.

III. Memristor Adder: The Basic Block

In this section, we present one of the main components of the parallel adder, i.e., the adder. Several memristor crossbar adders have been proposed in the literature. Table I summarizes their characteristics (i.e., logic and cell type; the cell is the repetitive structure found between the cross section of horizontal and vertical nanowires) and performance (i.e., delay and area). First, we discuss the working principle of the logic and cell types. Thereafter, we present the state-of-the-art adders. Finally, we select and discuss the two fastest adders in more detail for the parallel CIM adder implementation.

A. Working Principle

The adders of Table I are based on either implication [40] or Boolean logic. In implication logic, the logic operation "pIMPq" ("p implies q" or "if p, then q") and *FALSE* operation (always yields the logic value 0) form the computational basis of the complete logic space [41]. In Boolean logic, Boolean gates (e.g., NAND, NOR) form the computational basis of the complete logic space [37].

As the logic values are represented by resistances, resistive switching is required to perform logic operations. In Table I, two types of crossbar cells can be used: the bipolar resistive switch (BRS) cell [42] and the complementary resistive switch (CRS) cell [43]. The BRS cell consists of one memristor in which its state is represented by the resistance; the high and low resistances correspond to logic value 0 and 1, respectively. The CRS cell is composed of two stacked memristors (referred to as upper and lower memristors) with an equivalent resistance that is always equal to the high resistance of a BRS cell; the 0 (1) logic state is represented by a high (low) resistance state in the upper memristor and a low (high) resistance in the lower memristor. This avoids sneak path currents in the crossbar.

The major difference between BRS and CRS cells leads to different requirements in the interconnect network, controllers, and peripheral circuit, and therefore impact the adder performance. In case BRS cells are used, the communication between two cells may be carried out by applying appropriate voltages to nano-wires, the so-called copy operation [44]. The CMOS controller is responsible for the communication and also makes sure that the logic (whether implication or Boolean) is properly being executed. The peripheral circuit includes voltage drivers and sense amplifiers to read out the logic states out. For CRS cells, the copy operation cannot be used to transfer the logic state between two CRS memristors, as the equivalent resistance value is the same for logic 0 and 1. Instead, the controller reads first a value from a source cell, and subsequently writes this into the destination cell. The read-out and write-in operations are performed by applying appropriate voltages to the nanowires. The read-out operation generates a current spike only for read 0 that can be detected by a current sense amplifier. Note that this adder needs a more complex sense amplifier as it must detect a current spike [45]. The crossbar based on CRS cells also requires a controller to control read operations and perform the logic operations. The controller consists of more states as it first has to perform read-out operations before calculations can be made. Next, we describe the adder's features.

TABLE I
STATE-OF-THE-ART MEMRISTOR ADDERS

Adder	Ref.	Characteristics		Performance	
		Logic type	Cell type	Delay (32-bit) (steps)	Area (32-bit) (memristors)
Lehtonen	[40]	Impl.	BRS	$88m + 48$ (2864)	$3m + 5$ (101)
Kvatinsky	[46]	Impl.	BRS	$5m + 18$ (178)	$9m$ (288)
Siemon	[47]	Impl.	CRS	$4m + 5$ (133)	$m + 2$ (34)
Snider	[37]	Bool.	BRS	$24m$ (768)	$80m$ (2560)
Xie	[48]	Bool.	BRS	$7\frac{m}{2}$ (112)	$432\frac{m}{2}$ (6912)

Impl.: Implication Bool.: Boolean m: no. of bits

B. State-of-the Art Memristor Adders

Lehtonen's adder [40] is based on implication logic using BRS cells. This is the first published memristor adder based on implication logic; hence, it has a lower performance than the other adders. In addition, the adder may suffer from sneak path currents and destructive operations (i.e., operations that destroy the values stored in the memristors). As only a 1-bit adder was proposed in [40], an m-bit ripple-carry adder can be formed using m 1-bit adders.

Kvatinsky's adder [46] is based on implication logic using BRS cells. In [46], two multibit adders are proposed: a sequential and a fast adder. Note that Table I only mentions the fast adder. Both adders suffer from destructive operations as the implication logic affects the data stored in the cells. The fast adder also suffers from sneak path currents. In comparison with Lehtonen's adder, Kvatinsky's adder is optimized for multibit additions; hence, it is faster and smaller.

Siemon's adder [45] is based on implication logic using CRS cells. In [45], two multibit adder designs were proposed: precalculation (PC) and toggle-cell (TC) adder; Table I mentions only the TC adder. The TC adder is a parallelized version of the PC adder. Hence, the PC adder requires fewer memristors, but needs more execution steps than the TC adder. Both adders require fewer steps and memristors than

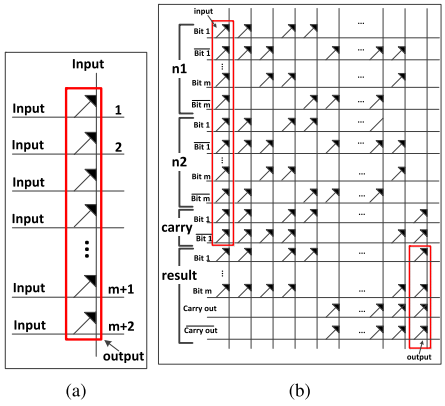


Fig. 3. Multibit memristor adders. (a) CRS adder. (b) FBLC adder.

Kvatinsky's multibit adder. The TC adder does theoretically not suffer from sneak paths as it employs CRS cells; however, due to the manufacturing process, resistance variations could lead to different cell resistances [48]. The drawback of this adder is that it suffers from destructive read-out operations. In addition, communication between memristors is devious as the controller first needs to read the cells' resistance state and subsequently, determine the next set of voltages. As CRS cells are employed, this adder needs relatively more complex controllers and sense amplifiers.

Snider's adder [37] is based on Boolean logic using BRS cells. The author proposed only a half adder that can be intuitively extended to a 1-bit adder. Subsequently, an m-bit ripple-carry adder can be formed by cascading multiple 1-bit adders. This is the first Boolean memristor adder; hence, its execution steps and memristors is not optimized. Furthermore, the adder may also suffer from sneak path currents.

Xie's adder [47] is based on Boolean logic using BRS cells. In [47], a 1-bit, a 2-bit, and a 4-bit adder were proposed using fast Boolean logic circuit (FBLC). These adders were based on lookup tables. To build an m-bit adder, a ripple-carry adder can be formed using the smaller adders. In Table I, the m-bit adder is formed from $m/2$ 2-bit adders. The 2-bit adder has been selected as the basic block due to its delay and area efficiency. This adder is the fastest among all listed adders. However, the adder requires a large number of memristors and additionally may suffer from sneak path currents.

C. Selected Adders

In this section, we select the two fastest multibit memristor adders (i.e., Siemon's and Xie's adder) as basic blocks for the CIM parallel adder. We refer to these basic blocks as CRS adder and FBLC adder, respectively. The FBLC adder is slightly faster, but requires a much larger area. The crossbar implementations of both adder are shown in Fig. 3 where the cells are represented by arrows; both adders are described next.

CRS adder is m-bit wide and needs (m+2) memristors [as shown in Fig. 3(a)]. The inputs are represented by voltages and are applied by the controller. Note that the inputs can be stored within the crossbar as well; this requires, however, extra memristors and an additional read-out operation before the normal operation can start. Depending on the required function, the input voltages are applied to the crossbar in a particular sequence, and finally, the outputs are represented by the resistance of the output memristors.

TABLE II
CRS ADDER CONTROL VOLTAGES

Instruction	Operation	Column voltage	Row voltage
WI	initialize	V_{high}	V_{low}
	write-in	$V_{high/low}$	V_{high}
RO	read-out	V_{high}	V_{low}
WB	write-back	V_{low}	V_{high}
CP	compute	$V_{high/low}$	V_{low}

The crossbar is controlled by a sequence of operations including: write-in (WI), read-out (RO), write-back (WB), and compute (CP). Before the operations can be applied, the crossbar is once entirely reset to a high resistance. The WI operation writes a logic value into a memristor. The RO operation reads a logic value from a cell; the logic value is determined by the sense amplifier. The RO operation is destructive and changes the value of the memristor to logic value 1. The task of the WB operation is to recover the destroyed value. Finally, the CP instruction uses implication logic [45], [49] and existing memristor logic values to update the value of the current computation step in the crossbar.

The number of execution steps depends on the adder data-width and takes place as shown in Fig. 4(b) and (c). In Fig. 4(b), $c0$ represents the carry-in, c_i the carry at bit i, and s_i the calculated sum at bit i. As the sum is calculated in two steps, the intermediate value of sum is denoted by s_i' [45]. The operations are executed by applying appropriate voltages on the horizontal and vertical nanowires. Table II shows these voltages (i.e., V_{high}, V_{low}) for each state. In case the control voltage is dependent on intermediate results, the voltage $V_{\text{high/low}}$ is used. We apply the half-select voltage to the nonactive nanowires, to reduce and prevent sneak path currents. The exact control values are described in [49].

FBLC adder is composed of $m/2$ 2-bit adders. Fig. 3(b) depicts a single m-bit adder; the 2-bit adder is obtained for $m = 2$. As the inputs/outputs are stored as resistances, dedicated areas need to be reserved for them; they are represented by the rectangles in the figure. Depending on the required function (e.g., copy or compute), the controller applies a sequence of control voltages to the crossbar nanowires.

The FBLC adder is controlled by a sequence of operations including: configure all minterms, evaluate all minterms, evaluate results, invert results, and send outputs (SO) [as shown in Fig. 4(c)] . We slightly optimize the state machine published in [50] by combining the receive inputs and SO states to a single SO state. Also for this adder, the memristors in the crossbar must be first initialized to a high resistance state during the start state. Note that the m-bit adder consists of $m/2$ 2-bit adders. Therefore, the above operations are repeated $m/2$ times (denoted by i). During each operation, control voltages are applied; they are write voltage (V_{w}), half-select

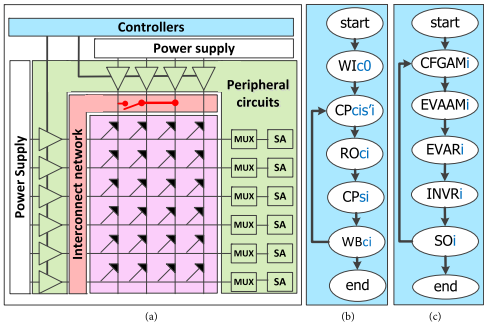


Fig. 4. CIM parallel adder. (a) Implementation. (b) CRS adder state machine. (c) FBLC adder state machine.

voltage (V_{hw}), and ground (GND). To communicate between two BRS cells, the controller performs a copy operation by applying appropriate voltages to the rows and columns of the crossbar [44]. The exact control voltage values are described in [47].

IV. CIM Parallel Adder Implementation

In this section, we describe the CIM parallel adder implementations based on CRS and FBLC adders. Fig. 4(a) shows its four main components: the memristor crossbar (the pink part), interconnect network (the red part), controller (the blue part), and peripheral circuit (the green part). These components are explained further in the following sections.

A. Memristor Crossbars

The first step to implement the memristor crossbar consists in replacing the adders in Fig. 2 with the CRS or FBLC memristor adders. However, the memristors in the crossbar might interfere with each other as all memristors in a row or column receive the same control voltage at each step. Therefore, we need to satisfy the following design requirements to guarantee functional correctness.

1) Per crossbar, only one row or column can be accessed at a time.
2) It is not allowed to control two memristors with different resistances that are located in the same row or column with the same voltage, as sneak path currents might impact the correctness.

Taking these restrictions into consideration, we propose a strategy to do the adder replacement. This strategy must ensure parallelism in each addition stage and consists of two steps. The first step places adders that execute in parallel in different subcrossbars. This resolves two issues: 1) the adders may operate independently without having conflicts in the control voltages and 2) it avoids sneak path currents. The second step aligns the adders within a crossbar. This optimizes both the controller and crossbar area. Note that the adders of the later stages require more bits than the adders of the first stage to prevent overflows. However, to make a perfect alignment, only m-bit wide adders are considered, regardless at which stage the adder is used. As a result of the replacement strategy, the adders in the same column of Fig. 2 will be placed in different subcrossbars. Each subcrossbar has its own controller and peripheral circuit. The above applies to both implementations. Next, more details of each implementation are given.

1) CRS Implementation: The pink area of Fig. 5(a) shows the crossbar of the CRS parallel adder implementation. As the read operation is destructive, the inputs (e.g., $n1$–$n16$ in Fig. 2) can be either stored in the crossbar using additional memristors or in the CMOS layer using registers. For simplicity, we assume that the inputs are stored in registers in the CMOS layer; hence, the inputs are not shown in Fig. 5(a). The crossbar consists only of CRS adders organized in subcrossbars; each subcrossbar corresponds to a row in Fig. 2.

Only one adder can be active at a time in each subcrossbar. However, to optimize for area, we have placed in Fig. 5(a) the adders of the first addition stage (which includes the first and the last column of Fig. 2) in the first and last rows of each subcrossbar; therefore, the first stage requires two computational stages due to sequential execution. However, as a result, a much better area efficiency is obtained.

In total, $N/4$ subcrossbars are required each with four adders. One subcrossbar will have only three adders and thus an empty spot. Therefore, using this placement, a total area consisting of $N/4$ (subcrossbars) $\cdot$ 4 (adders per subcrossbars) $= N$ adders is required. Note that the binary implementation of Fig. 2 requires $N - 1$ adders, and therefore a very efficient implementation is obtained.

2) FBLC Implementation: The pink area of Fig. 5(b) shows the crossbar of the FBLC parallel adder implementation. To ensure parallelism, the same adder placement topology has been used as for the CRS implementation. All FBLC adders are represented by multiple rectangles; each presents a 2-bit FBLC adder with inputs represented by the long dark bars and outputs by the dark squares. The b inside each 2-bit adder represents the bit index. For example, b 3–4 represents the second 2-bit adder that has bit indexes 3 and 4 of the input data and the carry out of the adder $b1 - 2$ as inputs. As the control voltages for the FBLC adder are independent on the data, the controller can be shared among the subcrossbars. Also in this implementation, adders of the first and last column (see Fig. 2) are placed in the same crossbar. Theoretically, as the control voltages are independent of the data, both adders of the first and last column can execute simultaneously even if they are located in the same crossbar. However, they facilitate sneak paths currents, and therefore, we assume sequential execution.

B. Interconnect Network and Communication Scheme

The interconnect network is strictly speaking part of the peripheral circuit and is discussed here separately due to its importance. It is used to enable the data communication between two memristor adders efficiently. Note that the interconnect network is not shown in Fig. 5(a) and (b) in order to keep the figures simple. The communication in the CRS implementation is realized by the read and write control operations; hence, it requires no extra interconnect network.

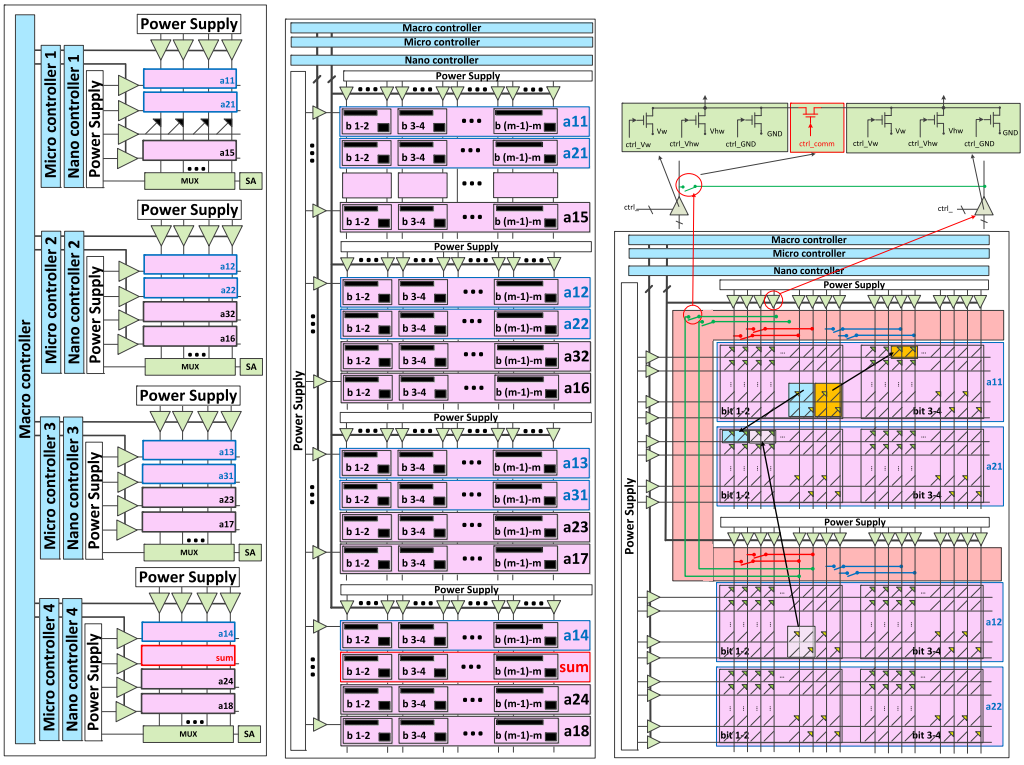


Fig. 5. Detailed implementations. (a) CRS parallel adder. (b) FBLC parallel adder. (c) FBLC interconnect network.

The FBLC implementation communicates through memristor copy operations. However, as a single copy operation can only be performed horizontally or vertically in the crossbar [44], we propose to add an interconnect network in the CMOS layer. It reduces the copying time between memristors that are not located on the same row or column memristors that are located in different subcrossbars; the interconnect network uses transistor switches to connect/disconnect two nanowires and thus is able to create a direct path between any two memristors in the crossbar. Further details are explained next.

1) CRS Implementation: In the CRS parallel adder implementation, both the logic 0 and logic 1 are represented by cells with an equivalent high resistance [50]; hence, copying the memristor resistance cannot be used for communication. To move data between two cells, the controller first has to read the value from the source cell (RO operation), and then forward this value to the destination cell (WI operation). As the read-out operation destroys the value stored in the source cell [50], the value is also rewritten into the source cell (write-back operation) in parallel with the write-in. Therefore, the communication needs two steps. Depending on the bandwidth, multiple communications can be performed in parallel. In this paper, we assume no limitations on the bandwidth. In case the communication takes place between subcrossbars (each with its own controller), an additional interconnect network infrastructure is added between the controllers. This part of the interconnect network will not be discussed in this paper due to space limitations.

2) FBLC Implementation: The FBLC parallel adder implementation uses BRS cells in which logic 0 and 1 are represented as high and low resistances, respectively. Hence, we can directly transfer a logic value between two cells using a copy operation. To implement the communication efficiently, we use a transistor switch [51] to create a direct path between two memristors that are not in the same row/column. Thereafter, a copy operation can be used to transfer a data bit in a single time step. An example of the solution is shown in Fig. 5(c). Communication between two bits within an adder (carry propagation) is represented by the blue lines (e.g., the carry result of the first 2-bit adder is copied to the input of the second 2-bit adder that is represented by the orange blocks). The communication between two adders located in the same and different subcrossbars is represented by red and green lines, respectively. Each communication wire can be implemented by a pass gate or pass transistor without creating conflicts with other wires as shown in the top part of Fig. 5(c). For example,

the transistor controlled by $\text{ctrl}_{\text{comm}}$ is used to create a direct path between any two nanowires. On the left and right sides of this transistor reside the voltage drivers of the nanowires that are connected by the switch. The voltage drivers are part of the peripheral circuit, which is described next.

C. Peripheral Circuit

The peripheral circuits [the green parts in Fig. 5(a) and (b)] include the interconnect network, voltage drivers, multiplexers, and sense amplifiers and are used to gain access to the crossbar. As the peripheral circuits are placed in the CMOS layer, it is crucial to balance the area of the peripheral circuits and the memristor crossbar that is stacked on top of it. In Section VI, we will present the area analysis used for both components. The voltage drivers form the interface between the controllers and memristor crossbar; each memristor row/column requires one. The size of a voltage driver, which impacts its drive strength, depends on the memristor load. An example of a possible implementation of a voltage driver is shown in the top part of Fig. 5(c) for the FBLC parallel adder. It consists of three voltage sources (V_w, V_{gnd}, and V_{hw}) and three pass transistors; each transistor is used to select the desired control voltage (e.g., ctrl_V_{w} is used to activate V_w). The remaining components depend on implementation.

1) CRS Implementation: In the CRS parallel adder implementation, a single sense-amplifier is connected to each subcrossbar [see Fig. 5(a)]. In this paper, we use the sense amplifier published in [52]. Only a single sense amplifier is needed, as each adder operate sequentially bit by bit. This requires, however, multiplexers to determine which bit of the word is read out. The multiplexers can be implemented with pass transistors; each vertical nanowire will be connected to such a pass transistor and the controller decides which nanowire (or bit) will be connected to the sense amplifier.

2) FBLC Implementation: As shown in Fig. 5(b), the FBLC parallel adder implementation does not require sense amplifiers as the resistively stored data do not have to be transferred from the memristor crossbar to the controller. A sense amplifier may be required to read the final sum from the crossbar. However, the area of this sense amplifier is negligible. As the FBLC implementation requires more rows and columns than the CRS implementation, it needs more voltage drivers and voltage drivers with larger driving strength.

D. Controllers

The controller controls the memristor crossbar through the interconnect network and peripheral circuits [blue part in Fig. 5(a) and (b)]. The controller includes three levels: macro, micro, and nano. Each level consists of a state machine. The macrocontroller determines which addition stage (of the $\log_2(n)$ stages) is being executed and therefore which adders are active over time; this controller is the same for both adder implementations. The microcontroller executes the operations required to complete a single bit addition as shown in Fig. 4. The microinstructions are subsequently translated into nanoinstructions; the nanoinstructions consist of the control voltages.

1) CRS Implementation: The microcontroller of the CRS parallel adder implementation consists of a state machine that requires five states to complete the addition of the first bit and four states for each successive bit [as shown in Fig. 4(a)]. As the controller depends on the data input and as it embeds communication, a separate controller is required for each subcrossbar to ensure their concurrent operations. In addition, buffers are required to store data during communication (after the read-out operations). Hence, its controller is more complex in comparison with the controller of FBLC implementation.

2) FBLC Implementation: The microcontroller of FBLC parallel adder implementation is a state machine that consists of five states for each 2-bit addition [as shown in Fig. 4(b)]. As the controller provides the same control voltages for all the active adders, one controller is shared among all subcrossbars; this significantly reduces the controller's energy and area.

V. Evaluation Models and Metrics

This section first presents the evaluation models of the two parallel adder implementations. Thereafter, it shows the model of the multicore-based parallel adder implementation. Finally, it discusses the evaluation metrics.

A. Memristor Implementation Models

For evaluation, we build separate models for each implementation. The models consist of the four main components: the memristor crossbar, interconnect network, peripheral circuit (including voltage drivers, sense amplifiers, and multiplexers), and controller. The models are based on the analysis mentioned in Section IV with the following assumptions.

1) There are unlimited resources available, i.e., no resource sharing is applied. Note that the number of required adders increases with the problem size.
2) There are no costs attributed to the initialization of the inputs as this cost is negligible compared with the total computational time. The inputs are assumed to be stored in registers residing in the CMOS layer for the CRS parallel adder implementation and in the memristor crossbar for FBLC implementation.
3) The cost (i.e., energy, delay, and area) to move data between the four main components is ignored. However, the communication within the main components is considered. This includes, for example, the communication between adders located in different subcrossbars.

The parameters of both models are explained next. Table III contains the delay, area, and energy cost of the basic units of the main components. The top part of Table III shows the general device technology parameters that are used for the memristor crossbar and CMOS layer, while the bottom part of Table III shows the cost of the basic units such as the controller, voltage drivers, and so on. The crossbar is assumed to be implemented with a memristor feature size equal to $F_m = 5$ nm [53]. For this feature size, the memristor write delay equals $D_m = 200$ ps, the energy consumption per write operation $E_m = 1 fJ$, and the area $A_m = 4 \cdot F_m^2 = 100$ nm^2 [53]. The CMOS part is based on $F_c = 40$-nm technology. The metrics are described next.

TABLE III
BASIC MODEL PARAMETERS

Component	Basic Unit	Parameter	CRS	FBLC
Device Technology				
Memristor Technology		F_m - Feature size (nm)	5 [54]	
		D_m - Delay (ps)	200 [54]	
		E_m - Energy (fJ)	1 [54]	
		A_m - Area (nm^2)	100 ($4F_m^2$) [54]	
CMOS Technology		F_c - Feature size (nm)	40	
Parallel Adder				
Peripheral Circuit	Voltage Driver [55–57]	D_{vd} - Delay (ps)	5.6 (D_{pt} = 1 pass transistor delay)	
		E_{vd} - Energy (fJ)	25.2 (E_{pt} = 1 pass transistor energy	
		A_{vd} - Area (um^2)	$3 \cdot A_{pt}$ (A_{pt} = 1 pass transistor area = $2 \cdot F_c^2$)	
	Sense Amplifier [53]	D_{sa} - Delay (ps)	28 [53]	not applicable
		E_{sa} - Energy (fJ)	8.5 [53]	
		A_{sa} - Area (um^2)	$13 \cdot F_c^2$ [53]	
	Multiplexer [58]	D_{mux} - Delay (ps)	D_{pt}	
		E_{mux} - Energy (pJ)	E_{pt}	
		A_{mux} - Area (um^2)	$(m+2) \cdot A_{pt}$	
Controller*		D_{cont} - Delay (ps)	675	825
		E_{cont} - Energy (pJ)	10.7	21.8
		A_{cont} - Area (um^2)	0.0074	0.0011
Interconnect network	CMOS part	D_{icc} - Delay	covered by controllers	masked by voltage driver
		E_{icc} - Energy (pJ)		E_{pt}
		A_{icc} - Area (um^2)		A_{pt}
	Crossbar part	D_{icx} - Delay (ps)	400 (2 cycles $\cdot$ D_m)	200 (1 cycle $\cdot$ D_m)
		E_{icx} - Energy (fJ)	3 (3 memristors $\cdot$ E_m)	1 (1 memristors $\cdot$ E_m)
		A_{icx} - Area (um^2)	0	0
Crossbar	Adder	D_{add} - Delay (ns)	26.6 ($N_{as}\cdot$ D_m with N_{as} = 133 steps)	16.2 ($N_{as}\cdot$ D_m with N_{as} = 81 steps)
		E_{add} - Energy (fJ)	226.4	$100.5 \cdot m/2$ [48]
		A_{add} - Area (nm^2)	3400 (34 memristors $\cdot$ A_m)	$43200 \cdot m/2$ ($432 \cdot m/2$ memristors $\cdot$ A_m)

(*): Synthesized with RTL compiler with N=8, m=32

The three basic units of the peripheral circuit (i.e., the voltage driver, sense amplifier, and multiplexers) are discussed first. The voltage driver consists of the control voltage sources and three pass transistors [see Fig. 5(c)]. The task of the voltage driver is to gain access to the crossbar by either performing read or write operations. During such operations, only one of the three pass transistors is active at a time. Therefore, the delay and energy consumption of the voltage driver are equal to those of a single pass transistor, while the area equals the area of three pass transistors. In this paper, we do not consider the exact size of the pass transistors, as detailed analysis is required to estimate this cost accurately. Nevertheless, our estimation (with a transistor width/length ratio of two) is sufficient to depict the trends. The delay and the energy consumption are obtained from HSPICE simulations using the PTM library [58]. The area of each pass transistor is assumed to be $2 \cdot F_c^2$.

The current sense amplifier is used only in the CRS parallel adder, and its delay and energy have been taken from [52] and scaled down to 40-nm technology node. The area is estimated by the total number of transistors.

The multiplexers can be realized by parallel pass transistors, similarly as they have been applied for column multiplexing in memories. The delay and energy consumption of the multiplexers equals the delay and energy consumption of a single pass transistor, as only one pass transistor is active at a time. The number of multiplexers that are required per subcrossbar equals $m + 2$, the number of vertical nanowires. Therefore, the total area of the multiplexers in a single crossbar equals $(m + 2)$ times the area of a single pass transistor (A_{pt}).

The next component in Table III is the controller. The controllers for both implementations are written in VHDL and synthesized with Cadence RC compiler using the TSMC 40-nm library [59]. The synthesis results (delay, energy (power x delay), and area) have been reported for the parallel adder with $N = 8$ number of inputs and data-width $m = 32$.

The next component in Table III is the interconnect network. The interconnect network has active parts both in the crossbar and CMOS layer; therefore, the delay, energy, and area are reported for both parts separately. The communication in the CRS parallel adder implementation is handled by the CMOS controller. Therefore, there are no costs attributed separately to the CMOS part. For the crossbar, costs are attributed only to the delay and energy as there is no dedicated interconnect network within the crossbar. A CRS adder requires two cycles to complete a single write operation (one for a read operation and one for the write-in/-back operations), where each cycle lasts D_m. We assume that the energy consumption equals $2 \cdot E_m$. This is based on the fact that at least one and possibly three memristors undergo a value change during a single bit transfer; the destructive read operation might alter the value of a single memristor, the write-in operation will update the value of the target memristor, and the write-back will restore the value of the source memristor if it got corrupted during the read-out. As the read-out and write-back operations are value dependent, we assume that two memristors will be written on average. For the FBLC implementation, the CMOS part of the interconnect network consists of pass transistors that connect/disconnect different nanowires. The delay of the pass transistor is masked by the delay of the voltage driver, as the controller activates the voltage driver and pass transistor simultaneously. The area and the energy of the CMOS part in the interconnect network consist of the cost attributed to a single pass transistor (E_{pt} and A_{pt}). With respect to the crossbar, communication (i.e., a single copy operation) takes one memristor write delay D_m. As only one target memristor

TABLE IV
SCALING MODEL PARAMETERS

Architecture parameters				
Parameters			CRS	FBLC
N_s - #Stages			$log_2 N + 1$	
N_x - #Sub-crossbars			N/4	
N_r - #Rows per sub-crossbar			4	$4 \cdot 27 = 108$
N_c - #Columns per sub-crossbar			$m + 2 = 34$	$16 \cdot m/2 = 8m$
Component parameters				
Component	Basic unit	Parameters	CRS	FBLC
Peripheral Circuit	Voltage Driver	Delay	$N_s \cdot N_{as} \cdot D_{vd}$	
		Energy	$N_s \cdot N_x \cdot (N_r + N_c) \cdot N_{as} \cdot E_{vd}$	
		Area	$(N_r + N_c) \cdot N_x \cdot A_{vd}$	
	Sense Amplifier	Delay	$N_s \cdot (m+1) \cdot D_{sa}$	not applicable
		Energy	$(N-1) \cdot (m+1) \cdot E_{sa}$	
		Area	$N_x \cdot A_{sa}$	
	Multiplexers	Delay	$N_s \cdot (m+1) \cdot D_{mux}$	
		Energy	$(N-1) \cdot (m+1) \cdot E_{mux}$	
		Area	$N_x \cdot A_{mux}$	
Controller		Delay	$N_s \cdot D_{cont}$	
		Energy	Synthesized by RTL	
		Area	Synthesized by RTL	
Interconnect Network	CMOS part	Delay	embedded in controllers	0
		Energy		$(N-2) \cdot m \cdot E_{icc}$
		Area		$(N-2) \cdot m \cdot A_{icc}$
	Crossbar part	Delay	$(N-2) \cdot D_{icx}$	embedded in SO state
		Energy	$(N-2) \cdot m \cdot E_{icx}$	$(N-2) \cdot m \cdot E_{icx}$
		Area	0	0
Crossbar		Delay	$N_s \cdot D_{add}$	
		Energy	$(N-1) \cdot E_{add}$	
		Area	$N \cdot A_{add}$	

will be written, the energy equals E_m. Note that there is no cost attributed to the area in the crossbar.

The final entry of Table III mentions the delay, energy, and area of a single crossbar adder with $m = 32$. The CRS adders requires 133 steps [45] to complete a single addition, while the FBLC adder requires 81 steps [47]. Note that each step takes a memristor write operation D_m. The energy required to complete a single addition is calculated by the total number of average switching memristors per addition. For the CRS adder, we created a MATLAB script to determine the average number of changing memristors for 32-bit additions based on 1000 random input numbers. For FBLC adder, we take these data from [47]. The area of the crossbar area equals $(m + 2) \cdot A_m = 34 \cdot 100 = 3400$ nm^2 for the CRS implementation, while 27 (rows) $\cdot$ 16 (columns) $\cdot A_m = 43\,200$ nm^2 per 2-bit FBLC adder [47]. The area for an m-bit FBLC adder is subsequently obtained by multiplying this number with $m/2$.

Table IV shows how the cost of each basic unit scales with the problem size N (i.e., the number of inputs). By combining the information presented in Tables III and IV, the total cost can be derived for the entire implementation. The top part of Table IV shows the general architecture parameters of the parallel adder implementations. The architecture is designed to complete the parallel addition in a total number execution stages (N_s) equal to $\log_2 N + 1$. Note that the first stage consumes two execution steps. The implementation consists of $N/4$ subcrossbars in which each subcrossbar contains four adders. Hence, the size of each subcrossbar is obtained by multiplying the number of rows of a single adder by four. This results in a size of 4 by $(m + 2)$ for CRS and 108 by 8 m for FBLC. The numbers of rows and columns of each subcrossbar are denoted by N_r and N_c, respectively. The bottom part of Table IV shows the detailed cost per basic unit. Each component contributes to the total delay, energy, and area. The total delay of each unit is calculated based on its number of execution steps (see Table III) and their occurrence frequency (see Table IV). A similar methodology can be applied for the area and energy.

First, the basic units of the peripheral circuit are discussed. Voltage drivers are required for each row and column of the crossbar. The total delay of the voltage driver equals the delay per voltage driver D_{vd} multiplied by the total number of execution steps $N_s \cdot N_{\text{as}}$. The total energy is the product of the total number of execution steps $N_s \cdot N_{\text{as}}$, the number of sub-crossbars N_x, the number of required voltage drivers per subcrossbar equal to $N_r + N_c$, and the energy consumption for each voltage driver E_{vd}. The total area equals the product of the total number of crossbars N_x, the number of rows and columns per crossbar $N_r + N_c$, and the area per voltage driver. The sense amplifiers and multiplexers are required only for the CRS implementation. One sense amplifier is required per subcrossbar, and it is activated only during (RO) operations; each addition requires $m+1$ RO operations. The total delay of the sense amplifiers equals the product of the number of stages N_s, the number of times a sense amplifier is active per addition equal to $m + 1$, and the delay to complete one sense operation D_{sa}. Similarly, the total energy consumption equals the product of the number of additions equal to $N - 1$, the number of times the sense amplifier is active per addition equal to $m + 1$, and the energy per single sense operation E_{sa}. The total required number

of sense amplifiers equals N_x; therefore, the area can be obtained by multiplying this number with the area per sense amplifier A_{sa}. The multiplexers are activated at the same time as the sense amplifier, as they determine which bit is read by the sense amplifier. Hence, the delay, energy, and area are calculated in a similar way as the sense amplifier. The only difference is the cost of each basic unit.

Next, the controller is discussed. The total controller delay depends on the number of execution steps $N_s \cdot N_{\text{as}}$ and the delay D_{cont} of the controller. The energy and area of the controller are taken from the RTL synthesis reports.

The interconnect network is discussed next. Each adder, except for the last one, transfers its m-bit result to the next adder leading to a total of $N-2$ word transfers. We assume that the m bits are transferred simultaneously. Hence, it requires one delay step that takes D_{icx} (for the crossbar) + D_{icc} (for the CMOS layer) time to transfer a single m-bit number. In an addition stage, each data transfer is assumed to be executed sequentially as conflicts might rise at the destination adders as two inputs are required for each addition. Hence, the number of communication steps equals $N-2$. With respect to the CMOS part, the delay, energy, and area of the interconnect network of the CRS implementation are embedded in the controller. For the FBLC implementation, the delay is masked by the voltage driver delay, and hence, the delay cost is zero. The energy equals the product of the number of communication steps $N-2$, the number of active pass transistors (one per bit), and the energy of each connection E_{icc}. The area equals the product of the number of communication steps $N-2$, the number of connections per word (one per bit), and the area required per connection A_{icc}. Next, we describe the cost of the crossbar part. For the CRS implementation, the delay equals the product of the number of word transfers equal to $N-2$ and the delay of each word transfer in crossbar D_{icx}. For the FBLC implementation, the communication delay is handled by the SO state of microcontroller. Therefore, the cost is embedded in the delay of the adder D_{add}. For both implementations, the energy equals the product of the number of communications steps $N-2$, the number of transferred bits (m), and the energy cost per bit transfer E_{icx}; note that E_{icx} is implementation dependent. For both implementations, no extra memristors are required for interconnect network; hence, the area equals zero.

Finally, the delay, energy, and area of the crossbar are obtained in the same way for both implementations. The delay is derived from the number of execution steps $N_s \cdot N_{\text{as}}$ and the delay per addition D_{add}. The energy is calculated from the number of additions $N-1$ and the energy consumption per addition E_{add}. Finally, the area is based on the number of subcrossbars equal to $N/4$, the number of adders per crossbar equal to 4, and the area A_{add} per adder.

B. Multicore Implementation Model

To compare the two memristor implementations with a CMOS implementation, we build a traditional multicore model with the same functionality. This model consists of clusters; each cluster includes 32 adders, 512-kB cache, and 1-GB DRAM memory; each is based on 32-nm technology. The assumptions used for memristor implementation model are also applied for multicore implementation model.

1) The number of required cluster increases with the problem size similarly as in [36].
2) We assume at startup a warm cache with an ideal cache coherency policy.
3) We consider only the cost to move data between the main components, i.e., among clusters, among adders, between adders and caches, and between caches and DRAM memory.

The parameters of each component are explained next. The adders are ripple-carry adders with the same delay, energy, and area as specified in [36] using 32-nm technology. The delay, energy, and area of the cache and DRAM memory were model using CACTI 5.3 [60] using 32-nm technology.

C. Metrics

To evaluate the models of the previous section, six metrics are used for each component: total delay (D), total energy (E), total area (A), energy delay efficiency (η_{ED}), energy efficiency (η_E), and area efficiency (η_A). The latter three metrics are derived from the former three and number of operations (#ops). Their equations are shown in the following:

$$\eta_{\text{ED}} = \frac{\#\text{ops}}{D \cdot E}; \quad \eta_E = \frac{\#\text{ops}}{E} \quad \eta_A = \frac{\#\text{ops}}{A}. \tag{1}$$

Although the energy and area of the components and basic units can be summed up to obtain the total energy consumption and area, this is not applicable for the total delay. Instead, the total delay is calculated from the cycle time (which depends on the critical path) and the total number of cycles (execution steps $N_s \cdot N_{\text{as}}$). For CRS implementation, the critical path includes the delay of the controller, voltage driver, memristor, sense amplifier, and multiplexer. For FBLC implementation, the critical path contains fewer basic units and includes only the delay of controller, voltage driver, and memristor. For multicore implementation, the total delay, energy, and area sum up that of the components.

VI. Results and Discussion

In this section, we first analyze the implementation feasibility of the CIM parallel adder and compare the results of the metrics for the two implementations. Thereafter, we compare them with the multicore-based implementation. Finally, we evaluate them and discuss their advantages and limitations.

A. CRS and FBLC Performance Results

Fig. 6 shows the six metrics for both implementations evaluated for different N. The cost includes all components. In Fig. 6, the graphs are normalized to the CRS implementation with $N = 8$. Fig. 6 shows the scaling trends of both implementations. The delay grows as expected proportionally to the logarithm of the number of inputs, thus confirming the scalability. The FBLC implementation has a slightly lower execution delay mainly due to the absence of multiplexers and sense amplifiers in its critical path. In terms of energy,

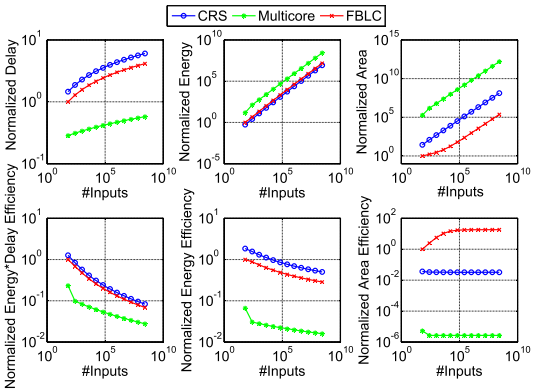


Fig. 6. Performance results of CRS, FBLC, and multicore implementation.

the FBLC implementation consumes nearly twice more energy than the CRS implementation due to the larger crossbar; the larger crossbar has a higher switching activity and consumes more power in the voltage drivers and in the pass transistors of interconnect network. Nevertheless, the area of the CRS implementation is larger than the FBLC implementation. In fact, the controller is responsible for this high area consumption. We will show the details in the component analysis later.

In terms of the derived metrics, the CRS implementation achieves a better EDP mainly due to its lower energy consumption. In terms of energy efficiency, the CRS implementation performs better due to its lower energy consumption. In terms of area efficiency, the FBLC implementation performs better due to its lower area. Note that the energy consumption and area of both implementations depend on the area of the controller. This explains the outliers at smaller input sizes. Overall, we observe that the area efficiency of both implementations becomes stable at the large input sizes. This shows that the scalability of both implementations is feasible.

Fig. 7(a) and (b) shows the cycle time and area breakdown of both implementations at the component granularity (i.e., at the level of memristor crossbar, interconnect network, controllers, and peripheral circuits). For both implementations, the cycle time is mostly determined by the delay of the controller followed by the memristor crossbar. For the CRS implementation, the peripheral circuit also contributes to the delay, while this is not the case for the FBLC implementation. From this cost-break down, we observe that both implementations are scalable as: 1) the relative percentage of each component changes minimally when the input size increases; this denotes that there is no critical component for larger n and 2) the absolute value (which is not shown in Fig. 7) of the cycle time increases marginally for larger N.

With respect to the area consumption, we observe for the CRS implementation that the largest fraction of the area comes from the controller [see Fig. 7(a)]. This can be explained by the fact that in CRS, the memristor crossbar and the peripheral circuits are relatively small due to the usage of implication logic. However, the area penalty is paid mostly in the controller as its size linearly increases with the input size. Note that every subcrossbar needs its own nanocontroller that is proportional to the input size. Therefore, the controller always consumes the largest fraction of the total area.

The largest fraction of the area also comes from the controller for the FBLC implementation, for small values of N. Furthermore, it reduces for increasing N [see Fig. 7(b)]. The relative contribution of each component stabilizes for the FBLC implementation. The largest part of the area comes from the crossbar and peripheral circuit. The above behavior can be explained by the fact that both the crossbar and peripheral circuit increase with larger size, while the size of the controller is nearly constant as it is shared between subcrossbars. Note that only one additional state is required in the macrocontroller when the input size doubles; this area is negligible. The area of the interconnect network increases marginally and its fraction remains stable for larger N.

Overall, FBLC implementation achieves a better scalability at larger input sizes even though it requires a larger crossbar. This is mainly due to its efficient controller. As the memristor layer is stacked on the CMOS layer, the FBLC implementation may be a candidate for actual implementation.

B. Comparison With Multicore-Based Parallel Adder

Fig. 6 also shows the comparison between the two memristor implementations and the multicore-based implementation. In terms of delay, the multicore implementation achieves nearly one order of magnitude better performance than CRS and FBLC implementation. This can be explained by the fact that the delay of a memristor adder is much slower than a CMOS adder. In addition, we ignore the impact of the cache coherence policy leading to optimistic results. Hence, the advantage in delay performance is explainable. In terms of energy, the multicore implementation loses by two orders of magnitude in comparison with both CRS and FBLC implementation. The reason behind this is mainly from the cache and DRAM energy consumption. In terms of area, the multicore implementation has nearly five orders of magnitude larger area than the other two implementations. This is again caused by the large area consumption of the caches and DRAM. In terms of the three combined metrics, the memristor implementations have an improvement of at least two orders of magnitude for each of the combined metrics.

C. Discussion and Limitations

The two memristor implementations show different results for the measured metrics, and therefore tradeoffs can be made. The CRS implementation requires a larger control state machine, needs more buffers to store the intermediate results, and requires current sense amplifiers. The FBLC implementation has relatively a larger crossbar and requires less CMOS support. In addition, FBLC is based on Boolean logic that may take advantage of today's existing synthesis tools. In this paper, we investigated only the impact of two logic designs, i.e., Boolean and implication logic. However, note that there are many more memristor designs that can be used to perform computation in the crossbar, as summarized in [61].

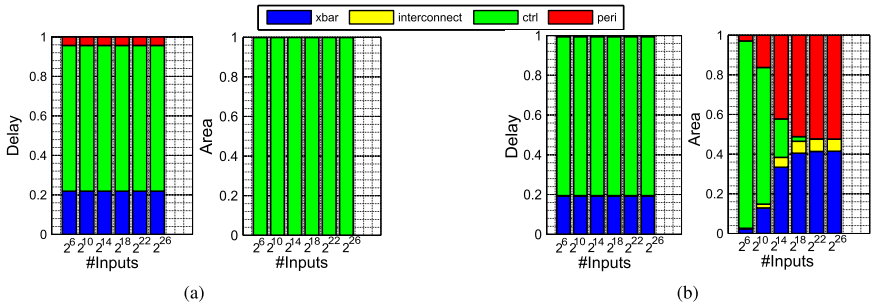


Fig. 7. Delay and area analysis of memristor implementation's components. (a) CRS implementation. (b) FBLC implementation.

The CIM architecture has the potential to deal with big data problems in general as both implementations show a good scalability with respect to the delay. Although we used only the parallel adder as a case study, the scaling trends are also valid for a wide range of applications. The reason for this is that the CRS parallel adder implementation already utilizes unique controller for each subcrossbar. Therefore, it will not affect the performance of other designs significantly in case different types of controllers are required. For the FBLC implementation, the microstate machine can be used to implement any look-up table, as the state machine is independent of the data and the functionality. Therefore, the controller can also be shared among the subcrossbars for more complex functions.

Despite their huge potential as alternative computing devices, memristors are still in their infancy stage and are facing many challenges. First, the integration of memristos with CMOS is still an open research question. To the best of our knowledge, this is the first paper that investigates the impact of the CMOS controller on an entire memristor design. The controllers were designed using TSMC 40-nm technology. Based on this preliminary work, the CMOS part might be a concern. Further investigation requires not only the optimization of the CMOS part but also its efficient use to control the crossbars (e.g., shared by different stacked crossbars). Note that in our work we assumed 40-nm CMOS technology (as it is the smallest node we have access to) and 5-nm memristor feature size; it seems that this integration is possible as presented in [62] where demonstrated prototypes showed the stacking of 100-nm memristors on top of 500-nm CMOS. In addition, Cheng and Strukov [63] also believe that stacking with different feature sizes is possible.

Second, memristor technology suffers from limited endurance, today typically around (10^{12}) [64]. This is insufficient for general purpose computing. Nevertheless, this could still be sufficient enough for specific applications such as those that are mostly in power-off state (e.g., wearable or monitoring devices) [65], [66]. In addition, researchers strongly believe that the endurance will reach 10^{16} [65].

Third, more research is required to isolate nanowires in the crossbar; isolation is needed to create multiple subcrossbars. One possible implementation is to use isolation material between them, similarly as proposed in [67].

Finally, there is a lack of libraries with well-optimized memristor designs. Currently, only a limited number of computational units (such as adder and multiplier) and communication schemes have been proposed. For example, with respect to this paper, the area of the m-bit FBLC adder can be significantly reduced if only one single 2-bit adder is used $m/2$ times sequentially over time. As a direct consequence, the number of required voltage drivers also reduces. In addition, other optimizations can be made, such as in the controller. By pipelining the controller, the clock frequency can be significantly reduced, leading to much lower execution times, especially for the FBLC implementation. The FBLC implementation does not require read-out operations, and therefore, a pipelined controller can be efficiently executed. Finally, in the current state of the art, no efficient solutions have been presented for the communication between basic units. Therefore, we used a simplistic evaluation for the communication between the memristor crossbar and CMOS layer. Research on efficient and appropriate communication and routing scheme between the CMOS and crossbar layers is still on-going.

VII. CONCLUSION

In this paper, we showed two memristor crossbar implementations of CIM parallel adder. CIM alleviates the memory bottleneck by utilizing the memristor devices in the crossbar for both computation and storage. We analyzed for both implementations (one based on Boolean logic and the other on implication logic) their impact on the total cost. This cost includes the delay, energy, and area of the crossbar, interconnect network, peripheral circuit, and CMOS controller. Our results show that the implementation of both designs is feasible. In fact, a tradeoff can be made between the performance and resource usage. Finally, the results also show that CIM architecture efficiently deals with large-scale problems. In comparison, the memristor implementation outperforms a similar multicore implementation by at least two orders of magnitude in terms of energy delay efficiency, energy efficiency, and area efficiency. In short, CIM is a viable candidate for next generation computers.

REFERENCES

[1] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *Proc. 38th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2011, pp. 365–376.

[2] S. Kaxiras, *Architecture at the End of Moore* (Advances in Atom and Single Molecule Machines). Berlin, Germany: Springer-Verlag, 2013.

[3] Y. Taur, "CMOS design near the limit of scaling," *IBM J. Res. Develop.*, vol. 46, nos. 2–3, pp. 213–222, Mar. 2002.

[4] L. Rocks and R. P. Runyon, *Chapter 20: The Energy Crisis* (The Frontiers Collection). Berlin, Germany: Springer-Verlag, 2012.

[5] J. W. McPherson, "Reliability trends with advanced CMOS scaling and the implications for design," in *Proc. IEEE Custom Integr. Circuits Conf. (CICC)*, Sep. 2007, pp. 405–412.

[6] S. Borkar, "Design perspectives on 22 nm CMOS and beyond," in *Proc. Design Autom. Conf. (DAC)*, Jul. 2009, pp. 93–94.

[7] G. Gielen *et al.*, "Emerging yield and reliability challenges in nanometer CMOS technologies," in *Proc. Conf. Design, Autom. Test Eur. (DATE)*, Mar. 2008, pp. 1322–1327.

[8] J. W. Janneck, "Computing in the age of parallelism: Challenges and opportunities," Computer Science Department, Lund University, in Multicore Day, 23 Sep 2013.

[9] K. Lahiri and A. Raghunathan, "Power analysis of system-level on-chip communication architectures," in *Proc. Int. Conf. Hardw./Softw. Codesign Syst. Synth. (CODES+ISSS)*, Sep. 2004, pp. 236–241.

[10] S. A. McKee, "Reflections on the memory wall," in *Proc. Comput. Front.*, 2004, pp. 1–6.

[11] A. W. Burks, H. H. Goldstine, and J. von Neumann, "Preliminary discussion of the logical design of an electronic computing instrument (1946)," in *Perspectives on the Computer Revolution*, Z. W. Pylyshyn and L. J. Bannon, Eds. Norwood, NJ, USA: Ablex Publishing Corp., 1989, pp. 39–48. [Online]. Available: http://dl.acm.org/citation.cfm?id=98326.98337

[12] W. Xue *et al.*, "Enabling and scaling a global shallow-water atmospheric model on Tianhe-2," in *Proc. IEEE 28th Int. Parallel Distrib. Process. Symp.*, May 2014, pp. 745–754.

[13] X. Zhang *et al.*, "Optimizing and scaling HPCG on tianhe-2: Early experience," in *Proc. 14th Int. Conf. Algorithms Archit. Parallel Process. (ICA3PP)*, Dalian, China, Aug. 2014, pp. 28–41.

[14] G. Bell and J. Gray, "What's next in high-performance computing?" *Commun. ACM*, vol. 45, no. 2, pp. 91–95, 2002.

[15] D. G. Elliott, M. Stumm, W. M. Snelgrove, C. Cojocaru, and R. Mckenzie, "Computational RAM: Implementing processors in memory," *IEEE Des. Test Comput.*, vol. 16, no. 1, pp. 32–41, Jan./Mar. 1999.

[16] J. Draper *et al.*, "A prototype processing-in-memory (PIM) chip for the data-intensive architecture (DIVA) system," *J. VLSI Signal Process. Syst. signal, Image Video Technol.*, vol. 40, no. 1, pp. 73–84, 2005.

[17] B. J. Jasionowski, M. K. Lay, and M. Margala, "A processor-in-memory architecture for multimedia compression," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 15, no. 4, pp. 478–483, Apr. 2007.

[18] E. Upchurch, T. Sterling, and J. B. Brockman, "Analysis and modeling of advanced PIM architecture design tradeoffs," in *Proc. Innov. Archit. Future Generat. High-Perform. Process. Syst.*, Jul. 2003, pp. 66–75.

[19] P. M. Kogge, "EXECUBE—A new architecture for scaleable MPPs," in *Proc. Int. Conf. Parallel Process. (ICPP)*, vol. 1. Aug. 1994, pp. 77–84.

[20] D. Patterson *et al.*, "A case for intelligent RAM," *IEEE Micro*, vol. 17, no. 2, pp. 34–44, Mar./Apr. 1997.

[21] J. Torrellas, "FlexRAM: Toward an advanced intelligent memory system: A retrospective paper," in *Proc. IEEE Int. Conf. Comput. Design (ICCD)*, Sep./Oct. 2012, pp. 3–4.

[22] J. Draper *et al.*, "The architecture of the DIVA processing-in-memory chip," in *Proc. Int. Conf. Supercomput.*, 2002, pp. 1–12.

[23] T. L. Sterling and H. P. Zima, "Gilgamesh: A multithreaded processor-in-memory architecture for petaflops computing," in *Proc. ACM/IEEE Conf. Supercomput.*, Nov. 2002, p. 48.

[24] D. Keitel-Schulz and N. Wehn, "Issues in embedded DRAM development and applications," in *Proc. 11th Int. Symp. Syst. Synth.*, Dec. 1998, pp. 23–28.

[25] D. Keitel-Schulz and N. Wehn, "Embedded DRAM development: Technology, physical design, and application issues," *IEEE Design Test Comput.*, vol. 18, no. 3, pp. 7–15, May 2001.

[26] A. Farmahini-Farahani, J. H. Ahn, K. Morrow, and N. S. Kim, "NDA: Near-DRAM acceleration architecture leveraging commodity DRAM devices and standard memory modules," in *Proc. IEEE 21st Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2015, pp. 283–295.

[27] S. H. Pugsley *et al.*, "NDC: Analyzing the impact of 3D-stacked memory+logic devices on MapReduce workloads," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, Mar. 2014, pp. 190–200.

[28] R. Balasubramonian *et al.*, "Near-data processing: Insights from a MICRO-46 workshop," *IEEE Micro*, vol. 34, no. 4, pp. 36–42, Jul./Aug. 2014.

[29] A. Shafiee *et al.*, "ISAAC: A convolutional neural network accelerator with *in-situ* analog arithmetic in crossbars," in *Proc. ISCA*, Jun. 2016, pp. 14–26.

[30] M. V. Wilkes, "The memory wall and the CMOS end-point," *ACM SIGARCH Comput. Archit. News*, vol. 23, no. 4, pp. 4–6, Sep. 1995.

[31] P. Somavat, S. Jadhav, and V. Namboodiri, "Accounting for the energy consumption of personal computing including portable devices," in *Proc. 1st Int. Conf. Energy-Efficient Comput. Netw.*, 2010, pp. 141–149.

[32] J. Suh, E.- G. Kim, S. P. Crago, L. Srinivasan, and M. C. French, "A performance analysis of pim, stream processing, and tiled processing on memory-intensive signal processing kernels," *ACM SIGARCH Comput. Archit. News*, vol. 31, no. 2, pp. 410–421, 2003.

[33] P.-E. Gaillardon *et al.*, "The programmable logic-in-memory (PLiM) computer," in *Proc. Design, Autom. Test Eur. Conf. Exhibit. (DATE)*, Mar. 2016, pp. 427–432.

[34] A. Morad, L. Yavits, S. Kvatinsky, and R. Ginosar, "Resistive GP-SIMD processing-in-memory," *ACM Trans. Archit. Code Optim.*, vol. 12, no. 4, p. 57, 2016.

[35] S. Hamdioui *et al.*, "Memristor based computation-in-memory architecture for data-intensive applications," in *Proc. Design, Autom. Test Eur. (DATE)*, Mar. 2015, pp. 1718–1725.

[36] H. A. D. Nguyen, L. Xie, M. Taouil, R. Nane, S. Hamdioui, and K. Bertels, "Computation-in-memory based parallel adder," in *Proc. IEEE/ACM Int. Symp. Nanosc. Archit. (NANOARCH)*, Jul. 2015, pp. 57–62.

[37] G. Snider, "Computing with hysteretic resistor crossbars," *Appl. Phys. A, Solids Surf.*, vol. 80, no. 6, pp. 1165–1172, Mar. 2005.

[38] Y. V. Pershin and M. D. Ventra, "Memcomputing: A computing paradigm to store and process information on the same physical platform," in *Proc. Int. Workshop Comput. Electron. (IWCE)*, Jun. 2014, pp. 1–2.

[39] A. Haron, J. Yu, R. Nane, M. Taouil, S. Hamdioui, and K. Bertels, "Parallel matrix multiplication on memristor-based computation-in-memory architecture," in *Proc. Int. Conf. High Perform. Comput. Simulation (HPCS)*, Jul. 2016, pp. 759–766.

[40] E. Lehtonen and M. Laiho, "Stateful implication logic with memristors," in *Proc. IEEE/ACM Int. Symp. Nanosc. Archit. (NANOARCH)*, Jul. 2009, pp. 33–36.

[41] J. Borghetti, G. S. Snider, P. J. Kuekes, J. J. Yang, D. R. Stewart, and R. S. Williams, "'Memristive' switches enable 'stateful' logic operations via material implication," *Nature*, vol. 464, pp. 873–876, Apr. 2010, doi: 10.1038/nature08940.

[42] R. Bruchhaus *et al.*, "Bipolar resistive switching in oxides: Mechanisms and scaling," *Current Appl. Phys.*, vol. 11, no. 2, pp. e75–e78, Mar. 2011.

[43] E. Linn, R. Rosezin, C. Kügeler, and R. Waser, "Complementary resistive switches for passive nanocrossbar memories," *Nature Mater.*, vol. 9, pp. 403–406, Apr. 2010, doi: 10.1038/nmat2748.

[44] L. Xie, H. A. D. Nguyen, M. Taouil, S. Hamdioui, and K. Bertels, "Interconnect networks for memristor crossbar," in *Proc. IEEE/ACM Int. Symp. Nanosci. Archit. (NANOARCH)*, Jul. 2015, pp. 124–129.

[45] A. Siemon, S. Menzel, R. Waser, and E. Linn, "A complementary resistive switch-based crossbar array adder," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 5, no. 1, pp. 64–74, Mar. 2015.

[46] S. Kvatinsky, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Memristor-based material implication (IMPLY) logic: Design principles and methodologies," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 10, pp. 2054–2066, Oct. 2014.

[47] L. Xie, H. A. D. Nguyen, M. Taouil, S. Hamdioui, and K. Bertels, "Fast Boolean logic mapped on memristor crossbar," in *Proc. IEEE Int. Conf. Comput. Design (ICCD)*, Oct. 2015, pp. 335–342.

[48] D. Niu, Y. Chen, C. Xu, and Y. Xie, "Impact of process variations on emerging memristor," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, Jun. 2010, pp. 877–882.

[49] E. Linn, R. Rosezin, S. Tappertzhofen, U. Böttger, and R. Waser, "Beyond von Neumann—Logic operations in passive crossbar arrays alongside memory operations," *Nanotechnology*, vol. 23, no. 30, p. 305205, 2012.

[50] R. Rosezin, E. Linn, L. Nielen, C. Kugeler, R. Bruchhaus, and R. Waser, "Integrated complementary resistive switches for passive high-density nanocrossbar arrays," *IEEE Electron Device Lett.*, vol. 32, no. 2, pp. 191–193, Feb. 2011.

[51] R. Zimmermann and W. Fichtner, "Low-power logic styles: CMOS versus pass-transistor logic," *IEEE J. Solid-State Circuits*, vol. 32, no. 7, pp. 1079–1090, Jul. 997.

[52] S. Sundaram, P. Elakkumanan, and R. Sridhar, "High speed robust current sense amplifier for nanoscale memories: A winner take all approach," in *Proc. Int. Conf. VLSI Design Held Jointly 5th Int. Conf. Embedded Syst. Design (VLSID)*, Jan. 2006, pp. 1–6.
[53] ITRS, "The international technology roadmap for semiconductors ITRS," Semicond. Ind. Assoc., Washington, DC, USA, Tech. Rep., 2011.
[54] X. Tang, P.-E. Gaillardon, and G. D. Micheli, "A high-performance low-power near-Vt RRAM-based FPGA," in *Proc. Int. Conf. Field-Program. Technol. (FPT)*, Dec. 2014, pp. 207–214.
[55] P. Gaillardon *et al.*, "Design and architectural assessment of 3-D resistive memory technologies in FPGAs," *IEEE Trans. Nanotechnol.*, vol. 12, no. 1, pp. 40–50, Jan. 2013.
[56] W. Zhao *et al.*, "Design and analysis of crossbar architecture based on complementary resistive switching non-volatile memory cells," *J. Parallel Distrib. Comput.*, vol. 74, no. 6, pp. 2484–2496, Jun. 2014.
[57] SAED, "Digital standard cell library SAED_EDK90_CORE databook," Synopsys, Mountain View, CA, USA, Tech. Rep., 2008.
[58] NIMO, "Predictive technology model," Arizona State Univ., Phoenix, AZ, USA, Tech. Rep., 2012.
[59] TSMC, "TSMC standard cell databook," Taiwan Semicond. Manuf. Company Ltd., Hsinchu, Taiwan, Tech. Rep., 2008.
[60] S. Thoziyoor *et al.*, "Cacti 5.3," HP Lab., Palo Alto, CA, USA, Tech. Rep., 2008.
[61] L. Xie, H. A. D. Nguyen, M. Taouil, S. Hamdioui, and K. Bertels, "Boolean logic gate exploration for memristor crossbar," in *Proc. IEEE Int. Conf. Design Technol. Integr. Syst. Nanosc. Era (DTIS)*, Apr. 2016, pp. 1–6.
[62] Q. Xia *et al.*, "Memristor-CMOS hybrid integrated circuits for reconfigurable logic," *Nano Lett.*, vol. 9, no. 10, pp. 3640–3645, 2009.
[63] K.-T. T. Cheng and D. B. Strukov, "3D CMOS-memristor hybrid circuits: Devices, integration, architecture, and applications," in *Proc. ACM Int. Symp. Int. Symp. Phys. Design*, 2012, pp. 33–40.
[64] J. J. Yang, D. B. Strukov, and D. R. Stewart, "Memristive devices for computing," *Nature Nanotechnol.*, vol. 8, no. 1, pp. 13–24, 2013.
[65] M.-F. Chang *et al.*, "Endurance-aware circuit designs of nonvolatile logic and nonvolatile SRAM using resistive memory (memristor) device," in *Proc. 17th Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan./Feb. 2012, pp. 329–334.
[66] S. Hamdioui *et al.*, "Memristor for computing: Myth or reality?" in *Proc. Design, Autom. Test Eur. (DATE)*, 2017.
[67] C. Yakopcic, T. M. Taha, and R. Hasan, "Hybrid crossbar architecture for a memristor based memory," in *Proc. IEEE Nat. Aerosp. Electron. Conf. (NAECON)*, Jun. 2014, pp. 237–242.

Hoang Anh Du Nguyen (S'15) received the M.Sc. degree in computer engineering from the Delft University of Technology, Delft, The Netherlands, where she is currently working toward the Ph.D. degree at the Computer Engineering Laboratory.

Her current research interests include computation-in-memory architecture for big-data, memristor based systems, and synthesis automation.

Lei Xie (S'15) received the B.S. and M.Sc. degrees in microelectronics from Xi'an Jiaotong University, Xi'an, China. He is currently pursuing the Ph.D. degree with the Computer Engineering Laboratory, Delft University of Technology, Delft, The Netherlands.

His research interests include memristor-based logic circuit design.

Mottaqiallah Taouil (S'10–M'15) received the M.Sc. and Ph.D. degrees (with Hons.) in computer engineering from the Delft University of Technology, Delft, The Netherlands.

He is currently a Postdoctoral Researcher with the Dependable Nano-Computing Group at Delft University of Technology. His current research interests include reconfigurable computing, embedded systems, very large scale integration design and test, built-in-self-test, and 3-D stacked integrated circuits, architectures, design for testability, yield analysis, and memory test structures.

Razvan Nane (S'11–M'14) received the Ph.D. degree in computer engineering from the Delft University of Technology, Delft, The Netherlands, in 2014.

He is currently a Postdoctoral Researcher at Delft University of Technology. He is the Main Developer of the DWARV C-to-VHDL hardware compiler. His current research interests include high-level synthesis for reconfigurable architectures, hardware/software co-design methods for heterogeneous systems, and compilation and simulation techniques for emerging memristor-based in-memory computing high-performance architectures.

Said Hamdioui (M'99–SM'11) received the M.S.E.E. and Ph.D. degrees (Hons.) from the Delft University of Technology, The Netherlands. He is currently a Chair Professor on Dependable and Emerging Computer Technologies with the Computer Engineering Laboratory, Delft University of Technology (TUDelft), The Netherlands. Prior to joining TUDelft, he worked for Intel Corporation (Californina, USA), Philips Semiconductors R&D (Crolles, France), and for Philips/ NXP Semiconductors (Nijmegen, The Netherlands). His current research includes dependable CMOS nano-computing (including reliability, testability, and hardware security) and emerging technologies and computing paradigms (including 3D stacked ICs, memristors for logic and storage, and in-memory-computing). He holds one patent and has authored one book and co-authored over 170 conference and journal papers.

Prof. Hamdioui is a member of the Association for European NanoElectronics Activities /ENIAC Scientific Committee Council, an Associate Editor of the IEEE TRANSACTIONS ON VLSI SYSTEMS, and he serves on the editorial board of IEEE DESIGN & TEST and the *Journal of Electronic Testing: Theory and Applications*. He has delivered various keynote speeches, distinguished lectures, and invited presentations and tutorial at major international forums/conferences/schools and at leading semiconductor companies.

Koen Bertels (M'05) received the Ph.D. degree in computer information systems from the University of Antwerp, Antwerp, Belgium. He is currently a Professor and the Head with the Computer Engineering Laboratory, Delft University of Technology, Delft, The Netherlands, where he is researching on quantum computing as a Principal Investigator with the Qutech Research Center. He has co-authored more than 30 journal papers and 150 conference papers. His current research interests include heterogeneous multicore computing, investigating topics ranging from compiler technology, runtime support, and architecture.

Prof. Bertels has been the General and Program Chair for various conferences such as FPL, RAW, and ARC.

Memristive Devices for Computation-In-Memory

Jintao Yu, Hoang Anh Du Nguyen, Lei Xie, Mottaqiallah Taouil, Said Hamdioui
Laboratory of Computer Engineering, Delft University of Technology, the Netherlands
Email: {J.Yu-1,H.A.DuNguyen,L.Xie,M.Taouil,S.Hamdioui}@tudelft.nl

3

***Abstract*—CMOS technology and its continuous scaling have made electronics and computers accessible and affordable for almost everyone on the globe; in addition, they have enabled the solutions of a wide range of societal problems and applications. Today, however, both the technology and the computer architectures are facing severe challenges/walls making them incapable of providing the demanded computing power with tight constraints. This motivates the need for the exploration of novel architectures based on new device technologies; not only to sustain the financial benefit of technology scaling, but also to develop solutions for extremely demanding emerging applications. This paper presents two computation-in-memory based accelerators making use of emerging memristive devices; they are Memristive Vector Processor and RRAM Automata Processor. The preliminary results of these two accelerators show significant improvement in terms of latency, energy and area as compared to today's architectures and design.**

I. Introduction

Today's and new emerging applications, such as data-intensive/big-data applications (e.g., DNA sequencing) and internet-of-things (IoT), are extremely demanding with respect to computing power, energy consumption, and storage. These applications will not only strongly shape our near future, but also impact the semiconductor and computer industry. However, their requirements are difficult to fulfill with today's CMOS based computer architectures, as they face sever challenges both at architectural and device level. Current computer architectures face three walls [1]: (1) the memory wall due to the growing gap between processor and memory speed and the the limited memory bandwidth; (2) the power wall as the practical power budget for cooling has been reached; (3) the instruction-level parallelism (ILP) wall due to the growing difficulties in extracting enough parallelism in software/code that can run on the mainstream parallel hardware today. The CMOS devices also face three walls [2]: (1) the leakage wall as the static power is becoming dominant at small technology nodes (due to volatile technology and low Vdd) and it may even be higher than the dynamic power, (2) the reliability wall as technology scaling leads to reduced device lifetime and higher failure rate; (3) the cost wall as the cost per device from a pure geometric scaling of technology point of view is plateauing. Both architecture and device walls have slowed down the performance gains of CMOS-based architectures. All these motivate the need to look for alternative architectures while considering emerging device technologies.

Many alternatives architectures are under investigations. Resistive computing [3–5] and neuromorphic computing architectures [6,7] using memristive devices, and quantum computing using quantum dots [8] are couple of examples. Resistive computing architectures based on memristive devices are attractive, as they enable in-memory computing (reducing the memory wall) [2,9]. In addition, the memristive devices have zero standby power [6] (helps reducing both the leakage and power wall), great scalability (reduces the cost wall), high density (reduces the cost wall), and they are CMOS compatible (reduces the cost wall).

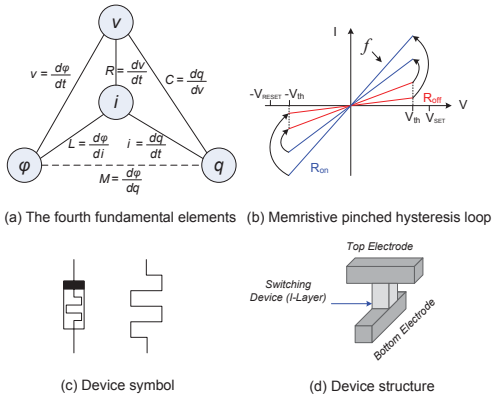
This paper discusses two memristive device based accelerators to demonstrate how computation-in-memory architectures can realize significant improvements, due both to the architecture itself as well as to the used technology to implement them. First, a memristive based vector processor, referred to as Memristive Vector Processor (MVP), is presented; MVP can be used as an accelerator for conventional machines and shows approximately one order of magnitude improvement in performance and energy efficiency. Thereafter, a general model for hardware-based automata processing is introduced and implemented with memristive devices. This implementation is referred to as RRAM-AP; RRAM-AP's key kernel (i.e., the vector dot product operator) outperforms the state-of-the-art SRAM-based implementation by 40% less delay and 27% less energy, at even smaller chip area.

The reminder of this paper is organized as follows. Section II describes briefly the fundamentals of memristive devices. Section III and IV present MVP and RRAM-AP, respectively. Finally, Section V concludes the paper.

II. Basics of Memristive Devices

The memristive device, or *memristor* for short, is the fourth type of fundamental two-terminal electrical components, next to the resistor, capacitor, and inductor. It was initially predicted in 1971 by the circuit theorist Leon Chua [10]. He observed a missing element that can be described as a function of flux ϕ and charge q, as shown (with the dashed line) in Fig. 1a. In theory, a memristive device is a passive element that can be described by the current integral (charge q) through or voltage integral (flux ϕ) across its two terminals; The beauty of the memristive device is its ability to memorize the history (i.e., the internal state). The essential fingerprint of memristive devices is the pinched current-voltage hysteresis loop, as illustrated in Fig. 1b. When a memristive device is floating or when the voltage *v(t)* across it equals zero, the current *i(t)* is also zero. Therefore, based on its hysteresis curve, the memristor has at least two distinctive states: a high (R_H) and low (R_L) resistive state. A memristive device switches

978-3-9819263-0-9/DATE18/©2018 EDAA



(a) The fourth fundamental elements (b) Memristive pinched hysteresis loop

(c) Device symbol (d) Device structure

Fig. 1. Main characteristics of a memristive device.

from high (low) to low (high) state by applying a voltage V_{SET} (V_{RESET}) with an absolute value larger than its threshold voltage V_{th}. Another signature of the memristive devices is that the pinched hysteresis loop shrinks with a higher excitation frequency f as shown in Fig. 1b. Fig. 1c shows the two typical symbols used to denote memristive devices; the black square represents the positive terminal.

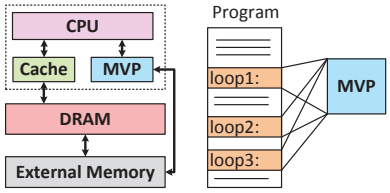
After a silent period for more than thirty years, a practical memristive device was fabricated and demonstrated by HP in 2008 [11]. HP built a metal-insulator-metal device using titanium oxide as an insulator and identified the memristive behaviour over its two-terminal node as described by Leon Chua; as shown in Fig. 1d. The device resistance is modulated by controlling positive charged oxygen vacancies in the insulator layer using different voltages. After the first memristive device was fabricated, several memristor devices based on different types of materials have been proposed such as spintronic, amorphous silicon, and ferroelectric memristors [6].

III. Memristive Devices for Vector Processing

Memristor-based Computation-In-Memory (CIM) concept was proposed to eliminate the communication between the CPU and memory by leveraging memristors for both storage and computation in the same physical crossbar [3,12,13]. Here, we use the CIM to realize an accelerator we refer to as Memristive Vector Processor (MVP). The rest of this section will describe the working principle of MVP, the targeted applications and some analytical evaluation results to show the potential of such an architecture.

A. Working principle

MVP is proposed to accelerate applications with a huge number of vector operations. It can be used as an accelerator for a conventional processor, as shown in Fig. 2a. Similarly as in conventional architectures, the processor fetches, decodes and executes a program using a memory hierarchy consisting of cache(s), DRAM, and external memory. The part of the program which is memory intensive will be offloaded to



(a) Architecture (b) Expected Application

Fig. 2. Memristive Vector Processor architecture.

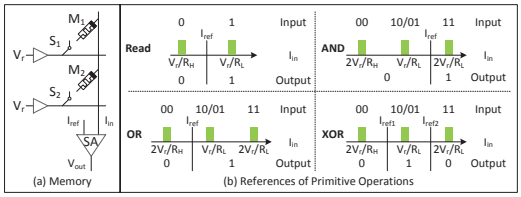


Fig. 3. Scouting logic [14].

MVP. The distinct feature of MVP is its crossbar memory implementation using memristive devices, which enables not the storage of huge amount of data (due to its nano scale size), but also the processing of operations within the memory (i.e., no need for data movement).

The processing in MVP is performed based on scouting logic operations [5,14] ; they transform memory read operations into logical operations. Normally, when a memory cell is being read, a read voltage V_r is applied to the activated row as shown in Fig. 3a. Subsequently, a current will flow through the bit line to the input of the sense amplifier (SA) where it is compared to a reference current. Depending on the cell value (either low (R_L) or high (R_H) resistance), the output of the SA will produce either logic 1 or 0. Inspired by this read operation, scouting logic is able to implement OR, AND and XOR gates. Instead of reading a single memristor at a time, scouting logic activates two (or more) memory rows simultaneously. As a result, the input current to the sense amplifiers is determined by the equivalent input resistance of the activated rows. This resistance results in three possible values: R_H, $R_H//R_L \approx R_L$, or $R_L/2$; by changing the reference current of the SA, different gates can be realized (as shown in Fig. 3b). Therefore, using this scheme allows MVP to perform logical operations by just a small modification of the peripheral circuit of the crossbar mememory. It eliminates the necessity of temporary registers, loading latency and energy to move data from memory to registers. It also increases the parallelism of the architecture and does not impact the the endurance of the memristive devices.

B. Potential targeted applications

With its unique capability, MVP is able to accelerate data intensive applications. These applications consist of intensive memory accesses that consume an enormous amount of energy and degrade the overall performance due to data

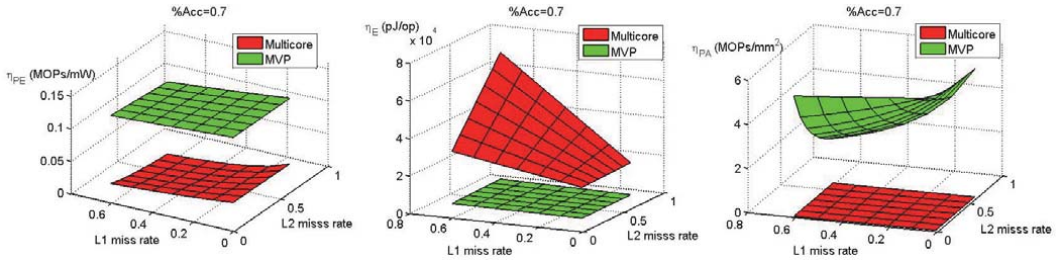


Fig. 4. Evaluation results for MVP and multicore architectures.

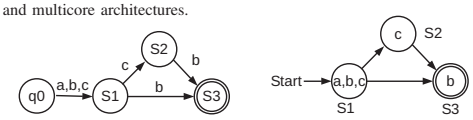
movements through the memory hierarchy; note that loading a word from the on-chip SRAM or off-chip DRAM costs much more energy (50x and 6400x, respectively) as compared with an ALU operation [15,16]. Therefore, eliminating data movements/ communication significantly improves the overall performance.

An example of a program that could benefit from MVP is illustrated in Fig. 2b. The program consists of multiple loops processing a dataset that is preloaded and mapped on MVP. Each time a loop is called, the processor sends a (macro)-instruction to MVP; the instruction is locally decoded and executed. The result is returned to the processor. This feature occurs in multiple applications such as database management [17], DNA sequencing [18–20], and graph processing [21].

C. Evaluation Results

To evaluate MVP architecture, its estimated performance is compared to a multicore architecture. The models and assumptions for the multicore architecture and MVP are similar to those in [3,9]; e.g., the multicore architecture consists of 4 cores (ALU only), two levels of caches (32 KB L1 and 256 KB L2) and 4 GB DRAM. The MVP architecture consists of one core (ALU only), two levels of caches (32 KB L1 and 256 KB L2), 2 GB DRAM, and a MVP with a 2 GB non-volatile crossbar memory with a modified read-out circuity (as explained in [14]) in order to enable computation-in-memory. Three metrics are used for the evaluation: (1) performance energy efficiency η_{PE} (defined by MOPs/mW), (2) energy efficiency η_E (defined by pJ/op), and (3) performance area efficiency η_{PA} (defined by MOPs/mm^2).

Fig. 4 shows the results of the evaluation metrics for both architectures for different L1 and L2 cache misses (up to 60%)and by assuming that 70% of the program instructions can be accelerated on MVP (%Acc=0,7); i.e., the 30% non-accelerated instructions is executed by the conventional processor and the 70% accelerated part by MVP; see Fig. 2. As MVP architecture contains a conventional part (i.e., CPU, caches, DRAM and external memory), only 10x improvement is obtained with respect to the performance-energy efficiency. MVP architecture also achieves one order of magnitude energy efficiency improvement in comparison with the multicore architecture, and has a higher performance area efficiency. Therefore, the MVP architecture has the potential of realizing significant improvements, despite the high switching latency and low endurance of memristor devices. The improvements are the result of a significant reduction of cache and DRAM accesses, and the usage of non-volatile memory. The reduction of memory accesses leads to a lower latency and lower energy consumption, while the non-volatile memory reduces the static power practically to zero.



(a) NFA (b) Homogeneous automata

Fig. 5. Example notations for NFAs and homogeneous automata.

IV. Memristive Devices for Automata Processing

Automata-based processing is widely used in diverse fields, including network security [22], computational biology [23], and data mining [24]. Its hardware implementation, referred to as *automata processors (APs)*, has significant advantages over von Neumann architectures regarding throughput and energy efficiency as they enable computation-in-memory [25–27]. Memristive devices, which are the enablers of Resistive Random-Access Memories (RRAM) and computation-in-memory, are potential candidates for implementing the APs as it will be shown in this section. We will refer to this implementation as RRAM-AP. Moreover, it will be shown that RRAM-AP outperforms the two known hardware implementations of APs, being the Micron Automata Processor [25] which is based on SDRAM, and the Cache Automation [27] which is based on SRAM; we will refer to them by SDRAM-AP and SRAM-AP, respectively, to maintain the naming consistent with RRAM-AP. Next, we will first introduce basic knowledge and notations of automata. Subsequently, we propose a generic model for automata processors. Thereafter, we present RRAM-AP implementation, and show its superiority.

A. Automata Basics

A Non-deterministic Finite Automata (NFA) can be represented by a 5-tuple: $(Q, \Sigma, \delta, q0, C)$. Q represents a finite set of states (which are denoted with circles in the illustrative example of Fig. 5a), Σ is a finite set of possible input symbols (that can be used to generate an input sequence), δ is the transition function describing the set of possible transitions

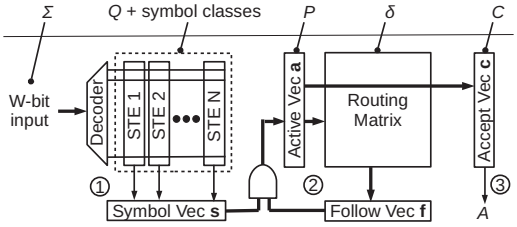


Fig. 6. General architecture for automata processors.

among the states, $q0$ is one of the states from Q and presents the *start state*, C is a subset of Q and contains the *final states* or *accepting states*; they are denoted with a double circle in the state diagram af Fig. 5a as shown for the final state S3.

During operation (i.e., execution of an input sequence), some states can be *active*; they are denoted by P. Initially, P equals to $q0$. At each processing step, the NFA consumes one symbol I from the input sequence. Based on I and δ, P is updated. Once all symbols of the input sequence are processed, the NFA output is determined by P and C. If $P \cap C \neq \emptyset$, then we say that the NFA *accepts* the input sequence; otherwise, the sequence is *rejected*. The acceptance of the input sequence can be represented by a Boolean value A.

Homogeneous automaton is a special type of NFA that is relatively easy to implement by APs [25]. It requires that a state can only be reached by transitions with the *same* input symbol(s). These input symbols belong to the *symbol class* of this state. For example, in the NFA shown in Fig. 5b, S3 can be reached by two transitions (from S1 and S2, respectively) both with the same symbol b; b belongs to the symbol class of S3. Here, the NFA shown in Fig. 5a is a homogeneous automaton and can be therefore redrawn as depicted in Fig. 5b. Note that the input symbols are only related to the *states* in homogeneous automata and not the *state transitions* as is the case for normal NFAs; e.g., the symbol b is not on the incoming edges/transition of the state S3 (see Fig. 5a) but rather within the node representing S3 (see Fig. 5b). Any NFA can be translated into its equivalent homogeneous automaton and therefore implemented using APs [25].

B. Generic Automata Processor Model

Before implementing RRAM-AP, we need to understand the key operations conducted by an AP. Therefore, we next present a generic model for APs to identify these operations. This generic model is shown in Fig. 6 and consists of three major processing steps:

1) Input symbol processing: It decodes each symbol I (presented with W bits) of the input sequence by activating only one of the 2^W wordlines, and identifies all states that have an incoming transition occurring on I. These states and the remaining sates are presented by column vectors called State Transition Elements (STEs), and are pre-configured based on Q and the corresponding symbols (symbol class). Each STE presents one state of the N states of Q. The result of this step is mapped to a vector called Symbol Vector $\mathbf{s}$.
2) Active state processing: It generates: (1) all the possible states that can be reached from the current active states P (stored in a vector called Active Vector $\mathbf{a}$) based on these states and the transition function δ (stored in the routing matrix), and stores the result in the Follow Vector $\mathbf{f}$; (2) the next active states (i.e., Active Vector) by bit-wise ANDing $\mathbf{s}$ and $\mathbf{f}$.
3) Output identification: In order to decide about the value of A (i.e., whether the input sequence is accepted or not), the intersection of $\mathbf{a}$ and the *Accept Vector* $\mathbf{c}$ (pre-configured based on C) is checked. That is, if $P \cap C \neq \emptyset$, then $A = 1$ (accept), otherwise $A = 0$ (reject).

Next we will elaborate the above three processing steps.

1) Input symbol processing: As mentioned, the purpose of this is to calculate the Symbol Vector $\mathbf{s}$ for each input symbol. This is done based on the selected row (from the 2^W rows) and the configuration of STEs. Let's assume that for each input symbol, an *Input Vector* $\mathbf{i}$ of 2^W elements is generated where only one element is high (corresponding to the selected wordline); the remaining elements are 0. In addition, assume that the configuration of STEs can be presented by a matrix $\mathbf{V}$ where each column $\mathbf{V_n}$ presents the STE of the state n. Then the nth element of the Symbol Vector $\mathbf{s}$ corresponding to $\mathbf{V_n}$ can be calculated as:

$$s[n] = \mathbf{i} \cdot \mathbf{V_n} = \sum_{k=0}^{2^W} i[k] v_n[k], \ \forall n \in [1, N] \qquad (1)$$

In this equation, the addition and the multiplication represent the Logic OR and AND, respectively. For the example of Fig. 5b, if we assume $\Sigma = \{a, b, c, d\}$, then,

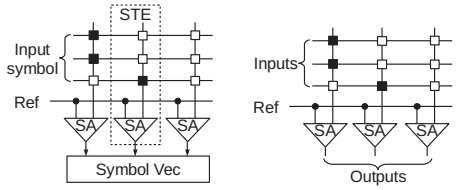
$$\mathbf{V} = \begin{bmatrix} \mathbf{V_1} & \mathbf{V_2} & \mathbf{V_3} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

This means that S1's symbol class is $\{a, b, c\}$, S2's is $\{b\}$, and S3's is $\{c\}$. If we further assume that the current input symbol is b, then $\mathbf{i} = [0\ 1\ 0\ 0]$, and $\mathbf{s} = [1\ 0\ 1]$. This means that b is in the symbol classes of S1 and S3.

2) Active states processing: This step calculates the Follow Vector $\mathbf{f}$ which presents the possible states that can be reached from the current active states stored in the Active Vector $\mathbf{a}$. The transition function is implemented by the routing matrix as shown in Fig. 6, and can be conceptually presented as a two-dimensional vector $\mathbf{R}$. Hence, the nth element of Follow Vector $\mathbf{f}$ can be calculated as:

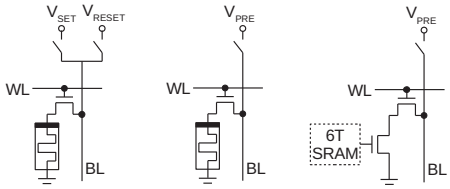
$$f[n] = \mathbf{a} \cdot \mathbf{R_n} = \sum_{i=0}^{N-1} a[i] R_n[i], \ \forall n \in [1, N]. \qquad (2)$$

The interpretation of the addition and the multiplication in this equation is the same as in Equation (1). The next active states (to be also stored in the Active Vector $\mathbf{a}$) are easily calculated by using bitwise AND operation.



(a) Used as STEs (b) Used as routers

Fig. 7. Vector dot product operator used as switches and STEs.



(a) Program circuit (b) RRAM cell (c) SRAM cell [27]

Fig. 8. Different implementations of a configurable bit.

$$a[n] = f[n] \ \& \ s[n], \ \forall n \in [1, N]. \tag{3}$$

For the example of Fig. 5b, the matrix $\mathbf{R}$ that belongs to the transit function is

$$\mathbf{R} = \begin{bmatrix} \mathbf{R_1} & \mathbf{R_2} & \mathbf{R_3} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}.$$

This means that S1 cannot be reached from all the states ($\mathbf{R_1}$), S2 can only be reached from S1 ($\mathbf{R_2}$), and S3 from both S1 and S2 ($\mathbf{R_3}$). For $\mathbf{a} = [1\ 0\ 0]$ (only S1 is active), $\mathbf{f} = [0\ 1\ 1]$ according to Equation (2). This means S2 and S3 are reachable states from the active states. If we assume the next input symbol is b, which leads to $\mathbf{s} = [1\ 0\ 1]$ as discussed above, then the new active vector $\mathbf{a} = [0\ 0\ 1]$ according to Equation (3). This means that S3 becomes the next active state.

3) Output identification: The output value A of NFA is easily calculated using the Active Vector $\mathbf{a}$ and the Accept Vector $\mathbf{c}$. The former stores the active states generated by the input sequence while the later stores the defined accepting states of NFA.

$$A = \mathbf{a} \cdot \mathbf{c}^\top = \sum_{n=0}^{N-1} a[n]c[n]. \tag{4}$$

$A = 1$ means that the input symbol sequence is accepted by the NFA; otherwise, the string is rejected. For the example of Fig. 5b, $\mathbf{c} = [0\ 0\ 1]$. This means only S3 is an accepting state. If we assume the same example as above ($\mathbf{a} = [0\ 0\ 1]$), then $A = 1$.

C. RRAM-AP Implementation

The automata processing model described above contains only two types of logic operations, which are vector dot product (Equation 1, 2, and 4) and vector bit-wise AND (Equation 3). In practice, we cannot implement the complete routing matrix of Equation 2, as it requires too much resource. SDRAM-AP and SRAM-AP both use hierarchical routers to implement the routing matrix. Their implementations do not support all NFA transitions; nevertheless, there is enough flexibility to route all possible transitions of typical applications [25,27]. While SDRAM-AP does not reveal many implementation details, SRAM-AP uses a two-level structure that consists of global and local switches [27]. These global and local switches also conduct vector dot product operations.

For our implementation, we adopt SRAM-AP's for the routing matrix, use the hardware structure shown in Fig. 7a for STEs, and the one in Fig. 7b both for global and local switches. The black and white boxes represent different configuration bits. Each column generates the vector dot product of the input vector and the configuration bits of this column.

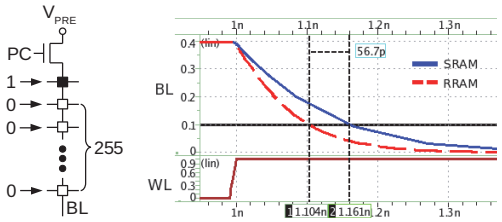
An NFA is configured to RRAM-AP by programming RRAM devices to either low or high resistance. We use one transistor and one RRAM device (1T1R) to implement a configurable bit as shown in Fig. 8b. During the configuration, the word line WL selects the row to be programmed, and the programming voltage is applied to the bit line BL as shown in Fig. 8a. The programming voltage can be either SET or RESET voltage. Logic 1 corresponds to the memristor's low resistance, and logic 0 to high resistance. The bit line is pre-charged before evaluation, and the word lines are selected, e.g., by the input symbols. Note that for the routing matrix, multiple word lines can be activated in parallel. The vector dot product is calculated when all the word lines are set; if all the corresponding selected cells contain a high resistance (i.e., logic 0), then the pre-charged bit line remains high, and the sense amplifier (SA) will read a logic 0 (inverted output). Similarly, if at least one of the cells contains a low resistance (i.e., logic 1), then BL will be discharged. The SA's output will subsequently be a logic 1.

The characteristics of memristors provide opportunities for RRAM-AP to outperform previous designs. For example, SRAM-AP uses eight transistors to implement the configurable bit as shown in Fig. 8c [27], whose area is much larger than the 1T1R structure. In addition, the SRAM cells also suffers from leakage power. As memristors are non-volatile devices, RRAM-AP can resume the last configured NFA after shut down and reboot without reprogramming it. On the other hand, RRAM-AP also inherits some drawbacks, such as the longer and power-hungry programming phase, and lower endurance, in comparison with SDRAM and SRAM.

D. Preliminary Results

The APs can be built by using only vector dot product and bit-wise AND operators. Except for the vector dot product operator, we assume that the remaining part of RRAM-AP is implemented in a similar way as SRAM-AP (incl. bit-wise AND, wiring, and sense amplifiers). Hence, we compare only the dot product operator. Note that SRAM-AP outperforms SDRAM-AP regarding the throughput and energy consumption; therefore, we limit our comparison to SRAM-AP.

The simulated circuit consists of a single vector dot product operator with a length of 256 as shown in Fig. 9a. We use



(a) Simulated circuit (b) SPICE simulation result

Fig. 9. SPICE simulation results of a vector dot product operator.

32 nm PTM model for CMOS transistors and ASU model [28] for RRAM. We configure RRAM's parameters based on a two-state device, similarly as presented in [29], e.g., the RRAM's high and low resistances are approximately 100 MΩ and 1 kΩ respectively; the SET and RESET threshold voltages are 1.3 V and 0.5 V. To simulate the slowest discharge process, only the first cell is configured to logic 1 (indicated by the black box), and the remaining 255 cells are configured to be 0 (indicated by white boxes). The bit line BL is pre-chared to 0.4 V (lower than RRAM's threshold voltages). When BL is discharged to 0.1 V, the sense amplifier (not included in the circuit) will read a 1. The reference voltage of the SA is set to 0.25 V.

The HSPICE simulation results are shown in Fig. 9b. The word line WL is enabled at 1 ns, and then BL starts discharging. BL's voltages in SRAM and RRAM-based designs are illustrated with solid blue line and dashed red line, respectively. The discharge time through RRAM (104 ps) is 35% less than the SRAM-based implementation (161 ps). This is mainly because transistors have relatively large intrinsic capacitance. During bit-line discharge, the RRAM cell of Fig. 8b has only one transistor in its path while the SRAM-based design has two (See Fig. 8c). The energy consumed during the charge and discharge processes is 2.09 fJ for the RRAM-based design and 5.16 fJ for the SRAM-based design. The former is 59% less than the latter. Considering that the remainder part of RRAM-AP is implemented in a similar way as SRAM-AP, RRAM-AP outperforms SRAM-AP at the chip level regarding latency, energy, and area.

V. CONCLUSION

In this work, we have discussed two potential applications of memristive devices and computation-in-memory, i.e., Memristive Vector Processor and RRAM Automata Processor. Memristors' unique properties provide us an important opportunity to improve conventional designs at both architectural and device level. However, the drawbacks of memristor technology, such as the impact of endurance, require further research.

REFERENCES

[1] J. L. Hennessy *et al.*, *Computer architecture: a quantitative approach*. Elsevier, 2011.

[2] S. Hamdioui *et al.*, "Memristor for computing: Myth or reality?" in *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2017, pp. 722–731.

[3] ——, "Memristor based computation-in-memory architecture for data-intensive applications," in *DATE'15*. EDA Consortium, 2015, pp. 1718–1725.

[4] P. E. Gaillardon *et al.*, "The programmable logic-in-memory (plim) computer," in *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2016, pp. 427–432.

[5] S. Li *et al.*, "Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories," in *DAC'16*. New York, NY, USA: ACM, 2016, pp. 173:1–173:6.

[6] J. J. Yang *et al.*, "Memristive devices for computing," *Nature nanotechnology*, vol. 8, pp. 13–24, 2013.

[7] S. Furber, "Large-scale neuromorphic computing systems," *Journal of neural engineering*, vol. 13, p. 051001, 2016.

[8] X. Fu *et al.*, "A heterogeneous quantum computer architecture," in *CF'16*. ACM, 2016, pp. 323–330.

[9] H. A. Du Nguyen *et al.*, "On the implementation of computation-in-memory parallel adder," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2017.

[10] L. Chua, "Memristor-the missing circuit element," *IEEE Transactions on circuit theory*, vol. 18, pp. 507–519, 1971.

[11] D. B. Strukov *et al.*, "The missing memristor found," *nature*, vol. 453, pp. 80–83, 2008.

[12] M. Barbareschi *et al.*, "Memristive devices: Technology, design automation and computing frontiers," in *DTIS'17*. IEEE, 2017, pp. 1–8.

[13] H. A. Du Nguyen *et al.*, "Memristive devices for computing: Beyond cmos and beyond von neumann," in *25TH IFIP/IEEE International Conference on Very Large Scale Integration*. IEEE, 2017, pp. 1–8.

[14] L. Xie *et al.*, "Scouting logic: A novel memristor-based logic design for resistive computing," in *VLSI (ISVLSI), 2017 IEEE Computer Society Annual Symposium on*. IEEE, 2017, pp. 176–181.

[15] A. Danowitz *et al.*, "Cpu db: recording microprocessor history," *Communications of the ACM*, vol. 55, pp. 55–63, 2012.

[16] A. Pedram *et al.*, "Dark memory and accelerator-rich system optimization in the dark silicon era," *IEEE Design & Test*, vol. 34, pp. 39–50, 2017.

[17] K. Wu, "Fastbit: an efficient indexing technology for accelerating data-intensive science," in *Journal of Physics: Conference Series*, vol. 16, no. 1. IOP Publishing, 2005, p. 556.

[18] K. K. Soni *et al.*, "Efficient string matching using bit parallelism," *International Journal of Computer Science and Information Technologies*, 2015.

[19] R. D. Cameron *et al.*, "Bitwise data parallelism in regular expression matching," in *Proceedings of the 23rd international conference on Parallel architectures and compilation*. ACM, 2014, pp. 139–150.

[20] D. Lavenier *et al.*, "Dna mapping using processor-in-memory architecture," in *Workshop on Accelerator-Enabled Algorithms and Applications in Bioinformatics*, 2016.

[21] S. Beamer *et al.*, "Direction-optimizing breadth-first search," *Scientific Programming*, vol. 21, pp. 137–148, 2013.

[22] F. Yu *et al.*, "Fast and memory-efficient regular expression matching for deep packet inspection," in *2006 Symposium on Architecture For Networking And Communications Systems*, Dec 2006, pp. 93–102.

[23] I. Roy *et al.*, "Discovering motifs in biological sequences using the micron automata processor," *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, vol. 13, pp. 99–111, Jan. 2016.

[24] K. Wang *et al.*, "Sequential pattern mining with the micron automata processor," in *CF'16*. ACM, 2016, pp. 135–144.

[25] P. Dlugosch *et al.*, "An efficient and scalable semiconductor architecture for parallel automata processing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, pp. 3088–3098, Dec 2014.

[26] C. Bo *et al.*, "Entity resolution acceleration using the automata processor," in *Big Data*, Dec 2016, pp. 311–318.

[27] A. Subramaniyan *et al.*, "Cache automaton," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-50 '17. New York, NY, USA: ACM, 2017, pp. 259–272.

[28] P. Y. Chen *et al.*, "Compact modeling of rram devices and its applications in 1t1r and 1s1r array design," *IEEE Transactions on Electron Devices*, vol. 62, pp. 4022–4028, Dec 2015.

[29] X. A. Tran *et al.*, "High performance unipolar aloy/hfox/ni based rram compatible with si diodes for 3d application," in *2011 Symposium on VLSI Technology - Digest of Technical Papers*, June 2011, pp. 44–45.

A Computation-In-Memory Accelerator Based on Resistive Devices

Hoang Anh Du Nguyen, Jintao Yu, Muath Abu Lebdeh, Mottaqiallah Taouil, Said Hamdioui
Delft University of Technology
Delft, The Netherlands
{H.A.DuNguyen},{J.Yu-1},{M.F.M.AbuLebdeh},{M.Taouil},{S.Hamdioui}@tudelft.nl

3

ABSTRACT

Today's computing architectures suffer from the three well-known bottlenecks, which are the memory, the power and the instruction-level parallelism walls. Emerging non-volatile technologies, such as memristor, enable new resistive architectures that alleviate at least two of such bottlenecks, as they can process data within the memory with almost no leakage. In this paper, we propose a novel resistive computing architecture by extending a conventional architecture with a resistive based Computation-In-Memory accelerator (CIMX). We evaluate the delay, energy and area of the conventional and CIMX architecture using an analytical model and a simulation framework. The results (both based on the analytical model and simulation framework) show that the proposed architecture achieves at least one order of magnitude improvement in terms of performance, area, and energy efficiency for the considered benchmarks.

KEYWORDS

Computation-in-Memory, Accelerator, Resistive Computing, Memristor

ACM Reference Format:
Hoang Anh Du Nguyen, Jintao Yu, Muath Abu Lebdeh, Mottaqiallah Taouil, Said Hamdioui. 2019. A Computation-In-Memory Accelerator Based on Resistive Devices. In *Proceedings of the International Symposium on Memory Systems (MEMSYS '19), September 30-October 3, 2019, Washington, DC, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3357526.3357554

1 INTRODUCTION

Performance improvement of current computing architectures is gradually saturating due to three well-known bottlenecks: memory wall, power wall and instruction-level parallelism (ILP) wall [1]. These bottlenecks are direct consequences of the von Neumann load/store architecture, speed gap between processor and memory, and the difficulty of exploiting the maximum parallelism, respectively. In order to address these walls, novel architectures using emerging technologies are under research as an alternative for future computing systems [2].

Memristor is one of the promising technologies to enable new architectures [3, 4, 5, 6] due to its great scalability, high integration density, and its near-zero standby power, etc. [2, 7, 8]. Based on such devices, several potential resistive computing architectures (e.g., Computation-In-Memory (CIM) [3, 9, 10], Resistive Associate Processor [11, 5], Programmable Logic-in-Memory (PLiM) [4], Pinatubo [6] etc.) have been proposed as an alternative to today's load/store architectures. The preliminary results of these resistive architectures show several orders of magnitude improvement for different metrics such as energy and area efficiency [3, 9]. However, most of the proposed memristor logic designs (such as threshold/majority logic [12, 13], implication logic [14, 15] and Boolean logic [16, 17, 18]) to build such architectures require multiple accesses of the memristor devices in order to perform a single logic operation; these do not only impact the latency, but more important, they significantly reduce the lifetime of the devices (due to limited endurance) and increases the overall write energy. In [6, 19], the authors partially addressed these problems by performing logic operations, while reading the memory, but then using customized read circuitry (e.g., sense amplifiers). During these read operations, the memristors do not switch and hence, the device lifetime is not affected and the overall energy is reduced. Nevertheless, the authors require multiple steps to perform XOR operations. More importantly, as the nonvolatile memory is deployed as the main memory, it does not entirely solve the endurance problem due to frequent write operations. In addition, the authors did not consider communication infrastructure between the processor and memory (i.e., maintaining memory coherency when parts of the computation occur in memory).

In this work, we propose a novel Computation-in-Memory (CIM) core based on scouting logic. In our previous work [20], we explored the possibility of performing logic operations within non-volatile memories by sensing (and *scouting*) the data stored in the memory, similarly as in [6]. Hence, a memory access provides by itself the result of a pre-defined logic operation. 'Scouting' the current or voltage drop while accessing two (or more) operands at the same time can be used to perform a logic function (e.g., by using a modified sense amplifier). Different from [6], the non-volatile CIM memory is used as an accelerator with infrequent writes (thereby addressing the endurance) and moreover, scouting logic is able to execute all logic operations within a single cycle (thereby addressing the performance) rather than multiple cycles. The contributions of this paper are the following: (1) a novel computation-in-memory core that utilizes scouting logic; we refer to this accelerator as CIM core; (2) an extension of the conventional instruction set with in-memory instructions that supports code execution of kernels on CIM core; (3) an analytical evaluation model of the conventional and CIM-based architectures; it can be used for fast design space exploration; (4) a simulation framework that is able to evaluate both

MEMSYS '19, September 30-October 3, 2019, Washington, DC, USA

ACM ISBN 978-1-4503-7206-0/19/09…$15.00
https://doi.org/10.1145/3357526.3357554

the conventional architecture and CIM-based architecture. Three kernels of different applications are used as case studies.

The remainder of this paper is organized as follows. Section 2 presents the state-of-the-art in memristor logic design, the working principle of scouting logic, and its potential. Section 3 presents potential resistive architectures and the CIM accelerator. Section 4 provides an analytical evaluation model, and design exploration of the conventional and proposed architecture. Section 5 presents a simulation framework and evaluates several benchmarks. Section 6 subsequently evaluates the results, discusses the potential of CIM-based architecture based on the analytical and simulation results. Finally, Section 7 concludes this paper.

2 SCOUTING LOGIC AND ITS POTENTIALS

This section first describes the state-of-the-art in memristor based logic design. Thereafter, it explains the working principle of scouting logic. Finally, it discusses its performance, pros and cons.

2.1 State-of-the-art on Memristor-Based Logic

Several memristor-based logic designs have been proposed [13, 12, 14, 15, 16, 18, 17, 21]. The most popular design suitable for computation-in-memory are Snider logic [18, 16], implication logic [15, 14] and Memristor-aided (MAGIC) logic [22]. They are briefly discussed next.

Snider logic provides two primitive gates: NOT and NAND [16]. It uses high resistance R_H and low resistance R_L to represent a logic 1 and 0, respectively. The two-input NAND gate of Fig. 1(a) is used as an example to explain the working principle of Snider logic. We assume that the inputs are stored in memristor M_1 and M_2, while the output is produced in M_o. The gate requires an extra resistor R_s ($R_L \ll R_s \ll R_H$). To perform NAND, two steps are needed. First, M_o should be RESET to R_H. Next, control voltages V_h (half-select voltage) and V_w (write voltage) are applied to the input and output memristors, respectively; V_h and V_w need to satisfy the following constraint: $V_w > V_{reset} > V_h = \frac{V_w}{2}$, where V_{reset} is the minimum voltage to switch a memristor from R_L to R_H [23, 18]. In case one of the inputs is 0, the equivalent resistance of M_1 and M_2 is around R_L (see e.g. Fig. 1(a) where $M_1 = R_L$ and $M_2 = R_H$). Therefore, the voltage V_x across the resistor R_s is around V_h as $R_L \ll R_s \ll R_H$. As a result, the voltage across the output memristor M_o is $V_w - V_x \approx V_w - V_h = \frac{V_w}{2} < V_{\text{reset}}$, and therefore M_o stays at R_H (logic 1).

Implication logic provides only a single primitive gate, which is material implication (IMP) [15]. Opposite to Snider logic, IMP uses R_H and R_L to represent logic 0 and 1, respectively. IMP gate uses two memristors M_1 and M_2 to store the inputs; the output is overwritten into memristor M_2. The working principle of IMP is similar to Snider logic. Multiple sequential IMP gates can be used to realize AND, OR, or XOR gates.

MAGIC logic provides two primitive gates: NOR and NOT [22], where the NOT gate is constructed in a similar way as the NOR gate but without the second input. Similar to implication logic, MAGIC uses R_H and R_L to represent logic 0 and 1, respectively. Fig. 1(b) shows a NOR gate; memristors M_1 and M_2 store the inputs, while the output is produced in M_o. To perform NOR, two steps are needed. First, M_o is SET to R_L. Next, control voltages V_w (write voltage)

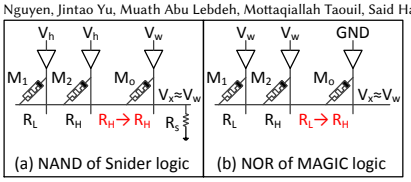


Figure 1: Logic designs suitable for resistive computing

and *GND* (Ground) are applied to the input and output memristors, respectively. Note that the value of V_w must satisfy the following constraints: $2V_{reset} < V_w < V_{set}$, where V_{set} is the minimum voltage required to switch a memristor from R_H to R_L. In case one of the inputs are 1, the equivalent resistance of M_1 and M_2 is around R_L. Therefore, the voltage across M_o is $V_x \approx \frac{V_w}{2} > V_{\text{reset}}$ and the output memristor switches to R_H (logic 0).

The three previous discussed designs suffer from major concerns. First, the limited memristor endurance puts a constraint on using these designs, as they require multiple write steps/accesses to execute a single operation. In addition, the multiple write steps also affect the performance. Second, processing a logic operation requires several control voltages (e.g., V_w) that have to drive memristor devices, therefore, higher power requirement. All of these indicate the severe limitation of such designs in realizing computation-in-memory architectures. One alternative design to solve these limitations is to perform the operation within the peripheral circuits of the memory, as will show next.

2.2 Scouting Logic

Scouting logic executes logic operations by modifying the read circuitry [20]. Fig. 2(a) shows a resistive memory with two cells based on 1T1R cells. Normally when a cell is read, (e.g., memristor M1), a read voltage V_r is applied to its row and switch S_1 is activated. Subsequently, a current I_{in} will flow through the bit line to the input of the sense amplifier (SA). This current is compared to the reference current I_{ref}. If I_{in} is greater than I_{ref} (i.e., when R_1 has a low resistance R_L), the output of the SA changes to V_{dd} (logic 1). Similarly, $I_{in} < I_{ref}$ (i.e., when R_1 has a high resistance R_H), the output changes to logic 0. For proper operations, I_{ref} should be fixed between the high and low current as depicted in the top left part of Fig. 2(b).

Inspired by this read operation, we demonstrate how to implement OR, AND and XOR scouting logic gates, which are frequently used bitwise logic operations [24, 25]. Instead of reading a single memristor at a time, scouting logic activates two (or more) inputs of the gate simultaneously (e.g., M_1 and M_2 in Fig. 2(a)). As a result, the input current to the sense amplifier is determined by the equivalent input resistance ($R_1 // R_2$). This resistance results in three possible values: $\frac{R_L}{2}$, $\frac{R_H}{2}$ and $(R_L // R_H) \approx R_L$. Hence, the input current I_{in} also can have only three values. By changing the value of I_{ref}, different gates can be realized. To implement an OR gate, I_{ref} should be set between $\frac{2V_r}{R_H}$ and $\frac{V_r}{R_L}$ as depicted in the left bottom part of Fig. 2(b)). As a result, the output is 0 only when $R_1 // R_2 = \frac{R_H}{2}$. Similarly, to implement an AND operation, I_{ref} should be set between $\frac{2V_r}{R_L}$ and $\frac{V_r}{R_L}$. The XOR operation needs two references and the output is logic 1 only when $R_1 // R_2 \approx R_L$.

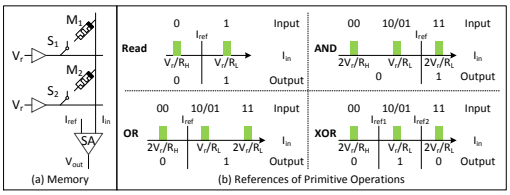


Figure 2: Scouting logic [20]

Table 1: Parameters

Parameter	Description	Value
	Technology	
	Memristor (TaO_x) [27, 2]	
F (nm)	Feature size	90
R_L ($k\Omega$)	Low resistance	200
R_H ($M\Omega$)	High resistance	10
A_{cell} (μm^2)	Area of a 1T1R cell	0.0486
T_{sw}(ns)	Switching time (max of SET and RESET)	1.71
V_r (V)	Read voltage	0.9
	CMOS	
	UMC 90nm Library	

Table 2: Different operation performance

	Delay (ns)	Power (uW)			Area (um^2)
		OR	AND	XOR	
CSA	2.73	15.99	17.19	17.59	29.7216
VCA	9.31	8.65	6.00	11.01	22.324

The above implies two requirements for the SA. First, it should be operational with both a single and two references. Second, it must support a re-configurable reference. Two SA designs including a current-based SA (CSA) and voltage-based SA (VSA) that satisfy both requirements are described in [20]; in addition, scouting logic is verified using Cadence Spectre simulations. The simulation model consists of the 1T1R array of Fig. 2(a) connected to one of the SA designs; both SAs have been verified. The 1T1R array and SAs are described by a SPICE netlist, while the memristor model [26], CMOS controller and voltage drivers by Verilog-A modules. The simulation parameters are extracted from [27] and summarized in Table 1. The SA designs are simulated with the PTM 90nm [28] library. The functionality of scouting logic has been verified, and the delay, power and area of the OR, AND and XOR operation for both SA designs (including the cost of their controllers) have been reported; see Table 2 [20]. The CSA design is much faster (3.4x), but has a higher area overhead (1.5x) and consumes more power (1.6-2.8x). In this paper, we select VSA design for the following section due to its favourably lower power consumption.

As a relatively old technology is used, the read voltages mentioned are typically a bit higher than reported [7]. To predict the performance for a 5nm memristor [2] design, we scaled these numbers down. It is worth mentioning that lowering the read voltage might impact the correctness of the scouting logic, as it is strongly dependent on the development of both memristor technology and the modified sense amplifier design.

2.3 Potentials of Scouting Logic

Compared to the state-of-the-art resistive logic designs, scouting logic executes the logic operations during a read operation. Therefore, scouting logic comes with the following advantages:

- Scouting logic executes a logic operation within a single access (i.e., read operation), and hence it improves the performance.
- Scouting logic requires lower control voltages (i.e., V_r is lower than the write voltage V_w), and hence it reduces the power and energy consumption together with better performance.
- Scouting logic only reads the resistance values of memristors instead of writing them. As a result, it has a lower endurance requirement, which is currently a significant challenge for memristor technology [29].
- Scouting logic requires simple control logic, as it does not need to control the sequential steps required by other resistive logic designs, such as implication logic.

3 RESISTIVE COMPUTING ARCHITECTURE

In this section, we first describe potential Computation-in-Memory (CIM) architectures that can utilize scouting logic. Thereafter, we select and explain the most suitable one together with its instruction set in more detail.

3.1 Potential Architectures

With scouting logic, it is feasible to implement a non-volatile memory that supports CIM. In other words, the memristor-based memory module is equipped with one or more scouting logic units. Fig. 3b, 3c, 3d, and 3e show four possible ways to embed a CIM core into the conventional computer architecture of Fig. 3a; they are respectively CIM used as cache, as main memory, as complete memory hierarchy, and as accelerator. Next, we discuss the pros and cons of each of these architectures.

Fig. 3b shows the CIM cache (CIMC) architecture where CIM replaces one or more cache levels. Here, CIM operates in a similar way as conventional caches by storing temporary data that will likely be reused. However, besides the normal cache functionality, CIM can be used to perform in-memory operations. With respect to the normal cache functionality, no modifications are required to the processor, as these internal cache functionality is transparent to processor. The in-memory operations are performed in the cache and prevent moving data towards processors. Hence, it reduces the cost (i.e., latency and power) of moving data between caches and processors, and vice-versa. In addition, CIM provides larger cache size due to small footprint of resistive devices [30, 31, 32]. However, as parts of the instructions are executed in caches, the processors must be made aware of these operations. Therefore, some modifications are required for both the cache controller and processor instruction set to handle coherency. They impact the instruction set, compiler and cache controller. Moreover, as cache is more frequently accessed than main memory, the limited endurance and high read/write latency of resistive devices are the main bottlenecks of this approach [31]. Hence, this architecture makes sense only if the endurance is high enough ($>10^{16}$).

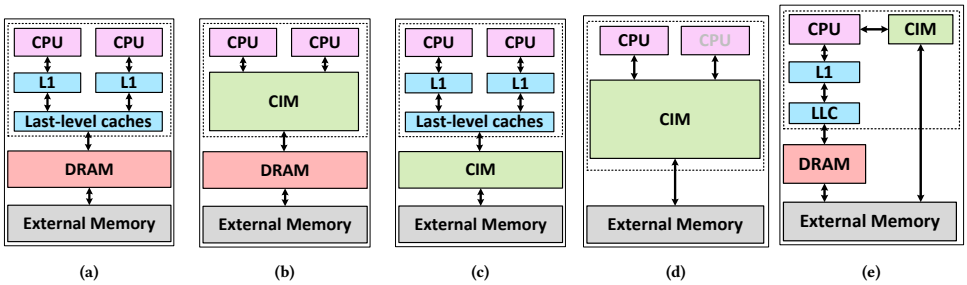


Figure 3: Conventional (a) and potential resistive architectures: CIMC (b), CIMM (c), CIMU (d), CIMX (e)

Fig. 3c shows the "CIM main" (CIMM) where CIM replaces the main memory. Here, CIM operates similarly as a conventional DRAM memory. The scouting logic operations can be used in addition to normal read operations to perform bit operations in the memory; a similar approach has been evaluated for DRAM in [33]. The trade-off between capacity and performance is similar as in the approach where CIM is used to replace the cache. In fact, resistive devices in general are often used as non-volatile memory devices. Several prototypes of up to 32GB memory have been reported recently [34, 35, 36]. These prototypes show a lot of advantages in using resistive devices in main memory such as high density, non-volatility, nano scale devices [37]; these advantages enable high storage volume and low start-up time [37, 38, 39, 40]. However, a lot of efforts are still required to use non-volatile memories in conventional architectures due to some challenges such as high manufacturing cost, high write latency, endurance limitation, and memory coherence management and complexity [41, 40, 42].

Fig. 3d shows the CIM universal (CIMU) where CIM replaces the entire memory hierarchy. One or multiple processors are used to execute the conventional instructions and fetch data directly from CIM; in-memory instructions can be still used to accelerate memory-intensive parts of an application. This reduces the pressure on the registers. However, to be competitive in terms of performance, the memory must have a low write and read latency. In addition, as a lot of write operations are expected also the endurance needs to be high. Despite possible performance issues, this architecture could have a huge potential in terms of energy and area efficiency.

Fig. 3e shows the CIM accelerator (CIMX) where CIM is used as an on-chip data-centric accelerator. CIM executes parts of the instructions that cannot be efficiently handled by conventional processors; for example, a loop that consists of simple operations on a huge data set. As accelerators are often read-favored, the impact on endurance is expected to be minimal. The data-centric accelerator differs from a traditional accelerator such as an FPGA or GPU in the amount of data that can be stored in the accelerator; this huge data storage alleviates the latency and energy cost required to move data back and forth from memory in traditional accelerators. However, it also has commonalities with the traditional accelerators. First, the performance gain is proportionally dependent on the number of parallel operations that can be performed simultaneously. Second, the memory coherence management is managed in a similar way, i.e., the accelerator can be seen from the processor as an extended memory space, similarly as for FPGA accelerators [43, 44]. Third, the accelerators can be executed in parallel with the host processor in case no data dependencies occur.

All the four discussed CIM-based architectures have both pros and cons. In this paper, we evaluate the resistive architecture depicted in Fig. 3e, as it is the only case where CIM can be used as an accelerator and thus has a lower requirement on the endurance. Currently, an endurance of 10^6 to 10^{12} are reported [7, 29]; this is neither sufficient for general purpose computing nor conventional memory. However, it is strongly believed that memristor endurance will increase and reach 10^{16} in the future [29]. In the next sub-section, we will present the CIMX architecture in detail.

3.2 Resistive Computing Architecture

The CIMX architecture of Fig. 3e consists of a conventional processor with cache, main DRAM memory, novel data-centric CIM core and an external memory. CIMX consists of a large non-volatile memory equipped with scouting logic. Both main memory and CIM core can fetch data from the external memory. CIM core is addressable from the processor and uses an extended address space. In this paper, we assume that the data that is stored in the CIM core is not duplicated on the main DRAM memory [45, 46]; hence, no memory coherency schemes are required. The CIM core is initialized with data from the external memory, e.g., database(s); this initialization needs to be performed only once. Note that infrequent modifications are required when read-dominant applications are executed on CIM core.

Similarly as for the FPGA based accelerators [43, 44], pragma insertions are used to denote that particular functions will be executed on CIM core. The pragma insertions are translated into a sequence of in-memory instructions; these are executed in CIM core. The instruction format for an in-memory instruction is constructed in a similar manner as for conventional processor instructions. It includes an opcode to denote the in-memory instruction type and the addresses to operate on. To signal that this operation should not be executed on the conventional processor, a specific opcode is used for the instruction. As a result, the instruction is directly forwarded to the memory controller for further processing.

Fig. 4 presents the internal details of CIM core. The accelerator consists of a memory controller, configuration registers, address decoder, non-volatile memory cell arrays, and a read/write circuitry. The *controller* receives in-memory instructions from the processor;

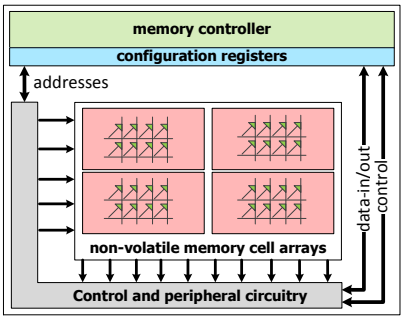


Figure 4: CIM core

each instruction specifies the operation type (read, write, logic or, etc.) and addresses to operate on. The configuration registers are used to decode the instructions and configure the data path. Once the operation finishes, the final result is sent back to the processor in case applicable. The *control and peripheral circuitry* specify which rows are activated and performs a read (including a logical operation) or write operation. The *non-volatile memory* consists of multiple arrays where each array consists of a 1T1R crossbar array (i.e., each cell consists of a transistor and memristor).

It is worth noting that the CIM core considered here can only execute logical operations. More research is required to extend the instruction set with arithmetic operations. Due to this current limitation, the next section will only consider logical operations in CIM core.

3.3 CIM Instruction Set Architecture

In order to support the execution of instructions in CIM core, we create an CIM instruction set based on scouting logic operations. The CIM instruction set includes two types of instructions related to memory and processing. Memory related instructions are typical load/store instructions used to retrieve/save data from/into memory. Processing related instructions consist of logical operations that are executed by scouting logic as mentioned in section 2.

Here we consider the logic instructions CIM_AND, CIM_OR, CIM_XOR representing the and, or, and xor instructions as explained in section 2, respectively. We add an additional compare instruction denoted by CIM_CMP; this instruction will be needed for one of the benchmarks that will be simulated in section 5.2 (i.e., bitmap indexing [33, 47]). The CIM_CMP is composed of a CIM_XOR instruction followed by a bit-wise OR operation; the later operation is executed within the CMOS peripheral circuitry. The instruction set can be easily extended with arithmetic operations when their implementations are available.

4 ANALYTICAL EVALUATION RESULTS

In this section, we first present an analytical evaluation model for both the conventional and CIMX architecture. Thereafter, we evaluate the performance of both architectures using these models. Finally, we describe their shortcomings and limitations.

4.1 Analytical Evaluation Model

In order to evaluate CIMX, we would like to compare its performance, energy consumption and area with a conventional architecture. To be able to realize this, we create two models one for the conventional and one for CIMX architecture. The models can be used to evaluate different applications.

We characterize an application based on its problem size and percentage of logical instructions. The problem size species the amount of program instructions. The percentage of logical instructions is the amount of accelerated instructions (X) that can be offloaded to CIM core. We denote the percentage of logical instructions by n_l, remaining instructions by n_r (arithmetic, conditional, etc.) and the percentage of their corresponding memory accesses by m_l, and m_r, respectively. Note that n_l+n_r=1.

For the **conventional architecture**, we use the Intel Xeon E5-2680 [48] multicore as a baseline. We assume the following:

- The number of cores is n_p=4, each with a frequency of 2.5GHz.
- Each core contains (i) an ALU that is capable of executing non-memory instructions (logical, arithmetic, conditional instructions) in one cycle, (ii) a two level cache (L1 of 32KB and L2 of 256KB) with access latencies of one and two cycles [49], and a miss rate of mr_{L1} and mr_{L2}, respectively. Note that the actual miss rates are application dependent.
- The cores share a main DRAM memory of 4GB with an access latency of 175 cycles (165 cycles for communication and 10 cycles for retrieving the results) [49]. The page fault rate of the main DRAM memory is pf_{dr}=0.1% with a penalty of 800 cycles [50].

For the **CIMX architecture**, we assume the following:

- It contains a single host processor with the same characteristics as an individual core in the conventional architecture. It contains the following:
  (i) an ALU that executes non-logical instructions in one cycle.
  (ii) 32KB L1 cache and 256KB L2 cache. As parts of the application are offloaded to CIM core, the miss rates of the part of the application that are executed on CPU changes. To model this change, we use the *square root* rule of miss rate [51, 52]. This rule says how the cache miss rate changes for different cache sizes. However, in our models we keep the cache sizes between both architecture the same. Instead, we assume that the same relation holds for different problem sizes (due to offloading parts of the application to CIM core). Note that when the problem sizes decreases, it has a similar effect of increasing the cache. Specifically, the miss rates for CIMX (due to accelerated parts) are modeled by $(1 - X)^{(-0.3)}$ in which X represents the accelerated ratio and -0.3 the minimum coefficient in the *square root* rule, i.e., the most pessimistic value.
- 1GB DRAM.
- a CIM core with n_a=1, 048, 576 parallel memory arrays (each with a size of 512 columns x 512 rows) with a total capacity equal to 32GB and an area of nearly equal to 3GB DRAM.

- Unidirectional communication between the processor and CIM core is assumed to be equal to L1 access latency (i.e., 1 cycle). Note that both L1, L2 cache and CIM core are on-chip.
- A logical instruction takes 9.3 *ns* on CIM core based on the VSA (see Table 2); this is equivalent to 20 CPU cycles.

Defining $m = m_l + m_r$ and $m_p = \frac{m}{n_p}$, the CPI of both architectures is modeled as below:

$$CPI_{CONV} = \begin{cases} \frac{(n_l + n_r) \cdot 1}{n_p} + \ldots & t_{comp} \\ m_p \cdot (1 - mr_{l1}) + \ldots & t_{L1} \\ m_p \cdot mr_{l1} \cdot (1 - mr_{l2}) \cdot 2 + \ldots & t_{L2} \\ m \cdot mr_{l1} \cdot mr_{l2} \cdot (1 - pf_{dr}) \cdot 175 + \ldots & t_{DRAM-hit} \\ m \cdot mr_{l1} \cdot mr_{l2} \cdot pf_{dr} \cdot 800 & t_{DRAM-pf} \end{cases} \quad (1)$$

$$CPI_{CIMX} = \begin{cases} n_r \cdot 1 + \ldots & t_{comp-others} \\ \frac{n_l}{n_a} \cdot 4 + \ldots & t_{comp-bw} \\ 1 + \ldots & t_{CPU-CIM} \\ \frac{n_l}{n_a} \cdot 4 & t_{CIM-write} \\ m_r \cdot (1 - mr_{l1}) + \ldots & t_{L1} \\ m_r \cdot mr_{l1} \cdot (1 - mr_{l2}) \cdot 2 + \ldots & t_{L2} \\ m_r \cdot mr_{l1} \cdot mr_{l2} \cdot 175 & t_{DRAM-hit} \end{cases} \quad (2)$$

The CPI of the conventional architecture (see Eq. 1) consists of the sum of the CPU computing latency t_{comp}, the L1 cache accessing latency t_{L1}, the L2 cache accessing latency t_{L2}, the DRAM memory accessing latency assuming a main memory hit $t_{DRAM-hit}$, and the DRAM page fault accessing latency $t_{DRAM-pf}$. Note that for the conventional architecture, there is no differentiation between logical and non-logical instructions and their corresponding memory accesses (i.e., m=m_l+m_r).

The CPI model of CIMX (see Eq. 2) consists of the sum of the non-logical instruction latency $t_{comp-others}$ executed by the processor, the logical instruction compute latency $t_{comp-bw}$ executed by the CIM core, the on-chip communication latency between CPU and CIM core $t_{CPU-CIM}$ estimated by 1 cycle, the non-volatile write latency $t_{CIM-write}$, the L1 cache accessing latency t_{L1}, the L2 cache accessing latency t_{L2}, the DRAM memory accessing latency for a hit rate $t_{DRAM-hit}$. Note that it is assumed that CIM core is large enough to store all the data needed for the accelerator.

Table 3 summarizes the technology parameters. The table contains the delay, area, and power consumption of the conventional CPU (i.e., the logic operations, caches (L1, L2) and DRAM) and of CIM core (i.e., crossbar array and scouting logic). Note that the scouting logic related parameters were already provided in Table 2. Here, we assume the VSA implementation due to its favourably lower power consumption. The data provided in Table 3 is used to evaluate the delay, energy, and area of each architecture. The delay is calculated from the CPI models and the problem size. The energy is computed from the delay, and dynamic and static power. The area is obtained by summing up the areas of all individual components. Note that the controllers of both the conventional and CIMX architectures are not included in the estimation.

For the evaluation, we use the following metrics:

- Performance: defined by the delay (in s).

Table 3: Evaluation parameters

Component	Parameters	
CMOS Technology		
32-bit logic gate	Delay (cycles)	1
	Dynamic power (uW)	33.69 [53, 54, 55]
	Static power (uW)	7.09
	Area (um^2)	32.65 [53, 54]
32KB L1 cache	Delay (cycles)	1
	Dynamic power (W)	0.24 [56]
	Static power (W)	0.012 [56]
	Area (mm^2)	14.2 [56]
256KB L2 cache	Delay (cycles)	2
	Dynamic power (W)	0.63 [56]
	Static power (W)	0.108 [56]
	Area (mm^2)	75 [56]
1GB DRAM	Delay (cycles)	175
	Dynamic power (W)	12.86 [56]
	Static power (W)	12 [56]
	Area (mm^2)	88.98 [56]
4GB DRAM	Delay (cycles)	175
	Dynamic power (W)	51.4 [56]
	Static power (W)	48 [56]
	Area (mm^2)	355 [56]
Memristor Technology		
Scouting logic operation	Delay (cycles)	4
	Power (W)	see Table 2
	Area (mm^2)	
Memristor	Write Delay (cycles)	1
	Write energy (fJ)	1 [2]
	Static energy (J)	0 [2]
	Area (nm^2)	$6F^2$=150 [2]
Non volatile memory	#col_per_row	512
	#row_per_array	512
	#array	1,048,576

- Energy: defined by energy (in J).
- Performance energy efficiency: defined by the number of million operations per second divided by the power consumption (in MOPs/mW).
- Energy efficiency: defined by the energy in pJ divided by the number of operations (in pJ/op).
- Performance area efficiency: defined by the number of million operations per second divided by the total area (in MOPs/mm^2).

4.2 Analytical Results

Based on the models developed in the previous section, we will evaluate the performance of the conventional and CIMX architectures for the 32GB problem size (PS); as this equals the CIM core capacity, we assume that initial data related to the accelerated part is stored in CIM core and does not required to be reloaded. We investigate the impact of two aspects on the performance of both architectures: (i) the L1 and L2 cache miss rates (ranging from 0% to 90%), and (ii) the percentage of logical instructions accelerated by CIM core %Acc. (including 30%, 60%, and 90% as the authors in [6] showed that at least 30% of a database application could be accelerated using in-memory computing). Using this analytical evaluation, we can quickly perform a design space exploration. Fig. 5 shows the performance metric of the conventional architecture

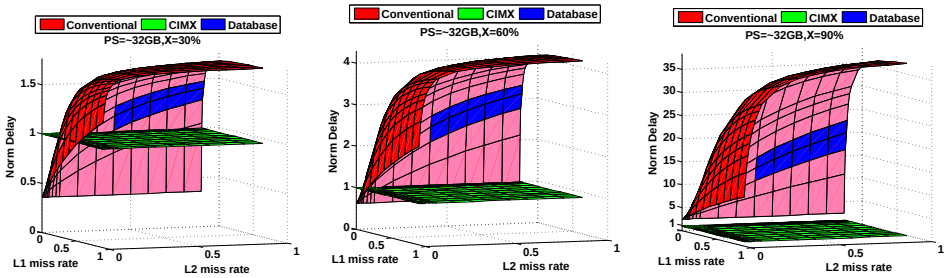


Figure 5: Analytical results of the performance (delay) metric for the conventional and CIMX architectures

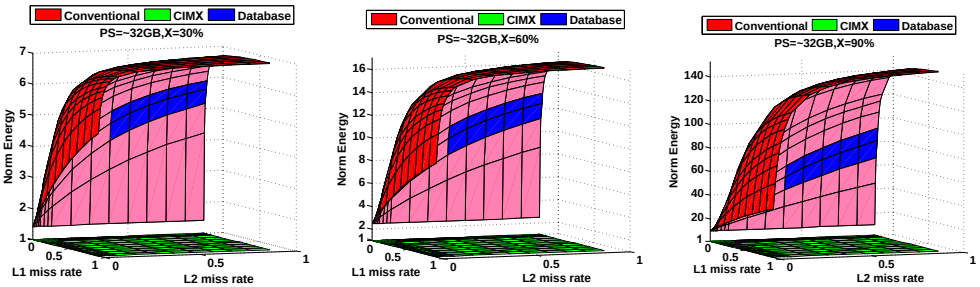


Figure 6: Analytical results of the energy metric for the conventional and CIMX architectures

(red planes) with respect to CIMX architecture (green planes). The results of the conventional architecture are normalized to those of the CIMX; note that the results of the CIMX architecture are presented by the green unit planes. The blue areas on the red planes are explained later. Overall, the performance speed-up of CIMX architecture increases for larger percentages of the instructions executed on CIM core (maximum 1.5x for 30% to more than 35x for 90%). This can be clearly observed as the gap between the red and green planes increases. CIMX architecture outperforms the conventional architecture for most of the L1 and L2 miss rates. High cache miss rates result in a longer memory access latency in the conventional architecture while CIMX architecture does not suffer much from this as computing takes place within the memory (i.e., CIM core). However, CIMX architecture may not achieve a speed-up for low miss rates (e.g., less than 20%). For example, when the miss rates of L1 and L2 are around 10%, a specific amount of instructions (e.g., more than 60%) have to be accelerated to realize a higher performance on CIMX architecture. Note that the miss rate of some big data applications is reported around 6-15% [57, 58, 59]. However, most big data applications such as database application have a much higher miss rates; this will be discussed further later in this sub-section. Moreover, the figure shows that the performance increase saturates for higher L1 and L2 miss rates. The reason for this is that the program executed on the conventional part of CIMX starts to dominate and that therefore the accelerated percentage is bounded by Amdahl's law.

Similarly, Fig. 6 shows the energy metrics for both architectures. Overall, similar trends are observed with respect to the percentage of accelerated instructions. However, the energy consumption of CIMX architecture is always lower, irrespective of the cache miss rates. In case 30% of the instructions are accelerated on CIM core, the conventional architecture consumes 1.5x to 7x less energy. This grows up to 140x in case 90% of the instructions are accelerated on CIM core. The high energy consumption of the conventional architecture can be partly attributed to the data movement back and forth between the CPU and the memory hierarchy. In addition, both cache and DRAM suffer from a much higher leakage current as compared with the non-volatile technology used in CIM core.

Fig. 7 shows the analytic results of the remaining metrics for the 90% acceleration ratio case; note that the results of CIMX are normalized to those of the conventional architecture in the performance energy efficiency and performance area efficiency. CIMX architecture achieves an improvement of two orders of magnitude in terms of performance energy efficiency, energy efficiency, and one order of magnitude in terms of performance area efficiency, mostly due to a higher performance (see Fig. 5c) and lower energy consumption (see Fig. 6c). To the best of our knowledge, CIMX architecture is the first architecture that integrates a conventional

architecture with a scouting logic based accelerator. In comparison to the state-of-the-art, the proposed architecture realizes significant improvements, despite the low performance of individual memristors. The improvement could be much higher if more instructions can be accelerated, and/or if the CIM core performance can be improved. Furthermore, applications with a bad data locality and high data volume can be significantly benefit from the proposed architecture.

We also explore database applications which have an L1 and L2 miss rate ranging from 2 to 4%, and 40 to 90%, respectively [57]. The performance of CIMX in these ranges are marked in blue in Figs. 5, 6, and 7. For these particular miss rates, database applications can be executed up to 15x faster on CIMX architecture depending on the accelerated percentages. Meanwhile, CIMX architecture consumes 5x to 60x less energy than conventional architecture. This results in one order of magnitude improvement in terms of performance energy efficiency, energy efficiency, and performance area efficiency. This exploration quickly shows that database applications are a suitable candidate to be accelerated on CIMX architecture.

The proposed architecture also shows more than 150x improvement in energy efficiency. Therefore, it can be considered as a potential platform for low-power or portable embedded applications. To make CIMX architecture better suitable for low power/ portable embedded applications, the complex memory hierarchy has to be removed (i.e., no L2 cache and DRAM). We evaluated the impact of the L1 cache miss rate (ranging from 0% to 70%) and percentage of accelerated instructions (ranging from 30% to 90%) for the gigabyte (GB) problem size. Fig. 8 shows the evaluation metrics of such modified architecture. We observe that the modified architecture achieves a performance energy efficiency higher than 100 MOPs/mW, which is much better than most application-specific designs [60]. In addition, the energy efficiency is reaching the requirement of 1pJ/op for embedded applications [60] as shown in Fig. 8(b). This shows the necessity of integrating resistive devices in embedded systems to realize energy and area efficiency.

4.3 Limitations of Analytical Model

The analytical model assumes a uniform profile of the application (e.g., the same L1 miss rate throughout the entire application). However, an application consists normally of multiple parts with different behavior, such as memory-intensive, computation-intensive or bandwidth-intensive parts. In addition, modeling the communication between the kernel is not trivial. Furthermore, both architectures do not consider the cost of the memory controller due to their modeling complexity. For these reasons, a simulation framework is necessary. Nevertheless, the analytical approach can be used for quick insights and to perform a fast exploration. In the next section, we propose a simulation framework that can be used to simulate both conventional architectures (Fig. 3a) and CIMX architecture (Fig. 3e).

5 SIMULATION FRAMEWORK AND RESULTS

In this section, we first explain the simulation framework. Thereafter, we discuss the simulation setup and applied benchmarks. Finally, we present the simulation results for both the conventional and CIMX architectures.

5.1 Simulation Framework

Fig. 9 shows the simulation framework (SiCIM) used to simulate the two architectures. The inputs of the framework consist of three components: application, architecture and technology parameters. The application is a program or benchmark (written in C/C++ language) possibly annotated to specify the parts that must be executed on the CIM core. The architecture parameters specify the simulated architecture such as instruction sets, cache size, cycles per instruction, etc. The technology parameters contain technology related info such as technology node, frequency and power consumption. Although there is a strong dependency between the technology and architectural parameters, this dependency is not shown in the simulation framework. Instead, we have simulated this dependency for the kernels outside this framework using HSpice by involving both the technological and architectural parameters. The outputs of the framework consist of the delay, energy and area of the simulated application.

The core of the framework consists of three phases as shown in Fig. 9, and explained next. Note that in order to support CIMX simulation, multiple existing tools are used and possibly modified, such as Intel Pin [61], MCProf [62], SiNUCA [63], mcPAT [64], NVSIM [65], Cacti [66].

Pre-processing phase: in the pre-processing phase, a profiling tool (i.e., MCProf [62]) is used to extract the memory-intensive parts of the application. Code 1 illustrates an example where the memory intensive part consists of a "for loop". To force the execution of this part in CIM core, pragma's are added (as of now manually) to the code as shown in Code 2 (lines 6 and 10). After profiling, SiNUCA trace generator [63], which uses Intel Pin [61], generates three types of traces: (i) static traces containing all instructions that have to be executed (ALU, memory, branch instructions), (ii) dynamic traces containing information related to the control flow such as branch instructions (i.e., taken or not taken), (iii) memory traces containing the memory addresses of read and write instructions. Note that the use of pragma's for CIM core instructions will result in updated traces. To realize this, we have extended the trace generator with the new CIM instructions, similarly as performed in [67]. For example, the OR operation of line 8 in Code 1 is replaced by an OR operation on CIM core (_cim_btw_or) in Code 2; the _cim_btw_or instruction is defined in the library of the trace generator and used to generate the updated traces for CIMX, similarly as performed for HMC instructions in [67].

Processing phase: in the processing phase, the SiCIM engine simulates the generated traces by executing them on the specified architecture (i.e. the processor type, cache size, DRAM size, CIM core configuration, etc.). For CIMX architecture, the non-accelerated part of the program is executed on the host processor, while the CIM core instructions are dispatched from the host processor to the CIM core. The SiCIM engine is based on the SiNUCA simulator [63] which we extended with CIM instructions. The SiCIM engine generates execution statistics of the simulation, which includes the number of cycles to execute the program, the number of memory accesses (cache and main memory), and the number of CIM instructions. The CIM core instructions are executed under a lock/unlock scheme to maintain data coherency similarly as in HIVE [63, 68]; this scheme ensures that data-dependent instructions are executed

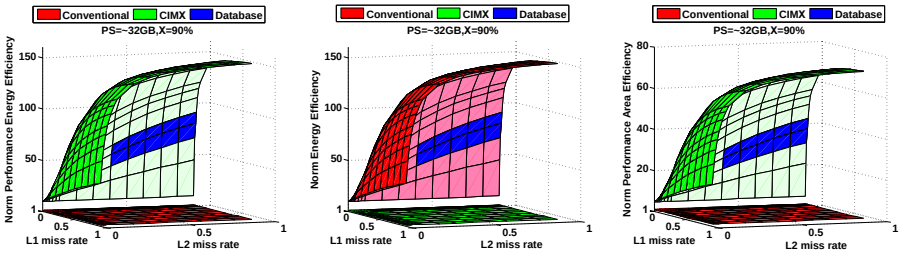


Figure 7: Analytical results of the performance energy, energy, and performance area efficiency metrics

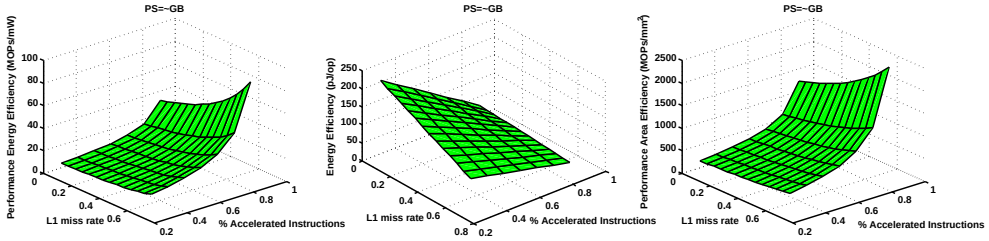


Figure 8: Analytical results for modified CIMX architecture

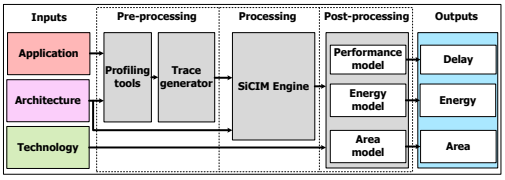


Figure 9: Simulation framework

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

Code 1: Source Code for Conventional Architecture

in the correct order, while data-independent instructions can be executed in parallel between the host and accelerator.

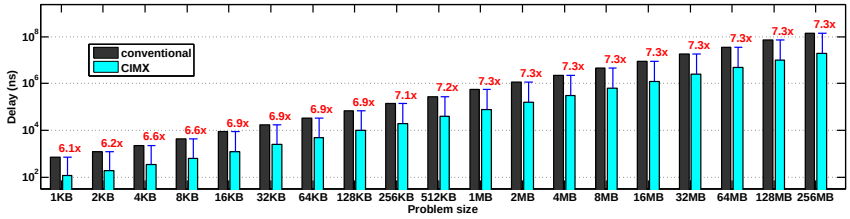
Post-processing phase: in the post-processing phase, the output statistics of the processing phase are used to calculate the delay, energy and area metrics using a combination of three tools: (i) mcPAT, a processor simulator [64], (ii) NVSIM, a non-volatile memory simulator [65], and (iii) Cacti, a cache and main memory (DRAM) simulator [66].

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

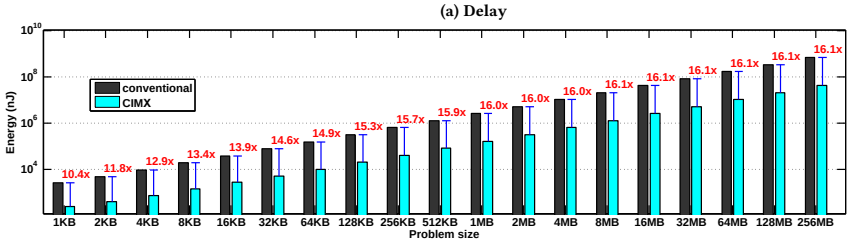
Code 2: Source Code for CIMX Architecture

5.2 Simulation Setup and Benchmarks

The simulation parameters of both architectures are summarized in Table 4. The conventional architecture is based on an Intel Xeon CPU E5-2650 that consists of a single 64-bit core, 32KB L1, 256KB L2 and 1GB DRAM. CIMX architecture has a similar build up, but instead has 500MB DRAM memory and a CIM core with a capacity of 500MB; note that data is assumed to be initialized in the DRAM and CIM core; hence, the cost of data initialization is not considered in the simulation time as it is not the focus of this work. This configuration is selected to maintain a reasonable simulation time and capability (i.e., Cacti, NVSIM, SiNUCA) for both architectures. The processor operates in both architectures on 64-bit data widths, while CIM core operates on 1024-Byte data widths, which is equivalent to the execution of 1024*8/64=128 CPU instructions in parallel.



(a) Delay



(b) Energy

Figure 10: Performance comparison for vector OR kernel

Table 4: Architecture configuration

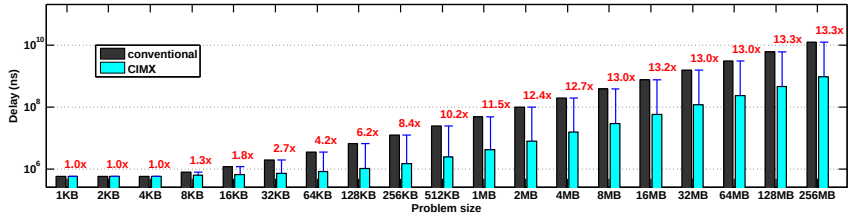
Configuration	Conventional Arch.	CIMX Arch.
Processor	1 core @ 2GHz	1 core @ 2GHz
Caches	32KB L1	32KB L1
	256KB L2	256KB L2
Main memory	1GB DRAM	500MB DRAM
		500MB CIM

In this paper, we simulate three frequently-used application kernels on both the conventional and CIMX architecture, without modifying the underlying algorithms. They are the *vector OR*, *XOR encryption*, and *QUERY SELECT* database kernel. They are the memory-intensive parts of vector processing [69], encryption [70] and database applications [71], respectively. They contain bulk bitwise operations that perform poorly on conventional architectures due to high cache miss rates [58, 72, 57, 73]. The three kernels are further described next.

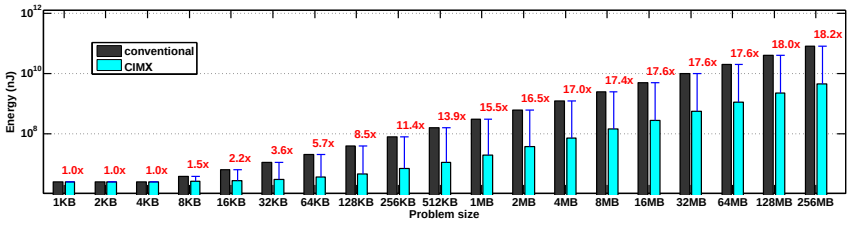
- The *vector OR* kernel performs an OR operation of two vectors. The size of the vectors represents the simulated problem size. In case the vector size is larger than the data-width of a single OR instruction (64-bit for CPU and 1024-byte for CIMX), multiple OR instructions are executed using full loop unrolling.
- The *XOR encryption* kernel performs an XOR operation of a string sequence and a predefined (secret) key. It is used for one-time-pad cryptography [70]. The size of the string sequence represents the problem size. In case the string sequence size is larger than the data-width of a single XOR instruction (64-bit for CPU and 1024-byte for CIMX), multiple XOR instructions are executed using a loop with multiple iterations.
- The *QUERY SELECT* database kernel performs the query-06 of the TPC-H benchmark [74], which includes 22 queries written in SQL language. In order to simulate these queries, the queries need to be converted to C/C++ language (to be fed into the simulation framework). As this process is not trivial, the manually created trace for query-06 selection with a 1GB database size presented in [75] is used. The query-06 performs compare instructions (as mentioned in Section 3.3) and checks if the requested data is available in the database or not; this includes conjunctions and disjunctions in large volume tables without join operations between tables. Note that a complete query execution may require other operations (e.g., min, max, etc.) that must be performed on the host processor side. However, in this work, we only focus on the kernel that performs bitwise operations.

5.3 Simulation Results

Fig. 10 shows the simulation results of the delay and energy consumption of both architectures for the vector OR kernel, respectively. The results clearly show that CIMX architecture outperforms the conventional architecture by 6x and 7x in terms of delay, and by 10x and 16x in terms of energy consumption. The benefits are mainly due to computation-in-memory (reduction in data transfer) and use of non-volatile technology. Moreover, the obtained delay and energy improvements strongly depend on the amount of supported parallelism; note that the more instructions can be executed in parallel, the higher the advantage of using CIMX. For example, with a larger problem size, the delay speed-up increases from 6.1x to 7.3x.



(a) Delay



(b) Energy

Figure 11: Performance comparison for XOR encryption kernel

Similarly, Fig. 11 shows the results of the XOR encryption kernel. For small problem sizes (less than 4KB), CIMX has no delay and energy improvement due to the overhead of non-accelerated part. However, the improvement is nearly 2x when the problem size is doubled. The XOR encryption is more complex than the vector OR kernel, as the non-accelerated part (i.e. the part that executes on the host CPU) consists of a much larger fraction of the total execution code. In case of large problem sizes, (i.e, larger than 512KB), CIMX architecture achieves more than one order of magnitude improvement both in terms of delay and energy in comparison with the conventional architecture. The speed-up results from two factors: (i) the exploited parallelism similarly as in the case of vector OR kernel, and (ii) the cache misses due to inefficient conditional instructions created by the loop.

Fig. 12a shows the results of the QUERY SELECT database kernel and in particular the compare instructions of query-06 from the TCP-H benchmark [74]. CIMX architecture achieves 4x higher performance as compared to the conventional architecture while consuming 6x less energy. The area of both architectures is shown in Fig. 12b. The improvement results from also the exploited parallelism similarly as in the case of vector OR kernel; however, as the database size does not fit within the CIM core, the speed-up is not as optimistic as the one obtained from the analytical model. With the use of non-volatile memory, the area of CIMX architecture is approximately 23% lower than that of the conventional architecture, resulting in a smaller footprint.

Overall, CIMX achieves nearly one order of magnitude improvement in delay and energy with a smaller footprint for the three simulated benchmarks. These improvements are mainly due to reduction in data movement and use of non-volatile technology.

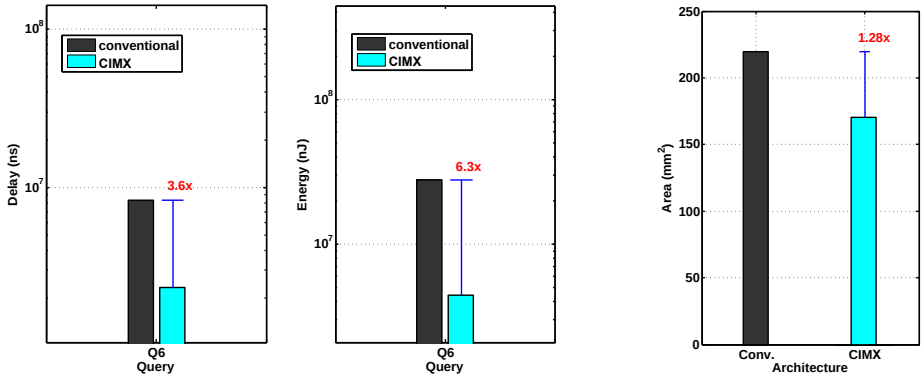
6 DISCUSSION

In this section, we first compare the analytical model and simulation framework. Thereafter, we address the limitations of this work and suggest possible solutions.

6.1 Comparison of Analytical and Simulation Results

The comparison is based on two aspects, i.e., the accuracy and required evaluation effort.

Accuracy: The analytical and simulation results show that CIMX architecture achieves significant improvements in terms of performance/delay, energy and area. The analytical results are generally more optimistic as they assume an ideal application with a uniform profile, where consecutive kernels can be accelerated without considering the communication between them. However, typically, a program contains various kernels, and hence, the improvements depend on the accelerated parts. For example, the XOR encryption kernel achieves no speed-up in case of small problem sizes (smaller than 32KB), but the speed-up increases to more than 13x in the case of a relatively larger problem size (256MB). In addition, the improvements of the simulator are limited due to the limited parallelism of CIM core; in the analytical model, the number of parallel CIM instructions can be easily modified, while this is harder to realize in the simulator. Another difference is the configuration of the number of cores between the model and the simulation framework. In the analytical model, we assumed that the conventional architecture consists of several cores. However, the three kernels we explored with the simulation framework were run on a single core; note that a lot of efforts are needed to parallelize them. Nevertheless, the results of both the analytical model and simulation framework show similar trends and the CIMX architecture achieves nearly one



(a) Compare instructions of QUERY SELECT database kernel **(b) Area metric**

Figure 12: Delay, energy and area comparison of conventional and CIMX architecture

order of magnitude improvement in terms of speed up and energy reduction.

Evaluation effort: Even though the simulation framework provides more realistic results than the analytical model, it is time-consuming to port and execute an application on the simulation framework. First, it requires (as of now) manual insertions of pragma's and the user has to specify the CIM instructions. Second, as the simulation framework is based on a cycle-accurate simulator, even relative simple programs containing less than a hundred lines of code are transformed to traces with thousands/millions of instructions, which take hours if not days to simulate. In contrast, the analytical model provides a quick design exploration and gives a quick insight where the highest improvements can be obtained. Hence, as a lot of effort is required to run the application on the simulation platform, the analytical evaluation can be used to quickly analyze the performance estimation. This requires however accurate inputs to the model such as L1 and L2 cache miss rates. They can be retrieved for example by performing a workload characterization by for example using the Linux tool perf [76] or Intel *emon* that has been used to characterize database applications [57].

6.2 Limitations and Future Work

The proposed architecture is still an on-going research. Therefore, there are still a lot of issues to be addressed as explained next.

Mapping of complex functions on CIM core: More research is required to map complex functions on the CIM core. First, it would be ideal if arithmetic operations could be executed on CIM core. This reduces the communication between processor and main memory even further. Moreover, for database applications, operations such as *AVERAGE*, *MIN*, and *MAX* can also be performed on CIM core when these arithmetic operations are integrated; this will subsequently improve the overall performance. In case, a query has a complex operation that cannot be performed by CIM core, the results of the compare instructions have to be transferred back to the host processor to perform this complex operation; it impacts the performance if this operation happens frequently. Therefore, it is essential to implement essential complex operations on CIM core. Third, to increase the performance even more, the instruction set can be extended with macro instructions [77] such as dot product or matrix multiplication [78]. Another approach is to integrate the arithmetic units in the CMOS layer near the memory controller, similarly as carried out in near-data computing [71]. Current prototypes have already demonstrated this approach by stacking memristors on top of a CMOS layer [79, 80]. Moreover, future work should also focus on the instruction set, as the overhead to implement and execute the in-memory instructions are often overlooked in current designs.

Memory controller and communication: The memory controller and the communication between processor and accelerator should be investigated more accurately. Optimizing the inter-communication (i.e., between processor and CIM core) and intra-communication (i.e., within CIM core) is of great importance. Similar work in the field has provided solutions to intra-communication among banks and arrays [81, 6]. However, design exploration and optimization of such components still needs further investigation and could significantly improve the overall performance and robustness [82, 77, 83, 84, 85].

CIMX architecture improvements: CIMX was designed with the intention to minimize the required number of modifications to conventional architectures. It is worth noticing that much higher performance and energy improvements can be achieved in case more radical architectures are considered, such as the one in Fig. 3d. As the CIM core stores a huge amount of data, the main DRAM might become superfluous. This will eliminate the expensive off-chip communication and enable higher performance and energy improvements. With expected technology developments, it may become feasible to allocate more and more data on the same chip. Current prototypes have demonstrated resistive memories with GBs of data [36, 86, 87]. In addition, CIMX in general and scouting logic in particular are feasible to be implemented using other memory

technologies such as SRAM and DRAM [81, 88]. Note that CIMX is based on resistive memories which generally have a low power consumption and small footprint. In order to implement CIMX using other memristor devices such as phase change memory (PCM) [89], a thorough investigation is required to estimate CIMX performance. A benefit of using PCM is that it is more robust with multi-bit cells. [90].

Application exploration and simulator automation: Although we considered three kernels, more applications have to be explored for CIMX architecture. Note that the considered CIMA architecture contains an accelerator that can only perform logical operations. Hence, it is worth exploring various applications with a high percentage of logical operations. In fact, some previous work has proposed quite a lot of applications with these characteristics such as database processing, graph processing, security encryption, and bio-sequencing [6, 81, 33, 45, 47, 91]. In this paper, we have shown potential examples such as *QUERY SELECT* kernel (database applications) and *XOR encryption* kernel (security encryption). Due to some manual processes in the simulation, it is currently not straightforward to map more potential applications. Nevertheless, exploring potential applications is essential once the simulation framework is further automated.

7 CONCLUSION

In this paper, we proposed a Computation-In-Memory Accelerator architecture that uses scouting logic to perform in-memory logical operations. Our analytical and simulation results show that CIMX architecture outperforms conventional architectures with at least one order of magnitude improvements in terms of delay and energy while consuming less chip area. These improvements can be realized for big data applications (such as those with a low data locality, large bulk bitwise operations), which perform poorly on the general purpose computers. Therefore, the proposed architecture is a viable candidate for high performance application-specific computers, embedded and/or low power system.

ACKNOWLEDGMENT

The results in this paper have been obtained in the framework of the project "Computation-in-memory architecture based on resistive devices" (MNEMOSENE), which has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 780215.

The authors would also like to thank Marco Antonio Zanata Alves (UFPR), Luigi Carlo (UFRGS), and Paulo Cesar Santos (UFRGS) for many fruitful discussions and support on the simulation framework and TCP-H benchmarks.

REFERENCES

[1] David A Patterson. 2006. Future of computer architecture. In *Berkeley EECS Annual Research Symposium (BEARS), College of Engineering, UC Berkeley, US.*
[2] ITRS. 2010. ITRS ERD report. http://www.itrs.net.
[3] Said Hamdioui et al. 2015. Memristor based computation-in-memory architecture for data-intensive applications. In *DATE*. IEEE.
[4] Pierre-Emmanuel Gaillardon et al. 2016. The programmable logic-in-memory (plim) computer. In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 427–432.
[5] Amir Morad et al. 2016. Resistive gp-simd processing-in-memory. *ACM Transactions on Architecture and Code Optimization (TACO)*, 12, 4, 57.
[6] Shuangchen Li et al. 2016. Pinatubo: a processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories. In *Design Automation Conference (DAC), 2016 53nd ACM/EDAC/IEEE*. IEEE, 1–6.
[7] J Joshua Yang *et al.* 2013. Memristive devices for computing. *Nat. Nano.*
[8] Manuel Le Gallo et al. 2017. Mixed-precision memcomputing. *arXiv preprint arXiv:1701.04279*.
[9] Hoang Anh Du Nguyen et al. 2015. Computation-in-memory based parallel adder. In *NANOARCH*. IEEE.
[10] Adib Haron et al. 2016. Parallel matrix multiplication on memristor-based computation-in-memory architecture. In *High Performance Computing & Simulation (HPCS), 2016 International Conference on*. IEEE, 759–766.
[11] Leonid Yavits et al. 2015. Resistive associative processor. *CAL*.
[12] Garrett S Rose et al. 2012. Leveraging memristive systems in the construction of digital logic circuits. *Proceedings of the IEEE*, 100, 6, 2033–2049.
[13] Ligang Gao et al. 2013. Programmable cmos/memristor threshold logic. *TNANO*.
[14] E Linn et al. 2012. Beyond von neumannâĂŤlogic operations in passive crossbar arrays alongside memory operations. *Nanotechnology*.
[15] Shahar Kvatinsky *et al.* 2014. Memristor-based material implication (imply) logic: design principles and methodologies. *TVLSI*.
[16] Ioannis Vourkas et al. 2012. A novel design and modeling paradigm for memristor-based crossbar circuits. *Nanotechnology*, 11, 6, 1151–1159.
[17] Shahar Kvatinsky et al. 2012. Mrl memristor ratioed logic. In *CNNA*. IEEE.
[18] Lei Xie et al. 2016. Boolean logic gate exploration for memristor crossbar. In *DTIS*. IEEE.
[19] Ying Wang et al. 2015. Propram: exploiting the transparent logic resources in non-volatile memory for near data computing. In *Proceedings of the 52nd Annual Design Automation Conference*. ACM, 47.
[20] L. Xie et al. 2017. Scouting logic: a novel memristor-based logic design for resistive computing. In *IEEE Computer Society Annual Symposium on VLSI*. IEEE, 1–6.
[21] HA Du Nguyen et al. 2017. Memristive devices for computing: beyond cmos and beyond von neumann. In *Very Large Scale Integration (VLSI-SoC), 2017 IFIP/IEEE International Conference on*. IEEE, 1–10.
[22] Shahar Kvatinsky et al. 2014. MagicâĂŤmemristor-aided logic. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 61, 11, 895–899.
[23] Julien Borghetti *et al.* 2010. Memristive switches enable stateful logic operations via material implication. *Nature*, 464, 7290, 873–876.
[24] Virat Agarwal et al. 2010. Scalable graph exploration on multicore processors. In *SC*. ACM.
[25] Jerry Chou et al. 2011. Parallel index and query for large scale data analysis. In *SC*. ACM.
[26] Anne Siemon et al. 2014. Simulation of tao x-based complementary resistive switches by a physics-based memristive model. In *ISCAS*. IEEE.
[27] Feng Miao *et al.* 2011. Anatomy of a nanoscale conduction channel reveals the mechanism of a high-performance memristor. *Advanced Materials*.
[28] Nanoscale Integration et al. 2006. Predictive transistor model. http://ptm.asu.edu/.
[29] Meng-Fan Chang et al. 2012. Endurance-aware circuit designs of nonvolatile logic and nonvolatile sram using resistive memory (memristor) device. In *Design Automation Conference (ASP-DAC), 2012 17th Asia and South Pacific*. IEEE, 329–334.
[30] Shruti Patil et al. 2011. Performing bitwise logic operations in cache using spintronics-based magnetic tunnel junctions. In *Proceedings of the 8th ACM International Conference on Computing Frontiers*. ACM, 33.
[31] Jue Wang et al. 2014. Endurance-aware cache line management for non-volatile caches. *ACM Transactions on Architecture and Code Optimization (TACO)*, 11, 1, 4.
[32] Chris H Kim et al. 2003. A forward body-biased-low-leakage sram cache: device and architecture considerations. In *Low Power Electronics and Design, 2003. ISLPED'03. Proceedings of the 2003 International Symposium on*. IEEE, 6–9.
[33] Vivek Seshadri et al. 2015. Fast bulk bitwise and and or in dram. *IEEE Computer Architecture Letters*, 14, 2, 127–131.
[34] Youngdon Choi et al. 2012. A 20nm 1.8 v 8gb pram with 40mb/s program bandwidth. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2012 IEEE International*. IEEE, 46–48.
[35] Richard Fackenthal et al. 2014. 19.7 a 16gb reram with 200mb/s write and 1gb/s read in 27nm technology. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014 IEEE International*. IEEE, 338–339.
[36] Tz-yi Liu et al. 2014. A 130.7mm 2-layer 32-gb reram memory device in 24-nm technology. *IEEE Journal of Solid-State Circuits*, 49, 1, 140–153.
[37] Chun Jason Xue et al. 2011. Emerging non-volatile memories: opportunities and challenges. In *Hardware/Software Codesign and System Synthesis (CODES+ ISSS), 2011 Proceedings of the 9th International Conference on*. IEEE, 325–334.
[38] Simone Raoux et al. 2008. Phase-change random access memory: a scalable technology. *IBM Journal of Research and Development*, 52, 4.5, 465–479.
[39] Takayuki Kawahara. 2010. Scalable spin-transfer torque ram technology for normally-off computing. *IEEE Design and Test of Computers*, 1, 52–63.

[40] Zoha Pajouhi et al. 2015. Device/circuit/architecture co-design of reliable stt-mram. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*. EDA Consortium, 1437–1442.
[41] Omer Zilberberg et al. 2013. Phase-change memory: an architectural perspective. *ACM Computing Surveys (CSUR)*, 45, 3, 29.
[42] Benjamin C Lee et al. 2009. Architecting phase change memory as a scalable dram alternative. In *ACM SIGARCH Computer Architecture News* number 3. Volume 37. ACM, 2–13.
[43] Stamatis Vassiliadis et al. 2004. The molen polymorphic processor. *IEEE transactions on computers*, 53, 11, 1363–1375.
[44] Razvan Nane et al. 2014. High-level synthesis in the delft workbench hardware/software co-design tool-chain. In *Embedded and Ubiquitous Computing (EUC), 2014 12th IEEE International Conference on*. IEEE, 138–145.
[45] Junwhan Ahn et al. 2016. A scalable processing-in-memory accelerator for parallel graph processing. *ACM SIGARCH Computer Architecture News*, 43, 3, 105–117.
[46] Amirali Boroumand et al. 2017. Lazypim: an efficient cache coherence mechanism for processing-in-memory. *IEEE Computer Architecture Letters*, 16, 1, 46–50.
[47] Gary Benson et al. 2013. A bit-parallel, general integer-scoring sequence alignment algorithm. In *Annual Symposium on Combinatorial Pattern Matching*. Springer, 50–61.
[48] Pawel Gepner et al. 2012. Performance evaluation of intel xeon e5-2600 family cluster using scientific and engineering benchmarks. In *Parallel Distributed and Grid Computing (PDGC), 2012 2nd IEEE International Conference on*. IEEE, 230–235.
[49] Hadi Esmaeilzadeh et al. 2011. Dark silicon and the end of multicore scaling. In *Computer Architecture (ISCA), 2011 38th Annual International Symposium on*. IEEE, 365–376.
[50] Tadaaki Yamauchi et al. 1997. A single chip multiprocessor integrated with dram. In *Workshop on Mixing Logic and DRAM, held at the 24th International Symposium on Computer Architecture*.
[51] Allan Hartstein et al. 2006. Cache miss behavior: is itâĹŽ 2? In *Proceedings of the 3rd conference on Computing frontiers*. ACM, 313–320.
[52] Allan Hartstein et al. 2008. On the nature of cache miss behavior: is itâĹŽ 2. *The Journal of Instruction-Level Parallelism*, 10, 1–22.
[53] Cristina Meinhardt et al. 2013. Finfet basic cells evaluation for regular layouts. In *Circuits and Systems (LASCAS), 2013 IEEE Fourth Latin American Symposium on*. IEEE, 1–4.
[54] Anish Muttreja et al. 2007. Cmos logic design with independent-gate finfets. In *Computer Design, 2007. ICCD 2007. 25th International Conference on*. IEEE, 560–567.
[55] David M Fried et al. 2003. A fin-type independent-double-gate nfet. In *Device Research Conference, 2003*. IEEE, 45–46.
[56] S Thoziyoor et al. 2008. Cacti 5.3. *HP Laboratories, Palo Alto, CA*.
[57] Anastassia Ailamaki et al. 1999. Dbmss on a modern processor: where does time go? In *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK* number DIAS-CONF-1999-001, 266–277.
[58] Leonidas I Kontothanassis et al. 1994. Cache performance in vector supercomputers. In *Proceedings of the 1994 ACM/IEEE conference on Supercomputing*. IEEE Computer Society Press, 255–264.
[59] Friman Sánchez et al. 2006. Performance analysis of sequence alignment applications. In *Workload Characterization, 2006 IEEE International Symposium on*. IEEE, 51–60.
[60] Mark Woh et al. 2009. Anysp: anytime anywhere anyway signal processing. In *ACM SIGARCH Computer Architecture News* number 3. Volume 37. ACM, 128–139.
[61] Chi-Keung Luk et al. 2005. Pin: building customized program analysis tools with dynamic instrumentation. In *Acm sigplan notices* number 6. Volume 40. ACM, 190–200.
[62] I. Ashraf et al. 2014. MCProf: Memory and Communication Profiler. Technical Report, Computer Engineering Lab CE-TR-2014-02. Delft University of Technology, Delft, Netherlands.
[63] M. A. Z. Alves et al. 2015. Sinuca: a validated micro-architecture simulator. In *2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems*, 605–610. DOI: 10.1109/HPCC-CSS-ICESS.2015.166.
[64] S. Li et al. 2009. Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures. In *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 469–480.
[65] X. Dong et al. 2012. Nvsim: a circuit-level performance, energy, and area model for emerging nonvolatile memory. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 31, 7, 994–1007. ISSN: 0278-0070.
[66] S. J. E. Wilton et al. 1996. Cacti: an enhanced cache access and cycle time model. *IEEE Journal of Solid-State Circuits*, 31, 5, 677–688.
[67] Aline Santana Cordeiro et al. 2017. Intrinsics-hmc: an automatic trace generator for simulations of processing-in-memory instructions. *XVIII Simpósio em Sistemas Computacionais de Alto Desempenho-WSCAD*.
[68] Marco AZ Alves et al. 2016. Large vector extensions inside the hmc. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2016*. IEEE, 1249–1254.
[69] Brian R Gaeke et al. 2001. Memory-intensive benchmarks: iram vs. cache-based machines. In *Parallel and Distributed Processing Symposium., Proceedings International, IPDPS 2002, Abstracts and CD-ROM*. IEEE, 7–pp.
[70] Jun Yang et al. 2005. Improving memory encryption performance in secure processors. *IEEE Transactions on Computers*, 54, 5, 630–640.
[71] Paulo C Santos et al. 2016. Exploring cache size and core count tradeoffs in systems with reduced memory access latency. In *Parallel, Distributed, and Network-Based Processing (PDP), 2016 24th Euromicro International Conference on*. IEEE, 388–392.
[72] Bruce Schneier. 2007. *Applied cryptography: protocols, algorithms, and source code in C*. john wiley & sons.
[73] Kimberly Keeton et al. 1998. Performance characterization of a quad pentium pro smp using oltp workloads. In *ACM SIGARCH Computer Architecture News* number 3. Volume 26. IEEE Computer Society, 15–26.
[74] Transaction Processing Performance Council. 2015. Tpc-h, a decision support benchmark. (2015).
[75] Paulo C Santos et al. 2017. Operand size reconfiguration for big data processing in memory. In *Proceedings of the Conference on Design, Automation & Test in Europe*. European Design and Automation Association, 710–715.
[76] Linux. 2015. Perl linux. https://perf.wiki.kernel.org.
[77] Hoang Anh Du Nguyen et al. 2017. On the implementation of computation-in-memory parallel adder. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25, 8, 2206–2219.
[78] Jintao Yu et al. 2018. Memristive devices for computation-in-memory. In *Design, Automation and Test in Europe DATE*.
[79] Dmitri B Strukov et al. 2010. Hybrid cmos/memristor circuits. In *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*. IEEE, 1967–1970.
[80] Kwang-Ting Tim Cheng et al. 2012. 3d cmos-memristor hybrid circuits: devices, integration, architecture, and applications. In *Proceedings of the 2012 ACM international symposium on International Symposium on Physical Design*. ACM, 33–40.
[81] Vivek Seshadri et al. 2017. Ambit: in-memory accelerator for bulk bitwise operations using commodity dram technology. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 273–287.
[82] Ioannis Vourkas et al. 2016. Alternative architectures toward reliable memristive crossbar memories. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 24, 1, 206–217.
[83] Rotem Ben Hur et al. 2016. Memristive memory processing unit (mpu) controller for in-memory processing. In *Science of Electrical Engineering (ICSEE), IEEE International Conference on the*. IEEE, 1–5.
[84] Peyman Pouyan et al. 2014. Reliability challenges in design of memristive memories. In *CMOS Variability (VARI), 2014 5th European Workshop on*. IEEE, 1–6.
[85] Amirali Ghofrani et al. 2013. Towards data reliable crossbar-based memristive memories. In *Test Conference (ITC), 2013 IEEE International*. IEEE, 1–10.
[86] Akifumi Kawahara et al. 2013. An 8 mb multi-layered cross-point reram macro with 443 mb/s write throughput. *IEEE Journal of Solid-State Circuits*, 48, 1, 178–185.
[87] C Park et al. 2015. Systematic optimization of 1 gbit perpendicular magnetic tunnel junction arrays for 28 nm embedded stt-mram and beyond. In *Electron Devices Meeting (IEDM), 2015 IEEE International*. IEEE, 26–2.
[88] Arun Subramaniyan et al. 2017. Cache automaton. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture* (MICRO-50 '17). ACM, Cambridge, Massachusetts, 259–272. ISBN: 978-1-4503-4952-9. DOI: 10.1145/3123939.3123986.
[89] Stefano Ambrogio et al. 2018. Equivalent-accuracy accelerated neural-network training using analogue memory. *Nature*, 558, 7708, 60.
[90] Can Li et al. 2018. Analogue signal and image processing with large memristor crossbars. *Nature Electronics*, 1, 1, 52.
[91] JongWook Han et al. 1999. Optical image encryption based on xor operations. *Optical Engineering*, 38, 1, 47–55.

4

CIRCUIT LEVEL

This chapter presents a logic automation tool and interconnect network schemes. In terms of logic automation tool, it proposes a generic synthesis framework which can utilizes various logic design methods to generate memristor circuits from hardware description language (HDL) of a function. Subsequently, it shows two case studies of 2-bit counter and 8-bit adder to illustrate the framework. In terms of interconnect network, it proposes three schemes: direct (using only copy operation within memristor crossbar), indirect (using CMOS circuits outside of memristor crossbar), and hybrid (using the combination of direct and indirect method). Thereafter, it uses parallel addition as a case study to describe an interconnect network for in-memory computing architectures. All three proposed interconnect network schemes are evaluated and discussed.

The content of this chapter is based on the following research article:

1. **H. A. Du Nguyen**, L. Xie, M. Taouil, S. Hamdioui and K. Bertels, "Synthesizing HDL to memristor technology: A generic framework," 2016 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH), Beijing, 2016, pp.43-48.
2. **H. A. Du Nguyen**, L. Xie, J. Yu, M. Taouil and S. Hamdioui, "Interconnect networks for resistive computing architectures," 2017 12th International Conference on Design & Technology of Integrated Systems In Nanoscale Era (DTIS), Palma de Mallorca, Spain, 2017, pp. 1-6.

4.1. PROBLEM STATEMENT

As memristive devices are still immature, there are still open questions on circuit synthesis flow and interconnect network. First, the circuit synthesis flow is essential to provide basic components for architectures. Second, it is essential to find the best method to implement interconnect network between the basic components. This chapter will explore these aspects.

Circuit Synthesis Flow: Many memristive logic designs have been proposed to provide manual designs of basic components. However, these manual designs are limited in terms of functionality and performance. In fact, it is difficult to tune those basic components for an architecture targeting a specific application. Hence, a generic synthesis tool that is not limited to a logic design methodologies is required for architects to explore various basic components easily.

Interconnect Network: Several existing work has implemented basic operations for the interconnect network. However, large scale interconnect networks among basic components have not been explored. Therefore, it is essential to explore different possible interconnect networks for large scale circuits.

4.2. MAIN CONTRIBUTIONS

The main contributions in the above aspects are as follows.

Circuit Synthesis Flow [77]: We propose a generic synthesis framework that generates memristor circuits and its controllers from a hardware description language (HDL) file, as shown in Fig. 4.1. As HDL files are frequently used in FPGA, there are a large existing library of these basic components for exploration. In addition, the framework can be incorporated with various memristive logic design methods; hence, it can generate a wide range of components that are flexible enough to be explored and tuned for specific applications. In this work, two case studies of a 2-bit counter and 2-bit adder are used to illustrated the framework.

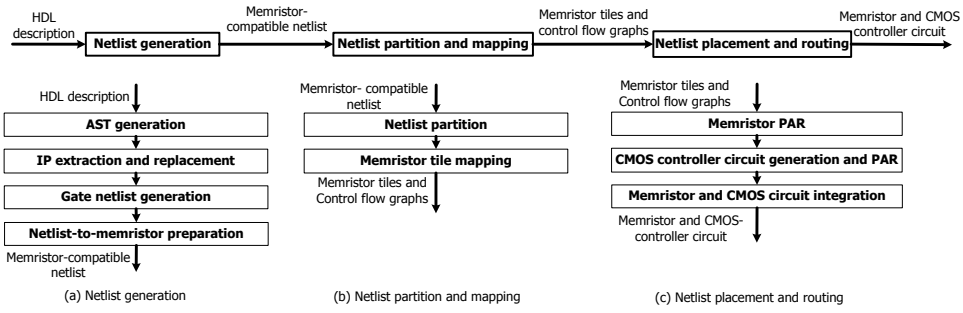


Figure 4.1: Proposed synthesis framework

Interconnect Network [78]: We propose three interconnect network schemes that use

only memristors, only CMOS circuits or both. The memristor interconnect network is based on the basic operation *copy* proposed in the previous work [79]. The CMOS interconnect network utilizes the CMOS circuits (i.e., controllers) to read from the source memristor and write to the destination memristor. The hybrid interconnect network use memristors for direct copy between two memristors on the same row or column, and CMOS transmission gates to connect between two memristors located on different rows and/or columns. A parallel adder is used as case study to illustrate the three proposed interconnect network schemes. The results show that a hybrid interconnect network scheme has the highest efficiency in terms of delay, energy and area, as shown in Fig. 4.2. Hence, it is essential to utilize both memristor operations and CMOS devices to build an effective interconnect network scheme for resistive computing architectures.

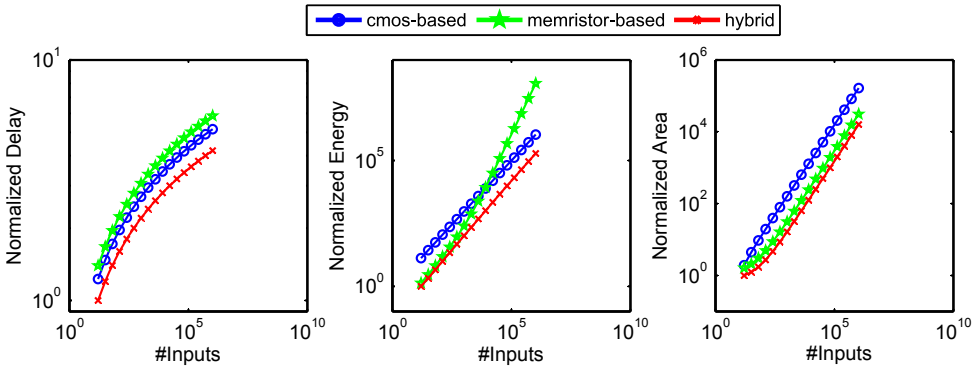


Figure 4.2: Performance of Proposed Interconnect Network Schemes

Synthesizing HDL to Memristor Technology: A Generic Framework

H.A. Du Nguyen, Lei Xie, Mottaqiallah Taouil, Said Hamdioui, Koen Bertels
Laboratory of Computer Engineering, Delft University of Technology
Mekelweg 4, 2628 CD Delft, The Netherlands
H.A.DuNguyen@tudelft.nl

ABSTRACT

Memristors are emerging devices with huge potentials. It has been shown that they can be used not only to design non-volatile memories, but also logic circuits. In the latter, memristor devices are stacked on a CMOS circuit which generates the required control signals needed by the memristors to perform the required functionality. This paper sets a step towards automating this process; it proposes a generic synthesis framework to map logic circuits on memristor crossbar. The framework takes HDL descriptions as input and generates both its memristor circuitry and its associated CMOS control. The framework consists of three phases: (i) netlist generation, (ii) partition and mapping, and (iii) placement and routing. To illustrate the framework, a combinational and a sequential circuit are investigated. The results are validated using HSPICE simulations.

1. INTRODUCTION

As CMOS technology is reaching the inherent physical limits of down-scaling, sub-20nm technology nodes suffer from significant challenges [1–4], such as saturated performance gain, increased static power consumption, decreased reliability, complicated fabrication process, and increased manufacturing and testing cost. These issues have led to a research growth on emerging technologies as an alternative or a complementary one to CMOS. Memristor technology is one of the promising candidates due to its high scalability, zero static energy and CMOS compatibility [5,6]. Furthermore, memristors can be used to implement both processing and storage elements [7]. Since the memristor device was demonstrated by HP in 2008 [7], a lot of research on memristor based logic designs using Boolean logic [8,9], implication logic [10,11], threshold logic [12] etc. have been proposed. In addition, various computing architectures based on memristors [13–16] have been reported. Although there are many challenges still to be solved, memristor crossbars have shown to have huge potentials for large scale designs thanks to their high density [14,17,18]. However, hand-made designs will not allow the exploration of such potentials for large-scale circuits and architectures. In addition, existing CMOS synthesis tools cannot be used for memristor crossbar designs; a memristor is a passive device that uses resistance (instead of voltage) to represent data, and memristor crossbar designs need to be configured through control signals generated by CMOS circuitry. Therefore, there is a need of automation tools for memristor circuits in order to fully explore their potentials. The framework should support various memristor logic design methods, and must be as intuitive as a CMOS synthesis framework, i.e. it must start with a hardware description (in Verilog or VHDL).

Currently, only limited work has been reported about the automation of the memristor-based designs such as implication logic [19], threshold logic [20], Boolean logic [21] used for arithmetic circuits, or neuromorphic computing circuits [22]. In [19], the authors proposed a method to map Boolean functions on implication logic based memristor circuits and to reduce the number of computation steps. In [20], the authors proposed an algorithm to map Boolean functions on threshold based memristor circuits with the target to reduce energy consumption in comparison with LUT-based FPGA threshold logic. In [21], the authors proposed a formal method to map Boolean functions on Boolean logic based memristor circuits while reducing sneak path currents. These work focus mostly on the mapping of small Boolean functions on a memristor circuit. In addition, each of them targets a specific logic type only. Another major limitation is that CMOS controllers are typically ignored, while they are fundamental for the place-and-route phase as they impact the interconnection, power, area, delay, etc. In [22], the authors proposed an efficient partitioning, placement and routing of memristor crossbars used for neuromorphic computing systems. However, this work only takes neural network models as input, hence, it is not flexible for all computational problems. In conclusion, exploring large design based on memristor technology needs automation tools.

This paper sets a step towards the automation of logic synthesis for memristor circuits. It proposes a generic synthesis framework, which may target all memristor logic types, memristor circuit design methods and memristor circuit structures (discrete memristors and memristor crossbars [18]). The framework uses a Hardware Description Language (HDL) file as input and synthesizes it to a memristor circuit with its corresponding CMOS control circuits. To verify the generated circuits, the framework produces additional HSPICE simulation files and also reports several performance metrics such as delay, area, energy. To the best of our knowledge,

Nanoarch '16, July 18-20, 2016, Beijing, China

DOI: http://dx.doi.org/10.1145/2950067.2950098

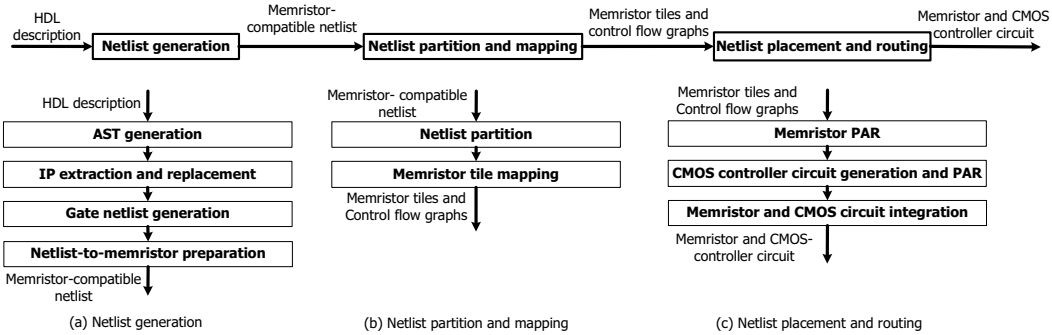


Figure 1: Proposed synthesis framework

our work is the first to present a complete synthesis framework for memristor circuits. The main contributions are:

- a generic memristor synthesis framework from HDL to memristor circuits.
- validation of the entire framework using two case studies: a 2-bit counter and 8-bit adder.

The rest of this paper is structured as follows. Section 2 briefly describes the proposed synthesis framework. Section 3, 4, and 5 present the three main phases of the framework: netlist generation, netlist partition and netlist placement and routing. Section 6 provides a case study of an 8-bit adder, and its verification using HSPICE simulation. Section 7 discusses the framework's opportunities and challenges. Finally, Section 8 concludes this paper.

2. PROPOSED SYNTHESIS FRAMEWORK

The framework synthesizes a HDL file (Verilog, VHDL, etc.) and generates a placed and routed memristor circuit, and its corresponding CMOS controller. In this paper, we will focus on the memristor crossbar; nevertheless, the framework is also applicable to discrete memristors [18].

Fig. 1 shows the main components of the framework consisting of three phases: (i) netlist generation, (ii) netlist partition and mapping, and (iii) netlist placement and routing.

The netlist generation phase translates a HDL file to a memristor-compatible netlist, which contains conventional logic gates with predefined memristor-based IPs.

During the netlist partition and mapping (PAM) phase, the memristor-compatible netlist is first partitioned. Subsequently, each partition is translated to a memristor circuit. Along this process, a control flow graph (CFG) is generated to keep track of the execution order of the partitioned netlist.

In the last phase, the netlist placement and routing (PAR), first all the memristor partitions are placed on the crossbar. Then, CMOS controllers are generated based on the CFG of the previous phase. Thereafter, the CMOS controllers are placed in such a way that the crossbar is stacked on the CMOS part, and the overall communication and area are optimized. Finally, the placed memristor partitions are routed and connected to the CMOS controllers.

In the next section, the three phases will be elaborated. We will do that while using an example in order to provide more insights in the framework. The example is a 2-bit counter, which is a sequential circuit. The HDL description of this counter is shown in Fig. 2. The 2-bit counter has a

```
 1. module counter (clk, rst, D, Q);
 2. input clk, rst;
 3. input  [1:0] D;
 4. output [1:0] Q;
 5. reg    [1:0] tmp;
 6.   always @(posedge clk)
 7.     begin
 8.       if (rst)
 9.         tmp = D;
10.       else
11.         tmp = tmp + 1'b1;
12.     end
13.  assign Q = tmp;
14. endmodule
```

Figure 2: Verilog description of a 2-bit counter

2-bit initial value (D), clock (clk), and reset (rst) as inputs and 2-bit output (Q).

3. NETLIST GENERATION

The netlist generation starts with a HDL file and produces a memristor-compatible netlist with gates and memristor IPs. Fig. 1a shows the four main steps of the netlist generation: (i) Abstract Syntax Tree (AST) generation, (ii) IP extraction and replacement, (iii) gate netlist generation, and (iv) netlist-to-memristor preparation.

In step 1, inspired by CMOS circuit synthesis, i.e., the Yosys synthesis framework [23], the netlist generation starts with AST generation. The AST is a tree representation of the source code, and includes operations (addition, subtraction, division, etc.) and processes (sequential and combinational). Fig. 3 shows a graphic representation of the 2-bit counter AST (its Verilog description in Fig. 2). The AST consists of five declarations (*WIRE* nodes), one process (*ALWAYS* node) and one assignment (*ASSIGN* node). In each node, there are corresponding children nodes depending on the statement in the Verilog description. For example, the statement $assign\ Q = tmp$; (line 13 of Fig. 2) is converted to an *ASSIGN* node with two children nodes with identifiers $ID{:}Q$ and $ID{:}tmp$ (see top-right part of Fig. 3).

In step 2, IP extraction and replacement, specific nodes in the AST (e.g., *ADD*, *SUB*) are identified and may be replaced by predefined memristor IPs. This is analogue to CMOS circuit synthesis where a node in the AST may be implemented by specific library components (e.g., low-power adders or fast multipliers). A memristor IP has a similar to function as a regular IP, but is implemented by memristor circuits. For example, the node *ADD* shown in Fig. 3 is identified and replaced by a memristor design of a 1-bit

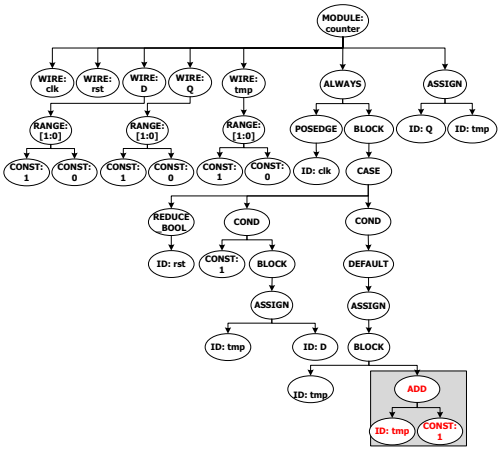
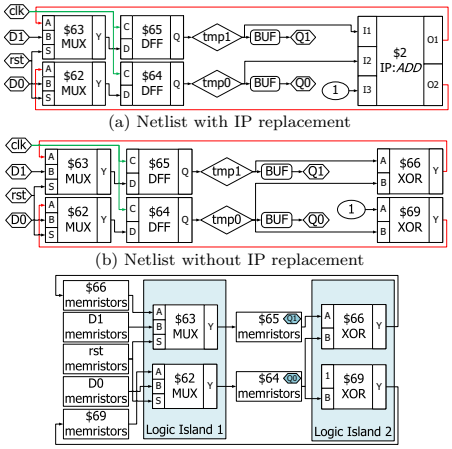


Figure 3: Abstract syntax tree with IP replacement



(a) Netlist with IP replacement

(b) Netlist without IP replacement

(c) Generated memristor-compatible netlist

Figure 4: Processed netlist step by step

adder (an available library component). This prevents such nodes from being further processed and transformed into gates which may lead to a sub-optimal memristor circuit.

In step 3, the gate netlist generation, a gate netlist is created from the AST representation in a similar way to the CMOS circuit synthesis flow. In this step, multiple synthesis optimization techniques such as constant reduction [23], logic optimization [24], etc. can be reused. Fig. 4a shows the 2-bit counter gate netlist. Notice that the node *ADD* has been replaced with a memristor design *"IP:ADD"*. In case the node *ADD* is not replaced with the memristor IP, the gate netlist shown in Fig. 4b would have been generated. We discuss the rest of the framework using the gate netlist of Fig. 4b, which is without the memristor IPs.

The final step modifies the gate netlist into a memristor-compatible netlist; a memristor-compatible netlist is a circuit described by logic gates and predefined memristor-based IPs that can be mapped directly to a memristor circuit (more details in Section 4). As memristors are residing in the crossbar, they are directly controlled from the CMOS controller. Therefore, there is no need for a control-related circuit such as a clock tree in the crossbar. Hence, the netlist-to-memristor preparation step handles these signals by moving them to CMOS controllers and removing related clock logic out of the netlist. As the clock logic is removed from the crossbar, synchronization is carried out by CMOS controller. This ensures the functionality of the circuit. Note that the synthesis framework should support both sequential and combinational circuits. A possible solution to maintain the synchronization is to reorganize the logic inside the netlist based on the pipelines in the design. The logic between two pipeline stages is grouped into a memristor logic island and the flip-flops into memristor-based registers. All logic islands are synchronized, execute in parallel, and update their results to the corresponding registers simultaneously to ensure the robustness of the circuit. All inputs, outputs and temporary values are stored in memristor-based registers. The constant values are integrated into logic islands, which will be optimized further in the mapping phase. Fig. 4c shows the memristor-compatible netlist of the 2-bit counter. In this netlist, the logic blocks are separated into two logic islands. As memristors are non-volatile devices, all the memristor-based registers can be embedded into the logic islands; for example, inputs/outputs ($D0$, $D1$, \$66, \$69) and temporary values (\$64, \$65) are shifted into Logic Island 1 and Logic Island 2, respectively. Hence, we have to only deal with logic islands. Note that this optimization may not be feasible for non-resistance based logic types.

4. NETLIST PARTITION AND MAPPING

Netlist partition and mapping (PAM) starts with the memristor-compatible netlist and produces multiple partitions mapped on memristor crossbar with their corresponding CFGs. Fig. 1b shows the two steps: (i) netlist partition and (ii) memristor tile mapping.

In step 1, the netlist partition, the netlist is partitioned into multiple smaller fragments. This step is required as the maximal affordable memristor circuit size (referred to as a *tile*) may be limited due to various constraints such as sneak path currents, high programming power, signal degradation etc. [25]. Depending on the selected design method (e.g., Boolean logic in Crossbar (BL) [8,9], Complementary Resistive Switch-Based (CRS) logic [11], memristor material implication logic (IMPLY) [10] etc.), the partition is performed using pre-defined constraints such as area, power, number of inputs/outputs, number of partitions, etc. A review of existing partition algorithms is presented in [26]; any of these algorithms can be used in this framework. To illustrate this step, we consider the two logic islands of the memristor-compatible netlist shown in Fig. 4c and select the BL design method [8] as it provides a simple implementation. This method is based on Boolean logic truth tables and poses some constraints. For instance, the size of the tile depends on the number of inputs, outputs and truth table's minterms. The number of minterms is determined by the number of inputs and the functionality of the block. Hence, the number of input is one of the partition constraints. If we assume that the selected design method allows maximum three inputs per partition, then Logic Island 1 (LI1), which consists of five inputs, needs to be partitioned. After partitioning LI1, we achieve the two sub-partitions I1.P1 and I1.P2 as shown in Fig. 5a and 5b. Logic Islands 2 (LI2) already satisfies the 3-input constraint as shown in Fig. 5c.

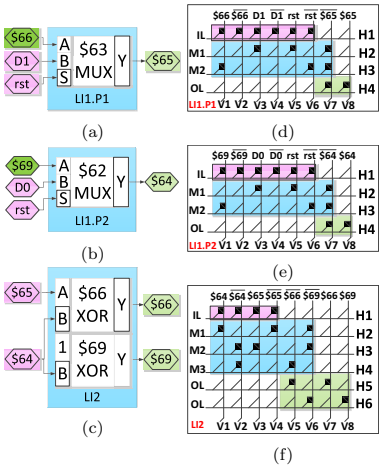


Figure 5: (a),(b),(c) Partitioned logic islands (d),(e),(f) Memristor crossbar layouts

In step 2, the memristor tile mapping, the partitioned netlist is mapped on the crossbar. The mapping method depends on the selected design method, which can be Boolean logic, implication logic, majority logic, etc. Hence, the proposed framework is generic in the sense that it can support any logic design method. Also here, we use BL to get more insights in this step. BL converts a single truth table to a memristor crossbar layout. The layout includes placement and routing of active and disabled memristors in a tile. When control voltages are applied to certain memristors, a predefined procedure is performed to calculate the outputs from inputs based on minterms evaluation.

For instance, consider the partition 1 of Logic Island 1 (LI1.P1); it only contains a MUX which can be described with the Boolean equation: $\$65 = D1.rst + \$66.\overline{rst}$, or $\overline{\$65} = \overline{D1.rst}$. $\overline{\$66.\overline{rst}}$. A truth table is created and mapped to memristor tile as shown in Fig. 5d. In the crossbar, diagonal lines with a head present active memristors, while ones without a head are disabled memristors. The tile includes the input latch (IL) on the first row which is used to capture the inputs and their complementary values; two minterm rows which implements $M1 = \overline{D1.rst}$ and $M2 = \overline{\$66.\overline{rst}}$, respectively; and an output latch (OL) which computes the AND of the two minterms and inverts the result to get the final result \$65. In Fig. 5d, H# and V# are used to denote the horizontal and vertical nanowires, respectively. We will use the notation (H#, V#) to present the position of a memristor at a junction formed by the horizontal H# and vertical V# nanowires. The IL is mapped in memristors located at (H1, V1-V4); the remaining two memristors in H1 are disabled. The two minterms M1 and M2 are located at H2 and H3; each minterm is implemented by placing active memristors at junctions formed by the horizontal nanowire presenting the minterm and the vertical nanowires associated with (a) the minterm's inputs, and (b) the output. For example, for M1, the junctions at (H2, V3)=D1, (H2, V5)=rst, and (H2, V7)=$\overline{\$65}$ consist of active memristors, while the remaining junctions are disabled. The two minterms are ANDed in parallel by column V7=$\overline{\$65}$. The OL is realized by H4; the result of the AND is stored at (H4,V7), which is thereafter inverted and stored at (H4,V8).

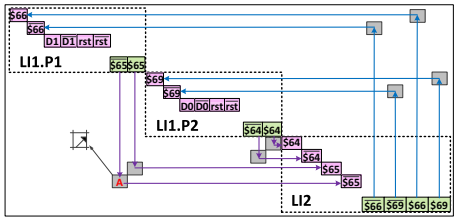


Figure 6: Placed and routed memristor tiles

Fig. 5e and 5f show the final results of the implementation of Logic Island 1 - partition 2 (LI1.P2) and Logic Island 2 (LI2), respectively on the crossbar using BL design method. A similar mapping as that used for Logic Island 1 - partition 1 (LI1.I1) is applied here.

Along with above steps, a control flow graph (CFG) is recorded based on the execution order of the partitioned netlists. Currently, this step is carried out manually. The possibility to automate this process is under investigation. This CFG is used in the next phase to create the CMOS controller. Fig. 7a shows the CFG of the 2-bit counter. The CFG includes four stages: external input capture, logic island execution, synchronization and register update.

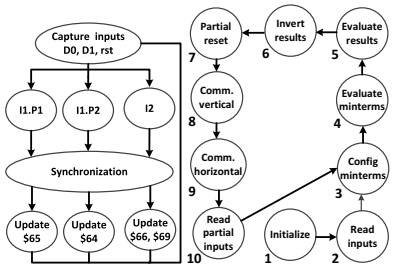
5. NETLIST PLACEMENT AND ROUTING

The netlist placement and routing (PAR) places the memristor tiles in the memristor crossbar stacked on their corresponding CMOS controllers, and constructs interconnection among them. The objective of this phase is to optimize the inter-tile communication and minimize the area consumption. Fig. 1c shows the three steps of the netlist PAR: (i) memristor PAR, (ii) CMOS controller circuit generation and PAR, and (iii) memristor and CMOS circuit integration.

In step 1, the memristor PAR, multiple memristor tiles generated in the previous phase are first placed. The implementation of PAR scheme depends on the area constraints and technology feasibility such as the possibility to isolate tiles, etc. For illustrative purposes, we use a simple PAR scheme; note that any available PAR scheme can be used. For the PAR scheme used in this paper, memristor tiles are merged into a crossbar and organized diagonally; the result is shown in Fig. 6.

Note that diagonal scheme under-utilizes the resources, and therefore other PAR schemes can be applied for efficient design. For example, an optimal PAR scheme will require the use of isolation of the different tiles; this could be realized by breaking the nanowires between the tiles, and inserting voltage drivers to control the isolated tile [9]. In this case, the tiles can be placed horizontally, vertically or in different memristor layers (3D configuration).

After the placement, the memristor tiles are routed using an interconnect network. Very limited work has been published about interconnects and communication for memristor crossbar stacked on CMOS. Three interconnect network options are possible for memristor circuits: using metal wires outside the crossbar, using memristors for copy operations, or combining both options. In the first option, metal wires are used for the communication between the memristors. Pass-transistors may be required as switches to control the communication. In the second option, using memristor-based copy operations, active memristors are placed at specific locations in the crossbar [27]. These active memristors per-



(a) CFG (b) State machine

Figure 7: CFG and state machine of CMOS controller

form copy operations when appropriate control voltages are applied. For this work, we illustrate routing by using memristor based copy operation; obviously, these memristors require an appropriate CMOS control circuit. Fig. 6 shows the placed and routed memristor tiles of the 2-bit counter. In this figure, active memristors used for interconnection are depicted with gray boxes. The inputs \$64, \$65, \$66, \$69 and their complementary are organized in a diagonal style to receive updates from the other tiles in parallel. Consider the output \$65, it is copied vertically to memristor *A*, then horizontally to input \$65. Similarly, all tiles' outputs are copied to corresponding tiles' inputs when applicable. Note that the interconnect network also guarantees the synchronization. In the 2-bit counter (as shown in Fig. 4c), results \$66 and \$69 need to be transferred backward to partition LI1.P1 and LI1.P2, while results \$65 and \$64 of partition LI1.P1 and LI1.P2 need to be transferred forward to LI2. Hence, a loop is formed in this circuit. With the preparation step in netlist generation phase (phase 1), the synchronization is carried out by executing logic islands, copying results and updating register islands simultaneously. The loop is synchronized without additional control effort.

In step 2, the CMOS controller generation and PAR, firstly the CMOS controller is generated from the CFGs. The CFGs are used to construct a HDL description state machine of the CMOS controller. The state machine is synthesized by CMOS synthesis flow while minimizing the interconnect network with the stacked crossbar. The CMOS controller of the 2-bit counter is shown Fig. 7b; it is constructed from the CFGs from the Fig. 7a. The *capture inputs* block is converted to state 1-2. At power-on, the state machine initializes the whole crossbar to high resistance (initial state) and reads all inputs. The *I1.P1, I1.P2, P2* blocks are converted into states 3-6. They calculate results in all logic islands. The *synchronization* and *update* blocks are converted to states 7-9; they communicate by transferring results from one logic island to another logic island. Finally, the *capture inputs* block is converted to state 10; This state updates the primary inputs. At this state, new inputs can be fed into the circuit or previous register values remain in the circuit to continue counting up.

Finally, the memristor and CMOS controller circuit have to be (stacked and) integrated by creating the interconnects between them. During this last step, the physical layout is generated. This step still requires further investigation.

6. CASE STUDY

6.1 Simulation setup

To verify the framework, the generated memristor crossbar are validated with HSPICE simulations using the ideal memristor model described in Verilog A [9,28]. The memristor crossbar is described in an HSPICE netlist, while the CMOS control state machine and its corresponding voltage drivers are described in Verilog A. The simulation parameters are defined as shown in Table 1 [9,28], which targets Boolean logic designs [8]; the R_H, R_L, and R_S represent the high resistance, low resistance, and reference resistor of the memristor crossbar, respectively. V_{th}, V_w, and V_{hw} represent the memristor threshold voltage, write voltage, and half-select voltage, respectively. The simulation is performed using a Linux server machine with 8 cores (at 2.66 GHz) and 32 GB memory.

Table 1: Simulation parameters [8,9,28]

Parameters	Value	Parameters	Value	Parameters	Value
R_L	$200K\Omega$	R_S	$2M\Omega$	V_w	2.9V
R_H	$4G\Omega$	V_{th}	2V	V_{hw}	1.45V

Table 2: Specifications of 8-bit adder partitions

Partition	Inputs	Outputs	Tile size (rows×cols)
P1	a0, b0, a1, b1	y0, y1, \$1	15×14
P2	a2, b2, a4, b4, \$1	y2, \$2, \$3	15×16
P3	a3, b3, \$2, \$3	y3, y4, \$4	21×14
P4	a7, b7, \$4	\$5, \$6, \$7	11×16
P5	a5, b5, \$7	y5, \$8	18×12
P6	a6, b6, \$8	y6, y7	17×12

6.2 Memristor-based 8-bit adder verification

In addition to the sequential 2-bit counter that we used through the paper to illustrate the framework, we present here an 8-bit combinational adder as a study case. Similar to the 2-bit counter, we start with the Verilog design of an 8-bit adder (with two 8-bit inputs and an 8-bit output, no carry-in and carry-out). Next, we briefly describe the result of the three phases of the synthesis framework:

- In phase 1, *netlist generation* creates a memristor-compatible netlist of 8-bit adder. As the adder is an sequential circuit, the result consists of one logic island and no register island. Due to the relative large design, the obtained netlist is not shown here.
- In phase 2, *netlist partition and mapping* first partitions the memristor-compatible netlist generated in phase 1. For illustrative purposes, we set a constraint of maximum five inputs per partition. This results in six partitions mapped on memristor tiles as shown in Table 2; each tile has a number of inputs, outputs and consumes a particular number of memristors. For instance, tile *P1* has four primary inputs (*a0*, *b0*, *a1*, *b1*) and three outputs; i.e., y0 and y1 are primary outputs, and \$1 is fed to tile *P2*. Similar specifications are applicable for other partitions.
- In phase 3, *netlist placement and routing*, the above tiles are placed on the crossbar using the same PAR scheme as that applied to the 2-bit counter. It is worth mentioning that the 8-bit adder design is used to illustrate the feasibility of the proposed synthesis framework rather than to obtain an optimal design.

Fig. 8 shows the simulation result of the 8-bit adder in 55 steps. As the carry is rippled through memristor tiles, the lower bit results (e.g. y0, y1) are ready after several steps, while the higher bit results (e.g. y6, y7) are only available near the end. The two case studies showed that our approach can generate the memristor circuits and CMOS controllers for both combinational and sequential circuit.

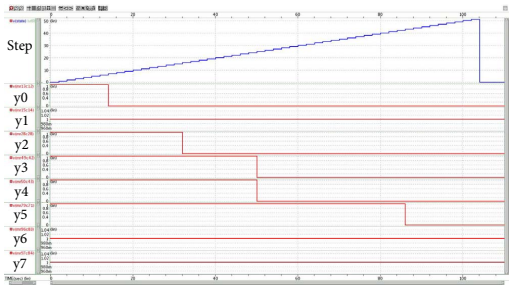


Figure 8: Results of adding '00111111' and '10000011'

7. DISCUSSION AND FUTURE WORK

The proposed framework sets a step towards the synthesis automation of memristor-based circuit design. It has many advantages and features. First, it is generic as its input are just HDL files; this enables a wide range of existing applications to be mapped on memristor technology for exploration. In addition, this enables the framework to utilize basic CMOS synthesis flows that include many mature techniques and tools. Second, the framework is flexible as it poses no constraints on HDL input size, is applicable to all existing/possible memristor logic design methods, and supports both memristor implementations (i.e., crossbar or discrete memristors). Third, the framework is scalable as its flow is independent of the complexity and the size of the circuit. Last, the framework provides a complete memristor-based circuit design in the sense that both memristor circuits and their corresponding CMOS controllers are generated and integrated together.

Nevertheless, many challenges need to be solved to get the framework close to full automation.

- Managing the complexity: The partition and mapping phase is strongly related to memristor logic design methods, the possibility to stack multiple memristor layers, the maximum tile size, etc. Research on these topics are getting more attention. In addition, the placement and routing phase has to handle the complexity of the interconnect and communication between memristor layers, as well as between memristor circuits and their CMOS controllers; optimal methods still need to be investigated.
- Efficiency of the solution: The efficiency is strongly dependent on memristor-based technology (switching speed, area, etc.), circuit (logic design methods) and architecture (e.g., crossbar with or without selector, crossbar with isolation, etc.).
- Efficient PAR scheme on the crossbar to minimize the use of area.
- Scalability and complexity of CMOS control circuits with increase of the crossbar size.

8. CONCLUSION

In this paper, we have presented a generic synthesis framework for memristor circuits. The framework creates memristor circuit and their controller for HDL-based designs. Without the loss of generality, the framework feasibility was validated by two study cases: 2-bit counter and 8-bit adder. At the current state of memristor technology, a feasible synthesis framework is necessary to investigate the possibility and efficiency of a memristor circuit further. As there is no existing generic synthesis framework, the work is a step towards the exploration of large-scale memristor circuits.

9. REFERENCES

[1] K. Stefanos, "Architecture at the End of Moore", Advances in Atom and Single Molecule Machines, pp.1-10, 2013.
[2] S. Borkar, "Design challenges of technology scaling", *MICRO*, pp.23-29, 1999.
[3] S. Hamdioui et al., "Reliability Challenges of Real-Time Systems in Forthcoming Technology Nodes", *DATE*, pp.129-134, 2013.
[4] S. Borkar, "Design Perspectives on 22nm CMOS and Beyond", *DAC*, pp.93-94, 2009.
[5] ITRS, "The International Technology Roadmap for Semiconductors", 2011 .
[6] R. Waser et al., "Redox-Based Resistive Switching Memories Nanoionic Mechanisms, Prospects, and Challenges", *AM*, pp.2632-2663, 2009.
[7] D. Strukov et al., "The missing memristor found", Nature, 2008.
[8] L. Xie et al., "Fast Boolean Logic Mapped on Memristor Crossbar", *ICCD*, pp.335-342, 2015.
[9] G. Snider, "Computing with hysteretic resistor crossbars", Applied Physics A, pp.1165-1172, 2005.
[10] S. Kvatinsky et al., "Memristor-Based Material Implication (IMPLY) Logic: Design Principles and Methodologies", *TVLSI*, pp.2054-2066, 2014.
[11] A. Siemon et al., "A Complementary Resistive Switch-Based Crossbar Array Adder", *JETCAS*, 2015.
[12] L. Gao et al., "Programmable CMOS/Memristor Threshold Logic", *TNANO*, pp.115-119, 2013.
[13] M. Di Ventra et al., "Memcomputing: a computing paradigm to store and process information on the same physical platform", Nature Physics, pp.200-202, 2013.
[14] S. Hamdioui et al., "Memristor based Computation-in-Memory Architecture for Data-Intensive Applications", *DATE*, pp.1718-1725, 2015.
[15] K. Ma et al, "Architecture exploration for ambient energy harvesting nonvolatile processors", *HPCA*, 2015.
[16] E. Linn et al., "Beyond von Neumann-logic operations in passive crossbar arrays alongside memory operations", *TNANO*, pp.305205, 2012.
[17] H.A. Du Nguyen et al., "Computation-in-memory based parallel adder", *NANOARCH*, pp.57-62, 2015.
[18] M. Pinaki et al., "Memristors: Devices, Models, and Applications", Proceedings of the *IEEE*, 2012.
[19] F. S. Marranghello et al., "Improved logic synthesis for memristive stateful logic using multi-memristor implication", *ISCAS*, pp.181-184, 2015.
[20] F. Deliang et al., "Design and Synthesis of Ultralow Energy Spin-Memristor Threshold Logic", *TNANO*, pp.574-583, 2014.
[21] A. Velasquez et al., "Automated synthesis of crossbars for nanoscale computing using formal methods", *NANOARCH*, pp.130-136, 2015.
[22] W. Wen et al., "An EDA framework for large scale hybrid neuromorphic computing systems", *DAC*, pp.1-6, 2015.
[23] W. Clifford, "Yosys Open SYnthesis Suite", http://www.clifford.at/yosys/, 2013.
[24] Berkeley Logic Synthesis and Verification Group, "ABC: A System for Sequential Synthesis and Verification", 2012.
[25] Geoffrey W. Burr et al., "Access devices for 3D crosspoint memory", *JVSTB*, pp.040802, 2014.
[26] Charles J. Alpert et al., "Recent Directions in Netlist Partitioning: A Survey", Integrated VLSI Journal, 1995.
[27] L. Xie et al., "Interconnect networks for memristor crossbar", *NANOARCH*, pp.124-129, 2015.
[28] E. Lehtonen et al., "Memristive Stateful Logic", Memristor Networks, pp.603-623, 2014.

2017 12th International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS)

Interconnect Networks for Resistive Computing Architectures

H.A. Du Nguyen, Lei Xie, Jintao Yu, Mottaqiallah Taouil, Said Hamdioui
Laboratory of Computer Engineering, Delft University of Technology, The Netherlands
Email: H.A.DuNguyen@tudelft.nl

***Abstract*—Today's computing systems suffer from a memory/communication bottleneck, resulting in high energy consumption and saturated performance. This makes them inefficient in solving data-intensive applications at reasonable cost. Computation-In-Memory (CIM) architecture, based on the integration of storage and computation in the same physical location using non-volatile memristor crossbar technology, offers a potential solution to the memory bottleneck. An efficient interconnect network is essential to maximize CIM's architectural performance. This paper presents three interconnect network schemes for CIM architecture; these are (1) CMOS-based, (2) memristor-based and (3) hybrid cmos/memristor interconnect network scheme. To illustrate the feasibility of such schemes, a CIM parallel adder is used as a case study. The results show that the hybrid interconnect network scheme achieves a higher performance in comparison with the CMOS-based and memristor-based interconnect scheme in terms of delay, energy and area.**

4

I. INTRODUCTION

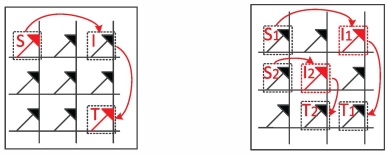
Classical Von Neumann computer architectures are unable to deal with big data problems efficiently due to the memory bottleneck [1], high energy consumption [2] and inefficient programming methodology [3]. Architectures based on resistive computing [4–6], such as Computation-in-Memory (CIM) architecture [6], are emerging and try to address the aforementioned problems [7]. CIM architecture alleviates the memory bottleneck by using the same physical devices to implement both logic and storage units. However, such computing architectures require efficient interconnect network scheme to explore their potential. The interconnect network and communication schemes applied in CMOS technology are not applicable to memristor crossbar due to fundamental differences in their working principle. In CMOS technology, logic signals are represented by voltages that propagate through wires. In CIM, logic signals are represented by resistances that are programmed by control voltages. Therefore, it requires a distinct interconnect network.

Recent research in the field has been focusing on implementing logic inside the memristor crossbar [8–11]. However, limited work has investigated possible interconnect network and communication schemes; so far only a single publication has addressed this topic. In previous work [12], the authors have proposed different communication schemes (i.e., unicast, multicast, and broadcast) using an interconnect network fully integrated in the memristor crossbar. The scheme has a high area overhead and the communication cost depends on the relative positions between the source and target memristor. This complicates the place-and-route phases [13]. Therefore, there is a need to explore different possible interconnect network schemes in order to optimize overall performance.

In this paper, we develop and investigate the potential of different interconnect network schemes and use the CIM parallel adder [14] as a case study. Note that the proposed interconnect network schemes can be applied to any crossbar-based resistive computing architecture. The contributions of this paper are the following:

- We propose two new interconnect networks for CIM architecture.
- We compare them with the previously interconnect network proposed in [12]; the CIM parallel adder is used as a case study.
- We evaluate the overhead of the three interconnect network schemes in terms of delay, energy and area.

The rest of this paper is organized as follows. Section II briefly describes the state-of-the-art and the CIM parallel adder implementation. Section III presents the interconnect network schemes. Section IV discusses the evaluation model and results. Finally, Section V concludes this paper.



(a) Single *copy* operation (b) Diagonal scheme

Fig. 1: Interconnect Network Proposed in [12]

II. BACKGROUND

This section first presents the state-of-the-art in memristor crossbar based interconnect networks. Thereafter, it discusses the CIM parallel adder and its crossbar implementation [10].

A. State-of-the-Art Interconnect Network Schemes

In [12], the authors proposed a complete interconnect network for the memristor crossbar. Copy operations are used to move data between two memristors. A *copy* operation is carried out by applying appropriate control voltages to the source and target memristor nanowires. A direct *copy* operation takes one memristor write cycle and occurs when both memristors share the same horizontal or vertical nanowire.

To perform a *copy* operation between two memristors residing on different rows and columns, an extra copy operation is required as shown in Fig. 1a. Here, an additional memristor located at one of its two intersection points referred to as diagonal scheme. In the figure, S presents the source memristor and T the target memristor, while I is used as intermediate memristor. As a result, the communication cost doubles, i.e., two memristor write cycles. In order to transfer multiple bits simultaneously, the diagonal scheme can be extended (as shown in Fig. 1b); it performs the data transfers (i.e., from S_1 to T_1 and S_2 to T_2) in two cycles.

978-1-5090-6377-2/17/$31.00 ©2017 IEEE

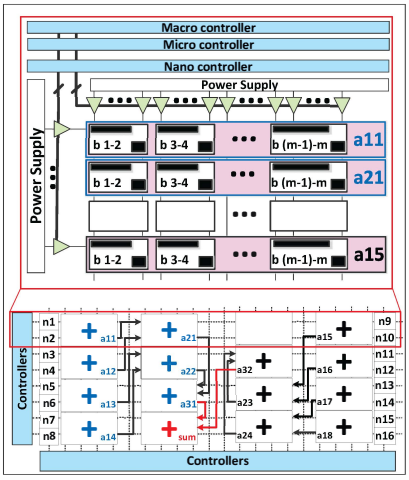


Fig. 2: CIM-based Parallel Adder (16 inputs)

In [8], the authors present a memristor adder based on Complementary Resistive Switches (CRS). As both high and low resistance have the same equivalent resistance, copy operations cannot be used. Instead, the authors propose a CMOS interconnect network consisting of a controller with additional logic (such as sense amplifier, write driver) to read from the source memristor and write to the destination. However, no implementation details were reported.

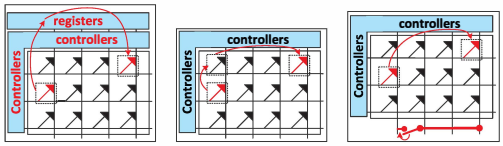
B. CIM Parallel Adder

The lower part of Fig. 2 shows a parallel adder (based on the binary tree) implemented using the CIM concept, referred to as CIM parallel adder [14]. It consists of two components: (i) the computation and storage part which reside in the memristor crossbar, and (ii) the CMOS controlling part.

The memristor crossbar has two purposes, i.e., it performs computations (additions) and it is used as a storage (memory) device. In Fig. 2, the adders are indexed by a_{xy} where x presents the addition stage and y the adder index within a stage. The adders in this architecture are not reused as they relatively do not impact the interconnect network schemes. Dedicated storage cells for the inputs are presented by n_i, where i the index of the i^{th} input. Each storage cell consists of a memristor that is able to store a single bit. The placement of the adders and memory is flexible as both are implemented using memristors. To utilize the area of the binary tree implementation efficiently, half of the inputs are mapped from left side, while the other half from right side.

The upper part of Fig. 2 shows the CIM parallel adder implementation using the Fast Boolean Logic Circuit (FBLC) style [10]. Each adder a_{xy} is replaced by an FBLC 32-bit ripple carry adder that is composed of 16 two-bit adders; the two-bit adders are selected for their area efficiency [10]. Each two-bit adder is represented by a rectangle with two dark areas in the upper part of Fig. 2; the longer dark areas present the input area, while the smaller dark areas the outputs. The crossbar, the controller, and the peripheral circuit are marked in pink, blue and green area, respectively.

Fig. 3: State Machine for Micro Controller



(a) CMOS (b) Memristor (c) Hybrid

Fig. 4: Proposed Interconnect Network

The CMOS part consists of the controller and peripheral circuit. The controller is hierarchically divided in three layers, i.e., macro, micro and nano controller. The macro-controller controls the addition stage of the binary tree. The micro-controller executes a single addition. Each two-bit addition has four states as shown in Fig. 3; these states are repeated 16 times to complete a 32-bit addition. The nano-controller translates the macro- and micro-controller instructions to appropriate control voltages which are subsequently applied to the appropriate nanowires of the crossbar. Each controller is implemented with a state machine. More details of the nano-controller can be found in [10,12].

The crossbar implementation requires a peripheral circuit to support the memristor crossbar operations. For example, the peripheral circuit includes voltage drivers to control the nanowires, sense amplifiers to read out memristor states, and multiplexers accompanied with the sense amplifiers to select which nanowires are accessed, and logic circuitry to create connections between different crossbar nanowires.

III. INTERCONNECT NETWORK SCHEMES

This section describes the overview of interconnect network schemes. Thereafter, it integrates them in the CIM parallel adder. Note that the CIM parallel adder is only used for illustrative purposes. The proposed interconnect networks can be applied in general to any resistive computing architectures.

A. Overview

Fig. 4 shows the three proposed interconnect networks. They are referred to as: (1) CMOS, (2) memristor, and (3) hybrid scheme. Each scheme impacts the crossbar and CMOS logic differently, as explained next.

Fig. 4a shows the CMOS interconnect network scheme. In this scheme, data movements in the crossbar transit through registers in the CMOS layer and consist of two memristor operations (i.e., a read and write operation). First, the controller reads the value from a source memristor. It stores the value temporarily in a CMOS register. Second, it writes the register value into the target memristor. This interconnect network scheme requires sense amplifiers, multiplexers, registers and a controller that applies appropriate control voltages. The communication requires two cycles with a the cycle time equal to the maximum delay of the read and write operation. The delay of the read operation consists of the delay of the controller, sense amplifiers, multiplexers and the time to write the CMOS registers. The write path delay consists of the controller delay and the write time of the target memristor.

Fig. 4b shows the memristor interconnect network scheme. In this scheme (based on the work published in [12]), the

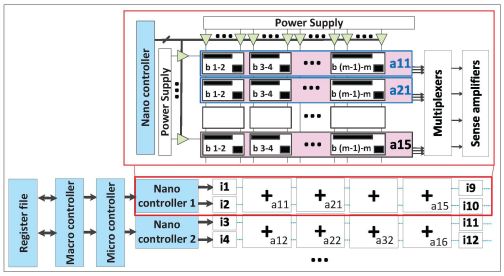


Fig. 5: CMOS-based Interconnect Network

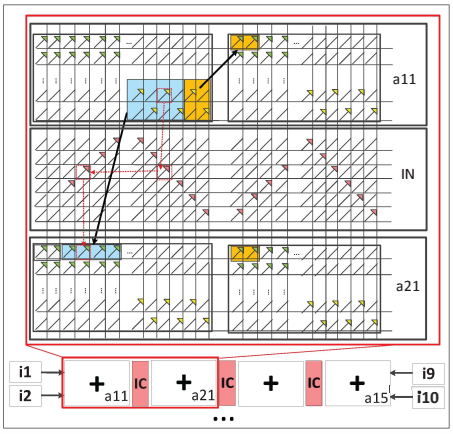


Fig. 6: Memristor-based Interconnect Network

controllers need one or two steps to transfer data from the source to target memristor. Each step consist of a memristor *copy* operation. This scheme requires an extra intermediate memristor at the intersection points between the source and target memristors. In case multiple bits are transferred simultaneously, the diagonal scheme may be used [12]. The maximum delay of the scheme is two memristor write cycles in case there are no conflicts between intermediate and reserved (for logic or storage) memristors. In case of a conflict, more than two copy operations are required.

Fig. 4c shows the hybrid interconnect network scheme. This scheme also uses a *copy* operation to transfer data from source to target memristor. It performs this operation in a single cycle by creating a direct path between source and destination using a CMOS switch as depicted in the bottom part of Fig. 4c. This switch can be implemented by a pass-gate. The transfer delay equals the delay of the controller to perform the *copy* operation and to activate the pass transistor.

As described above, the three proposed schemes are capable of transferring data between any two memristors, and support parallel communication. However, they impact the delay, energy and area consumption of the memristor crossbar and CMOS control differently. Note that the above concept applies to data transfers between adders within the same sub-crossbar (intra-crossbar) and two different sub-crossbars (inter-crossbar). In the following sections, the three schemes are applied to the CIM parallel adder as a case study.

B. CMOS Interconnect Network

Fig. 5 shows the CIM parallel adder based on the CMOS interconnect network scheme. Each sub-crossbar has its own nano-controller. Therefore, communication within a sub-crossbar is handled by a single controller while two controllers are involved in the communication between different crossbars.

In intra-crossbar communication, (e.g., between adders $a11$ and $a21$), the controller reads out the value from the source adder ($a11$), stores it in a register, and writes the value in the target adder ($a21$). In inter-crossbar communication, (e.g., between adders $a12$ and $a21$), the communication is handled by two controllers and may be implemented in two ways. The first method adds an additional network on the CMOS side to move data between the controllers. For example, a source controller (e.g., *nano-controller 2* in Fig. 5) reads out the value of adder $a12$, transmits the value via the interconnect network (which is also implemented by CMOS devices) to the target controller *nano-controller 1*; the target controller writes the received values into the inputs of the target memristors (i.e., inputs of $a21$). The second method is through a register file. In this case, the source controller writes the read-out value to a predefined register; thereafter, the target controller reads the value from the predefined register. Note that, the register file may become large when there are a lot of data transfers. In case the size is too big, a memory can be used.

A read-out operation might destroy the value stored in the source adder [8]. Hence, a write-back (restore) operation is also required. The read-out, write-in and write back operations are carried out by applying appropriate control voltages to the nanowires and are required for each data transfer.

Due to the synchronization between two controllers, the read-out and write-in operations executed sequentially using two different steps, while the write-back operation is executed simultaneously with the write-in operation. Therefore, with respect to the crossbar, the delay equals two memristor write operations, the energy three memristor write operations per bit (including the write-in, write-back operation and the possible destructive read of the source memristor), and the area is zero (as no extra memristors are required). With respect to the controller, each crossbar needs its own nano-controller. In addition, two states are required for the two communication steps in the micro controller in comparison with the original design [12]. With respect to the peripheral circuit, a 2-bit adder has three values to be read out (one carry bit and two sum bits) simultaneously; hence three sense amplifiers are required per adder. As only one 2-bit adder is active at a time in a sub-crossbar, the sense amplifiers can be shared, while multiplexers are used to forward the data from the currently working adders. In short, only three sense amplifiers are required per sub-crossbar. Note that the sense amplifier may be a current [15] or voltage sense amplifier [16] depending on the adder logic type. In the CIM parallel adder implementation using FBLC style, a voltage sense amplifier is preferred [16]. Finally, a register file is required to store temporary read values.

C. Memristor Interconnect Network

Fig. 6 shows the CIM parallel adder based on the memristor-based interconnect network scheme. In this scheme, we use the state-of-the-art interconnect network proposed in [12]. As the

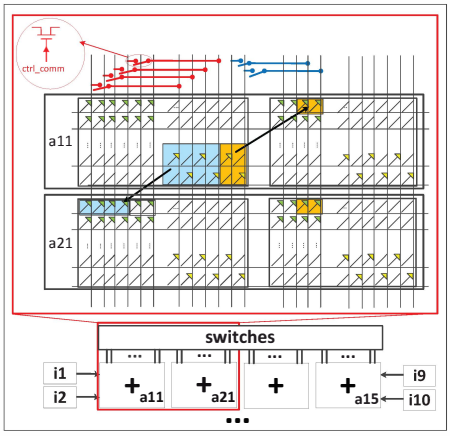


Fig. 7: Hybrid Interconnect Network

FBLC adder is used, the nano-controller can be shared among the sub-crossbars as they require the same control voltages at each time step.

Note that each adder is built from multiple two-bit adders that are placed on the same adder row. To simplify the state machines, communication takes directly place after partial results are obtained. Consider for example the output of adder $a11$ which will be transferred to the input of adder $a21$. The target adder $a21$ is located in a different row. As soon as the first two-bit adder of adder $a11$ produces its outputs, they will be moved to the proper locations. More precisely, the carry output (in orange in Fig. 6) of this adder needs to be transferred to the second two-bit adder of $a11$, while the sum output (in blue in Fig 6) is needed at the first two-bit adder of $a21$ (for the addition in the next stage). As the source and target memristors are on different rows, additional intermediate memristors (see Fig. 1) are required to create a path between them. The red dotted arrow lines present the communication paths between source and destination adder.

As multiple bits need to be transferred simultaneously, the diagonal scheme of Fig. 1b is used as shown in the upper part of Fig. 6; denoted by Interconnect Network (IN). In this scheme, all communications from one adder stage to the next one can be performed simultaneously with a latency of three cycles due to two intermediate hops. Note that only one IN block is needed per sub-crossbar as one adder is active at the time.

Next, we discuss the delay and area cost. With respect to the crossbar, the communication delay between source and destination equals 3 write operations for each n=2-bit addition, the energy equals 3 memristor write operations per data transfer (as 3 writes are required due to the need of two intermediate memristors). Each sub-crossbar needs $2 \cdot (n+1)$ additional rows for the IN block as an n-bit adder has n-sum bits and one carry bit and their complementary values as outputs. With respect to the CMOS part, three additional states are required for the micro-controller to perform the communication steps. With respect to the peripheral circuit, six additional voltage drivers are required per sub-crossbar to drive the rows of the IN block.

TABLE I: Simulation Parameters [9,18]

Parameters	Value	Parameters	Value	Parameters	Value
R_L	$100k\Omega$	R_H	$1G\Omega$		
V_w	1.4V	V_{hw}	0.7V	V_{th}	1V

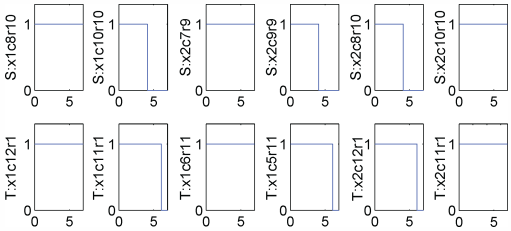


Fig. 8: HSPICE verification of hybrid communication scheme

D. Hybrid Interconnect Network

Fig. 7 shows the CIM parallel adder based on the hybrid interconnect network scheme which consists of both CMOS devices and memristors. This scheme also utilizes the memristor *copy* operation as mentioned in [12]. However, instead of using additional intermediate memristors, CMOS switches are used to create direct paths between source and target memristors. For instance, in the upper part of Fig. 7 two nanowires are connected by a transmission-gate [17], which is controlled by the CMOS controller. When the switch is open, the two nanowires are disconnected and normal computations can be performed. As the switch closes, the memristors on the two different nanowires are directly connected. Therefore, a single *copy* operation [12] can be used to transfer data within and between sub-crossbars. As the inputs of the target adder and the outputs of the source adder are both effectively located in the same row, multiple bits can be copied simultaneously.

We discuss the cost of the scheme next. With respect to the crossbar, the delay of a *copy* operation equals one write operation, the energy consists of writing the target memristor (per single *copy* operation [12]), and the area is zero (as no extra memristors are required). With respect to the CMOS part, additional logic is required to control and implement the switches [17]. The micro controller requires one additional state to perform the communication step. No sense amplifiers and multiplexers are required for the peripheral circuit.

IV. RESULTS

This section first verifies the proposed interconnect network schemes. Thereafter, we present a model to evaluate them. Last, we discuss the performance results.

A. HSPICE Verification

We verify the proposed interconnect network schemes in HSPICE simulations using the ideal memristor model [9,18] using the same simulation approach as in [12]. The memristor parameters are shown in Table I; the low resistance R_L = $100k\Omega$ represents a logic 0, the high resistance R_H = $1G\Omega$ represents a logic 1, the memristor threshold voltage V_{th} equals 1V, the memristor write voltage V_w 1.4V, and the half-select voltage V_{hw} 0.7V. Due to space limitations, only the results of the hybrid scheme are included. Fig. 8 shows the data transfer

TABLE II: Basic Model Parameters

Component	Basic Unit	Parameter	CMOS-based	Memristor-based	Hybrid
Device Technology					
Memristor Technology		F_m - Feature size (nm)	5 [19]		
		D_m - Delay (ps)	200 [19]		
		E_m - Energy (fJ)	1 [19]		
		A_m - Area (nm^2)	100 ($4F_m^2$) [19]		
CMOS Technology		F_c - Feature size (nm)	40		
CIM Parallel Adder Interconnect Network Schemes					
Peripheral Circuit	Voltage driver	D_{icc} - Delay	not applicable	masked by adder voltage driver	
		E_{vd} - Energy (pJ)		E_{pt}	E_{pt}
		A_{vd} - Area (um^2)		$3 \cdot A_{pt}$	A_{pt}
	Sense Amplifier [16]	D_{sa} - Delay (ps)	20	not applicable	
		E_{sa} - Energy (pJ)	3.69		
		A_{sa} - Area (um^2)	0.017		
	Multiplexer [20]	D_{mux} - Delay (ps)	D_{pt}		
		E_{mux} - Energy (pJ)	E_{pt}		
		A_{mux} - Area (um^2)	$16 \cdot 4 \cdot A_{pt}$		
Controller (Synthesized for 16 inputs, 32-bit input)		D_{cont} - Delay (ps)	825	825	825
		E_{cont} - Energy (pJ)	0.115	0.103	0.100
		A_{cont} - Area (um^2)	2234	1869	1147
Memristor Crossbar		D_{icx} - Delay (ps)	400 (2 cycles $\cdot$ D_m)	600 (3 cycles $\cdot$ D_m)	200 (1 cycle $\cdot$ D_m)
		E_{icx} - Energy (fJ)	2 (2 memristors $\cdot$ E_m)	3 (3 memristors $\cdot$ E_m)	1 (1 memristors $\cdot$ E_m)
		A_{icx} - Area (um^2)	0	200 (2 memristors $\cdot$ A_m)	0

from source memristors (first row) to the target memristor (second row); in the y-axis, S and T represent the source and target memristor, x the crossbar index, c and r the column and row index; the x-axis shows the time steps. The data values are selected randomly. The first two plots of both the source and target memristors present simultaneous *copy* operation that occurs in one single sub-crossbar from different source rows ($r10$ and $r9$) to one target row ($r1$) (see orange blocks in Fig. 7). The next two plots present similar *copy* operations which take place between two different sub-crossbars (from the output rows of sub-crossbar $x1$ to the input row of sub-crossbar $x2$, e.g., as shown by the blue blocks in Fig. 7). The last four plots shows that simultaneous *copy* operations can occur within the same or between different sub-crossbars.

B. Evaluation Model

The interconnect network schemes are evaluated using the CIM parallel adder model. The model includes four components: memristor adders, peripheral circuit (voltage drivers, sense amplifiers, multiplexers), controller (including register file). The basic adder implementation is the same for the three interconnect schemes. Table II shows the basic parameters of the model. The top part of the table shows the memristor and CMOS technology parameters. The memristor device has a feature size F_m of $5nm$, delay D_m of $200ps$, energy consumption E_m of $1fJ$ per transition, and area A_m of $100nm^2$. The CMOS device has a feature size F_c of $40nm$. This technology node is also used to synthesize the controller. The lower part of Table II shows the basic parameters of the peripheral circuits, controller.

The above basic model parameters are used to estimated the performance of the proposed interconnect network schemes. The delay per cycle is calculated by the sum of the components that are part of the critical path. The total delay is calculated by taking the product of the delay per cycle and the number of cycles. The total energy and area are obtained by summing up the energy and area cost of all the sub-components.

C. Results

Fig. 9 shows the total delay, energy and area of the CIM parallel adder implementation with the three proposed interconnect network schemes, while Fig. 10 shows the cost breakdown per component. Fig. 10 contains six plots related to the energy and area of each component (i.e., crossbar, controller and peripheral circuit). The delay is not considered here due to space limitations; in addition, the delay is based on the critical path and not on the total sum of the individual components. The results of each metric are described next.

Delay: In terms of delay, the hybrid scheme outperforms marginally the other two schemes as it (i) does not have a sense amplifier and multiplexer delay in its critical path, (ii) requires fewer controller states, and (iii) needs fewer memristor writes operations for communication.

Energy: In terms of energy consumption, the memristor-based scheme performs the worst due to the additional writes required for the intermediate memristor and control states. In comparison with the CMOS-based scheme, the hybrid scheme has a lower energy, as it requires fewer CMOS devices and has a lower delay.

The energy breakdown for each component is shown in the top part of Fig. 10. Note that the interconnect network is only a small part of the FBLC based CIM parallel adder, and therefore, nearly the same energy results are observed for the crossbar part. Nevertheless, the memristor-based scheme has the highest crossbar energy consumption due to the writing of the intermediate memristors. In comparison with the CMOS-based scheme, the hybrid scheme has slightly a lower energy consumption as each data transfer requires a single write. With respect to the controller, the CMOS-based scheme has the highest energy consumption due to the additional control logic and register file. In the CMOS-based scheme, read-out operations may be destructive and write-back operations are required to preserve the value of the source memristor, hence the additional states also cost more energy. Note that the hybrid scheme's controller requires only one additional state for the micro-controller, while the CMOS- and memrisor- based schemes need two and three additional states, respectively. With respect to the peripheral circuit, the CMOS-based scheme also has the highest energy consumption, due to the presence

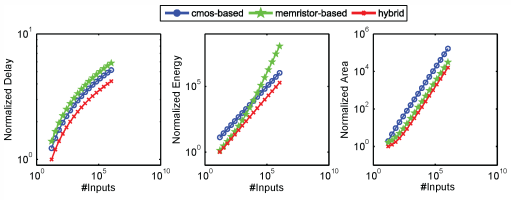


Fig. 9: Performance of Proposed Interconnect Network Schemes

of sense amplifiers and multiplexers. Note that the switches used for the interconnect network in the hybrid scheme show relatively a marginal energy consumption. The memristor-based scheme has a slightly higher energy consumption in comparison with the hybrid scheme, mostly due to the presence of additional voltage drivers.

Area: In terms of area, the CMOS-based scheme has the highest area overhead due to the presence of additional peripheral circuits, a register file and a larger controller. The other two schemes have nearly the same area overhead. We will discuss the cost breakdown next.

Each component's area is shown in the lower part of Fig. 10. With respect to the crossbar, the memristor-based scheme consumes slightly more area than the other two schemes as additional memristors are required to perform the communication. With respect to the controller, the hybrid scheme has the smallest area overhead as only one additional state is required for the micro-controller. The controller of the memristor-based scheme has slightly a smaller area than the CMOS-based scheme, as the CMOS-based scheme needs a register file and additional logic to address the register file. With respect to the peripheral circuit, the memristor-based scheme has slightly a higher area overhead as compared to the other two schemes due to the presence of voltage drivers. The peripheral circuits of the hybrid scheme requires less area than the CMOS-based scheme due to the presence of sense amplifiers and multiplexers.

In summary, the hybrid communication scheme shows benefits in terms of delay, energy and area due to fewer control states, less peripheral circuits and no need of additional memristors are required for the interconnect network. The results of this paper are useful for memristor designers, as it is essential to select appropriate interconnect schemes to improve the overall performance.

V. CONCLUSION

In this paper, we showed three feasible interconnect network schemes for resistive computing architectures in general, and for CIM architecture in particular. The result showed that a hybrid interconnect network scheme has the highest efficiency in terms of delay, energy and area. This shows that an interconnect network fully integrated in the memristor crossbar is not efficient enough, and can lead to vast performance reduction. Similarly, an interconnection network solely based on CMOS devices also showed to be inefficient. In conclusion, it is essential to utilize both memristor operations and CMOS devices to build an effective interconnect network scheme for resistive computing architectures.

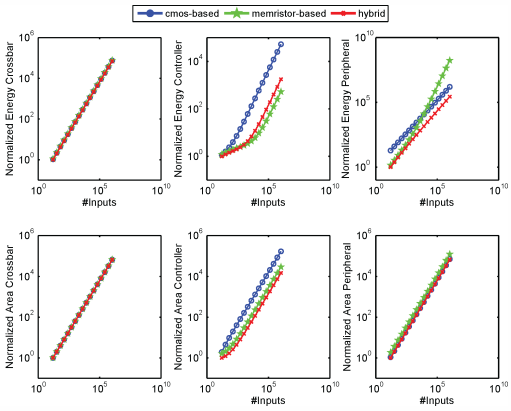


Fig. 10: Performance Breakdown of Each Component

REFERENCES

[1] S. Kaxiras, *Architecture at the End of Moore*, ser. Advances in Atom and Single Molecule Machines. Springer Berlin Heidelberg, 2013.

[2] B. Hoefflinger, "The energy crisis," in *Chips 2020*, ser. The Frontiers Collection, 2012, pp. 421–427.

[3] H. Esmaeilzadeh *et al.*, "Dark silicon and the end of multicore scaling," *SIGARCH Comput. Archit. News*, vol. 39, pp. 365–376, 2011.

[4] P.-E. Gaillardon *et al.*, "The programmable logic-in-memory (plim) computer," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2016, pp. 427–432.

[5] R. B. Hur *et al.*, "Algorithmic considerations in memristive memory processing units (mpu)," *International Symposium on Nanoscale Architecture (NANOARCH)*, 2016.

[6] S. Hamdioui *et al.*, "Memristor based computation-in-memory architecture for data-intensive applications," in *IEEE Conference on Design, Automation & Test in Europe*, 2015, pp. 1718–1725.

[7] S. Hamdioui *et al.*, "Memristor for computing: Myth or reality?" in *Design, Automation and Test in Europe DATE*, 2017.

[8] A. Siemon *et al.*, "A complementary resistive switch-based crossbar array adder," *IEEE journal on emerging and selected topics in circuits and systems*, vol. 5, pp. 64–74, 2015.

[9] G. Snider, "Computing with hysteretic resistor crossbars," *Applied Physics A*, vol. 80, pp. 1165–1172, 2005.

[10] L. Xie *et al.*, "Fast boolean logic mapped on memristor crossbar," in *IEEE Conference on Computer Design (ICCD)*, 2015, pp. 335–342.

[11] S. Kvatinsky *et al.*, "Magicmemristor-aided logic," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 61, pp. 895–899, 2014.

[12] L. Xie *et al.*, "Interconnect networks for memristor crossbar," in *Symposium on Nanoscale Architectures*, 2015, pp. 124–129.

[13] J. Yu *et al.*, "Skeleton-based design and simulation flow for computation-in-memory architectures," in *Nanoscale Architectures (NANOARCH), 2016 IEEE/ACM International Symposium on*. IEEE, 2016, pp. 165–170.

[14] H. A. Du Nguyen *et al.*, "Computation-in-memory based parallel adder," in *Symposium on Nanoscale Architectures*, 2015, pp. 57–62.

[15] S. Sundaram *et al.*, "High speed robust current sense amplifier for nanoscale memories: a winner take all approach," in *Conference on VLSI Design (VLSID)*, 2006.

[16] W. Dehaene *et al.*, "Variability-aware design of low power sram memories," *Doctoral Thesis*, 2009.

[17] R. Zimmermann *et al.*, "Low-power logic styles: Cmos versus pass-transistor logic," *IEEE journal of solid-state circuits*, vol. 32, 1997.

[18] E. Lehtonen *et al.*, "Memristive stateful logic," in *Memristor Networks*. Springer, 2014, pp. 603–623.

[19] ITRS, "The international technology roadmap for semiconductors ITRS," Semiconductor Industry Asso, Tech. Rep., 2011.

[20] SYNOPSYS, "Digital standard cell library saed_edk90_core databook," SYNOPSYS ARMENIA Educational Department, Tech. Rep., 2008.

5

CONCLUSION

This chapter summarizes the overall achievements of this dissertation and highlights some future research directions. Section 5.1 presents a summary of the main conclusions of this dissertation. Thereafter, Section 5.2 discusses possible future research directions.

5.1. SUMMARY

Chapter 1, "Introduction", briefly introduced in-memory computing and the research focus of this dissertation. It first described the motivation of this dissertation. Thereafter, it showed the state-of-the-art of in-memory computing with respect to the following aspects: device, logic, architecture, compiler and application; it also discussed the opportunities and challenges of each aspect. Finally, it proposed the contributions of this dissertation which addresses some circuit and architecture challenges for in-memory computing.

Chapter 2, "Overview and Classification", first introduced the potential of memristive devices for multiple applications such as memory, logic, and architectures (e.g., neuromorphic and computation-in-memory). Thereafter, it proposes a classification that is based on three metrics: computation location, memory technology and computation parallelism. The classification is comprehensive and complete; it covers both existing and potential architectures that are grouped into four classes: CIM-A, CIM-P, COM-N and COM-F. They perform computations using the memory array, the peripheral circuits, extra logic within the memory System-in-Package but outside the memory core and logic circuits that are located off-chip, respectively. Finally, the existing architectures, as well as their pros and cons, are discussed and evaluated quantitatively.

Chapter 3, "Architecture Level", introduced two novel in-computing architectures; each represents CIM-A and CIM-P class. Computation-in-Memory (CIM) Architecture is a novel computing paradigm that tightly integrates computation and storage into the same physical devices. The computations are performed using the memory array; hence, this architecture belongs to CIM-A class. This work proposed a CIM-based architecture that performs parallel addition using the memristor crossbar. Thereafter, we described two implementations of the CIM-based parallel adder using two distinct logic design styles (e.g., Boolean logic using conventional memristive devices, and implication logic using complementary resistive switch CRS devices). The two implementations are discussed intensively, in particular their controller, interconnect network, and communication schemes. Their performance, energy and area are also compared with a multicore architecture using an analytical model. Computation-in-Memory Accelerator (CIMX) proposed a CIM core (i.e., a memristor crossbar with computation capability) and several architectures that integrate CIM core into different memory hierarchies of a conventional architecture (e.g., cache, main memory) and as an accelerator. The CIM core performs computations using peripheral circuits; hence, the architecture is classified as a CIM-P architecture. This work described a selected architecture that uses CIM as an accelerator. Subsequently, an analytical model and simulation framework were proposed to evaluate and compare the proposed architecture with a conventional architecture.

Chapter 4, "Circuit Level", proposed a logic synthesis automation tool and interconnect network schemes. First, a logic synthesis framework is proposed to map a circuit described in hardware-description-language (HDL) into memristor circuit. The generic framework is illustrated using two case studies which include a 2-bit comparator and a 8-bit adder. Thereafter, three interconnect network schemes are proposed using primitive

copy, *read*, *write* operations [79]. They are applicable to in-memory computing architectures based on resistive devices; they are (1) only using memristor crossbar, (2) using CMOS circuits (controller and peripheral circuits) , and (3) using both memristor crossbar and CMOS circuits. Thereafter, a case study, CIM parallel adder is used to illustrate these three schemes and evaluate their performance, energy and area.

5.2. FUTURE RESEARCH DIRECTIONS

Several recommendations are suggested to improve the state-of-the-art further. They are organized by the different research topics as listed below.

ARCHITECTURE LEVEL

1. **Computation-in-Memory (CIM) Architectures**

   - Investigate the integration of memristor crossbars with CMOS controllers. To the best of our knowledge, our work is the first to investigate the impact of the CMOS controller on an entire memristor design. Based on this preliminary work, the CMOS part might be a concern. Further investigation requires not only the optimization of the CMOS part, but also its efficient use to control the crossbars (e.g., sharing them by different crossbars, pipelining the controller to reduce the clock frequency).
   - Develop libraries with well-optimized memristor designs. Currently, only limited number of computational units (such as adder, multiplier) and communication schemes have been proposed. Therefore, libraries with optimized memristor designs are essential to further explore the potential of CIM architectures.

2. **Computation-in-Memory Accelerator (CIMX)**

   - Improve CIMX architecture. CIMX was designed with the intention to minimize the required number of modifications to conventional architectures. It is worth noticing that much higher performance and energy improvements can be achieved in case more radical architectures are considered (e.g., using CIM core to replace the whole memory hierarchy). As CIM core stores a huge amount of data, the main DRAM might become superfluous. This will eliminate the expensive off-chip communication and enable higher performance and energy improvements.
   - Investigate memory controller and communication. The memory controller and the communication between processor and accelerator should be investigated more accurately. Optimizing the inter-communication (i.e.,between processor and CIM core) and intra-communication (i.e., within CIM core) is of great importance. Design exploration and optimization of such components could significantly improve the overall performance and robustness.
   - Explore potential applications. Although we considered several kernels, more applications have to be explored for CIMX architecture. Note that the consid-

ered CIMX architecture contains an accelerator that can only perform logical operations. Hence, it is worth to explore various applications which includes high percentage of logical operations. In fact, some previous work has proposed quite a lot applications with these characteristics such as database processing, graph processing, security encryption, and bio-sequencing [64–69]. In this dissertation, we have shown potential examples such as *QUERY SELECT* kernel (database applications) and *XOR encryption* kernel (security encryption). However, more potential applications can be explored using the proposed analytical and simulation framework, in order to further exploit the potentials of CIMX architecture in particular and CIM core based architectures in general.

- Automate the simulation framework. Due to some manual processes in the simulation, it is currently not straightforward to map more potential applications. Nevertheless, exploring potential applications requires the simulation framework to be fully automated.

CIRCUIT LEVEL

1. **Logic Synthesis Framework**

   - Map complex functions on CIM core. More research is required to map complex functions on the CIM core. Currently, there are only a limited number of operations that can be executed on CIM core. It would be ideal if arithmetic operations could be executed on CIM core. This reduces the communication between processor and main memory even further. Moreover, to increase the performance even more, the instruction set can be extended with macro instructions [73] such as dot product or matrix multiplication [88]. Another approach is to integrate the arithmetic units in the CMOS layer near the memory controller, similarly as carried out in near-data computing [94]. Current prototypes have already demonstrated this approach by stacking memristors on top of a CMOS layer [95, 96].
   - Automate the framework. The proposed framework can be used to generate a lot of conventional circuits that is described HDL to memristor circuits; this can help accelerate the implementing arithmetic circuits and provide inputs for architecture exploration. Therefore, effort is required to automate the proposed framework. In order to do that, it is essential to investigate the technology feasibility (e.g., crossbar isolation, switching latency, endurance, etc.) as well as the complexity and scalability of CMOS control circuit.

2. **Interconnect Network Schemes**

   - Improve large-scale interconnect network. For large CMOS circuits, the interconnect network typically becomes the bottleneck in terms of performance and power consumption. Therefore, for large memristive circuits, this problem might exist as well. Currently, only small scale memristive circuits are explored, however, it is essential to investigate the scalability of the proposed interconnect network in large-scale memristive circuits.

REFERENCES

[1] P. Zikopoulos, C. Eaton *et al.*, *Understanding big data: Analytics for enterprise class hadoop and streaming data*. McGraw-Hill Osborne Media, 2011.

[2] G. Lafuente, "The big data security challenge," *Network security*, vol. 2015, no. 1, pp. 12–14, 2015.

[3] A. Pantelopoulos and N. G. Bourbakis, "A survey on wearable sensor-based systems for health monitoring and prognosis," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 40, no. 1, pp. 1–12, 2010.

[4] S. E. Thompson and S. Parthasarathy, "Moore's law: the future of si microelectronics," *Materials today*, vol. 9, no. 6, pp. 20–25, 2006.

[5] S. Kaxiras, "Architecture at the end of moore," in *Architecture and Design of Molecule Logic Gates and Atom Circuits*. Springer, 2013, pp. 1–10.

[6] Y. Taur, "Cmos design near the limit of scaling," *IBM Journal of Research and Development*, vol. 46, no. 2.3, pp. 213–222, 2002.

[7] S. Hamdioui, S. Kvatinsky, G. Cauwenberghs *et al.*, "Memristor for computing: Myth or reality?" in *Proceedings of the Conference on Design, Automation & Test in Europe*. European Design and Automation Association, 2017, pp. 722–731.

[8] J. McPherson, "Reliability trends with advanced cmos scaling and the implications for design," in *Custom Integrated Circuits Conference, 2007. CICC'07. IEEE*. IEEE, 2007, pp. 405–412.

[9] B. Hoefflinger, "The energy crisis," in *Chips 2020*. Springer, 2011, pp. 421–427.

[10] S. Borkar, "Design perspectives on 22nm cmos and beyond," in *Design Automation Conference, 2009. DAC'09. 46th ACM/IEEE*. IEEE, 2009, pp. 93–94.

[11] H. Sutter, "The free lunch is over: A fundamental turn toward concurrency in software," *Dr. Dobb journal*, vol. 30, no. 3, pp. 202–210, 2005.

[12] S. H. Fuller and L. I. Millett, "Computing performance: Game over or next level?" *Computer*, vol. 44, no. 1, pp. 31–38, 2011.

[13] D. A. Patterson, "Future of computer architecture," in *Berkeley EECS Annual Research Symposium (BEARS), College of Engineering, UC Berkeley, US*, 2006.

[14] A. W. Burks, H. H. Goldstine, and J. Von Neumann, "Preliminary discussion of the logical design of an electronic computing instrument," in *The Origins of Digital Computers*. Springer, 1982, pp. 399–413.

[15] T. Sterling and J. B. Brockman, "Analysis and modeling of advanced pim architecture design tradeoffs," in *in Proceedings of the 6th International Workshop on Innovative Architecture for Future Generation HighPerformance Processors and Systems (IWIA03*. Citeseer, 2003.

[16] S. A. McKee, "Reflections on the memory wall," in *Proceedings of the 1st conference on Computing frontiers*. ACM, 2004, p. 162.

[17] H. Esmaeilzadeh, E. Blem, R. S. Amant *et al.*, "Dark silicon and the end of multicore scaling," in *Computer Architecture (ISCA), 2011 38th Annual International Symposium on*. IEEE, 2011, pp. 365–376.

[18] J. W. Janneck, "Computing in the age of parallelism: Challenges and opportunities," Lund University, Multicore Day Keynote, 2013.

[19] Y. V. Pershin and M. Di Ventra, "Memcomputing: A computing paradigm to store and process information on the same physical platform," in *Computational Electronics (IWCE), 2014 International Workshop on*. IEEE, 2014, pp. 1–2.

[20] S. Hamdioui, L. Xie, H. A. D. Nguyen *et al.*, "Memristor based computation-in-memory architecture for data-intensive applications," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*. EDA Consortium, 2015, pp. 1718–1725.

[21] D. B. Strukov, G. S. Snider, D. R. Stewart *et al.*, "The missing memristor found," *nature*, vol. 453, no. 7191, p. 80, 2008.

[22] R. S. Williams, "How we found the missing memristor," *IEEE spectrum*, vol. 45, no. 12, 2008.

[23] D. L. Lewis and H.-H. S. Lee, "Architectural evaluation of 3d stacked rram caches," in *3D System Integration, 2009. 3DIC 2009. IEEE International Conference on*. IEEE, 2009, pp. 1–4.

[24] K. Suzuki and S. Swanson, "A survey of trends in non-volatile memory technologies: 2000-2014," in *Memory Workshop (IMW), 2015 IEEE International*. IEEE, 2015, pp. 1–4.

[25] C. Xu, D. Niu, N. Muralimanohar *et al.*, "Understanding the trade-offs in multi-level cell reram memory design," in *Design Automation Conference (DAC), 2013 50th ACM/EDAC/IEEE*. IEEE, 2013, pp. 1–6.

[26] P. N. Kumar, M. M. Latha, and K. P. Kumar, "A survey on memristor," *Journal of Innovation in Electronics and Communication Engineering*, vol. 3, no. 1, pp. 66–71, 2013.

[27] O. Mutlu, "Memory scaling: A systems architecture perspective," in *Memory Workshop (IMW), 2013 5th IEEE International*. IEEE, 2013, pp. 21–25.

[28] J. A. Mandelman, R. H. Dennard, G. B. Bronner *et al.*, "Challenges and future directions for the scaling of dynamic random-access memory (dram)," *IBM Journal of Research and Development*, vol. 46, no. 2.3, pp. 187–212, 2002.

[29] A. Maislos *et al.*, "A new era in embedded flash memory," *Flash memory summit*, 2011.

[30] G. S. Sandhu, "Emerging memories technology landscape," in *Non-Volatile Memory Technology Symposium (NVMTS), 2013 13th*. IEEE, 2013, pp. 1–5.

[31] R. Waser, "Redox-based resistive switching memories," *Journal of nanoscience and nanotechnology*, vol. 12, no. 10, pp. 7628–7640, 2012.

[32] S. Bhatti, R. Sbiaa, A. Hirohata *et al.*, "Spintronics based random access memory: a review," *Materials Today*, 2017.

[33] E. Chen, D. Apalkov, Z. Diao *et al.*, "Advances and future prospects of spin-transfer torque random access memory," *IEEE Transactions on Magnetics*, vol. 46, no. 6, pp. 1873–1878, 2010.

[34] J. E. Green, J. W. Choi, A. Boukai *et al.*, "A 160-kilobit molecular electronic memory patterned at 10 11 bits per square centimetre," *Nature*, vol. 445, no. 7126, p. 414, 2007.

[35] M. Radosavljević, M. Freitag, K. Thadani *et al.*, "Nonvolatile molecular memory elements based on ambipolar nanotube field effect transistors," *Nano Letters*, vol. 2, no. 7, pp. 761–764, 2002.

[36] C. Li, D. Zhang, X. Liu *et al.*, "Fabrication approach for molecular memory arrays," *Applied physics letters*, vol. 82, no. 4, pp. 645–647, 2003.

[37] C. Li, W. Fan, B. Lei *et al.*, "Multilevel memory based on molecular devices," *Applied Physics Letters*, vol. 84, no. 11, pp. 1949–1951, 2004.

[38] B. Halg, "On a micro-electro-mechanical nonvolatile memory cell," *IEEE Transactions on Electron Devices*, vol. 37, no. 10, pp. 2230–2236, 1990.

[39] R. Cabrera, E. Merced, and N. Sepúlveda, "A micro-electro-mechanical memory based on the structural phase transition of vo2," *physica status solidi (a)*, vol. 210, no. 9, pp. 1704–1711, 2013.

[40] B. C. Lee, E. Ipek, O. Mutlu *et al.*, "Phase change memory architecture and the quest for scalability," *Communications of the ACM*, vol. 53, no. 7, pp. 99–106, 2010.

[41] S. Raoux, F. Xiong, M. Wuttig *et al.*, "Phase change materials and phase change memory," *MRS bulletin*, vol. 39, no. 8, pp. 703–710, 2014.

[42] R. Waser and M. Aono, "Nanoionics-based resistive switching memories," *Nature materials*, vol. 6, no. 11, p. 833, 2007.

[43] J.-G. Zhu, "Magnetoresistive random access memory: The path to competitiveness and scalability," *Proceedings of the IEEE*, vol. 96, no. 11, pp. 1786–1798, 2008.

[44] G. Fuchs, N. Emley, I. Krivorotov *et al.*, "Spin-transfer effects in nanoscale magnetic tunnel junctions," *Applied Physics Letters*, vol. 85, no. 7, pp. 1205–1207, 2004.

[45] M. Hosomi, H. Yamagishi, T. Yamamoto *et al.*, "A novel nonvolatile memory with spin torque transfer magnetization switching: Spin-ram," in *Electron Devices Meeting, 2005. IEDM Technical Digest. IEEE International*. IEEE, 2005, pp. 459–462.

[46] EU-Commission, "Next generation computing roadmap." European Union, 2014.

[47] J. J. Yang, D. B. Strukov, and D. R. Stewart, "Memristive devices for computing," *Nature nanotechnology*, vol. 8, no. 1, pp. 13–24, 2013.

[48] M. L. Gallo, A. Sebastian, R. Mathis *et al.*, "Mixed-precision memcomputing," *arXiv preprint arXiv:1701.04279*, 2017.

[49] Y. Choi, I. Song, M.-H. Park *et al.*, "A 20nm 1.8 v 8gb pram with 40mb/s program bandwidth," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2012 IEEE International*. IEEE, 2012, pp. 46–48.

[50] R. Fackenthal, M. Kitagawa, W. Otsuka *et al.*, "A 16gb reram with 200mb/s write and 1gb/s read in 27nm technology," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014 IEEE International*. IEEE, pp. 338–339.

[51] T.-y. Liu, T. H. Yan, R. Scheuerlein *et al.*, "A 130.7mm 2-layer 32-gb reram memory device in 24-nm technology," *IEEE Journal of Solid-State Circuits*, vol. 49, no. 1, pp. 140–153, 2014.

[52] R. Waser *et al.*, "Redox-based resistive switching memories–nanoionic mechanisms, prospects, and challenges," *Advanced Materials*, no. 21, pp. 2632–2663, 2009.

[53] H.-S. P. Wong, C. Ahn, J. Cao *et al.*, "Stanford memory trends," 2017. [Online]. Available: https://nano.stanford.edu/stanford-memory-trends

[54] H. Du Nguyen, J. Yu, L. Xie *et al.*, "Memristive devices for computing: Beyond cmos and beyond von neumann," in *Very Large Scale Integration (VLSI-SoC), 2017 IFIP/IEEE International Conference on*. IEEE, 2017, pp. 1–10.

[55] J. Reuben, R. Ben-Hur, N. Wald *et al.*, "Memristive logic: A framework for evaluation and comparison," in *Power and Timing Modeling, Optimization and Simulation (PATMOS), 2017 27th International Symposium on*. IEEE, 2017, pp. 1–8.

[56] H. S. Stone, "A logic-in-memory computer," *IEEE Transactions on Computers*, vol. C-19, no. 1, pp. 73–78, 1970.

[57] D. Patterson, T. Anderson, N. Cardwell *et al.*, "A case for intelligent ram," *IEEE micro*, vol. 17, no. 2, pp. 34–44, 1997.

[58] Y. Kang, W. Huang, S.-M. Yoo *et al.*, "Flexram: Toward an advanced intelligent memory system," in *Computer Design (ICCD), 2012 IEEE 30th International Conference on*. IEEE, 2012, pp. 5–14.

[59] J. T. Pawlowski, "Hybrid memory cube (hmc)," in *Hot Chips 23 Symposium (HCS), 2011 IEEE*. IEEE, 2011, pp. 1–24.

[60] S. Hamdioui, K. L. M. Bertels, and M. Taouil, "Computing device for "big data" applications using memristors," Nov. 21 2017, uS Patent 9,824,753.

[61] J. Yu, T. Hogervorst, and R. Nane, "A domain-specific language and compiler for computation-in-memory skeletons," in *Proceedings of the on Great Lakes Symposium on VLSI 2017*. ACM, 2017, pp. 71–76.

[62] J. Yu, R. Nane, I. Ashraf *et al.*, "Skeleton-based synthesis flow for computation-in-memory architectures," *IEEE Transactions on Emerging Topics in Computing*, 2017.

[63] R. B. Hur and S. Kvatinsky, "Memristive memory processing unit (mpu) controller for in-memory processing," in *Science of Electrical Engineering (ICSEE), IEEE International Conference on the*. IEEE, 2016, pp. 1–5.

[64] S. Li, C. Xu, Q. Zou *et al.*, "Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories," in *DAC*. IEEE, 2016.

[65] V. Seshadri, D. Lee, T. Mullins *et al.*, "Ambit: In-memory accelerator for bulk bitwise operations using commodity dram technology," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2017, pp. 273–287.

[66] V. Seshadri, K. Hsieh, A. Boroum *et al.*, "Fast bulk bitwise and and or in dram," *IEEE Computer Architecture Letters*, vol. 14, no. 2, pp. 127–131, 2015.

[67] J. Ahn, S. Hong, S. Yoo *et al.*, "A scalable processing-in-memory accelerator for parallel graph processing," *ACM SIGARCH Computer Architecture News*, vol. 43, no. 3, pp. 105–117, 2016.

[68] G. Benson, Y. Hernandez, and J. Loving, "A bit-parallel, general integer-scoring sequence alignment algorithm," in *Annual Symposium on Combinatorial Pattern Matching*. Springer, 2013, pp. 50–61.

[69] J. Han, C.-S. Park, D.-H. Ryu *et al.*, "Optical image encryption based on xor operations," *Optical Engineering*, vol. 38, no. 1, pp. 47–55, 1999.

[70] H. A. Du Nguyen, J. Yu, M. Abu Lebdeh *et al.*, "A classification of memory-centric computer architectures," *Submitted*, vol. 0, no. 0, p. 0, In Revision.

[71] H. A. Du Nguyen, J. Yu, M. Abu Lebdeh *et al.*, "A survey of memory-centric computer architectures," *Submitted*, vol. 0, no. 0, p. 0, In Revision.

[72] H. A. Du Nguyen, L. Xie, M. Taouil *et al.*, "Computation-in-memory based parallel adder," in *NANOARCH*. IEEE, 2015.

[73] H. A. Du Nguyen, L. Xie, M. Taouil *et al.*, "On the implementation of computation-in-memory parallel adder," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 8, pp. 2206–2219, 2017.

[74] J. Yu, H. A. Du Nguyen, L. Xie *et al.*, "Memristive devices for computation-in-memory," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2018*. IEEE, 2018, pp. 1646–1651.

[75] L. Xie, H. A. D. Nguyen, J. Yu *et al.*, "Scouting logic: A novel memristor-based logic design for resistive computing," in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2017, pp. 335–340.

[76] H. A. Du Nguyen, L. Xie, J. Yu *et al.*, "A computation-in-memory accelerator based on resistive devices," *Submitted*, vol. 0, no. 0, p. 0, In Revision.

[77] H. A. Du Nguyen, L. Xie, M. Taouil *et al.*, "Synthesizing hdl to memristor technology: A generic framework," in *2016 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*. IEEE, 2016, pp. 43–48.

[78] H. Du Nguyen, L. Xie, J. Yu *et al.*, "Interconnect networks for resistive computing architectures," in *Design & Technology of Integrated Systems In Nanoscale Era (DTIS), 2017 12th International Conference on*. IEEE, 2017, pp. 1–6.

[79] L. Xie, H. A. Du Nguyen, M. Taouil *et al.*, "Interconnect networks for memristor crossbar," in *Nanoscale Architectures (NANOARCH), 2015 IEEE/ACM International Symposium on*. IEEE, 2015, pp. 124–129.

[80] L. Xie, H. A. D. Nguyen, M. Taouil *et al.*, "Fast boolean logic mapped on memristor crossbar," in *Computer Design (ICCD), 2015 33rd IEEE International Conference on*. IEEE, 2015, pp. 335–342.

[81] L. Chua, "Memristor-the missing circuit element," *IEEE Transactions on circuit theory*, vol. 18, no. 5, pp. 507–519, 1971.

[82] P.-E. Gaillardon, L. Amar, A. Siemon *et al.*, "The programmable logic-in-memory (plim) computer," in *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2016, pp. 427–432.

[83] D. Bhattacharjee, R. Devadoss, and A. Chattopadhyay, "Revamp: Reram based vliw architecture for in-memory computing," in *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2017, pp. 782–787.

[84] A. Haj-Ali, R. Ben-Hur, N. Wald *et al.*, "Efficient algorithms for in-memory fixed point multiplication using magic," in *Circuits and Systems (ISCAS), 2018 IEEE International Symposium on*. IEEE, 2018, pp. 1–5.

[85] S. Hamdioui, K. L. M. Bertels, and M. Taouil, "Computing device for "big data" applications using memristors," Nov. 21 2017, uS Patent 9,824,753.

[86] P. Chi, S. Li, C. Xu *et al.*, "Prime: a novel processing-in-memory architecture for neural network computation in reram-based main memory," in *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3. IEEE Press, 2016, pp. 27–39.

[87] D. Fujiki, S. Mahlke, and R. Das, "In-memory data parallel processor," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 2018, pp. 1–14.

[88] J. Yu, L. Xie, M. Taouil *et al.*, "Memristive devices for computation-in-memory," in *Design, Automation and Test in Europe DATE*, 2018.

[89] K. Wu, "Fastbit: an efficient indexing technology for accelerating data-intensive science," in *Journal of Physics: Conference Series*, vol. 16, no. 1. IOP Publishing, 2005, p. 556.

[90] K. K. Soni, R. Vyas, and V. Sharma, "Efficient string matching using bit parallelism," *International Journal of Computer Science and Information Technologies*, 2015.

[91] R. D. Cameron, T. C. Shermer, A. Shriraman *et al.*, "Bitwise data parallelism in regular expression matching," in *Proceedings of the 23rd international conference on Parallel architectures and compilation*. ACM, 2014, pp. 139–150.

[92] D. Lavenier, J.-F. Roy, and D. Furodet, "Dna mapping using processor-in-memory architecture," in *Workshop on Accelerator-Enabled Algorithms and Applications in Bioinformatics*, 2016.

[93] S. Beamer, K. Asanović, and D. Patterson, "Direction-optimizing breadth-first search," *Scientific Programming*, vol. 21, no. 3-4, pp. 137–148, 2013.

[94] P. C. Santos, M. A. Alves, M. Diener *et al.*, "Exploring cache size and core count tradeoffs in systems with reduced memory access latency," in *Parallel, Distributed, and Network-Based Processing (PDP), 2016 24th Euromicro International Conference on*. IEEE, 2016, pp. 388–392.

[95] D. B. Strukov, D. R. Stewart, J. Borghetti *et al.*, "Hybrid cmos/memristor circuits," in *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*. IEEE, 2010, pp. 1967–1970.

[96] K.-T. T. Cheng and D. B. Strukov, "3d cmos-memristor hybrid circuits: devices, integration, architecture, and applications," in *Proceedings of the 2012 ACM international symposium on International Symposium on Physical Design*. ACM, 2012, pp. 33–40.

About the Author

Hoang Anh Du Nguyen was born in Danang city, Vietnam on 25 February 1987. She received the B.E. degree in Electronics and Telecommunications (a 5-year degree with a specialization in Electronics) in 2011 from Danang University of Technology. Thereafter, she concluded her M.Sc. in Computer Engineering from Delft University of Technology in August 2013. Then, in October 2013, she joined the Department of Quantum and Computer Engineering at Faculty of Electrical Engineering, Mathematics, Computer Science at Delft University, the Netherlands, to pursue the Ph.D. degree under the supervision of Dr. ir. Mottaqiallah Taouil, Prof. dr. ir. Said Hamdioui, and Prof. dr. ir. Koen Bertels. Her Ph.D. project was supported by the European project MNEMOSENE. Her research interests include Computer architectures, Resistive Computing, Computation-in-Memory, architectural simulation and memristor-based automation frameworks.

List of Publications

International Journals

4. **H. A. Du Nguyen**, J. Yu, M. Abu Lebdeh, M. Taouil, S. Hamdioui, *A Survey of In-Memory Computer Architectures*, to be submitted.

3. **H. A. Du Nguyen**, J. Yu, M. Abu Lebdeh, M. Taouil, S. Hamdioui, *A Classification of In-Memory Computer Architectures*, under review.

2. L. Xie, **H. A. Du Nguyen**, M. Taouil, S. Hamdioui, K.L.M. Bertels, *A Mapping Methodology of Boolean Logic Circuits on Memristor Crossbar*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), Volume 37, Issue 2, Feb 2018.

1. **H. A. Du Nguyen**, L. Xie, M. Taouil, S. Hamdioui, K.L.M. Bertels, *On the Implementation of Computation-In-Memory Parallel Adder*, IEEE Transactions on VLSI Systems (TVLSI), Volume 25, Issue 8, Aug 2017.

International Symposiums and Conferences

16. **H. A. Du Nguyen**, J. Yu, M. Abu Lebdeh, I. Ashraf, M. Taouil, S. Hamdioui, "A Computation-In-Memory Accelerator based on Resistive Devices," The international symposium on Memory Systems (MEMSYS), Washington DC, USA, September, 2019, pp. 1-14 (to be appeared).

15. **H. A. Du Nguyen**, J. Yu, L. Xie, M. Taouil, S. Hamdioui, D. Fey, *Memristive Devices for Computing: Beyond CMOS and Beyond von Neumann*, IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC), Abu Dhabi, UAE, October, 2017, pp. 1-10

14. **H. A. Du Nguyen**, L. Xie, M. Taouil, S. Hamdioui, K.L.M. Bertels, *Interconnect Networks for Resistive Computing Architectures*, IEEE International Conference on Design & Technology of Integrated Systems In Nanoscale Era (DTIS), Dresden, Palma de Mallorca, Spain, April, 2017, pp. 1-6

13. **H. A. Du Nguyen**, L. Xie, M. Taouil, S. Hamdioui, K.L.M. Bertels, *Synthesizing HDL to memristor technology: A generic framework*, IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH), Beijing, China, July, 2016, pp. 43-48

12. **H. A. Du Nguyen**, L. Xie, R. Nane, M. Taouil, S. Hamdioui, K.L.M.Bertels, *Computation-In-Memory Based Parallel Adder*, IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH), Boston, USA, July, 2015, pp. 57-62

11. J. Yu, **H. A. Du Nguyen**, M. Abu Lebdeh, M. Taouil, S. Hamdioui, *ESL: A Robust Memristor-based Logic Design Resilient to Resistance Variation*, IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH), Qingdao, China, July, 2019, pp. 1-6.

10. M. Abu Lebdeh, U. Reinsalu, **H. A. Du Nguyen**, S. Wong, S. Hamdioui, *Memristive Device Based Circuits for Computation-in-Memory Architectures*, IEEE International Symposium on Circuits and Systems (ISCAS), Sapporo, Japan, May, 2019, pp. 1-5.

9. S. Hamdioui, **H. A. Du Nguyen**, M. Taouil, A. Sebastian, M. Le Gallo, S. Pande, S. Schaafsma, F. Catthoor, S. Das, F. G Redondo, G. Karunaratne, A. Rahimi, L. Benini, *Applications of computation-in-memory architectures based on memristive devices*, Design, Automation & Test in Europe Conference & Exhibition (DATE), Florence, Italy, March, 2019, pp. 1-6.

8. J. Yu, **H. A. Du Nguyen**, M. Abu Lebdeh, M. Taouil, S. Hamdioui, *Time-division Multiplexing Automata Processor*, Design, Automation & Test in Europe Conference & Exhibition (DATE), Florence, Italy, March, 2019, pp. 1-6.

7. L. Xie, **H. A. Du Nguyen**, J. Yu, A. Kaichouhi, M. Taouil, M. Alfailakawi, S. Hamdioui, *Scouting Logic: A Novel Memristor-Based Logic Design for Resistive Computing*,IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Bochum, Germany, July, 2017, pp. 151-156

6. L. Xie, **H. A. Du Nguyen**, J. Yu, M. Taouil, S. Hamdioui, *On the Robustness of Memristor Based Logic Gates*, IEEE International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS), Dresden, Germany, April, 2017, pp. 158-163

5. L. Xie, **H. A. Du Nguyen**, M. Taouil, S. Hamdioui, K.L.M. Bertels, M Alfailakawi, *Non-Volatile Look-up Table Based FPGA Implementations*, IEEE International Symposium on Design and Test (IDT), Hammamet, Tunisia, December, 2016, pp. 165-170

4. L. Xie, **H. A. Du Nguyen**, M. Taouil, S. Hamdioui, K.L.M. Bertels, *Boolean Logic Gate Exploration for Memristor Crossbar*, IEEE International Conference on Design & Technology of Integrated Systems In Nanoscale Era (DTIS), Istanbul, Turkey, April, 2016, pp. 1-6

3. L. Xie, **H. A. Du Nguyen**, M. Taouil, S. Hamdioui, K.L.M. Bertels, *Fast Boolean Logic Mapped on Memristor Crossbar*, IEEE International Conference on Computer Design (ICCD), New York City, USA, October, 2015, pp. 335-342. **Best Paper Award**

2. L. Xie, **H. A. Du Nguyen**, M. Taouil, S. Hamdioui, K.L.M. Bertels, *Interconnect Networks for Memristor Crossbar*, IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH), Boston, USA, July, 2015, pp. 124-129

1. S. Hamdioui, L. Xie, **H. A. Du Nguyen**, M. Taouil, K.L.M. Bertels, *Memristor Based Computation-in-Memory Architecture for Data-Intensive Applications*, Design, Automation & Test in Europe Conference & Exhibition (DATE), Grenoble, France, March, 2015, pp. 1718-1725

Workshops

11. **H. A. Du Nguyen**, L. Xie, M. Taouil, S. Hamdioui, K.L.M. Bertels, *A Synthesis Framework for Memristor Crossbar*, Workshop on Memristor Technology, Design, Automation and Computing (MDAC) in conjunction with HiPEAC, Stockholm, Sweden, January, 2017

10. **H. A. Du Nguyen**, L. Xie, M. Taouil, R. Nane, S. Hamdioui, K.L.M. Bertels, *CIM Based Parallel Adder Implementations and Evaluations*, Workshop on Memristor Technology, Design, Automation and Computing (MDAC) in conjunction with HiPEAC, Prague, Czech, January, 2016

9. **H. A. Du Nguyen**, L. Xie, M. Taouil, R. Nane, S. Hamdioui and K.L.M. Bertels, *CIM Architecture Communication Schemes*, International Workshop on In-Memory and In-Storage Computing with Emerging Technologies (IMISCET) in conjunction with PACT, Haifa Israel, September, 2016

8. **H. A. Du Nguyen**, L. Xie, M. Taouil, R. Nane, S. Hamdioui and K.L.M. Bertels, *CIM Based Parallel Adder Implementations and Evaluations*, ICT.OPEN, Amersfoort, Netherlands, March, 2016

7. L. Xie, **H. A. Du Nguyen**, J. Yu, M. Taouil, S. Hamdioui, *FPGA Implementations Based on Memristor Logic Circuits*, ICT.OPEN, Amersfoort, Netherlands, March, 2017

6. J. Yu, **H. A. Du Nguyen**, L. Xie, M. Taouil, R. Nane, S. Hamdioui, K.L.M. Bertels, *FPGA Implementations Based on Memristor Logic Circuits*, ICT.OPEN, Amersfoort, Netherlands, March, 2017

5. L. Xie, **H. A. Du Nguyen**, M. Taouil, S. Hamdioui, K.L.M. Bertels, *Non-Volatile Look-up Table Based FPGA Implementations*, Workshop on Memristor Technology, Design, Automation and Computing (MDAC) in conjunction with HiPEAC, Stockholm, Sweden, January, 2017

4. L. Xie, **H. A. Du Nguyen**, M. Taouil, S. Hamdioui and K.L.M. Bertels, *Boolean Logic and Interconnect for Memristor Crossbar*, ICT.OPEN, Amersfoort, Netherlands, March, 2016

3. S. Hamdioui, M. Taouil, **H. A. Du Nguyen**, M.A.B. Haron, L. Xie, K.L.M. Bertels, *CIMx: Computation in-Memory Architecture Based on Resistive Devices*, International Workshop on Cellular Nanoscale Networks and their Applications (CNNA), Dresden, Germany, August, 2016

2. L. Xie, **H. A. Du Nguyen**, M. Taouil, S. Hamdioui, K.L.M. Bertels, *Memristor Crossbar Based Logic and Interconnect Design*, Workshop on Memristor Technology, Design, Automation and Computing (MDAC) in conjunction with HiPEAC, Prague, Czech, January, 2016

1. S. Hamdioui, M. Taouil, **H. A. Du Nguyen**, M.A.B. Haron, L. Xie, K.L.M. Bertels, *Memristor: The Enabler of Computation-in-Memory Architecture for Big-Data*, International Conference on Memristive Materials, Devices & Systems (MEMRISYS), Paphos, Cyprus, November, 2015

Other Publications

1. **H. A. Du Nguyen**, Z. Al-Ars, G. Smaragdos, C. Strydis, "Accelerating complex brain-model simulations on GPU Platforms." Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition, Grenoble, France, March, 2015, pp. 1-6.

EPILOGUE

Being a PhD student for five years, I have spent probably half of my time preparing papers (including writing a draft, revising that draft multiple times, and even rewriting a section or the whole paper). Therefore, this book to show my PhD work would be incomplete without a chapter telling my experiences in writing papers and articles (referred to as paper in the rest of this epilogue). Unlike many courses and books that I had a chance to attend and read, I would not consider writing a paper as a practise-based process, that you cannot learn without actually doing it. I would not either agree that writing a paper is an abstract process which is based on the feeling of the authors. Instead, I think that writing a paper consists of several unspoken rules. If the rules are explained well enough, PhD students can spend less time and avoid hurtful experiences in dealing with this matter. The following sections try to explain these rules in a systematic and the simplest way. Note that the following principles can be regarded as the obvious to lots of experienced researchers, however, to my experience, they are vague to new PhD students.

COMPONENTS OF A PAPER

The component of a paper varies depending on its content, however, these following general rules are applied:

- A paper has five mandatory components: abstract, introduction, main contributions (i.e., methodology, implementation, theory), results, and conclusion.
- In addition, there are various optional components such as related work, discussion, future work.
- These components can be organized into one component per section, or more than one component per section. For example, introduction and related work, results and discussion, or discussion and future work are often merged.

Several crucial points are required in each of the mandatory components, which will be explained in the following sections.

ABSTRACT

An abstract is a summary of the paper; it rephrases each section in the paper in one sentence:

- It starts with a motivation in the introduction, summarizes the contributions, states the results, and concludes with the main message of the paper.

- It is normally the most read section in the paper; hence, a key, strong, and necessary message should be stated clearly here.
- The terms that can only be defined later in the paper, and very specific context-based vocabularies should be avoided in the abstract.

INTRODUCTION

Introduction includes three parts; each part comprises of one paragraph:

- Problem statement: this explains the problem and motivates why the work is essential (i.e., to provide the missing solution for a problem).
- State-of-the-art: this explains what has been done to solve the problem so far, why there are still missing solutions, and what makes the work different/outstanding from current published work. This paragraph first lists generic points from published work, compares it with the proposed work, and states that the proposed work can solve partly/entirely the problem.
- Contributions: this first states the main contributions of the paper/articles, then lists all the detailed points of the paper's contributions. This paragraph can also be linked to the related sections in the paper, unless another paragraph is used to lists the organization of the rest of the paper.

MAIN CONTRIBUTIONS

Main contributions present the idea, implementation, and theory that are proposed in the paper; they are the most important body of the paper:

- The main contributions include one or more sections; each normally links with a contribution mentioned in the introduction.
- The main contributions should be presented from the context of the big picture (i.e., system architecture, general concept) to details (component implementation, proof of concept).
- In articles that are based on a previous published papers, a section to summarize the main ideas and results from the published paper is also required.

PRESENTING RESULTS

Presenting results are normally the key to determine the acceptance of the paper. Hence, an effective presentation should directly reflex the main message from the results:

- There are multiple ways to present results; the most effective way is using visualizations such as graph. Note that there are many types of graphs, therefore, it is essential to choose the right type. For example, if the results focus on presenting percentage of a component, a pie graph should be used; if the results focus on the improvements based on increase data sizes, a bar or lines graph should be used.

- Each number/data used in the graph always has a meaning, and focuses on the message of the result. For example, if the difference is the main message, a number showing the difference between two measurements presents effectively the result.
- Each result/graph always needs description and explanation. The results are first described (i.e., the trend of increasing, decreasing), thereafter, explained (i.e., why such results are obtained).

CONCLUSION

A conclusion is the last message of a paper:

- A conclusion concisely summarizes the contributions of the paper, states the obtained results, and relates to the societal/communal impact of the results.
- Lengthy discussions, new insights, and new terminologies should be avoided in the conclusion.

SECTION, PARAGRAPH, AND SENTENCES

There are multiple ways to write; one way is using the top-down model (i.e., first topic sentence, then details) while the other way follows the bottom-up model (i.e., first details explained, then conclusion sentence). However, several common rules still apply:

- Each section should start with a brief introduction of its main idea and components.
- Each paragraph and sentence only focus on one idea.
- Each paragraph and sentence start with a part related to the previous paragraph/sentence, then introduce new content. That creates a flow in the paper.

WORDING AND FORMATTING

Choosing the language style and words to use in a paper is a personal choice, however, several rules are required to avoid confusion:

- Be simple. Complex and rarely used vocabulary should be avoided in scientific writing.
- Defined before used. Unless the abbreviation is widely understood (i.e., UNESCO), it must be defined at least one time before being used in the paper.
- Introduced when required. Whenever the terms, concepts, or ideas are necessarily used, the author should introduce them, not too early (far before from where it is first used) or too late (after it is being used).

Except for special formatting that is required for each conference paper or article, simple formatting rules always apply:

- Be consistent. For example, if a term is used in the introduction, it must be used throughout the paper; once another term is used, either the new term contains a different meaning and needs redefined, or the new term refers to another concept. Another example is about the section titles or figure captions, some authors use all capitalized letters, or capitalized first letters, or no capitalized letter; they are all correct as long as they are consistently used in the paper.
- All figures, tables, and equations are referenced and explained either briefly or in details.
- Citations are required at any statement that is not derived logically from the current text.

DEALING WITH REVIEWERS

There comes a time when the paper is finished, and sent out to reviewers. After a while, that piece of work that every author is proud of comes back, under the eyes of the reviewers with full of questions, doubts, and mistakes. Here comes a time when you have to deal with reviewers in a professional manner. I quote here the tips from IEEE [Authors@IEEE Newsletter: Volume 3, Issue 10, October 2018] that I totally agree with:

- **Appreciate the opportunity to improve.** It's very easy to feel defensive or discouraged when you receive a long list of suggested edits from reviewers. View the suggestions in a positive light instead by seeing them as opportunities to improve your article before publication. The reviewers and the editor have invested significant time in your article to help you improve it for the scientific community.
- **Respond to every comment.** Copy all of the reviewers' suggestions from the decision letter into a new file and separate them into individual suggestions. This is the basis for your response to reviewers, which will be submitted with your revised article. Read each suggestion carefully, implement the appropriate changes in your article, and then explain each change in the response to reviewers document just below the original suggestion. Keep your responses professional, factual, and concise. If you disagree with a reviewer's suggestion, state that you have not implemented the suggestion and provide your reasons for not doing so in the response to reviewers document. The editor may accept your explanation.
- **Read it again before resubmitting.** Set aside the revised article and the response to reviewers and then return and fully read them both again. You may find additional edits when reading through the documents with fresh eyes. Resubmit to the journal once you are satisfied with the revised article and the response to reviewers.

Preparing papers is an essential process of being a researcher. Writing papers is not only a way to convey the idea to the community, but also a way to shape authors' thoughts and organize the work in a better setting. A set of rules is necessary to create a standard format of the writing process, and to save time and energy of researchers, especially PhD students who are always under stress of publications. I hope they are useful.