Delft University of Technology

Computation-in-Memory for Modern Applications using Emerging Technologies

Shahroodi, T.

DOI
10.4233/uuid:80a31436-92dd-4d85-89c6-e8d2e0f5d666

Publication date
2024

Document Version
Final published version

Citation (APA)
Shahroodi, T. (2024). *Computation-in-Memory for Modern Applications using Emerging Technologies*. [Dissertation (TU Delft), Delft University of Technology]. https://doi.org/10.4233/uuid:80a31436-92dd-4d85-89c6-e8d2e0f5d666

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Copyright
Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

Takedown policy
Please contact us and provide details if you believe this document breaches copyrights.
We will remove access to the work immediately and investigate your claim.

This work is downloaded from Delft University of Technology.
For technical reasons the number of authors shown on this cover page is limited to a maximum of 10.

Computation-in-Memory for Modern Applications

USING EMERGING TECHNOLOGIES

Computation-in-Memory for Modern Applications

USING EMERGING TECHNOLOGIES

Proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus prof. dr. ir. T.H.J.J. van der Hagen,
voorzitter van het College voor Promoties
in het openbaar te verdedigen op dinsdag 12 March 2024 om 12:30 uur

door

Taha Shahroodi

Master of Science in Informatik
Eidgenössische Technische Hochschule (ETH) Zürich, Switzerland,
geboren te Teheran, Iran.

Dit proefschrift is goedgekeurd door de promotoren.

promotor: Prof. dr. ir. S. Hmadioui
promotor: Dr. ir. J.S.S.M. Wong

Samenstelling promotiecommissie bestaat uit:

Rector Magnificus,	voorzitter
Prof. dr. ir. S. Hamdioui,	Technische Universiteit Delft, promotor
Dr. ir. J.S.S.M. Wong,	Technische Universiteit Delft, promotor

Onafhankelijke leden:

Dr. ir. S. Stuijk	Technische Universiteit Eindhoven, NL
Prof. dr. ir. I. O'Connor	École Centrale de Lyon, France
Dr. ir. T.G.R.M. van Leuken	Technische Universiteit Delft
Prof. dr. ir. W.A. Serdijn	Technische Universiteit Delft
Dr. S.D. Cotofana	Technische Universiteit Delft
Prof. dr. ir. P.F.A. Van Mieghem	Technische Universiteit Delft, reservelid

Keywords: Computation-in-Memory, Processing-in-Memory, Bioinformatics, Computer Architecture, Hardware/Software Co-Design, Memristor

Printed by: Ipskamp Printing, the Netherlands

Front & Back: "buitenbeentje" by T. Shahroodi.

Copyright © 2024 by T. Shahroodi

ISBN 978-94-6384-534-2

An electronic version of this dissertation is available at
http://repository.tudelft.nl/.

To my loving parents, Dariush and Tahere, and the best brother ever, Emad.

Contents

Summary xiii

Samenvatting xv

Acknowledgements xvii

1 Introduction 1
- 1.1 Motivation . . . 2
- 1.2 Limitations of Traditional Computing Systems . . . 3
- 1.3 Problem Discussion . . . 10
- 1.4 Solution Direction and Research Topics . . . 11
  - 1.4.1 Identify and improve Bioinformatics and Neural Network kernels using CIM . . . 11
  - 1.4.2 Exploring emerging (memory) technologies for CIM . . . 11
- 1.5 Thesis Statement . . . 12
- 1.6 Contributions . . . 12
- 1.7 Thesis Outline . . . 14

2 Background and State-of-the-Art 17
- 2.1 Computation-In-Memory (CIM) . . . 18
  - 2.1.1 CIM designs classification based on computation location . . . 19
  - 2.1.2 Illustration of generic CIM tile . . . 21
  - 2.1.3 Potential Emerging Technologies for CIM . . . 21
  - 2.1.4 Primitive functions in CIM classes . . . 29
  - 2.1.5 Abstraction of CIM Design Choices . . . 33
- 2.2 Modern Applications . . . 35
  - 2.2.1 Bioinformatics and genomics . . . 35
  - 2.2.2 Neural Network . . . 41
- 2.3 State-of-the-Art CIM Designs and Simulators . . . 44
  - 2.3.1 General-Purpose State-of-the-Art CIM Designs and Simulators . . . 44
  - 2.3.2 Specific-Purpose State-of-the-Art CIM Designs and Simulators . . . 46

3 Swordfish 49
- 3.1 Background and Motivation . . . 51
  - 3.1.1 Genome Sequencing Pipeline . . . 51
  - 3.1.2 Basecalling . . . 52
  - 3.1.3 Memristor-based CIM and Associated Non-Idealities . . . 53
  - 3.1.4 Programmable Inference Architecture . . . 55

3.2 Swordfish Framework . . . 55
3.2.1 Swordfish Overview . . . 55
3.2.2 Partition & Map . . . 56
3.2.3 VMM Model Generator . . . 57
3.2.4 Accuracy Enhancer . . . 58
3.2.5 System Evaluator . . . 62
3.2.6 Swordfish Evaluation Challenges . . . 63
3.3 Evaluation Methodology . . . 63
3.3.1 Implementations and Models . . . 63
3.3.2 Simulation Infrastructure . . . 64
3.3.3 Evaluation Metrics . . . 64
3.3.4 Datasets and Workloads . . . 64
3.4 Swordfish Evaluation . . . 65
3.4.1 Effect of Quantization on Accuracy without Accuracy Enhancement . . . 65
3.4.2 Effect of Non-idealities on Accuracy without Accuracy Enhancement . . . 66
3.4.3 Effect of Accuracy Enhancement on Quantized Basecallers . . . 69
3.4.4 Effect of Accuracy Enhancement on Non-idealities . . . 70
3.4.5 Throughput Analysis of SwordfishAccel . . . 74
3.4.6 Area vs. Accuracy Analysis . . . 75
3.4.7 Verdict on Realistic-SwordfishAccel . . . 76
3.5 Discussions and Future Works . . . 76
3.5.1 Applicability of Swordfish Looking forward . . . 76
3.5.2 Other DNN-based Applications . . . 77
3.5.3 Better Accuracy Enhancement Techniques . . . 77
3.6 Conclusion . . . 78

4 RattlesnakeJake 79
4.1 Proposal and Architecture . . . 81
4.1.1 RattlesnakeJake's Algorithm . . . 81
4.1.2 RattlesnakeJake's Architecture . . . 82
4.1.3 RattlesnakeJake Algorithm to Hardware Mapping . . . 84
4.2 Evaluations . . . 84
4.2.1 Evaluation Methodology . . . 84
4.2.2 Accuracy Analysis . . . 85
4.2.3 Throughput and Execution Time . . . 86
4.3 Discussions and Future Works . . . 87
4.3.1 RattlesnakeJake for Long Sequence Alignment . . . 87
4.3.2 Potential Design Improvements . . . 87
4.4 Conclusion . . . 87

5 SieveMem 89
5.1 Motivation and Profiling . . . 91
5.1.1 Shared Kernels in Filters . . . 91
5.1.2 Data Movement in Fitlers . . . 93

5.2 Proposal and Architecture . . . 93
5.2.1 SieveMem Architecture . . . 93
5.2.2 SieveMem Example Support for SHD . . . 94
5.2.3 BandedKrait Algorithm . . . 96
5.2.4 BandedKrait on SieveMem (Mem-BandedKrait) . . . 97
5.3 Evaluations . . . 97
5.3.1 Evaluation Methodology . . . 97
5.3.2 Execution Time of Supported Kernels . . . 98
5.3.3 Filtering Accuracy . . . 99
5.3.4 Filtering Speed . . . 99
5.3.5 End-to-end Alignment Speed . . . 101
5.4 Discussions and Future Works . . . 102
5.4.1 SieveMem for Long Sequence Alignment . . . 102
5.4.2 Potential Design Explorations . . . 102
5.5 Conclusion . . . 102

6 FilterFuse 105
6.1 Motivation . . . 108
6.1.1 Long Reads vs. Short Reads . . . 108
6.1.2 Limitations of SotA filters for long reads . . . 108
6.2 LongGeneGuardian Algorithm . . . 110
6.3 FilterFuse Architecture . . . 111
6.3.1 FilterFuse Overview . . . 111
6.3.2 Tile Architecture . . . 112
6.3.3 Sub-Array Architecture . . . 113
6.3.4 Bank and Bank-Group Architecture . . . 115
6.3.5 Rank Architecture . . . 115
6.3.6 Data Mapping in FilterFuse . . . 117
6.3.7 Long Read Compatibility . . . 118
6.3.8 LongGeneGuardian on Software vs. on FilterFuse . . . 119
6.4 Evaluation Methodology . . . 120
6.5 Evaluation Results . . . 121
6.5.1 Design Space Exploration . . . 121
6.5.2 Filtering Accuracy . . . 122
6.5.3 Filtering Speed . . . 123
6.5.4 End-to-end Alignment Speed . . . 123
6.5.5 Area and Power Analysis . . . 125
6.6 Conclusion . . . 125

7 Demeter 127
7.1 Background and Motivation . . . 129
7.1.1 Metagenomic Profilers . . . 130
7.1.2 Problems of Food Profilers . . . 130
7.1.3 hyperdimensional computing . . . 130

7.2 Demeter . . . 134
7.2.1 Step 1: Define the HD Space . . . 135
7.2.2 Step 2: Build Demeter's Reference Data Structure . . . 135
7.2.3 Step 3: Demeter's Read Conversion . . . 135
7.2.4 Step 4: Multi-Species Classification per Read . . . 136
7.2.5 Step 5: Species Level Abundance Estimation . . . 137
7.3 Demeter's Evaluation . . . 137
7.3.1 Methodology . . . 137
7.3.2 Demeter's Accuracy Analysis . . . 138
7.3.3 Demeter's Software Performance Analysis . . . 139
7.3.4 Demeter's Memory Analysis . . . 141
7.4 Demeter's PIM-enabled Accelerator . . . 142
7.4.1 Overview of Demeter's Accelerator . . . 143
7.4.2 Item Memory (IM) Design . . . 143
7.4.3 Encoder Design . . . 144
7.4.4 Associate Memory (AM) Design . . . 146
7.4.5 Similarity Check Hardware . . . 147
7.4.6 Controller Unit . . . 149
7.5 System Integration of Acc-Demeter . . . 149
7.5.1 Address Translation . . . 149
7.5.2 Coherence . . . 149
7.5.3 Interrupts . . . 149
7.5.4 ISA Extensions and Programming Interface . . . 149
7.6 Acc-Demeter's Evaluation . . . 150
7.6.1 Methodology . . . 150
7.6.2 Acc-Demeter's Performance Analysis . . . 150
7.6.3 Acc-Demeter's Power and Area Analysis . . . 153
7.7 Discussions and Future Works . . . 154
7.8 Conclusion . . . 154

8 KrakenOnMem 157
8.1 Motivation . . . 159
8.1.1 Kraken2's Execution Breakdown . . . 159
8.1.2 Limitation of Previous PIM-enabled Designs . . . 160
8.2 KrakenOnMem Design . . . 161
8.2.1 A High-Level Overview . . . 162
8.2.2 TL-PIM: Matching Mechanism . . . 164
8.2.3 TL-PIM: Taxonomic Retrieval . . . 166
8.2.4 TL-PIM: Controller . . . 167
8.2.5 Relation between LCA-Arrays and Key-Arrays . . . 167
8.2.6 Optimizations . . . 169
8.2.7 KrakenOnMem Profiling Walk Through . . . 169

8.3 Discussions and Future Works . . . 170
8.4 Evaluation Methodology . . . 171
8.5 Experimental Results . . . 172
8.5.1 Performance Analysis . . . 172
8.5.2 Power and Area Analysis . . . 176
8.6 Conclusion . . . 178

9 LightSpeed 179
9.1 TacitMap for BNN . . . 180
9.2 Evaluations . . . 182
9.2.1 Networks and Datasets . . . 183
9.3 Conclusion . . . 184

10 EinsteinBarrier 185
10.1 EinsteinBarrier Architecture . . . 186
10.1.1 oPCM-based WDM-enabled ECore . . . 187
10.1.2 oPCM-based ECore Overheads . . . 190
10.2 Evaluation Methodology . . . 190
10.2.1 Implementations and Models . . . 190
10.2.2 Designs and Baselines . . . 191
10.2.3 Networks and Datasets . . . 191
10.3 Evaluation Results . . . 191
10.3.1 Performance Analysis . . . 191
10.3.2 Energy Analysis . . . 192
10.4 Discussions and Future Works . . . 193
10.5 Conclusion . . . 193

11 Conclusion 195
11.1 Summary . . . 196
11.2 Future Research Directions . . . 199
11.2.1 Extending the Proposed Techniques . . . 199
11.2.2 Leveraging and Cascading the New-Found CIM Designs in End-to-End Pipelines . . . 200
11.2.3 New Bottlenecks after Exploiting the Proposed CIM Designs . . . 202

Epilogue 207

Bibliography 209

Curriculum Vitæ 241

List of Publications 243

SUMMARY

Modern applications like Genomics and Machine Learning (ML) hold the potential to reshape our understanding of diseases' genetic origins and guide machines in executing tasks and making predictions without our explicit programming. The successful, widespread integration of these modern applications can usher in advancements in diagnostics, individualized medicine, and routine tasks such as language interpretation, image analysis, and object categorization. However, our traditional computing infrastructures fall short when accommodating the distinct characteristics of these new applications. Specifically, (1) these applications handle an immense and ever-expanding data working set, and (2) each succeeding version of these applications and their associated use cases necessitates quicker and more energy-efficient analysis of these vast data sets. This is because our traditional computing systems largely hinge on (1) the von-Neumann architecture, a design that distinctly positions processing entities (like CPUs and GPUs) away from storage components (like memories and flash drives), and (2) the CMOS-based technology. While attempting to meet the performance and energy demands of our modern applications, these fully CMOS-based systems based on von-Neumann architecture have increasingly struggled and hit inherent roadblocks, with data movement overhead being the predominant issue.

To alleviate the data movement bottleneck, contemporary research revisits a concept historically known as Computation-In-Memory (CIM) or, alternatively, Processing-In-Memory (PIM). At its core, CIM emphasizes positioning computational capabilities close to, or within, the memory units storing the data. This placement might be within memory chips, in memory controllers, amid caches, or embedded in the logic layers of 3D-stacked memories. As a computational model, architectures leveraging CIM (referred to as CIM architectures) stand to tackle the issue of data movement overhead inherent in the von-Neumann architecture by diminishing or outright eradicating the data movement between computational locales and data storage areas. Moreover, from a technological perspective, emerging memory technologies, including memristive devices and circuits, show potential to replace traditional memory systems, addressing some of the challenges posed by CMOS-based designs.

Irrespective of the specific CIM architecture deployed to optimize performance or energy efficiency in modern applications, there are substantial practical challenges to address and ponder upon first. Both system designers and developers face these hurdles and design decisions, which are critical to surmount CIM's widespread acceptance across various computational areas and application domains.

In this dissertation, our focus is twofold: (1) We delve into the acceleration and streamlined execution of various steps in two pivotal application realms: genomics and ML; and (2) We explore several emerging memory technologies alongside circuit and architectural strategies, that show promise in enhancing CIM designs, specifically tailored for modern applications.

Therefore, in this thesis, we identify and propose strategies and designs to ameliorate the constrained performance of key kernels in genomics and ML. Recognizing that applications within these realms consist of diverse functions or kernels, it is imperative for a designer to possess a thorough understanding of them. Each function/kernel can be characterized by distinct data and control flows, calling for varied features to be enabled in either a von-Neumann or a CIM architecture. To enhance the efficacy of each function/kernel, we first profile them individually and then within a larger context of their corresponding pipeline, followed by discerning the best avenues for their memory mapping in a CIM architecture. We then undertake a concurrent assessment of essential adjunct components alongside the memory array, commonly referred to as the peripheries. For a designer, proficiency in the applications executable on a CIM system leveraging emerging memory technologies is indispensable. Grasping the fundamental characteristics of CIM and having an overarching view of its scope becomes vital prior to its integration. We aim to aggregate critical application features, improvement opportunities, and design decisions and refine them to their core essence. Through this, we aspire to shed light on present design options and identify kernels demanding heightened attention. Such insights can be instrumental in revealing prospective directions, encompassing supported kernels along with their respective merits and trade-offs.

We exploit emerging technologies and architect state-of-the-art CIM designs that optimally serve the targeted kernels, keeping a holistic improvement perspective at the forefront. Delving into emerging (memory) technologies, such as memristive devices like PCM and STT-MRAM, is crucial. These devices provide a suite of advantages, including non-volatility, compactness, and a natural aptitude for conducting logical operations (for instance, the logical AND). Additionally, other emerging technologies, such as integrated photonics, have the potential to enhance the CIM paradigm further with their capacity for high-frequency and low-latency functions. Our ambition is to integrate multiple such technologies, harnessing their distinct attributes, to craft a CIM design that surpasses the SotA counterparts across key benchmarks, be it in execution speed or energy.

This thesis demonstrates that when CIM is fused with emerging (memory) technologies, there is a marked enhancement in the performance of several Genomics pipelines and Machine Learning applications. It is our aspiration and conviction that the evaluations, methodologies, and findings detailed in this dissertation will empower the broader community to comprehend and address contemporary and upcoming challenges that revolve around enhancing the performance and energy efficiency of modern applications through the integration of (re)emerging computing paradigms and technologies. Additionally, our work provides insights for adapting these technologies to novel applications, ensuring they deliver optimal benefits.

SAMENVATTING

Moderne toepassingen zoals Genomics en Machine Learning (ML) hebben het potentieel om ons begrip van de genetische oorsprong van ziekten te hervormen en machines te begeleiden bij het uitvoeren van taken en voorspellingen zonder expliciete programmering. De succesvolle, wijdverbreide integratie van deze moderne toepassingen kan vooruitgang betekenen op het gebied van diagnostiek, gepersonaliseerde geneeskunde en routinetaken zoals taalinterpretatie, beeldanalyse en objectcategorisatie. Onze traditionele computerinfrastructuren schieten echter tekort bij het om de kenmerken van deze nieuwe toepassingen te accommoderen. In het bijzonder behandelen deze toepassingen een enorme en steeds uitbreidende dataset (1), en elke volgende versie vereist snellere en energiezuinigere analyse van deze enorme datasets (2). Dit komt doordat onze traditionele computersystemen voornamelijk afhankelijk zijn van de von-Neumann-architectuur (1) en op CMOS-technologie (2) gebaseerd zijn. Bij het proberen te voldoen aan de prestatie- en energie-eisen van moderne toepassingen, hebben deze volledig CMOS gebaseerde von-Neumann-architecturen steeds meer problemen ondervonden en lopen ze tegen inherente obstakels, waarvan het dataverplaatsing bij gegevensverwerking het belangrijkste probleem is.

Om het probleem van dataverplaatsing bij gegevensbeweging aan te pakken, herziet hedendaags onderzoek een concept dat historisch bekend staat als Gegevensverwerking-In-Geheugen (GIG). In de kern benadrukt GIG het plaatsen van rekenkundige mogelijkheden dichtbij, of binnen, de geheugeneenheden die de gegevens opslaan. Deze plaatsing kan in geheugenchips zijn, in geheugencontrollers, te midden van caches of ingebed in de logische lagen van 3D-geheugens. Architecturen die gebruik maken van GIG (aangeduid als GIG-architecturen) hebben als doel het probleem van gegevensbeweging dat inherent is aan de von-Neumann-architectuur aan te pakken door de gegevensbeweging tussen verwerkingslocaties en geheugens te verminderen of volledig te elimineren. Bovendien tonen opkomende geheugentechnologieën, waaronder memristorcomponenten en -circuits, potentieel om traditionele geheugensystemen te vervangen en enkele uitdagingen van op CMOS gebaseerde ontwerpen aan te pakken.

Ongeacht de specifieke GIG-architectuur die wordt ingezet om de prestaties of energie-efficiëntie in moderne toepassingen te optimaliseren, zijn er aanzienlijke praktische uitdagingen die aangepakt moeten worden. Zowel systeemontwerpers als ontwikkelaars staan voor deze hindernissen en ontwerpbeslissingen, die cruciaal zijn voor het overwinnen van de brede acceptatie van GIG in verschillende computergebieden en toepassingsdomeinen. In deze dissertatie ligt onze focus op twee aspecten: (1) We duiken in de versnelling en stroomlijning van verschillende stappen in twee cruciale toepassingsgebieden: genomics en ML; en (2) we verkennen verschillende opkomende geheugentechnologieën naast circuit- en architectuurstrategieën, die belofte tonen om GIG-ontwerpen te verbeteren die specifiek afgestemd zijn op moderne toepassingen.

Daarom identificeren en stellen we in dit proefschrift strategieën en ontwerpen voor

om de beperkte prestaties van essentiële kernels in genomics en ML te verbeteren. Omdat toepassingen binnen deze gebieden bestaan uit diverse functies of kernels, is het cruciaal voor een ontwerper om een grondig begrip hiervan te hebben. Elke functie of kernel kan worden gekenmerkt door verschillende gegevens- en besturingsstromen, wat vraagt om verschillende functies in zowel een von-Neumann- als een GIG-architectuur. Om de effectiviteit van elke functie of kernel te vergroten, profileren we ze eerst individueel en vervolgens binnen de grotere context van hun pipeline, gevolgd door het bepalen van de beste geheugenmapping in een GIG-architectuur. Vervolgens voeren we een gelijktijdige beoordeling uit van essentiële aanvullende componenten naast de geheugenarray, vaak aangeduid als de perifere componenten. Voor een ontwerper is bekwaamheid in de toepassingen die kunnen worden uitgevoerd op een GIG-systeem dat gebruik maakt van opkomende geheugentechnologieën onmisbaar. Het begrijpen van de fundamentele kenmerken van GIG en het hebben van een alomvattend beeld van de reikwijdte ervan is cruciaal voorafgaand aan de integratie. We streven ernaar om kritieke toepassingskenmerken, verbeteringsmogelijkheden en ontwerpbeslissingen samen te brengen en ze te verfijnen tot hun kern. Hierdoor hopen we inzicht te verschaffen in de huidige ontwerpmogelijkheden en kernels te identificeren die meer aandacht vergen. Dergelijke inzichten zijn instrumenteel om toekomstige richtingen te bepalen, waaronder de ondersteunde kernels samen met hun respectieve voordelen en afwegingen.

We maken gebruik van opkomende technologieën en ontwerpen GIG-architecturen die optimaal de kernels implementeren, met een holistisch verbeteringsperspectief voor ogen. Het verkennen van opkomende (geheugen)technologieën, memristorcomponenten zoals PCM en STT-MRAM, is cruciaal. Deze componenten bieden een reeks voordelen, waaronder niet-vluchtigheid, compactheid en een inherente geschiktheid voor het uitvoeren van logische bewerkingen (bijvoorbeeld de logische EN-operatie). Daarnaast hebben andere opkomende technologieën, zoals geïntegreerde fotonica, het potentieel om het GIG-paradigma verder te verbeteren door hun hoogfrequente en snelle eigenschappen. Onze ambitie is om meerdere van dergelijke technologieën te integreren, gebruik makend van hun unieke kenmerken, om een GIG-ontwerp te creëren dat de tegenhangers overtreft op belangrijke benchmarks, zowel qua uitvoeringssnelheid als energieverbruik.

Dit proefschrift toont aan dat wanneer GIG wordt samengevoegd met opkomende (geheugen)technologieën, er een merkbare verbetering is in de prestaties van verschillende genomics-pipelines en Machine Learning-toepassingen. Het is onze ambitie en overtuiging dat de evaluaties, methodologieën en bevindingen die in dit proefschrift worden beschreven, de bredere gemeenschap in staat zullen stellen om hedendaagse en toekomstige uitdagingen die draaien om het verbeteren van de prestaties en energie-efficiëntie van moderne toepassingen door de integratie van (her)opkomende rekenparadigma's en technologieën te begrijpen en aan te pakken. Bovendien biedt ons werk inzichten voor het aanpassen van deze technologieën aan nieuwe toepassingen, zodat ze optimale voordelen kunnen opleveren.

ACKNOWLEDGEMENTS

This thesis marks three years of my efforts throughout my graduate studies and many individuals who, directly or indirectly, have contributed to its completion or how I experienced it. Therefore, I would like to take this moment to highlight them and express my gratitude.

First, I thank my promoters, **Said Hamdioui** and **Stephan Wong**. In the research group **Said** established at TU Delft, I have gained essential skills that will shape my career path. **Said**, from the outset of my Ph.D., you placed faith in my research and my organizational skills, even nominating me as our research group's representative on the department's Ph.D. council. For that, I am thankful.

I thank **Stephan Wong**, my promoter, daily supervisor, and constant advocate. **Stephan**, your astute feedback on my work has been pivotal in advancing my research. Your extensive experience across research, academia, and administrative tasks and willingness to share them have enriched my understanding of the system. Beyond this, I also deeply appreciate the autonomy you granted me in exploring research avenues I was passionate about. Your faith in me was evident when you entrusted me with teaching responsibilities, shared your instructional methods and early graduation techniques, and backed my internships. Most crucially, your unwavering presence, shielding me from internal and external strife and uplifting my spirit during challenging times, has been invaluable. My heartfelt thanks for everything.

I owe an immeasurable gratitude to my enduring advisor and life mentor, **Onur Mutlu**. Having the privilege of being his student since 2018 has been transformative. I am deeply thankful for his unwavering support and guidance throughout this journey. **Onur**, your persistent push for grand visions, unyielding commitment to research, and challenges to elevate my capabilities have sculpted me as a researcher standing here. The environment you fostered within SAFARI, coupled with your unwavering trust in me, has endowed me with the insights, tools, and perspective essential for my personal and professional evolution. I am confident that the wisdom and skills imparted by **Onur** and those gained under his supervision will remain integral to my growth in research, science, and engineering. Cheers, **Onur**.

I am grateful to the committee members of this thesis, **Sander Stuijk**, **Ian O'Connor**, **Rene van Leuken**, **Wouter Serdijn**, and **Sorin Cotofana**. Your insightful feedback has been crucial in enhancing my thesis, and I eagerly anticipate witnessing the influence of our future collaborative efforts.

I would like to extend my gratitude to my academic partners. **Ian O'Connor** and **Alberto Bosio**, thank you for welcoming me into your research group. The knowledge you passed on about optical phase change memory and FeFET, the critical discussions on my work, and the essential measurement data you provided have significantly enhanced my research journey. The one-on-one meetings in Lyon hold particular importance to me. I am optimistic about the continuance of our rewarding collaboration. My thanks

to **Asif Ali Khan** for bridging our collaboration. Our technical exchanges, particularly those centered around pre-alignment filters and racetrack memory, were enlightening. I am also grateful for the platform you provided me to present a technical discourse on my research at the Chair for Compiler Construction (CCC) in TU Dresden. Moreover, our casual, non-research conversations during our interactions in Germany added a pleasant dimension to our professional relationship. My sincere thanks to each one of you.

I am grateful to researchers and members of the various departments at TU Delft, the SAFARI research group at Eidgenössische Technische Hochschule (ETH) Zürich, and Institut des Nanotechnologies de Lyon (INL) at École Centrale de Lyon (ECL) for their steadfast support throughout this journey. While the path to a Ph.D. can often feel solitary and lonely, each of you played a role in mitigating that solitude, transforming what is typically an individual endeavor into a collective experience. **Mahdi**, your presence as a sounding board and officemate has been invaluable. I also extend my heartfelt gratitude to other members of the TU Delft family, including but not limited to, **Robert**, **Moritz**, **Christiaan**, **Folkert**, **Sumit**, **Amin**, **Abdullah**, **Motta**, **Alireza**, **Michael**, **Guilherme**, **Matti**, **Asmae**, **Hanzhi**, **Abdulqader**, **Rajendra**, **Sicong**, **Pantazis**, **Fouwad**, **Arne**, **Mark**, **Yash**, **Heba**, **Francis**, and **Paul**. Our shared moments, from delightful lunches and daily coffee breaks to numerous discussions, have been highlights of my time in the office. Equally, my time at INL and SAFARI was enriched by interactions with friends like **Minesh**, **Haocong**, **Skanda**, **Rahul**, **Ataberk**, **Raphael**, **Tracy**, and many more. I cherish the moments of academic collaboration and the bonds of friendship forged. Your support has been a cornerstone in my academic and personal growth, and I wish each of you success and happiness in your future endeavors. Thank you all.

Throughout my life, spanning the Netherlands, Switzerland, France, Germany, and Iran, I was fortunate to encounter many incredible souls who enriched my life beyond the confines of my work. **Mina**, **Danial**, and **Bahador**, I cherish our enduring friendship, one that defies the vast geographic distances between us. Your messages and laughs always brightened my mornings. My heartfelt gratitude extends to **Shibashish**, **Melanie**, **Sarah**, **Skanda**, **Maria**, **Minesh**, **Linus**, **Julius**, **Ninad**, and **Sandra**, housemates, officemates, classmates, or clubmates, who transitioned into my dear close friends and loved ones. Our shared moments, from our late-night discussions and explorations in the Zürich forests and city to aimless rides and random events in Italy or the Netherlands, have been foundational in making my journey so fulfilling. You have been the backbone of encouragement and positivity, always ready to lend an ear, and your ever-present support, abundant encouragement, and undying positivity were all one can ask for. My dearest friends, your consistent companionship was instrumental in helping me find my footing in Zürich and the Netherlands. From introducing me to the city's nuances to crafting countless treasured memories, I am deeply appreciative. Here is to more delightful evenings at the lake, in the forest, or in our fantastic hobby room and gardens. Lastly, a big shout-out to my sports companions, **Karin**, **Bram**, **Florence**, **Fiske**, and **Thomas**. Your relentless mantra of "5×/week, no excuses", our "4 more" signals, tough sparrings, and our regular dinners and meetings marked by spirited dialogues on topics ranging from coding in Rust and hackathons to finance to fashion have been truly exhilarating. Every minute with you was not just about health or fitness but also about bonding and growth. Cheers to all of you and the memories shared, and to many more in the future.

I am eternally grateful to my family and friends from back home or abroad. Their deep love and enduring support have been the bedrock upon which I have built my dreams. A special mention goes to my cousin, **Eiman**. Despite the whirlwind of leading a highly successful startup, **Eiman** always found moments to offer me priceless advice and wisdom. I extend my gratitude to **Ali** for his unwavering support in the Netherlands, both emotionally and physically. Our dinners at his residence were delightful and a highlight for my weeks. To my beloved grandparents, aunts, uncles, and extended family, your unwavering support has been a cornerstone of my endeavors. Those occasional messages, brimming with love and buoyant words of motivation, have been my guiding light, pushing me forward even during challenging times. Your faith in me has made all the difference, and I remain forever indebted to your kindness and belief.

To the very foundation of my being, the pillars of my strength and determination, I owe the deepest debt of gratitude to my parents, **Dariush** and **Tahere**, and my brother, **Emad**: Every step I took in this demanding world of scientific endeavor, I found my footing on the strong base you laid down for me. Your tireless dedication, unyielding efforts, and immeasurable love were my guiding stars. **Dariush**, from you, I learned the art of resilience, even in life's harshest moments and in the face of profound sorrow. Your wisdom has been the compass by which I have navigated challenges. The world could shift under my feet, but the lessons you imparted and the love you have shown me remain steadfast. Thank you! **Tahere**, in every setback and moment of self-doubt, your emotional anchor and words of encouragement set me back on course. Your unwavering belief in me and gentle encouragement have always been my saviors. **Emad**, my dear brother, the bond we share is beyond words. You have been a sanctuary of patience, an embodiment of love, and a wellspring of support. Our shared moments, be they of joy, fights, or simply day-to-day life, are treasures in my heart. Even when I was difficult and lost my way, you stood steadfast, echoing your belief in me. Beyond just being siblings, our bond has grown into an unrivaled camaraderie. You have witnessed my peaks and valleys, constantly extending a hand of support, love, and understanding. My heart brims with gratitude for every gesture, word, and a lifetime of shared memories and moments with you. **Emad**, not just my brother, but my beacon and inspiration, thank you!

1

Introduction

The minute you get away from fundamentals
–whether its proper technique, work ethic, or mental preparation–
the bottom can fall out of your game, your schoolwork, your job, whatever you are doing.

Michael Jordan

Modern applications such as genomics and Machine Learning (ML) promise transformative advancements to human lives. However, the execution of these applications has bottlenecked with traditional computing infrastructures, primarily due to data movement overheads inherent in von-Neumann and CMOS-based designs. Recently, researchers explored the Computation-In-Memory (CIM) paradigm to reduce these overheads, placing computation close to memory storage. When combined with emerging technologies like memristive devices, such CIM architectures can replace traditional systems and address data movement overhead and some other limitations inherent to fully CMOS-based technologies. However, implementing CIM brings practical challenges for system designers and developers, impacting widespread acceptance. In this chapter, we first discuss the impacts of modern applications on our lives and their unique features separating them from traditional ones. We then discuss the limitations of traditional systems mainly based on von-Neumann architecture and CMOS technology for high-performance and/or energy-efficient execution of modern applications. We motivate CIM and emerging memory technologies as a potential solution for efficient, futuristic systems for modern applications. After that, we discuss the main opportunities and challenges such CIM designs introduce. Following, we discuss the research directions and topics this thesis explores. Finally, we present our thesis statement and list our contributions and thesis organization.

1

1.1. MOTIVATION

Genomics and Machine Learning (ML) are two examples of modern applications revolutionizing our world. Genomics is a branch of genetics that investigates the interactions between genes and non-genic genome regions and how they mold observable traits and biological functions. Genomics brings about a remarkable increase in our comprehension of biology and illness. Genomics sheds light on the genetic foundations of diseases, paving the way for superior diagnostics and treatment approaches. This way genomics lays the groundwork for progress in personalized medicine, allowing for treatment methods to be customized based on a person's genomic data. Advancing genomics studies that hold the key to unlocking the potential of precision medicine, facilitating virus surveillance, and driving advancements in healthcare [1–17].

ML is an application domain focusing on the development of algorithms and statistical models for computer systems to "learn" from experience, i.e., to perform tasks and make decisions/predictions without being explicitly programmed to do so. Currently, we use ML applications in many of our day-to-day tasks such as language processing [18] object recognition [19], and image classification [20, 21]. In particular, momentous developments in Deep Neural Network (DNN) in the past decade have led to significant improvements in the accuracy and execution time of computer vision tasks such as object detection and recognition [22–24].

These modern applications, i.e., genomics and ML, share two main critical features:

- **Their data working set size is rapidly growing.** For example, one can characterize many of the existing genomics pipelines as simple and data-hungry sequences of operations that require high parallelization. Fig. 1.1 depicts the significant reduction in the cost of data acquisition in genomics, next to Moore's law ascending growth, helping the exponential growth of the data working set size of its kernel. It is estimated that the working datasets of the genomics domain scale faster than those produced by YouTube and Twitter for Machine Learning-based (ML-based) applications [25–27].

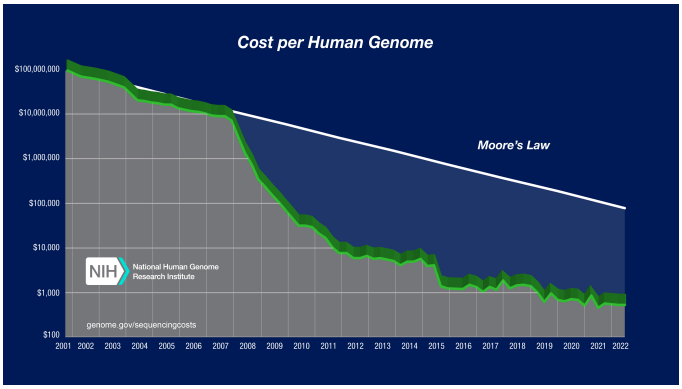

Figure 1.1: The nature of the reductions in DNA sequencing costs over the years against Moore's Law [26].

ML applications are the same. For another example, Fig. 1.1 presents the trend in training datasets of language-based ML over the years. The growth of the data working set is apparent from the models developed for this task. Other ML tasks follow a similar trend (Section 2).

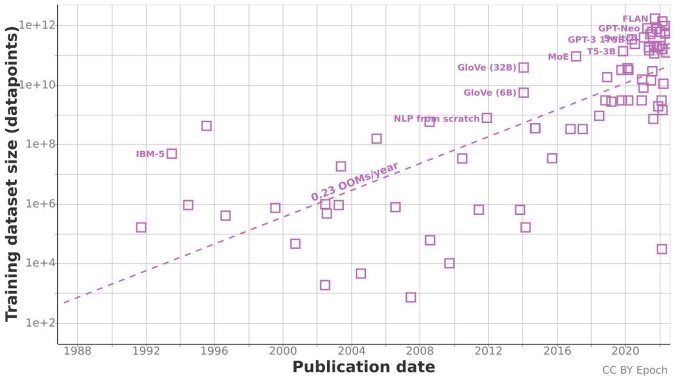

Figure 1.2: The growth rate of dataset size in ML for language based tasks such as text generation and classification [28].

- **The demand for faster analysis of such large data working sets is increasing.** Particularly in genomics, the swift processing of genomics data plays a crucial role in unleashing the full potential of precision medicine, bolstering virus surveillance capabilities, and propelling advancements in healthcare. Moreover, ML applications also demand fast analysis for enhanced automation, expedited explorations, and more efficient decision-making processes.

Unfortunately, our traditional computing systems only rely on von-Neumann architecture [29–31] and CMOS technology that fundamentally have already faced major limitations while they tried to keep up with the high performance and energy-efficiency demand of genomics and ML applications [15, 32].

1.2. LIMITATIONS OF TRADITIONAL COMPUTING SYSTEMS

Traditional computing systems adopt solely on CMOS technology and the von-Neumann architecture: a processor-centric architecture that separates the processing units (CPUs, GPUs, etc.) and storage units (memories, flashes, etc.) [29–31]. This choice introduces some challenges and limitations for our modern applications that we discuss next.

The sole reliance on CMOS technology recently introduced technological limitations when we try to keep up with the high performance and energy-efficiency demand of modern applications [32]. In the past decade, we have witnessed significant advancements in semiconductor technology regarding the feature size of the transistors and the number of transistors on a single chip. These advances advocate two famous laws in

computer architecture: Moore's Law and Dennard's Law. Moore's Law [33] and Dennard's Law [34] are two closely-related observations regarding the advancement of semiconductor technology, focusing on different aspects. Moore's Law states that the number of transistors on a chip will double approximately every two years. This exponential growth leads to an increase in computational power. Dennard's Law, on the other hand, targets the power consumption and performance scaling of transistors as their feature size decreases, i.e., they shrink in size. Dennard's Law states that as transistors become smaller, their power density remains constant. This allows for increased performance at the same power consumption level.

Unfortunately, semiconductor advancements regarding Moore's and Dennard's laws are slowly coming to an end due to three main limitations:

- **Leakage Wall:** The continuous scaling of transistors results in the reduction of gate thickness and channel length between the drain and source, shrinking down to a few atomic layers. Consequently, there is an increased likelihood of quantum mechanical tunneling, leading to higher gate leakage current. Moreover, shorter channels can contribute to elevated off-state drain leakage. Additionally, as supply and threshold voltages are scaled down, the static power during the off state of the transistor experiences an increase, forming a leakage wall.

- **Reliability Wall:** With the scaling down of transistor size, even minor variations in the fabrication processes can significantly impact the functionality of the transistors, forming a reliability wall. Additionally, the continued scaling poses challenges in terms of reliable insulation and conduction due to limitations in the dielectric and wiring materials.

- **Cost Wall:** As the size of transistors decreases, the cost of fabrication experiences an exponential increase. This cost escalation is driven by various factors such as equipment, lithography processes, masks, and testing expenses, forming a cost wall.

These limitations motivate us to seek alternative technologies to continue our performance and energy improvements with less reliance on scaling the transistors' feature sizes and their number on a specific chip.

To understand the architectural limitations arising from adapting von-Neumann architecture, Fig. 1.3 depicts this architecture first.

As Fig. 1.3 shows, a von-Neumann architecture comprises three main components: ❶ a processing unit (e.g., central processing unit (CPU) or graphics processing unit (GPU)), ❷ a memory, and ❸ input or ❹ output devices. The processing unit executes the program instructions and performs logical or arithmetic operations on the data. The data is loaded from the memory or the input devices, and the results are written back into the memory or sent out to the output devices. The Von Neumann architecture offers four advantages making it suitable for our traditional computing systems and applications:

- Simplicity: The Von Neumann architecture is simple and straightforward, making it easy to understand and implement. This architecture separates the computer's mem-

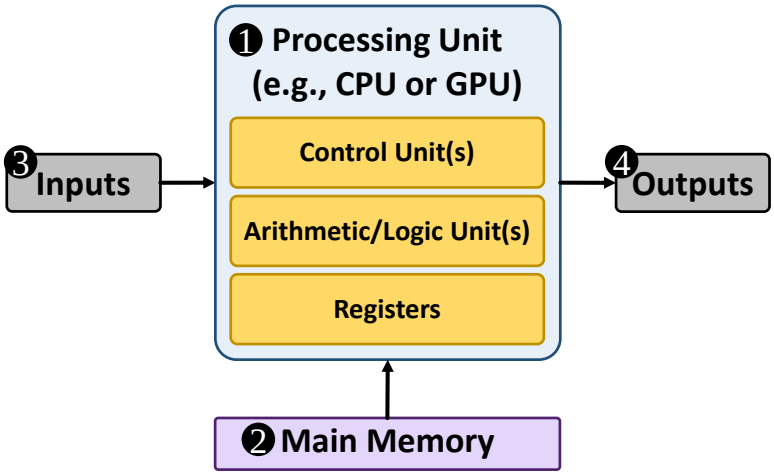

Figure 1.3: Von-Neumann architecture, i.e., processor-centric architecture, in traditional computing systems.

ory into two distinct types: one for instructions and one for data. This architecture follows the fetch-decode-execute cycle.

- Uniformity: Using a single memory space for both data and instructions simplifies the architecture. This also brings versatility. Such a clear separation allows instructions to be stored in the same form as data. This means that any program can be treated as data and manipulated accordingly.

- Self-modifying code: This architecture allows a program to modify itself since data and instructions are stored in the same memory.

- Hardware Cost: The Von Neumann architecture reduces the hardware cost by utilizing the same memory and peripheries (e.g., buses) for both instructions and data.

- Precedent: The Von Neumann architecture is well-understood and widely used. This establishes a precedent that advocates other advances in computer technology.

Using a single-level, large memory unit in the von-Neumann architecture historically introduced two main challenges for traditional applications: (1) high cost regarding latency and energy consumption for data transactions between the CPU and main memory and (2) requirement of very large memory unit to accommodate the large data working set. For example, previous work [35] shows that the simple off-chip communication between the memory unit (DRAM) and CPU can take up to 200 CPU cycles. Other works [36, 37] show that the energy consumption for moving data for a floating point operation to the CPU consumes more than two orders of magnitude more energy than the energy for the operation within the CPU.

However, system designers overcome these challenges by proposing a hierarchy for memory in the von-Neumann architecture [29, 31]. Fig. 1.4 depicts this memory hierarchy in the traditional computing systems based on von-Neumann architecture.

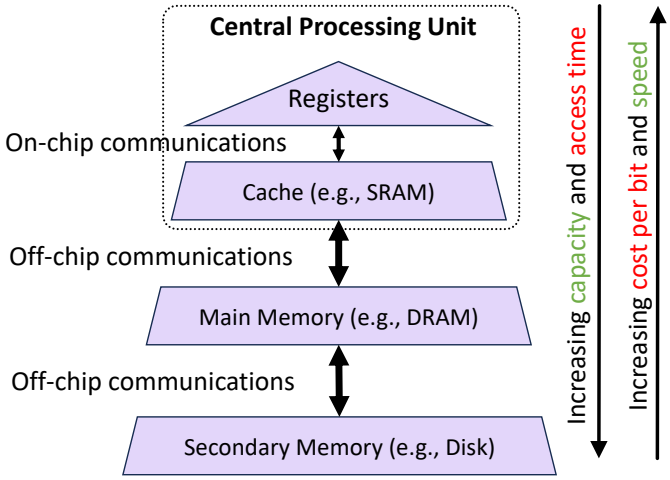

Figure 1.4: Memory hierarchy in traditional von-Neumann-based systems.

The memory hierarchy tackles the high cost of data movement by exploiting the temporal locality of data and having an SRAM-based cache memory located in the same chip as the CPU. This SRAM cache reduces the access time for the data. However, due to cost reasons (e.g., area overhead), designers keep SRAM caches smaller than the main memory and typically smaller than the data working set of even traditional applications. This hierarchy also tackles the problem of larger than the main memory problem of traditional applications by placing secondary memory storage, also known as a disk. The operating system (OS) manages data transfer from the disk (i.e., secondary memory) to the main memory. Current systems use flashes or hard disk drivers for this secondary memory. Although the secondary memory scales up easily, access time would be much higher due to the technology limitation and architectural complexity.

The sole reliance on the aforementioned von-Neumann architecture and memory hierarchy comes with its own architectural limitations. Beyond just technological advancements in semiconductors, the evolution of architectural design has been instrumental in enhancing the efficiency of computer systems over the past several decades. Nonetheless, the potential for performance enhancement from this factor has recently become limited due to three well-known walls:

- **Power Wall:** The miniaturization of transistors has also allowed a greater number to fit within a fixed area. Table 1.1 illustrates this trend, demonstrating the increased transistor count in various Intel processor generations. With this increase in the number of transistors, processors can now support multiple cores, allowing parallel task execution. This, however, escalates power consumption corresponding to the increased core count. Nevertheless, the amount of power that a chip

can draw is finite, and dissipating heat effectively from these chips poses a significant challenge. Consequently, the 'Dark Silicon' phenomenon arises, evidenced by over half of a processor chip remaining inactive at any one time due to power constraints [38]. Fig. 1.5 depicts this phenomenon where the clock frequency, directly linked to power usage and system performance, plateaued recently. These trends underscore that enhancements in system performance can no longer rely solely on escalating clock frequencies and core counts.

Intel Processor	Transistor Count	Year
Intel 4004	2300	1971
Intel 8086	29000	1978
Intel i860	1000000	1989
Pentium 1	3100000	1993
Pentium 4	112000000	2004
Core i7	731000000	2008
Quad-core + GPU i7	1160000000	2011
Quad-core + GPU i7 Ivy Bridge	1400000000	2012
Quad-core + GPU GT2 i7 Skylake	1750000000	2015
28-core Xeon Plantinum 8180	8000000000	2017

Table 1.1: The number of transistors in different generations of Intel processors over the years.

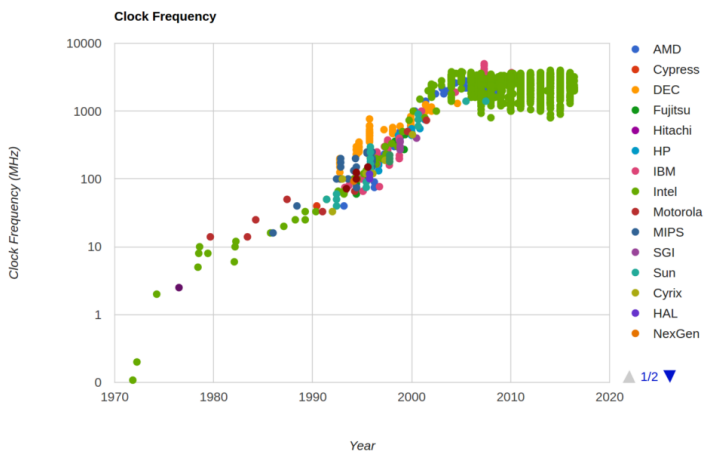

Figure 1.5: Maximum clock frequency achieved by a range of processors across the last several decades [39].

- **ILP Wall:** Instruction Level Parallelism (ILP), which enables the simultaneous execution of multiple instructions, is another potential route for boosting system performance. However, its efficiency is contingent on the innate parallelism of the applications or algorithms. Beyond this threshold, increasing hardware parallelism does not correspondingly raise performance levels. Fig. 1.6 contrasts the predicted performance enhancement per Moore's Law with the actual performance boost as

the transistor size is reduced. A key factor contributing to this discrepancy is the inherent limit on application parallelism.

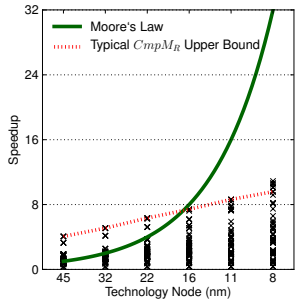

Figure 1.6: Speedup across various process technology nodes, encompassing all organizations and topologies of PARSEC benchmarks [38].

- **Memory Wall:** While processors have seen substantial improvements in energy efficiency and performance, data transmission through I/O ports has lagged behind in terms of progression. As previously mentioned, communication between the processor and off-chip memories, including primary and secondary memory, takes a minimum of 200 CPU cycles [35], and it consumes significantly more power than an internal CPU floating-point operation [36, 37]. Additionally, the volume of data transferable concurrently is capped by the chip's I/O port count. This phenomenon is known as the memory wall. The memory wall significantly influences a system's efficiency, both in energy consumption and performance terms.

These architectural limitations motivate us to seek alternative architectures that can tackle power, ILP, and memory walls and enable our continual improvements regarding performance and energy.

To make matters worse, as discussed previously, the data working set size of modern applications such as Machine Learning or genomics is rapidly growing. The features and requirements of modern applications, with the existing challenges of traditional computing systems, push main memory to quickly become a significant bottleneck across a wide variety of applications, such as those in Machine Learning (ML) and genomics. This happens because such applications require continuous data movement between the compute and memory units in such systems, making memory the system bottleneck. This dilemma, also known as the **data movement bottleneck**, only worsens as this performance and energy consumption gap between two units grows and the sizes of applications' data working set increase. For example, previous works in 2013, 2014, and 2018 [40–42] report that data movement accounts for 35%, 40%, and 62% of the total system energy in various such workloads, respectively.

To alleviate data movement bottleneck, recent works prompt re-examining an old idea, interchangeably called Processing-In-Memory (PIM) or Computation-In-Memory (CIM). The key idea behind CIM is to place some form of computing capability near or inside memory where the data is stored. This can be inside the memory chips, in the

memory controllers, within the caches, or in the logic layer of 3D-stacked memories. CIM is a computational paradigm, and architectures that utilize CIM (i.e., CIM architectures) can potentially resolve some of the problems discussed in the von-Neumann architecture by reducing or completely eliminating the data movement between where the computation is performed and where the data is stored.

Note that the idea behind CIM architectures has been around in the field for more than five decades [43–61]. Historically, prior endeavors were not widely adopted due to three main reasons. (1) Challenges associated with combining processing units with conventional main memories (i.e., DRAM), (2) the memory scaling issues prevalent in contemporary technology and applications, and (3) the less significant impact of data movement of traditional applications constraints on system expense, energy, and performance.

However, as we discussed before, the new requirements of modern applications and advancements in contemporary memory architectures, such as the 3D-stacked combination of logic and memory or emerging memory technologies such as memristors, an array of CIM architectures have been investigated in recent studies for various applications [42, 62–99].

Moreover, looking at it from a technology viewpoint, emerging (memory) technologies, such as memristive devices and circuits based on integrated photonics, hold the promise to supplant conventional memory systems, thereby dealing with or mitigating the problems highlighted earlier. The primary technologies in the memristive sphere are phase change memory (PCM), spin-transfer torque magnetic random-access memory (STT-MRAM), and resistive random-access memory (ReRAM). Unlike traditional memories, which use the presence or absence of charge to represent data, these devices denote data as varying resistance levels, making them non-volatile compared to SRAM-based cache and DRAM-based primary memory. This property lowers the memory's static power consumption, thereby addressing the leakage wall. Regarding physical size, memristors are compact compared to SRAM but align with the size of DRAM or flash memories. Yet, some types of memristive technology permit the storage of multiple bits in a single device, which effectively increases the density. The read and write latency of these devices aligns with that of DRAM and is notably faster than flash memory. Note that even though these devices lag behind SRAM and DRAM regarding endurance and programming energy, they still outperform Flash memories [100–102]. Lastly, as the fabrication processes are still under development, these devices are subject to various imperfections, which may potentially lower their reliability. Numerous research efforts are currently underway to rectify the inconsistencies and non-idealities found in these devices [103, 104].

Therefore, the importance of revisiting CIM in the current times cannot be overstated, using a rejuvenated viewpoint that embraces innovative methodologies and concepts. This should involve the utilization of the latest memory technologies, consideration of practical systems and applications, and an attitude oriented towards simplifying its assimilation and practicality.

1.3. PROBLEM DISCUSSION

No matter the strategy implemented for a CIM design for better performance and/or energy efficiency of a modern application, significant practical hurdles exist that both system designers and developers must confront to facilitate its universal acceptance within the computational field and across varied workload domains. Below, we list some open questions and research opportunities at various level of the problem stack that needs to be resolved before CIM adoption for modern applications becomes a reality.

- **Circuit Level.** CIM fundamentally relies on the connectivity between memory devices. To date, previous works [87, 105–107] propose numerous array structures and circuit designs that enable distinct basic operations or facilitate the execution of the same operations in varying manners. Basic operations, such as logical ones, serve as the foundational elements for more complex operations, and their efficiency dramatically impacts the entire system. In addition to the array structure and operations, the inclusion of various analog and digital peripheral circuits is necessary to utilize computation within a memory array fully. However, the design of circuits for CIM is still in its infancy, necessitating further research to discover the most effective design solutions.

- **Architecture Level.** A CIM architecture also requires consideration of numerous architectural factors due to their potentially substantial impact on the system's performance, energy consumption, size, and even accuracy (at the application level). Although some previous works [69, 108–110] touch base on these matters, more research is required to find the most optimal solutions for managing memory arrays, a potentially specific instruction set dedicated to controlling CIM operations, and tailored communication networks for the system's needs to manage data across multiple memory arrays that might be involved in one application.

- **Compiler Level.** While CIM circuits and architectures can facilitate certain operations within memory arrays, there is a need for a compiler that can dissect an application into sections that can be performed within the memory, breaking it down into operations and instructions the memory can understand. In addition, the manner in which data is arranged in the memory, considering the system's requirements and constraints, can have a significant effect on the system's metrics. Since CIM itself is still in its early developmental stages, optimized compilers for CIM systems have only received minimal attention up to this point [111–113].

- **Algorithm and Application Level.** A CIM design only suits specific algorithms and applications that are data-intensive, and their operations are supported by the underlying architectures and circuits. Some previous works [108, 114, 115] already target this research direction. However, extensive research is needed to profile the applications, find the suitable ones for a CIM design, and then make the necessary adjustments (e.g., changes to the algorithmic, providing support for the necessary data flow at the architecture and circuit levels, or adjusting the application to tolerate the potential accuracy or performance losses due to device non-idealities) to enable the execution.

A true CIM design can execute the right application or workload and achieves improvements in the application metrics (such as execution time, energy efficiency, etc.) using a tailored (general or specific) algorithm, compiler, architecture, data flow, circuit, and device.

1.4. SOLUTION DIRECTION AND RESEARCH TOPICS

In this dissertation, we seek to investigate (1) acceleration and efficient execution of two application domains, namely genomics and ML, and (2) a few emerging memory technologies and circuit and architectural techniques that might benefit us in a CIM design, having our genomics and ML applications in mind.

We divide the research to be done to achieve these investigations into two steps:

- Identifying and improving the bottlenecked performance of potential kernels in genomics and ML.
- Exploiting emerging technologies to design a state-of-the-art CIM architecture that benefits the target kernels with an end-to-end improvement goal in mind.

Through this integrated approach, we aim to drive transformative advancements in these fields.

1.4.1. IDENTIFY AND IMPROVE BIOINFORMATICS AND NEURAL NETWORK KERNELS USING CIM

We target two application domains: genomics and ML. The applications in these domains have different functions or kernels in that a designer must be well-versed. Different functions may involve different data and control flows and may necessitate diverse features to be activated in a von-Neumann architecture or a CIM one. To enhance the efficiency of each function, we first profile them separately and then investigate the optimal ways to map them onto the memory. We also evaluate the necessary components needed alongside the memory array. It is important for a designer to be well-versed in the applications that can be run on a CIM system using emerging memory technologies. Gaining a comprehensive understanding of CIM's principal attributes and a clear perspective regarding CIM's entire breadth is crucial before embarking on its adoption. We aim to collect pivotal data and distill them to their essence. This process could illuminate existing design alternatives and the areas that require additional focus. This valuable insight may assist in pinpointing potential future trajectories, such as supported kernels and their benefits and overheads.

1.4.2. EXPLORING EMERGING (MEMORY) TECHNOLOGIES FOR CIM

Exploring emerging (memory) technologies, such as memristive devices (e.g., PCM and STT-MRAM) that offer several properties such as non-volatility, compactness, and inherent capability for performing logical operations (e.g., logical AND), or integrated photonics that offer high frequency and low latency operations is essential for enhancing the CIM paradigm one step further. We aim to consider multiple of these technologies and use their unique features to build a CIM design that can beat the SotA alternative in one or several comparative metrics such as execution time and/or energy consumption.

1.5. THESIS STATEMENT

Our approach is encompassed by the following thesis statement:

> *CIM can improve the performance of multiple Genomics pipelines and Machine Learning applications.*

To do this, our methods require solving the aforementioned issues with traditional systems and recent CIM works. Therefore, this dissertation addresses the following open questions.

- What kernels or pipeline stages of genomics studies or ML applications can benefit from CIM?
- What is the best memory technology used for such CIM designs?
- What architectures are of potential directions to consider?
- Can we efficiently deploy CIM for our applications using the best memory technologies?
- What does the term "efficiently" means for each kernel and application domain?

1.6. CONTRIBUTIONS

This dissertation makes the following contributions, addressing the discussed research topics and embracing the thesis statement:

- ***Evaluation of CIM potential for accelerating basecalling.*** We identify an acceleration opportunity for basecalling, the very first computational step in any genome analysis study. We propose a novel hardware/software co-design framework that can effectively accelerate the state-of-the-art Deep Neural Network-based basecalling step on a widely accepted CIM architecture using emerging memristor devices. This framework enables us to, for the first time, to the best of our knowledge, account for the effects of existing non-idealities in the underlying memristors on the end-to-end accuracy of an application that requires significant acceleration but cannot tolerate accuracy loss. We publish this work in [116].
- ***A pre-alignment filtering algorithm for short read filtering using CIM.*** We first identify that SotA pre-alignment filters for short reads are the (new) performance bottleneck to focus on in a genome analysis pipeline that requires sequence alignment of short reads. We then establish that even pre-alignment accelerators on graphics processing units (GPUs) or field-programmable gate arrays (FPGAs) still do not resolve this performance bottleneck and are themselves limited by the rate at which the data is being fed to them. We propose a hardware/software co-designed (HW/SW co-designed) accelerator based on CIM capable of pre-alignment filtering for short-sequence alignment. We first propose a lightweight and hardware-friendly filtering algorithm. We then exploit emerging non-volatile memories as underlying devices for hardware acceleration. We publish this work under [117].

- ***A CIM architecture for short-read pre-alignment filtering.*** We profile all the previous pre-alignment filters for short-sequence alignment and discover that they share most of their kernels. We propose a CIM architecture capable of handling these shared kernels. We show that our architecture can accelerate the overall pre-alignment filtering of short reads by removing the overhead of data movement for all the essential kernels. We extensively evaluate the architecture for previous pre-alignment filters. Our design can also support any future filter as long as they also require similar kernels. We show this by taking our previous filtering algorithm discussed previously, although it was not involved in our profiling study. We publish this work in [118].

- ***An algorithm and CIM architecture for pre-alignment filtering of long reads.*** We acknowledge the industry's move towards sequencing long reads. We devise an enhanced data handling and architecture to allow pre-alignment filtering for long reads. Our architecture is based on CIM to take into account that with moving from long reads to short reads, the data movement increases, which can negatively affect the performance. We publish this work in [119].

- ***A high-performance and energy-efficient food profiler using CIM.*** We pinpoint two critical sources of inefficiency in SotA profilers currently used for food monitoring: (1) requirements for high-end servers with large storage and memory and (2) random accesses to large working datasets, incurring unnecessary data movement. We propose an end-to-end, hardware/software co-designed food profiling framework that efficiently profiles species of a food sample. We reduce the food profiling problem to a multi-object (multi-species) classification problem using hyperdimensional (HD) computing (HDC) followed by an abundance estimation step. We propose a CIM accelerator to mitigate the costs of data movement and shift operations in our HDC solution and simultaneously solve the second problem of profilers as well. This work has been published in [120].

- ***A CIM method for k-mer matching and framework for taxonomic profiling.*** We build the first hardware/software co-designed framework for taxonomic profiling that exploits *real* memristor (i.e., STT-MRAM) devices and the CIM paradigm. We propose an optimized framework for accelerating Kraken2[1] that notably improves execution time and energy consumption of taxonomic profiling with a negligible area overhead. We achieve this by proposing a memristor-based substrate, called *TL-PIM*, that accelerates the bottleneck of Kraken2 (and many other profilers), the *Table Lookup* operation. We publish this work in [121].

- ***A high-performance data mapping for Binary Neural Network using CIM.*** We advance SotA CIM accelerators for BNNs by providing a highly parallel data mapping that is compatible with any CIM design capable of performing VMM, e.g., memristor-based crossbars such as ePCM-based or ReRAM-based ones. Our proposed data mapping is designed with the conventional 1T1R memory crossbar

[1] Kraken2 is currently the most widely-used and one of the most promising taxonomic profilers based on recent metagenomics challenges.

structure in mind and is, therefore, compatible with many of the already evolving crossbar architectures. We publish this in [122].

- ***A CIM accelerator based on optical phase change material for Binary Neural Network.*** We propose an oPCM hardware-based CIM implementation incorporating our discussed optimized data mapping for BNNs. Our accelerator ensures maximum parallelism through exploring the potential provided by the features of CIM architecture and the inherent properties of oPCM (via wavelength division multiplexing (WDM). Our design realizes an order of magnitude improvement in latency/throughput without losing the accuracy of the network. We publish this in [123].

1.7. THESIS OUTLINE

- This dissertation is organized into fourteen chapters.

Chapter 2 introduces CIM and its classifications, emerging memory technologies, genomics pipelines, and Neural Network. We first discuss the history of CIM and its types. We then present a background on two emerging technologies: resistive memory devices and integrated photonics. We then introduce two application domains that we believe can benefit from these computing and technological paradigms.

Chapter 3 explores CIM for the acceleration of basecalling, an important step in genomics pipelines. We first provide a background on what basecalling is and what the issues are with SotA basecallers. We then propose a hardware/software co-designed framework to analyze (1) the potential of CIM for basecalling acceleration and (2) strategies to mitigate the possible negative effects of non-idealities. We evaluate the SotA basecaller on a SotA CIM architecture and provide key suggestions and recommendations for system designers of future emerging accelerators for basecalling.

Chapter 4 targets CIM acceleration for another step in the genomics pipeline, pre-alignment filtering. We first introduce pre-alignment filtering, where it exists, and how it helps the general pipeline. We target pre-alignment of short genomic sequences (called short reads), as currently, the majority of our genomic data are in the form of short reads. We then present a lightweight algorithm that is compatible with CIM architecture.

Chapter 5 also targets pre-alignment filtering step. However, in this chapter, we first profile the existing pre-alignment filters for short reads. We identify the existing kernels and the ones they share. We also identify data movement as the main bottleneck of these short-read filters. We finally propose a CIM architecture that not only supports all the shared kernels of previous filters but also eliminates the data movement overhead. Our architecture is capable of accelerating the pre-alignment filtering for short reads.

Chapter 6 considers the long-reads for the first time. Knowing that the industry is moving towards obtaining long reads, this chapter presents a hardware accelerator for pre-alignment filtering of long reads. Note that due to their size, the architecture and algorithms discussed in Chapter 4 and Chapter 5 do not directly work for long reads. We identify data movement as a potential contributor to the execution time of the filtering. We first present a CIM-friendly algorithm to enable long-read filtering. We finally discussed the corresponding CIM architecture.

Chapter 7 explores CIM for food profiling, one of the possible final applications in a

genomic pipeline. We first introduce food profiling. We then discuss its importance and the limitations of current profilers in the industry. We also present some background on hyperdimensional computing (HDC), a classification method that we found suitable for our problem. We then propose a platform-independent framework for food profiling using hyperdimensional computing. This way, we overcome the first problem of current profilers, which is their massive data structure. We then present a CIM accelerator using memristors to enable food profiling faster than existing ones.

Chapter 8 introduces a CIM architecture for taxonomic profiling. We first introduce metagenomics studies and the role taxonomic profilers play in it. We then discuss the limitations of existing profilers by profiling the SotA taxonomic profiler. We identify data movement and one single kernel as the key limiters. We propose a CIM design to accelerate the main kernel that constitutes most of the execution time. We then present the first HW/SW co-designed framework to accelerate taxonomic profiling using a CIM-based system.

Chapter 9 studies the acceleration of BNNs with CIM architectures. We first present a background on BNNs. Then, we introduce an efficient data mapping that provides high parallelization for the required operation in a BNN on any CIM design capable of performing VMM. Our design uses a conventional 1T1R memory crossbar structure and requires only the VMM compatibility, which is common in many of the memristor-based crossbars such as ReRAM-based or ePCM-based ones.

Chapter 10 remains on the topic of BNNs while advancing the CIM accelerator with other emerging technologies. We first introduce integrated photonics with PCM, which is commonly known as optical phase change memory (oPCM). We then present an oPCM hardware-based CIM implementation incorporating the proposed highly parallel data mapping with only VMM compatibility as its requirement. We exploit oPCM to ensure maximum parallelism through exploring the potential provided by the features of CIM architecture and the inherent properties of oPCM (via wavelength division multiplexing (WDM).

Chapter 11 summarizes this dissertation and presents future research directions.

2

Background and State-of-the-Art

This chapter presents the necessary background for our thesis in three main groups. First, we introduce the Computation-In-Memory (CIM) paradigm, a classification of CIM designs, and the architecture of a CIM tile. Subsequently, we discuss some potential emerging memory technologies for CIM designs. Here, we present a theory of memristor devices and a detailed comparison of three examples of emerging memories with traditional ones. We then present the main crossbar structures for memristor-based CIM designs. We also touch base on the fundamentals of using integrated photonics with memristors in using optical phase change memory-based systems. After that, we discuss a few primitive functions supported by emerging memory technologies and CIM architectures. Second, we discuss two main groups of modern applications, namely genomics and Machine Learning. We discuss different computational pipelines in genomics and the various steps and kernels that exist in them. We also provide more motivational data for the data explosion in these applications and some essential background for Binary Neural Networks. Third, we discuss the state-of-the-art CIM designs and simulations, from general-purpose proposals to application-specific ones.

2.1. COMPUTATION-IN-MEMORY (CIM)

Presently, the prevalent von-Neumann architectural paradigm in our computational infrastructure contributes to performance loss and higher energy consumption because of the excessive data transfers it necessitates. This design principle, as depicted in Fig. 1.3, partitions the processing and memory/storage components, with the two being interconnected through energy-intensive links. As an upshot of this setup, data is perpetually relayed back and forth between these units to conduct computations.

The process of manipulating data that resides in memory requires significant time and energy. The Central Processing Unit (CPU) or an associated accelerator initiates the sequence by dispatching a request to the memory controller. This controller subsequently interacts with the memory module. Following this, data is retrieved from the memory, routed back to the memory controller, and deposited in the CPU's cache and registers, and only then can the CPU perform computations on it.

Currently, only the CPU or respective accelerators possess computational capabilities, leaving all other components relegated to roles of data storage and transfer, with no computational functionality. This disparity results in a loss of performance and energy efficiency. A recent study [42] discloses that this processor-memory split is responsible for over 62% of the total system energy consumption across four key mobile consumer workloads, thereby underscoring the urgency for a more holistic computational strategy that reduces energy squandering.

At least five factors play a part in the performance and energy complications linked to data transfers between the processor and memory, further accentuating the adverse effects of data movement within our computational systems.

1. The narrow width of the off-chip bus between the memory controller and main memory. This limitation reduces bandwidth and increases latency, hindering the concurrent execution of memory requests.

2. Complex mechanisms like multi-level cache hierarchies and latency tolerance/hiding mechanisms that we developed to tolerate main memory data access latency. Though efficacious for traditional applications, these approaches bear high costs in terms of chip area, energy, and latency, further complicating the architectural layout of the system.

3. Questionable caches regarding their efficiency. Many instances reveal unused data in caches, resulting in wasted hardware area and memory bandwidth. Modern workloads with their diverse access patterns often render these caches inefficient or even redundant, thereby exacerbating the energy wastage in systems that are centered around the processor.

4. Random memory access patterns in modern applications, such as graph processing and sparse linear algebra, that struggle with inefficiency in caches, a main memory bus, and the main memory itself. The stochastic nature of these random accesses also undermines the effectiveness of prefetching mechanisms.

5. The interconnects between the processor and main memory are long and energy-consuming, imposing additional latency to every data access and consuming sig-

nificant energy in moving data to/from the DRAM memory. Moreover, the latency and energy consumption of these interconnects do not scale favorably with advancements in technology node generations, which only serves to compound the cost associated with data movement.

The issues identified largely stem from the prevailing processor-centric design approach. To mitigate this ongoing cycle of performance deterioration and energy inefficiency, it is imperative to transition towards a more data-centric design principle.

The rising gap between processing and memory/communication technologies has resulted in communication costs outstripping computation costs in terms of energy consumption. The energy required for main memory access is roughly 115 times that of an arithmetic operation [124]. Consequently, data movement is responsible for 40%, 35%, and 62% of the total system energy in scientific, mobile, and consumer applications, respectively. This excessive energy consumption significantly hampers both efficiency and performance across various computing platforms [124, 125].

The root causes of the low performance, energy inefficiency, and heightened system complexity can be traced back to the processor-centric or von-Neumann design model. To counter these challenges, a fundamental change in design philosophy is needed, transitioning from a processor-centric to a data-centric approach. This includes 1) reducing data movement during computational tasks and 2) enabling computation to occur where the data resides instead of being confined to the processor. To enact this shift, it is crucial to dismantle the traditional divide between computing and memory units, leading to the emergence of a novel model known as Processing-In-Memory (PIM) or Computation-In-Memory (CIM). From now on, we only use the term CIM.

2.1.1. CIM DESIGNS CLASSIFICATION BASED ON COMPUTATION LOCATION

Fig. 2.1 demonstrates the four potential locations where a computational result can be generated, effectively creating four classes for computing. If the result is produced outside of the memory core, we call the class Computation-Outside-Memory (COM). In such scenarios, the computation can either occur within additional logic circuits inside the memory System-in-Packages (SiP) or, akin to traditional von-Neumaan architecture, within computational cores (CPU or GPU). These instances are categorized as COM-Near (COM-N) or COM-Far (COM-F), respectively. An instance of COM-N is represented by 3D-Stacked Memory, where extra logic is integrated into the memory system to facilitate computation. The other two classes are (1) CIM-Array (CIM-A), in which the computation result is generated within the memory array, and (2) CIM-Periphery (CIM-P), in which the result of the computation is generated within the memory periphery.

CIM-A designs typically offer a few key advantages:

- They enable maximum bandwidth for data "transfer" between the computation and storage units.
- They enable high levels of parallelism.
- They can facilitate the logic cascading of universal functions.

However, these designs also exhibit some common limitations, such as:

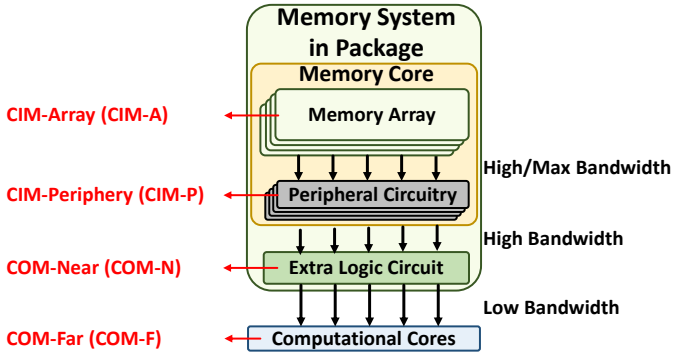

Figure 2.1: Four locations for computation [126].

- Frequent write operations can cause endurance and energy consumption issues.
- The memory array and its controller necessitate considerable design efforts.
- Performance can be hampered due to high latency associated with device programming and logic cascading needed for complex functions.

Similarly, CIM-P designs typically come with advantages of their own:

- They do not compromise the endurance of the memory array or the state of the stored data.
- They require less redesign effort for the memory array.
- They allow high to maximum bandwidth, depending on the complexity and area of the periphery.

However, they also present some common drawbacks:

- Performance may be hampered if due to shared peripheral resources such as sense amplifiers (SAs).
- Cascading functions cannot be achieved without read/write operations.

Existing literature based on COM-N, CIM-P, and CIM-A are all known as works in the CIM (PIM) paradigm. However, when we speak of CIM in this thesis, we mainly target those that fall into the category of CIM-A and CIM-P. Therefore, to understand the potentials of these two designs more deeply, in the following we discuss the memory crossbar and its periphery.

2.1.2. ILLUSTRATION OF GENERIC CIM TILE

Here, our objective is to provide a clear delineation of the data movement in an abstract CIM tile and its capability to generate intricate functions.

Fig. 2.2 presents a generalized overview of a CIM tile, also known as CIM array. A CIM tile is composed of a memory crossbar, along with its digital and analog peripherals.

CIM tile is designed to accept digital data and instructions [127], alternatively referred to as control signals. Internally, a controller directs all circuit operations in line with these instructions. Data traverses four stages: 1) Input processing f_{in}, 2) Crossbar array f_{Xbar}, 3) Sensing f_{sens}, and 4) Output processing f_{out}. To this date, innovative circuit designs have been proposed to broaden the array of functions feasible at each stage. When implementing an application using CIM tiles, the designer handpicks a function and the corresponding circuit for each stage. Consequently, we end up with an accelerator tailored to a particular application. So far, research on memristor-based CIM has primarily concentrated on crafting accelerators that cater to specific applications. Yet, to attain a design as universal as possible, there is a need to support an extensive range of functionalities at each stage concurrently, as opposed to distinct circuits for individual functions. The forthcoming discussion will elaborate on the basic functionalities currently available at these four stages. Recognizing these functions empowers us to construct more elaborate functionalities within the CIM-tile.

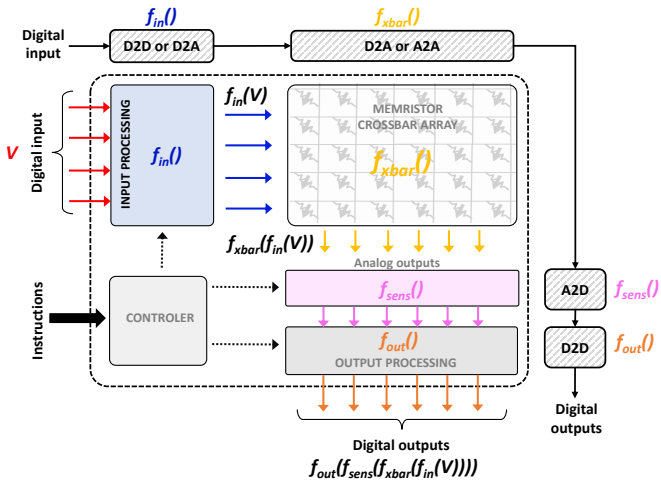

Figure 2.2: Generalized CIM tile/array [128] for computation in four steps: (1) input processing, (2) crossbar array processing, (3) sensing, and (4) output processing

2.1.3. POTENTIAL EMERGING TECHNOLOGIES FOR CIM

MEMRISTORS

Prior to the 1970s, the core components of circuitry were resistance (R), capacitance (C), and inductance (L), linking the key units of electricity: charge, current, voltage, and flux. Resistance correlates current with the rate of change in voltage, expressed mathemati-

cally as $dv = Rdi$. Similarly, capacitance represents the connection between charge and voltage, depicted as $dq = cdv$. The third element, inductance, is a bridge between flux and current, illustrated by $d\Phi = Ldi$. Fig. 2.3 demonstrates these elements and their relations to each other.

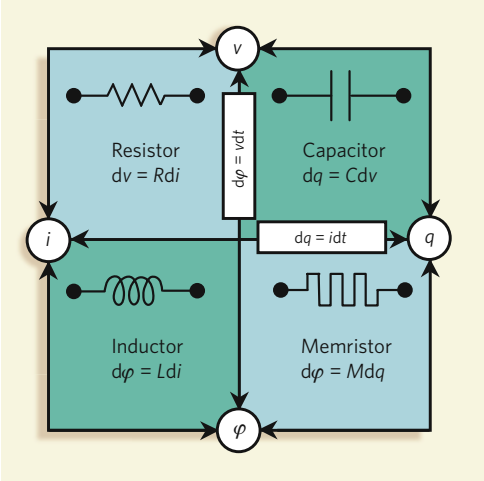

Figure 2.3: Four fundamental circuit elements and their relation [129].

In the pivotal year of 1971, Leon Chua proposed a new fundamental electrical component, memristors. This unique element established a link between magnetic flux and charge. Mathematically, this relationship is represented in Equation 2.1. Unlike the resistor (R), the memristor presents a dynamic interplay between current and voltage. This means that its current behavior depends not merely on voltage and current but also on the memristor's specific state.

$$M(q) = d\Phi/dq => M(q(t)) = (d\Phi/dt)/(dq/dt) = V(t)/I(t) \tag{2.1}$$

Equation 2.2 presents the memristor's behavior as state-dependent Ohm's law, where the resistance is an interplay of the state variable x, voltage, and current. Here, R is the memristor's resistance function.

$$M(q) = d\Phi/dq => M(q(t)) = (d\Phi/dt)/(dq/dt) = V(t)/I(t) \tag{2.2}$$

Equation 2.3 presents the state of the memristor, where the rate of change of the state variable over time depends on the state variable itself, voltage, and current.

$$dx/dt = f(x, V, I) \tag{2.3}$$

The first tangible manifestation of a memristor was reported by HP labs in 2008 [130]. The memristive device, a two-terminal component, demonstrated a characteristic 'pinched' hysteresis loop at the origin of the current-voltage (I-V) graph. This pinched hysteresis loop indicates a resistance shift contingent on the history of resistance and

the applied voltage or current. Essentially, it signifies that variations in voltage (or current) can instigate changes in the state of resistance.

Fig. 2.4 presents an overview of switching mechanisms of bipolar and unipolar memristors. Bipolar memristors switch between low and high resistance states by applying a positive and negative voltage, respectively. Contrarily, unipolar memristors operate independently of the polarity of the programming voltage and respond solely to its magnitude.

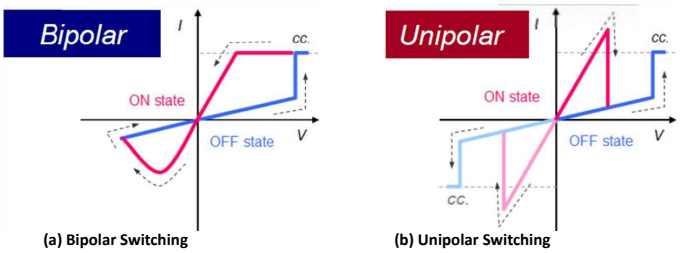

Figure 2.4: Memristor switching dynamics for (a) bipolar and (b) unipolar devices [131, 132].

Various technologies and materials are available for the construction of memristor devices. The three leading memristors are:

- **Resistive Random-Access Memory (ReRAM).** Resistive random-access memory (ReRAM or RRAM) devices are structured as a metal-insulator-metal stack. In a bipolar ReRAM, the act of setting and resetting is governed by the shift in the polarity of the programming voltage (for instance, 2V), which either forms or dissolves the conductive filament. For non-invasive reading of the device, a lower voltage (such as 0.2V) is used, and the current (or voltage) running through (or across) the device is monitored. Programming the device demands a higher voltage/current and extended latency [113]. These devices can therefore be in two states: high resistance state (HRS) and low resistance state (LRS).

- **Phase Change Memory (PCM).** Phase change memory (PCM) devices date back to the 1960s. A PCM device operates based on the attribute of certain materials, for instance, Ge2Sb2Te5. These materials are capable of undergoing a rapid and reversible shift from a highly resistive amorphous phase to a conductive crystalline phase, thanks to Joule heating. A typical PCM device features a mushroom-like shape, wherein the lower electrode confines heat and current. This confinement leads to a near-hemispherical form of the amorphous region when in a high resistance state (HRS). The transition to a low resistance state (LRS) is achieved by crystallizing the amorphous region [133].

- **Spin-Transfer Torque Magnetic Random-Access Memory (STT-MRAM).** The data storage element in spin-transfer torque magnetic random-access memory (STT-MRAM) is the magnetic tunnel junction (MTJ), which encodes one bit of data into

two bi-stable magnetic states. The MTJ fundamentally comprises two ferromagnetic layers separated by an ultra-thin dielectric tunnel barrier (TB). The top ferromagnetic layer, referred to as the free layer (FL), allows for the magnetization to be altered by the application of a spin-polarized current. Conversely, the magnetization of the bottom ferromagnetic layer, or the pinned layer (PL), is rigidly pinned in a specific direction. Consequently, the magnetization of the FL can be either parallel (P state) or anti-parallel (AP state) to the PL [134], capturing the high resistance state (HRS) and low resistance state (LRS) states.

Fig. 2.5 demonstrates a pictural presentation of LRS and HRS in ReRAM, PCM, and STT-MRAM.

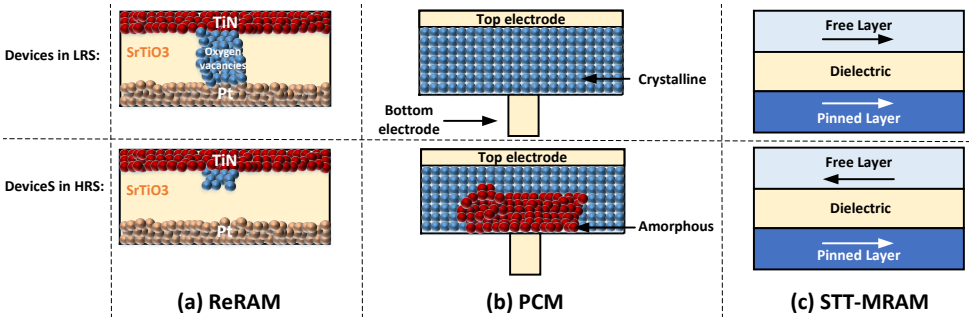

Figure 2.5: LRS and HRS of (a) ReRAM, (b) PCM, and (c) STT-MRAM.

Table 2.1 presents a comparison of memristors as emerging memory technologies with conventional memories. Access time indicates the speed at which data can be stored and retrieved (read and write latency). Cycling endurance denotes the count of times a memory device can undergo programming (or switching). The access energy quantifies the energy expended to read from or write into the device. The notation F signifies the smallest feature size.

	Conventional Memories			**Emerging Memories**		
	SRAM	**DRAM**	**Flash**	**ReRAM**	**PCM**	**STT-MRAM**
Size (F^2)	120-150	10-30	10-30	10-30	10-30	10-30
Volatality	Yes	Yes	No	No	No	No
Read latency	~1 ns	~3 ns	~100 ns	~10 ns	~10 ns	~5 ns
Write latency	~1 ns	~10 ns	~1 ms	~10 ns	~10 ns	~5 ns
Write energy	~fJ	~10 fJ	~100 pJ	~1 pJ	~10 pJ	~1 pJ
Endurance	$\sim 10^{16}$	$\sim 10^{16}$	$\sim 10^{4} - 10^{6}$	$\sim 10^{7}$	$\sim 10^{12}$	$\sim 10^{15}$
Scalability	Medium	Medium	Medium	High	High	High

Table 2.1: Comparison of conventional and emerging memories [102, 135, 136].

From Table 2.1, we made three key observations. First, emerging memory technologies offer high-density storage. Assuming these technologies can support multi-bit storage, their density could surpass that of conventional memory technologies. Second,

unlike SRAM and DRAM, emerging memory technologies are non-volatile, leading to lower static energy consumption. They operate at lower voltage levels than flash memory, making them suitable for embedded applications. While their read and write times are inferior to SRAM, they are comparable to DRAM and significantly better than Flash. Third, The endurance of emerging memories trails slightly behind volatile memories but is comparable to, and even exceeds, Flash memory.

Note that, unfortunately, emerging memories often struggle with high programming energy. This factor is the primary reason why researchers frequently use these devices for applications where frequent reprogramming is not required. Moreover, memristors typically show non-ideal behaviors that pose substantial challenges in the realm of dependable computing. These non-ideal behaviors primarily originate from fabrication imperfections in the device and inaccuracies in programming. These non-idealities can be briefly summarized below:

- **Non-zero G_{min} error.** In memristors, varying resistance levels are used to encode information. For instance, the logical value '0' could be assigned to either a high or low resistance level. Regardless of the resistance level, there would be a current that passes to the sensing circuit. This means that even when a memristor represents a digital zero with a high resistance level, a non-zero output current is generated when a non-zero input voltage is applied. This characteristic introduces complexities when executing mathematical computations with memristor devices [137].

- **Conductance drift.** The conductance values of memristor devices are subject to change over time. This alteration can be triggered even by the recurring reading of the device. As a result, to restore its initial state, the device necessitates reprogramming after a certain duration. Read disturb is a notable form of conductance drift in which a read operation can cause the real value stored in the device to change inadvertently.

- **Programming Noise.** Theoretically, memristors can flip between specific conductance levels, each characterized by distinct values. However, actual scenarios exhibit each level presenting a spectrum of conductance values, typically following a Gaussian distribution. The correlation between the variability in programming voltage (or current) and the programmed value is significant. Memristors possessing high conductance levels face diminished noise margins between the levels. Consequently, the damaging effects of programming noise become more severe. This implies that employing a high-resolution memristor cell poses a considerable challenge to a system regarding its precision and functionality, making it a significant concern.

- **Endurance and Retention.** Memristor devices face endurance challenges, also known as the aging problem, arising from the damaging effects of the programming process. For instance, in ReRAM devices, this aging could be attributed to alterations in oxygen vacancies induced by oxygen diffusion. Such endurance-related concerns limit the frequency at which these devices can be reprogrammed,

making memristors more suitable for applications not necessitating repeated reprogramming. At present, memristor devices demonstrate endurance exceeding 10^6. Furthermore, the data retention span for memristor devices can extend to approximately a decade [138].

- **IR-drop.** IR-drop arises due to the resistance encountered by electrical wires. The farther the memristor is from its driver, the greater the impact of this wire resistance. Consequently, there is a reduction in the voltage necessary for programming and reading operations, diverging from its original value. This deviation can lead to a memristor being programmed with incorrect values (a form of programming noise) or inaccurate values being extracted from the device during a reading operation.

When memristor devices are placed within a crossbar configuration, they can serve as storage units and provide additional in-memory functionalities and enable CIM. Fig. 2.6 captures a few widely-used and promising crossbar structures proposed by previous works that we briefly discuss below:

- **1R crossbar array.** In a 1R crossbar array, each cell comprises a single memristive device (1R). The individual cells are accessed via wordline (WL) and bitline (BL). This specific array arrangement delivers high density relative to alternative structures. To read a device, we initiate the designated WL with 'V_{read}' while the remaining WLs are grounded. Consequently, the values of all memristors in the activated row are read by monitoring the current flow in the bitlines. Despite its benefits, this design is hampered by the issue of sneak path current. As also demonstrated in Fig. 2.6-(a), the current in the bitlines could potentially leak through other memristors in the non-active rows. As a result, the current sensed by a sense amplifier may not accurately represent the values stored in the memristors. The programming process of a 1R crossbar array has two steps: (1) initialization and (2) writing. During initialization, the devices in a chosen row are reset to HRS, corresponding to logic '0', by applying VDD to the WL and grounding all the BLs. In the writing phase, the ground is applied to the selected WL, and VDD is applied only to those BLs intending to program their memristor to LRS, representing logic '1'. For the non-selected WLs, a $1/2(VDD)$ voltage is applied to prevent the non-selected devices from switching. In this process, the voltage across the selected device in the first column is VDD (i.e., $VDD > V_{set}$), while the voltage across the deselected device in the second column is $VDD - VDD/2$ (i.e., $VDD - VDD/2 < V_{set}$).

- **1T1R common source-line crossbar array.** To mitigate the issue of sneak path current in the 1R crossbar array, the 1t1R structure pairs an access transistor with each memristor device as shown in Fig. 2.6-(b). In this structure, the memristors in a row are connected by a common source-line (SL). To read a memristor's value, we apply VDD to its associated wordline to activate the access transistor. Furthermore, V_{read} is applied to the source-line, and the current (or voltage) is detected on the bitline. To simultaneously program all the memristors located in a single row, we require two distinct voltage levels; VDD and $2VDD$. The source-line is driven by VDD, and depending on whether we aim to set the device to LRS (logic '1') or

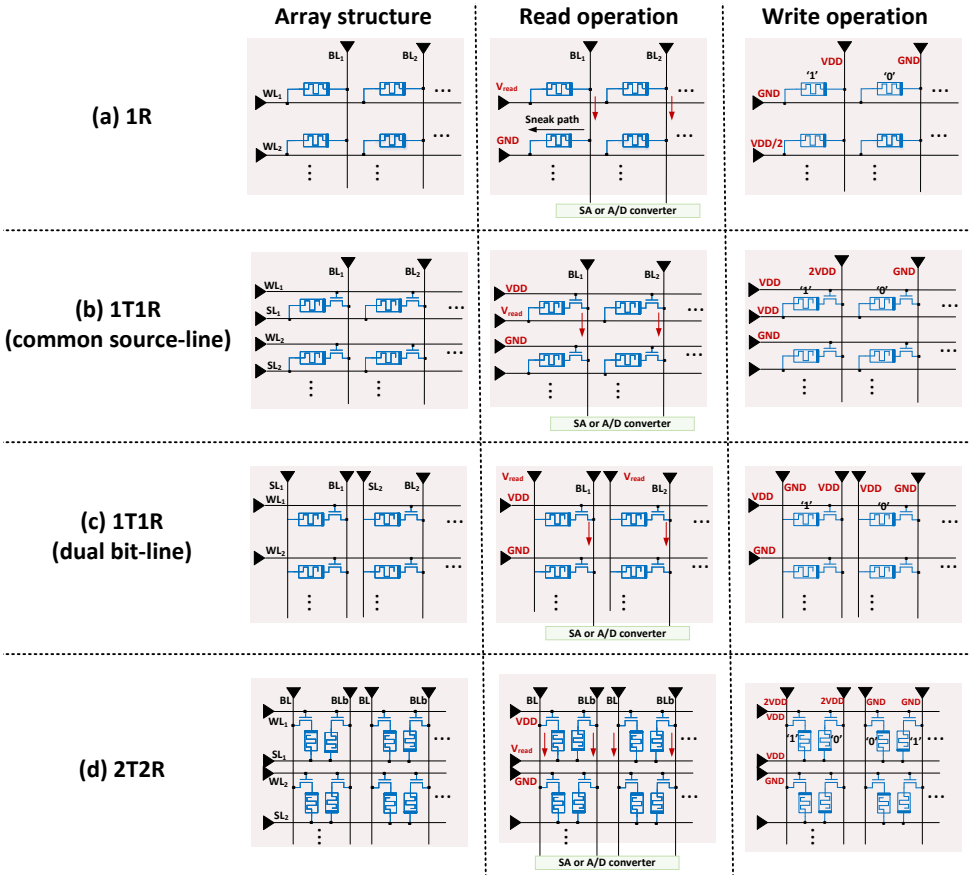

Figure 2.6: Prevalent crossbar structures and their functioning during read and write processes: (a) 1R passive array, (b) 1T1R common source-line array, (c) 1T1R dual bitline array, and (d) 2T2R array structure.

HRS (logic '0'), $2VDD$ or GND is imposed on the bitlines, respectively. This structure is particularly suited for Vector-Matrix-Multiplication (VMM) operations.

- **1T1R dual bitline crossbar array.** In a 1T1R dual bitline crossbar array (Fig. 2.6-(c)), an access transistor is still assigned to each memristor device. However, the memristors collectively use the source-line within a single column of the crossbar. Dual bitline arrays exhibit lower latency and energy consumption due to the reduced voltages during the write operation. This allows for a thinner oxide access transistor and lowers the transistor's effective resistance. However, this array structure requires more space relative to the common source-line array due to the parallel orientation of the bitlines (BLs) and source-lines (SLs). This structure is well suited for 'streaming' logical operations.
- **2T2R crossbar array.** In a 2T2R crossbar structure, each cell is composed of two memristors (2R) and two access transistors (2T), and each cell stores the actual data as well as its complementary value. Fig. 2.6-(d) depicts this structure. This structure needs a differential sensing mechanism similar to the one used in traditional SRAM. The data and its complementary value traverse the bitline and bitline bars, with a minor difference between the voltage or current of these two lines detected by a differential sense amplifier. This results in improved performance, higher sensing margin, and greater resilience to variation and non-idealities. However, as alluded to, these advantages come at the expense of reduced area efficiency. Furthermore, the efficacy of this structure for computational tasks, such as VMM and XOR operations.

As mentioned above, using memristors in discussed array structures for CIM requires considerations regarding the accuracy for memory or computational operations beyond the non-idealities that memristors themselves impose, as discussed before. A few examples of these new potential sources of errors are (1) the effective resistive load (known as R_{Load}) in their circuit [139], (2) The wire resistance and sneak paths, due to imperfect wires (i.e., wires with different resistances) and the changes in the voltages of the internal nodes while performing a VMM operation [140, 141], and (3) non-ideal sensing circuit or ADCs, which happens due to rigid or hard-to-accurately-change references used for distinguishing/sensing the end result [128, 139].

OPTICAL PHASE CHANGE MEMORY

Phase change materials (PCMs) are currently the leading alternatives for non-volatile computation in silicon photonics-based platforms [142]. A design that combines integrated photonics with PCM is commonly known optical phase change memory (oPCM). Compared to diffractive computing in free-space optics [143], oPCM-based designs offer CMOS-compatible manufacturing. Compared to previous photonic-based platforms, oPCM offers higher speed and lower energy consumption for the electronics interface. This is because conventional photonic-based platforms require large and power-hungry phase shifters used in their calibration and reconfiguration [144]. Therefore, a design based on non-volatile PCMs that also exploits integrated photonics can potentially reduce both the cost and the overall footprint of photonic cores for similar logical operations [145]. Cardoso et al. [145] show that, with a realistic noise level, using PCM devices

in a multi-level fashion hurts the accuracy of an oPCM-based design when performing scalar multiplication. However, one can avoid this problem using fewer levels or states in PCM, such as in a binary state. In other words, the binary usage of PCM provides the easiest solution for differentiating between the states.

One can also utilize oPCM in a CIM design. Such an oPCM-based CIM design offers three benefits compared to the same design with electronic-based PCM as the underlying technology:

- **Higher parallelization through wavelength division multiplexing (WDM).** An oPCM-based CIM design allows multiple vectors, which can be processed simultaneously in different parts of the frequency space [146] by WDM.
- **Higher scalability.** oPCM-based CIM designs do not need to combat Joule heating, electromagnetic crosstalk, and capacitance that custom silicon computing platforms using electronic-based PCM require [147, 148].
- **Lower design overheads and considerations.** Current oPCM-based CIM designs are unaffected by variability, resistance drift, and cyclability challenges that affect manufactured electronic-based CIM designs [149–151].

2.1.4. PRIMITIVE FUNCTIONS IN CIM CLASSES

Having the generic illustration of a CIM tile (Section 2.1.2), our classification of four possible computation locations (Section 2.1.1) and crossbar structures we discussed above, one can realize many primitive functions such as NAND, XOR, VMM, AND, etc. in and near the crossbar array. Below, we briefly discuss a few of these functions and how they work. But we refer the enthusiastic reader to previous works to investigate other possible functions enabled by CIM [107, 152–157].

- **NAND [152].** Fig. 2.7 depicts a visual representation of how to realize the NAND using memristors. In this method, one must configure/program the output memristor, where the NAND result will be stored. In Fig. 2.7, the output memristor starts with an initial state of R_{off}, i.e., logical '1'. Then, the bitlines associated with the initial two inputs are subjected to V_{wh}, while the output bitline receives V_{ω}, with the source-line remaining unconnected. With both inputs set to 1, equivalent to R_{off}, a zero voltage passes through the horizontal line, resulting in the output device experiencing a voltage V_w, which in turn switches its resistance from R_{off} to R_{on}. However, if any of the inputs is 0, represented by R_{on}, the horizontal line sees a V_{wh} voltage. Consequently, a voltage difference of $V_{\omega} - V_{wh}$ is present across the output device, an insufficient condition for a state transition in resistance. This is an example of functions in the crossbar (f_{Xbar} in Fig. 2.2 and CIM-A in Fig. 2.1). Note that other methods [158], such as using a 3-input minority function with incrementally decreasing the Minority operation's voltage to a point where all memristor inputs must be at R_{ON} to trigger a state transition in the output memristor, to realize a NAND also exist.
- **XOR [159].** XOR using memristors requires five memristor devices, with two of them functioning as auxiliary elements. Fig. 2.8-(a) depicts the required

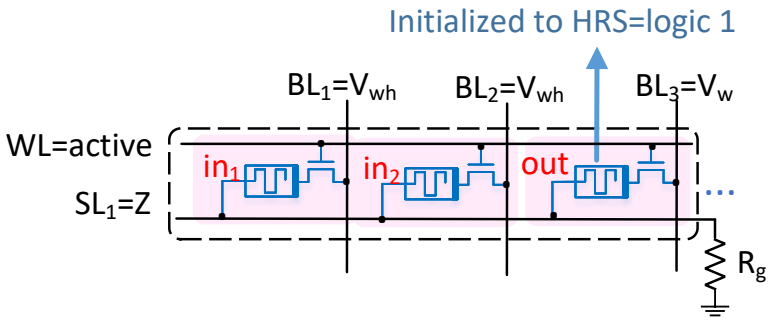

Figure 2.7: NAND design using memristors [152].

connections and initial setup of these memristors. When both memristor devices—representing input parameters—are either in the ON or OFF state, the shared node between these devices essentially serves as ground, preserving the initial state of the output memristor. However, when this is not the case, the voltage at the shared node reaches Vx, resulting in the transition of the output memristor to the LRS state with the assistance of the auxiliary memristors. Fig. 2.8-(b) presents the crossbar configuration and the respective data mapping necessary for realizing XOR in the CIM crossbar structure. This implementation of XOR is also an example of functions in the crossbar (f_{Xbar} in Fig. 2.2 and CIM-A in Fig. 2.1).

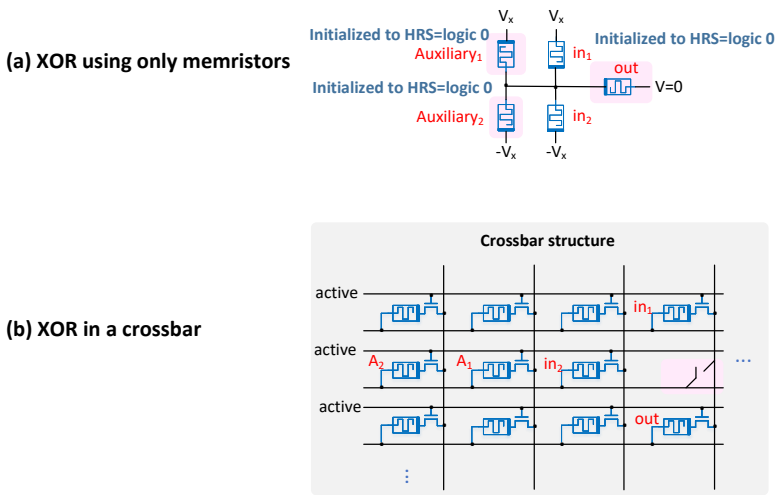

Figure 2.8: XOR function (a) using memristors only and (b) in CIM crossbar [159]

- **VMM [90].** CIM crossbars are very attractive for their capability to execute Vector-Matrix-Multiplication (VMM). These implementations of VMM are examples of

functions in the sensing step (f_{sens} in Fig. 2.2 and CIM-P in Fig. 2.1). Fig. 2.9 presents an overview of how a crossbar of memristors can support Vector-Matrix-Multiplication (VMM) operations. Take the VMM operation in Fig. 2.9-(a) as an example. Fig. 2.9-(b) illustrates how the parameters of this VMM map to a memristor-based crossbar.

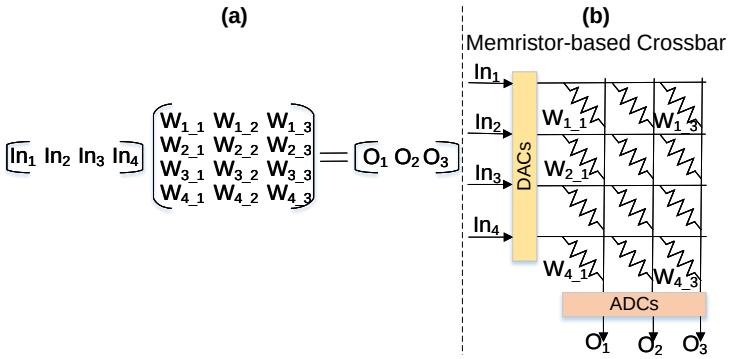

Figure 2.9: Memristor-based CIM support of VMMs.

For Fig. 2.9-(b) to support VMM, one first maps the weight matrix in Fig. 2.9-(a) to conductances of the memristor devices in the crossbar. Then, we apply the input vector of indexed Ins as voltages to the digital to analog converter (DAC) connected to the wordlines of each row in the crossbar. Based on Kirchhoff's and Ohm's law, a current equivalent to accumulated current for element-wise multiplication of individual and corresponding inputs and weights in a column reaches each analog to digital converter (ADC). Thus, each column performs a Multiply-and-Accumulate (MAC) operation in the analog domain, providing us with a VMM operation across multiple columns. Since the columns can work simultaneously, the VMM has an O(1) time complexity in this design. Each ADC converts the currents into digital outputs (Os) and passes them through other systems components for further processing.

As discussed in Section 2.1.3, oPCM crossbars can also be utilized in a CIM design, where they offer several benefits. The abstract idea of VMM on oPCM-based crossbar is very similar to the one on the memristor-based one. However, to understand how an oPCM-based crossbar supports VMM, consider Fig. 2.10 depicting a 3×3 oPCM-based crossbar.

Assume that the amplitude of the electric field (E-field) in each row represents the input data vector (V_{in}). A set of optical DACs (oDACs), along with a splitter tree structure, generates $v_{in_i} \times E_{Laser}/\sqrt{N}$ for every row. It is assumed that the input coupling coefficients in each unit cell (k_{in_j}) are equally distributed across the row, resulting in each unit cell receiving $v_{in_i} \times E_{Laser}/\sqrt{NM}$ through its bent waveguide. The output coupling coefficients (k_{out_i}) receive light uniformly and represent the computed product for every unit cell in a specific column. Unlike [160], this introduces an electric field loss of $1/\sqrt{N}$, allowing the entire array to function

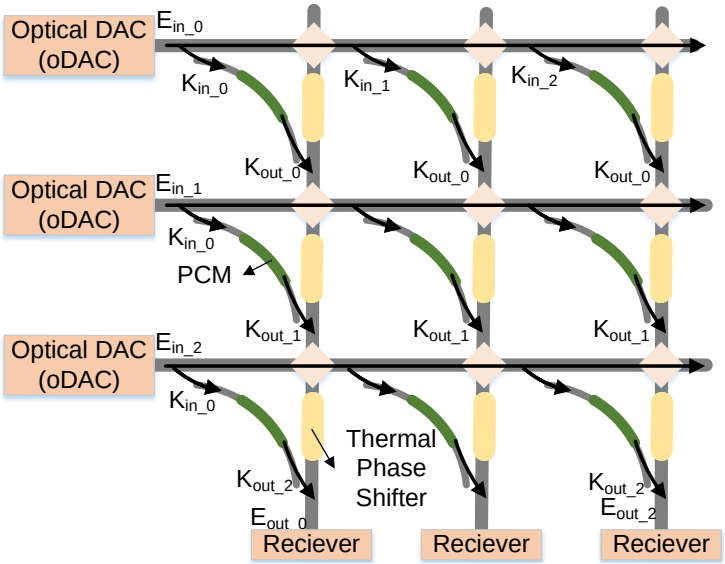

Figure 2.10: Memristor-based CIM support of VMMs.

at a single light frequency within a compact area. This feature is crucial for scaling the arrays. The resulting E-field at the end of each column adheres to Equation 2.4.

$$E_{out_j} = \frac{E_{laser}}{N\sqrt{M}} \times \sum_{i=0}^{N-1} |V_{in-i}| \times w_{i-j} \tag{2.4}$$

The overall E_{out} vector equals VMM of the input data (v_{in}) and the 3×3 weights matrix. This is converted back to the electrical domain using coherent detection by coupling each signal with a certain part of the input laser light in a DC. The outputs of the couplers enter balanced photodiodes (PD), with $I_{out_j} \propto |E_{Laser}||E_{out_j}|$. This method maintains coherency across the entire array, necessitating precise optical path lengths and phase shift matchings. To compensate for potential phase errors caused by process variations or random phase fluctuations, previous work [161] has suggested incorporating a small thermal-phase shifter in each unit cell throughout the column waveguides.

- **AND and OR [106].** Logical operations can also be achieved within the SA of CIM crossbar arrays. Scouting Logic is the famous example of a work that does this and is an example of a function in the sensing step (f_{sens} in Fig. 2.2 and CIM-P in Fig. 2.1). For Scouting Logic, both operand vectors are first programmed into the crossbar array, as shown in Fig. 2.11. With choosing the reference or references of the SA, one can implement the logical OR, AND, and XOR functions. The primary benefit of this approach, as compared to integrating these functions within the crossbar itself, is a reduction in the frequency of device programming. This is

particularly critical due to the endurance issues and the substantial energy consumption involved in the programming of memristor devices.

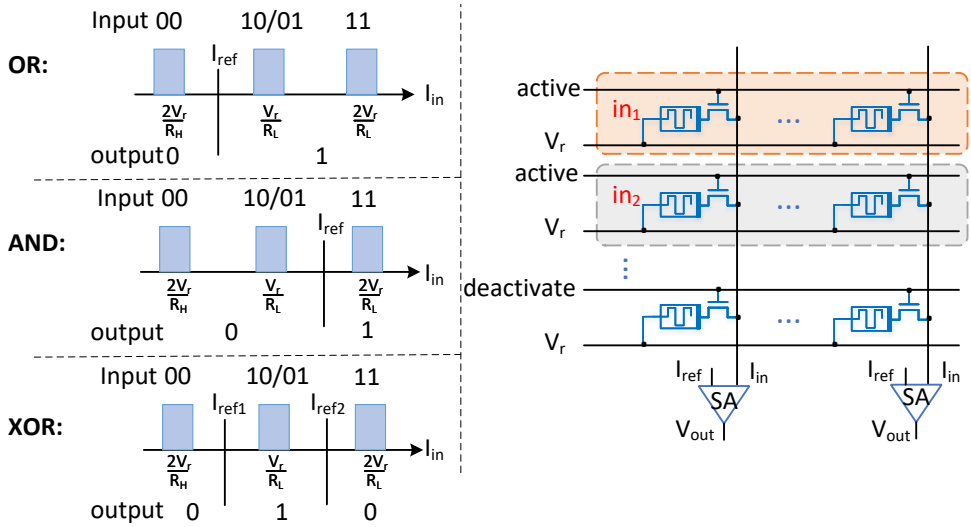

Figure 2.11: Scouting Logic [106] for AND, OR, and XOR.

2.1.5. ABSTRACTION OF CIM DESIGN CHOICES

After reviewing prior sections, it becomes evident that we can have two main strategies for deploying CIM designs: (1) accelerators tailor-made for specific applications, and (2) universal and general CIM design capable of handling multiple applications. In this thesis, we explore both of these research directions.

To this date, the former approach has been the most prevalent one. Works in this direction are geared towards predetermined data flow and controls, offering little flexibility. While this direction allows for a high degree of optimization, its use is restricted due to its rigidity - any changes at the application level may necessitate a complete redesign of the accelerator.

The latter approach, universal CIM, has been less explored as it requires the capability of executing a variety of applications which brings with it greater complexity and more stringent requirements. To facilitate this direction, one can think of different strategies at various abstraction levels, some of which Fig. 2.12 captures.

Fig. 2.12-(a) depicts three potential design abstraction levels of CIM. At the nano-level, a CIM tile might only support a rudimentary function at one of the stages, with no significant functionalities accommodated in the remaining stages. MAGIC [107] or Scouting [106] are two examples of this design choice. Alternatively, various functionalities can be supported at each tile stage, culminating in a complex function.

At the intra-tile/micro level, we have two design choices: Static or Programmable CIM tile. In the case of Static, each stage of the tile is custom-designed to provide specific functionality. However, in a programmable tile, multiple functions are assigned to each

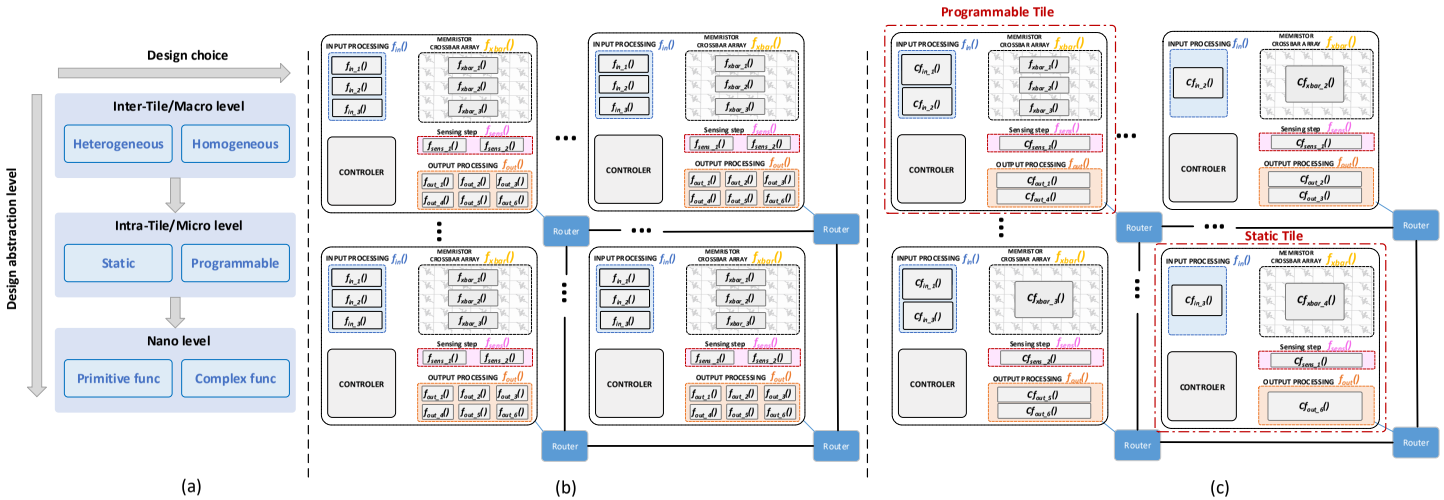

Figure 2.12: (a) CIM design choices at separate abstraction levels, (b) homogeneous design: each individual tile possesses sufficient compute capability and generality to accommodate a diverse range of functionalities, and (c) heterogeneous design: the tiles are tailored to execute specific kernels efficiently [128].

step of the tile.

At the highest abstraction level, inter-tile or macro level, designers can opt for a homogeneous CIM tiles system where all tiles are identical and capable of a wide range of functionalities and requirements. On the other end of the spectrum, there is the option for a heterogeneous design wherein each CIM tile is customized for a specific purpose, collectively offering a comprehensive solution for the programmer. A hybrid solution can also be considered, wherein a degree of heterogeneity is introduced, but each CIM tile is designed to support as many functions as feasible.

Fig. 2.12-(b) and Fig. 2.12-(c) represent design choices at the inter-tile abstraction level. If we opt for a homogeneous design, executing an application becomes easier due to the availability of more resources, as the tiles are designed to be versatile. However, this could result in overheads regarding area, energy, and latency. Moreover, the complexity of the tile may necessitate a sophisticated controller. Alternatively, in a heterogeneous design, resources for a specific application may be limited, but the tiles could potentially offer better energy, area, and performance efficiency. Yet, due to the diversity, a more intricate task offloading scheme might be required. Moreover, inter-tile communication might cause additional overhead if the tiles selected for an application are located far apart. The decision between these options largely hinges on how adaptable the circuit in the CIM tile will be to the functions and requirements of the applications. Regardless of the chosen approach, having flexible control over this storage unit, both in terms of inter- and intra-tiles for varying execution flows, is of utmost importance. This can be managed completely at the hardware level or partially at the software level with the aid of a compiler (or scheduler) [113]. This becomes even more crucial in heterogeneous designs where specific workloads must be mapped and scheduled for particular CIM tiles.

To summarize, the path to a generalized CIM-tile design is paved by (1) expanding the functionality of a CIM tile with a standard circuit, (2) developing a flexible interface between CIM tiles and between the tiles and the host, and (3) designing an advanced control system capable of effectively handling various scenarios.

2.2. MODERN APPLICATIONS

2.2.1. BIOINFORMATICS AND GENOMICS

Genomics is a branch of genetics that utilizes recombinant DNA, DNA sequencing, and bioinformatics techniques to sequence, put together, and delve into the structure and function of genomes, which encompasses all the DNA in a single organism's cell. This field investigates the interactions between genes and non-genic genome regions and how they mold biological functions and observable traits. Genomics brings about a remarkable increase in our comprehension of biology and illness. By offering an all-encompassing view of gene operations and their interactions, genomics sheds light on the genetic foundations of diseases, paving the way for superior diagnostics and treatment approaches. Moreover, it lays the groundwork for progress in personalized medicine, allowing for treatment methods to be customized based on a person's genomic data. Beyond the sphere of human health, genomics has broad-ranging consequences in other domains like agriculture, where it assists in breeding crops

with coveted traits and ecology by facilitating an understanding of species and their evolutionary processes. All in all, advancing genomics studies that hold the key to unlocking the potential of precision medicine, facilitating virus surveillance, and driving advancements in healthcare [1–14, 16, 17].

Bioinformatics is an interdisciplinary field aiming to understand large and complex biological data (i.e., data in genomics) through mathematical and computational models that use computer programming. Bioinformatics studies consist of several genome analysis pipelines designed to enrich our understanding of a particular problem in genomics. Although a complete list of pipelines and available methods is out of the scope of this report, Fig. 2.13 presents a few famous genomic pipelines with the algorithms and kernels that each step of them uses. Each of the rectangular boxes in Fig. 2.13 are called a kernel or genomics step interchangeably. The names on the top or bottom of each box represent famous example algorithms or examples of that step. We call each path from left to right a pipeline. In the following, we briefly discuss a few of the kernels that are most connected to this thesis.

- **Basecalling.** *Basecalling* is a computational step required to acquire strings of DNA nucleotide bases (i.e., {A, C, G, T}) from noisy electrical signals generated by modern sequencing machines [162–166]. Basecalling is the very first computation step in many genome analysis studies working with these nucleotide sequences, i.e., DNA reads [167, 168]. The accuracy of basecalling directly affects the accuracy and the computational effort (in terms of required algorithms) of subsequent genome analysis steps. Moreover, the speed of basecalling also determines how fast one can run through all computational steps of a genomic study [169]. Therefore, accurate and fast basecalling is critical for advancing genomic studies that hold the key to unlocking the potential of precision medicine, facilitating virus surveillance, and driving advancements in healthcare [1–14, 16, 17].

  Current SotA basecallers leverage Deep Neural Networks (DNNs) to achieve high accuracy [170–172]. Recent works [169, 173–175] heavily investigate the use of DNNs for basecalling as they can provide the highest accuracy compared to the Hidden Markov Model (HMM) based procedures [176].

  There are generally two approaches for improving the accuracy and/or performance of a basecaller: software-based and hardware-based. Software-based methods propose new algorithms (e.g., DNNs [174, 175, 177] instead of HMMs [176]) or faster and/or smaller DNN architectures [177, 178]. On the other hand, hardware-based approaches propose various hardware platforms for the target algorithm (i.e., DNN or HMM) to improve performance with minimal to zero impact on accuracy [179].

- **Alignment/Mapping.** Sequence alignment[1] is a fundamental step in most genomic studies that help us with outbreaks surveillance, precision medicine, and other medical advances [1, 3, 7]. Sequence alignment is finding the similarity/closeness between a reference genome sequence (hereafter called reference)

[1] Also known as mapping

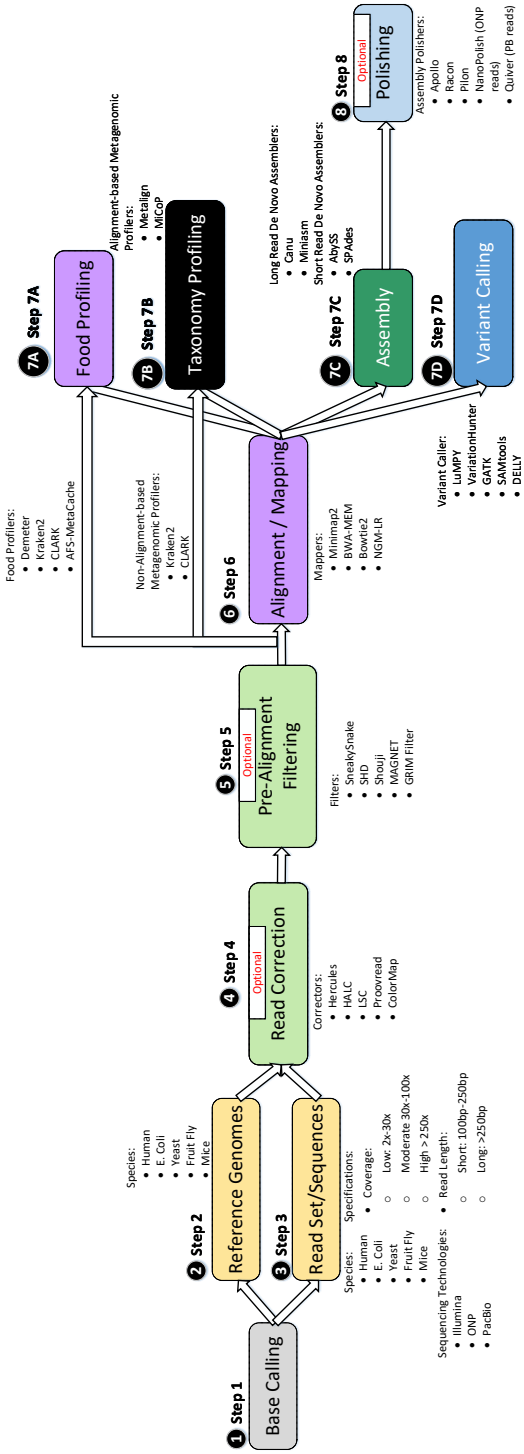

Figure 2.13: Overview of several genome analysis pipelines, corresponding tools, and algorithms.

and a DNA read sequence (hereafter called read). Unfortunately, the DNA base-pairs (e.g., A, C, G, T) in references and reads may not always be identical at the location the read actually comes from for two reasons: (1) errors that arise when obtaining the sequences (a process called genome sequencing [180, 181]), and (2) genetic differences that exist among an individual organisms' DNA and corresponding reference [182]. Therefore, the sequence alignment process should be able to tolerate such differences, commonly known as edits: deletion, insertion, or substitution. To deal with this requirement, SotA sequence alignment methods employ computationally costly dynamic programming-based (DP) algorithms such as Needleman-Wunsch or Smith-Waterman algorithms [183–185] to account for edits while avoiding duplicate works. Unfortunately, these DP algorithms are computationally costly and incur long latencies and energy inefficiencies when applied to large DNA sequences. These limitations directly affect the medical studies that benefit from sequence alignment.

- **Pre-alignment Filtering.** Pre-alignment filtering is a heuristic-based method to mitigate the cost of sequence alignment by quickly eliminating the need for performing the expensive DP given a pre-defined threshold called "edit distance." SneakySnake [186], Shouji [187], MAGNET [188], and SHD [189] are a few widely-used examples of such filters. SneakySnake [186] is the most recent of such filtering techniques that proposes to reduce the approximate string matching (ASM) problem to the single net routing (SNR) problem to find the optimal path with the least routing cost. This tweak enables SneakySnake to filter most unnecessary alignments in a parallel and highly accurate manner. Alser, et al. [186] show that this conversion also makes SneakySnake suitable for other high-performance computing (HPC) architectures, e.g., GPUs.

  Pre-alignment filters are typically compared based on 4 rates [95, 186, 187]: True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN) rates. TP is the ratio of pairs that filter correctly accepts/passes as they require DP-based alignment. The higher the TP, the better. TN is the ratio of pairs that filter correctly rejects/filters as they are too far apart to be mapped and therefore do not require DP-based alignment. TN is proportional to the FP rate, and we always want to maximize it. FP is the ratio of pairs that filter incorrectly accepts/passes even though they do not require DP-based alignment. The lower the FP, the better, as we would spend less time and cost for the alignment. FN the ratio of pairs that filter incorrectly rejects/filters even though they will be mapped and therefore require DP-based alignment to find the final mapping. An ideal filter has a FN of 0.

- **Taxonomy Profiling.** Recent advances in high-throughput sequencing (HTS), namely producing sequenced data with high-throughput and low cost, initiated metagenomics [11, 190, 191]. In metagenomics, researchers study the behavior of many species altogether in a sample taken directly from an environment. The results of such a study help researchers to capture the complex relationship between different species without cultivating or isolating them individually in a very costly or yet impossible procedure for some species.

Taxonomic profiling is the first step of any metagenomic study [192, 193] and it determines the relative abundance of different taxonomy ranks (e.g., species, genus, family) in a given sample. Taxonomic profiling is divided into two main categories [194]: reference-free and reference-based profilers.

Reference-free Profilers. MetaPhlAn [195, 196], PhymmBL [197], and PhyloPythiaS+ [198] are a few examples of reference-free profilers. These profilers are typically slow. They also require a relatively long query sequence for their composition feature needed for the classification. Therefore, although they are actively being investigated, researchers and industry do not currently use them for taxonomic profiling.

Reference-based Profilers. Reference-based taxonomic profilers can be further divided into two main classes: alignment-based and non-alignment-based (also known as heuristics). The alignment-based profilers are highly accurate (especially at the species rank). However, alignment-based profilers are very slow due to the high computational cost of their alignment. A few examples of such profilers are Metalign [192], MG- RASTv.4 [199], MEGAN6 [200], and Taxator-tk [201]. Non-alignment-based profilers, or heuristics, replace the time-consuming alignment operation with a faster *Table Lookup* operation. This way, taxonomic profilers in this group trade the required execution time with the memory needed for their table and lookup operation, i.e., they gain speed but require more memory. Kraken [202], Kraken2 [193], and CLARK [203] are a few examples of such profilers.

Kraken2+Bracken[2] always stands among the top taxonomic profilers and binners, varying just a little from dataset to dataset, based on the most comprehensive benchmark for metagenomics, Critical Assessment of Metagenome Interpretation (CAMI) challenge [205, 206]. It is worth noting that most of the highly ranked profilers in the CAMI challenge are non-alignment-based profilers.

Kraken [202] is a non-alignment-based taxonomic profiler that utilizes exact-match database queries of small substrings from the main read, called k-mers. Kraken, first, stores all k-mers within the sequence into a set. It then maps each k-mer inside the set into the lowest common ancestor (LCA) taxon of all the genomes in the reference database that have the special k-mer. This LCA texa and its ancestors in the taxonomy tree form the classification tree used to classify the input sequence. The classification path is defined as the maximum scoring root-to-leaf (RTL) path in the classification tree, and the sequence S is assigned to the label of the corresponding leaf. Kraken owns its efficiency not only to the classification algorithm but also to its database creation, in which it uses the notion of minimizers [207] and two tables for performing an efficient search. Kraken groups similar k-mers using minimizers, defined as the smallest M-mers among all M-mers in a k-mer when sorted lexicographically. Since adjacent k-mers share the same minimizers in practice, Kraken stored the k-mers with similar minimizers consecutively and sorted them in the lexicographical order of their canonical representations. This enables Kraken to query a k-mer by

[2] Bracken [204] is an orthogonal method to Kraken2 and other profilers to re-distribute reads in the taxonomic tree and improve the accuracy. We discuss Bracken further next.

looking up in an index position and finding the range where k-mer with the same minimizer as the query has been stored and then perform a binary search within the region to find the exact k-mer and corresponding taxonomy ID.

Kraken, while effective, uses a memory-intensive algorithm for assigning queries to the lowest common ancestor (LCA) taxonomic label. Kraken2 [193] improves performance and memory consumption of Kraken by building a more compact reference database using probabilistic hash functions. Kraken2 significantly reduces the memory requirement to a third while maintaining accuracy. Kraken2 also takes advantage of block-based and batch-based parsing within the critical sections to further improve thread scaling, similar to what has been done in Bowtie [208]. Both Kraken and Kraken2 use extensive index (hash) tables to store a pre-built data structure to help accelerate their assignment. Therefore, they both perform multiple *Table Lookup* operations to map a DNA read to an LCA.

To prevent underestimating the abundance of some species, we typically use Bracken [204] (Bayesian Reestimation of Abundance after Classification with KrakEN) along with Kraken2. Bracken proposes to probabilistically re-distribute reads in the taxonomic tree so that estimating the abundance of species will become possible. The re-distribution in Bracken works in both directions: 1) Reads that are originally assigned to nodes above species levels will be re-distributed to this level, 2) Reads that are originally assigned to nodes in the strain level will be re-distributed to their parent species node. Bracken is orthogonal to Kraken and Kraken2.

- **Food Profiling.** The urgent need for a real-time, efficient, and accurate food monitoring system is apparent when one considers the economic impacts and health risk issues due to human errors and/or intentional fraud regarding everyday food. For example, a worldwide annual loss of $10 to $24 billion of dollars is estimated only for the frauds happening in the fish industry [209]. The Halal meat scandal [210] and the black fish scandal [211] are just a few other preventable examples that could have been quickly prevented if we had accurately and efficiently monitored all the food productions in real-time.

  Food profiling is the first step and the only computationally expensive task in a food monitoring system. The food profiling task entails the functionality of determining the existing species in a food sample and their relative abundance [212, 213]. Today's food profilers work with sequenced data as we can capture a more accurate profile using the sequences of a food sample. The rapid drop in the cost of DNA sequencing in the past decades and the expectation for a continual trend [26, 27] is expected to lead the way for profiling to become the main bottleneck of this pipeline.

  Currently, the industry utilizes state-of-the-art (SotA) taxonomic profilers from metagenomic studies for food profiling due to the similarity of problem statements in food profiling and metagenomics profiling. However, as alluded, such profilers are developed as the first step of metagenomic studies [11, 190, 191]: a new, yet different, line of research that allows us to study many species that are

taken directly from their environment altogether, as opposed to studying them individually. Due to the high cost associated with alignment and assembly for large reference datasets, to this date, we still prefer heuristics statistical-based profilers to assembly- or alignment-based ones. However, even these profilers are not yet cheap or economical and prevent large-scale, real-time studying. Their cost is mainly related to the required memory for profilers' data structure and algorithms. Such large data structures or sophisticated algorithms force us to use high-end servers and are needed to fulfill complex goals of subsequent metagenomic analysis, namely capturing complex operations between organisms and discovering insights on species that can not be clonally cultured in labs. This high cost of profiling in a metagenomics profiler prevents us from efficiently profiling food samples in real-time, the end goal of a food monitoring system.

Most of the kernels in the pipelines of Fig. 2.13 work on large amounts of data. Fig. 1.1 depicts the significant reduction in the cost of data acquisition in genomics, next to Moore's law ascending growth, helping the exponential growth of the data working set size of its kernel. It is expected that the available data in the field and the rate at which they are being produced soon surpass that of giant data-intensive applications such as YouTube and Twitter exploit and produce [25, 26]. Therefore, they require many memory accesses, which can quickly become very costly and turn into the bottleneck of the whole procedure. Simultaneously, we demand faster analysis for those larger data working set. This trend worsens the data movement problem in genomic pipelines.

Moreover, sequencing machines are heterogeneous systems that already use various memory technologies (DRAM, SDD, etc.) and computational units (CPU, GPU, and FPGA) since their benefits justify their cost. Therefore, having new architecture and emerging technologies installed in those machines as well is not a far-fetched idea if we can harvest their power efficiently.

We conclude that kernels on genomics pipelines might also benefit from the CIM computing paradigm and emerging memory technologies although such a research direction is yet not well-studied. Note that although not all of the genomics steps in Fig. 2.13 might be able to use CIM and emerging memory technologies, some still can.

2.2.2. NEURAL NETWORK

Neural Networks (NNs) have become essential in various applications like language processing [18] object recognition [19], and image classification [20, 21]. In particular, momentous developments in Deep Neural Network (DNN) in the past decade have led to significant improvements in the accuracy and execution time of computer vision tasks such as object detection and recognition [22–24].

Hardware implementations of NNs significantly impact the performance metric (i.e., accuracy, execution time, energy consumption, etc.) of the underlying application. However, current DNN hardware implementations are relatively slow and costly to run [214–216] because of their already large working dataset size, growing working dataset size, and inefficient underlying hardware that we discuss further next.

Regarding the working dataset size, SotA NNs already work with many parameters, e.g., BERT has 110 million parameters [217], causing the implementation to face the issue widely known as the 'memory wall', discussed in Section 1.2.

Fig. 2.14 and Fig. 2.15 depicts the growth rate of the dataset size in NNs we use for image classification, recommendation (i.e., collaborative filtering), speech recognition, text-to-image generation, and gaming (e.g., Atari and Go) tasks. We observe that while today NNs are already huge and work with many parameters, their working dataset size is also steadily and significantly increasing as time passes by.

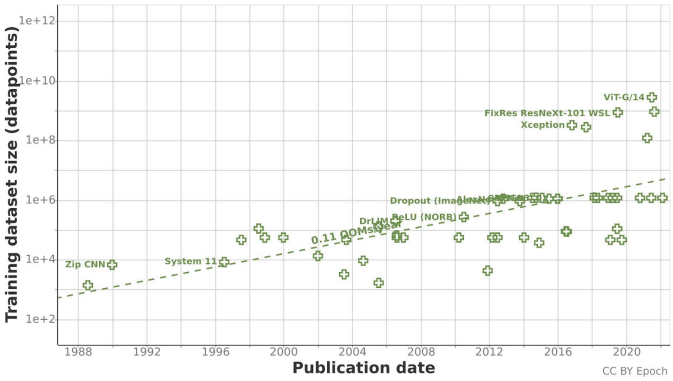

Figure 2.14: The growth rate of dataset size in ML for vision based tasks such as image generation and classification [28].

Finally, conventional systems used for the execution of NNs typically use (1) von-Neumann architecture that suffers from data movement between the processor and memory even further [218] and (2) expensive hardware such as storing weights in 6 transistors SRAM cells [216, 219] that increases the hardware cost.

Hence, developing a high-throughput, cost-effective hardware realization of DNNs while being accurate is critical.

A CIM architecture is suitable for mitigating the data movement overhead in many applications. Moreover, CIM based on emerging technologies can handle simple operations such as Matrix-Matrix-Multiplication (MMM), which is the key computation in NNs[3]. A CIM design can also offer the high parallelism suited for running NNs with high speed. Hence, we conclude that CIM is a suitable candidate to be used for the hardware acceleration of NNs.

Although many network architectures exist for NNs, here we briefly discuss the necessary background for only Binary Neural Network (BNN) as the later chapters in this thesis require a basic understanding of the features of this architecture.

A BNN works with binarized weights and activations (e.g., {-1, 1} or {0, 1}) instead of datatypes with higher precision. This change provides the BNN with two advantages [215, 221, 222]. First, it reduces the storage requirement of the NN. Second, it changes the MAC operation from high-resolution multiplication and addition to a low-cost and simpler XNOR followed by an Popcount operation [221, 222]. Equation 2.5 depicts this conversion, where $\circledast$ is convolution, $\odot$ is XNOR, and Popcount (or population count) of a vector or specific value is the process of finding the number of set bits (1s) in

[3] It is known that MMM (or VMM) is the dominant operation in NNs [220].

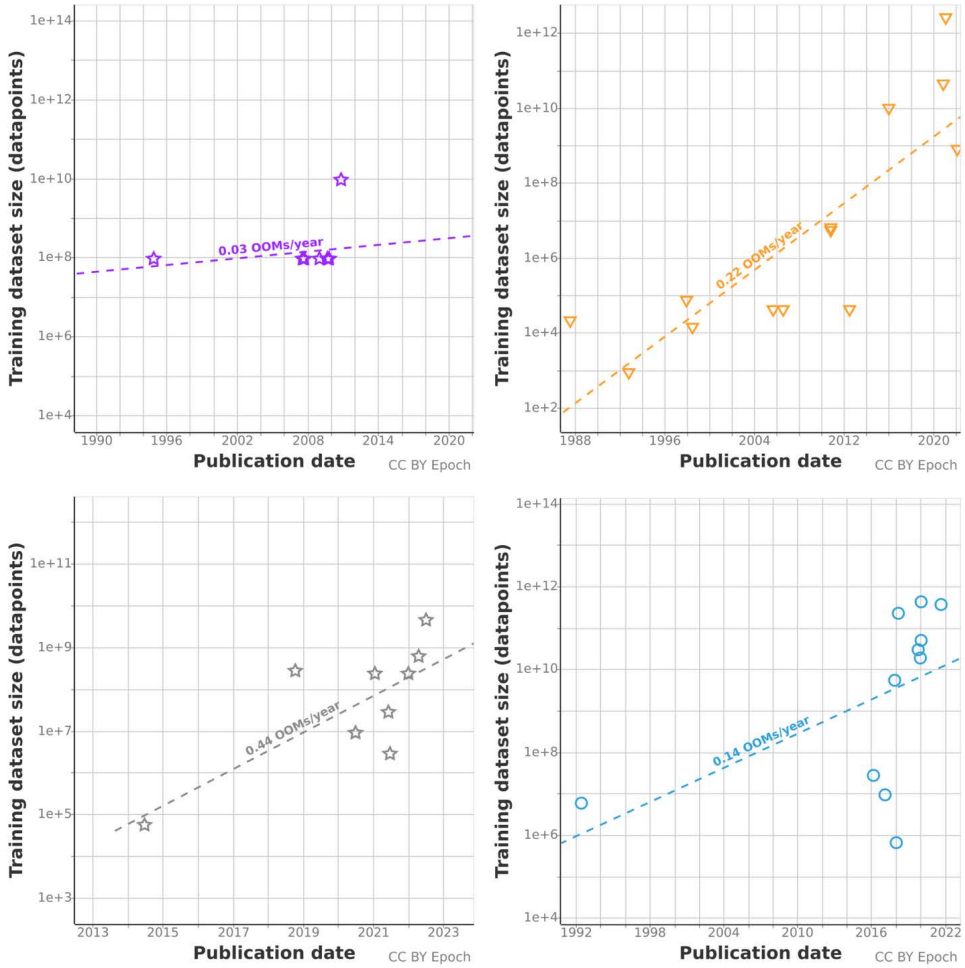

Figure 2.15: The growth rate of dataset size in ML for four tasks: (top left) Recommendation, (top right) Speech, (bottom left) Drawing, and (bottom right) Games tasks [28].

that vector/value. In Equation 2.5, In and W are equally-sized input and weight vectors of target BNN, i.e., vector length is the length of either In or W vector.

$$In \circledast W = 2 \times Popcount(In^{'} \odot W^{'}) - Vector\ Length \tag{2.5}$$

Unfortunately, naively reducing both activations and weights to binary representations hurt the overall accuracy compared to high-precision (floating point or fixed point) networks. Therefore, to combat this accuracy loss, previous works [215, 221] generally follow two software-based techniques. First, tracking the updates of parameters during training via higher resolutions (floating or fixed point) while keeping the actual weights binarized. Second, using binarized activations and weights only for hidden layers and keeping the input and output layers in higher resolutions.

2.3. State-of-the-Art CIM Designs and Simulators

2.3.1. General-Purpose State-of-the-Art CIM Designs and Simulators

Some works aim to realize the a generalized platform based on CIM that covers more than a single application. However, as we see next, these works been rather limited thus far.

UPMEM

UPMEM [223, 224] stands out as the pioneering commercial generic platform for CIM. Rather than utilizing emerging non-volatile memories, this platform is grounded on computation proximate to DRAM memory arrays. UPMEM incorporates numerous Data Processing Units (DPUs) located uniquely within the DRAM memory chips, close to the data, allowing them to execute data-intensive operations while dramatically minimizing off-chip data movements. These DPUs are governed by a high-level application executing on the primary CPU, which manages task orchestration.

ReVAMP

ReVAMP [225] is a design that incorporates resistive memory into a pipelined processor, using the memory to substitute both the cache and the register file. This design introduces a novel instruction, known as 'Apply', to perform a computation operation (Majority) within the memory crossbar. Note that, as previously discussed, emerging memory technologies, such as memristors, tend to have slower performance and quicker aging compared to SRAM. As a result, replacing cache and the register file with memristors could potentially cause a decrease in the processing speed of the processor.

PUMA

PUMA[4] is a complete set of (micro)architecture, simulator, and compiler that supports the execution of many ML applications [108, 111, 226], using memristor crossbars enhanced with general-purpose execution units. In other words, PUMA design facilitates general-purpose CIM, aided by programmability. PUMA uses a spatial architecture and

[4] Programmable Ultra-efficient Memristor-based Accelerator.

provides the necessary programmability and generality to execute a wide range of ML-based applications on memristor-based crossbars. PUMA architecture comprises three pipeline stages: fetch, decode, and execute. PUMA enriches crossbars with an instruction execution pipeline and a specialized Instruction Set Architecture (ISA). The newly defined in-memory instructions within this ISA primarily facilitate data communication between memory units or carry out scalar operations in digital peripheries. Fig. 2.16 depicts the overall Core architecture in PUMA. PUMA places various vector and scalar functional units in the periphery to support a wide range of operations adjacent to the crossbar. PUMA compiler converts high-level code into PUMA ISA.

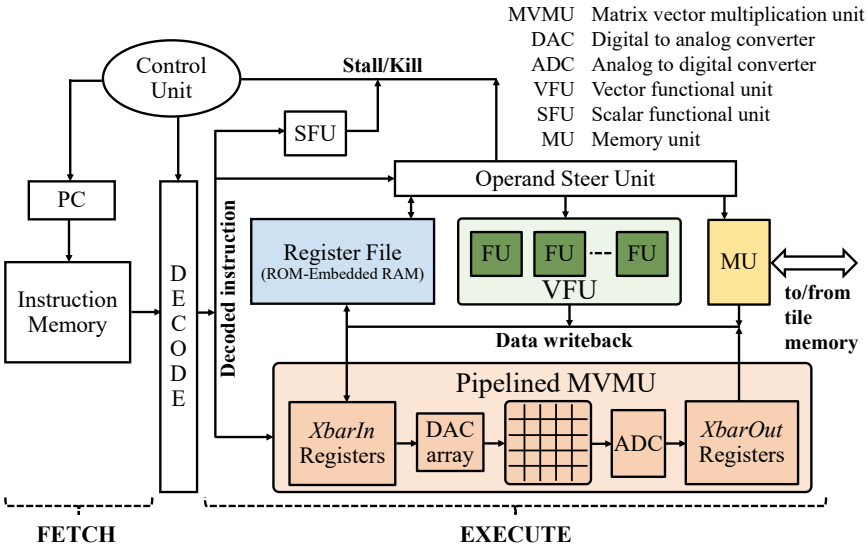

Figure 2.16: The Core architecture in PUMA [108].

IN-MEMORY DATA PARALLEL PROCESSOR

Daichi et al. introduce "In-memory data parallel processor" [156], new in-memory instructions for their CIM design. These instructions represent complete operations that must be executed on crossbars, drawing on the Single Instruction, Multiple Data (SIMD) execution model to exploit substantial data parallelism. In this work, the basic operations facilitated within the crossbar include multiplication, addition, and subtraction. The communication between crossbars are modeled behaviorally where shared buses and H-Tree networks are used without detailed reasoning.

GENERAL SIMULATORS

The design of in-memory memristor-based architectures demands consideration of a multitude of factors. These considerations significantly influence system performance, energy consumption, area utilization, and even the accuracy of operations. As a result, the need for optimization and design space exploration arises, leading researchers to de-

velop high-level simulators that can achieve these tasks within a reasonable simulation time frame.

NVSim [227] and NVMain [228] are among the first memory-oriented simulators developed specifically for non-volatile memories. These tools, inspired by CACTI [229] can provide estimations for access time, energy consumption, and area utilization of non-volatile memories. Similarly, MNSIM [110] presents a behavioral simulation platform for neuromorphic accelerators, estimating design parameters through analytical equations.

Lee et al. propose a system-level simulator [230], which employs probability functions to measure the accuracy of ReRAM cells, thereby allowing evaluation of an application's behavior in terms of accuracy.

Zahedi et al. [113] propose a cycle-accurate simulator capable of executing in-memory instructions/operations. Their simulator enables users to monitor all control signals and the contents of crossbars/registers, resulting in more precise results for performance and energy consumption at the tile level.

2.3.2. Specific-Purpose State-of-the-Art CIM Designs and Simulators

SotA CIM for genomics

A few works improve genomic kernels using the CIM paradigm. GenASM [231] exploits the 3D-stacked memory systems and proposes a framework to accelerate the approximate string matching (ASM) problem. DARWIN [232] is a co-processor hardware accelerator for sequence alignment. DARWIN uses a filtering algorithm (D-SOFT) and a new algorithm to perform the long sequence alignment using constant memory. GenCache [233] exploits in-cache operations and proposes a fast sequence alignment accelerator. This way, GenCache improves the memory bandwidth demands of GenAx [234]. RADAR [235] also proposes a DNA alignment accelerator. However, unlike GenCache, RADAR uses 3D-stacked ReRAMs to solve the data movement problem of the BLASTN algorithm. PIM-Aligner [236], AlignS [237] and [238] are all similar proposals (same author) for performing alignment operations on SOT-MRAM. GenieHD [239] and HDNA [240] use hyperdimensional computing (HDC) for (partial) sequence alignment of a single reference genome divided into multiple pieces[5]. Finder [241] and EXMA [242] propose CIM-enabled accelerators for FM-index using ReRAM as their underlying memory technology. Helix [179] designs a basecaller on top of SOT-MRAM. Laguna et al. [243] propose an in-memory architecture using TCAMs for seed-and-vote read mapping. GRIM-Filter [95] is the first and only pre-alignment filter implemented using the CIM paradigm. GRIM-Filer is optimized to exploit 3D-stacked memory. Sieve [244] is a high-throughput k-mer matching technique that uses in DRAM computation. MEDAL [245] proposes a PIM-enabled accelerator for seeding on DIMM between DRAM modules. MEDAL utilizes the memory bandwidth very efficiently and has fine-grained memory accessibility. NEST [246] is a DIMM-based PIM-enabled architecture for k-mer counting.

Unfortunately, few studies (1) consider genomic kernels as a potential CIM candidate, (2) develop necessary end-to-end data mapping, execution flow, and operations for kernels in genomic kernels, (3) provide insights into how the bottleneck shifts after

[5] Neither GenieHD nor HDNA is capable of producing the exact type and location of edits between their query and the reference genome as in the typical outputs (.sam file) of a sequence aligner. Hence, the term "partial."

employing CIM-enabled accelerators, and (4) target one memory technology for all kernels as a candidate memory technology to be included in next generation of sequencing machines. Our research aims to solve these shortcomings. We hope our works enable scientists and engineers to continue obtaining insights from the gnomic data, albeit their ever-growing datasets using less costly (more energy efficient) and faster CIM-enabled implementation.

SOTA CIM FOR NNS

Several previous works propose CIM designs for specific types of NNs. Most of the proposed accelerators by previous works are used solely for inference, while a few also perform training. This is because the training phase requires frequent device programming and captures small iterative weight changes, necessitating complex datatype and data mapping. Also, more complex operations and data flow support are needed. Due to these complexities, researchers often favor using CIM accelerators mainly for inference, even though a few works target hardware acceleration for training as well [247, 248].

We classify these inference-only accelerators based on the structure of the underlying NN they support into four categories (1) Convolutional Neural Network (CNN) and Deep Neural Network (DNN), (2) Spiking Neural Network (SNN), (3) Recurrent Neural Network (RNN), and (4) Binary Neural Network (BNN) as their hardware realization typically varies significantly from one another. We have already discussed BNNs and their specific CIM accelerators in Section 2.2.2. We discuss the rest in the following briefly.

- For CNNs and DNNs we have PRIME [76], ISAAC [90], Pipelayer [157], and other CIM accelerators [249]. PRIME offers a microarchitecture using ReRAM devices, part of which serves as an accelerator for NNs, the rest as storage. ISAAC introduces a pipelining mechanism for CNN using CIM architecture to reduce inter-layer buffer size. Pipelayer and another accelerator [249], both designed for training and inference. Pipelayer exploits inter and intra-layer parallelism by analyzing data dependency. However, Liu et al. [249] also consider the impact of memristors' inherent variations and programming voltage errors on different networks into account.

- For SNNs, some accelerators suggest using memristors and CIM architecture as required devices such as neurons, synapses, and neuronal circuits. Numerous studies [250–257] model neurons with memristors to simplify circuits traditionally made with many transistors through CMOS technology. Another work [257] exploits memristor devices' stochasticity to mimic the functionality of a spiking neuron. Zhao et al. [258] use memristor-based inhibitory synapses to reduce the complexity of lateral inhibition and homeostasis mechanism in SNNs. Several works [259–263] focus on memristor-based learning, primarily supervised and unsupervised Spike Time Dependent Plasticity (STDP) learning.

- RNNs, used in language modeling, speech recognition, and sequence classification, have a distinct structure and data flow as the current output relies on both the input and the previous output. Multiple works [264–270] implement RNNs on memristor crossbars. One example [267] demonstrates an LSTM network with

memristor crossbar arrays achieving high speed-energy efficiency. Another work [266] shows software weight-mapping and programming strategies for hardware analog conductances that maintain accurate weight programming despite device variability.

3

Swordfish: A Framework for Evaluating DNN-based Basecalling using Computation-In-Memory with Non-Ideal Memristors

Basecalling, *an essential step in many genome analysis studies, relies on large Deep Neural Networks (DNNs) to achieve high accuracy. Unfortunately, these DNNs are computationally slow and inefficient, leading to considerable delays and resource constraints in the sequence analysis process. A Computation-In-Memory (CIM) architecture using memristors can significantly accelerate the performance of DNNs. However, inherent device non-idealities and architectural limitations of such designs can greatly degrade the basecalling accuracy, which is critical for accurate genome analysis. To facilitate the adoption of memristor-based CIM designs for basecalling, it is important to (1) conduct a comprehensive analysis of potential CIM architectures and (2) develop effective strategies for mitigating the possible adverse effects of inherent device non-idealities and architectural limitations. This chapter proposes Swordfish, a novel hardware/software co-design framework that can effectively address the two aforementioned issues. Swordfish incorporates seven circuit and device restrictions or non-idealities from characterized real memristor-based chips. Swordfish leverages various hardware/software co-designed solutions to mitigate the basecalling accuracy loss due to such non-idealities. To demonstrate the effectiveness of Swordfish, we take Bonito, the state-of-the-art (i.e., accurate and fast), open-source basecaller as a case study. Our experimental results using Swordfish show that a CIM architecture can realistically accelerate Bonito for a wide range of real datasets by an average of 25.7×, with an accuracy loss of 6.01%.*

This chapter is partially based on the candidate's work [116, 271].

As we briefly discussed in Section 2.2.1, *Basecalling* is the first computational step required to translate noisy electrical signals generated by modern sequencing machines to strings of DNA nucleotide bases (i.e., {A, C, G, T}), also known as DNA reads or simply reads [162–168, 272, 273]. The accuracy of basecalling directly affects the overall accuracy and the computational effort (in terms of required algorithms and their complexity and runtimes) of subsequent genome analysis steps. The speed of basecalling also determines how fast one can run through all computational steps of a genomic study [169, 272, 274]. Therefore, accurate and fast basecalling is critical for advancing genomic studies that hold the key to unlocking the potential of precision medicine, facilitating virus surveillance, and driving advancements in healthcare and science [1–14, 16, 17, 273, 275].

Current State-of-the-Art (SotA) basecallers leverage Deep Neural Networks (DNNs) to achieve high accuracy [170–172, 175, 177, 274]. However, SotA DNN-based basecallers encounter different shortcomings when implemented using different approaches. Specifically, DNN-based basecaller designs on central processing units (CPUs) and graphics processing units (GPUs) face multiple major shortcomings: (1) they are computationally intensive and slow [169, 272, 274], (2) they require extensive data movement between the processor and memory [42, 218, 276], and (3) they are limited by the use of costly hardware, such as expensive SRAM memories that require 6 transistors for storing only 1 bit of information [215, 216]. When implemented on a hardware accelerator, these DNN-based basecallers face two other limitations: (1) They rely on costly floating-point (FP) computations, which place high demands on the required system's memory bandwidth and compute units with FP capability. This makes hardware acceleration difficult due to the large number and size of neural network model parameters. (2) They use costly Machine Learning (ML) techniques such as skip connections[1] [175, 177, 277], leading to added computation, memory, and storage overheads (e.g., to store the activation parameters that are fed to the last layers of the NN) [274]. Therefore, over the past decade, both industry and academia [15, 76, 90, 278–280] have explored the use of Computation-In-Memory (CIM)[2] using memristor-based devices to accelerate DNNs.

This growing interest in using CIM for resolving the shortcomings of DNNs is driven by two main factors: (1) the potential of the CIM paradigm to process data where it resides to reduce the large performance and energy overheads of data movement and (2) the analog operational properties of these nanoscale emerging technologies (e.g., memristors) that intrinsically support efficient Vector-Matrix-Multiplication (VMM), multiple of which are used to implement a Matrix-Matrix-Multiplication (MMM) that is the most dominant operation in DNNs. However, the memristor-based CIM solutions for basecalling can greatly degrade the DNN inference accuracy due to (1) the limited quantization levels supported by memristor devices [76, 90] and (2) non-idealities of memristive devices and circuits used to adopt memristor-based memory arrays, such as sneak paths [282, 283] and the non-linearity of peripheral circuitry [284–286]. To propose viable solutions for accelerating the large-scale DNN-based basecallers, these aspects must be considered at all computing stack layers, i.e., application, architecture,

[1] Skip connection is an ML technique that allows skipping a few neural network layers and forwarding the output to the input of a layer further ahead.

[2] Interchangeably, also referred to as Processing-In-Memory (PIM) [281].

and device. Such considerations are only possible with a framework capable of evaluating the impact of the non-idealities in memristor-based CIM architecture on the end-to-end basecalling accuracy. This framework should also be able to account for the overhead that the solutions to overcome the accuracy loss may bring.

To this end, we propose *Swordfish*, a modular and extensible hardware/software co-designed framework that allows us to (1) evaluate the impact of memristor non-idealities and CIM limitations on the accuracy and performance of basecalling and (2) investigate potential mitigation techniques and measure their effect on accuracy for each non-ideality (**Contribution #1**). Swordfish is used to investigate the acceleration of basecalling via emerging computing paradigms and technologies. Specifically, with Swordfish, we comprehensively investigate the potential of accurate acceleration of a SotA basecaller (Bonito) on a SotA CIM architecture (PUMA [108]) by accounting for the non-idealities of the underlying devices and technologies of the underlying architecture, for the first time (**Contribution #2**). Swordfish integrates real-world applications with multiple critical comparison metrics, distinct mitigation strategies to tackle the challenges of novel hardware, and comprehensive real measurements to guide the modeling of memristors. Our evaluations using Swordfish show that on a wide range of real genome datasets, PUMA accelerates Bonito, a SotA basecaller, by an average of 25.7× realistically (i.e., the average throughput improvement is 25.7× when we consider essential mitigation techniques to prevent huge accuracy loss). This performance still comes at the cost of a 6.01% accuracy loss (Section 3.4). Our evaluations also yield several key suggestions and recommendations for DNN, hardware, and system designers of future emerging accelerators with memristors for DNN-based basecallers and other applications that have two most important metrics (e.g., accuracy and performance) to consider in their evaluation (**Contribution #3**). Specifically, our investigation using Swordfish results in multiple unique insights: (1) Our results challenge the prevalent assumption that DNN-based applications will automatically succeed on memristor-based CIM due to inherent redundancy in large neural networks, (2) combining mitigation techniques at only one abstraction level (e.g., circuit or system level) does not necessarily improve the accuracy loss as they can potentially go against each other, and (3) combining multiple mitigation techniques at the circuit and system levels can offset the accuracy loss induced by non-idealities significantly.

3.1. BACKGROUND AND MOTIVATION

This section briefly discusses the necessary background and motivation for this work. We refer the reader to comprehensive reviews [168, 273, 281, 287, 288] for more details.

3.1.1. GENOME SEQUENCING PIPELINE

The genome sequencing pipeline consists of computational steps we employ to acquire genome sequences as strings of DNA characters (i.e., {A, C, G, T}) [162–168, 272, 273] for subsequent analysis in bioinformatics, e.g., cell type identification, identification of marker genes, and variant detection.

Although, currently, the most available data and tools in the genomics realm are for short reads [231, 234] (mainly produced by Illumina sequencers), working with highly ac-

curate long genome sequences is generally favorable as they reduce the computational cost of reconstructing the genome. For this reason, there is a large momentum towards accurate long-read sequencing [273]. Our work focuses on finding solutions and analysis tools that target long reads while also not discarding tools (e.g., GenAx [234] and GenASM [231]), designed for short reads. A leading method for long-read sequencing is the nanopore sequencing technology. Nanopore sequencers [289–291] translate raw signal squiggles into bases (A, C, G, T) using complex neural networks. Today, Oxford Nanopore Technologies (ONT) is a company that produces the most commonly used sequencers based on Nanopore technology.

Fig. 3.1 illustrates the nanopore genome sequencing pipeline [272] and the placement and execution time breakdown of each of its steps. We use SotA tools for each step and run the tool on the datasets described in Section 3.3.

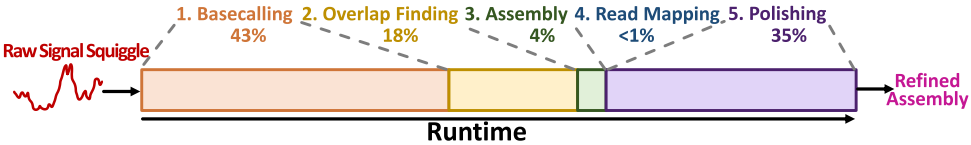

Figure 3.1: Overview of the nanopore genome sequencing pipeline and execution time breakdown of different steps.

We make two main observations. First, basecalling is the first computational step in the pipeline. Second, basecalling dominates the execution time of a single run in the pipeline. These steps make up more than 40% of the entire execution time. Our empirical observation aligns with those in prior works [179, 272, 292].

3.1.2. BASECALLING

Basecalling is responsible for converting raw electrical signals produced by a nanopore sequencer to digital genome symbols, i.e., [A, C, G, T] [162–165]. Recent works [169, 173–175] heavily investigate the use of DNNs for basecalling as they can provide high accuracy than Hidden Markov Model (HMM) based techniques [176].

There are generally two approaches for improving the accuracy and/or performance of a basecaller: 1) software-based and 2) hardware-based. Software-based methods propose new algorithms (e.g., DNNs [174, 175, 177] instead of HMMs [176]) or faster and/or smaller DNN architectures [177, 274]. Hardware-based approaches propose various hardware platforms for the target algorithm (i.e., DNN or HMM) to improve performance with (hopefully) small impact on accuracy [179, 274].

We observe four main shortcomings in SotA basecallers, which limit their execution time and/or hardware acceleration:

- SotA basecallers are slow and energy inefficient. For example, Guppy basecalls 3 Giga basepairs (Gbps) in ~6 hours while a following step in the genomics pipeline, such as read mapping using minimap2 [293] takes only ~0.11 hours [274].
- SotA basecallers use DNN models with costly skip connections [277]. For example, Bonito needs an additional ~21% of model parameters (along with associated memory

and storage overheads) for skip connections and requires additional computation on them. Note that a skip connection permits bypassing certain layers within the neural network, transmitting the output of one layer as the input to subsequent layers [277]. These connections are costly because they (1) typically force the network to perform additional computation, for example, to match the channel sizes, (2) incur extra memory and storage overhead, as they require storing the activation parameters that are fed to the later layers [42, 276], and (3) incur additional off-chip data movement overhead when these networks are run on conventional processor-centric hardware platforms, like CPUs and GPUs.

- SotA basecallers exploit 32-bit floating point precision for their model parameters [169, 175, 177]. This effectively increases (1) the required bandwidth and processing units, e.g., with FP compute capability, and (2) inefficiency in the hardware realization of the underlying models.

- SotA basecallers incur expensive data movement between the computation units and the memory units [179, 218, 274].

We emphasize that 40% of execution time spent on basecalling (Section 3.1.1), the first and arguably most critical step in the pipeline, is significant and worth accelerating. Today's best basecallers often underperform on SotA systems, generating bottlenecks. A potentially 40% decrease in genome analysis runtime implies a proportional reduction in power and energy, which is critical considering the extensive data and computational demands of modern genome analysis systems. Therefore, optimizing basecalling contributes greatly to improving the efficiency and sustainability of the genomics pipeline.

3.1.3. MEMRISTOR-BASED CIM AND ASSOCIATED NON-IDEALITIES

Resistive memories or memristive devices, such as ReRAM, PCM, and STT-MRAM [279, 294–296], have recently been introduced as suitable candidates for both storage and computation units that can efficiently perform vector-matrix multiplication [297] and logical bulk bit-wise operations [87, 106, 117, 118, 298], as they can follow Kirchhoff's law inherently [299]. Therefore, many recent works [76, 90, 106, 108, 114, 122, 298, 300, 301] exploit these devices in their CIM architectures. Memristor devices also enjoy non-volatility, high-density, and near-zero standby power [87, 106, 279].

A typical memristor-based memory crossbar capable of VMM and other logical operations is shown in Fig. 3.2 [76, 90, 106, 108, 298] alongside its possible non-idealities.

This memristor-based structure can suffer from at least four types of non-idealities or variations that can eventually affect the results of the enabled VMM operation, i.e., lead to errors in the VMM result: (1) The non-ideal digital to analog converter (DAC), due to the effective resistive load (known as R_{Load}) in its circuit [139], (2) Variation of synaptic conductance, which includes both imperfect programming operation (commonly known as write variations) and the process variation that exist in memristors [141, 302–304], (3) The wire resistance and sneak paths, due to imperfect wires (i.e., wires with different resistances) and the changes in the voltages of the internal nodes while performing a VMM operation [140, 141], and (4) non-ideal sensing circuit or analog to digital converters (ADCs), due to rigid or hard-to-accurately-change references used for

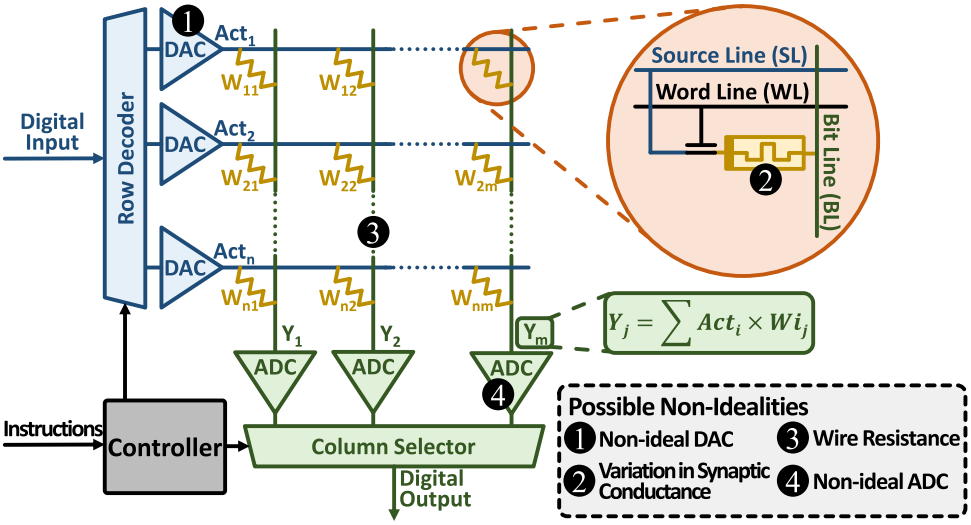

Figure 3.2: Overview of memristor-based crossbar arrays and possible non-idealities.

distinguishing/sensing the end result [139, 300]. Our work focuses on these specific non-idealities inherent to memristor technologies in a CIM architecture. While we do not explicitly address other circuit challenges and non-idealities, we acknowledge their presence and the existing solutions developed to mitigate them in electronic systems. For example, crosstalk [305–307], which involves interference between adjacent circuit traces or wires, can indeed lead to data corruption and compromise information integrity. However, we focus on the specific non-idealities relevant to our hardware architecture, not crosstalk. Note that industry-standard techniques, such as shielding and layout design, decoupling components, ground and power distribution, signal timing and margins, ECC and scrubbing, isolation and shielding, and crosstalk-aware clock distribution, have been extensively studied and developed to mitigate crosstalk issues. We assume that similar techniques can be applied to address any potential crosstalk concerns in memristor-based CIM systems.

Recent works [76, 90, 108, 308, 309] report impressive performance and energy improvements for DNN models executed on memristor-based CIM architectures, mainly assuming idealized underlying hardware. Moreover, DNNs are known to be resilient to some noise [310–315]. However, since memristor-based CIM architectures are indeed non-ideal and the resiliency of DNNs has a limit, to decide whether or not these platforms are indeed suitable for realizing our DNN-based basecaller, one needs to evaluate the impact of these non-idealities on the end-to-end application accuracy and account for the overhead that the solutions to overcome the accuracy loss may bring. Such a framework is missing among prior works and is a contribution of our work (Section 3.2).

3.1.4. PROGRAMMABLE INFERENCE ARCHITECTURE

We briefly discussed PUMA[3] in Section 2.3.1. PUMA (Programmable Ultra-efficient Memristor-based Accelerator) [108, 111, 226] is a complete set of (micro)architecture, simulator, and compiler that supports the execution of many ML applications, using memristor crossbars enhanced with general-purpose execution units. PUMA uses a spatial architecture and provides the necessary programmability and generality to execute a wide range of ML-based applications on memristor-based crossbars. For evaluations in Swordfish, we assume an PUMA-based architecture for two reasons. First, PUMA supports all the necessary types of NN layers in basecallers: CNN, LSTM, and linear. This is especially handy for our main target basecaller, Bonito. Second, the architecture, simulator, and compiler are open-sourced [111, 226] and well-documented for an extension, unlike many other rich architectures.

3.2. SWORDFISH FRAMEWORK

3.2.1. SWORDFISH OVERVIEW

Fig. 3.3 presents an overview of the Swordfish framework. Swordfish consists of 4 key modules:

- ❶ *Partition & Map* module that partitions and maps the Vector-Matrix-Multiplication (VMM) operations of the target DNN-based basecaller to the underlying CIM platform,
- ❷ *VMM Model Generator* module that generates an end-to-end model for possible non-idealities and errors of a VMM operation considering the underlying technology in the CIM design,
- ❸ *Accuracy Enhancer* module that implements online and offline mitigation techniques to counter accuracy loss, and
- ❹ *System Evaluator* module that analyzes the accuracy and throughput of basecaller while also providing an area overhead.

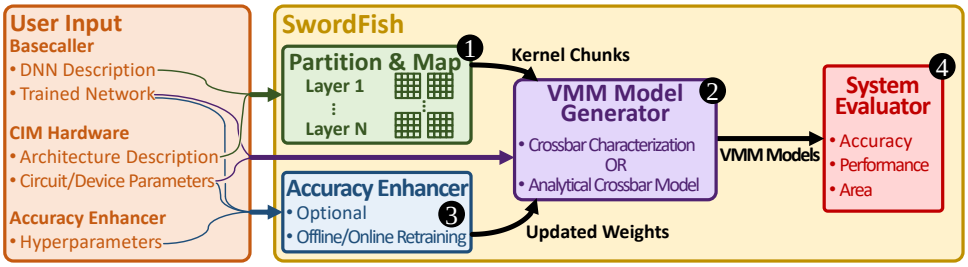

Figure 3.3: Overview of Swordfish framework.

[3] Programmable Ultra-efficient Memristor-based Accelerator.

We emphasize that the accuracy analysis in the System Evaluator module is critical and unlike evaluations of conventional platforms, e.g., field-programmable gate arrays (FPGAs) or GPUs. Its importance stems from the abundance of the underlying non-idealities, variations, limitations, and hardware perturbations of the emerging hardware paradigms [316]. From now on, we refer to the proposed framework as *Swordfish* and the actual implemented memristor-based CIM design for our target basecaller Bonito as *SwordfishAccel*.

3.2.2. PARTITION & MAP

To run the DNN of a basecaller on a CIM architecture, one should map each of the VMM operations in the target DNN to the analog memory arrays and the rest of the operations to the digital peripheral circuitry. The Partition & Map module takes care of this task in Swordfish by dividing individual functions of the basecaller into the analog or digital components of the underlying architecture. This process is required one time for every basecaller and has two steps.

In the first step, Swordfish decides which memory crossbars will perform each VMM operation of each layer. For Bonito basecaller, Swordfish decides which memory crossbars handle the VMM of the first convolutional layer and which crossbars are responsible for the VMMs of the following LSTM and linear layers. Swordfish assumes that all the underlying crossbars have the same size and readout peripheral circuitry (e.g., ADCs).

In the second step, Swordfish decides how it maps the weights to each crossbar. Swordfish supports different programming/writing techniques for memristor devices, such as write-read-verify (WRV) and Set/Reset pulse programming.

In mapping and evaluation, Swordfish makes the following widely common design choices:

- The input streams into the first layer of DNN. Swordfish does not divide the input into chunks and leaves this task to the host. Doing so helps Swordfish to evaluate the maximum throughput of a basecaller [175, 179], independently of the input size.

- The next layer starts its computation as soon as the previous layer of the basecaller produces enough values. This is also a common assumption for evaluating the maximum possible throughput of a DNN in simulation [90, 108].

- Multiple crossbar arrays can be simultaneously active and perform the necessary operations (VMM and other operations necessary for the target DNN, such as activation. This assumption ensures that full chip utilization is not limited due to power constraints. One can consider this parallelism to be analogous to the concurrent activation of multiple subarrays in different banks and bank groups in traditional DRAM [91–93].

- Swordfish optimizes its design decisions for the highest achievable accuracy, throughput, and memory utilization in the stated order. This is a common priority order for optimizations in basecallers [175, 179, 317].

3.2.3. VMM MODEL GENERATOR

VMM Model Generator is responsible for generating the non-ideal output per each VMM required by the basecaller. VMM Model Generator differentiates between constraints and non-idealities. This is essential in a CIM design where non-idealities or constraints do not necessarily lead to a loss in the accuracy of the application. To model the effect of these constraints and non-idealities on the accuracy of an application, Swordfish considers them at the lowest-level building block where they aggregate, i.e., where their results merge. In a memristor-based CIM architecture for a DNN-based basecaller, such an effective place to consider the effects of constraints and non-idealities is the VMM operation output. Therefore, the VMM Model Generator in Swordfish focuses on assessing the effects of each factor on a VMM operation, while our evaluations and analyses assess the end-to-end basecalling metric.

This module takes three types of inputs. First, it takes the results of the previous module (i.e., ❶ Partition & Map in Fig. 3.3) to determine the size of the VMM. Second, it takes the circuit and device description (i.e., constraints and non-idealities) that can affect accuracy. Examples inputs in this category are (1) the level of quantization, (2) the circuit variations (e.g., in inputs (e.g., DACs), wires, and outputs (e.g., ADCs) device), and (3) device variations. Third, it takes the weights of the target basecaller, which can be provided directly by the user or the Accuracy Enhancer module that applies multiple training mechanisms (Section 3.2.4). The module outputs the non-ideal output vector per each input vector and weight matrix (i.e., the expected vector result for a VMM).

Swordfish supports two different approaches for modeling a VMM. The first approach is to use a pre-calculated library of measurements on actual devices. The second approach is to use an analytical model (e.g., a fast crossbar model (FCM) [139]). Section 3.4 evaluates these approaches separately.

In the first approach, Swordfish queries a library that, for a given array size and input vector, returns an output vector randomly chosen from many ($\geq 10^4$) possible outputs based on measurements on an actual crossbar with the same dimensions as the length of the active input vector. The measurements in the library already contain all the possible non-idealities in the target VMM operation, i.e., non-idealities that may arise from DACs, ADCs, circuits, and devices in the crossbar. One can build this library by measuring multiple tiles several times. For each of these measurements, one should program the initial values of memristors within a tile with the weight values of the target DNN to be evaluated on Swordfish. In this paper, the distinct initial resistance states are based on the Bonito basecaller [175]. The random choice from the library aims to account for variations and non-idealities among different memristor-based tiles, which can arise from different initial values of each memristor device and/or manufacturing differences. By integrating real measurements and accounting for tile-to-tile differences, we believe our methods accurately reflect non-ideality distribution in practical settings. Although this approach accurately represents the VMM operation considering many possible non-idealities, it lacks the flexibility of separately studying or measuring the effects of each possible error due to different non-idealities. This approach is also limited to the crossbar configurations (for example, crossbars of 64×64 and 256×256) to whose measurements one has access (Section 3.3).

In the second approach, Swordfish utilizes existing analytical models that are avail-

able for ADCs, DACs, and variation profiles of the underlying devices in the crossbar. Fig. 3.4 illustrates the steps Swordfish uses in its VMM Model Generator for this approach.

Figure 3.4: An overview of the VMM Model Generator's second approach: using analytical models.

In Fig. 3.4, Swordfish applies the analytical model for a non-ideal DAC model (❶) to the input vector of the VMM operation (❶) and obtains the non-ideal input voltages as the output vector (❷). Swordfish then applies this new vector to a crossbar with an updated non-ideal weight matrix (❷), where non-idealities have been applied to the original weight matrix (from the VMM operation) based on the expected variations of each cell, which are usually obtained based on generic characterization of memristor-based crossbar arrays, i.e., without any peripheral circuitry or target weights specific to a particular DNN. The output is considered a non-ideal output current (❸) that Swordfish applies to a model of non-ideal ADC (❸) and obtains the output vector (❹), an output vector that might contain some errors.

Fig. 3.5 presents an overview of how Swordfish models the crossbar non-idealities for the second approach (i.e., the analytical model in the VMM Model Generator module) (❷ in Fig. 3.4). For this, Swordfish first takes the crossbar instances (① in Fig. 3.5) from the Partition & Map module. Swordfish considers these crossbar instances as separate matrices with digital weights (②). Then, Swordfish uses a non-linear model for the synaptic device states (③) to map the weight matrices of digital weights into ideal corresponding conductance matrices (④). After that, Swordfish applies to these metrics the synaptic variations for the crossbar (⑤) that are determined from an analytical model based on the estimated behavior of memristor devices within a crossbar array. The output consists of the same number of matrices, but now with adjusted weights (⑥). Swordfish finally applies to those matrices the profile of all known circuit-level non-idealities (⑦) by adding representative metrics for these non-idealities. The output consists of matrices accounting for all variations and non-idealities (⑧).

3.2.4. Accuracy Enhancer

Since accuracy is a critical metric in basecalling, Swordfish applies several mitigation techniques to deal with the non-idealities and their induced errors on the VMM and/or basecalling. More specifically, Swordfish supports four different accuracy enhancement techniques: (1) analytical variation-aware training (VAT) (offline), (2) knowledge distillation (KD) training, (3) read–verify–write (R-V-W) training, and (4) random sparse adaptation (RSA) retraining (online).

Analytical Variation-Aware Offline Training

Swordfish supports variation-aware training (VAT) [318–321] during the training of a target DNN as the simplest method to enhance the accuracy loss due to (1) quantiza-

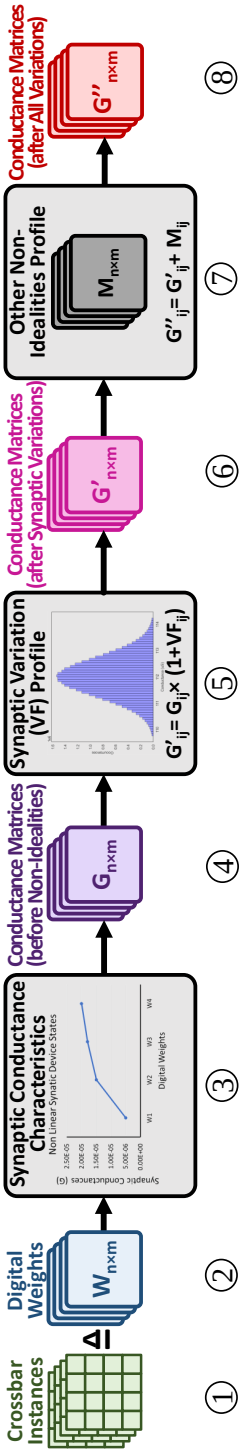

Figure 3.5: An overview of modeling crossbar non-idealities in Swordfish.

tion and (2) possible resistance variations per weight, which can be analytically or experimentally measured. Existing works randomly inject faults into the weights of the DNN [322], or model the potential errors at the end of each layer [318, 322]. Similarly, Swordfish utilizes the crossbar characterization for the errors per VMM (i.e., the error library in the first approach in VMM Model Generator) or an analytical crossbar model for the errors per VMM (i.e., as in the second approach in VMM Model Generator). Swordfish injects the modeled errors in the training and considers the rest of the devices unaltered. Swordfish repeats this process for each VMM and every layer and then retrains the basecaller network. This way, Swordfish ensures that its retraining yields a better estimate for the errors arising from non-idealities in the crossbar.

KNOWLEDGE DISTILLATION-BASED VARIATION-AWARE TRAINING

In addition to offline VAT based on injecting random errors or potential errors per layer discussed in Section 3.2.4, Swordfish is capable of supporting the knowledge distillation (KD) approach as a VAT as well, i.e., Swordfish exploits knowledge/weights that exist in an ideal (typically a FP32-based) basecaller baseline to guide the training of SwordfishAccel, our memristor-based CIM design for Bonito. In KD, two models exist: (1) the teacher (an ideal implementation using high precision data format, e.g., FP32-bit format) and (2) the student (SwordfishAccel quantized to 16-bit-width fixed-point presentation for both weights and activations). The goal is to mimic the teacher's output in the student by minimizing a loss function where the target is the result of applying the softmax on the quantile function associated with the standard logistic distribution (i.e., logit) of the teacher's training [323]. We refer the reader to previous works on KD [323, 324] for further detail on how a loss function can be implemented in such a system to minimize the difference of SwordfishAccel's output and the teacher model's softmax output.

READ-VERIFY-WRITE (R-V-W) TRAINING

Read-Verify-Write (R-V-W) is a conventional error mitigation technique for non-ideal memristor-based memories that provides cell-by-cell error compensation. R-V-W is used in open-loop-off-device (OLD) [325] where R-V-W programming and sensing loop help the actual resistance of the device to converge to the expected target resistance. This method involves many read-and-write operations and feedback control for memristors, making R-V-W a slow technique to mitigate accuracy loss. Note that to improve the accuracy in R-V-W, we need to increase the fraction of the retrained weights (memristor devices in our case), increasing the cost of the mitigation technique.

RANDOM SPARSE ADAPTATION ONLINE RETRAINING

Swordfish uses random sparse adaptation (RSA) [324] to map the learned DNN model to SwordfishAccel. RSA is used to mitigate the performance overhead of R-V-W technique [325, 326]. RSA by itself prevents only some of the non-idealities from being materialized as inaccuracies and can be an offline mechanism. However, SwordfishAccel combines it with an online training mechanism.

For its online retraining using RSA, Swordfish places a small on-chip SRAM-based memory next to memristor-based crossbars and distributes the learned DNN model (i.e., weights) between this SRAM and memristor-based crossbars. The key idea Swordfish uses is to map the weights that otherwise would map to error-prone memristor

devices to reliable SRAM cells. If one has access to the exact profile of the underlying memristor-based memory crossbars, one can exploit the knowledge on which memristors and columns are more error-prone and use this knowledge to decide which weight to map into the crossbar and which one to the SRAM. In our evaluations of Swordfish, we use this knowledge whenever we use the chip measurements already used in the first approach of the VMM Model Generator. However, Swordfish can also randomly choose memristor devices in the crossbar and map (i.e., hardwire) them to the SRAM. Random choice is the next best option without knowledge about the exact error pattern of a memristor-based crossbar. We used this method whenever we used the second approach (i.e., analytical model) in the VMM Model Generator (Section 3.2.3).

Fig. 3.6 presents how SwordfishAccel adopts RSA with an online retraining mechanism (e.g., KD) in a three-step approach:

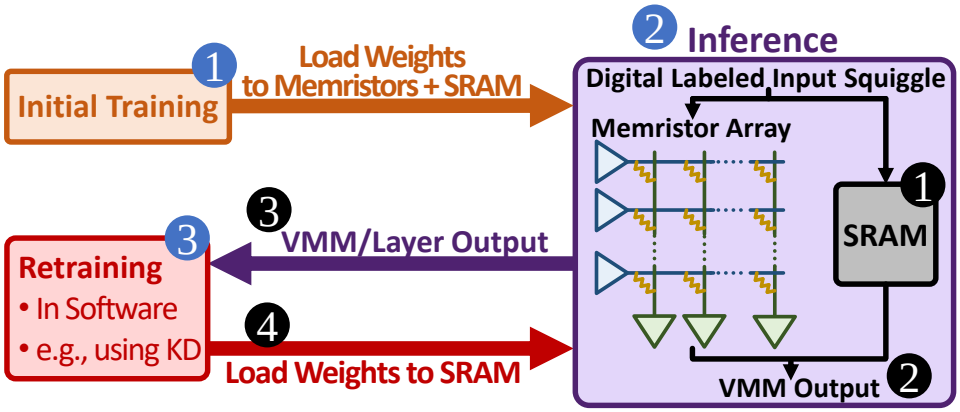

Figure 3.6: Swordfish's online error mitigation via RSA.

1. In the first step (❶), SwordfishAccel trains the original Bonito and loads the initial weights from the Bonito DNN model into the assigned memristor crossbar and the SRAM (❶). SwordfishAccel considers this model as the initial model for the student in KD.

2. In the second step (❷), SwordfishAccel performs a VMM operation as usual. However, whenever one or more of the assigned weights to SRAM (i.e., error-prone memristors or randomly chosen ones in Swordfish) is involved, SwordfishAccel reads the value from the SRAM memory instead of the memristor device. Swordfish does this by passing the inputs of corresponding devices through the SRAM value instead of the crossbar, zeroing the input for that particular memristor in the crossbar, and then summing up the values of both paths (❷).

3. In the third step (❸), SwordfishAccel returns the results of the VMM operation of each crossbar (❸) to the retraining component (KD in our example in Fig. 3.6) and performs online training on only the weights that are mapped to the SRAM memory to improve the accuracy loss due to non-idealities. Note that SwordfishAccel considers

the non-ideality models of crossbars, ADCs, and DACs to the student model for every training batch and trains the student. This includes both the initial training in Step ❶ and retraining in Step ❸.

4. SwordfishAccel then loads the new weights to the SRAM near the crossbars (❹) and repeats Steps ❷ and ❸.

SwordfishAccel uses KD-based variation aware training for its Step ❸ in Fig. 3.6 online retraining. However, any other retraining method can also replace KD in our example. Note that all the parameters are already quantized to 16-bit fixed-point precision to present the model in SwordfishAccel accurately. Swordfish leverages the weights from the converged teacher model to improve the convergence of the student model.

RSA in Swordfish comes at the price of extra area overhead for the considered on-chip SRAM memory, storage in the memory controller for mapping metadata, summation of the output from the crossbar with on-chip memory, and some additional control logic evaluated in Section 3.4.

3.2.5. SYSTEM EVALUATOR

The System Evaluator module puts the results of all previous modules of Swordfish together to evaluate the target DNN.

As inputs, this module takes the execution time for each VMM operation, the accuracy of each VMM operation for the last layer of the DNN (as it determines the final accuracy of the DNN), the number of active crossbars in each step of Swordfish, and information in peripheral circuitry.

The System Evaluator module has 3 outputs:

1. **Accuracy:** The System Evaluator module outputs an accuracy number for the evaluated DNN. In SwordfishAccel, this number shows the accuracy of the basecaller, commonly known as *read accuracy*, which is the fraction of the total number of exactly matching bases of a read to a reference to the length of their alignment (including insertions and deletions).

2. **Basecalling throughput:** The System Evaluator module outputs a number for inference throughput of the target DNN. In SwordfishAccel, this number is the basecalling throughput, defined as kilo-basepairs generated by the basecaller per second ($\frac{Kbp}{s}$). The higher the basecalling throughput, the better. This is the most important metric to evaluate a basecalling accelerator's performance. Our throughput evaluations in SwordfishAccel include the time required for read and write time for the inputs and outputs, respectively.[4]

3. **Area overhead.** The System Evaluator module of Swordfish also reports area overhead based on the underlying architecture to account for the overheads of a dedicated accelerator, e.g., SwordfishAccel.

[4] We use this command line in Linux: /usr/bin/time -v.

3.2.6. SWORDFISH EVALUATION CHALLENGES

Comprehensive, fair, and practical evaluation of Swordfish is challenging for two main reasons. First, most of the SotA basecallers are either not open-source [169, 179, 327] or support only specific reads [317]. Second, current simulators and frameworks mimicking memristor-based CIM designs are either not open-source, do not consider the underlying non-idealities of the devices, or only support a very limited number of non-idealities, emerging technologies, or neural networks [139, 328].

To evaluate Swordfish despite these challenges, we take two representative examples. Specifically, for the first challenge, we primarily compare our method with Bonito [175], an open-sourced, universally applicable tool currently under active development and maintenance by ONT (Section 3.1.1). Bonito stands out for its exceptional accuracy and performance over its predecessors like Guppy [169] and does not face the limited support for reads (e.g., Dorado [317]) or lack of open-source implementation and training code (e.g., Helix [179], Halcyon [329], Guppy [169], and SACall [327]). For the second challenge, we consider PUMA architecture (Section 2.3.1) as the baseline architecture for the two reasons mentioned in Section 3.1.4.

3.3. EVALUATION METHODOLOGY

This section discusses our experimental methodology.

3.3.1. IMPLEMENTATIONS AND MODELS

For the performance and area studies, we significantly extended the PUMA simulator and PUMA compiler to account for (1) Bonito's DNN architecture, (2) updated configurations in Core Architecture of PUMA [108] based on our memory models and the TSMC 40 nm [330] technology node used for peripheries, and (3) performance and area overheads introduced by non-idealities of memristors and their mitigation techniques. Note that we use Synopsys Design Compiler [331] and synthesize the additional components of our design in the target technology to obtain their execution time, power, and area. We apply the prominent technology scaling rules [332] to the configuration numbers of the PUMA architecture to ensure all of our design components are based on the same technology node.

For accuracy analysis (in both training and inference phases), we also extensively modified Bonito's open-source implementation [175] to consider the device characteristics and limitations of the architecture. Unfortunately, PUMA does not allow us for such analysis as it considers the effects of only quantization and write variations on accuracy.

We utilize prototyped cross-array memristors as our memory arrays and capture the variations in their spatiotemporal conductivity, execution time, and area overhead of necessary operations. We project our characterization results of real memories to our DNN evaluations. We also build a statistical model from our measurements to capture the full picture of a larger memory model for large-scale variations, timing, and area parameters. This model contains four types of variations: (1) input DACs, (2) synaptic variations, (3) wire resistance, and (4) output ADCs. The memory prototypes and models used for evaluations and simulations are based on the results of the EU project MNEMOSENE [333], concluded in 2020, generously provided by the involved parties.

The results have been tested heavily during the project and by various metrics found in the related literature. Table 3.1 shows the main parameters of our memristor-based crossbars.

Technology and device	ReRAM HfO_2/TiO_x [330]
Cell configuration	1T1R (NMOS T: 460 nm/40 nm
HRS/LRS	1 MΩ/10 kΩ
$\mathbf{n}_{min}/\mathbf{n}_{max}$	0.03, 30
Array Sizes	64×64 and 256×256
SA $\mathbf{V}_{min}$	40 mV

Table 3.1: Our array and device configurations.

Our study specifically evaluates Swordfish on ReRAM memristors for three reasons. First, the availability of actual chip measurements is essential for our non-ideality-centered study. Second, lower energy costs for writing/programming than alternatives like PCM. Third, ReRAM's established status within the memristor family provides reliable baselines and intuitions for device-level features, enhancing the credibility of our proposal.

3.3.2. Simulation Infrastructure

We ran our baseline Bonito basecaller and software implementation of Swordfish on a 128-core server with AMD EPYC 7742 CPUs [334], 500GB of DDR4 DRAM, and 8 NVIDIA V100 [335] cards. We train and evaluate Swordfish accuracy and software results on our NVIDIA cards (with 32-bit floating-point precision). We use the nvprof profiler [336] for the profiling experiments on GPU.

3.3.3. Evaluation Metrics

We use metrics output by the System Evaluator module for our comparisons. Section 3.2.5 clarifies these metrics.

3.3.4. Datasets and Workloads

Table 3.2 provides datasets from a MinION R9.4.1 flowcell [337, 338] we use in our evaluations.

Dataset (Organism)		# Reads	Reference Genome Size (bp)
D1	Acinetobacter pittii 16-377-0801	4,467	3,814,719
D2	Haemophilus haemolyticus M1C132_1	8,669	2,042,591
D3	Klebsiella pneumoniae NUH29	11,047	5,134,281
D4	Klebsiella pneumoniae INF042	11,278	5,337,491

Table 3.2: Read and Reference Datasets for our Basecalling Evaluation.

3.4. SWORDFISH EVALUATION

We first use Swordfish to investigate the impact of constraints and non-idealities of a PUMA-based architecture (Section 3.1.3) on the accuracy of the Bonito basecaller [175]. We call this design the Ideal-SwordfishAccel, as it achieves the highest performance for our memristor-based hardware accelerator without any accuracy enhancement technique. We then explore the effect of the accuracy enhancement mechanisms in Swordfish applied to deal with the inaccuracies of the memristor-based accelerator as it affects the Bonito basecaller's accuracy. The results of this design are presented under Realistic-SwordfishAccel.

3.4.1. EFFECT OF QUANTIZATION ON ACCURACY WITHOUT ACCURACY ENHANCEMENT

Since both the weights and activations in the original DNN are in FP32 format, Swordfish can opt for quantizing one or both of them. The degree of the quantization can differ depending on how much each parameter impacts the overall accuracy. Swordfish considers seven different configurations: the default configuration (DFP 32-32), where weights and activations use the FP32[5] format, and 6 FPP X-Y[6] formats, where X and Y denote the fixed-point precision of weights and activations, respectively. Swordfish currently only supports power-of-two precision levels for its quantized configurations. Table 3.3 presents the accuracy of different configurations.

	DFP 32-32	FPP 16-16	FPP 8-8	FPP 8-4	FPP 4-8	FPP 4-4	FPP 4-2
D1	97.32%	97.32%	97.12%	97.12%	95.42%	95.62%	93.62%
D2	97.32%	97.32%	96.72%	96.72%	94.92%	95.42%	92.42%
D3	97.32%	97.32%	96.02%	95.82%	93.62%	95.12%	93.72%
D4	97.32%	97.32%	96.42%	96.42%	94.22%	95.32%	93.62%

Table 3.3: Accuracy evaluation after quantization.

We make two major observations. First, Bonito's architecture can tolerate some quantization level without accuracy loss. More specifically, across all evaluated datasets, quantization down to 16 bits does not affect the accuracy at all, and quantization down to 8 bits reduces the accuracy by less than 9% even in extreme cases. We conclude that Ideal-SwordfishAccel can still reduce the precision of its network from a 32-bit FP format to 16-bit-width fixed point precision without accuracy loss. This way, Ideal-SwordfishAccel can (1) accelerate the network on a platform limited to fixed point format representation and (2) improve the energy efficiency of the network via lower data precision. This observation is on par with similar studies [90, 274, 339] exploiting quantization as a technique to improve the performance and energy efficiency of a DNN with a negligible accuracy loss.

Second, tolerance to quantization varies depending on the input dataset. This makes the effect of quantization on accuracy workload-dependent. However, the accuracy drop for different quantization configurations follows more-or-less a similar trend irrespective of the dataset, i.e., they all follow a decreasing trend with reduced data representa-

[5] FP stands for floating point.
[6] FPP stands for fixed point precision.

tion. We conclude that Swordfish's understudy network (Bonito) tolerates some quantization but will offer very low accuracy for extreme quantization (i.e., lower than 4-bit precision) irrespective of the dataset. We note that an accuracy drop of ~5% and higher is considered unacceptable for a future basecaller, as accuracy is the most critical metric in SotA basecallers. This observation is consistent with prior works on smaller [139] or different types of networks [274].

We conclude that quantization is a viable solution to tackle data representation constraints in hardware accelerators and, therefore, can be used in a framework such as Swordfish. However, accuracy loss due to quantization (applied with the expectance of accuracy loss due to variations and non-idealities) leads us to consider only down to 16 (or possibly 8) bits of precision for both weights and activations before a significant accuracy drop occurs. Therefore, the following studies consider only a 16-bit integer as the quantization level.

3.4.2. Effect of Non-idealities on Accuracy without Accuracy Enhancement

We examine the effect of four non-idealities on basecalling accuracy. The results presented in this section belong to the second approach of modeling non-idealities in the VMM Model Generator module, i.e., using analytical modeling (see Section 3.2.3).

Effect of Write Variation on Accuracy

Write variation can single-handedly impact the accuracy results of a VMM operation [324, 339]. Therefore, we analyze it separately.

Fig. 3.7 presents the effects of write variations on accuracy. The x-axis sweeps the write variation rate. The error bars account for the accuracy variations on different write variation rates over 1000 runs of the model. Since the models for write variation are circuit-dependent and have varying probabilities of affecting the stored/programmed data, this methodology provides us with a better insight into the effect of this non-ideality on accuracy.

We make two main observations. First, slight write variation can lead to a significant drop in the accuracy of end-to-end basecalling. To a great extent, this is on par with previous works' observation of the write variation impact on VMM accuracy [324, 339]. For example, the accuracy drops vary from 3.30% to 87.34% for D1 and from 3.24% to 85.76% for D4.

Second, the exact accuracy loss depends on the input dataset, i.e., the accuracy is workload-dependent and varies for the same write variation among different subfigures in Fig. 3.7. For example, for the same write variation rate of 25%, the accuracy on our two datasets (i.e., D2 and D4) can vary by 0.93%.

We conclude that write variation in Ideal-SwordfishAccel can debilitate the basecalling process significantly. In other words, write variation can eliminate all the potential performance and energy efficiency benefits of such a memristor-based design if not mitigated correctly. Therefore, unlike the quantization constraint, we should closely control the write variations in any future design for an acceptable basecaller. Fortunately, some previous works [324, 340, 341] propose mitigation techniques that, when combined, can provide us with reasonable (e.g., amount of ≤ 10%) write variation. From now on, we

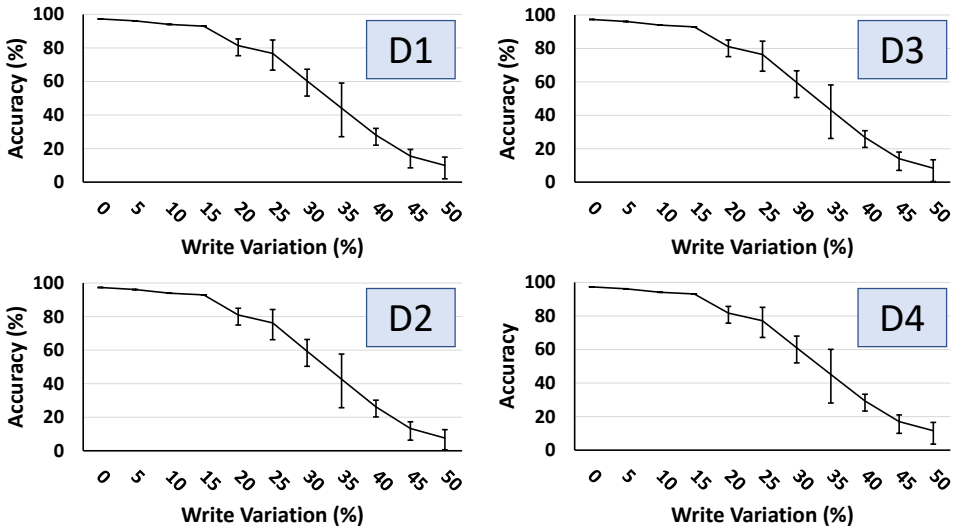

Figure 3.7: Accuracy after taking into account write variation.

consider only up to 10% write variation (as defined in Section 3.1.3) in our evaluations.

EFFECT OF COMBINED NON-IDEALITIES ON ACCURACY

Fig. 3.8 and Fig. 3.9 show the accuracy after considering all other sources of non-idealities (see Section 3.1.3) for our four datasets on two different crossbar sizes of 64×64 and 256×256, respectively. The error bars show the distribution when considering 10% write variation over 1000 runs. For each dataset, Fig. 3.8 and Fig. 3.9 present the accuracy results for five configurations presented as individual bars in the figures. The first three bars from the left present the results for individual non-idealities, i.e., synaptic+wire resistances (*Synaptic+Wires*), sensing+ADC circuitry (*Sense+ADC*), and DAC+driver circuitry (*DAC+Driver*), respectively, that Swordfish accounts for in its second approach of modeling non-idealities in the VMM Model Generator module, i.e., using analytical modeling (Section 3.2.3). The fourth bar, *Combined*, accounts for all the non-idealities from the same analytical model simultaneously. The fifth and last bar, *Measured*, considers all the non-idealities from the library of real chip measurements in the first approach of modeling non-idealities in the VMM Model Generator (see Section 3.2.3).[7] We make six main observations.

1. A combination of non-idealities (i.e., each of the bars labeled with "Combined" or "Measured" or the 4th and the 5th bar per dataset in Fig. 3.8 and Fig. 3.9) leads to a significant accuracy loss irrespective of the dataset or crossbar size. For example, observe the accuracy loss when considering all the non-idealities in an analytical way (bars labeled as "Combined"). The accuracy loss varies from 18.32% to 31.32% (① in

[7] We leave the exploration of every possible combination of individual non-idealities to future work.

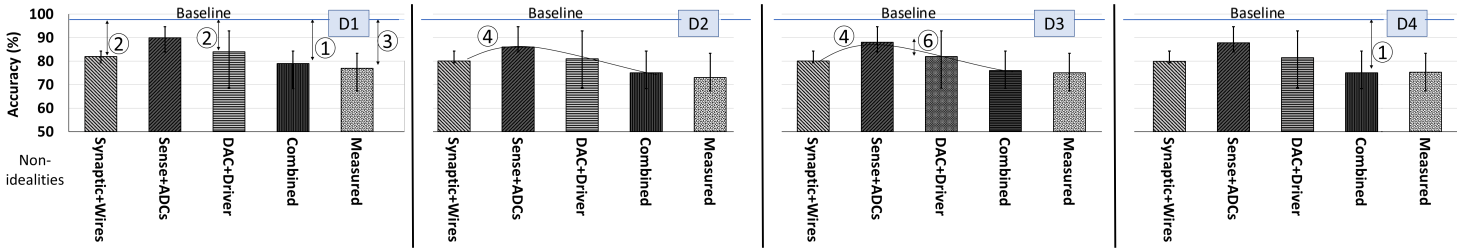

Figure 3.8: Accuracy after taking into account non-idealities on 64×64 crossbars for the 4 datasets.

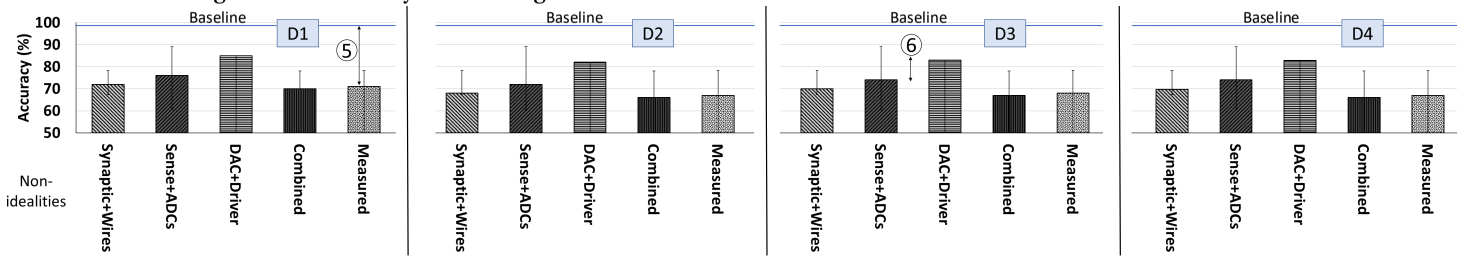

Figure 3.9: Accuracy after taking into account non-idealities on 256×256 crossbars for the 4 datasets.

Fig. 3.8) across different datasets (i.e., D1 to D4). The same trend can be observed in Fig. 3.9.

2. The impact of individual non-idealities (i.e., *Synaptic+Wires*, *Sense+ADC*, or *DAC+Driver*) on the accuracy (loss) is different. For example, observe the accuracy loss of *DAC+Driver* versus *Synaptic+Wires* in D1 (② in Fig. 3.8). For the same dataset, the accuracy loss varies from 13.32% for *DAC+Driver* to 15.34% for *Synaptic+Wires*. A similar difference also exists in crossbars of size 256×256 in Fig. 3.9.

3. The accuracy loss for combined non-idealities is non-additive. For example, in D1, the total accuracy loss of *Measured* is 35.96% (③ in Fig. 3.8) yet the simple addition of numerical accuracy loss of *Synaptic+Wires*, *Sense+ADC*, and *DAC+Driver* totals 20.32%. We conclude that certain errors mask others.

4. Accuracy loss values follow a similar trend irrespective of the dataset. See the trendlines ④ in Fig. 3.8 for D2 and D3. However, absolute accuracy loss values vary from one dataset to another.

5. The smaller the crossbar, the lower the accuracy loss. For example, for D1, we have lower accuracy loss (of 20.32% versus 26.33%) when using a 64×64 crossbar compared to a 256×256 crossbar (③ in Fig. 3.8 vs. ⑤ in Fig. 3.9 for the *Measured* configuration). This is because a smaller crossbar has mostly smaller accumulative noise induced in wires of a smaller array.

6. Different non-idealities affect the same dataset differently for different crossbar sizes. For example, the accuracy loss due to non-idealities in *DAC+Driver* is more dominant than those in *Sense+ADC* on a 64×64 crossbar, while this is the opposite for a 256×256 crossbar. See ⑥ in Fig. 3.8 and Fig. 3.9.

Even for small yet practical crossbars of size 64×64, the accuracy loss observed in this section under both *Combined* and *Measured* configurations in Fig. 3.8 and Fig. 3.9 is still significant (e.g., from 22.19% to 24.32%) and unacceptable for a basecalling step that affects many other steps of a genome sequencing pipeline. We conclude that non-idealities in the memristor-based CIM designs, especially when combined, can be detrimental to basecalling accuracy and must be accounted for and mitigated before considering such a design useful in any other aspect.

3.4.3. EFFECT OF ACCURACY ENHANCEMENT ON QUANTIZED BASECALLERS

Fig. 3.10 shows the results of applying Swordfish's accuracy enhancement techniques to a quantized Bonito basecaller. The x-axis presents six configurations for quantization as defined in Section 3.4.1. For each quantization configuration, we evaluate five accuracy enhancement techniques, namely *VAT*, *KD*, *R-V-W*, *RSA+KD* (see Section 3.2.4), and a combination of all techniques labeled as *All*. The y-axis shows the accuracy of each technique for the corresponding quantization configuration. The horizontal line marked as Baseline (DFP 32-32) is the baseline accuracy as defined in Section 3.4.1.

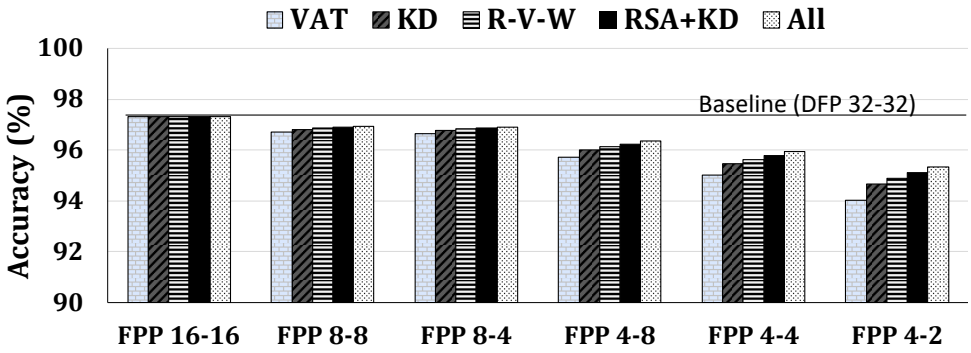

Figure 3.10: Accuracy enhancement after quantization.

We observe that retraining with quantization is an effective way to mitigate the accuracy loss induced by quantization. Our results show that with only 150 extra retraining epochs, accuracy improves by 5% on average, for a basecaller quantized down to 8-bit. By applying all quantization-aware retraining methods that we discuss in Section 3.4.1, Swordfish can retain the same accuracy as the Bonito basecaller with 32-bit floating point precision. This result is in agreement with the prior work on different types of neural networks [139]. However, Swordfish is the first work to show this result for genomic basecalling. From now on, we use 16-bit precision quantization for all evaluations we show in the remainder of this paper. We conclude that the proposed mitigation mechanisms effectively mitigate the accuracy loss due to a reasonable amount of quantization, e.g., from 32-bit to 16-bit in the Bonito basecaller.

3.4.4. Effect of Accuracy Enhancement on Non-idealities

Effect of Accuracy Enhancement on Write Variation

Fig. 3.11 presents the effects of our accuracy enhancement techniques (see Section 3.2.4) considering different write variation rates across our four datasets (D1-D4). The horizontal dotted line shows the baseline accuracy using DFP 32-32 (see Section 3.4.1) for the Bonito basecaller in all figures in Fig. 3.11. Fig. 3.11-(a)-(d) evaluate the effect of *VAT*, *KD*, *R-V-W*, *RSA+KD* separately. Fig. 3.11-(e) considers all of our accuracy enhancement mechanisms together (*Combined*), and Fig. 3.11-(f) averages the results of each accuracy enhancement technique over all the datasets (*Averaged*).[8] We make four major observations from Fig. 3.11.

First, individual accuracy enhancement mechanisms evaluated in Fig. 3.11-(a)-(d) all improve the accuracy. However, their effectiveness reduces as the write variation rate increases.

Second, the online mechanism (*RSA+KD*) in Fig. 3.11-(d) outperforms all the offline techniques in Fig. 3.11-(a)-(c). *R-V-W* in Fig. 3.11-(c) comes second in terms of accuracy. However, the difference between *RSA+KD* and *R-V-W* widens as the write variation rate

[8]The results in Fig. 3.11 consider the cases in which Swordfish maps only 5% of weights to the SRAM in our RSA-based online retraining approach (see Section 3.2.4). We will revisit this number in Section 3.4.5.

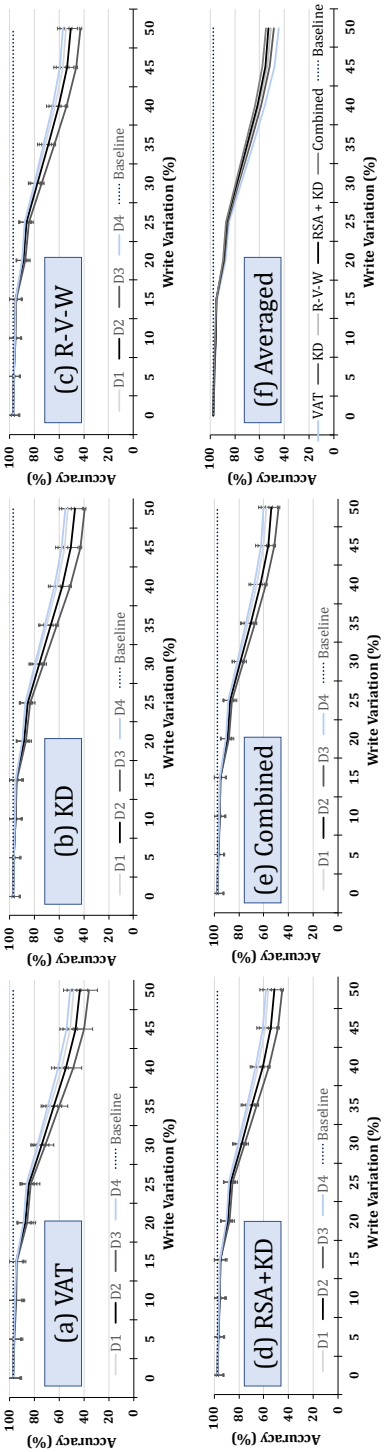

Figure 3.11: Accuracy after combining enhancement techniques over different write variations.

increases.

Third, combining all the accuracy enhancement mechanisms (*Combined* in Fig. 3.11-(e)) outperforms any individual technique over every single dataset and write variation rate.

Fourth, averaged over all the datasets (*Averaged* in Fig. 3.11-(f)), *Combined* mitigation techniques always produces the highest accuracy on average as well. However, on average, our online *RSA+KD* technique achieves a close accuracy (less than 0.001% difference) for low write variation rates, i.e., write variation less than 10%.)

These results suggest that even with multiple accuracy enhancement techniques, only minor write variations (e.g., less than 10%) can be tolerated. We conclude that a memristor-based CIM-enabeld accelerator for basecalling can be effective even with write variations, but such variations must be kept low (e.g., up to 10%). Fortunately, the projected write variation rate for memristor-based devices [139, 324] suggests the likelihood of achieving this percentage rate. For the rest of this manuscript, we assume a write variation of 10%.

EFFECT OF ACCURACY ENHANCEMENT FOR COMBINED NON-IDEALITIES

Fig. 3.12 presents the accuracy of basecalling with different accuracy enhancement techniques in crossbars of 64×64 for the modeled non-idealities. For the non-idealities, we consider the five variations of *Synaptic+Wires*, *Sense+ADC*, *DAC+Driver*, *Combined*, and *Measured* defined in Section 3.4.2. In Fig. 3.12, we evaluate five accuracy enhancement techniques of *VAT*, *KD*, *R-V-W*, *RSA+KD*, and *All* (as defined in Section 3.4.4) per non-ideality. Fig. 3.13 presents the same experiments for crossbars of 256×256. As we conclude in Section 3.4.4, we assume 10% write variation and 5% of the weights are mapped to the SRAM in the online retraining approach (see Section 3.2.4). We present our accuracy results averaged across all the evaluated datasets. We make four main observations from Fig. 3.12 and Fig. 3.13.

1. Combining of individual accuracy enhancement techniques does not improve the accuracy in an additive manner. For example, each of *VAT*, *R-V-W*, and *RSA+KD* in Fig. 3.12 improves accuracy due to *Synaptic+Wires* by 6.85%, 10.64%, 10.85%, respectively. However, when we consider all non-idealities together in the *All* configuration, accuracy improves by only 11.84% (❶ in Fig. 3.12).

2. The effectiveness of an individual accuracy enhancement technique depends on the underlying error and non-ideality it targets. For example, *VAT* is as effective as *RSA+KD* for non-idealities due to *DAC+Driver* (94.22% vs. 94.32%). However, the gap between the two approaches widens for non-idealities due to *Synaptic+Wires* (87.32% vs. 91.32%). See ❷ in Fig. 3.12.

3. Accuracy enhancement techniques improve accuracy with a similar trend over different crossbar sizes (❸ in Fig. 3.12 and Fig. 3.13). Although these results are averaged over our datasets, we note that one can make the same observation on each dataset as well.

4. Accuracy enhancement techniques are more effective for larger crossbars than for smaller ones (e.g., 256×256 compared to 64×64). This is expected because there is

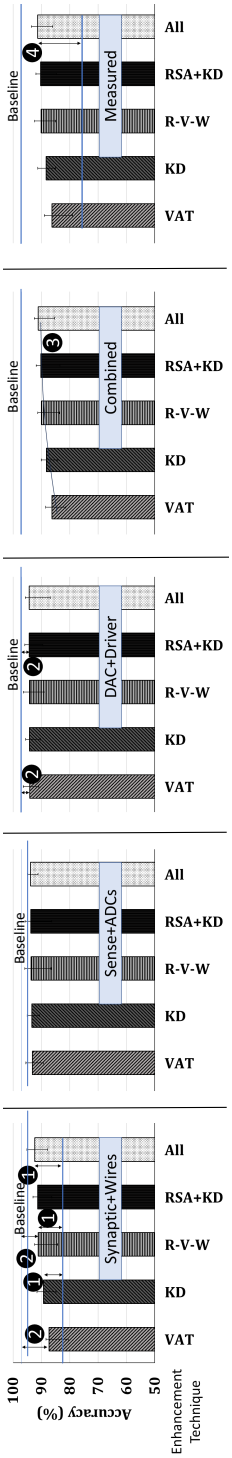

Figure 3.12: Accuracy after enhancement mechanisms for evaluated non-idealities on 64×64 crossbars.

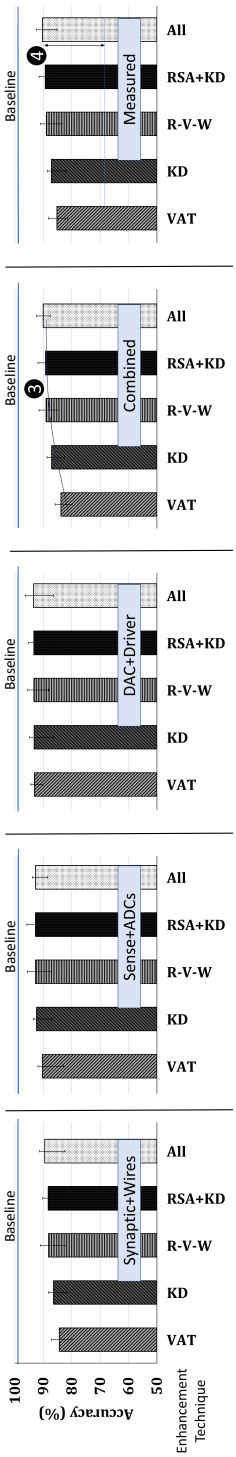

Figure 3.13: Accuracy after enhancement mechanisms for evaluated non-idealities on 256×256 crossbars.

more room for accuracy improvement for these larger crossbars, as their inaccuracies are higher. For example, we observe 22.07% improvement in accuracy for 256×256 crossbars (❹ in Fig. 3.13) compared to 16.24% for 64×64 (❹ in Fig. 3.12), after all of the accuracy enhancement techniques are applied (*All*) over all existing non-idealities (i.e., the *Measured* configuration).

We conclude that the basecalling accuracy of SwordfishAccel can match SotA levels by using robust techniques that build on each other employing reasonable crossbar sizes (e.g., 64×64) and successfully accounting for substantial circuit variations, like write variations.

3.4.5. THROUGHPUT ANALYSIS OF SWORDFISHACCEL

Fig. 3.14 shows the inference throughput for Bonito on a GPU (Bonito-GPU) card discussed in Section 3.3.2, Ideal-SwordfishAccel, Realistic-SwordfishAccel-RVW, Realistic-SwordfishAccel-RSA, and Realistic-SwordfishAccel-RSA+KD. We show the results for each of the four datasets and the average results over all datasets. The results are for a crossbar of size 64×64 and a write variation rate of 10%, and assuming 5% of weights are placed in SRAM for Realistic-SwordfishAccel-RSA and Realistic-SwordfishAccel-RSA+KD.

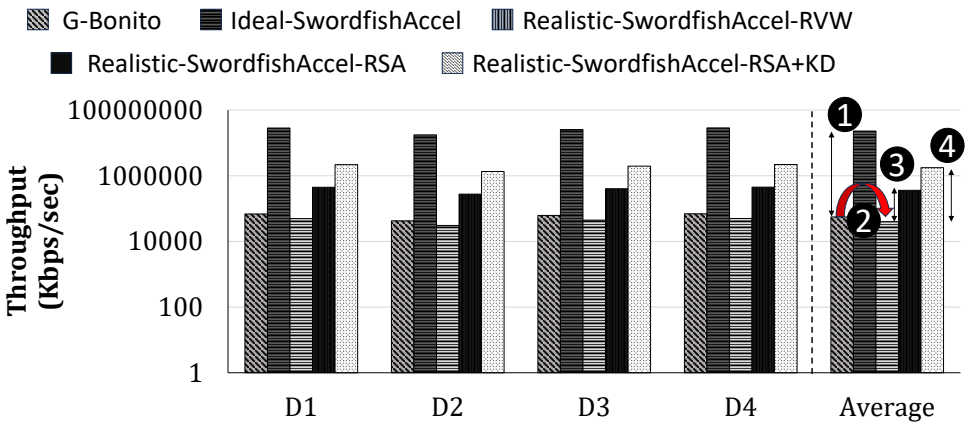

Figure 3.14: Throughput comparison of Swordfish variations.

We make four key observations. First, Ideal-SwordfishAccel improves the basecalling throughput over Bonito-GPU for all datasets, by 413.6× on average (❶ in Fig. 3.14). We expect such a large improvement in throughput because SwordfishAccel is highly optimized for the main dominant kernel in the underlying DNN of Bonito, namely VMM, and avoids unnecessary data movement while harvesting the maximum parallelism.

Second, all versions of Realistic-SwordfishAccel (i.e., Realistic-SwordfishAccel-RVW, Realistic-SwordfishAccel-RSA, and Realistic-SwordfishAccel-RSA+KD) have lower performance than Ideal-SwordfishAccel, irrespective of the dataset. Performance loss with a realistic Swordfish accelerator is expected because each realistic version adds

overheads to mitigate accuracy loss due to realistically-modeled non-idealities, which directly affect the performance of a VMM operation. For example, RSA adds overheads due to (1) the extra checks when reading some weights from the on-chip SRAM memory and (2) additional logic for combining the results from the memristor-based crossbar and on-chip memory readout.

Third, not all versions of Realistic-SwordfishAccel outperform Bonito-GPU. More specifically, if we use R-V-W for mitigating non-idealities (Realistic-SwordfishAccel-RVW in Fig. 3.14), the overhead due to additional verifications and writes significantly reduces the performance of basecalling throughput compared to Bonito-GPU by 30% on average (❷ in Fig. 3.14).

Fourth, Realistic-SwordfishAccel-RSA and Realistic-SwordfishAccel-RSA+KD provide, on average, 5.24× and 25.7× higher throughput compared to Bonito-GPU, respectively (❸ and ❹ in Fig. 3.14). Note that, for the same accuracy, Realistic-SwordfishAccel-RSA+KD requires fewer weights inside the SRAM than Realistic-SwordfishAccel-RSA due to the retraining using KD. Hence, Realistic-SwordfishAccel-RSA+KD is faster.

We conclude that a realistic basecalling accelerator designed using Swordfish by taking into account and mitigating all non-idealities of memristor-based CIM can significantly accelerate basecalling, yet its benefits are much lower than a corresponding accelerator that does not mitigate such non-idealities and thus has much lower accuracy.

3.4.6. AREA VS. ACCURACY ANALYSIS

Fig. 3.15 shows the tradeoff between accuracy and area in Realistic-SwordfishAccel-RSA+KD (see Section 3.4.5) for two different crossbar sizes (64×64 on the left and 256×256 on the right), with four different percentages of weights (i.e., 0%, 1%, 5%, and 10%) assigned to the SRAM memory (see Section 3.2.4). The area numbers show the absolute area for implementing Realistic-SwordfishAccel-RSA+KD considering the overhead of RSA+KD discussed in Section 3.2.4. The red dashed line shows the accuracy of the original Bonito basecaller. We make three main observations.

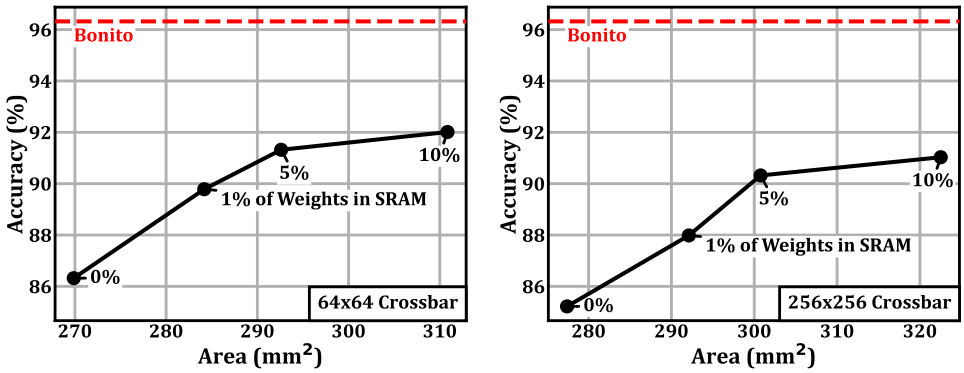

Figure 3.15: Accuracy vs. Area evaluation of Realistic-SwordfishAccel-RSA+KD.

First, the more weights are assigned to SRAM, the higher the accuracy of Realistic-SwordfishAccel-RSA+KD. This is expected because we effectively reduce the non-idealities of the system by using more SRAM cells to remap non-ideal memristors.

Second, the area of extra SRAM cells used in Realistic-SwordfishAccel-RSA+KD increases significantly with the percentage of weights assigned to SRAM. In contrast, the accuracy improvement saturates and does not increase significantly beyond 5% of weights assigned to SRAM.

Third, assigning only 5% of weights to SRAM is sufficient to be within 5% of Bonito-GPU's accuracy for the 64×64 crossbar.

We conclude that accounting for non-idealities in different ways exposes tradeoffs between accuracy and area overhead, which our Swordfish framework enables the designer to rigorously explore.

3.4.7. Verdict on Realistic-SwordfishAccel

Swordfish emphasizes the potential of significant throughput improvement for basecalling and possibly other large DNNs that require throughput acceleration while having stringent bound for another metric, e.g., accuracy using (re-)emerging computing paradigms and emerging technologies. Although Swordfish might not currently offer accuracy on par with state-of-the-art methods, the impressive 25.7× enhancement in performance marks it as an advantageous development. Note that Swordfish still maintains a competitive accuracy in basecalling by deploying a unique synergy of mitigation strategies and optimally-sized crossbar designs capable of weathering substantial circuit variations.

Swordfish's results for Bonito, a large DNN, challenge a widely-held belief that DNN-based applications naturally thrive on memristor-based CIM due to inherent redundancy in large networks. Swordfish applicability to other applications that utilize sizable DNNs, mandate high accuracy, and demand significant throughput requires tailored exploration. Yet nothing hinders researchers from adopting a Swordfish-like approach to examine their application on such architecture thoroughly. The decision on whether memristor-based CIM is the optimal acceleration technique remains case-specific.

We know there are problems with the memristors, but Swordfish still shows promise. Our results in Section 3.4 detail this. Our assessment of mitigation strategies has already identified promising ones and has sparked suggestions for future methods. These insights can be adopted in next-generation Swordfish deployments or similar accelerators. Given our results, we believe it is more productive to find solutions to the accuracy issues, some come with memristors becoming more mature, and some come with more potent HW/SW co-designed methods rather than discarding the approach.

3.5. Discussions and Future Works

3.5.1. Applicability of Swordfish Looking forward

Swordfish emphasizes the importance of a framework for evaluating multiple metrics when designing a memristor-based CIM accelerator targeting large DNNs that require throughput acceleration while having stringent bound for another metric, e.g., accuracy (in the presence of emerging technologies with many non-idealities).

Swordfish's realistic results, Realistic-SwordfishAccel, for Bonito, a large DNN, challenge the notion that DNN-based applications naturally thrive on memristor-based CIM due to the inherent redundancy present in large neural networks. Although Realistic--SwordfishAccel might not currently offer basecalling accuracy on par with state-of-the-art methods, its large (25.7×) enhancement in performance (Section 3.4.5) at a much higher accuracy than baseline CIM marks it as an advantageous development. Even in the presence of memristor-based CIM non-ideality, Swordfish still shows promise, and Realistic-SwordfishAccel still maintains a competitive accuracy in basecalling by deploying a unique synergy of mitigation strategies (against non-idealities and variations) on moderately-large crossbar designs (e.g., 64×64 or 256×256). Our results in Section 3.4 detail this. Given our results, we believe it is productive and important to find more solutions to the memristor-based CIM non-idealities going forward; we believe some solutions will come with memristors becoming more mature, and some will come with more potent accuracy enhancement techniques and HW/SW co-designed methods.

3.5.2. OTHER DNN-BASED APPLICATIONS

Our paper discusses Swordfish as a framework for accelerating basecalling using a memristor-based CIM architecture. Our results (Section 3.4) show the unique nature of the large DNN in Bonito, which, despite its inherent redundancy, does not quite reach SotA accuracy on memristor-based CIM, thus presenting an exciting challenge. This intriguing finding encourages a deeper exploration into CIM designs for large DNNs, reminding us not to rely solely on the scalability assumptions based on small network evaluations, such as simple CNNs for MNIST. Our results also demonstrate a large acceleration opportunity for basecalling using SwordfishAccel if we can mitigate the memristor-induced accuracy loss through HW/SW co-designed approaches. We believe other DNN-based applications that use memristor-based CIM accelerators (e.g., [139, 324, 342]) can also benefit from our approach and Swordfish. For example, large DNN models in autonomous driving (e.g., [342–344]) that require accurate yet high-throughout and low-latency execution can use a Swordfish-like approach to build memristor-based CIM accelerators for their underlying large DNNs. We believe and hope that Swordfish can aid such applications in terms of both accuracy and performance.

3.5.3. BETTER ACCURACY ENHANCEMENT TECHNIQUES

Our results show that accuracy enhancement can pave the way toward SwordfishAccel becoming a reliable solution. Our online retraining mechanism shows the highest potential to improve the accuracy loss. We believe there needs to be more research on better mitigation techniques for existing and future non-idealities in memristor-based designs. Specifically, we suggest hardware/software co-designed solutions such as our RSA+KD technique in Section 3.2.4. Hardware-based solutions to mitigate non-idealities [345] that are orthogonal to our RSA+KD approach is also an example of possible avenues of future work.

3.6. CONCLUSION

This paper introduces Swordfish, a modular and extensible framework for accelerating the evaluation of genomic basecalling via a memristor-based Computation-In-Memory architecture. Swordfish includes a strong evaluation methodology, mitigation strategies for hardware non-idealities, and characterization results to guide the modeling of memristors. Using Swordfish, we demonstrate the significant challenges of using non-ideal memristor-based computations for genomic basecalling and how to solve them by combining multiple mitigation techniques at the circuit and system levels. We demonstrate the usefulness of our findings by developing SwordfishAccel, a concrete memristor-based CIM design for our target basecaller Bonito that uses accuracy enhancement techniques guided by Swordfish. We conclude that the Swordfish framework effectively facilitates the development and adoption of memristor-based CIM designs for basecalling, which we hope will be leveraged by future work. We also believe that our framework is applicable to other DNN-based applications and hope future work takes advantage of this.

4

RattlesnakeJake: A Fast and Accurate Pre-Alignment Filter Suitable for Computation-in-Memory

Significant improvements in pre-alignment filter accuracy have shifted the execution bottleneck of short-read sequence alignment to the filtering step for many genomics datasets. Current pre-alignment filters move data from memory to the processing units, and when rejection is determined, this results in wasted energy and time. This chapter presents RattlesnakeJake, a hardware/software co-designed accelerator that speeds up and reduces the energy consumption of pre-alignment filtering and, hence, sequence alignment. RattlesnakeJake achieves this by (1) proposing a lightweight and hardware-friendly filtering algorithm, (2) adopting the Computation-In-Memory paradigm to avoid unnecessary data movement, and (3) exploiting resistive memories (memristors) to perform the low-level operations required by the proposed algorithm. Our preliminary results for RattlesnakeJake show an accuracy at the state-of-the-art (SotA) level and a significant improvement in the execution time of sequence alignment, irrespective of the evaluated dataset. The improvement for filtering varies from dataset to dataset and goes up to ~7× and ~80×, compared to SotA accelerators on GPU and CPU, respectively.

This chapter is partially based on the candidate's work [117].

As we briefly discussed in Section 2.2.1, sequence alignment of genomics data is a fundamental step in most genomic studies that help us with virus surveillance and precision medicine [1–11]. Currently, computationally costly dynamic programming-based (DP) algorithms are the solution of choice for sequence alignment. Pre-alignment filtering[1] has recently been introduced as a solution to heuristically replacing the need for expensive DP solutions in many cases. This consequently speeds up the overall process of sequence alignment significantly [186–188, 346]. With the achieved speedup, pre-alignment filters become the (new) performance bottleneck to focus on [186–188], with solutions using graphics processing units (GPUs) and field-programmable gate arrays (FPGAs) also being proposed. Unfortunately, none of these works resolve this new performance bottleneck. Moreover, despite these accelerations, there is still one bottleneck in all these works, i.e., the (large) movement of data that most of the time turns out to be unnecessary as the data is decided to be filtered out. This unnecessary data movement results in wasted time and energy consumption. Therefore, there is a need for a more efficient design to tackle this bottleneck in the sequence alignment pipeline and simultaneously avoid this wasted work, time, and energy consumption caused by data movement.

We propose RattlesnakeJake, a hardware/software (HW/SW) co-designed accelerator based on Computation-In-Memory (CIM) paradigm, capable of pre-alignment filtering for short-sequence alignment. RattlesnakeJake first proposes a lightweight and hardware-friendly algorithm. It then exploits emerging non-volatile memories as its underlying device for hardware acceleration. RattlesnakeJake chooses these devices since they offer greater densities, access speeds, and non-volatility than conventional memories such as DRAM or SRAM. RattlesnakeJake's hardware has a hierarchical design that supports the operations required in its algorithm and maps the full algorithm to memory units and their peripheries while also taking care of input data distribution and output data processing.

Our results show that RattlesnakeJake improves upon SotA pre-alignment filters on GPU and CPU by up to ~7× and ~80×, respectively, for the same real input datasets. These improvements stem from (1) the underlying lightweight filtering algorithm, (2) optimized data flow in RattlesnakeJake, and (3) the prevention of unnecessary data movement for filtering. RattlesnakeJake achieves all these benefits while neither replacing the sequence alignment nor introducing extra false negatives into the pre-alignment filter.

The major contributions of the work described in this chapter are:

- A memory-friendly filtering algorithm with no assumption on data alignment leading to extra penalty.
- A configurable memristor-based CIM-enabled architecture for short-read pre-alignment filtering.
- RattlesnakeJake, a HW/SW co-designed accelerator for pre-alignment filtering inside the memory.
- Extensive evaluation and comparison of RattlesnakeJake using real data against previous software and hardware pre-alignment filters.

[1] We use the term filter and pre-alignment filter interchangeably hereafter.

4.1. PROPOSAL AND ARCHITECTURE

This section discusses RattlesnakeJake's software and hardware design.

4.1.1. RATTLESNAKEJAKE'S ALGORITHM

SotA filters mitigate the cost of sequence alignment by approximating the difference (aka edits) between the DNA read sequence and reference genome pair using simple operations (e.g., Hamming distance). If the approximated difference is already greater than the threshold for an acceptable alignment, filters safely reject/filter the read sequence and avoid costly DP. SneakySnake [186] is currently the SotA filter (i.e., fastest with the highest accuracy).

We design RattlesnakeJake's algorithm to account for two limitations stemming from rigid data accessibility in a CIM-enabled design: (1) irregular bit-position of the start of a reference sequence within memory access, and (2) rigid dimensions of memory units such as crossbars, subarrays, etc.

Algorithm 1 RattlesnakeJake Algorithm

Input: Read, Ref, E, Length, k
Output: Accept

1: $N_{seg} \leftarrow \lceil Length/k \rceil$
2: $Matches \leftarrow 0$
3: **for** $i \in \{0..N_{seg}-1\}$ **do**
4: $\quad Match \leftarrow 0$
5: $\quad$ **for** $e \in \{-E..+E\}$ **do**
6: $\quad\quad ReadSeg \leftarrow Read[ki..k(i+1)-1]$
7: $\quad\quad RefSeg \leftarrow Ref[ki+e..k(i+1)-1+e]$
8: $\quad\quad$ **if** $ReadSeg == RefSeg$ **then**
9: $\quad\quad\quad Match \leftarrow 1$
10: $\quad\quad$ **end if**
11: $\quad$ **end for**
12: $\quad Matches \leftarrow Matches + Match$
13: **end for**
14: $Accept \leftarrow (Matches >= N_{seg} - E)$
15: **return** $Accept$

RattlesnakeJake reduces DP to exact matches between shifted versions of smaller sub-sequences/segments and processes their results together. Specifically, RattlesnakeJake divides the read sequence into segments of k-bps, which are compared to the corresponding segments of the reference. Each pair of segments is checked for an exact match to determine whether an edit is present within that segment. To account for deletions and insertions, RattlesnakeJake repeats this process for references shifted by $-E$ to $+E$ bps. If the segment has an exact match with any of the (shifted) reference segments, it concludes that the segment does not contain errors. RattlesnakeJake combines the results of all of the segments and exploits the pigeon-hole principle to deduct the approximate number of edits of the sequence pairing. Algorithm 1 presents RattlesnakeJake algorithm, where E is the number of permissible edits, and k is the segment size.

Fig. 4.1 illustrates an example of RattlesnakeJake algorithm for two 20bps long sequences with an edit threshold of 2. RattlesnakeJake splits the read into 5 segments of 4bps and compares them with the reference shifted by -2 to +2 bps. RattlesnakeJake finds that only segments 1 and 4 have matching segments with the reference, indicating that there are at least 3 edits. This exceeds our edit threshold ($E = 2$), and thus Rattlesnake-

Jake rejects the pair for alignment.

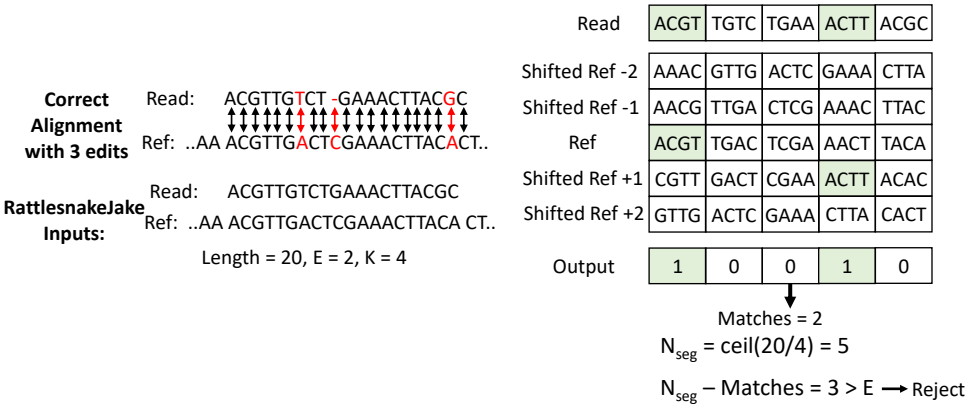

Figure 4.1: RattlesnakeJake with Length=20bps, E=2, and k=4bps.

RattlesnakeJake's hardware-friendly algorithm flexibly exploits two key trade-offs. First, a trade-off between accuracy and hardware-friendliness. Changing from DP (or SNR sub-problems in SneakySnake) to finding the exact matches of short segments may underestimate the number of edits between the read and reference. This leads to more reads passing the filter. However, exact matching is known to be well-supported in hardware. Moreover, the possibility of supporting exact matches for short strings is higher in a CIM-enabled crossbar. Second, a trade-off between required resources and achievable parallelism/speed. Finding the exact matches between each segment from the read and the corresponding segment from the reference and its shifted variants are independent problems and can be parallelized with extra resources. Our evaluations in Section 4.2 investigate these trade-offs in more detail.

4.1.2. RATTLESNAKEJAKE'S ARCHITECTURE

We envision RattlesnakeJake as a pre-alignment filtering accelerator inside the memory as shown in Fig. 4.2-(a). This way, RattlesnakeJake prevents unnecessary data accesses from memory to CPU or GPU via filtering. Due to our positioning, RattlesnakeJake follows a conventional hierarchical design; i.e., RattlesnakeJake contains several bank groups, banks, subarrays, and tiles. Fig. 4.2-(b) to -(e) present RattlesnakeJake and its top-down organization.

In this hierarchical architecture, a tile is an array of cells connected together in a conventional crossbar format of rows and columns. Each cell consists of a 1 transistor and 1 memristor device (aka 1T1R format) and can store 1 bit of data. A tile also includes all the necessary peripheries for read and write operations (e.g., DACs and SAs). A group of tiles, all working in parallel, create a subarray. This separation of tiles and subarrays is needed to (1) mitigate the overhead of the peripheries and shared components (e.g., TCAMS and input/output buffers) and (2) provide enough parallelism. Then, several subarrays constitute a bank, and multiple banks come together and create bank groups. This hierarchy is adopted to (1) fully utilize the available busses for input and output

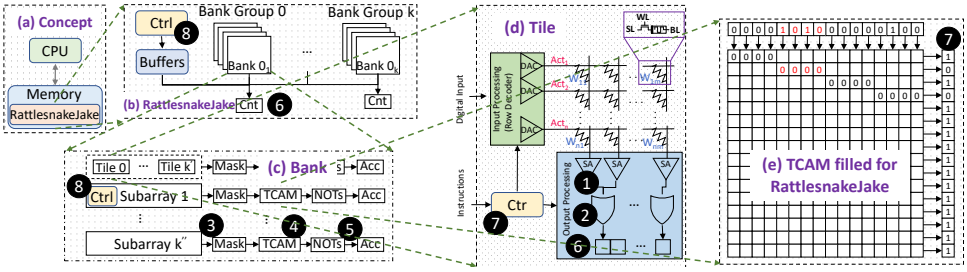

Figure 4.2: (a) RattlesnakeJake system placement, (b) Overview of RattlesnakeJake, (c) banks in RattlesnakeJake, (d) Tile and peripheral logics, and (e) Example filled TCAM.

movement and (2) provide the highest possible parallelism while minimizing the buffer overheads. A similar method is used in today's DRAMs and other memory technologies.

To support the required kernels in Algorithm 1, RattlesnakeJake augments normal memory units with extra hardware:

- ❶ Modified SA at tile level. RattlesnakeJake utilizes SAs enhanced with Scouting Logic [87, 106] so that they can perform XOR operation. This is the base operation needed for the exact match required in Algorithm 1.

- ❷ A series of OR gates at tile level. RattlesnakeJake utilizes these gates to account for encoding every base pair with 2 bits. This is a genomic-specific modification needed as DNA sequences can contain 4 types of bases encoded as two bits of data in RattlesnakeJake. Since the XOR produces a bit-wise result, RattlesnakeJake performs an XOR on every pair of two bits to obtain a base-level result.

- ❸ Logic for masking per subarray. At the bank level, the outputs of all addressed tiles are combined to form one bit vector. Tiles do not always contribute to the final result, for example, when the start of the reference does not line up with the start of a word (Section 4.1.1). In these cases, RattlesnakeJake masks the non-contributing tiles using a AND.

- ❹ 1 TCAM per subarray. RattlesnakeJake detects patterns of consecutive zeros in the output of each tile using this TCAM. Each TCAM is filled with rows of k consecutive zeros, and n - k don't-care values, where k is the #patterns in RattlesnakeJake and n is the dimension of the TCAM. Note that, although in principle RattlesnakeJake does not fully uses all TCAM rows, additional rows are added so that one can configure RattlesnakeJake for the detection of more patterns. This maintains the ability to implement other current/future filtering algorithms.

- ❺ A series of Not and AND gates per subarray. The output of each TCAM goes through negation and accumulation (via AND) phases and repeats until we have the result for all the shifted segments.

- ❻ A tree-based counter per bank group at the rank level. RattlesnakeJake uses these counters to effectively determine the minimum edit between the original read and the reference by counting the '1's in the output vector.
- ❼ Input and output buffers in all levels. RattlesnakeJake places appropriate buffers at different levels of the hierarchy to ensure seamless data flow among components with no data loss.
- ❽ Controller in all levels. RattlesnakeJake places small FSM-based controllers at each level of its hierarchy that oversees required operations and dataflow.

It is worth noting that recently, Shahroodi et al. proposed SieveMem, an architecture based on CIM that can support the existing kernels in pre-alignment filters [118]. In theory, SieveMem supports RattlesnakeJake's algorithm, and RattlesnakeJake is a simplified version of SieveMem.

4.1.3. RATTLESNAKEJAKE ALGORITHM TO HARDWARE MAPPING

Before runtime, RattlesnakeJake stores multiple shifted references in consecutive rows in the memory. RattlesnakeJake represents the bases as 2-bit values stored in 2 cells. Then, during runtime, RattlesnakeJake writes the segments of the read to a dedicated query row, such that the first bit of the segment sequence lines up with the first bit of the corresponding non-shifted reference sequence. RattlesnakeJake performs exact string matching at the tile level. To perform string matching, RattlesnakeJake performs an XOR between the query row and a row containing one of the reference sequences. If the segments are exact matches, this operation results in an output vector of only zeros. RattlesnakeJake uses the TCAM located at the subarray level to detect this pattern. RattlesnakeJake repeats this process for all $2E+1$ shifted references. If in none of the iterations, it detects an exact match, RattlesnakeJake concludes that the segment contains an error.

In RattlesnakeJake, multiple tiles operate in parallel and RattlesnakeJake accumulates their results at the subarray level to create a bit-vector, which contains a '1' for every segment containing an error. RattlesnakeJake returns this vector to the rank level where the number of '1's is counted and compared to the edit threshold. Based on this comparison, RattlesnakeJake accepts or rejects the pairing.

4.2. EVALUATIONS

4.2.1. EVALUATION METHODOLOGY

Implementation. RattlesnakeJake is evaluated using a cycle-accurate RTL-based simulation based on the proposed architecture Section 4.1. The design is verified by comparing the simulation results with the output of a software version of RattlesnakeJake algorithm (RattlesnakeJake-SW). RattlesnakeJake-HW uses a memory model based on a small RRAM chip prototype in TSMC 40 nm CMOS technology [330]. The model is from the EU project MNEMOSENE [333], provided to us by generous partners. The additional circuit and controller in RattlesnakeJake-HW also use TSMC 40 nm technology node in Synopsis Design Compiler [331] to obtain the latency, power, and area numbers. Here, we only discuss the latency results.

We run all of our experiments on a 28-core server with 192 GB memory equipped with Tesla-K80 and a processor operating at 2.4 GHz. We intend to open-source the implementations of RattlesnakeJake upon acceptance. Our evaluations consider the same platform and input datasets for all filters to provide a fair analysis.

Baselines. We compare RattlesnakeJake with SneakySnake (denoted with SS in the following figures) [186], SHD [189], Shouji [187], and GRIM-Filter [95]. These are four SotA pre-alignment filters, three of which have acceleration on GPU or FPGA, and one on 3D-stacked memories. We analyze the accuracy of RattlesnakeJake by comparing its output results with only the profiling outputs of these open-sourced filters.

Datasets. We use real genome datasets ($human_g1k_v38$ and $ERR240727_1$) for our reference database and input queries [186, 347, 348]. Similar to previous works [186], we create our datasets using MrFast [349] to create sets of read-reference pairs from the .fasta and .fastq files to evaluate the (pre) alignment algorithms. RattlesnakeJake uses Edlib [350] to create full-alignment results for accuracy, which will be used to verify the functionality of the pre-alignment filters.

4

4.2.2. ACCURACY ANALYSIS

Fig. 4.3 and Fig. 4.4 compare the false positive (FP) rate of several filters. FP rate in a filter shows the ratio between reads that wrongly pass the filter (i.e., should have been filtered) and go through alignment (i.e., DP) over all the reads. The lower the FP, the better. Note that RattlesnakeJake achieves the same True Positive (TP) and True Negative (TN) rate as SneakySnake, which are currently the best filtering rates.

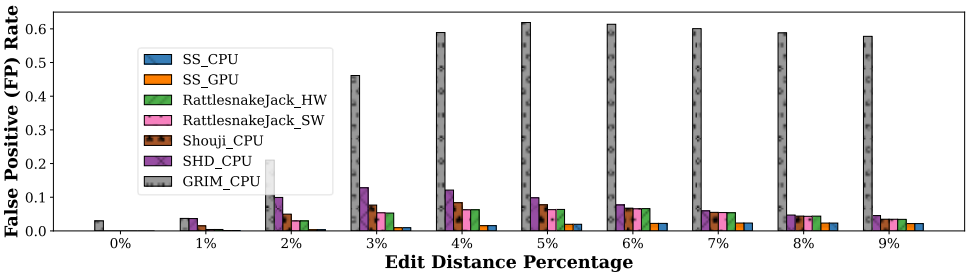

Figure 4.3: FP rate comparison on ERR240727_1 dataset for E=2.

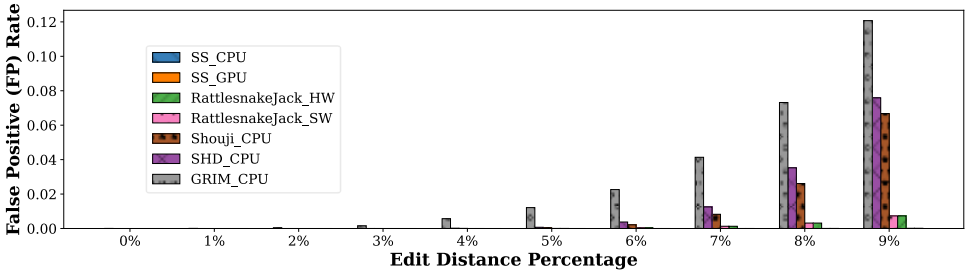

Figure 4.4: FP rate comparison on ERR240727_1 dataset for E=40.

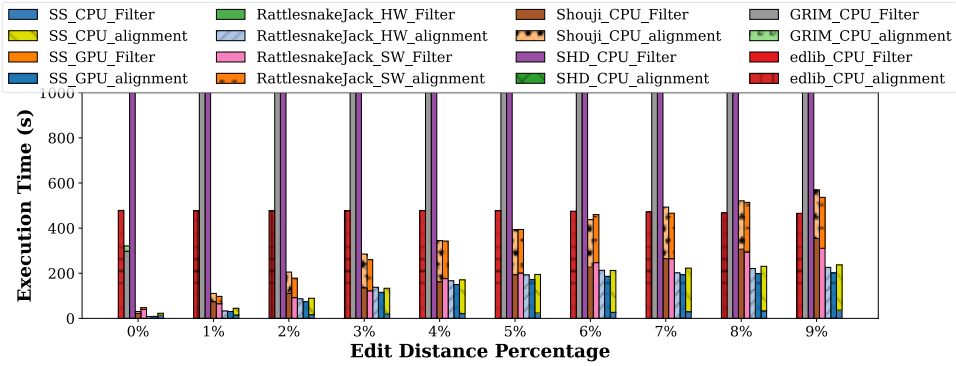

Figure 4.5: Exectution time on ERR240727_1 dataset for E=2.

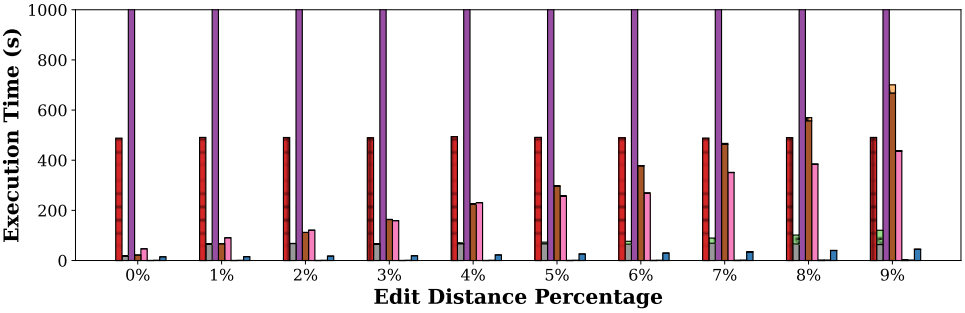

Figure 4.6: Exectution time on ERR240727_1 dataset for E=40.

We make three key observations. First, irrespective of datasets and edit threshold, RattlesnakeJake provides a low FP rate comparable with the SotA SneakySnake. Second, RattlesnakeJake outperforms SHD, Shouji, and GRIM-Filter by providing 40%, 22%, and 90%, respectively, fewer falsely-accepted sequences, on average across all of our datasets. Third, RattlesnakeJake-HW provides a close FP rate to RattlesnakeJake-SW (less than 1% difference). This means that the hardware limitation regarding the start point of the reference that changes the #segments does not significantly affect the accuracy of a hardware implementation over a software version that does not have the same limitation (and might have different #segments). We conclude that RattlesnakeJake is an accurate filter for alignment acceleration.

4.2.3. THROUGHPUT AND EXECUTION TIME

Fig. 4.5 and Fig. 4.6 present the execution time for filtering and alignment of different methods in two datasets and over several edit thresholds. We limit the y-axis that shows the execution time of filter+alignment to 1000s to better capture the trends and relative execution time of RattlesnakeJake in the system compared to other methods. We only present the results for our most reasonable configuration of RattlesnakeJake-HW.

We make three key observations. First, independent of the dataset and edit thresh-

old, both RattlesnakeJake-SW in CPU and RattlesnakeJake on hardware significantly reduces the end-to-end execution time of sequence alignment. Second, the more dissimilar datasets (e=9% vs. e=2%), the higher the benefits of RattlesnakeJake. The average speedup of end-to-end alignment time when using RattlesnakeJake is 30.37% more for e=9% over e=2% for ERR240727_1 dataset. Third, RattlesnakeJake-HW improves the filtering performance by up to ~7× and ~80× over currently the best-accelerated filter on GPU (SS_GPU_Filter) and CPU (SS_CPU_Filter), respectively. This improvement translates to a 54.68% and 84.21% for end-to-end alignment compared with SotA filter combined with SotA alignment and sheer Edlib on CPU, respectively, averaged over our datasets. We conclude that RattlesnakeJake effectively reduces the execution time of end-to-end alignment and takes a step towards mitigating the filtering bottleneck.

4.3. Discussions and Future Works

4.3.1. RattlesnakeJake for Long Sequence Alignment

RattlesnakeJake algorithm is also effective for long sequence alignment, where sequences are a size of 100Kbp. However, when it comes to hardware implementation, RattlesnakeJake, distributing the long reference or read sequences in the memory hierarchies requires different buffer sizes, control unit sequences, and potentially some additional logic. We leave the extension of RattlesnakeJake for long pre-alignment filtering to future work.

4.3.2. Potential Design Improvements

We evaluated RattlesnakeJake based on the measurements on a small RRAM chip prototype for tiles and TCAMs. However, a design space exploration is required to direct the final configuration of RattlesnakeJake to be deployed in future genomics systems. This exploration should consider different inputs, memory units and arrangements, and variations of device and circuit behavior (e.g., non-idealities) for different organizations. We leave this for future work.

Moreover, we believe a small reconfiguration of TCAMs at the subarray level and the references of SAs at the Tile level can provide more kernel support in RattlesnakeJake. These modifications, in addition to some modifications in RattlesnakeJake FSM controllers, can provide support for more pre-alignment filtering kernels and algorithms, making RattlesnakeJake compatible with previous and future filters. We leave this exploration to future work.

4.4. Conclusion

This chapter proposes a HW/SW co-designed accelerator, called RattlesnakeJake, based on memristor devices and CIM paradigm to prevent unnecessary data movement for sequence alignment by filtering dissimilar short sequences inside the main memory. When used in a larger genomics pipeline, RattlesnakeJake shifts the processing bottleneck back (again) to the DP step of the remaining sequences. Hence, our work calls for even more accurate filtering algorithms and better DP-based alignment algorithms.

5

SieveMem: A Computation-In-Memory Architecture for Fast and Accurate Pre-Alignment

As discussed in Chapter 4, high DNA alignment time slows down genomic studies. Pre-alignment filtering cuts this time but is now the new execution bottleneck in many genomics studies. Most of the pre-alignment execution time goes into moving sequences from memory to processors, even though many sequences get filtered out. Current accelerators have the same issue and lack future-proofing. This chapter addresses these shortcomings by introducing SieveMem. SieveMem is an architecture that exploits the Computation-In-Memory paradigm with memristive-based devices to support shared kernels of pre-alignment filters and algorithms inside the memory (i.e., preventing data movements). SieveMem architecture also provides support for future algorithms. SieveMem supports more than 47.6% of shared operations among all top 5 *SotA filters. Moreover, SieveMem includes a hardware-friendly pre-alignment filtering algorithm called BandedKrait, inspired by a combination of mentioned kernels. Our evaluations show that SieveMem provides up to 331.1× and 446.8× improvement in the execution time of the two most common kernels. Our evaluations also show that BandedKrait provides accuracy at the SotA level. Using BandedKrait on SieveMem, a design we call Mem-BandedKrait, one can improve the execution time of end-to-end sequence alignment irrespective of the dataset, which can go up to 91.4× compared to the SotA accelerator on GPU.*

This chapter is partially based on the candidate's work [118].

As we discussed in Section 2.2.1 and in Chapter 4, pre-alignment filtering[1] was recently introduced as a solution to significantly speed up the overall process of sequence alignment by heuristically replacing the need for expensive DP solutions for many inputs, given a pre-defined edit distance threshold between the inputs [186–188]. SotA pre-alignment filters speed up the short-read (100-250 base-pairs or bps) sequence alignment so much so that they themselves become the (next) bottleneck to be accelerated [187, 188]. Although one SotA work [186] accelerates the pre-alignment filters on graphics processing units (GPUs) and field-programmable gate arrays (FPGAs), this work still does not completely alleviate the bottleneck. Moreover, we find that data movement is a major issue in SotA pre-alignment filters, i.e., these filters waste a lot of time and energy when moving the sequences from the memory to processing units, most of which turn out to be unnecessary as the data is decided to be filtered out eventually [186]. Therefore, there is a need for a more efficient design to tackle the filtering bottleneck in the sequence alignment pipeline and simultaneously avoid the wasted work, time, and energy consumption caused by data movement in the system.

We propose SieveMem, an architecture based on Computation-In-Memory (CIM) principles capable of handling shared kernels in pre-alignment filtering for short-sequence alignment. We identify the shared kernels via an extensive profiling process using the same datasets and platforms for all the pre-alignment filters for a fair comparison and accurate recognition of bottlenecked operations. We also propose BandedKrait, a novel lightweight pre-alignment filter from shared kernels in previous filters (i.e., those supported by SieveMem) and its mapping into SieveMem architecture, a design called Mem-BandedKrait. SieveMem adapts the CIM paradigm since it requires prevention of data movement, processing a large amount of data, and performing relatively small and/or simple computations, the main characteristics a CIM architecture embraces [93, 108, 128]. SieveMem's design comprises two abstraction levels: (1) A low-level abstraction that supports the shared kernels in filters and (2) a high-level abstraction that supports filtering algorithms using the existing hardware in SieveMem and takes care of input data distribution and output data processing. SieveMem is designed to support filtering for short reads because most of the available data in the genomics realm is still short reads (sequences of length 100 to 250 base-pairs), even though the industry is slowly moving towards long-read sequencing. Therefore, supporting short reads filtering and alignment will remain relevant problems in upcoming years.

Our results show that SieveMem accelerates the execution time of the identified shared kernels by up to 331.1× and 446.8× for the two most common kernels in pre-alignment filters. The results also show that BandedKrait achieves an accuracy on par with SotA filter SneakySnake. When accelerated on SieveMem, Mem-BandedKrait accelerates pre-alignment filtering by up to 95.5× and 1292× over SotA filters on GPU and CPU, respectively, for the same real input datasets. SieveMem achieves all these benefits, neither replacing the sequence alignment nor introducing extra false negatives into the pre-alignment filtering process, i.e., SieveMem only affects the pre-alignment filtering step positively. Therefore, users can still employ SieveMem with any sequence aligner.

Our contributions are the following:

[1] We use the term filter and pre-alignment filter interchangeably hereafter.

- An configurable memristor-based accelerator (called SieveMem) that supports the most common shared kernels in pre-alignment filters.

- A memory-friendly filtering algorithm, called BandedKrait, accounting for the inflexibility of having the starting point access aligned with the memory units in memory/hardware. BandedKrait uses the same shared kernels and corresponding hardware of previous filters. BandedKrait is in essence the same algorithm as RattlesnakeJake in Chapter 4. However, due to the tight integration of RattlesnakeJake with its hardware, we separate them in this chapter to evaluate SieveMem fairly.

- A CIM-enabled realization of BandedKrait on SieveMem, called Mem-BandedKrait, for short-read pre-alignment filtering.

- An extensive evaluation of SieveMem's supported kernels, BandedKrait, and Mem-BandedKrait using real data against previous software and hardware pre-alignment filters.

5.1. MOTIVATION AND PROFILING

This section (1) identifies the shared kernels in filters and (2) motivates the CIM-enabled acceleration of filters. We refer to Section 5.3.1 for detail on our setup and datasets.

5.1.1. SHARED KERNELS IN FILTERS

We profile three SotA (i.e., accurate and fast) pre-alignment filters over different percentages of edit distances. Fig. 5.1, Fig. 5.2, and Fig. 5.3 present a breakdown of the execution time for each non-overlapping kernel in MAGNET, Shouji, and SHD, respectively, over our representative dataset.

We make three key observations. First, across all filters and edit distances, two operations, namely Hamming mask creation (HMC) and detecting short patterns (called Short Pattern Detect or SPD), make up most of the execution time. Note that we categorized checking hamming distance, leading zero counts, and SRS, in MAGNET, Shouji, and SHD, respectively, into SPD, as they are all essentially detecting short patterns. For example, HMC accounts for up to 66%, 84%, and 88% in end-to-end execution time of MAGNET, Shouji, and SHD, respectively. Moreover, up to 72% and 68% of the end-to-end execution time in MAGNET and Shouji, respectively, is spent on SPD.

Second, the relative percentage of these two operations varies per filter, edit distance, and dataset (and, therefore, is data dependent). However, the combined HMC and SPD account for a minimum 47.6% of filter's execution time, going up to 97.5%.

Third, apart from HMC and SPD, these filters also share other operations of the same nature. For example, kernels for counting edits, pre- and post-processing inputs and outputs, and extra reads and writes of intermediate results.

We conclude that HMC and SPD comprise most of the execution time in SotA filters. Acceleration of these two kernels can resolve the bottleneck in filters and simultaneously provide some assurance for future compatibility of filters that utilize the same kernels.

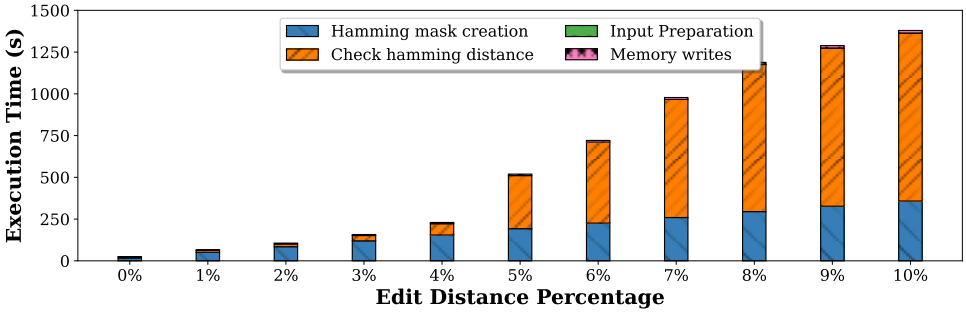

Figure 5.1: MAGNET's breakdown.

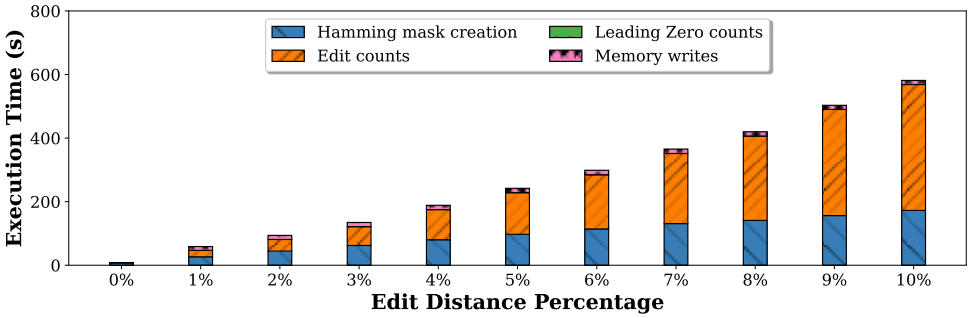

Figure 5.2: Shouji's breakdown.

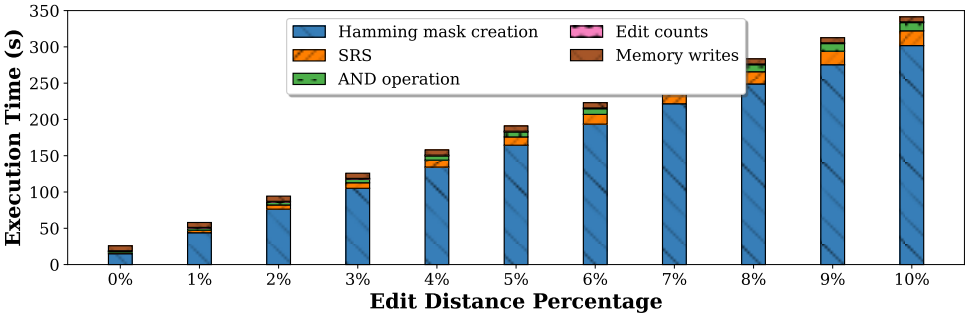

Figure 5.3: SHD's breakdown.

5.1.2. DATA MOVEMENT IN FITLERS

Fig. 5.4 presents the breakdown of execution time for a SotA accelerated filter, SneakySnake on GPU (Snake-on-GPU), on two very different datasets of short reads labeled as D1 and D2.

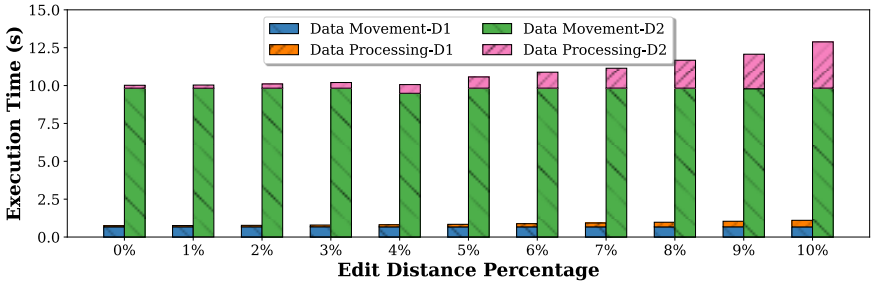

Figure 5.4: Data movement bottleneck in accelerated filters.

We observe that over both datasets, Snake-on-GPU spends a minimum of 60% of its execution time just transferring data from memory to GPU. This portion can go up to 98%, depending on the dataset and edit threshold. We conclude that data movement is the bottleneck of the overall performance in SotA accelerators for filtering.

The results presented in this section call for an acceleration of pre-alignment filters with an emphasis on eliminating data movement and supporting commonly shared kernels such as HMC and SPD.

5.2. PROPOSAL AND ARCHITECTURE

This section discusses (1) SieveMem and how it supports HMC and SPD and mitigate data movement, (2) BandedKrait, our new lightweight pre-alignment filtering algorithm, and (3) Mem-BandedKrait, hardware realization of BandedKrait on SieveMem.

5.2.1. SIEVEMEM ARCHITECTURE

Fig. 5.5-(a) presents the placement of SieveMem in a real system, i.e., part of the memory. Due to this placement, SieveMem follows a hierarchical structure similar to conventional memories, i.e., SieveMem consists of ranks, bank groups, banks, subarrays, and tiles (Fig. 5.5-(b), -(c), and -(d)).

However, to support the target shared kernels for filtering, SieveMem augments the substrate as we will discuss below:

Tile level changes: SieveMem enhances the SAs (❶) and adds a series of OR gates (❷). The modified SAs [87, 106] enable SieveMem to perform logical operations such as XOR, AND, etc., with a minimal area overhead on data in an entire row of a tile. This design is possible due to the nature of memristor devices that inherently follow Kirschof's law. The subsequent series of OR gates relates to the nature of our working datasets and encoding. We want SieveMem to enable filtering for DNA short-reads of {A, C, G, T}. This means we can encode each character with 2 bits in a hardware realization. Since our enhanced SAs

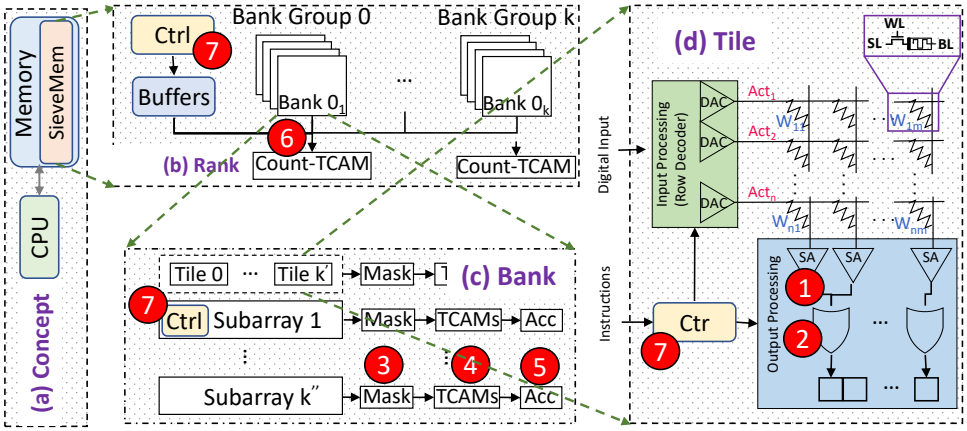

Figure 5.5: (a) SieveMem system placement, (b) to (d) An overview of SieveMem hierarchy and its additional components at different levels.

5

perform bitwise operations, SieveMem needs this series of OR to obtain a result based on base-pairs. Our walk-through example in Section 5.2.2 details this further.

Bank level changes: SieveMem adds a series of AND gates for masking ❸, 2 TCAMs ❹, and a series of AND gates ❺. SieveMem uses AND gates to select any section of the outputs from tiles. The masking gates are necessary for a true CIM-enabled design where we cannot guarantee our target data is aligned perfectly with crossbars. A limitation that previous CIM-enabled designs typically face [115]. SieveMem uses the two memristor-based TCAMs (called Pattern-detect and Output-select) for all the necessary pattern detection in different kernels of pre-alignment filters. The AND gates are used to accumulate the results over several related checks in the SieveMem, for example, checks for the same read sequence over shifted versions of the reference. Section 5.2.2 details how one can use different masks, pre-filled TCAMs, and bitwise gates to perform different operations and pre-alignment filtering algorithms.

Rank level changes: SieveMem includes a TCAM called Count-TCAM (❻) at the rank level per each bank group. SieveMem uses this TCAM to effectively calculate the minimum edit between the sequences.

Overall modifications: SieveMem adds a small FSM to control each hierarchy level's logic and memory operations (❼). SieveMem also includes some input and output buffers in different levels. Each level's FSM oversees the operations of the components in that level of SieveMem. These controllers are hierarchical, i.e., smaller FSMs control the operations at bank, subarray, and tile levels, all managed by a controller at the higher level. The buffers ensure seamless dataflow among different levels with no data loss.

5.2.2. SIEVEMEM EXAMPLE SUPPORT FOR SHD

SHD requires removing sequences of 1 to 2 zeros from the bit vector produced by the tile after the similarity checks via XOR. The original algorithm [189] performs this by detecting patterns of "101" and "1001". Here, SieveMem opts for an inverse detection,

i.e., SieveMem detects sequences of zeros that are 3-bits or longer. Therefore, the output vector should be the same as the input vector but with the removal of the short '0' sequences.

SieveMem provides support for this via the TCAMs at the bank level. To implement the required pattern detection and selection, the output should always be a '1', except for those bits which are part of a sequence of 3 zeros or more. This is the case when a pattern of **0**00, 0**0**0 or 00**0** is detected, where the bold character indicates the position of the bit in the original bit vector at the input of the Pattern-detect TCAM. Therefore, the Output-select TCAM essentially performs a NOR operation on the Pattern-detect output corresponding to those 3 patterns. If any of the three patterns is detected, it will result in a '1' in the intermediate signal. The Output-select TCAM will, therefore, not output a '1' but a '0'. Conversely, if none of the patterns is detected, then SieveMem can be sure that the bit is not part of a sequence of 3 or more zeros. The intermediate result will contain only 3 zeros for these patterns, and the Output-select TCAM will output a '1' for that bit position in the XOR result.

Since the 2 base-pairs to the left and rightmost extremities of the TCAM input require information about the base-pairs to the left and right, respectively, these positions will always be left '0', done by the left and rightmost columns of the Output-select TCAM. Since the output of the bottom two rows of the Pattern-detect TCAM is always '1' due to it being filled with don't-cares, the output of the Output-select TCAM, which looks for '0', will always be '0'.

Fig. 5.6 presents how one can pre-fill the two TCAMs of our SieveMem to support the detection pattern required by SHD. Note that the blank TCAM entries stand for don't-cares. Here, there is a '0' between the two '1's in the 6th bit counting from the left. We observe that the patterns surrounding this bit are "010", "101", and "010". None of these are "000". Therefore, the Output-select TCAM will activate the row in green.

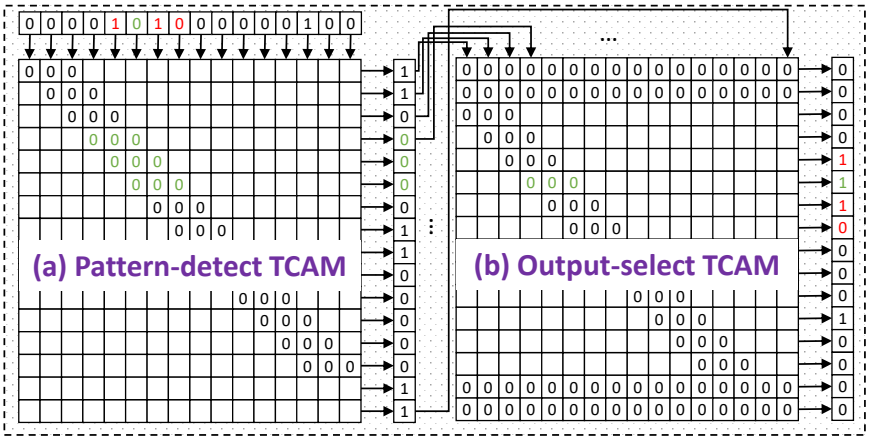

Figure 5.6: TCAMs values in SieveMem to support SHD.

At the rank level, SieveMem collects all bank group results and counts the number of edits in a word set of the read sequence. SieveMem uses Count-TCAM to detect patterns

and to assign a number for the edits of the detected patterns. Count-TCAM is a 4-bit wide TCAM used in algorithms where we need to split the final bit-vector into segments of k bits, e.g., SHD with k=4. Fig. 5.7 presents an example programming for Count-TCAM to support SHD.

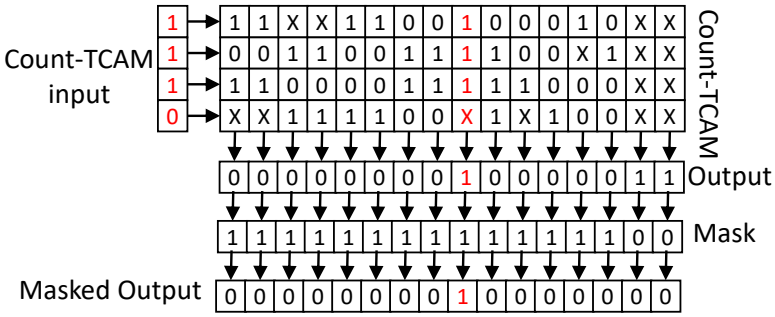

Figure 5.7: Filled Count-TCAM for SHD.

For the segment of "0000", SHD counts no edits, while it counts two edits for "0101", "0110", "1001", "1010", "1011" and, "1101". The rest of the cases are counted as single edits. Therefore, SieveMem compresses these into 14 entries with don't care cells, where the patterns that count for two edits have double entries. Note that SieveMem relays the inputs to the output without additional mutations. Therefore, a mask might be required for the output to ignore the unused entries.

5.2.3. BANDEDKRAIT ALGORITHM

SieveMem is capable of supporting simple, shared kernels in pre-alignment filters. Therefore, SieveMem can support any future algorithm that uses similar kernels with different control sequences. To show this, we devise a simple algorithm called BandedKrait[2,3].

BandedKrait reduces costly DP problem to a simpler exact matching problem between (shifted versions of) smaller segments of read and reference. To this end, BandedKrait divides the read sequence into segments of k-bps and compares them to the corresponding segments of the reference and its shifted versions. BandedKrait checks each pair for an exact match, and the results determine whether an edit is present within that segment. The repetition of this process for references shifted by $-E$ to $+E$ ensures that BandedKrait supports up to E deletions and/or insertions. BandedKrait uses the pigeon-hole principle on the combined results of segments to approximate the #edits in

[2] The banded krait (Bungarus fasciatus) is a species of elapid snake easily identified by its alternate black and yellow crossbands, all of which encircle the body.

[3] Recently, Shahroodi et al. [117] propose RattlesnakeJake, a hardware/software (HW/SW) co-designed accelerator based on Computation-In-Memory (CIM) paradigm, capable of pre-alignment filtering for short-sequence alignment. The software algorithm behind RattlesnakeJake is similar to BandedKrait. However, the hardware design of RattlesnakeJake and SieveMem differ. Since RattlesnakeJake is tightly integrated with its hardware, we separate BandedKrait and RattlesnakeJake to be able to evaluate the algorithm on SieveMem later on fairly

the sequence pairing. Algorithm 2 summarizes BandedKrait, where k is the segment size and E is the number of permissible edits between input read and the reference sequence.

Algorithm 2 BandedKrait Algorithm

Input: Read, Reference, E, ReadLength, k
Output: Accept

```
1:  N_segment ← ⌈ReadLength/k⌉
2:  Matches ← 0
3:  for i ∈ {0 : N_segment − 1} do
4:      Match ← 0
5:      for e ∈ {−E : +E} do
6:          ReadSegment ← Read[i × k : (i + 1) × k − 1]
7:          ReferenceSegment ← Ref[i × k + e : (i + 1) × k − 1 + e]
8:          if ReadSegment == ReferenceSegment then
9:              Match ← 1
10:         end if
11:     end for
12:     Matches ← Matches + Match
13: end for
14: Accept ← (Matches >= N_segment − E)
15: return Accept
```

BandedKrait flexibly explores two trade-offs: (1) accuracy vs. hardware-friendliness and (2) required resources vs. achievable parallelism or performance. Exact matching is known to be well-supported in hardware and specifically in a CIM-enabled crossbar. However, using exact matches as proximity to existing errors (compared to alternatives such as DP or SNR sub-problems in SneakySnake, for example) underestimates the number of edits. Therefore, more reads can pass BandedKrait, making it inaccurate. Moreover, finding the exact matches between each segment pair and the shifted variants are independent, parallelizable problems. However, exploiting that demands higher resources.

5.2.4. BANDEDKRAIT ON SIEVEMEM (MEM-BANDEDKRAIT)

BandedKrait is completely supported by SieveMem. If implemented on SieveMem, we call the design Mem-BandedKrait. Fig. 5.8 presents an example of how the two TCAMs in SieveMem are filled so that it can support the pattern required by BandedKrait algorithm. Mem-BandedKrait repeats this process for all $2E+1$ shifted reference segments. If no exact match is detected in any of the iterations, Mem-BandedKrait counts an error for that segment.

5.3. EVALUATIONS

5.3.1. EVALUATION METHODOLOGY

Implementation & setup. We implemented SieveMem in a cycle-accurate RTL-based simulation platform. We verify the design by comparing the simulation results with SieveMem's software outputs. We use the same implementation for the evaluation of Mem-BandedKrait. SieveMem hardware uses memory models from a small RRAM crossbar in TSMC 40 nm CMOS technology [330, 333]. These memories are provided to us by generous partners from the EU project MNEMOSENE [333]. The additional components of SieveMem discussed in Section 5.2.1 are also designed using TSMC 40 nm technology node in Synopsis Design Compiler [331]. We integrated the latency numbers into the

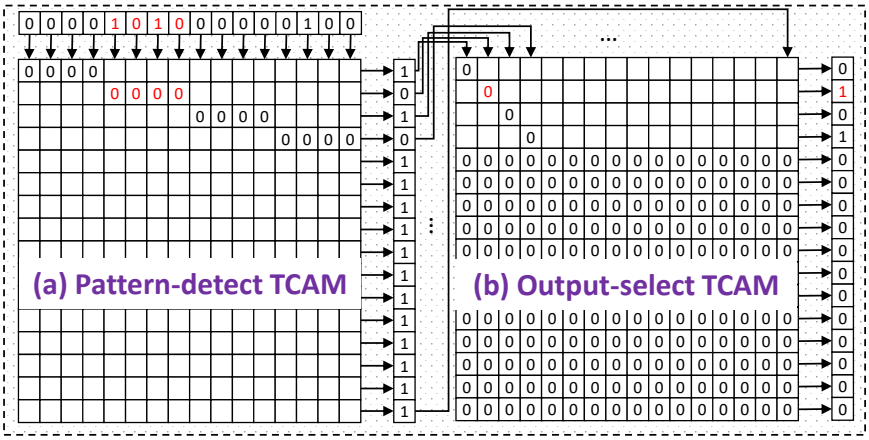

Figure 5.8: TCAMs values in SieveMem to support BandedKrait.

simulation platform. We run our experiments on a 12-core server with 16 GB memory, Tesla-K80 GPUs, and a Intel® Xeon® CPU E5-2680 operating at 2.4 GHz. Our evaluations consider the same platform and input datasets for all filters for a fair analysis.

Baselines. We compare different kernels supported in SieveMem with their accelerated version in literature. We also compare Mem-BandedKrait and BandedKrait with SneakySnake (SS) [186], Shouji [187], SHD [189], and GRIM-Filter [95], as SotA pre-alignment filters. We use open-sourced implementations of these filters. We consider SneakySnake on both CPU and GPU, Shouji and SHD on FPGA, and GRIM-Filter on 3D-stacked memories.

Datasets. We use real genome datasets, i.e., $human_g1k_v38$. Since SieveMem is designed for supporting filters (and their kernels) for short-reads, we use two sample sets [186] of $ERR240727_1$ and $SRR826471_1$ from Illumina reads [351] with read lengths varying from 100bps to 250bps, respectively. For end-to-end evaluation of alignment (in the case of comparing Mem-BandedKrait with other filters), we use Edlib [350] to create full-alignment results for accuracy. Edlib results also verify the functionality of filters.

5.3.2. EXECUTION TIME OF SUPPORTED KERNELS

Fig. 5.9 and Fig. 5.10 compare the execution time of performing the same number of HMC and SPD operations on CPU and SieveMem, over our two datasets for different edit distances. We choose SHD as the reference filter to align with the example we provided over SieveMem supporting SHD in Section 5.2.2. The results consider the necessary data movement. The y-axis has been limited to a low number (30 and 50) for improving readability. Since no approximation is used for these kernels, no accuracy loss occurs due to the underlying platform, and all three versions produce the same result.

We observe that SieveMem accelerates both HMC and SPD irrespective of the dataset. This improvement goes up to 331.1× and 446.8× for HMC and SPD, respectively, over the datasets. This is expected for two reasons: (1) fewer data movement

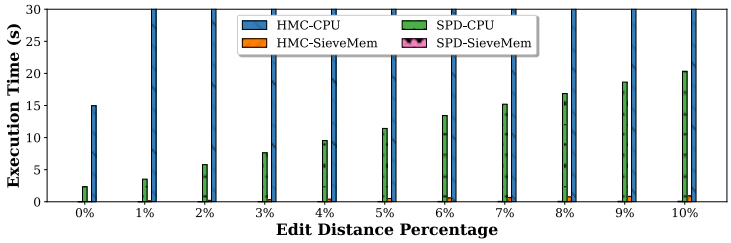

Figure 5.9: HMC and SPD on ERR240727_1 with E=40.

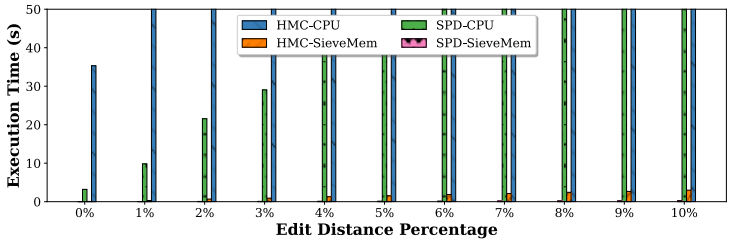

Figure 5.10: HMC and SPD on SRR826471_1 with E=100.

in SieveMem and (2) parallel computation of target operation with a reasonably high clock cycle. We conclude that not only SieveMem supports the shared kernels of pre-alignment filters, but also it significantly accelerates them.

5.3.3. FILTERING ACCURACY

Fig. 5.11 and Fig. 5.12 compare the false positive (FP) rate of several filters. FP rate in a filter shows the ratio between reads that wrongly pass the filter (i.e., could have been filtered) and go through alignment (i.e., DP) over all the reads. The lower the FP, the better. Note that in terms of True Positive (TP) and True Negative (TN) rates (the other two important metrics for the accuracy of a filter), BandedKrait and Mem-BandedKrait achieve the same rate as SneakySnake, which currently results in the best filtering rates.

We make three key observations. First, BandedKrait and Mem-BandedKrait provide low FP rates irrespective of edit threshold and dataset. In fact, their FP rates are on par with the SotA SneakySnake. Second, the FP rate of Mem-BandedKrait and BandedKrait is less than 1% apart; i.e., the hardware limitation regarding the reference's start point, which affects #segments, does not affect the accuracy significantly. Third, BandedKrait outperforms Shouji, SHD, and GRIM-Filter by providing, on average, 22%, 40%, and 90%, respectively, fewer number of falsely-accepted sequences. We conclude that Banded-Krait is an effective and accurate filter on both CPU and SieveMem architecture.

5.3.4. FILTERING SPEED

Fig. 5.13 and Fig. 5.14 present the execution time for different filtering methods over different edit threshold. The y-axis uses a logarithmic scale.

We make two key observations. First, although BandedKrait requires more time than

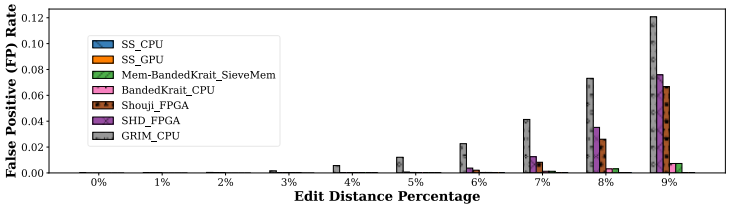

Figure 5.11: FP rate on ERR240727_1 with E=40.

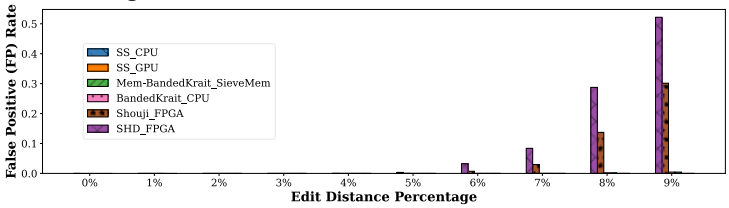

Figure 5.12: FP rate on SRR826471_1 with E=100.

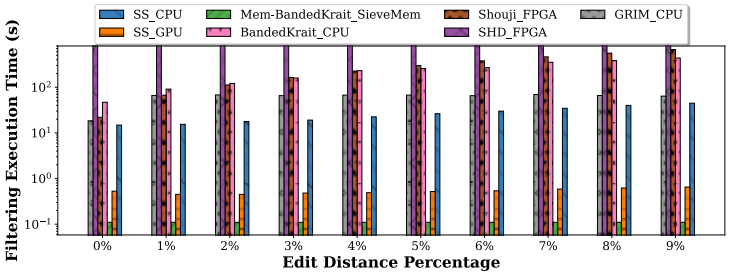

Figure 5.13: Filtering speed on ERR240727_1 with E=40.

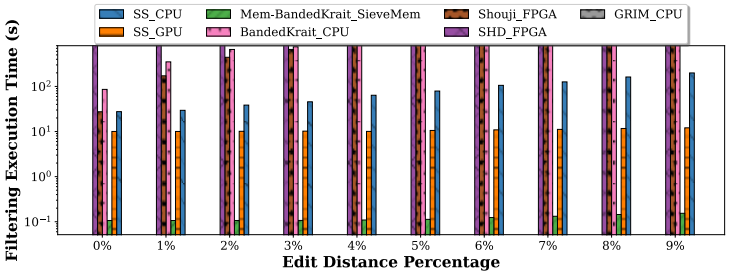

Figure 5.14: Filtering speed on SRR826471_1 with E=100.

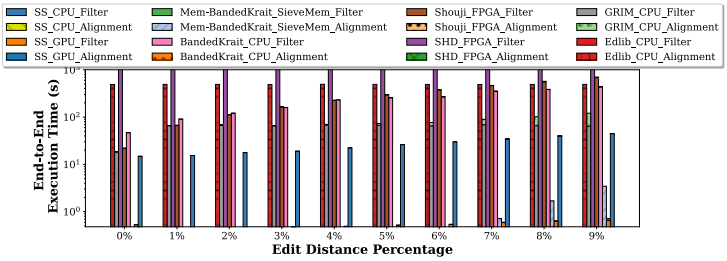

Figure 5.15: End-to-End speed on ERR240727_1 with E=40.

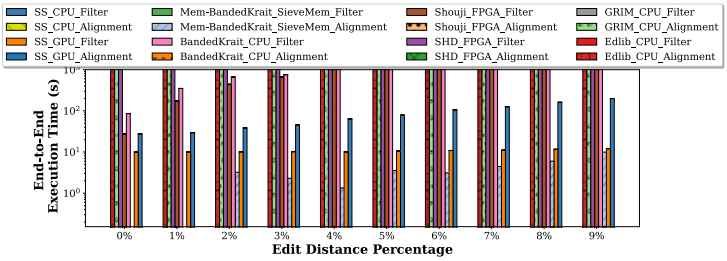

Figure 5.16: End-to-End speed on SRR826471_1 with E=100.

some SotA filters, Mem-BandedKrait significantly outperforms the fastest SotA filters on CPU (SS_CPU) and on GPU (SS_GPU) by up to 1292× and 95.5×, respectively, when processing the same amount of sequences. Note that SS_CPU and SS_GPU outperform all previous existing filters, independent of the dataset and edit threshold.

Second, Mem-BandedKrait provides better scalability for larger short reads (SRR826471 vs. ERR240727) than other methods. For example, the average speedup of Mem-BandedKrait over SS_CPU is 2.93× more on SRR826471 than on ERR240727 for the same edit threshold of E=5. This is because the performance on Mem-BandedKrait is only slightly affected by the length of inputs and target edit threshold, while, in original SS_CPU this change is more significant.

We conclude that BandedKrait and Mem-BandedKrait effectively reduce the execution time of filtering for the same #processed read and reference sequences.

5.3.5. END-TO-END ALIGNMENT SPEED

Fig. 5.15 and Fig. 5.16 present the execution time for end-to-end alignment for a combination of filters with Edlib for alignment over several edit thresholds. We limit the y-axis that shows the execution time of filter+alignmnet to 1000s to capture better the trends and relative execution time of in the system compared to other methods. The y-axis is in logarithmic scale.

We make two key observations. First, Mem-BandedKrait significantly reduces the end-to-end execution time of sequence alignment irrespective of the dataset or edit threshold. The improvements are so profound that they are hard to capture, even on the logarithmic scale. Particularly, the improvement in filtering translates to a 254.6× and

91.4× improvement in end-to-end alignment time compared with SotA filter combined with SotA alignment and sheer Edlib on CPU, respectively, averaged over our datasets.

Second, similar to the comparison of filters in Section 5.3.4, the speedup in the end-to-end alignment is higher for SRR826471 compared to that on ERR240727. For example, the average speedup of alignment using Mem-BandedKrait is 9.86× more on SRR826471 compared to on ERR240727 for edit threshold of E=5. This is due to the effect of filtering being even more effective on the SRR826471 dataset compared to the ERR240727 dataset as discussed in Section 5.3.4.

We conclude that Mem-BandedKrait is a fast pre-alignment filter and effectively reduces the execution time of end-to-end alignment such that it takes a step towards mitigating the filtering bottleneck.

5.4. DISCUSSIONS AND FUTURE WORKS

5.4.1. SIEVEMEM FOR LONG SEQUENCE ALIGNMENT

From the conceptual point of view, the BandedKrait algorithm is also effective for long sequence alignment, where sequences are a size of 100Kbp. However, when it comes to mapping to SieveMem, distributing the long reference or read sequences in the memory hierarchies requires complex bookkeeping, different buffer sizes, control unit sequences, and potentially some additional logic. However, to the best of our knowledge, pre-alignment filters are not currently deployed for long-read sequence alignment acceleration. We leave the exploration of BandedKrait on SieveMem for long pre-alignment filtering to future work.

5.4.2. POTENTIAL DESIGN EXPLORATIONS

SieveMem's best configuration. Current evaluations of SieveMem are based on the measurements on a small ReRAM chip prototype for tiles and TCAMs. However, a complete design space exploration is required to direct the final configuration of SieveMem before deploying it in future genomics systems. Such exploration should consider different inputs, memory units and arrangements, variations of devices, and circuit behavior (e.g., non-idealities) for different organizations. We leave this to an extended report.

Other memory technologies. We design SieveMem assuming memristors as the underlying technology due to the benefits they offer in terms of density, low power, and support for logical vector operations (please see Section 2.1 for more details). However, independent and separate works [93, 352, 353] propose supporting the same operations (XNOR, associate search, etc.) in other technologies as well. Having a complete comparison of SieveMem's versions with different technologies is an interesting work we leave for the future.

5.5. CONCLUSION

This chapter proposes a memristor-based CIM-enabled architecture for pre-alignment filters called SieveMem to (1) accelerate shared kernels in pre-alignment filters and (2) prevent unnecessary data movement for sequence alignment by filtering dissimilar short sequences inside the main memory. The chapter also discusses a CIM-friendly algorithm for pre-alignment filtering called BandedKrait that is suitable for implementa-

tion on SieveMem. Considering a larger genomics pipeline, accelerated BandedKrait on SieveMem is fast enough to shift the processing bottleneck back (again) to the DP step of the remaining sequences. Hence, our work demands even more accurate pre-alignment filtering and/or better DP-based alignment algorithms.

6

FILTERFUSE

With the industry moving towards sequencing of long reads (as they favor accurate and more efficient reconstruction of DNA), finding solutions that support efficient analysis of long reads becomes more necessary. The long execution time required for sequence alignment of long reads negatively affects genomic studies relying on sequence alignment. Although pre-alignment filtering as an extra step before alignment was recently introduced to mitigate sequence alignment for short reads, these filters do not work as efficiently for long reads. Moreover, even with efficient pre-alignment filters, the overall end-to-end (i.e., filtering + original alignment) execution time of alignment for long reads remains high, while the filtering step is now a major portion of the end-to-end execution time.

This chapter makes three contributions. First, it identifies data movement of sequences between memory units and computing units as the main source of inefficiency for pre-alignment filters of long reads. This is because although filters reject many of these long sequencing pairs before they get to the alignment stage, they still require to pay a huge cost regarding time and energy consumption for the large data transferred between memory and processor. Second, this chapter introduces an adaptation of a short-read pre-alignment filtering algorithm suitable for long reads. We call this LongGeneGuardian. Finally, it presents FilterFuse as an architecture that supports LongGeneGuardian inside the memory. FilterFuse exploits the Computation-In-Memory computing paradigm, eliminating the cost of data movement in LongGeneGuardian.

Our evaluations show that FilterFuse improves the execution time of filtering by 120.47× for long reads compared to state-of-the-art (SotA) filter, SneakySnake. FilterFuse also improves the end-to-end execution time of sequence alignment by up to 49.14× and 5207.63× compared to SneakySnake with SotA aligner and only SotA aligner, respectively.

This chapter is partially based on the candidate's works [119, 354].

As discussed in Section 2.2.1, sequence alignment is a pivotal process in genomics studies identifying similarities and differences in DNA, RNA, or protein sequences. By highlighting conserved regions and mutations, sequence alignment provides profound insights into the molecular function and evolution [184, 185]. SotA sequence aligners employ dynamic programming-based (DP) algorithms to achieve high accuracy with the cost of long latencies and energy inefficiencies, particularly when applied to large DNA sequences. These limitations directly affect the medical studies that benefit from sequence alignment.

Long reads and short reads are two types of sequencing reads used as inputs in the sequence alignment and are produced by different sequencing technologies [355–361]. These two types of reads differ in their length, error rate (random errors due to technology used in obtaining them) or accuracy, application, usability, and cost. Overall, both long and short reads have their strengths and weaknesses, and the choice of sequencing technology depends on the specific research question and experimental design. However, although long-read sequence alignment faces several challenges (e.g., high error rates in long-read technologies, computational complexity due to longer read lengths, and issues with reference genome bias and uniqueness), currently, the industry is moving towards long reads [360, 362–364]. This is because of the ability of long-read sequence alignment to resolve complex genomic regions, identify structural variations, and aid in epigenetic studies. Therefore, devising algorithms and/or hardware that can accelerate long-read sequence alignment while accurately mapping long reads to reference genomes and handling the unique characteristics of long-read datasets is of utmost importance in the coming years.

Previous works typically took two directions to address the inefficiency in sequence alignment for long reads [293, 362–367]. First, some works simplified the process using better sketching or chaining algorithms or heuristics for dynamic programming (DP) part of the general alignment algorithms [293, 366, 367]. This backtracking step involves irregular memory access patterns that are challenging for hardware implementation. Second, some works [95, 186] propose a filtering step before alignment, called pre-alignment filtering[1], to significantly speed up the end-to-end sequence alignment of (long) reads by heuristically replacing the need for expensive DP solutions for many inputs in the first place. These filters use a pre-defined edit distance threshold between the inputs and quickly determine whether or not an alignment (i.e., DP) should be granted. SotA pre-alignment filters [186] speed up the sequence alignment so much so that they themselves become the (next) bottleneck in the end-to-end sequence alignment procedure. Therefore, there is a need for a more efficient design to tackle the filtering bottleneck in the sequence alignment pipeline of long reads.

Unfortunately, we identify three shortcomings in pre-alignment filtering for target long reads by the industry. First, there is currently only one single filter, SneakySnake [186], that supports pre-alignment filtering for long reads. SneakySnake accelerates the pre-alignment filters for short reads on central processing units (CPUs), graphics processing units (GPUs), and field-programmable gate arrays (FPGAs). However, only the CPU version supports long reads due to strict assumptions on data and heuristics on the GPU and FPGA versions. Second, on both types of reads, even a SotA filter, e.g.,

[1] We use the term filter and pre-alignment filter interchangeably hereafter.

SneakySnake, becomes the new computational bottleneck when considering the end-to-end alignment process (i.e., filtering step + sequence alignment step) [95, 186]. Third, data movement bottlenecks the performance of SotA pre-alignment filters, particularly for long reads. This means that filters spend more time on moving sequencings from memory units to processing units compared to the time they spend on performing the computations necessary for the filtering. This shortcoming is important because most of the sequence pairs that go to the pre-alignment filters turn out to be unnecessary and will be filtered out eventually [186]. Therefore, there is a need for a design that can overcome these shortcomings and resolve the bottleneck by avoiding wasted work (i.e., time and energy consumption) caused by data movement in the system.

We propose LongGeneGuardian, a lightweight and memory-friendly pre-alignment filtering algorithm that supports long reads and performs on par with SotA pre-alignment filters regarding accuracy metrics. We then present FilterFuse, which is a hardware/software co-designed (HW/SW co-designed) accelerator based on Computation-In-Memory (CIM) that supports LongGeneGuardian for long reads. The simplicity of LongGeneGuardian and its minimum assumptions on data placement inside a typical memory array makes it compatible with a restricted yet realistic CIM design. FilterFuse architecture is memory/technology independent, i.e., the memory arrays can be of any memory technology, such as dynamic random access memory (DRAM) or resistive random-access memory (ReRAM), as long as they support a few key operations such as logical XOR and associative search.

Our results show that LongGeneGuardian achieves an accuracy on par with SneakySnake, the SotA filter, for the long-read filtering. LongGeneGuardian does not introduce any extra false negatives in the filtering process and does not replace the sequence alignment. Therefore, one can still employ LongGeneGuardian with any sequence aligner. When accelerated with memristor-based memory components, FilterFuse accelerates the filtering step by up to 120.47× over SneakySnake, for the same read long-read dataset. Our evaluations show that the using FilterFuse for pre-alignment filtering accelerates the end-to-end alignment by up to 49.14× and 5207.63×, compared to the case of SneakySnake+long-read aligner and a standalone long-read aligner, respectively.

We make the following contributions in this chapter:

- LongGeneGuardian: A lightweight filtering algorithm with no assumption on data arrangement in a memory crossbar, making it suitable for a CIM design.

- FilterFuse: A configurable CIM realization of LongGeneGuardian for long-read pre-alignment filtering inside the memory. FilterFuse is compatible with any memory technology that can inherently support logical vector XOR operation and associative search.

- An extensive evaluation of LongGeneGuardian and FilterFuse regarding the accuracy, execution time, and power consumption on different memory technologies using real data against previous SotA pre-alignment filters.

6.1. MOTIVATION

6.1.1. LONG READS VS. SHORT READS

Different sequencing technologies produce reads with various features regarding accuracy and read length. The accuracy of a sequencing machine is described as the percentage of base pairs it has correctly extracted from a DNA sample. The read length is the number of base pairs that constitute it. These two metrics determine the alignment threshold (Section 2.2.1) and the edit margin, also known as the region of interest (ROI), in the evaluation of pre-alignment filters[2].

The read accuracy matters as sequencing aims to compare samples of DNA to find differences/similarities between them. For this, we need to be able to differentiate between actual differences between the samples and sequencing errors. Sequencing errors are defined as differences between the extracted read and the DNA sample. These errors originate from deficiencies in sequencing technologies. Having highly accurate reads is favorable.

For two main reasons, it is favored to have long reads as long as the length does not (significantly) hurt the accuracy. First, long reads simplify the reconstruction of the original DNA compared to separate shorter reads. Like fitting pieces of a puzzle together, it is easier to do this with fewer long reads than many short reads. Second, the probability of a short read aligning with multiple parts of a reference genome is much higher than is the case with long reads. Consequently, finding the source of a mutation in the DNA is much harder with short reads.

Therefore, although short reads remain popular due to their availability in genomics libraries, the industry is moving towards accurate, long-read sequencing.

6.1.2. LIMITATIONS OF SOTA FILTERS FOR LONG READS

Support for long reads. Although previous works [95, 186–189] propose various methods of pre-alignment filters, their methods rarely work on both types of reads due to underlying assumptions on input data or how to determine the (approximate) similarity of two sub-strings. Specifically, among all the proposed filters, only SneakySnake supports long reads.

Filter is the new bottleneck. We profile SneakySnake as the filter and Parasail as a sequence aligner over different percentages of edit distances and datasets. We refer to Section 6.4 for detail on our evaluation methodology.

Fig. 6.1 presents the execution time for end-to-end alignment for our two representative long-read datasets. Note that the bar for "Aligner+CPU-Filter" is always 0, while the other three bars exist depending on the inputs and the filter's effectiveness. For better readability, we limit the results to the edit thresholds that benefit the most from a filtering step, as stated in the original SneakySnake [186]. The y-axis is in logarithmic scale.

We make two main observations. First, a pre-alignment filter improves the end-to-end execution time by up to 158.76×. This shows that this filtering is effective for long reads. Second, the majority of the new end-to-end execution time is spent on filtering rather than alignment. This majority is high enough to the extent that the alignment

[2] For long reads, the ROI is 2-7% of the read length [187, 188, 349].

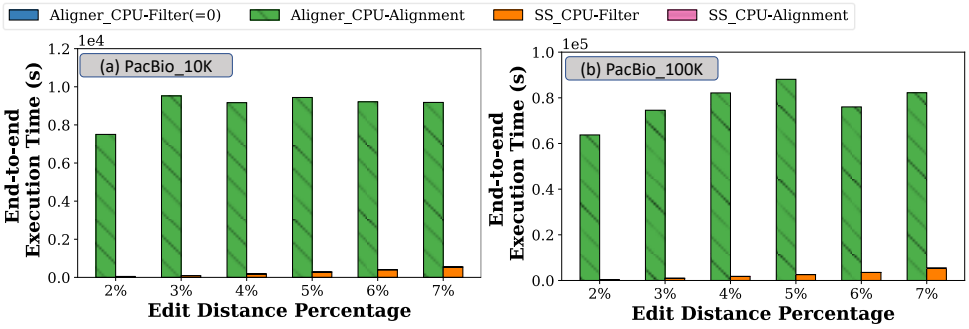

Figure 6.1: Contribution of SneakySnake to the end-to-end execution time for long reads.

time is almost invisible compared to the filtering time, even with a logarithmic scale. This shows that to improve the end-to-end execution time, the filtering step is the new bottleneck to focus on.

Data movement limits accelerated filters. Fig. 6.2 presents the breakdown of execution time for a SotA accelerated filter, SneakySnake on GPU (Snake-on-GPU). Since Snake-on-GPU does not support long read filtering, we broke down the sequences into smaller non-overlapping chunks (with some post-processings on the host side) to achieve a rough estimation of the contribution of data movement to the total execution time in a hardware accelerator.

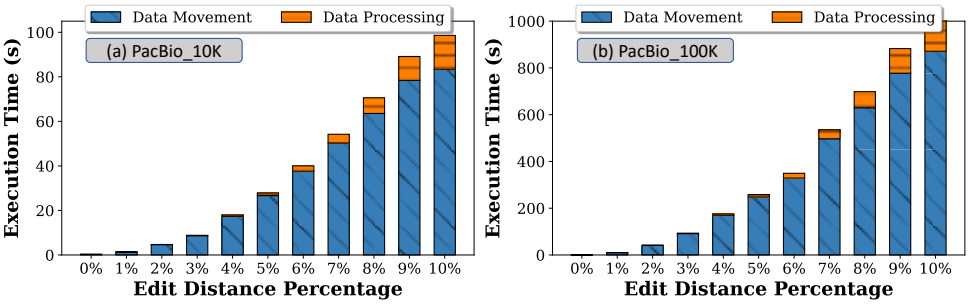

Figure 6.2: Contribution of filter to the end-to-end execution time for long reads.

We observe that Snake-on-GPU spends a minimum of 60% and up to 98% of its execution time just transferring data from memory to GPU. This shows that data movement constitutes the major part of the execution time for an accelerated filter.

No hardware acceleration support for long reads. SneakySnake [186] currently holds the highest accuracy and lowest execution time among all the existing pre-alignment filters. It supports both GPUs and FPGAs, designs called Snake-on-GPU and Snake-on-Chip, respectively. However, the open-sourced implementations of Snake-on-GPU and Snake-on-Chip only support short reads due to some heuristics and hard-coded assumptions in their implementations. Therefore, to this day, there is no hardware ac-

celerator for long-read pre-alignment filtering.

Why not SneakySnake (SotA filter) on CIM? Unfortunately, the SneakySnake is not CIM friendly for two reasons: (1) it requires support for leading zero count (LCZ) or at least flexible #shifts in data, which is costly in memory arrays, and (2) it requires perfect positioning of data inside memory units in respect to the boundaries of memory tiles.

Note that even the most hardware-friendly implementation of SneakySnake, Snake-on-Chip, introduces two main challenges for a CIM implementation:

- Snake-on-Chip requires the computation of an entire chip-maze of each sub-problem. The horizontal dimension of this chip maze is dependent on the number of bases of each sub-problem, and the vertical dimension is dependent on the examined edit distance. While the maze size is manageable for short reads, the resources required to store the chip maze for long reads would be too large to implement in a memory tile (the granularity we can expect a CIM design work with). For example, for long reads that reach up to 100Kbps and have edit-distance thresholds of up to 25%, required to process PacBio CLR reads, the chip maze would have to store up to 50 thousand rows of data per tile.

- Snake-on-Chip takes several iterations to find the optimal path through the chip maze. This process is time-consuming and requires a relatively large number of lookup tables to be implemented inside memory.

Takeaway

We need efficient accelerators for pre-alignment filters of long reads with an emphasis on eliminating data movement.

6.2. LONGGENEGUARDIAN ALGORITHM

We propose reconstructing a recent and less accurate algorithm, RattlesnakeJake [117], designed for short-read pre-alignment filtering. We call our variation LongGeneGuardian to clear the distinction in the following explanations. LongGeneGuardian draws inspiration from converting the problem into sub-problems of Snake-on-Chip but does not require the generation of the chip maze. LongGeneGuardian also simplifies the traversal of the chip maze, reducing the need for additional resources. Unlike RattlesnakeJake, LongGeneGuardian creates the shifted sub-sequences by shifting both read and reference segments to left and right, rather than keeping the read untouched and only shifting the reference.

LongGeneGuardian is based on three key observations:

1. If two strings differ by e edits, then all non-erroneous characters of the strings can be aligned in at most e shifts.

2. If two strings differ by e edits, then they share at most $e+1$ identical sections.

3. When you are interested in comparing two strings where you can shift one to the left by up to e_{total} shifts while keeping the other one fixed, you can achieve the same results by shifting the first one to the left by e_{left} and the second string to the right by $e_{right} = e_{total} - e_{left}$.

LongGeneGuardian exploits these observations and creates $2e+1$ shifted Hamming masks to account for e shifts to the left and right. Similar to Snake-on-Chip, LongGeneGuardian divides the problem of performing Hamming masks on these shifted versions into the simpler sub-problems of length T, called segments hereafter.

However, LongGeneGuardian differs from Snake-on-Chip in that it does not count the number of edits in each segment but detects any edit's presence. In doing so, the results of the two algorithms only differ if multiple errors occur in the same segment of the read. This leads to inaccuracy in LongGeneGuardian due to the abstraction of actual #edits in segments. However, our evaluations using real datasets in Section 6.5 show that it is unlikely that two edits exist in the same segment when the segments are small. Moreover, each read contains a small #edits relative to its read length. Therefore, the decrease in accuracy is still acceptable as LongGeneGuardian can still distinguish true mappings (similar) from obviously false mappings (dissimilar) and provide enough speed-up (Section 6.5).

LongGeneGuardian detects edits in the segments using this intuition: if the section of the read that is processed in one sub-problem contains no edits, at least one of the Hamming masks of that segment must be free of errors. This means that LongGeneGuardian can check whether any of the Hamming masks belonging to the segment contains only '0's. This allows the detection of #segments without edits. By subtracting this from the total #segments, LongGeneGuardian finds #segments that do contain errors.

A key advantage of LongGeneGuardian's approach is that every Hamming mask corresponding to the different shifts can be efficiently processed independently. This removes the need to collect all Hamming masks to create the chip-maze and removes the iterative nature of the chip-maze traversal step. This and the segmentation into sub-problems make LongGeneGuardian particularly suitable for CIM.

6.3. FILTERFUSE ARCHITECTURE

We implement LongGeneGuardian using CIM, called FilterFuse. While FilterFuse is designed to support long reads, it remains flexible and supports a wide range of data sets, edit-distance thresholds, and even short-reads filtering algorithms.

6.3.1. FILTERFUSE OVERVIEW

Fig. 6.3 presents an overview of the FilterFuse.

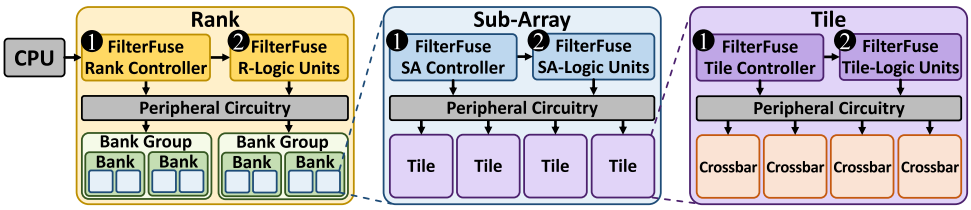

Figure 6.3: Overview of FilterFuse.

FilterFuse follows the typical memory hierarchies (i.e., bank groups, banks, sub-

arrays, and tiles) found in conventional memories to improve resource utilization. However, FilterFuse augments various hierarchy levels with small specialized controllers ❶ and logic units ❷ to enable the required operations. The controller at each level is implemented as an finite-state machine (FSM). It controls all the logic units and the lower-level controllers.

6.3.2. TILE ARCHITECTURE

Tiles are the lowest and one of the most critical architectural levels in FilterFuse. Fig. 6.4 presents the architecture of a tile.

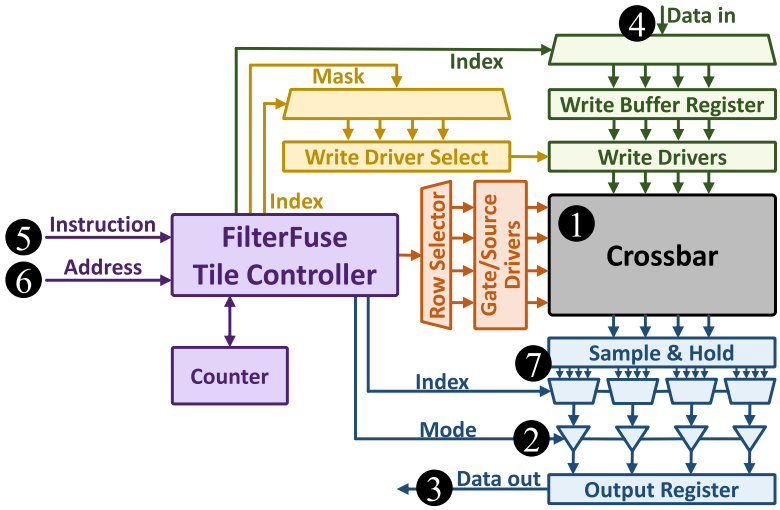

Figure 6.4: Overview of the tile architecture.

Each tile is an array of memory cells forming a crossbar structure ❶. FilterFuse supports any memory technology for its cell as long as it can support logical vector operation in the SA. Peripheral circuits include multiple write drivers, sense amplifiers (SAs ❷), row/column decoders, and multiplexers. The SAs are modified for the required logical vector operation, e.g., for DRAM or memristor-based crossbars, the SAs are based on Ambit [93] and Scouting Logic [87, 106], respectively.

Each tile has an n-bit data output (❸) and three inputs provided by the sub-array controller (besides the clock and reset signal):

- The 'data-in' ❹: n binary bits to be written to the crossbar.
- The instruction signal ❺: to determine the behavior of the tile controller, selecting whether the tile should be idle, read, write, or perform an XOR operation.
- The address signal ❻: to index the correct rows and columns of the crossbar to/from which the data should be written/read.

To execute LongGeneGuardian, FilterFuse first writes the read sequence to the appropriate memory locations. It then performs an XOR between n-bits of the read and

n-bits of the reference sequence (which is already written in the memory), where n indicates the number of SAs in the tile.

The peripheral components interface between the digital architecture and the (analog) crossbar. They also act as intermediate storage of input data to overcome the difference in timing between the read/write time of the crossbar and the clock period of other digital components.

If needed, in the case of memristor-based crossbars, for example, FilterFuse implements interleaving within the column multiplexing logic. The tile architecture indexes a series of de-multiplexers at the output of the sample-and-hold circuit (❼ in Fig. 6.4) to select the correct memory rows. This way, the correct digital output of the SAs is placed in an output register and can be accessed by the sub-array controller.

6.3.3. SUB-ARRAY ARCHITECTURE

Fig. 6.5 presents an overview of FilterFuse's sub-array architecture.

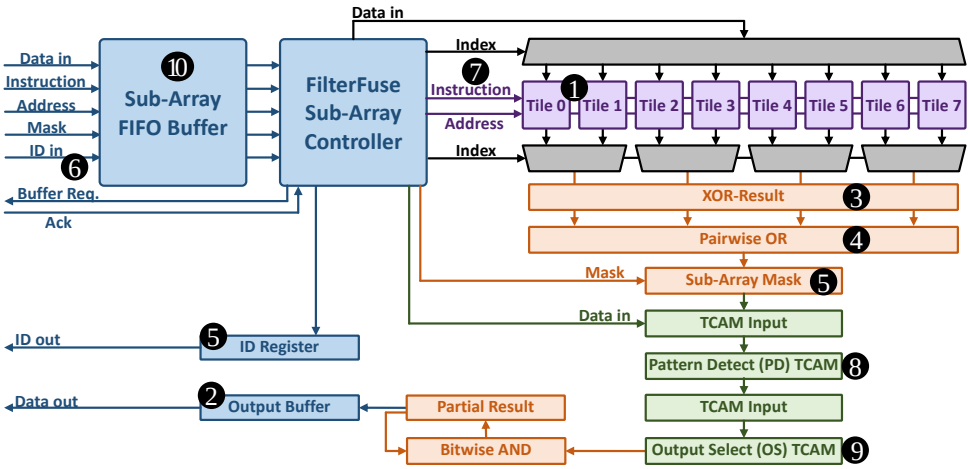

Figure 6.5: Overview of the sub-array architecture.

The sub-arrays in FilterFuse are the second main computational units, where multiple tiles (❶) are grouped together to execute the complete logic. Sub-arrays contain input and output buffers (❷ and ❿) required to reduce the stalling of the pipeline when the sub-array or output bus is occupied. Three main tasks of a sub-array are:

- Translating the results of bitwise XOR from tiles (❸) into base pair results (❹).
- Performing masking on tile results that are not part of the read/reference pairings (❺).
- Tracking what read/reference pairing is being processed using an ID signal (❻).

A sub-array takes seven steps to perform the necessary computation for a final bit-vector:

1. Passing the XOR instruction to the tiles to perform the operation between the read sequence and one of the shifted reference sequences (❼).

2. Retrieving the results from the output buffers of the selected tiles by indexing a series of multiplexers.

3. Combining the results into an XOR-result register (❸).

4. Performing a series of OR to convert a bit-level XOR result to a bit vector of base pair level result (❹).

5. Masking parts of the OR result that are not part of the actual read-reference pairing by AND with a sub-array mask (❺).

6. Querying a ternary content addressable memory (TCAM), called Pattern-Detect TCAM (PD-TCAM), with the masked results of the previous step (❽).

7. Querying a second TCAM, called Output-Select TCAM (OS-TCAM), with the results of PD-TCAM (❾).

Note that the length of the operands at the sub-array level is determined by the minimum sub-problem size, i.e., $2 * T$ (where T is the segment size in LongGeneGuardian) to account for the common 2-bit encoding scheme commonly used in genomics accelerators [234, 368–370].

FilterFuse uses PD-TCAM and OS-TCAM to detect patterns of '1's and '0' in its input and construct one iteration of the final bit-vector based on patterns detected by the PD-TCAM, respectively. This is necessary for LongGeneGuardian and many previous pre-alignment filters. We refer the reader to the discussion over these components in Chapter 5.

FilterFuse boosts filtering throughput by activating multiple sub-arrays in parallel to compute different read-reference pairings independently. To enable this parallel execution, FilterFuse stores a different part of the reference on each sub-array and uses the input dataset to determine which sub-arrays are required for the computation.

It is possible that a read-reference pairing requires a sub-array that is still busy computing a different pairing which creates contention over the sub-array, i.e., it stalls FilterFuse until the sub-array is freed up. To combat this contention, each sub-array contains a first in first out (FIFO) buffer (❿ in Fig. 6.5) that stores all the input signals for the computation of a read-reference pairing until the sub-array finishes computing its current pairing. This eliminates the need to stall the rest of the pipeline until the FIFO buffer is full. Note that having a FIFO buffer, the order in which the computation of pairings finishes might differ from how they are supplied to the tiles. To prevent any issue, FilterFuse assigns an ID to each pairing, which is also presented alongside the sub-array output (❷). To prevent the potential contention over the output bus, which occurs when multiple sub-arrays finish their computation simultaneously, FilterFuse uses a request-acknowledgment scheme, ensuring that outputs are read one after another.

6.3.4. Bank and Bank-Group Architecture

FilterFuse groups several sub-arrays as banks and then groups multiple banks into a bank group. This type of hierarchical structure is often found in conventional DRAM-based technologies [371]. Although these two levels in FilterFuse have similar functionality, FilterFuse adapts it for two main reasons. First, splitting the two levels reduces the fan-out of the required busses of each stage, which reduces the clock period required. Second, the multi-level approach improves the contention over the lower-level resources without adding excessive amounts of buffer overheads. These levels implement an acknowledgment scheme to determine the need for a stall due to the contention. Because the architecture consists of several layers, communication between the top level and the sub-array level happens over several clock cycles. The input buffers reduce this latency by providing an acknowledgment signal after only a single clock cycle.

6.3.5. Rank Architecture

The rank level is the highest level of FilterFuse that interfaces between the host device and FilterFuse. Fig. 6.6 provides a high-level overview of the rank-level architecture.

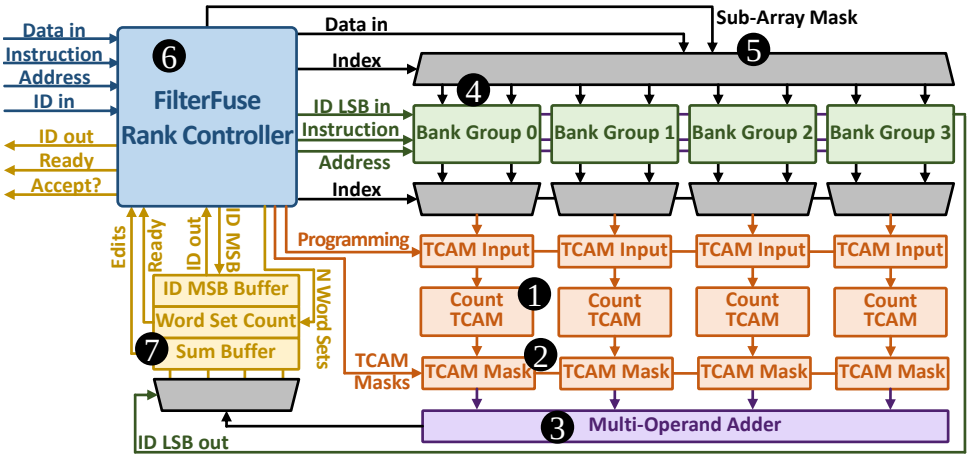

Figure 6.6: Overview of the rank architecture in FilterFuse.

FilterFuse performs three key tasks at the rank level:

- Processing inputs from the host device in the correct format and sending them to the appropriate memory locations.
- Providing instructions, addresses, data, IDs, and masks to control the lower levels, stalls, and the results.
- Tracking read/reference pairings and implementing the edit-counting, summation, and comparison to the edit-distance threshold.

At the rank level, FilterFuse uses multiple Count-TCAMs (❶) to collect the results of bank groups and calculate/count #edits in a word set of the input read. A Count-TCAM

detects the pattern and assigns the edit number of the detected patterns. Currently, FilterFuse uses a 4-bit wide TCAM for each Count-TCAM. This is also compatible with patterns in previous filters such as SHD, which required splitting the final bit-vector into segments of k=4 bits. Final masking on the output of Count-TCAMs (❷) and a multi-operand adder (❸) finalize the discovery of edits in the input read via FilterFuse based on LongGeneGuardian.

At the rank level, FilterFuse divides a word over the bank groups such that each part goes to a different tile, as is the case in DRAM [371] (❹). This writing scheme ensures that different parts of the read/reference are written to different sub-arrays. In filtering algorithms, this means each processing element only has access to a small part of the read/reference. Therefore, algorithms such as LongGeneGuardian that require larger segments to be examined by a single processing element require multiple words to be written before starting the algorithm. We call the number of required words words-per-bank (WPB) hereafter. To support a WPB of larger than 1, FilterFuse implements a series of input buffers. These input buffers require the host device to provide read/reference sequences in the correct order without the need for excessive pre-processing. To fill the buffers, consecutive words belonging to a read-reference pairing are loaded into the input buffer sequentially. Once the buffer is filled with a single word set (i.e., 4 words), FilterFuse empties them in parallel. Fig. 6.7 demonstrates an example of this scheme where FilterFuse writes 1 word-set to 4 bank-groups. Note that each block represents a number of bits equal to the number of SAs per tile.

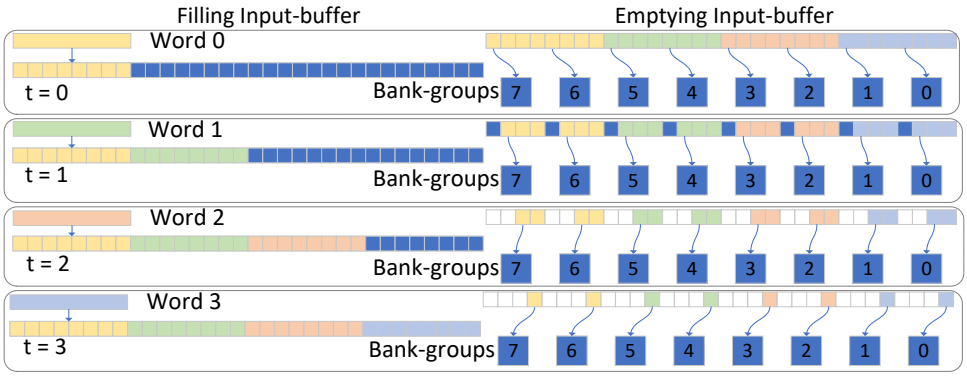

Figure 6.7: Filling (left) and emptying (right) the input buffer.

Since in a true CIM architecture, such as FilterFuse, the first base pair of the read does not always coincide with the start of a word-set, FilterFuse masks off the part that does not belong to the read-reference pairing. FilterFuse receives this mask through the address bus and passes this mask to the sub-arrays alongside the read-sequence data (❺). FilterFuse does not use the mask when filling the input buffers.

The rank level controller (❻) also handles the parallelism of new word sets via IDs, result-ready signals, and conservative buffering. For example, the rank-input-controller can load in a new word set during the computation of the previous word set as the sub-arrays can operate independently. Depending on the length of the read/reference pair,

this can either be the next word set belonging to the same pairing or the first word of the next pairing. The least significant bits (LSBs) of the ID of pairs are passed along with the input data to the sub-arrays, while the most significant bits (MSBs) are stored at the rank level in the ID MSB buffer, which is indexed by the LSB of the ID. This helps us to reduce the width of the ID busses. When the computation is completed, the results are returned, and the LSBs of the ID are matched up with the MSB again.

6.3.6. DATA MAPPING IN FILTERFUSE

FilterFuse assumes the reference sequence is already stored in memory after being split up into word-sets. This is a reasonable, common assumption that can also be achieved easily as a pre-processing step if it is not the case.

FilterFuse spreads the contents of the input buffer over the bank groups, writing the values to a set of tiles in a single sub-array per bank. FilterFuse writes consecutive word-sets to different banks to avoid contention as much as possible. Fig. 6.8 presents a high-level overview of this sub-division of two consecutive word-sets for the reference. We assume a hardware configuration with 32-bit words, 4 bank groups, tiles with 8 SAs, and a 4-WPB writing scheme.

	127							Words-set 1								0
Bank-group	3				2				1				0			
Bank	X				X				X				X			
Sub-array	Y				Y				Y				Y			
Tile	4Z+3	4Z+2	4Z+1	4Z	4Z+3	4Z+2	4Z+1	4Z	4Z+3	4Z+2	4Z+1	4Z	4Z+3	4Z+2	4Z+1	4Z
	255							Word-set 2								128
Bank-group	3				2				1				0			
Bank	X+1				X+1				X+1				X+1			
Sub-array	Y				Y				Y				Y			
Tile	4Z+3	4Z+2	4Z+1	4Z	4Z+3	4Z+2	4Z+1	4Z	4Z+3	4Z+2	4Z+1	4Z	4Z+3	4Z+2	4Z+1	4Z

Figure 6.8: Writing scheme for two consecutive parts of the reference.

Within each tile, FilterFuse writes a part of a word-set to a single row in the memory, with shifted versions of the word-set being in its adjacent crossbar rows. FilterFuse reserves the first row of each tile for the read sequence, hereafter referred to as the 'query row.' Fig. 6.9 demonstrates this mapping, where we assume a 16×16 crossbar, containing parts of 6 word-sets for evaluating an edit distance of 2. The zoomed-in version of one word-set highlights how FilterFuse stores the (shifted) references. Note that Fig. 6.9 does not account for interleaving for the sake of clarity.

We define the offset as the position of the reference segment with respect to the first base pair in the evaluated reference genome. The controller uses simple equations to find this offset.

Fig. 6.10 illustrates an example of the subdivision of an input read sequence into words and word-sets alongside the masking process. Here, we assume a 100 bps read encoded in 200 bits.

FilterFuse divides the read over two word-sets of 128 bits. It then finds the address

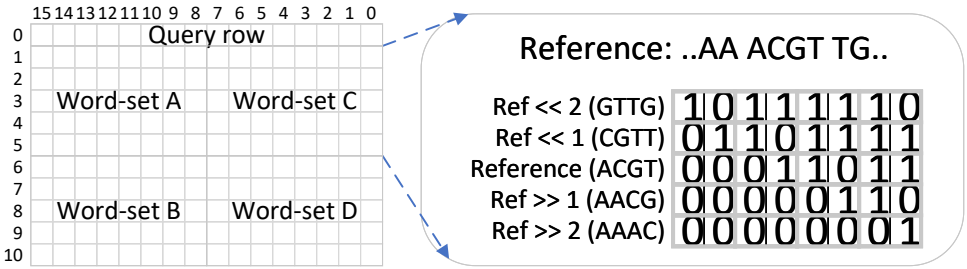

Figure 6.9: Reference mapping on a small crossbar.

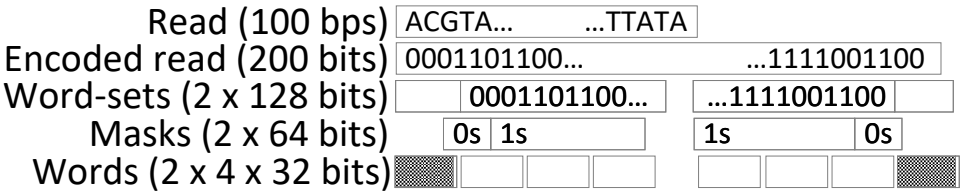

Figure 6.10: Reads subdivision into word-sets with masking.

6

of the word-set containing the first bit of the read sequence using the mentioned fixed equations. Note that since the start of the read does not necessarily coincide with the start of a word-set, FilterFuse masks the first part of the first and last part of the second word-set. FilterFuse determines this mask by subtracting the offset of the first base pair of the word-set from the offset of the read (seeding location). It then uses this local offset to find the length of the leading mask. FilterFuse always writes the read to the query row of the tiles such that the columns line up with the addressed part of the reference sequence.

6.3.7. LONG READ COMPATIBILITY

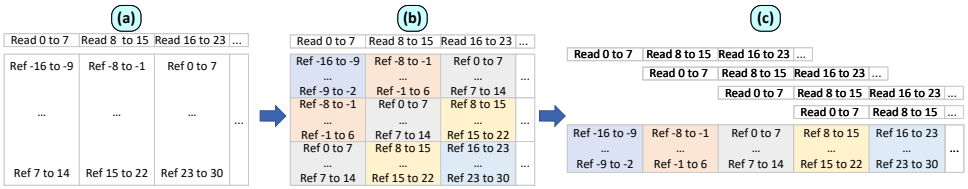

Figure 6.11: (a) High-level read mapping in FilterFuse, (b) split reads into shift-sets., and (c) the memory-optimized configuration for FilterFuse.

In previous sections, we assumed all shifted references are written to the same tile for ease of explanation. However, this assumption has two limitations for long reads. First, it limits the maximum edit distance FilterFuse can support to the rows within each

of its tiles. Since crossbars have limited dimensions (due to factors such as increasing read/write current requirements and leakage currents as the dimensions are scaled-up), the maximum supported edit distance becomes limited. Second, it demands an unreasonable memory capacity as the number of shifted references required for long reads exceeds 50 thousand shifts.

To enable processing long reads, FilterFuse splits up the evaluation of the different shifts into parts and processes them in different sub-arrays and over multiple iterations of LongGeneGuardian. FilterFuse can then aggregate the results at the rank level before calculating the edits. This requires FilterFuse to adopt two types of changes on top of the simplified examples of previous sections: (1) hardware changes at the sub-array and rank levels and (2) software changes at the input level.

Having the third observation in LongGeneGuardian in mind (Section 6.2), FilterFuse exploits the trade-off between the required memory capacity and the endurance of or the necessary write operations in the system, by splitting up read-sequences into segments of base pairs as we are splitting up the shifts. Fig. 6.11 presents an example of this trade-off, where the numbers indicate the ranges of bits that are evaluated in each shift set.

Fig. 6.11-(a) represents a case where the segments of the read sequence are compared to the entire set of shifted references in the same tile. The read sequence is split up into segments, as well as in sections of 8 shifts, which are evaluated separately. We refer to these partitions as shift sets. We observe that shift sets that share a diagonal evaluate the same sections of the reference sequence. Therefore, in Fig. 6.11-(c), FilterFuse only stores one of each shift-set, while the reads shifted with respect to the shift-sets.

The host device handles the shift of reads for FilterFuse and prevents incorrect results from the shifted outputs of sub-array results by passing the shift value alongside the pairing ID. FilterFuse saves the sub-array result for each shift-set in an AND buffer, which is placed alongside the sum buffer of discussed implementation (❼). This way, FilterFuse accumulates the partial bit-vector results of all shift sets in this buffer by performing a bitwise AND-operation between its stored value and the incoming result. When all shift sets of the read segment have been evaluated, FilterFuse uses contents of the AND-buffer as input to the Count-TCAM (❶). The rest of the procedure is identical to what was discussed previously. To prevent incorrect results from the shifted outputs of sub-array results (happening as the read sequence is shifted with respect to the start of each word set), we pass the shift value alongside the pairing ID.

6.3.8. LONGGENEGUARDIAN ON SOFTWARE VS. ON FILTERFUSE

Unlike the software implementation of LongGeneGuardian, where segments always start at the start of the read sequence, it is possible for the read sequence to start in the middle of a segment in FilterFuse, as a true CIM accelerator where references have already stored in a fixed position in the memory elements. Fig. 6.12 demonstrates an example for this scenario.

Consequently, LongGeneGuardian on Software always contains the minimum number of segments for a given reads sequence, while the implementation on FilterFuse can contain additional segments. On the one hand, this allows the FilterFuse to detect more edits than its software counterpart. On the other hand, this increases the likelihood of random matches occurring since the number of evaluated base pairs remains

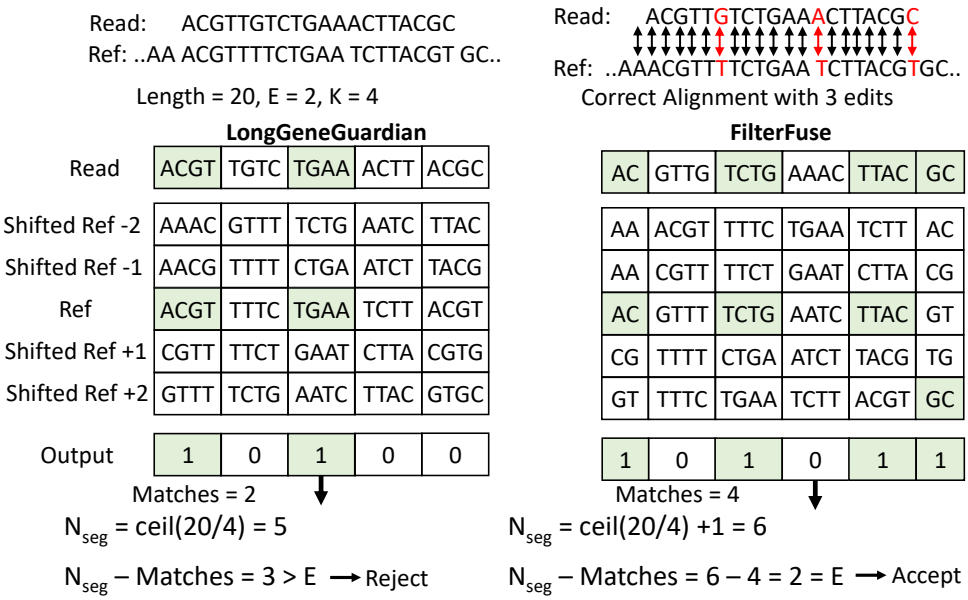

Figure 6.12: LongGeneGuardian on Software vs. on FilterFuse at the tile level.

6

the same, making the first/last segments shorter than the intended segment length. We verify the impact of the differences between the two implementations experimentally in Section 6.5.

6.4. EVALUATION METHODOLOGY

Implementations & Models. We implement LongGeneGuardian on software (C++) for its accuracy evaluations. We use a cycle-accurate RTL-based implementation to verify the functionality of FilterFuse. The analog components (i.e., ReRAM-based crossbars and TCAMs) are memory models from actual ReRAM crossbars in TSMC 40 nm CMOS technology [330], from the EU project MNEMOSENE [333]. The DRAM-based crossbars are from SIMDRAM [93, 372]. The additional components are also designed using TSMC 40 nm technology node in Synopsis Design Compiler [331]. We run all our experiments on a 12-core server with 16 GB memory, Tesla-K80 GPUs, and a Intel® Xeon® CPU E5-2680 operating at 2.4 GHz.

Baselines. We use Edlib [350] for the golden standard results of alignment for accuracy evaluations. For end-to-end evaluations, we feed the output of each filter to Edlib. We compare LongGeneGuardian and FilterFuse with open-sourced SneakySnake (SS) [186] on CPU, as the only SotA pre-alignment filter for long-reads[3]. We used two versions of FilterFuse: (1) FilterFuse-CMOS, where the tiles are SotA DRAM-based and TCAMs are SotA SRAM-based ones, and (2) FilterFuse-ReRAM, where both tiles and TCAMs are ReRAM-based ones. Note that as discussed in Section 6.2, FilterFuse can support short

[3] Neither the FPGA implementation nor the GPU one supports long reads.

reads as well. In that case, FilterFuse is quite similar to SieveMem discussed in Chapter 5.
Datasets. We use PBSIM3 [373, 374] to produce two datasets of long read-reference pairs from real datasets. We then feed these reads to Minimap2 [375], a SotA read mapper supporting long reads. We use the output to retrieve seed locations, by which we can retrieve the reference sequences corresponding to the read with SAM-tools [376]. We call these datasets PacBio_10K and PacBio_100K, with reads of length 10Kbp and 100Kbp, respectively.

6.5. EVALUATION RESULTS

6.5.1. DESIGN SPACE EXPLORATION

Fig. 6.13-(a) and -(b) present the Pareto optimal design space exploration of FilterFuse for average execution time against total die area of FilterFuse and its maximum total power, respectively. We mark most attractive designs that strike a sweet spot in the trade-off between execution time and the corresponding metric with red circles in Fig. 6.13 and call them area-optimized and power-optimized designs. Numbers label the Pareto optimal configurations, each of which is a different hardware configuration we tested for FilterFuse, but the full list is not presented for better readability.

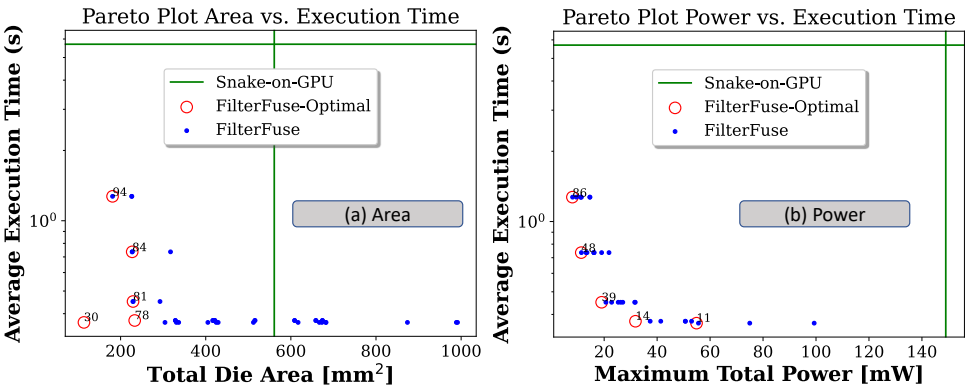

Figure 6.13: Pareto plots for performance per (a) area and (b) power.

From Fig. 6.13-(a) we make two observations. First, all of the area-optimized configurations have a smaller area than that of our GPU. Second, configuration #81 strikes a great balance of power while still optimizing for the area. More investigations also reveal that for the area-optimized designs, the configurations with large tile dimensions are favored. This is expected due to their smaller tile and control logic area.

From Fig. 6.13-(b) we make two observations. First, the power consumption is much lower for power-optimized configurations compared to that of the baseline GPU. Second, configuration #39 provides a prominent balance between area and power for the power-optimized Pareto optimal configurations. From further investigations, we also find that this optimization criterion favors configurations with large numbers of tiles per sub-array. We expect this as these configurations have fewer active tiles at a given moment, as each sub-array only uses part of its available tiles at a time.

6.5.2. FILTERING ACCURACY

Fig. 6.14 comapres the TP rate of SneakySnake with FilterFuse using different segment lengths for our datasets.

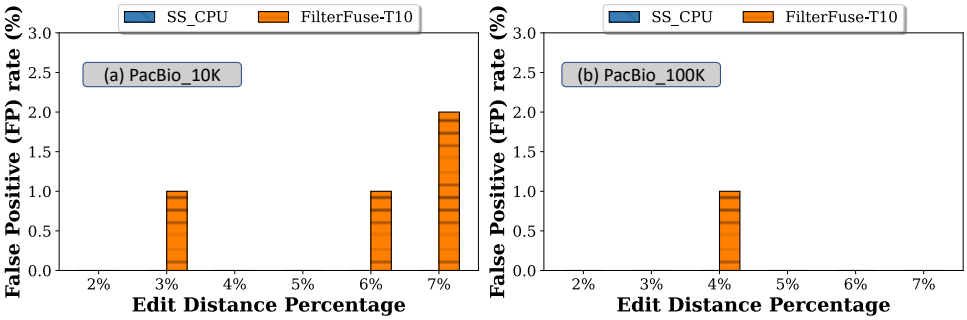

Figure 6.14: TP rate of filtering with different segment lengths.

We observe that the best accuracy varies with the edit distance, with a segment size of 8 bps being close to the optimal rate in our ROI. For this reason, in the upcoming sections, we will only present the results of this segment length.

Fig. 6.15 presents the positive rate (P-rate = FP-rate + TP-rate) of optimal FilterFuse, i.e., an indication of evaluated pairs that require alignment.

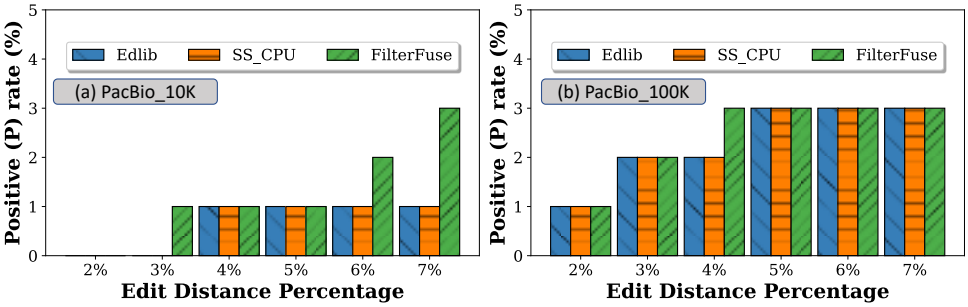

Figure 6.15: Positive rate of filtering with the best segment.

We observe that FilterFuse achieves the same or a maximum of 1% higher P-rate compared to SneakySnake and Edlib.

Fig. 6.16 presents the FP rate of FilterFuse and SneakySnake for various edit distances and datasets.

We observe that FilterFuse retains a low FP rate of < 2% compared to SneakySnake, which achieves the lowest FP rate among all previous filters.

From these results, we conclude that FilterFuse is an effective and accurate filter for long reads, achieving an accuracy as close as the SotA pre-alignment filter, SneakySnake.

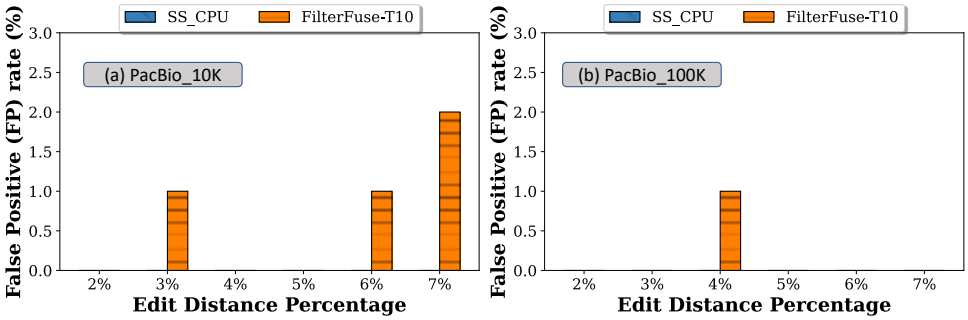

Figure 6.16: FP rate of long reads filtering with the best segment.

6.5.3. FILTERING SPEED

Fig. 6.17 presents the execution time of FilterFuse and SneakySnake over different edit thresholds and datasets.

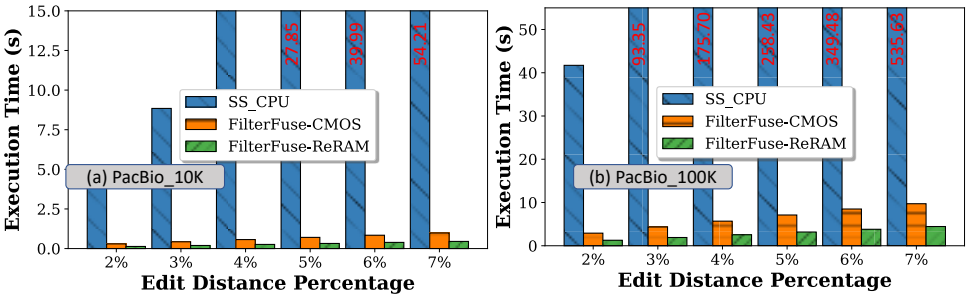

Figure 6.17: Speed of long reads filtering.

We make two observations. First, FilterFuse always outperforms SneakySnake irrespective of the read length, edit threshold, and underlying technology. This improvement can go up to 120.47×. Second, FilterFuse-ReRAM outperforms FilterFuse-CMOS. Further investigations show that this is because of the copy overhead used for XOR in the underlying tile of DRAM, even though the SRAM-based TCAMs offset some of the performance overhead. We conclude that FilterFuse is much faster than SneakySnake mainly due to exploiting CIM to eliminate the data movement overhead and its lightweight algorithm.

6.5.4. END-TO-END ALIGNMENT SPEED

A filter that is faster than the other but has lower accuracy (higher FP rate) might end up with higher end-to-end execution time when one also considers the time required for the alignment of the pairs that pass each filter. Therefore, it is necessary to compare the end-to-end execution, i.e., filtering and alignment for a given dataset, when comparing the effectiveness of a filter.

Fig. 6.18 presents the end-to-end execution time for a filter (FilterFuse or SneakySnake with Edlib) followed by alignment over several edit thresholds and datasets. To better capture the trends (and relative execution time), we limit the y-axis to 1000s. The y-axis uses a logarithmic scale.

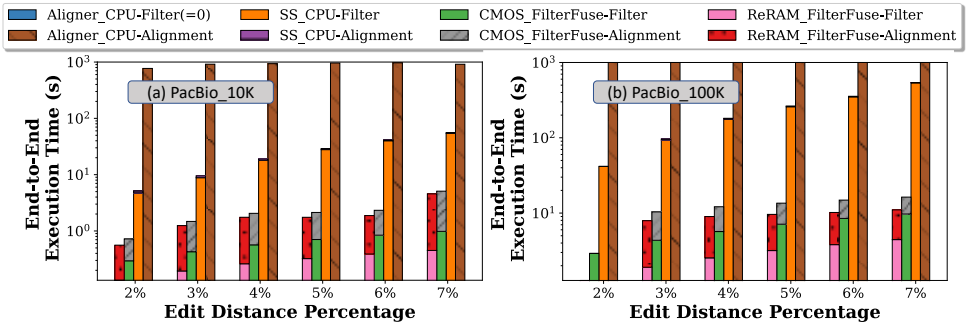

Figure 6.18: End-to-end speed of long reads filtering and alignment.

We make two observations. First, for all datasets, edit-distance thresholds, and memory technologies, FilterFuse shows an improvement of end-to-end execution time over the baseline SneakySnake solution in the ROI. This improvement goes up to 49.14× over SneakySnake. Second, as expected, FilterFuse-ReRAM improves the end-to-end time further than FilterFuse-CMOS, while the difference is less than 28.93%. We conclude that even with the decrease in the accuracy (Section 6.5.2), FilterFuse improves the performance of alignment significantly.

To determine whether FilterFuse resolves the filtering bottleneck, we also examine the new distribution of filtering time and alignment time. Fig. 6.19 presents this relative distribution over various edit distance thresholds and datasets.

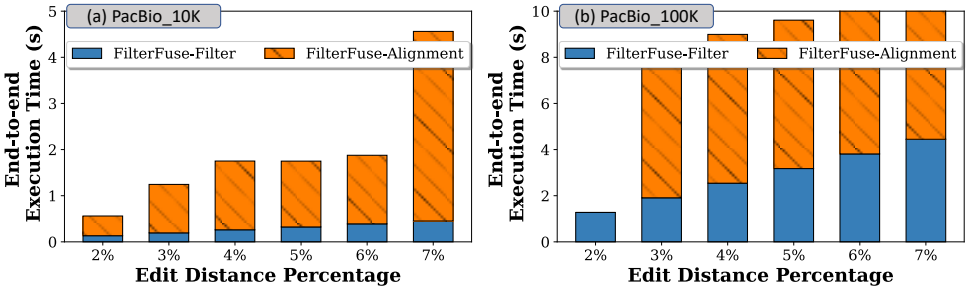

Figure 6.19: Filtering and alignment contribution in end-to-end execution time when using FilterFuse.

We observe that alignment constitutes a minimum of 59.67% of the end-to-end execution time, going as high as 90.10% of the end-to-end execution time. This means that FilterFuse improves the filtering step enough to move the bottleneck back to the

long read alignment step, making the alignment the next computational step to focus on again.

6.5.5. AREA AND POWER ANALYSIS

Table 6.1 presents the chip area and power consumption breakdown of the optimal configurations of FilterFuse.

Logic Unit	Area[mm^2]	Power[mW]
Crossbars	130582105.29	3.93
TCAMs	615813.12	0.00986
Control Logics	98032937.56	27.57
Total for FilterFuse	229230855.97	31.51

Table 6.1: Area and power breakdown of FilterFuse.

We make three main observations. First, most chip area is attributed to the crossbars and the tile-level control logic. The contributions of the higher-level components are negligible compared to the total chip area. Second, the largest contributor to the power is the power of the tile-level control logic. Three, TCAMs add insignificant area and power consumption overheads.

Table 6.2 compare the chip area and power consumption of FilterFuse with SotA design on our GPU, where the maximum GPU power is measured using nvidia-smi while running Snake-on-GPU. Note that area and power estimations of FilterFuse components that were scaled down to the evaluation technology node are scaled pessimistically, leading to conservative estimates.

Hardware	Area[mm^2]	Power[W]
NVIDIA Tesla K80	561	149
FilterFuse	229	31.5

Table 6.2: Area and power of FilterFuse vs. Snake-on-GPU.

We make two observations. First, FilterFuse has a smaller overall chip area than our Tesla-K80 GPUs. Second, FilterFuse shows a lower maximum power consumption than Snake-on-GPU, i.e., a reduction of 79.7%.

We conclude that FilterFuse also has area and power advantages over a typical GPU implementation.

6.6. CONCLUSION

This chapter proposes the first CIM architecture for pre-alignment filters of long reads, a major performance bottleneck in today's genome analysis of long reads. We call this FilterFuse. FilterFuse operates on a hardware-friendly algorithm, LongGeneGuardian, that is also compatible with the requirements of a true CIM architecture: simple operations and no assumption on the data placement. Considering the larger genomics pipeline and industrial move towards long-read sequencing, FilterFuse takes a large step in accelerating long-read genome analysis.

7

Demeter: A Fast and Energy-Efficient Food Profiler using Hyperdimensional Computing in Memory

Food profiling is crucial in any food monitoring system for health and fraud prevention but faces computational challenges. In this chapter, we aim to fix two key issues of current state-of-the-art (SotA) food profilers: (1) handling large data and (2) reducing data movement. We introduce Demeter, the first platform-independent framework for food profiling. Demeter overcomes the first limitation through the use of hyperdimensional computing (HDC) and efficiently performs the accurate few-species classification required in food profiling. We overcome the second limitation by the use of an in-memory hardware accelerator for Demeter (named Acc-Demeter) based on memristor devices. Acc-Demeter actualizes several domain-specific optimizations and exploits the inherent characteristics of memristors to improve the overall performance and energy consumption of Acc-Demeter. We compare Demeter's accuracy with other industrial food profilers using detailed software modeling. We synthesize Acc-Demeter's required hardware using UMC's 65nm library by considering an accurate PCM model based on silicon-based prototypes. Our evaluations demonstrate that Acc-Demeter achieves a (1) throughput improvement of 192× and 724× and (2) memory reduction of 36× and 33× compared to Kraken2 and MetaCache (2 SotA profilers), respectively, on typical food-related databases. Demeter maintains an acceptable profiling accuracy (within 2% of existing tools) and incurs a very low area overhead.

This chapter is partially based on the candidate's work [120].

As discussed in Section 2.2.1, the industry uses SotA taxonomic profilers from metagenomic studies for today's food profiling as food profiling and metagenomics profiling share a close problem statement. Unfortunately, these profilers are overkill for simply profiling a given food sample and, therefore, costly since those taxonomic profilers have been designed for different, more complex goals such as (1) capturing complex operations between organisms or (2) finding insights on species that cannot be clonally cultured in labs. Such profilers are also designed for working on larger, more complex, and randomly mixed genome sequences and demand a significant amount of resources that simply impede real-time monitoring of all food samples after production, shipment, or distribution; the ultimate goal of a food monitoring system. Therefore, a new solution must be sought after specifically for food profiling that is cheaper, faster, more energy-efficient, and yet accurate.

In particular, we pinpoint two critical sources of inefficiency in SotA profilers currently used for food monitoring, collectively called food profilers or profilers hereafter. First, all current (food) profilers work with significantly large working data structures, e.g., humongous hash tables or sorted lists, that require high-end servers with extensive storage and memory capabilities to be handled. This fundamentally limits performance scaling on par with that in sequencing technologies. Second, current profiling techniques incur a significant number of random accesses to large working datasets, and as a result, unnecessary data movement between their storage and memory plus their memory and compute units which cannot be otherwise done where the data resides due to (1) the size of the final data structures and (2) the required operations for tasks in hand. This directly translates to massive energy consumption and latency. For example, as shown in our evaluations Sections 7.3, 7.6, a widely used SotA profiler takes ~1 minute to profile one high-coverage sequenced food sample. However, it requires a super machine or cluster with at least 300 GB of memory and proportionally scaled-up compute power. These costs add up to an unbearable amount of required time and equipment for real-time monitoring of all existing and produced food samples. Therefore, a healthy economy regarding the food industry cannot keep using these profilers and demands cheaper, faster, more energy-efficient, and more accurate food profilers for the years to come.

Our goal in this chapter is to solve both limitations of previous profilers, namely (1) reliance on high-end servers and scaling problems due to required massive data structures and (2) incurring unnecessary data movement. **To this end,** we propose Demeter, an end-to-end, hardware/software co-designed food profiling framework that efficiently profiles species of a food sample. **The key idea** of Demeter is to reduce the food profiling problem to a multi-object (multi-species) classification problem using hyperdimensional (HD) computing (HDC) followed by an abundance estimation step. Demeter is a platform-independent framework and produces accurate results on any hardware platform such as a central processing unit (CPU), graphics processing unit (GPU), or application-specific integrated circuit (ASIC).

Our experiments show that although the accuracy of Demeter is comparable with existing SotA profilers, typical processing units (CPUs) are not exploiting the full parallelism offered by our HDC-based approach, prohibiting those platforms from outperforming SotA profilers. Moreover, we find two more optimization opportunities that can

be achieved with a wisely-chosen platform: (1) eliminating the cost of existing shift operations and (2) mitigating the significant amount of data movement involved in our HDC-based solution. Therefore, we propose an in-memory hardware accelerator for Demeter, Acc-Demeter, to mitigate the costs mentioned above and simultaneously solve the second problem of profilers as well. Acc-Demeter achieves these by (1) the physical attributes of nanoscale memristive-based devices[1], (2) Processing-In-Memory (PIM), where the data resides, and (3) zero-overhead shift operation in hardware. It is worth noting that, with the advent of portable sequencing machines, a move from cloud computing with sophisticated infrastructure towards an in-build profiler (or other genomics-related kernels) inside the sequencer is finally in the foreseeable future.

We make the following main contributions in this chapter:

- To our knowledge, Demeter is the first framework that enables food profiling via HDC. Demeter provides a five-step approach to determine the relative abundance of a set of the food read sequences at species-level. We design Demeter to (1) address the key problems of food profiling rather than accelerating regular metagenomic profilers and (2) be platform-independent (Section 7.2).

- We propose a PIM-enabled hardware accelerator for Demeter using memristor devices (Acc-Demeter) to extract Demeter's full potentials and solve the data movement problem in Demeter and previous profilers. We propose several optimization techniques for Acc-Demeter based on domain-specific knowledge of food profiling and our background on PCM cells characteristics and HDC operations. To our knowledge, Acc-Demeter is the first (in-memory) hardware accelerator for a food profiler (Section 7.4).

- We rigorously compared Demeter and Acc-Demeter to four SotA food profilers. We show that Demeter provides an accuracy level comparable with previous food profilers and within the accepted level of food monitoring systems. The default setting of Acc-Demeter enables a (1) throughput improvement of ~192× and 724× and (2) reduction in the required memory of ~36× and 33× compared to Kraken2 [193] and MetaCache [212], respectively, when querying on a typical food-related reference genome database, i.e., AFS20 [212]. Our design requires only ~8.9 mm^2 die area and can process ~9.45 Mbp per joule for our largest food-related database AFS31 [212] (Section 7.6).

7.1. BACKGROUND AND MOTIVATION

This section discusses the necessary background and introduction to (1) the current taxonomic profilers and their shortcomings when used for food profiling, (2) HDC. We devote the materials mainly to those closely related to or used by Demeter and not discussed in Chapter 2. For more detailed background information, we refer the reader to comprehensive reviews on these topics [126, 194, 288, 377–380].

[1] We choose phase change memory (PCM) devices as members of memristor families due to our accessibility to accurate measurements and models. However, in principle, our proposed techniques can be applied to any memristor-based memory technology, such as ReRAM or STT-MRAM.

7.1.1. METAGENOMIC PROFILERS

Constantly increasing the performance of sequencing technologies and the fast drop in the cost of DNA sequencing [26, 27] catalyzed the metagenomic studies [11, 190, 191]. These studies enable us to capture the big picture of the environment without isolating or cultivating individual organisms. For this purpose, one needs to perform taxonomic profiling: determining the relative abundances of species in a sample directly taken from the environment. Due to the high cost associated with alignment and assembly for large reference datasets, to this date, we still prefer heuristics statistical-based profilers to assembly- or alignment-based ones. However, even these profilers are not yet cheap or economical and prevent large-scale, real-time studying. Their cost is mainly related to the required memory for profilers' data structure and algorithms. Such large data structures or sophisticated algorithms force us to use high-end servers and are needed to fulfill complex goals of subsequent metagenomic analysis, namely capturing complex operations between organisms and discovering insights on species that can not be clonally cultured in labs. This high cost of profiling in a metagenomics profiler prevents us from efficiently profiling food samples in real-time, the end goal of a food monitoring system.

7.1.2. PROBLEMS OF FOOD PROFILERS

We use VTune [381] and profile three SotA profilers that are currently used for food samples as well, namely Kraken2, CLARK, and MetaCache, using their default datasets and parameters on the original platforms for which they have been designed. We make two main observations, which follow a similar trend reported in previous studies in genomics as well [193, 244, 382].

Observation 1. All these profilers induce large memory requirements for their data structures. For example, Kraken2 requires a minimum of 300 GB memory for its reference data structure. Even for smaller and less complex reference data bases such as those in food industry, Kraken2 still requires more than 50 GB of memory (Section 7.3.4).

Observation 2. All profilers induce high miss rates in L2 and L3 (~68 to 90%). The nature of their underlying algorithms causes this inefficiency because they always query a small fraction of keys in a large hash table and/or sorted list, leading to random memory access patterns. In other words, the arithmetic intensity of the profilers is too small to the extent that even increasing the number of threads does not help resolve the CPU stall cycles caused by memory accesses required by these misses.

Overall, current food profilers' large working data structure and their low arithmetic intensity lead to high storage cost, low performance, and high energy consumption. It also demands high-end servers. This motivates designs (such as Demeter and Acc-Demeter) that provide reduced working data structures, eliminate unnecessary data movements, and can liberate us from dependency on the clusters.

7.1.3. HYPERDIMENSIONAL COMPUTING.

Hyperdimensional (HD) computing [383], also called vector symbolic architecture (VSA) [384], is a brain-inspired computing paradigm which has been demonstrated to be effective in 1) solving cognitive tasks, such as analogical reasoning [385–387], 2) learning domains, such as text classification [388–391], gesture recognition [392], and latent se-

mantic analysis [393] and 3) privacy and security problems [394]. The name hyperdimensional comes from the fact that in the HD Computing realm, we deal with spaces and vectors, which mimic a large number of neurons and synapses existing in the brain's circuits.

The main difference between traditional computing and HD computing paradigm is their element representation and interpretation. In the traditional computing paradigm, the elements, such as a single bit or small group of them, can be interpreted without referring to the other elements. This is called localist representation. In the HD computing paradigm, on the other hand, a total set of bits interpret together and individual elements in representation do not have a specific meaning. To express this paradigm, we represent elements using high-dimensional vectors, hereafter called HD vectors. Therefore, each HD vector represents an element in a distributed manner, as opposed to the localist representation, and can be composed of real numbers [395–398], binary numbers [383, 399], bipolar [392, 396], or complex numbers [400]. The difference in representation introduces a few new properties which turn out to be highly powerful in some domains, which we discuss next.

HYPERDIMENSIONAL COMPUTING PROPERTIES

HD vectors in the HD computing paradigm have several features and properties that make them suitable for the tasks in which are being currently employed.

- **Robustness.** The HD computing-based architectures (or VSAs) are tolerant against failure due to random errors. The primary reason for this robustness comes from the redundancy in the representation of each element, or in other words, the fact that the set should be interpreted as a whole, and not as individual elements/bits. It is worth noting that the robustness increases as the dimensionality of the space (or similarity put, the size of the HD vectors) increases.

- **Holisic Representation.** Holistic representation of HD computing-based architectures brings them fault tolerance independent from the error's position. This means that the information degradation caused by an error, bit flip in an HD vector, is irrespective of the position of the error in HD computing.

- **Randomness.** To incorporate the seeming arbitrariness of brain structure, meaning that no two brains are identical, in HD computing, one should start with HD vectors which are randomly drawn from the hyperspace. Building on top of these randomly chosen initial vectors and recursively creating the next elements brings HD vectors the property of randomness.

HYPERDIMENSIONAL REPRESENTATIONS AND CONSTRUCTIONS

As alluded before, HD vectors can be in forms of binary, real, complex, and bipolar numbers [383, 392, 395, 396, 396–400]. Binary representations of HD vectors have been shown more practical and efficient, than the others, when it comes to the classification problems or one-shot reinforcement learning. Therefore, we also decided to proceed with binary HD representations in this thesis.

For determining the distribution in the binary representation, in the context of supervised learning, we should determine two major hyperparameters: 1) density of HD

vectors and 2) mapping function. For the density, we have two choices of sparse and dense representations. In the dense representation, the number of "1"s and "0"s in vector elements are equiprobable. This means that if we define the density of an HD vector as the hamming weight of the vector $|V|$ divided by the dimensionality of HD vector (N), p = $\frac{|V|}{N}$, then for dense representation we have $p \approx \frac{1}{2}$ This is not the case for the sparse binary representation. The mapping functions categorize based on their ability to map the features with similarity preservation of the origin or without it. Based on these two hyperparameters, we summarize the major operations on widely-used sparse and dense HD representations below.

Binary Sparse HD Representations. Sparse Distributed Representations (SDRs) are the closest representation to neurons in a biological brain as in any given time, only a small proportion of neurons is active. In the following, we describe the fundamentals of SDR and explain their common hyperparameters. We, however, refer the reader to prior work [399, 401] for a more detailed discussion on aspects of SDRs.

In SDR, the number of "1"s in the HD vector is much fewer than that of "0"s and can be realized easily when a large dimension HD vector is used. The similarity, shortened to "sim" hereafter, of two HD vectors V_1 and V_2 in SDR is estimated based on the number of overlapped "1"s which can be calculated using the elementwise conjunction of the two HD vectors:

$$sim = |V_1 \wedge V_2| \tag{7.1}$$

In the binary representation, this is equivalent to the dot products of these two vectors. The main reason behind this mapping is that since we treat "1"s and "0"s asymmetrically in the SDR, the geometric vectors will have an origin of 0^N and will be placed in the same hyperquadrant in our high dimensional space and their similarity will be non-negative.

HD vectors in SDR that represent unrelated entities, atomic SDR vectors, can be combined, or bundled, by elementwise disjunction, which preserves the similarity of the combined vectors. Since the density of the bundled vectors increases with the number of involved atomic HD vectors, we use the context-dependent thinning (CDT) procedure to keep the density close to the starting atomic HD vectors [399]. CDT takes an SDR vector V, constructed with bundling several other vectors using elementwise disjunction, as input, and thinned it in two steps: 1) It first permutes V for k times. These permutations should be all independent, random, and fixed. 2) The output of permutation then will be bundled with the original V using elementwise conjunction. In practice, cyclic shift with a fixed, yet randomly chosen unit of shifts, is used for the permutation. There are also examples in the prior works such as in [401] in which they applied the same random permutation T times recursively.

Binary Dense HD Representation. In the binary dense HD representation, which is called dense distributed representations or DDRs, the values of every single bit in the HD vector are equally probable and independent, unlike the values in SDR vectors. This was first proposed by Kanerva et al. [383, 393], in which they used dense HD vectors of size N = 10000 binary elements. In DDRs, the similarity of two HD vector is determined

by the Hamming distance of two vectors, normalized to the dimension or size of the vectors, N. This operation can be implemented as summation and then normalization of the bitwise XOR of two vectors. One can easily show that the hamming distance of any two arbitrary HD vector (not similar HD vectors) exploiting this representation is around 0.5 [383, 402].

Like SDRs, HD vectors in DDR are also generated independently, randomly, and recursively. Random vectors can be calculated using the cyclic shift operation, which is similar to the SDR case. The simplest bundling operation in DDR is the elementwise summation. However, this will result in a non-binary vector. To convert our new HD vector into a binary representation again, the bundling operation also performs an elementwise majority rule which outputs a binary HD vector. The elementwise majority results in "1" only when $\frac{G}{2}$ or more arguments are "1", and vice versa, for each position in the HD vector, where G is the number of vectors being bundled together. When there is an even number of "0"s and "1"s, one can break the tie by randomly generating a vector and adding it to the superposition. This effectively will be like drawing the output from a Bernoulli distribution B(1,0.5). The result of such bundling will be itself an HD vector in DDR and we refer enthusiastic readers to previous works for more details and examples [393, 402, 403]. We use DDRs in our proposal that we discuss in Section 7.

Bipolar HD Representation. In the sparse and dense HD representations, SDR and DDR, we encode the components in each vector using "0"s and "1"s. In the bipolar HD distributed representation (BDR), on the other hand, we encode the components into "-1" and "+1". From the computational point of view, bipolar representation is more convenient and posses a set of interesting features. The distance metric in BDR is simply the dot product, as in SDR. The bundling operation in BDR is implemented as an elementwise summation. However, since doing that will results in a non-bipolar HD vector, to put it back into the definition, we separate the results of the elementwise summation into the sign and magnitude and consider the sign, and restrict the bit value to a pre-defined threshold at which we clip the resulting integer. Since these operations are not as hardware friendly as operations needed in binary representations, current HD-based hardware accelerators do not exploit this representation. We refer the readers to [396, 397] for more detailed explanations.

CLASSIFICATION USING HDC

Like other reference-based classifiers, an HDC-based system also takes two steps: (1) training and (2) classification. An encoding mechanism is used in both steps. One famous example is the N-gram encoding mechanism that follows a two-step approach for encoding a string of size L to an HD vector of size D. **Step 1:** It combines N consecutive alphabets of the string and builds an HD vector that is orthogonal to them all and can preserve their relative order. This operation is called binding and is represented in Equation 7.2, where $\rho^i(X)$ represents the ith permutation of vector X and B_i are once randomly-generated representative HD vectors (also referred to as atomic or basis HD vectors) for the ith character of the string C_i. The string is a DNA sequence in Demeter in Section 7.

$$N-gram(C_1, C_2, ..., C_N) = Sh[\ldots Sh[Sh[B_1] \oplus \rho(B_2)] \oplus \ldots] \oplus \rho^{N-1}(B_N) \tag{7.2}$$

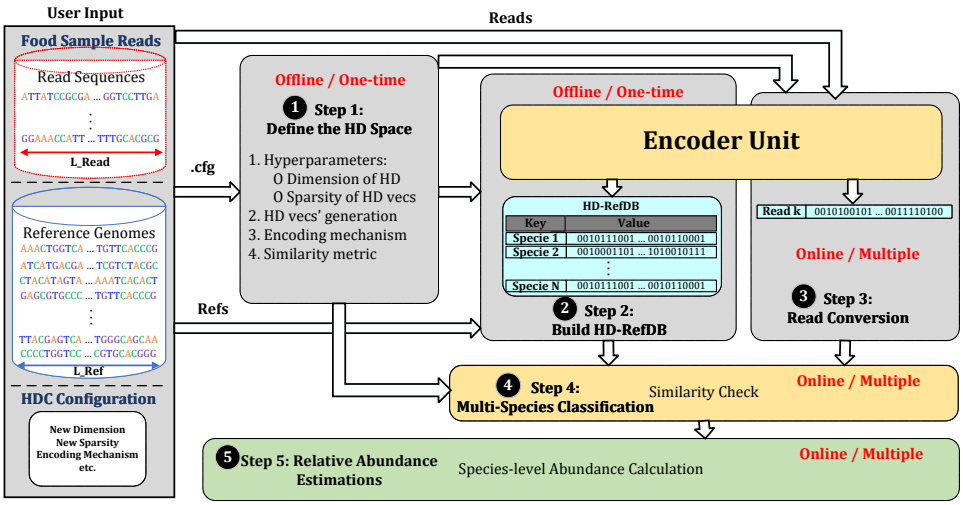

Figure 7.1: Overview of Demeter framework.

Step 2: The encoder performs an element-wise addition between all HD vectors corresponding to consecutive N-grams, called bundling, to present the entire input sequence. To binarize the final HD vector, the encoder applies a majority function over each position. This final vector is stored in associate memory (AM) and is called a prototype HD vector if the input was a reference genome. Otherwise, it is called query HD vector and will use it for classification.

The most common approach for classifying whether the sequence query belongs to any of the classes in AM after using N-gram encoding mechanism is to measure the hamming distance between the query HD vector (Q) and each of the prototype HD vectors (Ps) and decide based on a fixed distance or threshold (T). This can be easily performed with an XNOR of Q and each P followed by a pop-count[2] and thresholding operation, as shown in Equation 7.3.

$$Classification(i) = \begin{cases} 1, & \text{if } \sum_{j=1}^{D} Q(j) \bar{\oplus} P_i(j) \geq T \\ 0, & \text{otherwise} \end{cases} \tag{7.3}$$

7.2. DEMETER

Demeter is a configurable framework for food profiling and is based on three main insights: (1) current food profilers, whereas accurate, are neither memory- nor energy-efficient, (2) the primary sources of high cost and inefficiency in current food profilers is their large reference data structures and working sets, and (3) one can profile food samples quickly and accurately using HDC. Fig. 7.1 provides an overview of the five key steps

[2] Pop-count (population count) of a vector or specific value is the process of finding the number of set bits (1s) in that value.

in Demeter: ❶ defining an HD space, ❷ building an HD reference database (HD-RefDB), ❸ converting sample reads into HD space, ❹ determining the possible species assignment per sample read, and ❺ performing abundance estimation. We describe each step in more detail next.

7.2.1. STEP 1: DEFINE THE HD SPACE

As the first step (❶ in Fig. 7.1), Demeter defines an HD space for all subsequent operations and steps. This is a crucial step as it determines the operations in the remaining steps. Unfortunately, many previous HDC-based proposals did not support the user's input for determining the HD space and designed their space statically. Hence, such designs are more limited.

Demeter defines the HD space in 4 stages. **Stage 1:** Demeter fixes two hyperparameters: (1) The dimension of the HD space; i.e., the dimensionality of the HD vectors (element representations), and (2) The sparsity of each element (HD vector). **Stage 2:** Demeter generates a few atomic HD vectors and store them in memory (commonly called Item Memory (IM)). These vectors can be (1) the HD vectors that represent our genome alphabets or (2) the one-time randomly generated HD vectors that some encoding mechanisms use, for example, to introduce the concept of order between alphabets of one input. **Stage 3:** Demeter decides on the encoding mechanism to build the space with. This very encoding mechanism will be used throughout Steps ❷ and ❸ of Demeter. **Stage 4:** Demeter fixes the similarity metric and any other associated parameters (such as thresholds) based on the user's input or a common choice considering previous stages. Demeter stores a default value for each stage in a configuration file. Once the user summons Demeter, Demeter quickly checks if a configuration file matches with the user's requested HD space or not. Demeter only runs this step if such file does not exist or the user asked for a change.

7.2.2. STEP 2: BUILD DEMETER'S REFERENCE DATA STRUCTURE

Demeter takes two sets of inputs in step ❷: (1) HD space parameters defined in Step ❶ and (2) a reference genome database. Subsequently, Demeter builds a new reference database in its HD space out of all the considered reference genomes. This new database, called HD-RefDB, consists of one (or few) prototype HD vector(s) from any given reference genome in the original reference database and is stored in AM. HD-RefDB can be as varied as the number of combinations of possible hyperparameters, atomic vectors, and encoding mechanisms in Step ❶. This step aims to reduce the size of the working set for the classification task while avoiding accuracy drop. Since this step requires only simple arithmetics and is also highly parallelizable, it can still be accelerated on our proposed PIM-enabled accelerator (Section 7.4).

7.2.3. STEP 3: DEMETER'S READ CONVERSION

Demeter again takes two inputs in Step ❸: (1) HD space configuration and (2) read sequences of the food sample under study. Demeter translates each of these read sequences into one query HD vector. To prevent any extra storage cost and to pipeline computations of Step ❸ and Step ❹, Demeter forwards each query HD vector to the next step instead of storing them inside a memory unit while waiting for all of them to be con-

structed first[3]. Query HD vectors created in this step can require larger or smaller space than a read, depending on the initial length of the read sequences and the dimension of the HD space. Therefore, although Steps ❷ and ❸ share the encoding mechanism, their input and how Demeter treats the outcome are pretty different. Step ❸ neither introduces a new operation nor a procedure other than those that already exist from Step ❷. Therefore, it enjoys similar benefits as Step ❷, namely high parallelization and in-memory suitability. Demeter runs Step ❸ every time it profiles a new read of a food sample.

7.2.4. STEP 4: MULTI-SPECIES CLASSIFICATION PER READ

In this step, Demeter takes (1) the query HD vector (Step ❸), (2) HD-RefDB (Step ❷), and (3) similarity function and its corresponding parameters (Step ❶) as inputs. To determine the specie(s) that each read belongs to, Demeter performs a similarity check between the query HD vector and each of the prototype HD vectors in HD-RefDB. The similarity measure can vary depending on the vector representations and encoding approaches. Demeter allows various famous mechanisms for the similarity check, such as Hamming distance [380] and dot product [379]. Usually, this step can be implemented only with simple operations (Section 7.1.3). It also enjoys high parallelization, similar to the previous steps. Although a similarity metric and its related parameters highly relate to (1) the encoding mechanism and (2) hyperparameters of the HD space, such as representations, sparsity, and the dimension of HD vectors, and therefore it makes sense not to let them change arbitrarily, Demeter supports changing them in Step ❹ as well, without needing to re-run Steps ❷ or ❸. This is because some studies show that different similarity metrics and thresholds may outperform others depending on your application and data for a fixed set of hyperparameters and encoding mechanisms. Therefore, if one decides to change their reference database, they may need to play with these to find the right match, and Demeter allows such investigations. Currently, Demeter provides a default option.

Demeter may find out that the query HD vector is close to one, multiple, or none of the prototype HD vectors in HD-RefDB. This variety in possible outputs differentiates Demeter from many previous HDC-based designs [240, 296, 380, 403, 404]. In such works, mostly due to the characteristics of applications under study, researchers always assume that (1) the query HD vector can only belong to one of the prototype HD vectors, and (2) the class of the query HD vector will exist in the AM. However, none of these assumptions hold for a food profiler. One read from the food sample can be related to one, multiple, or none of the reference genomes in the original reference genome database. This is because the read sequences are mostly short strings with a reasonably high probability of existence in longer reference genome sequences. It is also not uncommon that the query HD vector does not belong to any of the reference genomes in the initial reference genome database. This case can happen when, for example, (1) there is either an unknown species in the food sample, (2) one incorrectly excludes the corresponding

[3] This is the default behavior in Demeter. However, we also provide the option for the user to keep and store these query HD vectors (HD-ReadDB) in case one needs to analyze them further. If one uses this option, the stored query HD vectors create another database representing the reads in our food sample, called HD-ReadDB hereafter.

reference genome in the initial reference genome database, or (3) an uncorrected sequencing error has happened. A food profiler should capture such cases. This difference between how many prototype HD vectors in HD-RefDB can be assigned to one query HD vector is a key difference that affects both the following abundance estimation step and final results. It also distinguishes this work further from previous HDC-based proposals for different applications. Step ❹ also enjoys high parallelization and in-memory suitability features similar to previous steps.

7.2.5. STEP 5: SPECIES LEVEL ABUNDANCE ESTIMATION

In Step ❺, Demeter performs a relative abundance estimation based on the results of Step ❹. This step is particularly needed for a food profiler in which one query HD vector can be similar to one or more classes/species. Demeter categorizes each query HD vector into (1) uniquely-mapped, (2) multi-mapped, and (3) unmapped, taking a two-step approach. In the first step, Demeter assigns the uniquely mapped query HD vectors to the species that they are similar to. In the second step, Demeter assigns the multi-mapped query HD vector to multi-species proportionally to the number of reads that have been uniquely aligned to in the first step divided by the length of species (reference genome). Demeter's Step ❺ can be extended to support different assignment policies for the multi-mapped reads. We leave investigating the effect of such methods for future work.

7.3. DEMETER'S EVALUATION

7.3.1. METHODOLOGY

We implement a multi-threaded highly-parallelized version of Demeter in `C++` using SeqAn library [405], called C-Demeter. SeqAn library is an open-source optimized library for biological data. C-Demeter verifies the accuracy of Demeter. We also implement a GPU version of Demeter, G-Demeter. G-Demeter uses CUDA streams for parallelizing data copy operation between shared memory and global memory with other computations as much as possible. It implements the similarity check using parallel reduction technique introduced by Harris et al. [406] in the shared memory. All of our experiments run on a 128-core server with AMD EPYC 7742 CPUs [334] and with 500 GB of DDR4 DRAM. G-Demeter runs on a NVIDIA RTX 2080Ti GPU. Our sensitivity analysis shows that binary HD vectors of size 40,000, with dense distributed representation (DDR [380]) and N-gram-based encoding mechanism, strike a sweet spot in the tradeoff between accuracy, required memory, and performance. Therefore, unless otherwise stated, our evaluations use these setups.

Accuracy Metrics. We capture the four fundamental rates from a (food) profiler when considering the presence and absence of each species in the output, i.e., True Positive (TP), False Positive (FP), False Negative (FN), and True Negative (TN) Rate. Based on these rates, Demeter reports two standard metrics of Precision and Recall [192, 193, 407] to assess the accuracy of our (food) profilers.

Performance Metrics. Performance analysis consists of three experiments: (1) Build time, (2) Query time, and (3) Query throughput or speed. This separation has two main reasons. (1) Build time is normally a one-time job and does not affect the overall pro-

filer's performance. Therefore, it is only fair to separate build time and query time. (2) Query time is simply the required time for profiling one single read. However, throughput is measured by million reads per minute ($\frac{MR}{m}$) and should be differentiated as it can get affected easily by other factors such as the size of the data structure, the classifier's parallelization capability, or the infrastructure's computation and storage/memory limitations (e.g., duplicating capabilities).

Datasets. We have two sets of datasets. (1) Genome sequences used as a reference database. (2) Genomes sequences used as food samples and input queries. We consider AFS20 and AFS31 [212, 213] as our reference genome datasets. These datasets are two-commonly used datasets consist of 20 and 31 food-related reference genomes related to animals whose sizes vary from 12 MB to 14 GB. AFS31 is currently also the biggest reference dataset used in food profiling. Food sample reads or queries are from calibrator sausage samples from ENA project ID PRJEB34001 [408] and PRJNA271645 [409]. These reads are real short-read sequences from a mixture of food ingredients such as chicken, turkey, etc., sequenced on an Illumina HiSeq machine.

Baselines. We compare Demeter against MetaCache [212] (the most accurate food profiler) Kraken2 [193], Kraken2+Bracken [204], and CLARK [203], the top 3 alignment-free and fastest metagenomic profilers that are also commonly used for food profiling.

7.3.2. DEMETER'S ACCURACY ANALYSIS

Figures 7.2 and 7.3 present the results for the precision and recall of all evaluated food profilers on the species levels over AFS20 for Kylo and Kal food samples [408, 409], respectively. Note that the relative abundance of higher taxonomy levels is not of importance in food profiling. Additionally, those calculations highly depend on the propagation method from species level to those levels. Therefore, they have been excluded from this study.

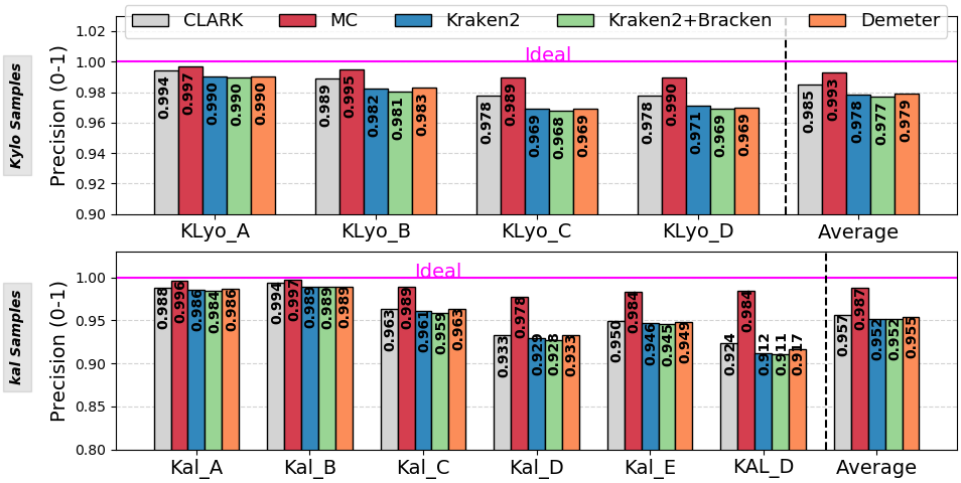

Figure 7.2: Precision rate for Kylo and Kal Samples on AFS20.

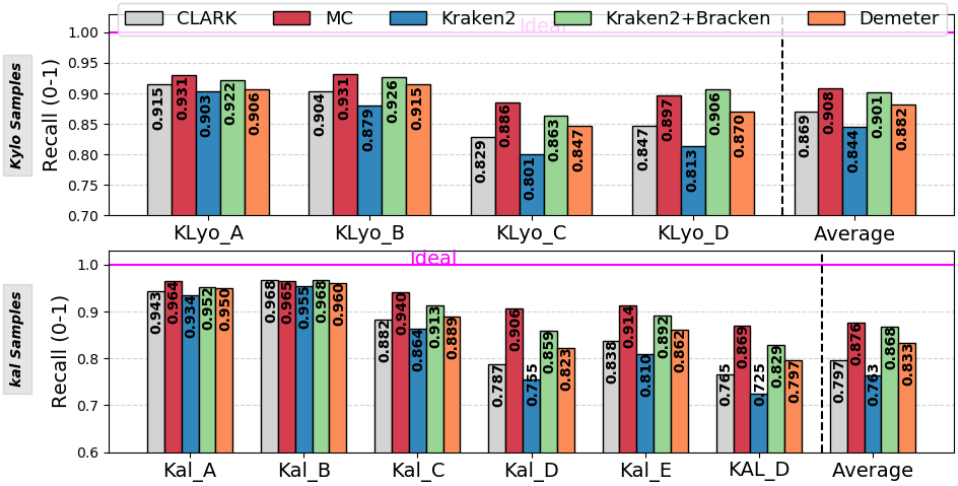

Figure 7.3: Recall rate for Kylo and Kal Samples on AFS20.

We observe that Demeter stands very close to the most accurate profiler, MetaCache, and has only 1.4% and 2.6% less precision and recall, respectively, for KLyo samples. Moreover, Demeter achieves similar results on AFS31 and Kal samples. Note that accuracy is very much data-dependent, and indeed this accuracy drop is acceptable for a food profiler. The results of the latest comparison between current (metagenomics) profilers [205] show an Std error of the mean ranging from 0 to 5% regarding the precision and recall among various widely-used profilers on different datasets.

We conclude that Demeter is accurate and achieves high precision and recall for food samples. These results show that Demeter's HDC-based classification approach followed by our abundance estimation technique does not hurt the accuracy of the profiler compared to baselines.

7.3.3. DEMETER'S SOFTWARE PERFORMANCE ANALYSIS

Fig. 7.4-a and Fig. 7.5-a present the time that each profiler takes to query one (short) read from the query food sample and classify its specie(s) over AFS20 and AFS31, respectively.

We observe that both C-Demeter and G-Demeter, whereas accurate, require higher query time compared to Kraken2. The time breakdown, using Intel VTune [381] and cudaEvents, reveals that both implementations are memory bound, meaning there exists a significant percentage of under-utilized slots due to data access issues.

We believe that there are two main reasons behind this problem. First, the shift operation per processed character in the encoding mechanism of Demeter. Both of these implementations store the large HD vectors into multiple registers. Every shift operation translates to multiple copy operations among those registers, which can become costly in terms of time and energy consumption. This is why the query time is higher than expected. Second, not all prototype HD vectors fit in the caches. Therefore, the software versions take a few cycles to read prototype HD vectors in batches, compare them to

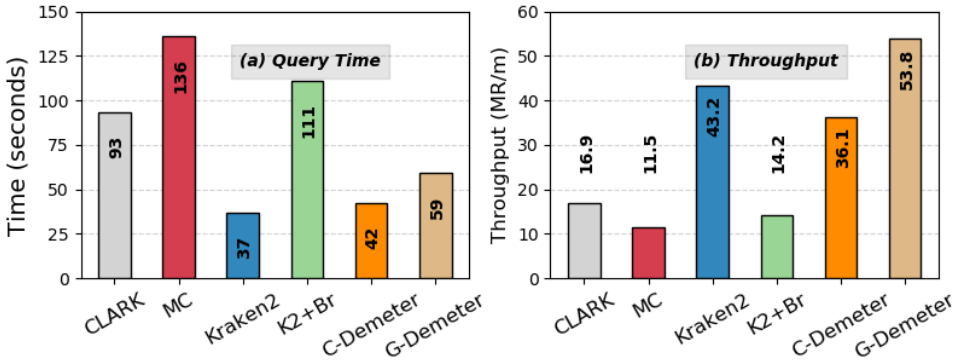

Figure 7.4: (a) Query time and (b) Query throughput on AFS20.

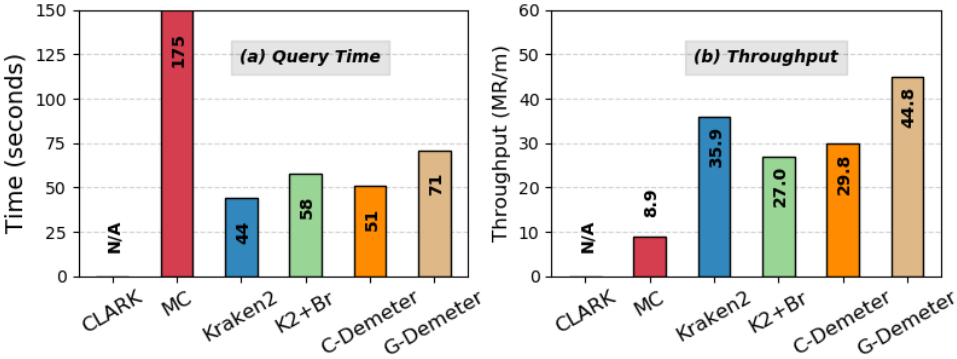

Figure 7.5: (a) Query time and (b) Query throughput on AFS31.

query HD vector, save the results, and continue with the next batch. Note that these also put a limit on the expected throughput.

Fig. 7.4-b and Fig. 7.5-b present throughput of different profilers over AFS20 and AFS31. We make three observations. First, C-Demeter achieves a lower throughput compared to Kraken2. The reasons behind this are similar to what was discussed for its longer query time. Second, we observe that G-Demeter improves the throughput by up to 24% (depending on the reference dataset) and therefore can be used for food profiling in the industry in the near future. Third, we observe that simply increasing the number of working threads by moving from C-Demeter to G-Demeter does not improve the throughput considerably. We ask to use the commodity GPUs to perform the food profiling to cut the cost in the short term. In the long term, we propose extending Demeter to ASIC designs (such as those we present next) that solve the new sources of inefficiency we discussed above.

However, our analysis also shows that even a massively-parallel implementation of Demeter, G-Demeter, does not fully utilize the parallelism offered by vector operations of HDC classification of Demeter, while also suffering from expensive copy-pasting among registers and its inability to perform the classification efficiently on a large vector in software.

7.3.4. DEMETER'S MEMORY ANALYSIS

To show a key source of improvement in Demeter (and an enabler for Acc-Demeter), we compared the memory requirement of Demeter with the other food profilers. Fig. 7.6 presents the required memory for each profiler on AFS20 and AFS31.

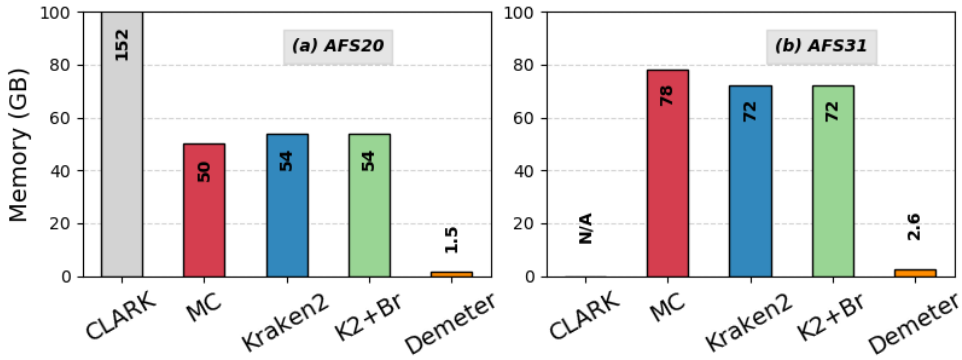

Figure 7.6: Required memory for (a) AFS20 and (b) AFS31.

We make the following two observations. First, Demeter requires ~33x and 36x less memory than Kraken2 and MetaCache for AFS20 database and ~27x and 30x less memory for them for AFS31 database, respectively. This makes Demeter the most efficient food profiler from a memory usage perspective. Second, the reduction in memory requirement for Demeter is to the extent that, for the first time, the data structure of the food profiler can fit into a standard size memory and does not require a colossal RAM managing further queries. This reduction is the main enabler behind Acc-Demeter. We

conclude that Demeter is very memory efficient.

7.4. DEMETER'S PIM-ENABLED ACCELERATOR

Demeter is positioned as a platform-independent food profiling framework that uses HDC. Demeter works with large HD vectors, is robust against errors, enjoys high parallelism, and exploits simple operations. These characteristics make Demeter a suitable candidate for hardware acceleration. However, the interest behind accelerating Demeter in a highly parallelizable and energy-efficient platform and specifically a PIM-enabled design goes beyond being simply its suitability and is a requisite for such a platform with two main motives.

Motivation 1: As discussed in Section 7.3.3, a software version of Demeter incurs a considerable cost on copy operations among registers holding intermediate HD vectors and classification. It also performs the classification poorly due to larger than cache HD-RefDB and low cache hit rate. These costs diminish all the benefits of Demeter that come from its small data structures and memory requirement. However, one can prevent this if Demeter is implemented in hardware as they can (1) realize the shift operation for free by only redirecting the output of each register to the next one and (2) perform the classification efficiently.

Motivation 2: A software-based implementation of Demeter still incurs a lot of unnecessary data movement for Steps ❷, ❸, and ❹. A hardware accelerator, especially a PIM-enabled one, can mitigate this problem greatly.

Therefore, we propose a PIM-enabled hardware accelerator for Demeter using PCM cells. One can accelerate Demeter using a PIM-enabled design on different memory technologies. We choose a memristor-enabled design for three main reasons. First, it is well-known that memristor-based memory technologies can perform vector-matrix multiplication [410–413] using Kirchhoff's law efficiently, making them suitable for our design. In this work, we manage to propose a hybrid row-major/column-major data mapping and intelligent data duplication scheme to perform encoding, classification, and profiling efficiently on PCM devices using this operation. Other technologies than memristors do not offer the same features for our hybrid data mapping.

Second, traditional technologies, such as non-memristor-based ones, are generally general-purpose and cost-driven. Moreover, their design does not allow even simple circuit modifications without high penalty on the area and cost. This makes them face a lot of pushback from the industry and unlikely to see future adoption. One of the advantages of memristors over them is their high density and scalability, and previous works show a wide range of accelerators using them.

Third, researchers already show the potential of accelerators based on emerging technologies for other ML-based algorithms [108, 411]. Also, multiple memory technologies already exist in current sequence machines. Therefore, it is not unreasonable to imagine one sort of these emerging memory technologies also be installed in these machines, especially for performing ML-based algorithms such as those for base-calling that are necessary for the sequencers [179].

In this work, we focus on PCM devices, as a member of the family of memristor devices, due to our accessibility to accurate device measurements and models for these devices and leave exploring other technologies for future research.

7.4.1. OVERVIEW OF DEMETER'S ACCELERATOR

Fig. 7.7 shows an overview of the proposed PIM-enabled hardware accelerator for Demeter, Acc-Demeter. Acc-Demeter consists of 5 key elements: ❶ Item Memory (IM), ❷ Encoder, ❸ Associate Memory (AM), ❹ Distance calculator, and ❺ Controller. IM and AM units are memory units, and we implement them as PCM arrays with their control circuitry. However, the encoder and distance calculator units are computing units implemented as the periphery. The controller is a simple FSM designed to harmonize the required steps of Demeter. The CPU initiates Demeter by gathering the user's input (Step ❶) and then booting the controller; i.e., it sends the start command, initializes the registers, and sets the addresses to consider for food samples and/or reference genomes in the controller. In a nutshell, Acc-Demeter accelerates Steps ❷, ❸, and ❹ of Demeter. The controller returns the results of Step ❹ to the CPU for final processing and performing the relative abundance estimation (Step ❺). We discuss these units in more detail next.

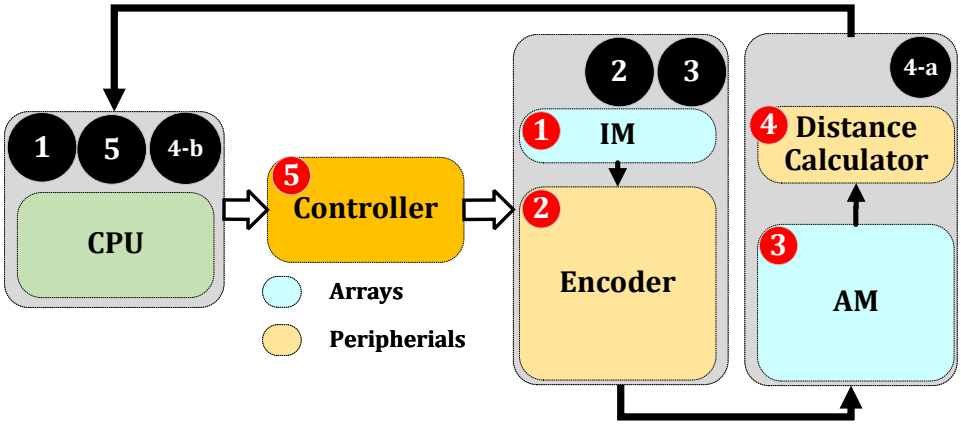

Figure 7.7: Overview of Demeter's in-memory accelerator.

7.4.2. ITEM MEMORY (IM) DESIGN

We implement our IM using PCM arrays and corresponding circuits, such as decoders. IM stores the atomic HD vectors. Binary "0" and binary "1" in an HD vector translate to amorphous and crystalline states, respectively. In the beginning, the user (or Demeter) generates 4 HD vectors for each DNA alphabet in Step ❶ of Demeter and stores them in the IM. Acc-Demeter reads these atomic HD vectors from IM every time it meets a new symbol. Once Demeter fixes the HD space, IM becomes a read-only memory. This allows us to prevent unwanted changes to the atomic vectors.

Fig. 7.8-(A) presents the IM design. The gate enabler provides access to cells that the row decoder activated. This way, the design of an entire array is achieved much easier, and the write/read disturbance effect is also mitigated to a great extent. However, this design also blocks the write on a row basis and only allows column-wise programming of IM. This does not complicate IM in any way because the atomic vectors are

generated once in the beginning by the host CPU and then stored in the IM for a long time. Note that random number generators are already well-optimized in CPUs. In addition, randomly generated values inside memristors are still in early stages [414–416], and Acc-Demeter can be modified later to benefit from a non-intrusive (compatible) random number generator in the future.

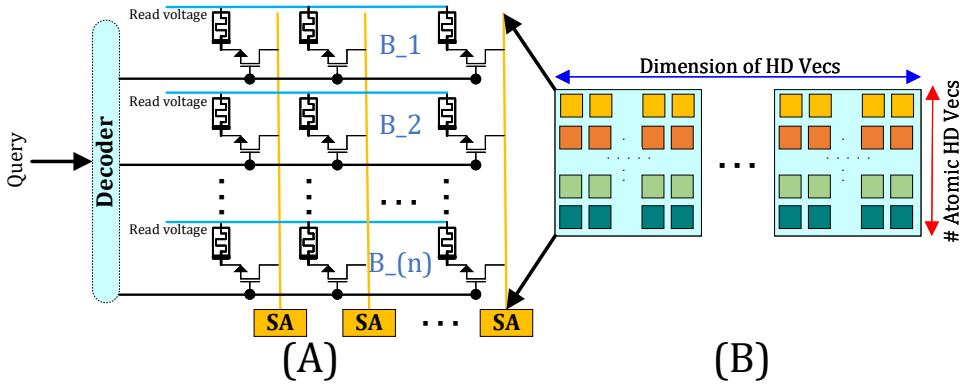

Figure 7.8: (A) IM design. (B) Data Mapping and placement of atomic HD vectors in IM.

Fig. 7.8-(B) presents (1) data mapping and (2) placement of HD vectors in the IM unit. Note that data mapping is a critical contribution of Acc-Demeter. Acc-Demeter uses a hybrid row-major and column-major data mapping for IM and AM units, respectively. IM enjoys a row-major data mapping for two reasons. First, a row-major data mapping of HD vectors allows Acc-Demeter to read the cells written in one row in one cycle. This is helpful as IM is used in the encoding procedure, which is the bottleneck. Second, the used PCM model provides more #columns than #rows. Therefore, even if there was a method to read column cells all at once but separately, one could only store smaller chunks of an HD vector on that column.

An important design choice regarding IM is related to the limited size of PCM arrays (512×2048 [296]). This limitation of array size (which also exists in mature memory technologies such as DRAM) prevents us from fitting an entire large HD vector in one row or column. Therefore, one needs to break such an HD vector into smaller chunks and store them into separate rows. Three options exist: (1) putting the chunks in the same array, (2) putting them in different arrays, (3) a hybrid approach. As shown in Section 7.6, encoder is the bottleneck of our operation. Therefore, to prevent exacerbating the overhead of the encoding procedure, IM breaks a HD vector to the largest power of two that is smaller than the number of columns available in an array (2048 in our case) and stores different chunks on different arrays. This is a direct tradeoff between the used area (#arrays) and performance. Fig. 7.8-(B) also shows this placement.

7.4.3. ENCODER DESIGN

The encoder is the main compute unit of Acc-Demeter. The encoder is implemented in the periphery of arrays and executes the binding and bundling operations via a sequence

of commands determined by the controller. Demeter is capable of handling different representations (Section 7.2). However, to reduce the complexity and make the design hardware friendly, the current design of Acc-Demeter only supports binary representations. In this setup, the N-gram encoding mechanism is the most common one, which Acc-Demeter supports. We suspect that other choices are also possible with the same hardware or minimal changes. We leave the exploration of those designs for future work.

Based on Equation 7.2, building an N-gram requires only simple XOR and shift operations. This bitwise XOR operation can be quickly computed after reading the atomic HD vector from the IM with an XOR gate in the periphery. Note that one can also implement XOR using bitwise AND ($\wedge$) and OR ($\vee$). However, this technique requires breaking the XOR operations into minterms whose numbers increase exponentially. Any attempt to reduce them, even if empirically works as in [296], will only produce approximated results and hurt the accuracy. Although some applications can tolerate such extreme accuracy loss, food profiling cannot. Note that the 2-minterm based encoding in [296] also affects the sparsity of N-grams (acknowledged in the paper) and limits the size of the N-gram. However, this is not the case in Acc-Demeter because all the operations accurately use XOR gates. This way, Acc-Demeter can benefit from larger N-grams and does not hurt the density of the HD vectors. As discussed in Section 7.3.3, the shift operation can quickly become a bottleneck for large HD vectors and strings in a software-based implementation. However, this does not happen here since Acc-Demeter realizes the shift for free by simply redirecting each Flip-Flop (FF)'s stored value to the neighboring one every clock cycle. Fig. 7.9 depicts a schematic illustration of the encoder unit.

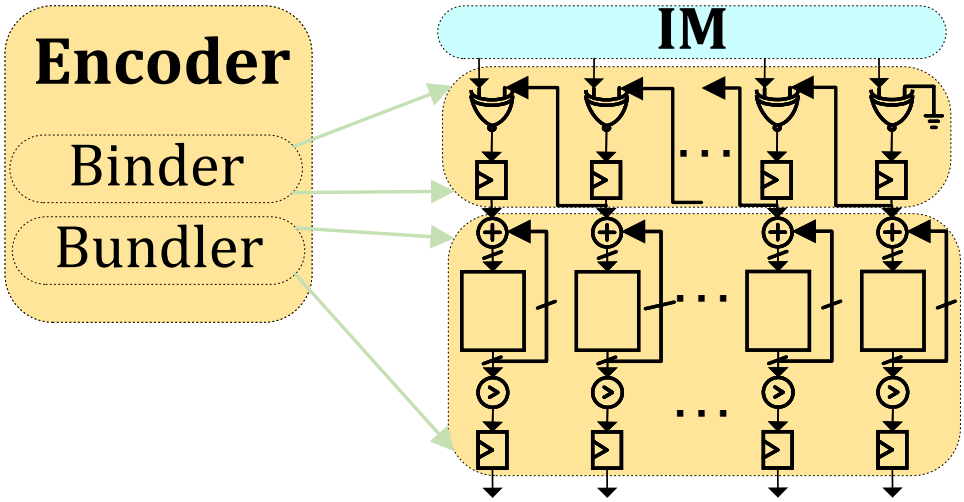

Figure 7.9: Encoder components and schematic.

From the hardware perspective, the encoder distinguishes the binding and bundling components completely. For the binding, the SA reads out the value from IM to one input of an XOR gate and uses the previously stored value of neighbor FFs as the second

input. The encoder then stores the results in a buffer and repeats the procedure. This design choice provides Acc-Demeter with the cascaded logical operations, with minimum changes to the memory array, and prevents any write back and pressuring the endurance of the PCM substrates. The encoder performs this sequence N times (enforced by the signals from the controller) to build an N-gram. After it finishes creating one N-gram, it passes the N-gram to the bundler unit, resets the buffer, and starts building the next N-gram until it hits either the last character of the input or set limit per final HD vector.

The bundler takes N-grams and adds them to a global HD vector that presents each position with a counter instead of only one bit per position. It then repeats this operation for M N-grams. Finally, the bundler applies a threshold (T) and makes a final binary HD vector representing all the processed characters while building this vector. At this point, the encoder is done. It passes the results to be stored as prototype HD vectors or used as query HD vector in AM and resets both the integer-based and binary HD vectors.

7.4.4. ASSOCIATE MEMORY (AM) DESIGN

The AM unit is implemented using PCM arrays and their corresponding circuitry, similar to the IM unit. This unit takes the output of the encoding mechanism (an HD vector) as input. Although the AM and the IM can technically be combined, Acc-Demeter considers separate hardware for three reasons. First, these units serve in subsequent and completely different steps in a profiling pipeline, naming encoding, and classification step. Such a distinct separation enables building a pipeline for them. Second, row-major and column-major data mapping in IM and introduce different parallelism opportunities for encoding and classification steps of a profiling pipeline, respectively. Row-major data mapping of IM parallelizes encoding of all bits in a single HD vector in each clock cycle. On the other hand, column-major mapping of parallelizes the similarity check of one query HD vector with all prototype HD vectors stored vertically in at that clock cycle. Third, separate hardware helps us to simplify IM design by using sense amplifiers instead of ADCs. Doing so brings various benefits in terms of area saving, energy consumption, and read-out time. Note that ADCs are usually the bottleneck of a memristor-based memory in terms of energy, area, and time [90] and that is why one only uses them when VMM or other logical operations such as Scouting Logic [106] are necessary.

Equation 7.3 shows that for the classification, we need to count the differences between the query HD vector and each prototype HD vector and then decide whether or not it can belong to the corresponding class. Although one can realize this in hardware by performing XNOR operation between the two vectors followed by a pop-count operation all in the periphery, such design comes with two drawbacks: (1) it requires the pop-count operation even after the XNOR, which introduces enormous area cost and significant delay ($log_2 D + 1$ cycles [389]), and (2) the AM unit, similar to the IM, only allows to write columns, not the rows. Since prototype HD vectors are not known from the beginning (unlike atomic HD vectors), this limitation forces us to save them all in another extra unit first and then write them back on a row basis. This is again inefficient.

However, Acc-Demeter proposes a new column-major data mapping and intelligent data duplication for this unit and exploits the characteristics of the PCM substrate to solve all these problems for HDC-based classification. It is well-known that memristor-based memory technologies can perform vector-matrix multiplication [410–413]. There-

fore, Acc-Demeter implements the required XNOR and following pop-count operations in Equation 7.3 in four steps, three of which happen in the unit and the last one in the Similarity Check unit.

Step 1: Acc-Demeter stores one prototype HD vector (or a chunk of one HD vector) in one column and its complement in the same column number of a second array. Fig. 7.10-A shows their placement in the AM unit. **Step 2:** Acc-Demeter applies the query HD vector (Q) to the rows of the first array and the complement of the query HD vector ($\bar{Q}$) to the rows of the second array with complement prototype HD vectors (Ps), shown in Fig. 7.10-B. **Step 3:** Acc-Demeter enables columns consecutively and effectively read out the number of ones in $Q.P$ and $\bar{Q}.\bar{P}$ in ADCs of each array. This way, it performs two vector-matrix multiplications using Kirchhoff's law, one between Q and all Ps in the first array and one between $\bar{Q}$ and all $\bar{P}s$. Section 7.4.5 describes **Step 4** that realizes XNOR and pop-count operation simultaneously. Fig. 7.10-B presents a high-level illustration of AM design.

Similar to the case in IM, the limited array size of PCM substrates also prevents Acc-Demeter from storing a full HD vector in one row or column of AM. To reduce the required area, and since the encoding is the bottleneck and not the classification (Section 7.6), in the AM, unlike IM, Acc-Demeter stores the chunks of HD vectors in the same array. Fig. 7.10-A takes a color-coding approach and depicts the way Acc-Demeter breaks prototype HD vectors into multiple chunks and stores them in columns of AM in and among tiles. It is worth noting that Acc-Demeter only writes to the PCM cells once in both IM and AM units unless either the configuration file in Step ❶ or the user the default reference genome database in Step ❷ changes. This prevents many writes to the devices, which still have limited endurance compared to traditional memory technologies.

7

7.4.5. SIMILARITY CHECK HARDWARE

The similarity check unit is a small computing unit that takes the two ADCs' output of similar columns from the two crossbars and adds them together (**Step 4**). Fig. 7.10-C depicts all the logic for this unit. The output of this unit is the results of XNOR and pop-count together. At this stage, the similarity check unit sends the results out to the host CPU to determine whether the similarity is close to the threshold and should be considered in the abundance estimation (❹-b, and ❺). The reason behind sending the results out instead of a winner-take-all (WTA) circuit used in previous works [296, 417] is two-folded. First, a WTA circuit assumes that the query matches one and only one prototype HD vector. However, as discussed in Section 7.2.4 and 7.2.5, this is not always the case when profiling the genomics data. Second, the relative abundance estimation techniques (Step ❺ in Fig. 7.1), although simple, require more complex and area-hungry logic circuits, which Acc-Demeter aims to avoid whenever possible. Therefore, since the results will be analyzed outside the PCM-substrate anyway and transferring such small data can be easily handled by interconnects between the host and Acc-Demeter, Acc-Demeter relies on the host CPU to perform the final steps of Demeter. Note that the host is aware of prototype HD vectors' mappings.

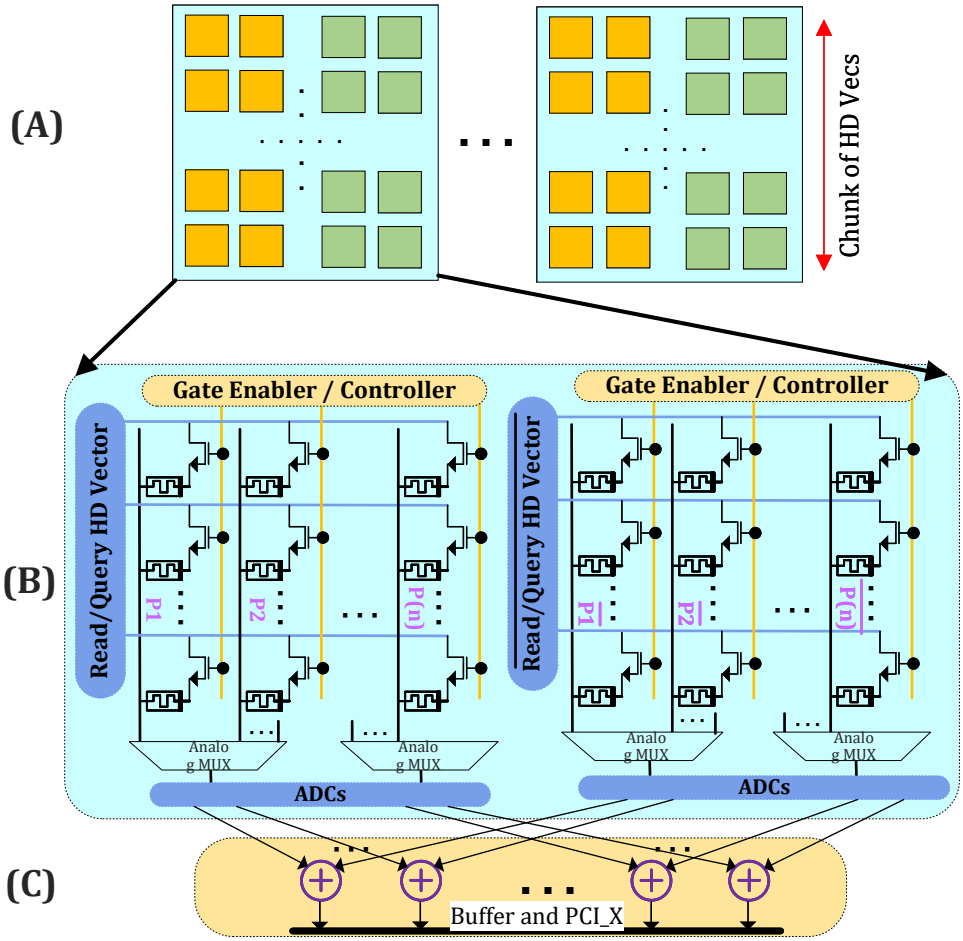

Figure 7.10: (A) Data mapping and placement of prototype HD vectors in , (B) High-level design, and (C) Partial hardware for Similarity Check unit.

7.4.6. CONTROLLER UNIT

The controller orchestrates all the operations of Acc-Demeter by generating control signals for other components. It gets the start signal and the address of food samples (or reference genomes) in the memory as its inputs. The controller outputs the results of the similarity check unit back to the host for final steps. The controller is designed as a simple FSM machine and operates based on parameters set in Step ❶.

7.5. SYSTEM INTEGRATION OF ACC-DEMETER

This section discussed Acc-Demeter's system integration stack that enables it to operate with the host processing system.

7.5.1. ADDRESS TRANSLATION

Acc-Demeter works with physical addresses, instead of virtual ones, and is relieved of address translation challenges that exist and dealt with in previous works [418, 419]. The CPU host uses the same translation lookaside buffer (TLB) lookup mechanism that exist for normal load/store operations to translate instructions's virtual memory addresses into their physical addresses when we have a Acc-Demeter's instruction.

7.5.2. COHERENCE

Acc-Demeter may require modified and/or generated atomic vectors (for the IM units) or loaded prototype vectors (for the AM units). Similar to previous works [372, 420, 421], ensuring that data for Acc-Demeter is up-to-date is a responsibility for programmers and can be achieved easily by flushing cache lines. Acc-Demeter is also capable of leveraging previous PIM coherence optimizations [422, 423] for further performance improvement.

7.5.3. INTERRUPTS

We assume that the pages required by Acc-Demeter's AM and IM units are already present. When this is not the case, we rely on the conventional mechanisms for handling the page faults to place this data into the correct arrays. Therefore, Acc-Demeter does not face page fault during the execution of food profiling since pages used by Acc-Demeter are already loaded and pinned into AM and IM units. Acc-Demeter may, however, face an interrupt during a context switch. In such cases, the context of the control unit in Acc-Demeter will be saved and then restored when the profiler resumes.

7.5.4. ISA EXTENSIONS AND PROGRAMMING INTERFACE

An expressive and efficient programming interface is a must for Acc-Demeter as it directly impacts the usability of Demeter framework. To enable easy communication between Acc-Demeter and the programmer, we envision to extend the ISA with a few instructions to allow the control unit knows about the required operations, their timing, and the place where data objects reside in IM and AM units. ISA extension is possible due to the unused opcode space in the host CPU, and has also been adopted in previous PIM-related architectures [93, 418].

Acc-Demeter requires 2 types of instructions: (1) *bbop_init address, size, n*: initialization of IM and AM units and (2) *bbop_op size, n*: instructions for performing differ-

ent operations in Acc-Demeter. *bbop_init* is the initialization instruction that informs the OS that the memory object is for Acc-Demeter. This way, the OS performs virtual-to-physical memory mapping required for AM and IM units. *bbop_init* takes the base physical address, the size of the vector, and the intended value. For Acc-Demeter's operations, we extend the CPU ISA with *bbop_op*. Acc-Demeter utilizes an array-based computation model, i.e., *src* and *dst* are the source and destination arrays. *bbop_op* is the opcode, where *size* and *n* are #elements in the array and #bits in each array element, respectively. This chapter assumes that the programmer will write suitable code for Acc-Demeter operations manually. We summarized the required CPU ISA extensions for these operations in Table 7.1.

Type	ISA Format
Initialization	*bbop_init, address, size, n*
Input Operation	*bbop_op, size, n*

Table 7.1: Acc-Demeter ISA Extensions.

7.6. ACC-DEMETER'S EVALUATION

7.6.1. METHODOLOGY

We emulate the execution of Acc-Demeter using a cycle-accurate RTL model and synthesized it using UMC 65 nm technology node in Synopsys Design Compiler [331]. We verify the correct behavior of our memory model using test benches and previous in-memory simulators [296, 411]. We consider a typical operation condition of temperature 25°and voltage 1.2V when evaluating our energy consumption. All the experiments for the PCM-based Acc-Demeter are carried out based on PCM statistical models that capture the variations in the spatiotemporal conductivity of the devices. PCM prototypes and analytical models used for validation and further simulations are based on the results of EU project MNEMOSENE [333], led and concluded by TU Delft in 2020. Table 7.2 shows the other parameters of our PCM crossbars.

Technology	PCM (512*2048 @1bit), Cell Size = 50 F^2
Current on Conducting Devices	0.1 µA
Read Voltage	0.1 V
Read/Write Latency	Read=2.8 ns, write=100 ns
ADC	9 bits resolution, 2 ns, 4 pJ per sample

Table 7.2: PCM configuration.

7.6.2. ACC-DEMETER'S PERFORMANCE ANALYSIS

This section compares the performance of SotA profilers compared to Acc-Demeter, our PIM-enabled accelerator design of Demeter.

BUILD TIME.

Fig. 7.11 shows the build time that each profiler takes to build its initial data structure for two reference databases AFS20 and AFS31.

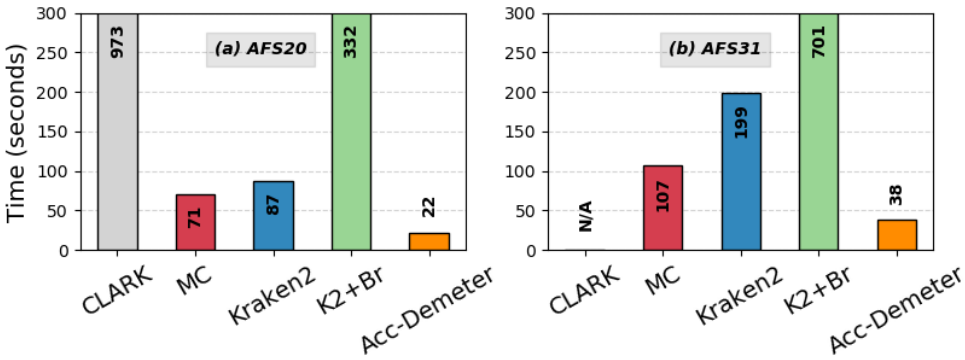

Figure 7.11: Build time on (a) AFS20 and (b) AFS31.

We make two observations. First, Acc-Demeter has the lowest build time among all previous food profilers. Acc-Demeter builds HD-RefDB corresponding to AFS20 and AFS31 ~3.2x and 2.8x faster, respectively than MetaCache, the next fastest profiler. Unlike previous HDC-based methods that are faster than their ML competitors due to the one-shot learning ability of HDC paradigm, Acc-Demeter outperforms SotA profilers due to its highly parallelized performance and simple operations being performed on Acc-Demeter's hardware. SotA food profilers parse the reference genomes only once, and the one-shot learning of Demeter is not particularly advantageous.

Second, CLARK exceeds the 500 GB memory of the system when running it for AFS31. This is in line with observations in [212]. Therefore, we excluded it from all analyses regarding AFS31 from now on. This case shows an excellent example of where metagenomic profilers, whereas good for lengthy and costly studies, may not be applicable for the scenario of food profiling and later food analysis and monitoring.

7

QUERY TIME.

Fig. 7.12 presents the time that each profiler takes to query one (short) read from the query food sample and classify its specie(s) over AFS20 and AFS31.

We make two key observations. Acc-Demeter improves the query time by ~74x/88x and 272x/350x compared to Kraken2 and MetaCache, respectively, on AFS20/AFS31. This shows that the acceleration of Demeter pays off and finally makes Demeter not only an accurate but also a fast food profiler.

Second, the query time for Acc-Demeter remains almost the same for both databases and does not change much. We further investigate this and realize a bottleneck shift: Step ❺ or abundance estimation that is being performed inside the CPU is now the bottleneck of Acc-Demeter. This happens because of the high-frequency Acc-Demeter achieved. However, this contrasts with other profilers that spend most of their time querying their massive data structure.

QUERY THROUGHPUT.

Fig. 7.13 shows the throughput of different profilers over AFS20 and ASF31.

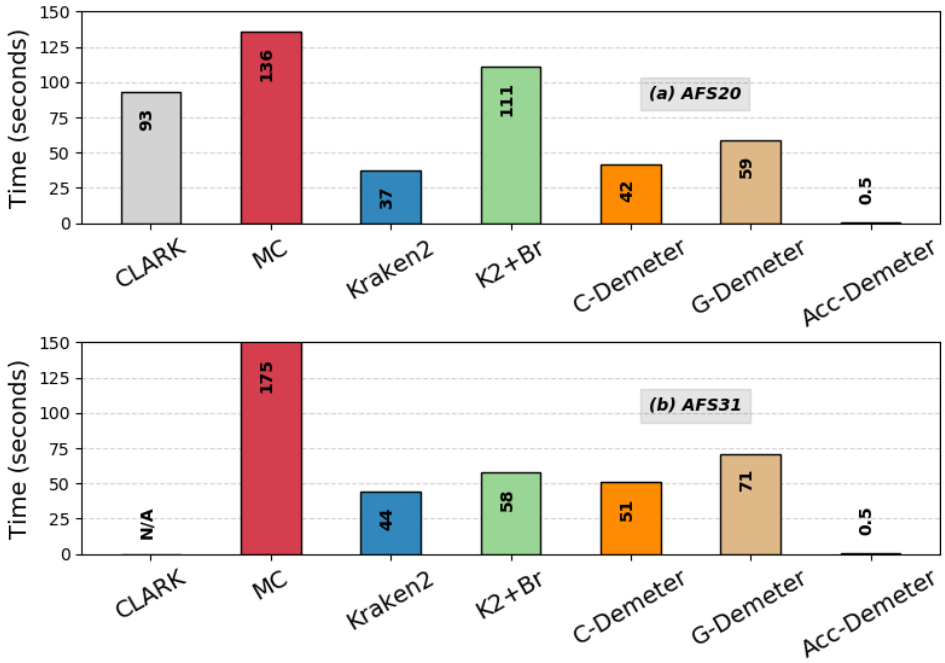

Figure 7.12: Query time on (a) AFS20 and (b) AFS31.

We make two observations. First, Acc-Demeter provides throughput improvement of ~192x and 232x for both AFS20 and AFS31, respectively, compared to Kraken2, the second food profiler regarding throughput. This more remarkable improvement in throughput than query time results from Acc-Demeter's ability to classify one query read in parallel with the encoding of the following query. Note that the throughput analysis of the previous profiler does not consider the time for loading their data structure. Second, similar to the query time, throughput is almost the same regardless of the database due to the bottleneck shift. We conclude that Acc-Demeter significantly outperforms all four SotA baselines for all performance metrics.

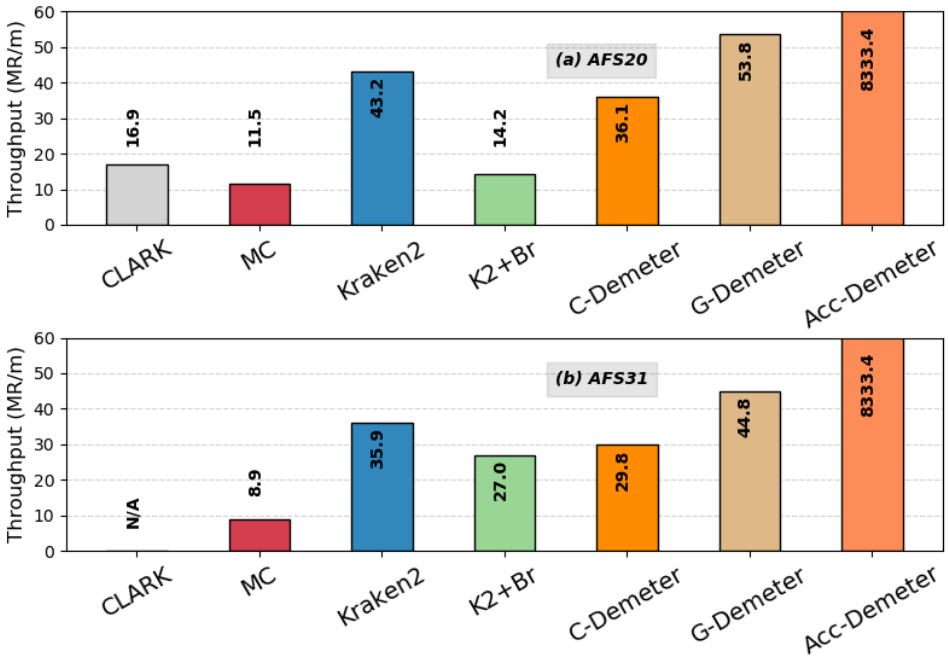

Figure 7.13: Query throughput on (a) AFS20 and (b) AFS31.

7.6.3. ACC-DEMETER'S POWER AND AREA ANALYSIS

Table 7.3 provides the area and energy consumption breakdown of different components in Acc-Demeter per query on AFS31.

Unit	Area (mm^2)	Area (%)	Energy (nj)	Energy (%)
IM	0.07	3.1	1.179E-06	7.4
Encoder	1.375	78.3	1.43E-05	90.6
AM	0.15	8.4	2.47E-07	1.56
Similarity	0.1815	10.2	6.91E-08	0.4

Table 7.3: Area and power breakdown of Acc-Demeter.

We make two observations. First, the logic for the encoder unit is the most energy and area hungry unit among all others, more than 90% and 78% energy and area of the whole Acc-Demeter. This is expected because (1) the encoder consists of many CMOS circuits whereas AM and IM are small memory units with PCM technology and (2) the encoder is in the heart of all operations in Demeter, and we spend most of our time in this unit. We argue that this amount of logic around our array is still justifiable. Second, compared to the die area in an Intel Xeon E5-2697 CPU [424], Acc-Demeter only has an area overhead of less than 2%. We conclude that Acc-Demeter is low-cost in terms of die area.

Our evaluations show that Acc-Demeter is capable of performing 9.45Mbp query per joule. Unfortunately, measuring the energy consumption of other profilers and having an apple-to-apple comparison between the energy consumption of this method with other ones is hard. However, Merelli et al. [425, 426] show that running Kraken2 with querying an even smaller data structure built from a reduced reference genome dataset, minikraken [425, 427], can incur more energy (maximum of $0.6\frac{Mbp}{j}$). This considerable difference happens because of three reasons: (1) Kraken2 queries a more complex data structure compared to Acc-Demeter and requires more complex operations, (2) Kraken2 queries a bigger data structure for its query, and (3) Kraken2 incurs significant data movement between the memory and the processing unit. All of these limitations exist in similar forms in CLARK and MetaCache. We conclude that Acc-Demeter is more energy-efficient than all four SotA baselines.

7.7. DISCUSSIONS AND FUTURE WORKS

Capacity. We define the capacity of Demeter as the ratio between the number of reference genomes encoded as prototype HD vectors to the size of HD space for a competitive profiling accuracy target. The higher #prototype HD vectors are, the bigger capacity is needed, resulting in bigger HD space and lower efficiency. Therefore, if one uses Demeter, as is, as a metagenomics profiler, they cannot expect similar improvements compared to SotA metagenomics profilers (e.g., Kraken2, on those datasets. We are currently investigating the additional techniques to enable Demeter for those cases as well. However, we leave further analysis of required changes to Demeter for supporting metagenomics profiling or other profiling studies with many reference genomes for future work.

Supported functions and representations. As discussed (Sections 7.1.3, 7.2, and 7.4), Acc-Demeter currently supports only binary representations and N-gram encoding mechanism. This is a design choice made for simplicity and is based on acceptable accuracy results of the software version. We leave the hardware for other encoding mechanisms and data representations for future work.

7.8. CONCLUSION

This chapter introduces Demeter, the first framework that enables profiling of food samples via HDC whereas strictly meeting the accuracy of state-of-the-art profilers. Demeter uses a five-step approach to enable species-level profiling using HDC. This chapter also introduces the first PCM-baed PIM-enabled hardware accelerator, called Acc-Demeter.

We evaluate Demeter on software and Acc-Demeter using a cycle-accurate model based on a small-scale PCM-based prototype. We design Demeter and Acc-Demeter to (1) address the key challenge of HDC-systems when facing a massive input, (2) eliminate the need for a powerful machine with very large memories, and (3) prevent unnecessary data movement between memory and processing units and therefore prevent wasting time and energy. We achieve significant performance and energy benefits over the SotA CPU implementations whereas achieving the same accuracy. We hope that future work builds on top of our framework and its hardware and extends it to further improve our food profiling systems.

8

KrakenOnMem: A Memristor-Augmented HW/SW Framework for Taxonomic Profiling

State-of-the-art taxonomic profilers that comprise the first step in larger-context metagenomic studies have proven to be computationally intensive, i.e., while accurate, they come at the cost of high latency and energy consumption. Table Lookup *operation is a primary bottleneck of today's profilers. In this chapter, we first propose TL-PIM, a hardware accelerator based on the processing-in-memory (PIM) paradigm to accelerate* Table Lookup. *TL-PIM leverages the in-memory compute capability of emerging memory technologies along with intelligent data mapping. Then, we integrate TL-PIM into Kraken2, a state-of-the-art metagenomic profiler, and build an HW/SW co-designed profiler, called KrakenOnMem. Results from a silicon-based prototype of our emerging memory validate the design and required operations on a smaller scale. Our large-scale calibrated simulations show that KrakenOnMem can provide an average of* 61.3% *speedup compared to original Kraken2 for end-to-end profiling. Additionally, our design improves the energy consumption by orders of magnitude compared to the original Kraken2 while incurring a negligible area overhead.*

This chapter is partially based on the candidate's work [121].

As discussed in Section 2.2.1, taxonomic profiling determines the relative abundance of existing species in a (biological) sample under study. It constitutes the first and most compute-intensive step of larger-context metagenomic studies (metagenomics for short). The goal of metagenomics is to better understand the role of each organism in our environment to improve our quality of life, e.g., by enhancing drugs [11]. The ability to improve the performance of taxonomic profilers will, therefore, have a huge impact on the overall speed of metagenomic studies and will remain a crucial line of research for decades to come.

Many recent works have improved the speed and/or accuracy of taxonomic profiling by various means, e.g., directly as heuristics in pre- and post-processing steps of profiling [192, 407], indirectly as pre-alignment filters [186] or innovative hardware designs for alignment [428–430]. However, the memory bandwidth and the (limited) cache capacities remain the two main bottlenecks even in these approaches [193, 212]. This is because *Table Lookup* (i.e., key matching and label retrieval) is a critical kernel in today's profilers and it is performed on data structures that are hundreds of gigabytes in size that cannot fit in caches of even high-performance computing (HPC) servers [193, 204, 244]. Note that it is also estimated that the working datasets that metagenomic studies should deal with scale faster than those produced by YouTube and Twitter by 2025 [25–27, 244], exacerbating this problem. Consequently, we need a fast, energy-efficient, scalable, and yet accurate design for taxonomic profilers (with an emphasis on their bottlenecks) to expedite the metagenomic studies and keep up with the fast data generation rate.

Our goal is to build the first hardware/software co-designed framework for taxonomic profiling that exploits *real* memristor (i.e., STT-MRAM) devices and the processing in-memory (PIM) paradigm. Using the PIM paradigm helps to prevent the high cost of data movement between memory and different levels of caches by performing the bottlenecked operations completely inside the memory, where the data resides. An in-memory solution can also scale up the active computational units without the need for expensive scale-up in the computational units of the server. **To this end,** we propose KrakenOnMem, an optimized framework for accelerating Kraken2[1] that notably improves execution time and energy consumption of taxonomic profiling with a negligible area overhead. KrakenOnMem is based on two main observations: (1) reference genomes rarely change, and (2) memristors are inherently capable of performing Vector-Matrix Multiplication (VMM). KrakenOnMem exploits these observations and addresses the bottlenecks of Kraken2 (and many other profilers), *Table Lookup*, using a memristor-based substrate, called TL-PIM hereafter. We perform an extensive design exploration for (1) data mapping, (2) logical operations, (3) array sizes, and (4) peripheral supports to optimize TL-PIM for *Table Lookup* and KrakenOnMem for taxonomic profiling based on Kraken2's algorithm. Our evaluations show that KrakenOnMem can provide up to 61.3% end-to-end speedup compared to the original Kraken2 implementation.

This chapter makes the following contributions:

- To our knowledge, KrakenOnMem is the first HW/SW co-designed framework to accelerate taxonomic profiling using memristor devices and the PIM paradigm. We design KrakenOnMem to target the key bottleneck of SotA taxonomic profilers.

[1] Kraken2 is currently the most widely-used and one of the most promising taxonomic profilers based on recent metagenomics challenges.

- We propose TL-PIM, an in-memory accelerator that executes *Table Lookup* efficiently using an intelligent data duplication and hybrid row-major and column-major data mapping. TL-PIM is designed to harvest the maximum parallelism and performance of the underlying hardware (Section 8.2).

- We rigorously compared TL-PIM and KrakenOnMem to (1) Kraken2 (open-sourced) as a SotA profiler and (2) Optimized Sieve as the latest in-memory accelerator[2]. We use a real small-scale prototype to validate our memory design. Our large-scale evaluations show that TL-PIM achieves an average 1386× and 111× speedup for *Table Lookup* operation compared to Kraken2 and Sieve, respectively. To capture the full potential of KrakenOnMem, we also perform a second set of analyses for end-to-end taxonomic profiling. KrakenOnMem achieves 61.3% and 1.17% speedup compared to Kraken2 and Sieve, respectively, for end-to-end taxonomic profiling. These improvements all come with the same level of accuracy as Kraken2 (Section 8.5).

- We investigate the possibility to adopt our designs in future (or non-heuristic-based) taxonomic profilers. TL-PIM integrated into Metalign, a SotA alignment-based taxonomic profiler, shows a 23.01% improvement for end-to-end profiling compared to original Metalign. This is achieved despite the fact that the bottleneck of Metalign does not lay on *Table Lookup* (Section 8.5).

8.1. MOTIVATION

In this section, we investigate the potential bottlenecks in Kraken2 and limitations of previous solutions for taxonomic profiling.

8.1.1. KRAKEN2'S EXECUTION BREAKDOWN

Methodology. We evaluate Kraken2 on a high-end server and measure the execution time of separate functions for end-to-end profiling of our input files. We use the default parameters of the tool for our study. Query reads come from the CAMI challenge, and Kraken2's standard (default) reference genome dataset is used for the references. We detail our evaluation methodology further in Section 8.4.

Results & Analysis. Fig. 8.1 depicts the percentage breakdown regarding execution time of Kraken2. We classify the various functions of Kraken2's implementation [431] into four main groups: (1) Building Taxonomy Tree, (2) Key Extraction, (3) *Table Lookup*, (4) Profiling. The building taxonomy tree function is run only once for each reference genome database. This function does not exist in the profiling phase and can be considered as a pre-processing function. We included the breakdown to show the relative time proportion to other frequently run functions. The key extraction function is responsible for reading the query files, extracting minimizers, performing hash functions, and producing the keys. The *Table Lookup* function tests each query key against all the keys in the table and returns the associate value (label) if it finds a match. The profiling function

[2] Sieve is a SotA k-mer (a substring of length k) matching accelerator that we tuned for a similar profiling approach.

is responsible for aggregating the results of the *Table Lookup* function and performs the final profiling processes alongside writing the data to a file.

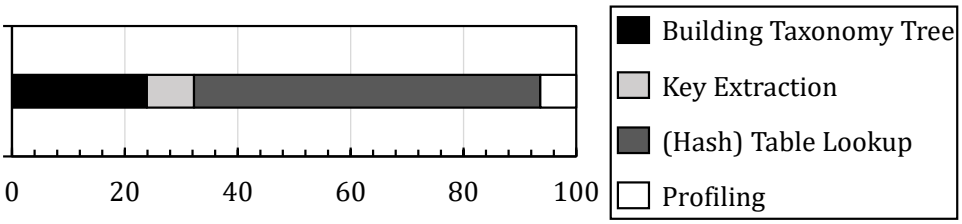

Figure 8.1: Kraken2's Execution Time Breakdown.

Based on Fig. 8.1, Kraken2 spends more than 60% of its total execution time on performing the *Table Lookup* function. Therefore, *Table Lookup* is currently the bottleneck of Kraken2 as a SotA taxonomic profiler. Our evaluations show that this humongous share does not change significantly as we increase the number of active threads in the system and *Table Lookup* remains the bottleneck of Kraken2.

Moreover, it is important to note that increasing the available memory bandwidth does not improve the performance of Kraken2 significantly [244]. This is simply because memory bandwidth is highly underutilized in the *Table Lookup* function as miss status holding registers (MSHR) in caches will be used up quickly and prevent using the memory bandwidth fully. Using cores with more MSHRs (e.g., Broadwell cores) is also not a suitable solution as they come with massive energy consumption (i.e., cost) and still waste DRAM bandwidth as Kraken2 still uses a small number of the retrieved cache lines for each *Table Lookup* [244].

We conclude that *Table Lookup* is currently the bottleneck in SotA Kraken2 profiler and will likely remain the main bottleneck of future non-alignment-based profilers for the same reasons unless hardware support is provided or profilers experience a cost-efficient, dramatic algorithmic change.

8.1.2. Limitation of Previous PIM-enabled Designs

A recent work, Sieve [244], proposes a high-throughput k-mer matching mechanism that uses in-DRAM processing. Sieve presents an Early Termination Mechanism (ETM) method that can interrupt the matching procedure of two k-mers as soon as the first mismatch occurs. This way Sieve can reduce the row activation required for its matching mechanism and reduce the latency and energy overheads compared to a naive implementation for k-mer matching. Technically, one can perform a *Table Lookup* function using Sieve's matching mechanism with a simple value retrieval approach. Therefore, it is reasonable to consider such an approach a suitable candidate for accelerating taxonomic profilers.

However, Sieve comes with four main limitations. First, high and unacceptable area overhead for a DRAM chip. Sieve requires up to 10.75% for its type III design which achieves the highest performance. Since DRAM chips are optimized for die area, this makes Sieve unlikely to be adopted in future systems. We compare Sieve's area with the

proposal in Section 8.5.2. Second, Sieve requires considerable data duplication and high #writes for vertical placement of each query k-mer and its duplicates inside group patterns in Region 1 defined in the original manuscript. Based on the examples provided in the original manuscript [244], this can be up to 4× higher than the actual number of query k-mers. Since query k-mers are extracted directly from reads/queries and vary per input sample, this is not a one-time cost and cannot be justified. Moreover, although the chosen memory technology in Sieve, namely DRAM, does not suffer from the endurance problem, such a decision still comes with endurance problems and a high cost (energy consumption and time) for each query. This limitation also prevents applying ideas presented by Sieve to emerging memory technologies that still suffer from low endurance. Third, Sieve only builds on DRAM as its underlying PIM infrastructure. Sieve justifies this by the technology maturity, availability of simulation tools, and cost advantages compared to SRAM. However, it left out exploring NVM-based technologies entirely. Such memory technologies have been the focus of many recent accelerators since they enjoy non-volatility, high-density, near-zero standby power, and low-cost logical operations [179, 432]. Fourth, Sieve incurs a significant amount of internal data movement associated with the multi-row activation needed for matching. This is unavoidable because Sieve requires copying the operand rows to designated ones.

We argue that Sieve's limitations are more than what can be expected for the cost of a taxonomic profiler preventing it from being adopted in future systems. Sieve's limitations and our experimental observations motivate us to develop an in-situ *Table Lookup* accelerator integrated with a host processing unit that accelerates taxonomic profiling. Our design has four key objectives: (1) It should provide high *Table Lookup* performance. (2). It should scale linearly with the required memory for Kraken2, rather than the parameters of the (hash) table. (3) It should not impose any significant overheads for its additional hardware, such as logic circuits in the periphery. (4) It should incur minimum #writes, data movement, and data duplication.

8.2. KRAKENONMEM DESIGN

The low arithmetic intensity and high energy inefficiency of *Table Lookup*, the primary bottlenecks in Kraken2, limit the maximum attainable performance and increase the energy consumption on server clusters typically used for profiling. This sub-optimal performance and energy consumption happens for three reasons. First, the extensive indexes used for taxonomic profiling. Second, irregular memory access and poor cache hit rate of profilers. Third, unnecessary data movement between memory (where the indexes initially reside) and the system's rigid cache hierarchy. In a nutshell, taxonomic profilers that use *Table Lookup* do not fully utilize the available bandwidth of memory systems [244] for their operation. We mitigate this problem by proposing a PIM-enabled accelerator for *Table Lookup* using memristor devices. We call this design TL-PIM hereafter. We integrate TL-PIM into a full system and propose an HW/SW co-designed framework for taxonomic profiling based on Kraken2's algorithm. This framework is called KrakenOnMem henceforth.

8.2.1. A High-Level Overview

Fig. 8.2 presents a high-level overview of our entire KrakenOnMem framework, i.e., TL-PIM and its integration with the host CPU and storage unit. KrakenOnMem consists of 4 main components: ❶ Host CPU, ❷ Main Memory, ❸ Storage, and ❹ TL-PIM. Current taxonomic profilers share the first three components with KrakenOnMem, i.e., KrakenOnMem only adds TL-PIM.

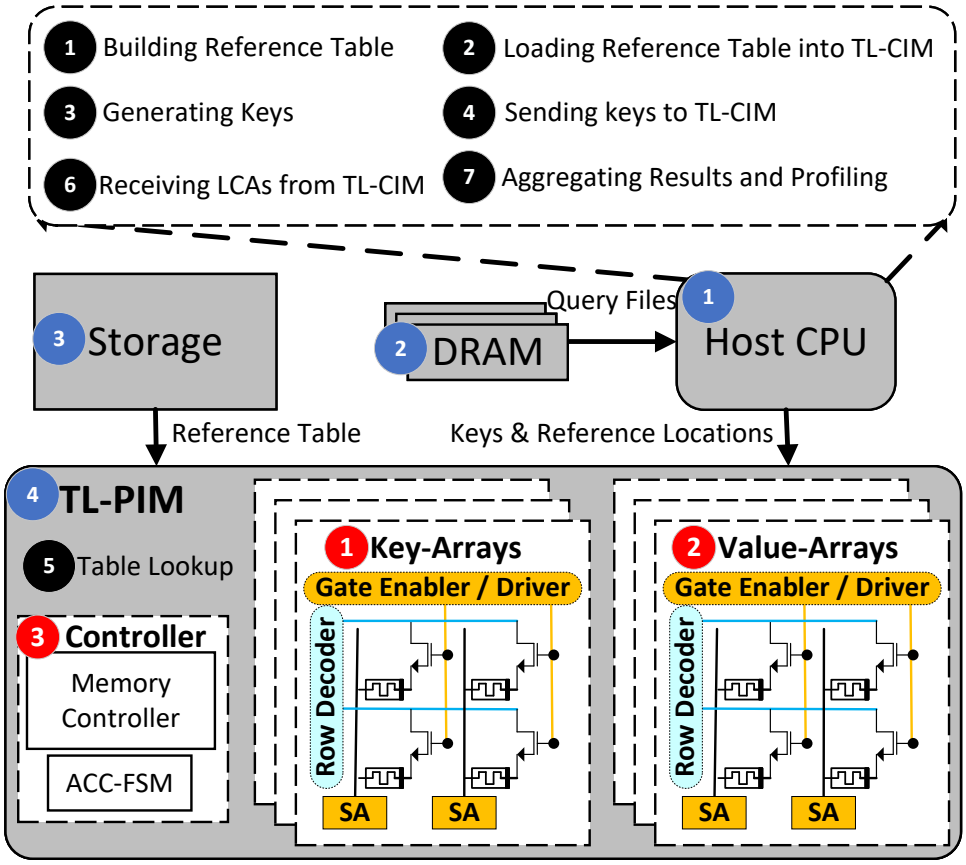

Figure 8.2: An Overview of KrakenOnMem.

Host CPU is responsible for the non-bottleneck steps of Kraken2. This includes ❶ building the reference (hash) table, ❷ loading the reference table into TL-PIM, ❸ reading the query read sequences from Fastq files and generating keys (e.g., extracting minimizers and calculating the hash values), and ❼ aggregating the retrieved taxonomic labels as the profiling result. Host CPU also sends the keys to TL-PIM (❹) and receives the results back from the TL-PIM (❻).

TL-PIM accelerates *Table Lookup* ❺ (the bottleneck) and subsequently helps the

overall performance and energy consumption. TL-PIM itself consists of 3 key components: (1) Key-Arrays: Memory arrays for matching units ❶, (2) LCA-Arrays: Memory arrays for taxonomic labels retrieval ❷, and (3) Controller ❸. Memory arrays are memristor arrays and their periphery circuits.

When designing the matching and retrieval units of TL-PIM, a designer faces three highly-correlated design choices and challenges, namely (1) data mapping, (2) matching mechanism, and (3) additional required hardware resources. For data mapping, TL-PIM should choose among possible options:

- Data layout: Row-major vs. Column-major
- Data distribution (i.e., keys or labels for queries or references): inter arrays vs. intra arrays vs. near arrays[3]

For the matching mechanism and its place, TL-PIM again have several options to choose from:

- Inside an array (e.g., using MAGIC [107] or other stateless methods)
- Outside the array but inside peripheries such as Sense Amplifiers (SAs) or Analoge Analog-to-Digital Converter (ADCs) (e.g., Pinatubo [87] or other stateful methods)
- Outside but near the array using simple logic after SAs (e.g., XOR and shift registers)
- A hybrid of previous options

Finally, TL-PIM should consider the required additional hardware resources for the matching or retrieval. These resources are for:

- The general control flow logic
- Modified peripheries (SAs or ADCs in the case of scouting or VMM operation)
- Fine-grained and complex control logic for analog operations in stateless mechanisms

We consider all possible (and logical) combinations of such design choices. KrakenOnMem is built based on the most efficient designs for each component and we discuss the reasonings in Sections 8.2.2 and 8.2.3. The controller is the brain of TL-PIM and orchestrates all necessary operations (Section 8.2.4). All components are highly efficient regarding their performance, area, and power. We discuss these in detail in Section 8.5.

[3]The inter-array distribution is defined as when data (keys or LCA labels) is stored in a single memory array. On the other hand, the intra-array distribution is when query- and reference-related data is stored in different/separate memory arrays. The case for near array data distribution happens when either query keys/labels or reference keys/labels are in a separate buffer next to the memory array that stores the other.

8.2.2. TL-PIM: Matching Mechanism

For matching a query and reference in memory arrays, previous works took three main approaches:

1. Approach 1:
   (a) Store both (or at least the reference) inside the memory
   (b) Read out the stored values
   (c) Perform an XOR/XNOR using logical gates
   (d) Perform a pop-count to determine the exact match
2. Approach 2:
   (a) Store both inside the memory
   (b) Read them into a modified SAs/ADCs to perform XOR/XNOR (e.g., Ambit [93] for DRAM, Pinatubo [87] for memristors)
   (c) Perform a pop-count on the output of SAs/ADCs
3. Approach 3:
   (a) Store both inside the memory
   (b) Perform XOR/XNOR inside the memory using analog computing (e.g., MAGIC [107])
   (c) Read out the result
   (d) Perform a pop-count

Unfortunately, all these techniques have one or more shortcomings in matching two short sub-strings or keys. We provide two examples of the inefficiency of these methods. First, storing queries inside memory incurs unnecessary write operations that take time, waste energy, and hurt endurance. Note that the query keys will change every time one wants to profile a new metagenomic sample exacerbating the problem. Second, the necessary logical gates to perform the pop-count operation after having the results of XOR/XNOR are energy and area inefficient (e.g., a tree of logical OR gates) or time inefficient (e.g., shift registers and counters), depending on the implementation. Note that, as also stated in previous works [90, 296], the additional logic units in peripheries are already responsible for most of the energy and area consumption of the whole design.

We propose a 1-cycle key matching (XNOR and pop-count in one cycle) using memristor devices in a typical memory array structure to mitigate previous shortcomings without introducing a customized memory layout. We exploit the inherent capability of memristor-based memory arrays for performing VMM operations. We also exploit the fact that the XNOR of two keys ($k1 \odot k2$) is functionally equivalent to $k1.k2 + \bar{k1}.\bar{k2}$. Therefore, one can use a column-major data mapping to achieve the intended operations. We call these memory units Key-Arrays hereafter.

The VMM function takes the query key and its complement as the input vector and a matrix of reference keys each placed with their complements in a single column as

the input matrix. Fig. 8.3 demonstrates the placement of input vector and one column of input matrix. Subsequently, by performing VMM operation and reading the results out using an SA that uses a customized reference voltage, one can perform the pop-count of $(k1|\bar{k1}).(k2|\bar{k2})$ of multiple reference keys with a query key in one cycle. For this purpose, the reference of SA is set to recognize any current higher than length (key) as logical 1 and lower currents as logical 0. This way, a 1 in the output of an SA shows a match between the query key and the reference key in the same column, and a 0 shows a mismatch.

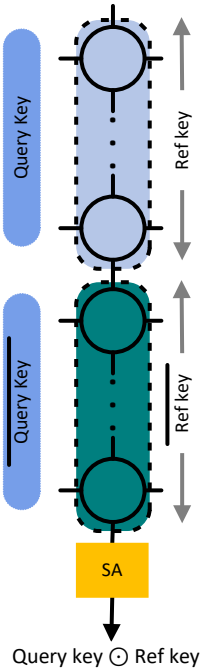

Figure 8.3: Matcher.

There are three points worth mentioning regarding our proposal. First, this method doubles the required memory cells for reference keys. However, as our evaluations in Section 8.5 and analysis in Section 8.1 show, our approach brings even further performance improvement than doubling (1) the whole available bandwidth and (2) compute power in the baselines. Additionally, a quick analysis of Kraken2 shows that reference keys are the smaller part of the hash table.

Second, unlike previous designs that require ADCs for their VMM operation, our method works with SAs. This simplification is because a Key-Array does not need to perform a complete VMM and get the exact number. We are simply interested in whether there is a match at all positions (which produces a current equal to the length of a key). ADCs represent a significant area overhead compared to the memory cells [90, 108]. Therefore, the current design overcomes a critical source of inefficiency if someone re-

places this design with those in previous works.

Third, in theory, the same results can be achieved using CAMs (but not TCAMs). However, using CAMs instead of our design have two main limitations TL-PIM aims to prevent: (1) The memory cells in a CAMs are different and fixed and cannot be reconfigured for other types of memory arrays (like those we use for taxonomic retrieval in Section 8.2.3). In other words, one cannot reuse Key-Arrays for other memory units if they use CAMs for matching. Such a choice limits the design, for example, for a case where one wants to support and load different reference tables; (2) CAMs require more complex controllers due to their different cell designs. Our design does not change the control circuits other than we already have in typical memories.

8.2.3. TL-PIM: TAXONOMIC RETRIEVAL

After finding a match between a query key and one of the stored reference keys in Key-Arrays, TL-PIM needs to retrieve the corresponding taxonomic label and send it back to the host CPU. LCA-Arrays are the memory units that store taxonomic labels. A LCA-Array uses row-major data mapping to store the labels. This way, it can retrieve the full label by only reading one row, which is impossible with a column-major mapping. In addition, LCA-Array applies a revised data mapping allowing the design to retrieve the label in 1 cycle. This mapping is due to the limitation of shared SAs among columns in emerging memories. Fig. 8.4 shows the proposed interleaved, row-major data mapping for each LCA-Array.

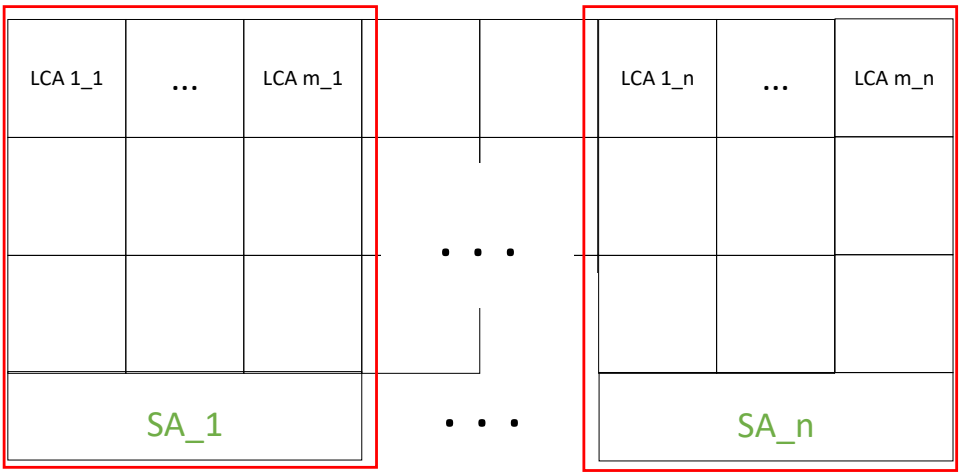

Figure 8.4: Taxonomic Label Retrieval.

Since SAs are shared among columns of a LCA-Array, to ensure TL-PIM can retrieve all the bits of one label in a single cycle, the LCA-Array needs to distribute the bits of a label and store them among columns that use different SAs. In other words, LCA-Arrays interleave bits of each LCA label among different SAs. $LCA\ X_Y$ in Fig. 8.4 presents the Yth bit of $LCA\ X$. For example, assume a 512×512 array where every 16 columns share one

SA. Additionally, assume that each taxonomic label has 17 bits. In this case, an LCA-Array puts the 1st bit of Label#1 in column#1, the 2nd bit of Label#1 in column#17 (=16+1), and 17th bit of Label#1 in column#273 (=17*16+1). It does the same for Label#2, i.e., it puts 1st bit of Label#2 in column#2, the 2nd bit of Label#2 in column#18 (=16+2), and so on. This way, for retrieving Label#1, TL-PIM only needs to read out the first bit of the first 17 SAs, which can be achieved in 1 cycle simultaneously.

8.2.4. TL-PIM: CONTROLLER

The Controller unit is the mind behind TL-PIM. The controller first receives the query key as a PCIe packet from the host CPU through the PCIe (Peripheral Component Interconnect Express) [433, 434]. Subsequently, it unpacks the package and distributes the query key to appropriate Key-Arrays. Additionally, the controller sends the proper signals to all memory units (Key-Arrays and LCA-Arrays). Examples of such signals are those for the VMM operation in Key-Arrays, select signals of MUXes in LCA-Arrays, and decoders' signals. Once TL-PIM compared the query key against all possible reference keys, it sets the PCIe interconnects response ready queue (RRQ). The finished requests will be forwarded to PCIe Out Queue (POQ). Each response can be either a taxonomic label or a NULL, meaning that the query key did not exist in the index table. The controller sends an interrupt signal to the host CPU when a packet is ready in POQ or empty slots in PIQ. The controller is a simple FSM machine and can be easily modified for future taxonomic profilers if the hardware requires the same sets of supported operations by other units.

8.2.5. RELATION BETWEEN LCA-ARRAYS AND KEY-ARRAYS

The ratio between the number of LCA-Arrays and Key-Arrays is not necessarily 1-to-1. A LCA-Array receives the results of N key-to-key comparisons per cycle per each Key-Array, where N is #SAs per Key-Array. Only one of all these N comparisons can be an exact match (Section 2.2.1). Therefore, in the worst case, or for the highest performance, TL-PIM needs to be able to retrieve the corresponding taxonomic label for that 1 match out of N possible cases in 1 cycle. This way TL-PIM can overlap retrieval operation of the previous matching with finding the next exact match of the same Key-Array. This scenario may require more than 1 LCA-Array per each Key-Array. In other words, the ratio between the required number of LCA-Arrays per each Key-Array is a design choice and tradeoff between the number of required LCA-Arrays and performance. This design choice also affects the utilization of each LCA-Array as each LCA-Array may end up not using all of its SAs or some columns in each SA (Section 8.5.2).

The ratio for the highest performance highly depends on the #SAs per Key-Array (or the number of evaluated keys per cycle), length of LCA labels, the capacity of SAs in a LCA-Array, and #SAs per LCA-Array. In other words, it depends not only on the device characteristics of memory arrays and their peripheries but also on the length of values (labels) in the reference table. This is the main reason that having the configurability between Key-Array and LCA-Array is favorable, and we use a typical memory layout for Key-Arrays instead of CAMs (Section 8.2.2). Fig. 8.5 presents a case were TL-PIM uses 2 LCA-Arrays per each Key-Array for achieving the highest performance.

Fig. 8.5 also demonstrates the expected utilization for Key-Arrays and LCA-Arrays by

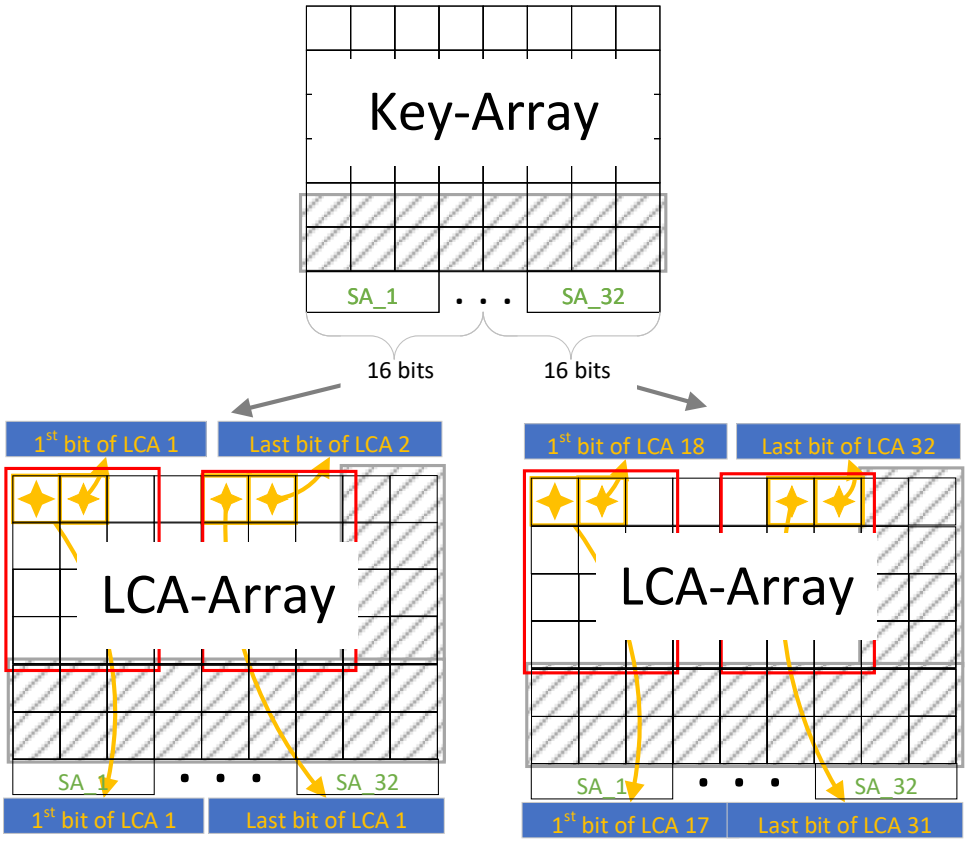

Figure 8.5: Relation and connection between one Key-Array and multiple LCA-Arrays.

diagonal gray patterns. For minimizing this inefficiency in Key-Arrays, TL-PIM places keys and their complement in each column, i.e., all columns are used. However, depending on the #Rows and required bits per key, TL-PIM can only fill $\lfloor \frac{\#Rows}{bits\ per\ key} \rfloor$. This means that some rows will remain empty (underutilized). For LCA-Arrays, memory utilization is lower, especially when opting for the maximum achievable performance. This means each LCA-Array may have to leave out a few SAs or columns of each SA depending on the number of results produced by corresponding Key-Array in each cycle, size of LCA-Array itself, and #columns that share SAs in the LCA-Array. Each LCA-Array may also have to not use some of its rows depending on expected #keys checked by corresponding Key-Array in total. We evaluate array utilizations in Section 8.5.2.

8.2.6. OPTIMIZATIONS

We apply several design optimizations to further boost the performance and/or energy consumption of our framework that we discuss here.

Optimization 1. We did not add any buffer or additional network among memory arrays in TL-PIM other than those that exist in typical memories. As alluded before (Sections 8.2.2 and 8.2.3), each Key-Array produces at most 1 bit consumed by corresponding LCA-Arrays. In addition, not all LCA-Arrays will consume the results of each Key-Array. This means that TL-PIM requires no network among its arrays. This allows TL-PIM to be less constrained regarding the area budget compared to all previous ML-related PIM-enabled architectures using memristor devices [90, 108].

Optimization 2. KrakenOnMem prevents broadcasting each query key to all Key-Arrays, and subsequently, it decreases the data movement in TL-PIM significantly. To this end, KrakenOnMem sorts the required hash table based on the keys alphanumerically and then stores them in Key-Arrays of TL-PIM according. Similar to Sieve [244], KrakenOnMem stores an 8-byte ID consisting of first and last reference keys in an array. Such a table will remain under 2 MB for a 500 GB reference table, and the host CPU or controller can easily store it. For each query key, KrakenOnMem first consults this table to find the correct Key-Array to send the request to in TL-PIM. This mechanism scales linearly with the size of memory considered for KrakenOnMem (equivalently the size of the hash table for references) rather than the length of the considered keys.

Optimization 3. We clock gate the LCA-Arrays that have no potential match for a particular query key. This is possible since we only get at most one output "1" from all Key-Arrays in each cycle because every query key can match only with one reference key by definition of the key in a hash table. This optimization saves the static energy of our system.

8.2.7. KRAKENONMEM PROFILING WALK THROUGH

KrakenOnMem performs an accurate and high-performance taxonomic profiling. In the boot up, the host CPU of KrakenOnMem loads the reference indexes (hash table) into TL-PIM's memory units based their required data mapping (Sections 8.2.2, 8.2.3). This is a one-time job, and we do not need to repeat it unless one changes the reference database that rarely happens. Fig. 8.6 summarizes how KrakenOnMem translates an existing Kraken2's database into appropriate data mapping in Key-Arrays and LCA-Arrays.

At the recipient of a metagenomics sample, the host CPU in KrakenOnMem reads

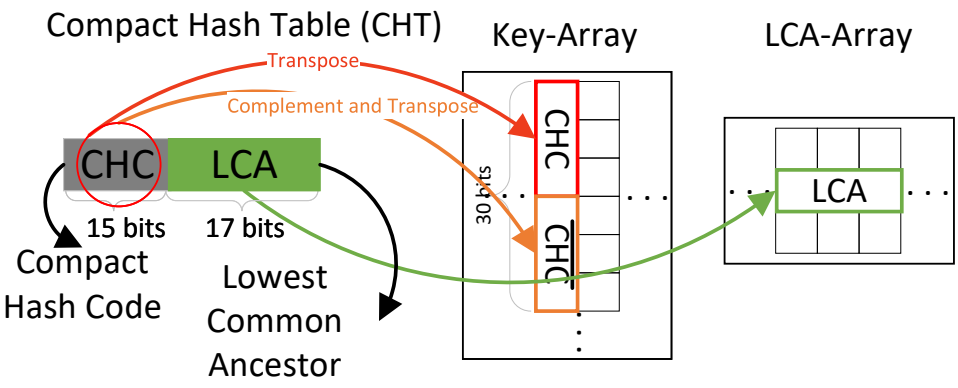

Figure 8.6: Mapping of Kraken2's Hash Table into TL-PIM for KrakenOnMem.

queries from Fastq files, extracts the k-mers, and calculates the corresponding keys. It then transfers the results to TL-PIM as queries. TL-PIM connects to the CPU host using PCIe [433, 434]. KrakenOnMem uses this connection to (1) send TL-PIM the necessary signals and query keys and (2) receive taxonomic labels from TL-PIM. KrakenOnMem hides the transfer latency of PCIe between host CPU and TL-PIM using the double-buffering technique [433]. TL-PIM unpacks the PCIe packets it gets from the PCIe input queue and distributes the keys to possible target Key-Arrays. After retrieving a taxonomic label, TL-PIM creates a response packet and stores it in PCIe's RRQ. A batch of these packets will be sent to POQ and eventually to the host CPU. The host CPU reads the response and aggregates them to achieve the final profiling of the read a key belongs to.

8

8.3. DISCUSSIONS AND FUTURE WORKS

Higher Memory Utilization. KrakenOnMem sacrifices memory utilization for performance, especially in LCA-Arrays. To ensure that KrakenOnMem can perform the *Table Lookup* operation in 1 cycle for the worst-case scenario, it has to leave some cells of the LCA-Arrays unused. However, it is possible to reach a higher utilization without a considerable performance loss. Currently, KrakenOnMem opts for this solution to provide the maximum performance one can expect. Different data layouts for LCA-Arrays can be investigated as future works for applying the same idea to other profilers or applications.

Optimization Possibilities for Rank-Level Profiling. KrakenOnMem can potentially improve profiling performance at different taxonomic ranks as well. An example of such optimizations can be the placement of keys related to a particular species on the columns of a Key-Array that do not share their SAs. This way, one can investigate the match of the query key to that species in 1 cycle. However, we leave the investigation of such optimizations for future work.

More Application Support. KrakenOnMem focuses on taxonomic profiling due to its importance and promise of being adopted with the newest technologies rather than the importance of general *Table Lookup* itself. However, TL-PIM from KrakenOnMem

can also be used in any other application bottlenecked by similar large (hash) tables. We leave the exploration of such applications and benefits TL-PIM and a similar HW/SW co-design to KrakenOnMem can provide to these applications for future work. The effectiveness, however, depends on how much of the issue this operation is in the original problem.

Support for TL-PIM Data Mapping. Currently, KrakenOnMem does not have API support for the required data mapping of flexible datasets and tables. We avoid virtual memory translation by mapping the KrakenOnMem's memory space directly to the CPU host. Therefore, currently, KrakenOnMem loads the required hash table to the TL-PIM memory arrays based on prior knowledge about the (hash) table in Kraken2. We leave building an API to support all data mapping required for an efficient *Table Lookup* for future work. Note that loading the hash table is a one-time (rare) task before starting the query operation. Since these hash tables rarely change and we usually reuse genomics databases, the cost of our approach is still acceptable for the intended long usage period.

Building Reference Table. Currently, KrakenOnMem does not build the required (hash) table or the taxonomy tree. Therefore, the current design always requires the (hash) table construction to be done first on another platform, which can be the primary host (CPU) used also in KrakenOnMem. However, this task is a one-time job and does not diminish the benefits of KrakenOnMem.

8.4. EVALUATION METHODOLOGY

We build KrakenOnMem based on Kraken2 to perform end-to-end taxonomic profiling. TL-PIM replaces the Kraken2's *Table Lookup* operation, which accelerates this significant bottleneck. We verify KrakenOnMem architecture using a cycle-accurate RTL model of the complete CMOS design with equivalent throughput based on the architecture in previous works [411, 413]. The memory model is validated and based on a small 4Gbit STT-MRAM chip prototype in TSMC 28nm CMOS technology [435, 436]. We use an analytical model based on this small prototype and extend the memory to the required size. The model is acquired from the results of the EU project MNEMOSENE [333], led and concluded by TU Delft in 2020. In other words, without access to large-scale production-level memristive devices, we evaluated our design using the next best approach: We implemented and prototyped many parts in FPGA and connected them to existing (small) memories and a high-performance AFE (analog -front-end) board for DACs, power supplies, and voltage and current references, to faithfully build first a small scale (using real size memory) and then an analytical model for our evaluation. We run all of our software experiments on a 128-core server running on AMD EPYC 7742 processors that operate at 2.25 GHz. We have 512 GB of 3200 MHz DDR4 DRAM available on this server.

Baselines. We compare KrakenOnMem mainly with Kraken2, the state-of-the-art taxonomic profiler. Kraken2 accompanied by Bracken [204] is one of the promising approaches for taxonomic profiling on different datasets once based on the latest results of the CAMI challenge [205]. We analyze the accuracy of KrakenOnMem by comparing its output results with only the profiling outputs of Kraken2. We also compare our platform with a state-of-the-art in-memory k-mer matcher, Sieve [244]. We consider Sieve only for the k-mer matching and query retrieval part (*Table Lookup*) and assume that the CPU takes care of the rest for a taxonomic profiling, similar to our platform. Finally, we

compare the proposal with Metalign, an alignment-based taxonomic profiler. This study demonstrates the potential of KrakenOnMem in general or TL-PIM in particular in other and future taxonomic profilers that are not bottlenecked by *Table Lookup* but still suffer from a similar inefficient operation.

Note that our evaluations do not include a GPU-based baseline for three reasons. First, currently, there is no GPU-based taxonomic profiler for metagenomics, let alone a GPU-based profiler based on Kraken2. Second, Sieve already outperforms GPUs in simple k-mer matching operation regarding performance and energy consumption. Third, GPUs are power-hungry and require significant data movement for reference index tables making them less likely to be adopted by the experts in the near future.

Performance Model. Kraken2 and Metalign open-sourced implementations report performance (execution time) directly when run on our servers. We use statistics of our synthesized design using TSMC 28nm technology node in Synopsys Design Compiler [331] to obtain the latency of main hardware components of KrakenOnMem, namely TL-PIM. We obtain the execution time of other steps by running each necessary steps of the Kraken2 on the host CPU. We take a similar approach for estimating Sieve's performance, considering improvements stated in the original paper for the matching.

Area and Power. Similar to performance, we also acquire the area and power consumption of KrakenOnMem's components from our synthesized design and memory model. This design considers a typical operation condition of temperature 25°and voltage 1.2V for power consumption evaluations. We measure the power consumed by our CPU host using Intel's PCM power utility [437].

Datasets. We use DustMasked MiniKraken for our reference database when testing our small prototype. This is a pre-built 4GB database from dustmasked bacterial, archaeal, and viral genomes in Refseq. For all other experiments, we used the default reference database of Kraken2 and Metalign. For query sequences, we use 3 datasets from the CAMI challenge: CAMI-low (RL), CAMI-medium (RM), and CAMI-high (RH).

8.5. EXPERIMENTAL RESULTS

KrakenOnMem produces the same list of matches, LCAs, and ultimately taxonomic profiling results as the original Kraken2. This was expected since KrakenOnMem does not change the order of steps in Kraken2 but it accelerates the bottleneck. Note that the order or rate at which we perform the matching and retrieval does not affect the profiling results as long as we ensure we have all the results before the final aggregation step. The same holds for the accuracy results of Sieve. From the results of the latest CAMI challenge [205], we know that Kraken2+Bracken stands among the high accuracy taxonomy profilers. Therefore, we conclude that KrakenOnMem also has high accuracy.

8.5.1. PERFORMANCE ANALYSIS

Our performance analysis consists of two separate sets of experiments. In the first set, we compare the performance of TL-PIM for *Table Lookup* with that in Sieve and Kraken2. Subsequently, we evaluate the end-to-end effect of our accelerator when employed for taxonomic profiling. In the second set, we slightly modify TL-PIM to be used for counting the matched k-mers instead of *Table Lookup*. Afterward, we compare this design

with KMC3 [438] used in Metalign. Finally, we evaluate the end-to-end effect of using such a design in Metalign.

***Table Lookup* and Heuristic-based Profiling.** Fig. 8.7 depicts the performance of Kraken2, Sieve, and TL-PIM when performing *Table Lookup*. The y-axis utilizes a logarithmic scale.

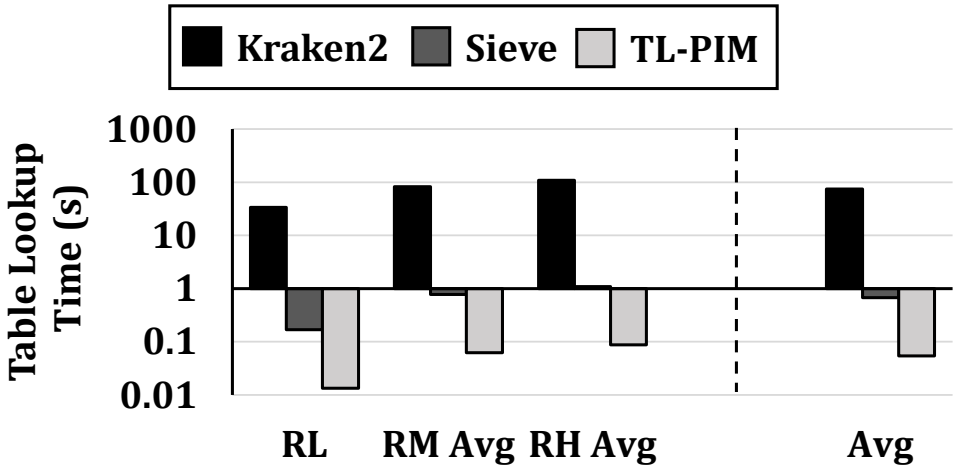

Figure 8.7: Performance comparison for *Table Lookup* between Kraken2, Sieve, and TL-PIM.

We make two observations. First, TL-PIM provides, on average, 1386× performance improvement over the original *Table Lookup* in Kraken2 and 111× performance improvement over Sieve for the same operation. Second, TL-PIM performs *Table Lookup* faster than Sieve and Kraken2 regardless of the dataset, on all three CAMI datasets by at least 100× and 1250×, respectively. TL-PIM outperforms original *Table Lookup* in Kraken2 due to its minimum data movement, high parallelism, and 1 cycle operation. It also outperforms Sieve due to the inefficiencies that Sieve introduces, such as duplicates and heavy internal data movement (Section 8.1.2). These results also show that ETM in Sieve, while effective, does not help Sieve outperform TL-PIM, even for queries with different complexity, such as those we used.

Fig. 8.8 presents end-to-end performance of taxonomic profiling for original Kraken2, Sieve, and KrakenOnMem. We observe that KrakenOnMem provides (1) 61.3% performance improvement over Kraken2, and (2) 1.17% performance improvement over Sieve. As expected, this is lower than improvements considering only *Table Lookup* because although this operation is the bottleneck in taxonomic profiling, KrakenOnMem still incurs some pre-processing and post-processing that become the new bottlenecks, reducing the overall benefit to some extent. In other words, we have diminishing returns due to the sequential nature of other sections of our application (i.e., Amdahl's Law).

Note that improvements of KrakenOnMem over Sieve are still significant for four reasons. First, although the overall speedup of KrakenOnMem compared to Sieve is small,

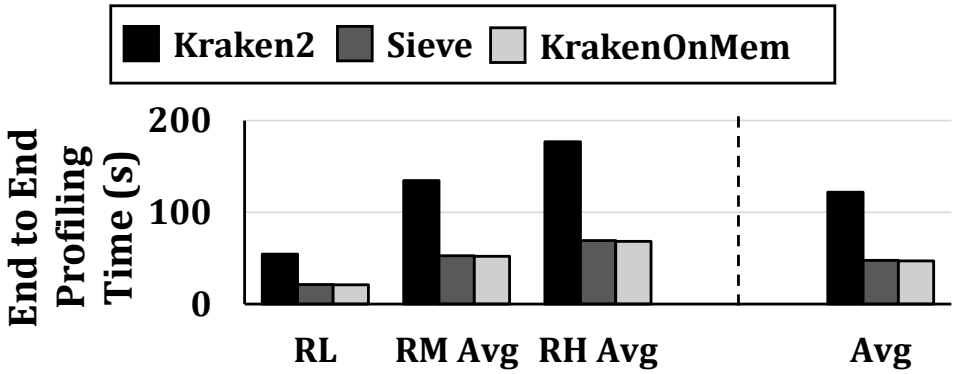

Figure 8.8: Performance comparison for profiling between Kraken2, Sieve, and KrakenOnMem.

the accelerated operation (*Table Lookup*) achieves a significant speedup. This opens up the possibility of future work focusing on the next performance bottleneck - a common (iterative) methodology in computer engineering. Second, *Table Lookup* will likely remain in future genomics pipelines for which the sequential part of the application can be different. Third, as we will discuss in Section 8.5.2, KrakenOnMem is also advantageous over Sieve in other aspects, e.g., area overhead and energy consumption. Fourth, the sped-up operation can and will potentially be used in all the upcoming metagenomics studies and profilers. Therefore, a modest 1.17% improvement can still translate into significant time and cost gains.

k-mer Counting and Alignmnet-based Profiling. Fig. 8.9 presents the results for pre-filtering stage in Metalign when tweaking TL-PIM to perform k-mer counting compared to the original KMC3 implementation in Metalign. The y-axis utilizes a logarithmic scale.

We observe that for all CAMI datasets, the PIM-enabled design provides on average a 1595× improvement compared to an SW version of the same operation in Metalign. We expected such significant improvements due to advances in data movement reduction and high parallelism in TL-PIM.

Fig. 8.10 demonstrates the aftermath when we apply the new TL-PIM for end-to-end taxonomic profiling using the Metalign approach as a SotA alignment-based profiler. We make two observations. First, our proposal provides, on average, 23.01% improvement in execution time for end-to-end profiling. This is expected as the k-mer counting operation in Metalign still affects the overall performance and cannot be masked or parallelized by other operations and steps.

Second, the end-to-end improvement in an alignment-based mechanism is much less than the achieved improvement over Kraken2 (23.01% vs. 61.3%). This happens because alignment-based taxonomic profilers are not bottlenecked by the *Table Lookup* operation, rather their required alignment operation. However, we argue that this improvement is still significant and shows that the *Table Lookup* procedure is costly for any type of profiler and worth the design, even if it is not the bottleneck.

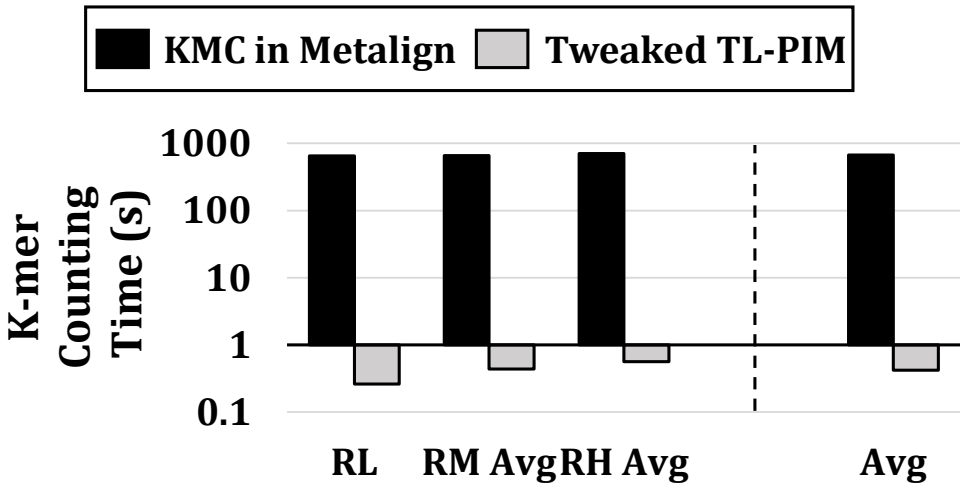

Figure 8.9: Performance comparison for k-mer counting between KMC and tweaked TL-PIM.

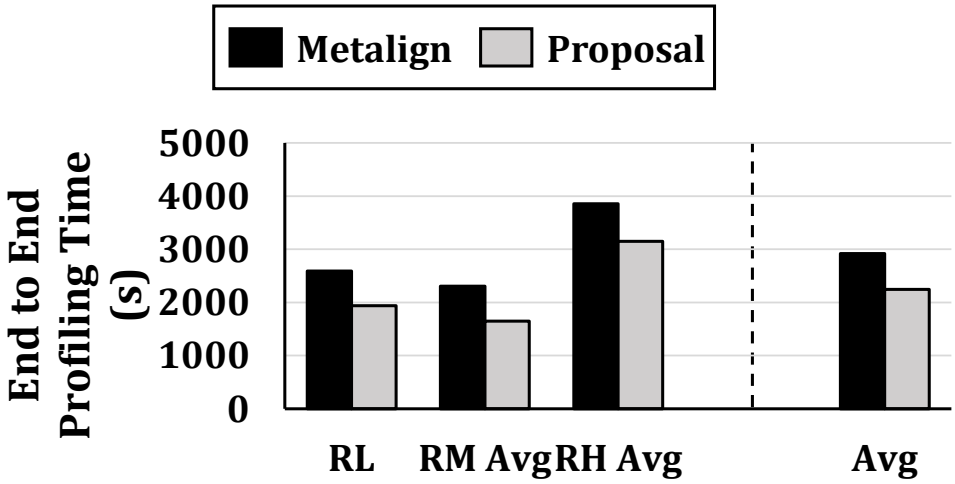

Figure 8.10: Performance comparison for profiling between original Metalign and Metalign equipped with tweaked TL-PIM.

8.5.2. POWER AND AREA ANALYSIS

Energy and Power. We compare the expected energy consumption of Sieve with the measured energy consumption of KrakenOnMem for *Table Lookup* operation. We consider the DRAM technology mentioned in the original manuscript for Sieve. We exploit three synthetic datasets to cover all spectrum of possible scenarios for Sieve, as its energy highly depends on the effect of the data-dependent ETM component. **Sc1** is when more than 95% of the mismatch/difference between query and reference keys exists in the first two characters of them. This scenario favors Sieve the most as the ETM component can save Sieve a lot of row activations and unnecessary comparisons. **Sc2** is the case where the average distance of the first mismatch among query and reference keys from their first bit is set to 10 bits. This is the main reported number in Sieve [244] for typical cases. The last scenario, **Sc3**, contains the results for when the mismatches cannot be found until the last two characters of query and reference keys. This is the worst-case scenario for Sieve, in which ETM favors the performance and energy the least. Fig. 8.11 presents the results.

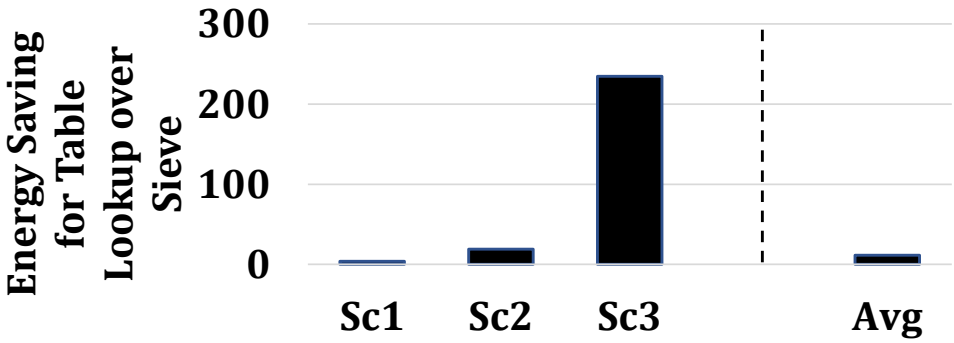

Figure 8.11: Energy saving of KrakenOnMem over Sieve for *Table Lookup*.

We make two observations. First, KrakenOnMem achieves a better energy consumption for all three scenarios, with an average of 11.34× energy saving over Sieve. One cycle matching and label retrieval, pipelined design, available parallelism, and analog computing are the reasons behind this significant improvement in energy consumption of *Table Lookup* in KrakenOnMem over Sieve. Second, Sieve offers a very data-dependent energy consumption that changes to an almost 62× depending on where the first mismatch among keys occurs. This shows that while ETM in Sieve might be compelling enough in average cases, it cannot be reliably used for a fixed/calculated energy consumption. KrakenOnMem solves this issue.

Our power evaluations show that KrakenOnMem can profile with a rate of $4.75\frac{Mbp}{j}$. This is while Merelli, et al. [425, 426] show that original Kraken2 on a 8-core XeonD processor can achieve only a maximum of $0.22\frac{Mbp}{j}$. Note that we have not directly compared the energy consumption of KrakenOnMem with Kraken2 for two reasons. First, our expertise in memory accelerators allows us to measure the expected profiling rate for

KrakenOnMem, but it would not be a fair comparison with a software-only solution. Second, only a few works [425, 426] provide energy numbers, and we utilized them for our rough energy comparisons with KrakenOnMem. These results showcase that KrakenOnMem consumes much lower energy for the same datasets than Kraken2.

Area. Although TL-PIM consists of memory arrays and controller logic, we only consider the area of its controller as additional area overhead. This is because sequencing machines are heterogeneous systems that already use various memory technologies (e.g., DRAM and SSD) and computational units (CPU, GPU, and FPGA) since their benefits justify their cost. Therefore, having memristors installed in those machines as well is not a far-fetched idea if we can harvest their power efficiently. The area overhead for the required controller in TL-PIM is 0.009 mm^2. This is a very modest overhead, only 0.002% of Skylake-SP, a modern Intel Processor at 14 nm [439]. Although any processor can be chosen for this comparison, depending on the final product, we pick the Skylake-SP processor for the area baseline only to be similar to works on PIM-based accelerators and non-volatile memories [231, 440]. Note that Sieve Type III, which we used in our performance and energy evaluations, incurs 10.9% area overhead for the required logic of (1) k-mer matching and (2) row-address latches over an 8-bank DRAM chip in 22nm technology mode. We use methods presented in [441] to scale the area consumption down to 28nm technology mode and find an overhead of higher than 6%. On the other hand, KrakenOnMem only incurs 0.0007% extra die area for its controller (as LCA- and Key-arrays exist regardless) over the same memory size using our STT-MRAM devices.

Memory utilization. Fig. 8.12 and Fig. 8.13 depict the memory utilization for Key-Arrays and LCA-Arrays, respectively, considering different array configurations, i.e., varying size and #SAs.

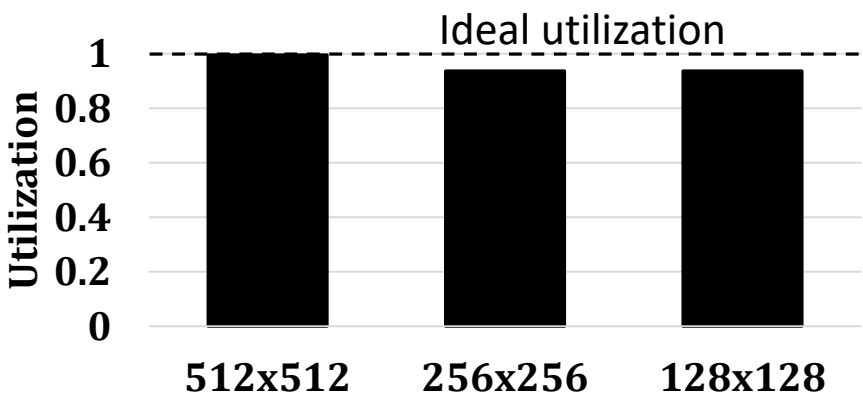

Figure 8.12: The memory utilization of Key-Arrays.

We make three observations. First, TL-PIM utilizes Key-Arrays close to the ideal case. For three standard evaluated array sizes of 512×512, 256×256, and 128×128, TL-PIM achieves a utilization higher than 93%. Second, utilization of LCA-Arrays is lower than that in Key-Arrays. However, they are still on average higher than 50%. This is expected in

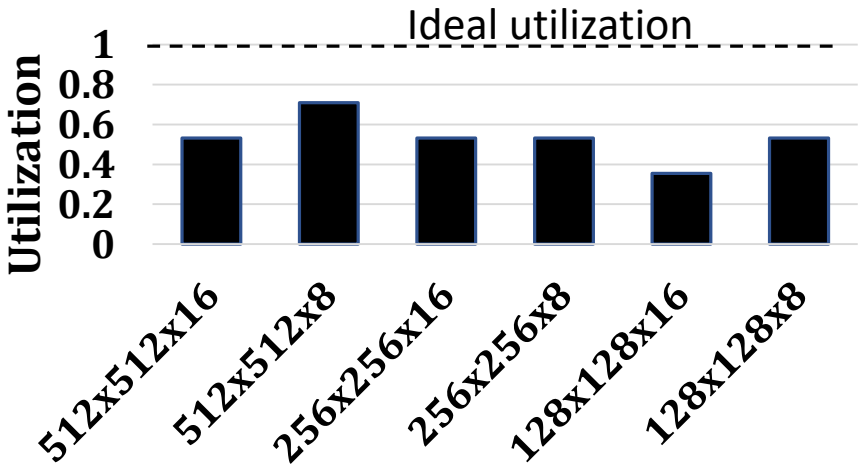

Figure 8.13: The memory utilization of LCA-Array.

any accelerator, including TL-PIM, that does not change the memory structure or data representation yet aims for the highest achievable performance. Third, utilization of Key-Arrays only depends on the array size, while LCA-Arrays' utilization is also affected by #columns that share one SA. The reason behind this is that the new parameters affect the number of LCA-Arrays per one Key-Array, as discussed in Section 8.2.5.

8.6. CONCLUSION

This chapter introduces KrakenOnMem, the first HW/SW co-designed framework for taxonomic profiling via in-memory hardware acceleration using emerging memory technologies. KrakenOnMem accelerates the bottleneck of Kraken2, a SotA taxonomic profiler, by a memristor-based PIM-enabled hardware called TL-PIM. TL-PIM enables *Table Lookup* operation while it simultaneously aims for (1) being data-independent, (2) maximizing the achievable performance for the worst-case scenarios, (3) having linear scalability, (4) maintaining its design optimization advantages for future designs, and (5) incurring minimal hardware and area overhead. The evaluation results show that TL-PIM alleviates the existing bottleneck to the extent that the end-to-end performance and energy consumption of Kraken2 significantly surpasses that of original Kraken2. We expect that ideas presented for TL-PIM and the HW/SW co-designed of Kraken2 enable the designs of future accelerators in other genomic applications. We also expect that the overall improvement in end-to-end taxonomic profiling introduced by KrakenOnMem further helps the upcoming metagenomic studies and opens new doors for improving our lives.

9

Lightspeed Binary Neural Networks using Memristor-based CIM Tiles

This chapter investigates the potential of the Computation-In-Memory paradigm using memristors to speed up and reduce the energy consumption of the Binary Neural Networks (BNNs). We propose a new data mapping for BNNs tailored for memory tiles capable of Vector-Matrix-Multiplication (VMM) operation. The preliminary results show a significant latency improvement irrespective of the evaluated network structure and size. The improvement varies from network to network and goes up to ~154×.

This chapter is partially based on the candidate's published and under review works [122, 123].

As discussed Chapter 2.2.2 in we have witnessed significant progress of Neural Network (NN) applications in the past few decades. However, this progress has, unfortunately, come with extensive demand for system resources, e.g., memory and energy consumption. This high demand has already slowed down the growth of active, deployed NNs and their adoption when factoring in the costs. A recent research direction is to use simpler (operation-wise) and smaller NNs, such as BNNs. BNNs enjoy lower memory requirements, simplified arithmetic operations, and near SotA accuracy on vision tasks [221, 222]. However, an optimized hardware implementation is still necessary for BNNs to smooth their cost-efficient adoption on our future systems.

A few works have investigated hardware realization of BNNs by introducing different mapping and data flow techniques [442, 443] or various circuitry and memristor-based crossbar structures (i.e., ReRAM or PCM) to perform required operations [444]. Unfortunately, none of these methods exploit the full potential of underlying emerging devices for BNNs due to inefficient data mapping and the sequential nature of how they perform the necessary operations.

This chapter presents two major contributions:

- TacitMap: A highly parallel data mapping for BNN operations on any CIM design capable of performing VMM, e.g., memristor-based crossbars such as electronic phase change memory-based (ePCM) or resistive random-access memory-based (ReRAM) ones. TacitMap is designed with the conventional 1T1R memory crossbar structure in mind and is therefore compatible with many of the already evolving crossbar architectures.

- A hardware acceleration of BNNs utilizing TacitMap instead of SotA data mapping, both on the same underlying memristor device technology, PCM.

- A detailed performance comparison between the proposed CMOS-compatible accelerator and previous SotA hardware accelerator for BNNs.

9.1. TACITMAP FOR BNN

To support necessary operations in Equation 2.5 (e.g., XNOR and Popcount) for hidden layers, we propose a data mapping for BNNs, called TacitMap. TacitMap requires an underlying technology inherently capable of VMM operation (e.g., memristor-based memory discussed in Section 2.1). Moreover, TacitMap is designed assuming the binary vectors of {0, 1} for activation and weight vectors. This eliminates the need for handling negative and positive weights separately [221]. Note that this approach understandably requires a one-time conversion of initially signed vectors to unsigned binary vectors of {0, 1}. Our evaluations (Section 10.3) take this overhead into account.

Fig. 9.1-(a) and -(b) present a comparison between how SotA mapping (hereafter called CustBinaryMap) [444] and TacitMap handles a single XNOR+Popcount of Equation 2.5, respectively, in the form of an example. For a detailed description of CSL, BL, WL, and SL, please refer to previous works [445, 446]. In Fig. 9.1, we assume input (In) and weight (W) vectors of length 2 bits. In_{X_Y} represent the Yth bit of Xth input. The same goes for other parameters. The bar on the parameters indicates the complement value. Note that other operations of Equation 2.5 (e.g., the multiplication

by 2 and the subtraction) are constant. Therefore, the mapping needs only to support XNOR+Popcount, and hardware can implement the constant operations (e.g., shift by 1) with minimum cost on the result of either mapping.

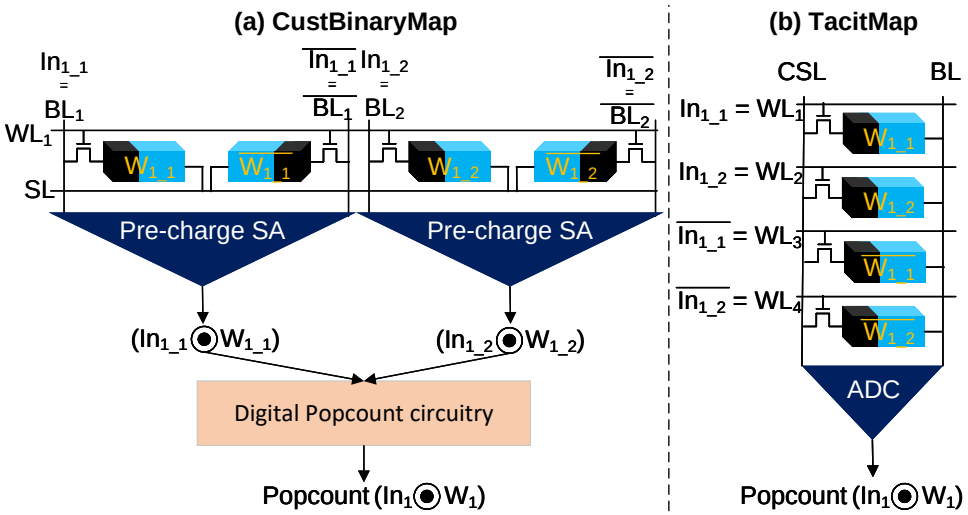

Figure 9.1: Concepts of TacitMap vs CustBinaryMap [444].

CustBinaryMap (Fig. 9.1-(a)) uses uses a 2T2R memory structure and places weight vectors *horizontally* in memory rows. Instead of storing the weight vectors as they are, this mapping requires the programmer to interleave the weight vectors and their complements in a bitwise manner and then either store every two bits of x and $\overline{x}$ in one of the devices in the 2T2R memory cell. In contrast, TacitMap (Fig. 9.1-(b)) uses a 1T1R memory structure and stores each weight vector *vertically* in a column. In TacitMap, instead of interleaving the weight vector with its complement, one first stores the weight vector and then, right below it, stores the complemented weight vector. Regarding the inputs, CustBinaryMap does the same interleaving of the input vector and its complement with the input vectors. The outputs are read through a modified SA called precharge sense amplifier (PCSA), which is the XNOR of the input vector and stored weight vector. Conversely, TacitMap concatenated the input vector and its complement and applied it to the rows and of the crossbar. The XNOR+Popcount is directly read out from the ADC. TacitMap offers three main benefits compared to CustBinaryMap:

- 1-step XNOR+Popcount in TacitMap compared to 2-step operation in CustBinaryMap. This enables TacitMap not to require any additional digital circuitry for Popcount.
- Column-wise XNOR+Popcount operation in TacitMap compared to row-wise operation in CustBinaryMap. This enables high parallelism opportunity for a design that utilizes TacitMap.
- Conventional μArch with multiple real-world chips (i.e., 1T1R cells + ADC [447, 448]) in TacitMap compared to heavily customized μArch (i.e., 2T2R cell structure with cus-

tomized SA) in CustBinaryMap. This makes TacitMap more suitable for extended and future hardware that might be used for BNNs.

Note that TacitMap and CustBinaryMap both use a similar number of memristor devices per 1 bit of weight kernel, i.e., they both use two devices to store both the bit and the complemented value.

To demonstrate the performance benefit of TacitMap against CustBinaryMap for performing BNN operations (i.e., multiple XNOR+Popcounts), Fig. 9.2 presents these mappings on the crossbar level.

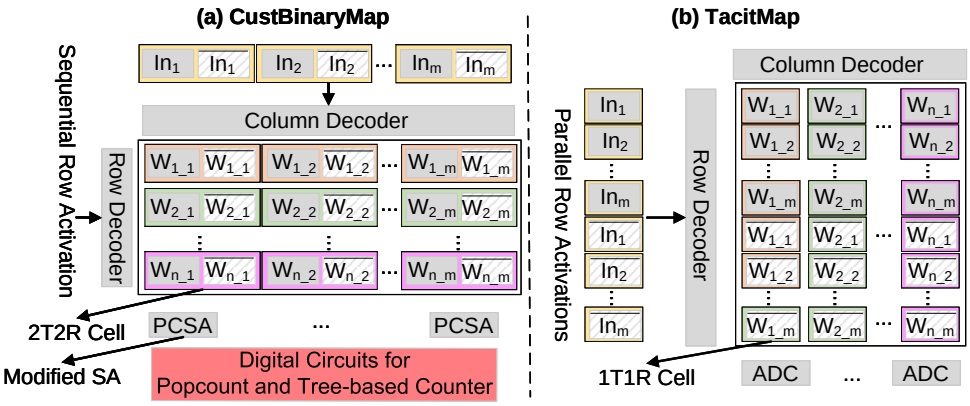

Figure 9.2: TacitMap vs CustBinaryMap data mapping.

We observe that TacitMap enables the crossbar to perform multiple (specifically n in Fig. 9.2) XNOR+Popcount via a single VMM operation in only 1 time step. TacitMap reads the results of these n XNOR+Popcount from ADCs simultaneously. In contrast, with CustBinaryMap, it takes a minimum of n time steps. This happens because when using CustBinaryMap, one first utilizes PCSA to perform the logical XNOR for one input and 1 weight vector of size m. To process n weight vectors, they must do this operation sequentially n times. Moreover, using this mapping, one also needs to perform post-processing on the read output on every final vector using two additional digital components: (1) a fully digital five-bit counter per crossbar column for local Popcount and (2) a tree-based Popcount circuit per several connected crossbars for a global Popcount.

TacitMap relies on VMM and is compatible with any technology for the crossbar that supports it, e.g., ePCM-based crossbars or oPCM-based ones.

9.2. EVALUATIONS

Implementations and Models. We build a cycle-accurate emulator using PyTorch derived from our device-aware extended circuits [113, 279]. We evaluate the effectiveness of TacitMap using ePCM-TacitMap that is TacitMap on electronic PCM-based cores. Our PCM and oPCM configurations are based on our previous results [146, 333]. We compare our designs against that of [444] as the SotA hardware accelerator for BNNs (Baseline-

ePCM). To eliminate the dependency on the type of device in the VMM-enabled crossbar, we also use the same PCM configuration in ePCM-TacitMap for the baseline.

9.2.1. NETWORKS AND DATASETS

We evaluate all designs over 6 BNNs with various sizes from MlBench [76]. Table 9.1 presents the topologies of these BNNs. The first three networks are convolutional networks. But the last ones are multilayer perceptrons (MLPs) with various scales from small to medium to large [76]. We use MNIST [449] and CIFAR-10 [450] for the datasets.

Network Name	Topology	Accuracy
LeNet-5	5x5x6, 2x2 Pooling - 5x5x16, 2x2 Pooling - FC(120) - FC(84) - FC(10)	98%
CNN-1	5x5x5, 2x2 Pooling - FC(720) - FC(70) - FC(10)	97%
CNN-2	7x7x10, 2x2 Pooling - FC(1210) - FC(1210) - FC(10)	98%
MLP-S	FC(784) - FC(500) - FC(250) - FC(10)	97%
MLP-M	FC(784) - FC(1000) - FC(500) - FC(250) - FC(10)	98.2%
MLP-L	FC(784) - FC(1500) - FC(1000) - FC(500) - FC(10)	98.4

Table 9.1: Configurations of evaluated BNNs.

Table 9.1 also presents the achieved accuracy for each BNN. The high accuracy supports the effectiveness of these networks for certain tasks with lower memory and storage overhead and simpler operations discussed in Section 2.2.2. Note that TacitMap does not affect the accuracy of the target BNN and simply accelerates them via handling their required XNOR+Popcount in parallel and more efficiently.

Evaluation Results. Fig. 9.3 presents the latency improvement of proposals normalized to SotA for the same underlying network. The y-axis uses a log scale.

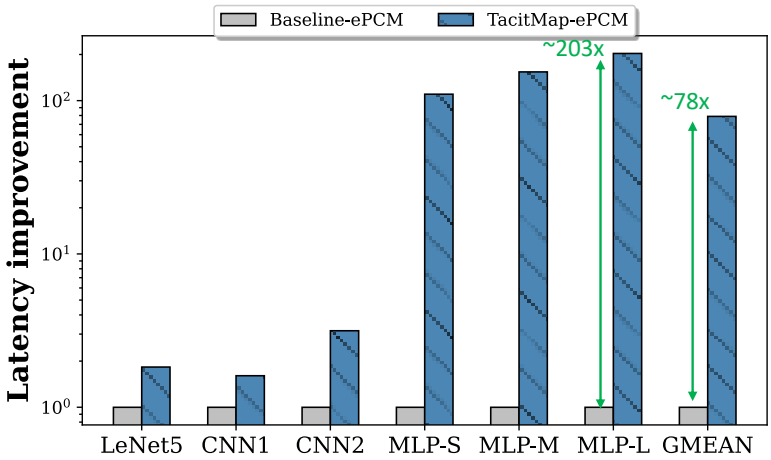

Figure 9.3: Latency improvements over Baseline-ePCM.

We make two key observations.

- TacitMap improves the latency irrespective of BNN. This is because, unlike the Baseline-ePCM, this data mapping not only parallelizes XNOR with Popcount but

also parallelizes both of these operations with many other sets via the proposed vertical data mapping. ePCM-TacitMap improves the performance by ~78× on average.

- The latency improvement is network-dependent. This is directly related to the available parallelism in the operations of understudy BNN. In our BNNs, the larger the BNN is, the more parallel XNOR and Popcount operations exist. Improvements vary from ~1.6× to ~203× for the evaluated BNNs.

9.3. CONCLUSION

This chapter proposes a CIM-based hardware accelerator using memristors and an efficient data flow called TacitMap for BNNs. Our evaluations on latency suggest an enormous potential for such CIM designs for NNs. Hence, our work encourages further investigations of CIM, memristors, and efficient data mapping.

10

High-Performance Hardware Accelerator for BNNs on PCM-based Integrated Photonics

State-of-the-Art (SotA) hardware implementations of Deep Neural Networks (DNNs) for vision and speech recognition on the cloud, mainly based on CMOS and Von-Neumann architectures, incur high latencies and costs. Binary Neural Networks (BNNs) are potential solutions to realize faster and cost-efficient implementation without losing accuracy. To achieve further improvements, this chapter builds on the idea Chapter 9, a new data mapping, called TacitMap, notably suited for BNNs implemented with a Computation-In-Memory (CIM) architecture. TacitMap maximizes the use of available parallelism, while CIM architecture eliminates the data movement overhead. This chapter advances the idea of TacitMap by proposing a hardware accelerator based on optical phase change memory (oPCM), called EinsteinBarrier for BNNs. EinsteinBarrier incorporates TacitMap and adds an extra dimension for parallelism through wavelength division multiplexing, leading to extra latency reduction. The simulation results show that EinsteinBarrier significantly improves execution time with sustainable energy consumption irrespective of the dataset and network. More specifically, EinsteinBarrier provides up to ~3113.2× improvement in execution time compared to SotA CIM baseline while maintaining the energy consumption within 60% of that in the CIM baseline.

This chapter is partially based on the candidate's published and under review works [122, 123].

As discussed Chapter 2.2.2, momentous developments in Deep Neural Network (DNN) in the past decade have led to significant improvements in accuracy and execution time of computer vision tasks such as object detection and recognition [22–24]. However, current DNN hardware implementations are relatively slow and costly to run [214–216]; they suffer from data movement between the processor and memory [218] and make use of expensive hardware such as storing weights in 6 transistors SRAM cells [216, 219]. Hence, developing a high-throughput, cost-effective hardware realization of DNNs while being accurate is critical.

Recently, researchers have proposed the use of simpler (operation-wise) and smaller Neural Networks (NNs) such as Binary Neural Networks (BNNs). BNNs enjoy lower memory requirements (binary values or vectors of {0, 1} or {-1, 1}) [215, 451, 452], simplified arithmetic operations (XNOR instead of multiplication or convolution) [221], and near state-of-the-art (SotA) accuracy on vision tasks [221, 222]. Previous works have investigated two directions to employ BNNs: (1) known platforms such as GPU and central processing unit (CPU) [221, 453, 454], and (2) alternative architectures such as those based on Computation-In-Memory (CIM) paradigm, with the focus on memristor-based CIM designs [279, 442–444]. The solutions in the latter direction overcome the data movement issue that significantly hampers the works in the former direction regarding performance and energy consumption. However, the works in this direction fail to (fully) exploit the inherent features of the underlying hardware. For instance, (a) there is still a lack of efficient data mapping, (b) conventional CIM architectures can typically perform at most one single vector operation (e.g., Vector-Matrix-Multiplication (VMM) or logical vector operation that is the most common operation in NNs) at a time, which limits the throughput, (c) these architectures face many design challenges such as crosstalks and large capacitances of the wiring within the memory IP of CIM, which make the design of such devices complex and limits their scalability.

This chapter advances the SotA CIM accelerators for BNNs by providing a high throughput accelerator based on an oPCM crossbar combined with an efficient mapping method tuned to maximize the parallelism. The proposed accelerator realizes an order of magnitude improvement in latency/throughput without losing the accuracy of the network. The main contribution of this chapter is:

- EinsteinBarrier: An oPCM hardware-based CIM implementation incorporating the TacitMap mapping. EinsteinBarrier ensures maximum parallelism through exploring the potential provided by the features of CIM architecture and the inherent properties of oPCM (via wavelength division multiplexing (WDM).

We extensively evaluate TacitMap and EinsteinBarrier and compare them with SotA implementations for various BNNs. Our results show that when exploiting oPCM and TacitMap, EinsteinBarrier improves the latency by up to ~3113.2×, compared to the same baseline.

10.1. EINSTEINBARRIER ARCHITECTURE

Fig. 10.1-(a) presents an overview of the EinsteinBarrier concept and its system placement. We envision EinsteinBarrier as an accelerator that is part of the memory itself.

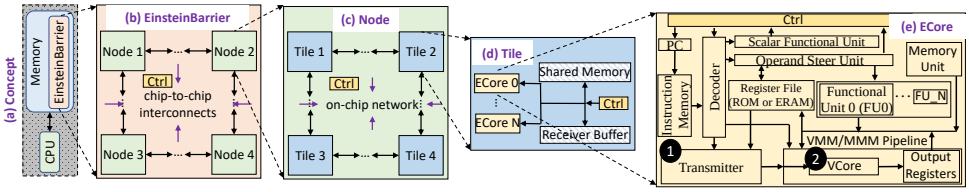

Figure 10.1: EinsteinBarrier system placement and overview.

Fig. 10.1-(b) to -(e) present different levels of hierarchy in EinsteinBarrier. EinsteinBarrier is a spatial architecture with four levels: Nodes, Tiles, External Cores (ECores), and VMM-enabled cores (VCores). This hierarchical organization of EinsteinBarrier is commonly used in other similar works as well [90, 108, 455]. In EinsteinBarrier, Nodes comprise connected Tiles via an on-chip network, Tiles are connected cores sharing a memory, and ECores contain the instruction execution pipeline, VCores, and execution units. EinsteinBarrier follows a three-stage in-order pipeline with fetch, decode, and execute as its stages. To support multiple simultaneous VMMs, which hereafter we call Matrix-Matrix-Multiplication (MMM), EinsteinBarrier extends the ISA discussed in earlier works [108, 456].

EinsteinBarrier offers four main characteristics: (1) Provide a configurable and hierarchical accelerator to support various BNNs, (2) support VMM-enabled crossbars based on emerging technologies as its computation core, (3) allow an extra dimension for parallelism and improving performance via WDM in its oPCM-based cores (i.e., effectively enable MMM), and (4) resolve the challenges ePCM faces for a CIM-based implementation of BNNs via adopting CMOS-compatible oPCM-based cores.

EinsteinBarrier hierarchical organization and the architecture of Tile and ECores provide us with the generality and reconfigurability needed to support various BNNs and multiple technologies (characteristics #1). The ECore architecture and the new μArch required for XNOR+Popcount brings the generality needed to support TacitMap and multiple technologies in VCores as long as they support VMM operation in a crossbar format (characteristics #2). The ECore and VCore designs prepare EinsteinBarrier particularly to adopt oPCM technology and its advantages (characteristics #3). Finally, simply by adopting CMOS-compatible oPCM-based VCores, EinsteinBarrier avoids many of the challenges of ePCM-based CIM architecture (characteristics #4).

10.1.1. oPCM-BASED WDM-ENABLED ECORE

EinsteinBarrier uses integrated photonics with PCM devices in the crossbar. This choice of oPCM-enabled ECores demands two specific extra components, namely VCore and transmitter, compared to other CIM enabled designs. This choice provides an extra dimension for parallelization through WDM and avoids Joule heating and resistance drift in electronic emerging memories [147, 149]. In the following, we first discuss the structure of the first component, a VCore. We then clarify the WDM capability of VCore compared to ePCM-based electronic crossbar using an example. Finally, we discuss the transmitter, the second component.

VCORE STRUCTURE

The oPCM-based VCore is a typical memory with PCM devices in a crossbar format. This means that this core is essentially an array of cells connected together in a conventional crossbar format of rows and columns. Each cell consists of a single PCM device and can store 1 bit of data (binary usage of PCM as discussed in Section 2.1.3). A tile also includes all the necessary peripheries for read and write operations (e.g., DACs and ADCs). In addition, EinsteinBarrier adds 1 more component to the readout circuitry of the oPCM core: transimpedance amplifiers (TIA) on the output (receiver). EinsteinBarrier uses TIA to feed comparators or ADCs acting as a decision and deserialization stage in the output.

WDM

Fig. 10.2 uses an example to present the concept and benefits of WDM on how WDM helps oPCM core to handle multiple VMM operations against how an ePCM-based core handles them. In this example, we assume 3 2-bit activation vectors of X_i distinguished by vectors with yellow, red, and blue colors. The indices demonstrate the bit number, i.e., first bit or second bit. Moreover, we assume 3 2-bit kernel vectors of k_i, j, where i denotes the activation/kernel vector and j denotes the bit position in that activation/kernel vector. Each of these kernel vectors is grouped in a box of orange, green, or pink color. Note that the complements use the same color, but their boxes are dashed.

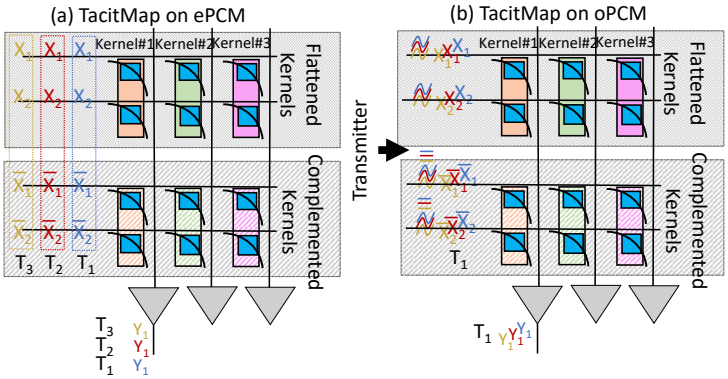

Figure 10.2: WDM in oPCM core.

For the example presented in Fig. 10.2, we want to calculate the XNOR+Popcount of these activations on given kernels. With the mentioned assumptions on the data, TacitMap requires three columns and 4 (2×2) rows of the crossbar to store the kernels and their complement. TacitMap requires three VMM operations[1] to process all the required XNOR+Popcounts; 1 VMM per each activation vector, wherein each VMM we use the vector of that activation concatenated by its complement as the input to the crossbar.

Fig. 10.2-(a) depicts the scenario for a conventional ePCM-based VMM-enabled crossbar. In this case, the required VMMs happen in consecutive time-steps, denoted

[1] For simplicity of our example, we assumed that the columns could be read out in parallel and they do not share an ADC. We will revisit this in Section 10.2.

by T_1, T_2, and T_3 in Fig. 10.2-(a). The input vector size in this scenario is 4, the number of inputs is 3, and the matrix is 4×3.

On the other hand, Fig. 10.2-(b) depicts the same scenario but for an oPCM-based VCore. Using an optical transmitter that we discuss next, one can combine our 3 input vectors together into a single input and feed that single input to the crossbar. Therefore, only 1 time-step, i.e., T_1, is required to finish the operation. Here, the input vector and the matrix size are still 4, and 4×3, respectively. But the number of input vectors is reduced to 1. This method is the WDM capability of oPCM we have discussed. Therefore, effectively, WDM-enabled an MMM of size $4 \times 4 \times 3$. We call the number of wavelengths that can be combined into a single wavelength and still be detectable later (with acceptable noise in TIA) the WDM capacity. Current technologies can comfortably support up to a capacity of $K = 16$ [146].

TRANSMITTER STRUCTURE

To support optical inputs and WDM, EinsteinBarrier adds a transmitter circuit (❶ in Fig. 10.1) at the ECore level feeding the VMM/MMM pipeline, where the actual oPCM-based core (❷) resides as the VCore. Fig. 10.3 presents a high-level overview of the transmitter circuit and components.

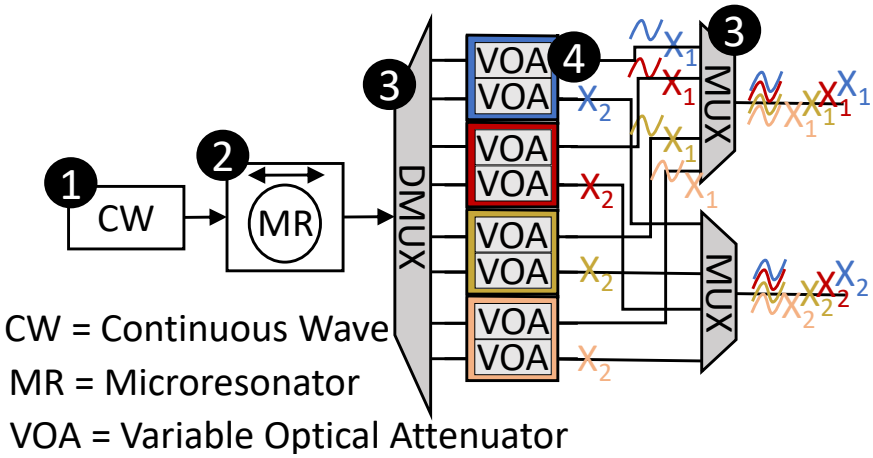

Figure 10.3: Transmitter overview.

Transmitter has four main components required for WDM: ❶ a laser to provide a single-wavelength continuous wave beam, ❷ a microresonator-based optical frequency comb to concentrate the optical power and excite new wavelengths based on non-linearities, ❸ DMUXs and MUXs for feeding individual waves to each variable optical attenuator (VOA) and creating a single wave carrying information on multiple bits from different vectors, and ❹ VOAs to encode the information of each input into waves via changing the amplitude. The colors and indices of X follow the same explanation as in Fig. 10.2. In Fig. 10.3, the transmitter combines four 2-bit vectors (of different colors) into a single vector of 2-bit width.

10.1.2. oPCM-BASED ECORE OVERHEADS

We show that using oPCM provides higher parallelism (simultaneous VMMs vs. single VMM) for the same vector operations via WDM. However, this extra parallelism comes at the cost of power for the additional components. Equation 10.1 presents the additional power needed for enabling WDM. Note that this additional power is for a complete MMM in EinsteinBarrier, and for a fair comparison in Section 10.3.2 we report the end-to-end effect of them on the application power consumption.

$$P_{Added} = P_{transmitter} + P_{crossbar} \tag{10.1}$$

Assume a core with WDM capacity of K and crossbars of size $M \times N$. On the crossbar side, the extra power is simply for processing and can be computed by Equation 10.2, where N is # of TIAs, each of which consuming 2 mW.

$$P_{crossbar} = N \times 2\,\text{mW} \tag{10.2}$$

The transmitter, on the other hand, should account for all the components we discussed in Section 10.1.1. The power overhead of the transmitter is presented in Equation 10.3, where it accounts for the required power for the laser, modulators, and tuning [457].

$$P_{total} = P_{laser} + 3 \times KM\text{mW} + \frac{3 \times KM + 1}{k} \times 45\,\text{mW}. \tag{10.3}$$

Please refer to Section 10.2 for the numbers used in these equations in our evaluations.

10.2. EVALUATION METHODOLOGY

Component	Description
VCore size [146]	64×64
oDAC [458]	168 fJ, 0.0012 mm
Microresonator [458]	0.72 mW
TIA [459]	2 mW
EDFA Pump [460]	$\frac{10\,\text{W}}{\text{amplifiedW}}$

Table 10.1: Evaluated system configurations.

10

10.2.1. IMPLEMENTATIONS AND MODELS

We implement EinsteinBarrier as a heavily extended version of PUMA architecture and compiler [111, 461]. This implementation[2] accounts for (1) WDM capability of oPCM cores, (2) new configurations related to integrated photonics, and (3) power and area overheads introduced by extra components of oPCM cores, e.g., laser. For the photonics components, we use our device-aware extended circuits [146, 457, 462, 463]. Our ePCM-based crossbars are based on extensive characterization done in the EU project MNEMOSENE project and previous works [146, 333], generously provided to us by the

[2] We intend to open-source our experimental setup upon acceptance.

partners. To evaluate additional CMOS circuitry of our design (e.g., such as MUXs), we use Synopsys Design Compiler [331] and synthesize them in the target technology to obtain their execution time, power, and area. We apply the prominent technology scaling rules [332] to the configuration numbers of PUMA architecture to ensure all of our design components are based on the same technology node. Table 10.1 captures some of the system parameters we use in our evaluations.

10.2.2. DESIGNS AND BASELINES

We evaluate the effectiveness of TacitMap and EinsteinBarrier separately using two different configurations: (1) ePCM-TacitMap that is TacitMap on electronic PCM-based cores (this is the same configuration as the one in Chapter 9), and (2) EinsteinBarrier that still uses TacitMap but utilizes oPCM-based VCores.

We use the design in [444], the SotA hardware accelerator for BNNs, as our baseline. To eliminate the dependency on the type of device in the VMM-enabled crossbar, we also use the same PCM configuration in ePCM-TacitMap for the baseline. This configuration is denoted by Baseline-ePCM.

10.2.3. NETWORKS AND DATASETS

We evaluate all designs over 6 BNNs with various sizes from MlBench [76]. Table 9.1 in Chapter 9 presents the topologies of these BNNs. The first three networks are convolutional networks. But the last ones are multilayer perceptrons (MLPs) with various scales from small to medium to large [76]. We use MNIST [449] and CIFAR-10 [450] for the datasets.

Table 9.1 also presents the achieved accuracy for each BNN. The high accuracy supports the effectiveness of these networks for certain tasks with lower memory and storage overhead and simpler operations discussed in Section 2.2.2. Note that neither TacitMap nor EinsteinBarrier affect the accuracy of the target BNN and simply accelerates them via handling their required XNOR+Popcount in parallel and more efficiently.

10.3. EVALUATION RESULTS

10.3.1. PERFORMANCE ANALYSIS

Fig. 10.4 presents the latency improvement of ePCM-TacitMap and EinsteinBarrier normalized to SotA for the same underlying networks. The y-axis uses a log scale.

The following four key observations can be made:

- Both ePCM-TacitMap and EinsteinBarrier improve the latency over Baseline-ePCM irrespective of the underlying network. On average, ePCM-TacitMap and EinsteinBarrier improve the performance by ~78× and ~1205×, respectively (❶). These are because, unlike the Baseline-ePCM, ePCM-TacitMap and EinsteinBarrier parallelize XNOR with Popcount and parallelize many XNOR+Popcounts via the proposed vertical data mapping.

- The latency improvement is network-dependent and varies from BNN to BNN. Specifically, the latency improvements over Baseline-ePCM vary from ~22× to ~3113.2× for

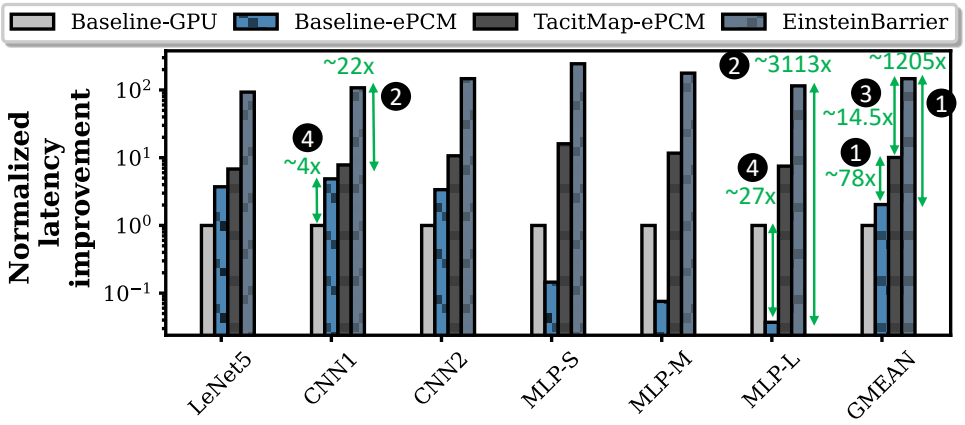

Figure 10.4: Normalized latency improvements over all networks.

EinsteinBarrier (❷). This happens due to (1) the relation between the size of the hidden layers (binary layers) and the first and last layers and (2) available parallelism in the XNOR+Popcount operations of each BNN. In evaluated BNNs, the larger the BNN is, the more parallel XNOR+Popcount operations exist.

- EinsteinBarrier improves the latency on average ~15× (❸) with the exact data flow compared to ePCM-TacitMap. This happens due to the extra parallelism dimension enabled by WDM and the fast crossbar read of oPCM core. This is while the improvement is still network-dependent. Unfortunately, the achieved improvement due to the technology is still lower than the WDM capacity (i.e., $K = 16$). This is simply due to the underlying network, and we expect that it goes higher for bigger networks. We leave exploring this to future work.

- Baseline-ePCM does not always improve the latency over a Baseline-GPU. For example, see ❹ in Fig. 10.4, while Baseline-ePCM is ~4× faster than Baseline-GPU for our first CNN, it is ~27× slower than Baseline-GPU for our MLP-L network. This happens since in some networks, such as our MLP workloads, Baseline-ePCM has to serialize XNOR+Popcount compared to Baseline-GPU, so much so that the benefits for reducing the data movement overhead diminish.

10

10.3.2. ENERGY ANALYSIS

Fig. 10.5 compares the energy consumption of Baseline-ePCM, ePCM-TacitMap, and EinsteinBarrier. All numbers are normalized to the Baseline-ePCM energy consumption of the same underlying network. The y-axis is in a log scale.

One can make the following two key observations:

- On average, ePCM-TacitMap increases the energy consumption compared to Baseline-ePCM by ~5.35×. This is because ePCM-TacitMap requires power-hungry ADCs while baseline is using SAs (❶).

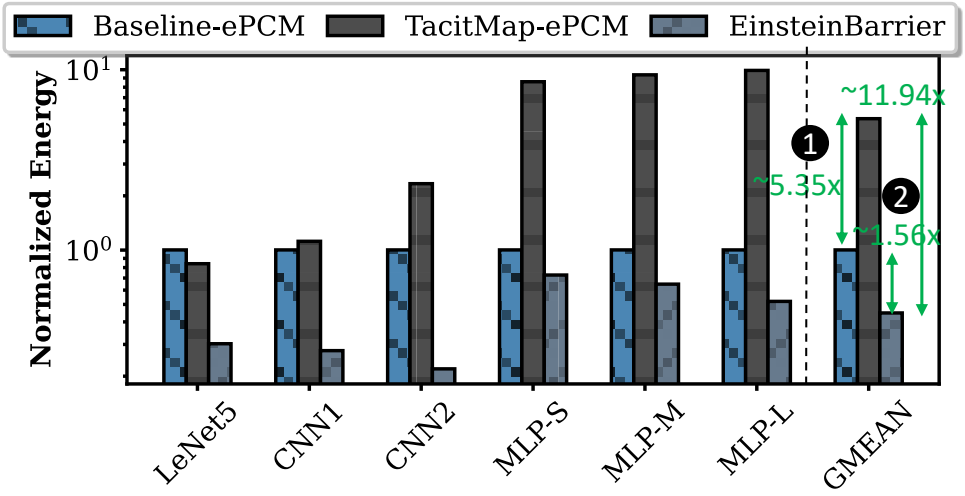

Figure 10.5: Normalized energy consumption over all networks.

- On average, EinsteinBarrier improves the energy consumption by 56% and ~11.94× over Baseline-ePCM and ePCM-TacitMap, respectively (❷). The improvement is achieved because EinsteinBarrier requires a lower number of crossbar activations by using the same crossbar, ADCs, and other peripheries but computing multiple outputs at the same time.

10.4. DISCUSSIONS AND FUTURE WORKS

Multi-Level PCM Devices. In particular, this work uses PCMs in a binary mode, i.e., 2 states per device. However, recent works [146, 464] show the potential for multi-bit devices at the cost of increased noise. We leave the exploration of extending TacitMap on multi-bit cells for future work.

Design Space Exploration of oPCM-based VCores. We currently evaluated EinsteinBarrier using fixed laser, array sizes, and other system configurations. This choice was made due to our limited access to specs of different components (particularly those needed in the transmitter), i.e., we do have only the power of the same components for a fixed build size. A study that can freely explore this design space is encouraged but left for future work.

10.5. CONCLUSION

This chapter proposes an CMOS-compatible oPCM-based hardware accelerator based on integrated photonics principles, called EinsteinBarrier, to exploit the possible parallelism with TacitMap (an efficient data flow for BNNs presented in Chapter 9) fully. Our latency and energy evaluations suggest an enormous potential for oPCM-based accelerators to improve performance in BNNs while improving their energy consumption. This is the first step towards an optimized and efficient hardware realization for BNNs us-

ing these emerging technologies. Hence, our work encourages further investigations of oPCM in the NN realm.

11

Conclusion

This dissertation demonstrates that CIM enhances the efficiency of various Genomics pipelines and Machine Learning (ML) applications. Fundamentally, the goal of this dissertation is twofold: (1) pinpoint and boost genomics and ML kernels with CIM, and (2) investigate emerging (memory) technologies suited for CIM. We combine analytical methods with hands-on experiments, crafting innovative CIM architectures and hardware/software co-designed strategies. These advancements enable faster and more energy-conservative processing in numerous genomics pipelines and vision-centered ML techniques. This chapter begins by highlighting the dissertation's accomplishments and the core findings of each chapter. It concludes by discussing potential avenues and research directions for subsequent future research.

11.1. SUMMARY

Chapter 1, Introduction. This chapter underscores the significance of emerging systems and architectures tailored for modern applications and their main features. First, we describe two exemplars of modern applications, namely genomics and Machine Learning. We spotlight their distinctive attributes, such as the extensive data working sets, the swift expansion of these already vast data sets, and their continuous urge for prompt analysis, distinguishing them from traditional applications. Subsequently, we delve into traditional computing systems rooted in Von-Neumann architecture and CMOS technology, emphasizing their inherent constraints that fuel the quest for novel computing concepts and techniques. We then unfold (1) the Computation-In-Memory computing model and (2) emerging memory technologies, explaining their potential to counteract current impediments in our traditional computing systems. Further, we address the obstacles and challenges researchers must navigate to fine-tune CIM designs utilizing these emerging technologies, ensuring they are apt for modern application demands. Subsequently, we outline the research directions of this thesis and articulate our core thesis statement. Finally, we recapitulate this thesis's contributions, outlining their relevance to the designated research domains.

Chapter 2, Background and State-of-the-Art. This chapter provides the necessary background and fundamentals of the Computation-In-Memory paradigm, emerging (memory) technologies, modern applications, and state-of-the-art designs using systems built on CIM or emerging memory technologies. We use the details in this chapter to set the stage for the next ones. First, we introduce CIM. We cover the terminologies used in CIM, a classification for CIM designs, and what a CIM tile is. Next, we touch upon memristor devices, often just called memristors. We highlight three main types: ReRAM, PCM, and STT-MRAM, comparing them with traditional memory technologies and noting their imperfect aspects (termed non-idealities). We then look at common tile structures used with memristors. Moving on, we discuss optical phase change memory, emphasizing its advantages over other memristor-based CIM methods. We list some basic operations that memristor-based CIM can handle. Then, we outline the design choices for CIM, discussing homogenous and heterogeneous designs. Afterward, we dive into two critical modern applications: genomics and Machine Learning. We explain why their needs align with what CIM and new (memory) technologies offer. We give examples of popular genomics pipelines and dive into a prominent category of ML tasks. Finally, we look at top-notch state-of-the-art CIM designs and (simulation) tools, starting with general-purpose ones and then narrowing down to more specialized accelerators.

Chapter 3, Swordfish: In-Memory Basecalling. This chapter focuses on basecalling, an early step in the genomics pipelines, and explores how to make it faster using memristor-based CIM. We introduce Swordfish, a new framework that helps us study how accuracy can drop because of device non-idealities and limitations of the memristor-based CIM design for basecalling. With Swordfish, we can analyze these challenges and test potential solutions to deal with them. Swordfish takes into account seven real-world issues found in memristor-based tiles. Using different hardware/software co-designed strategies from various techniques, Swordfish aims to maintain high accuracy. We tested Swordfish using Bonito, a state-of-the-art (i.e., accurate and fast), freely available basecaller. Our results show that our CIM design speeds up Bonito by around 25.7× and only

has a small drop of 6.01% in accuracy.

Chapter 4, RattlesnakeJake: In-Memory Algorithm for Short-Read Pre-Alignment Filtering. This chapter investigates a major (new) bottleneck in many genome analysis pipelines: the pre-alignment filtering in short-read sequence alignment. Current methods move short-read sequences to processing units only to sometimes reject them, wasting energy and time. To address this, we introduce RattlesnakeJake, a special algorithm and its corresponding CIM hardware that makes pre-alignment filtering for short reads faster and uses less energy. RattlesnakeJake offers a streamlined filtering method and uses memristors and the CIM paradigm to carry out needed operations while saving on data movement. RattlesnakeJake achieves an accuracy level on par with state-of-the-art (SotA) level and significantly speeds up short-read sequence alignment. Depending on the dataset, we have seen improvements of up to ~7× and ~80× compared to the best alternatives on GPU and CPU, respectively.

Chapter 5, SieveMem: In-Memory Hardware Acceleration for Short-Read Pre-Alignment Filtering. This chapter continues from the last, addressing the challenge of data movement delays in pre-alignment filtering tools for short reads. We noticed that current SotA accelerators for pre-alignment filtering are not well-suited for upcoming filtering algorithms using the same operations while suffering from their data movement overhead. To bridge this gap, we present SieveMem, an architecture designed around the Computation-In-Memory approach and memristor devices. SieveMem operates directly in memory, reducing unnecessary data movement. We craft SieveMem to be adaptable for future short-read algorithms by supporting the shared kernels seen in previous SotA pre-alignment filters. Our experiments show that SieveMem efficiently supports over 47.6% of shared tasks across the top five filters. We also refine RattlesnakeJake discussed in Chapter 4 and introduce a filtering algorithm, BandedKrait, that blends well with SieveMem. This combination of BandedKrait on SieveMem, which we term Mem-BandedKrait, showcases a significant speed boost, reducing processing times by up to 331.1× for two common operations. Lastly, when using BandedKrait on SieveMem, the overall sequence alignment time for short reads improves, even surpassing the best GPU accelerator by up to 91.4×.

Chapter 6, FilterFuse: In-Memory Hardware/Software Co-Designed for Long-Read Pre-Alignment Filtering. This chapter builds on our previous observations in Chapter 4 and chap5, delving deeper into the challenge of efficiently analyzing long reads instead, the industry's new preference for a more precise and efficient DNA reconstruction. Long-read alignment consumes much time, causing a lag, i.e., bottleneck, in genomics research reliant on this alignment. While pre-alignment filters boost short-read alignments, their effectiveness dwindles with long reads. Moreover, even with these pre-alignment filters, the full alignment process (both filtering and actual alignment) for long reads takes long, with filtering accounting for a big chunk of this time. The vast amounts of long-read data movement between storage and processors are a main contributor to this overhead. Although filters discard many of these reads before alignment, they come at a steep price in terms of time and energy. Addressing this, we introduce LongGeneGuardian, an adapted pre-alignment filter explicitly designed for long reads. To fully harness its potential, we present FilterFuse, an architecture running LongGeneGuardian directly in memory. Using the Computation-In-Memory approach,

FilterFuse sidesteps data movement costs. Our tests reveal that FilterFuse trims the filtering time by 120.47× when compared against the leading filter, SneakySnake. Further, it reduces the total end-to-end alignment time for long reads by up to 49.14× compared to pairing SneakySnake with a top-tier aligner and by 5207.63× against just the premier aligner.

Chapter 7, Demeter: In-Memory Algorithm and Accelerator for Food Profiling. This chapter explores ways to monitor food for safety and authenticity better. With advances in sequencing technologies, acquiring the food sequences becomes cheaper, while food profiling (the computational step) is becoming more time-consuming and the primary computational bottleneck. The current state-of-the-art (SotA) food profiling tools are expensive and unsuitable for quick monitoring. We aim to develop a more efficient profiler to handle large data structures and minimize data transfers for real-time systems. Therefore, we introduce Demeter, our new framework designed for food profiling. Using hyperdimensional computing (HDC), Demeter can efficiently categorize and identify a few species in food, addressing the large data structure challenges of previous food profilers. For the data movement problem, we introduce Acc-Demeter, an in-memory hardware system made with memristor devices. This setup makes Demeter faster and uses less energy. When we put Demeter to the test against other SotA food profilers, Demeter maintains a high accuracy level, staying within 2% of SotA food profilers. We synthesize Acc-Demeter's required hardware using UMC's 65nm library by considering an accurate PCM model based on silicon-based prototypes. Our hardware evaluations show that Acc-Demeter significantly outperforms two leading profilers regarding speed and memory use. More specifically, Acc-Demeter achieves a (1) throughput improvement of 192× and 724× and (2) memory reduction of 36× and 33× compared to Kraken2 and MetaCache (2 state-of-the-art profilers), respectively, on typical food-related databases. Plus, Acc-Demeter does not take up much extra space.

Chapter 8, KrakenOnMem: In-Memory Taxonomic Profiler. This chapter focuses on creating a fast and energy-efficient hardware accelerator for taxonomic profiling, a crucial first step in advanced metagenomic studies. While modern taxonomic profilers are accurate, they are slow and consume a lot of energy, with the *Table Lookup* operation being a primary bottleneck. To address this, we introduce TL-PIM, a hardware accelerator that uses CIM to speed up *Table Lookup*. By combining the capabilities of emerging memory technologies with intelligent data placement, TL-PIM enhances the efficiency of *Table Lookup*. We then combine TL-PIM with a SotA profiler, Kraken2, creating a more efficient hardware/software co-designed system named KrakenOnMem. Initial tests on a small-scale silicon-based memory prototype are promising. Pir large=scale calibrated simulations show that KrakenOnMem is much faster than Kraken2, providing an average of 61.3% speedup compared to original Kraken2 for end-to-end profiling. KrakenOnMem is also by orders of magnitude more energy-efficient than Kraken2, with only a tiny increase in the area overhead of the entire system for the accelerator.

Chapter 9, LightSpeed: In-Memory Data Mapping for BNNs. This chapter shifts the attention from the genomics field to the ML application domain, explicitly emphasizing the acceleration of Binary Neural Networks (BNNs). Recent studies highlight BNNs as efficient solutions for task vision, notably for enhancing power and storage efficiency without compromising accuracy. We identify significant data movement overheads by ana-

lyzing BNNs' performance on conventional systems with GPUs. This opens an opportunity for a CIM design to address this overhead. However, earlier CIM approaches that aim to reduce these overheads do not fully leverage the potential of the native hardware features. Addressing this, we introduce an innovative data mapping, called TacitMap, tailored for BNNs on CIM platforms. TacitMap is a highly parallel data mapping for BNN operations on any CIM design capable of performing VMM, e.g., memristor-based crossbars such as electronic phase change memory-based (ePCM) or resistive random-access memory-based (ReRAM) ones. TacitMap is designed with the conventional 1T1R memory crossbar structure in mind and is therefore compatible with many of the already evolving crossbar architectures. We benchmark TacitMap against SotA workloads for assorted BNNs through rigorous evaluations. The results of this chapter underscore that our approach trims the latency by a remarkable ~154× when compared to the prevailing SotA data mappings for BNNs on CIM frameworks.

Chapter 10, EinsteinBarrier: oPCM-based In-Memory Acceleration of BNNs. This chapter furthers our exploration of improving BNNs using CIM (discussed in Chapter 9) into hardware acceleration. We introduce a new accelerator, EinsteinBarrier, which leverages the unique features of optical phase change memory (oPCM). By integrating TacitMap from Chapter 9 and capitalizing on the efficiency of oPCM, especially with the help of wavelength division multiplexing (WDM), EinsteinBarrier achieves optimal parallel processing. Our results reveal that EinsteinBarrier significantly cuts down latency (up to ~3113.2×) when paired with TacitMap and remains energy-efficient (within 60%), staying close to the consumption levels of the SotA CIM baseline.

11.2. FUTURE RESEARCH DIRECTIONS

Although this dissertation focuses on enabling modern applications to overcome the performance and energy barriers of traditional systems using the CIM paradigm and emerging memory technologies, we believe that our works and the insights they produce are applicable in a more general sense and open up new research directions. This section reviews promising directions for future work.

11.2.1. EXTENDING THE PROPOSED TECHNIQUES

Throughout this dissertation, we uncover techniques grounded in universal principles. These methods fit the genomics realm and align with other (modern) applications. Think about domains grappling with ever-growing data sets and crave swift and energy-efficient processing; our strategies can also serve there. Now, consider emerging memory technologies. Be it NAND Flash memory [465, 466], ferroelectric field-effect transistor (FeFET), phase-change memory [278, 295, 467], magnetoresistive memory [468, 469], or racetrack memory (RM) [470], our approaches seem promising. Therefore, we propose to cast a wide net, explore and adapt our insights to new arenas, and identify and tackle the bottlenecks they present.

TO OTHER (MODERN) APPLICATIONS

While centered on CIM for specific aspects of genomics and select ML kernels, our designs are built on foundational principles. Therefore, these principles are not limited to

the realms we explored. At their core, our proposals and designs address challenges related to processing large datasets quickly and efficiently. This characteristic is not unique to only genomics or ML; many modern and even traditional applications share this trait. Take, for example, database searches. These searches, especially those that involve extensive table lookups, can be time-consuming and computationally intensive. Our approach, exemplified by tools like TL-PIM from our KrakenOnMem project (see Chapter 8), could potentially be adapted to streamline database operations, making them faster and more energy-efficient. Then, there is the potential of oPCM (see Chapter 10). Given its proven ability to accelerate multiple VMM operations with the help of WDM, we can envision its applicability in larger computational settings. Neural networks, especially Deep Neural Networks, might benefit substantially from this technology. The underlying operations and data flow in large-scale neural networks are similar to the challenges we addressed in genomics. In conclusion, while our work offers solutions for specific challenges, the underlying strategies have broader implications. We are excited about the potential of adapting and expanding these methods to other domains. We believe that future researchers and developers can draw inspiration from our work, tweak and refine the concepts, and devise innovative solutions for a plethora of applications beyond what we have touched upon.

TO OTHER MEMORY TECHNOLOGIES

Our research provides valuable insights into the application of emerging memory technologies for enhancing the performance and energy efficiency of contemporary applications dealing with large data working sets. Although we delved into the specifics of particular technologies, like PCM in Demeter (referenced in Chapter 7) and ReRAM in SieveMem (highlighted in Chapter 5), the principles and methodologies we have discussed have a broader reach. For instance, while we tailor our analysis to certain memory technologies, the underlying principles can potentially be expanded to encompass others like STT-MRAM or RM. Both of these, similar in nature to the ones we have examined, support akin logical operations, which suggests that they could benefit from strategies akin to what we have proposed, especially within a CIM design context. Indeed, many of these innovative memory technologies are still in their developmental phases, not yet achieving full commercialization. But envisioning and mapping out efficient architectures and designs for them is most important. As these technologies evolve, the research landscape will inevitably shift toward resolving their challenges. In such a scenario, the foundations we have laid in our work can act as a guiding light, assisting researchers in navigating the intricate pathways of CIM designs utilizing these upcoming memory technologies.

11.2.2. LEVERAGING AND CASCADING THE NEW-FOUND CIM DESIGNS IN END-TO-END PIPELINES

Our research lays the groundwork for more efficient systems in genomics and ML by leveraging CIM and emerging memory technologies. Therefore, future research can explore some potential avenues. In this section, we review several possibilities stemming from our work. However, we believe that the true potential of CIM, combined with emerging memory technologies, is vast, and as the field evolves, we will see unforeseen

applications and innovations that will undoubtedly surface.

CASCADED CIM ARCHITECTURES IN A GENOMICS PIPELINE

In this thesis, our primary focus has been on devising accelerators tailored for specific stages of the genome analysis pipeline, as depicted in Fig. 2.13. These accelerators, while impactful in isolation, might offer compounded benefits when integrated into a cohesive system, acting in concert with each other across multiple steps of the pipeline. An intriguing avenue for future research would be a holistic evaluation of the entire pipeline, gauging the collective impact when all these accelerators are operational together. Such an analysis would not merely be a sum of individual performances; the interplay between various accelerators, data flow intricacies, and synergistic effects could bring about unforeseen enhancements or challenges. For instance, the data exchange and communication between accelerators might be a significant aspect to address. How do these accelerators communicate? What is the best strategy to ensure efficient data transfer between them without bottlenecks? Furthermore, understanding the adoption costs of integrating multiple accelerators is pivotal. This includes the overhead of establishing the connectivity, ensuring compatibility, and potentially reconfiguring existing systems to accommodate these new additions. In essence, while our contributions provide valuable building blocks for improving individual pipeline stages, the next frontier would be to architect a seamless, integrated system where these accelerators cohesively drive the entire genomics pipeline to new heights of efficiency and performance.

COMBINED CIM ARCHITECTURES IN AN UNIFIED ACCELERATOR

Here, we have delved into various stages of the genomics pipeline, as illustrated in Fig. 2.13. For each segment, we have designed distinct CIM accelerators to optimize performance. Yet, a promising avenue for future studies is the integration of these individual techniques into a singular, unified accelerator. By fusing these separate techniques, we aim to harness the full power of in-memory computation and leverage the capabilities of emerging memory technologies. The key challenge, and potential advantage, lies in achieving this amalgamation without incurring significant area overhead. This would not only optimize performance but also drive down associated costs. In essence, while our current approach offers modular solutions tailored to each pipeline step, the future beckons a more holistic approach, crafting a streamlined and cost-effective accelerator that encapsulates the benefits of all individual techniques.

SYSTEM INTEGRATION OF ACCELERATORS WITH OUR APPLICATION-SPECIFIC ECOSYSTEM

Our thesis delves deep into designing accelerators using CIM and novel memory technologies, targeting enhancements in modern genomics and ML applications. Yet, it is crucial to note that these accelerators do not operate in a vacuum. Given that not all steps are covered by these solutions, they must mesh seamlessly with many other computational units and memory types. Consider sequencing machines, for instance. These systems inherently possess heterogeneity, encompassing a diverse mix of memory technologies, such as DRAM and NAND flash memories and computational cores, including CPU, GPU, and FPGA. This diversity exists because each component brings unique ad-

vantages, justifying the integration costs. Hence, a ripe avenue for subsequent studies is to delve into the fluid integration of our accelerators within this intricate landscape. Particularly, as mentioned in Section 11.2.2, understanding how a consolidated accelerator fits in becomes essential. Indeed, our current models adopt varying approaches, oscillating between standalone accelerators or those that collaborate with a host, which could range from a CPU to more specialized hardware like a GPU or FPGA. However, the assumptions underlying these models might necessitate a re-evaluation. This would be determined by the resources available in any heterogeneous environment our accelerator(s) aim to integrate into. In essence, tailoring our solutions to the specificities of their operational context remains a compelling challenge and opportunity for future explorations.

11.2.3. New Bottlenecks after Exploiting the Proposed CIM Designs

Throughout this thesis, we unveil individual accelerators that dramatically speed up specific kernels within the genomics pipeline. It is essential to underscore that while the macroscopic speedup for some tasks, like the taxonomic profiling discussed in Chapter 8, might appear subtle, the speed boosts for micro-tasks, like *Table Lookup*, is monumental. We recognize a pattern when we step back and envision the bigger picture. As each bottleneck is alleviated by one of our proposals, another one naturally surfaces. This domino effect is intrinsic to computer engineering, where enhancing one component may shift the performance bottleneck to another. What does this imply for future research? As we deploy our accelerators in full-fledged genome analysis systems, the interplay of bottlenecks will evolve. The challenge then morphs from addressing individual bottlenecks to understanding and rectifying the emergent bottlenecks that arise from these interplays. This iterative approach of identifying, resolving, and then moving to the next bottleneck forms the crux of our field, making future endeavors in this direction both exciting and imperative.

Other Works of the Author

In my time at TU Delft, I led nine successful projects. These projects shape this dissertation. The earlier chapters detail these works. Chapter 11 captures our key findings and suggests future, both immediate and distant, research paths.

During my graduate studies, I also worked on many research projects. I collaborated with peers from the Quantum and Computer Engineering (QCE) department at TU Delft. I also teamed up with the SAFARI group at ETH Zürich and the Chair for Compiler Construction (CCC) at TU Dresden. These projects taught me two things. First, they sharpened my research thinking and critical mindset. Second, they expanded my knowledge of computer architecture, especially about memory systems, in-memory processing, and bioinformatics. I will discuss these projects in the next parts of this chapter.

During my first Ph.D. year, I teamed up with the SAFARI group for various projects. My collaboration with Abdullah Giray Yağlıkçı led to BlockHammer [471]. BlockHammer is a fresh approach to defense against RowHammer, a vulnerability in today DRAM. BlockHammer outperforms previous defenses regarding protection, scalability, and compatibility with standard DRAM chips. I explored system-level methods to reduce energy use in VR and video streaming with Javad Haj-Yahya. We investigate several system-level techniques, such as bypassing the host DRAM and extending the display panel with a double remote frame buffer (DRFB) instead of DRAM's double frame buffer. As a result, we came up with BurstLink [472] that requires only minor tweaks to the display processes of today's mobile systems. With Rahul Bera, we crafted a hardware prefetching system. We envision the prefetching as a task for a reinforcement learning agent. Our outcome, Pythia [473], learns using various program features and system-level feedback. We designed Pythia to be tweaked in silicon without hardware changes, making it adaptable to diverse program needs.

In my second year, I delved deep into exploring the CIM paradigm, intertwining both traditional and emergent memory technologies with genomics. Initially, I collaborated with João Dinis Ferreira from Avaloq, leading to the conception of pLUTo [115]. Uniquely positioned, pLUTo leverages conventional memory technology (DRAM) to initiate lookup table operations, opting for bulk memory reads, LUT queries, and complex additional logic. Subsequently, my journey with Mahdi Zahedi from QCE involved a meticulous review of the SotA memristor-centric CIM designs that facilitate many logical and arithmetic functions. Our discourse unveils the intricate data flow within a generic CIM-tile and the innate capacity to devise complex functions. Furthermore, we lay out a spectrum of applications primed for in-memory execution and contemplate their generalization facets. We then outline future pathways for CIM-based computational systems, asserting that our design would illuminate challenges and uncharted research vectors around a generalized CIM tile. Shifting the focus to STT-MRAMs, I partnered with IMEC in Belgium and Abhairaj Singh at IBM Zürich. Our collective efforts birthed an adaptive referencing modality tailored for CIM architectures, emphasizing boolean binary logi-

cal functions. Our method enhances both the sensing margin and the performance of the operation [279]. Tailored for STT-MRAM devices constructed using a 28nm technology node, we seamlessly integrate our model into two pivotal domains: BNNs and text encryption and show the potential benefits of our design. Lastly, Minesh Patel from SAFARI at ETH Zürich and I draw from the rich tapestry of insights on DRAM devices and their vulnerabilities that we learned during M.Sc. and Ph.D. journeys. We jointly advocated for enhanced clarity into foundational DRAM reliability traits. This, we believed, would arm system designers with the tools to refine and adapt standard DRAM chips, catering to system-specific needs [474]. Our advocacy gains momentum through three case studies, spotlighting DRAM refresh overhead reduction, latency enhancement, and RowHammer defense design. Our discourse emphasizes a prevailing opacity that deters system designers from embracing optimization. We cap our discussion with insightful recommendations, championing transparency in prevailing and prospective DRAM-centric systems.

In my last two years, things got busier and more insightful. First, I teamed up with Can Firtina from SAFARI on two main projects. In our first project, we dug into ways to find exact and close matches (short strings) useful for read mapping and read overlapping. Our solution was BLEND [475]. In BLEND, we simplified the task to just one lookup for finding fuzzy matches (exact or very close matches). Using SimHash, BLEND could give us similar hash values for similar sets without many clashes. In the second project, we studied pHMMs and the Baum-Welch method. Our proposal, ApHMM [476], was a HW/SW co-designed framework. ApHMM reduced the computational and power use of the Baum-Welch method. We designed flexible hardware that can handle different pHMM types, remember data patterns on-chip, and skip over repeated tasks using a bloom filter. Next, Mahdi Zahedi from QCE and I embarked on three distinct projects. Our first project investigated how memory can handle both signed and unsigned arithmetic tasks. Taking cues from memristor crossbars, we suggested a two's complement approach for various arithmetic tasks [301]. We also crafted a streamlined digital layout to support MMM in an energy and area-efficient manner. Our second project dove into speeding up graph tasks using CIM, especially targeting sparse graphs. We proposed SparseMEM [114]. SparseMEM works with a novel data representation for compressed graph data and is supported by any memory technology. We also created an optimized ReRAM-based version of SparseMem, built on our data layout, to handle popular graph tasks. In our third shirt project, Mahdi Zahedi and I analyzed the energy and speed inefficiencies in running BNNs on memristor-based CIM designs. Our solution to mitigate these inefficiencies, BCIM [477], mimics ADC and the required following digital processing by a SA while it allows simultaneous row activation to maximize resource utilization on the crossbar and enhance the performance. We investigated the effect of the number of references for each SA. We also explored the distance between the values of the references. Moreover, we examined how to use weights and activation data of BNNs better to communicate less between layers, leading to more energy and performance efficiency. Lastly, extending my work on pre-alignment filtering, I collaborated with Asif Ali Khan from TU Dresden. We developed a pre-alignment filter with racetrack memory (RM) - another emerging memory tech. Our creation [478] designed a new data layout to make the most out of RM and overcome its sequential access challenge that previous

in-memory filters based on RM face.

EPILOGUE

In this dissertation, we dive deep into how certain tasks from genomics and ML can run on a CIM setup using new memory tech. We build a detailed understanding of our target applications' performance and energy bottlenecks. We find that while CIM systems and these new memory technologies promise better speeds, they come with challenges. These challenges, like non-ideal behaviors of devices and the need for specific logic close to or within memory, can make it hard for system designers to adopt them directly. This is especially true if we want to set up the system with the right data flows, actions, and connections to embrace CIM fully. Our work offers solutions to these challenges, focusing on genomics tasks and BNNs. We hope our findings and tools pave the way for more research. This research would ideally merge the advantages of CIM and new memory tech, like memristors, with the reliable benefits of traditional systems and CMOS technology, which we believe remain vital for innovation and other tasks.

BIBLIOGRAPHY

[1] Geoffrey Ginsburg and Kathryn Phillips. Precision Medicine: From Science To Value. *Health Affairs*, 37:694–701, 05 2018.

[2] Zahra Aryan, Attila Szanto, Angeliki Pantazi, Tejaswini Reddi, Carolyn Rheinstein, Winslow Powers, Evan Wilson, Rahul Deo, Shimul Chowdhury, Lisa Salz, David Dimmock, Shareef Nahas, Wendy Benson, Stephen Kingsmore, Calum MacRae, and Dana Vuzman. Moving Genomics to Routine Care: An Initial Pilot in Acute Cardiovascular Disease. *Circulation: Genomic and Precision Medicine*, 13, 08 2020.

[3] Michelle M Clark, Amber Hildreth, Sergey Batalov, Yan Ding, Shimul Chowdhury, Kelly Watkins, Katarzyna Ellsworth, Brandon Camp, Cyrielle I Kint, Calum Yacoubian, et al. Diagnosis of genetic diseases in seriously ill children by rapid whole-genome sequencing and automated phenotyping and interpretation. *Science translational medicine*, 11(489):eaat6177, 2019.

[4] Stephen Kingsmore, Laurie Smith, Chris Kunard, Matthew Bainbridge, Serge Batalov, Wendy Benson, Eric Blincow, Sara Caylor, Christina Chambers, Guillermo Angel, David Dimmock, Yan Ding, Katarzyna Ellsworth, Annette Feigenbaum, Erwin Frise, Robert Green, Lucia Guidugli, Kevin Hall, Christian Hansen, and Thomas Defay. A genome sequencing system for universal newborn screening, diagnosis, and precision medicine for severe genetic diseases. *The American Journal of Human Genetics*, 109, 08 2022.

[5] Geoffrey Ginsburg and Huntington Willard. Genomic and Personalized Medicine: Foundations and Applications. *Translational research : the journal of laboratory and clinical medicine*, 154:277–87, 12 2009.

[6] Joshua Bloom, Laila Sathe, Chetan Munugala, Eric Jones, Molly Gasperini, Nathan Lubock, Fauna Yarza, Erin Thompson, Kyle Kovary, Jimin Park, Dawn Marquette, Stephania Kay, Mark Lucas, TreQuan Love, A. Booeshaghi, Oliver Brandenberg, Longhua Guo, James Boocock, Myles Hochman, and Valerie Arboleda. Massively scaled-up testing for SARS-CoV-2 RNA via next-generation sequencing of pooled and barcoded nasal and saliva samples. *Nature Biomedical Engineering*, 5:1–9, 07 2021.

[7] Joshua Quick, Nicholas Loman, Sophie Duraffour, Jared Simpson, Ettore Severi, Lauren Cowley, Joseph Bore, Raymond Koundouno, Gytis Dudas, Amy Mikhail, Nobila Ouedraogo, Babak Afrough, Amadou Bah, Jonathan Baum, Beate Becker-Ziaja, Jan Boettcher, Mar Cabeza-Cabrerizo, Álvaro Camino-Sánchez, Lisa Carter, and Miles Carroll. Real-time, portable genome sequencing for Ebola surveillance. *Nature*, 530, 02 2016.

[8] Ramesh Yelagandula, Aleksandr Bykov, Alexander Vogt, Robert Heinen, Ezgi Özkan, Marcus Martin Strobl, Juliane Christina Baar, Kristina Uzunova, Bence Hajdusits, Darja Kordic, Erna Suljić, Amina Kurtovic-Kozaric, Sebija Izetbegovíc, Justine Schaeffer, Peter Hufnagl, Alexander Zoufaly, Tamara Seitz, Manuela Födinger, Franz Allerberger, Alexander Stark, Luisa Cochella, and Ulrich Elling. Multiplexed detection of SARS-CoV-2 and other respiratory infections in high throughput by SARSeq. *Nature Communications*, 12, 2021.

[9] Vien Le and Binh Diep. Selected Insights from Application of Whole Genome Sequencing for Outbreak Investigations. *Current opinion in critical care*, 19, 07 2013.

[10] Vladyslav Nikolayevskyy, Katharina Kranzer, Stefan Niemann, and Francis Drobniewski. Whole Genome Sequencing of M.tuberculosis for detection of recent transmission and tracing outbreaks: a systematic review. *Tuberculosis*, 98, 03 2016.

[11] John Wooley, Adam Godzik, and Iddo Friedberg. A Primer on Metagenomics. *PLoS computational biology*, 6:e1000667, 02 2010.

[12] Can Alkan, Jeffrey Kidd, Tomas Marques-Bonet, Gozde Aksay, Francesca Antonacci, Fereydoun Hormozdiari, Jacob Kitzman, Carl Baker, Maika Malig, Onur Mutlu, Cenk Sahinalp, Richard Gibbs, and Evan Eichler. Personalized Copy-Number and Segmental Duplication Maps using Next-Generation Sequencing. *Nature genetics*, 41:1061–7, 08 2009.

[13] Euan A Ashley. Towards precision medicine. *Nature Reviews Genetics*, 17(9):507–522, 2016.

[14] Lynda Chin, Jannik Andersen, and P Futreal. Cancer genomics: From discovery science to personalized medicine. *Nature medicine*, 17:297–303, 03 2011.

[15] Taha Shahroodi, Stephan Wong, and Said Hamdioui. A Case for Genome Analysis Where Genomes Reside. In *Embedded Computer Systems: Architectures, Modeling, and Simulation*, pages 453–458. Springer Nature Switzerland, 2023.

[16] Hans Ellegren. Genome sequencing and population genomics in non-model organisms. *Trends in Ecology & Evolution*, 29(1):51–63, 2014.

[17] María Alvarez-Cubero, Maria Saiz, Belén Martínez-García, Sara Sayalero, Carmen Entrala, Jose LORENTE, and Luis Martinez-Gonzalez. Next generation sequencing: an application in forensic sciences? *Annals of Human Biology*, 44:1–12, 09 2017.

[18] Daniel W. Otter, Julian R. Medina, and Jugal K. Kalita. A Survey of the Usages of Deep Learning for Natural Language Processing. *IEEE Transactions on Neural Networks and Learning Systems*, 32(2):604–624, 2021.

[19] Anamika Dhillon and Gyanendra Verma. Convolutional neural network: a review of models, methodologies and applications to object detection. *Progress in Artificial Intelligence*, 9, 12 2019.

[20] Wei Wang and Yujing Yang. Development of convolutional neural network and its application in image classification: A survey. *Optical Engineering*, 58:1, 04 2019.

[21] Dengsheng Lu. A Survey of Image Classification Methods and Techniques for Improving Classification Performance. *International Journal of Remote Sensing*, 28:823 – 870, 03 2007.

[22] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. *Commun. ACM*, 60(6):84–90, may 2017.

[23] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 11 2013.

[24] Ross Girshick. Fast R-CNN. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1440–1448, 2015.

[25] G. V. RESEARCH. Metagenomics market size, share and trends analysis report by product (sequencing and data analytics), by technology (sequencing, function), by application (environmental), and segment forecasts, 2018 - 2025., 2017.

[26] Wetterstrand KA. DNA Sequencing Costs: Data from the NHGRI Genome Sequencing Program (GSP). https://www.genome.gov/sequencingcostsdata.

[27] Barba, M, Czosnek, H and Hadidi, A. Cost in US Dollars per Raw Megabase of DNA Sequence. https://www.genome.gov/about-genomics/fact-sheets/DNA-Sequencing-Costs-Data.

[28] Pablo Villalobos and Anson Ho. Trends in training dataset sizes, 2022. Accessed: 2023-6-7.

[29] David A Patterson and John L Hennessy. *Computer organization and design ARM edition: the hardware software interface*. Morgan kaufmann, 2016.

[30] J. von Neumann. First draft of a report on the EDVAC. *IEEE Annals of the History of Computing*, 15(4):27–75, 1993.

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

[32] Nor Zaidi Haron and Said Hamdioui. Why is CMOS scaling coming to an END? In *2008 3rd International Design and Test Workshop*, pages 98–103, 2008.

[33] Gordon E. Moore. Cramming more components onto integrated circuits, Reprinted from Electronics, volume 38, number 8, April 19, 1965, pp.114 ff. *IEEE Solid-State Circuits Society Newsletter*, 11(3):33–35, 2006.

[34] R.H. Dennard, F.H. Gaensslen, Hwa-Nien Yu, V.L. Rideout, E. Bassous, and A.R. LeBlanc. Design of ion-implanted MOSFET's with very small physical dimensions. *IEEE Journal of Solid-State Circuits*, 9(5):256–268, 1974.

[35] Sriseshan Srikanth, Lavanya Subramanian, Sreenivas Subramoney, Thomas M. Conte, and Hong Wang. Tackling Memory Access Latency through DRAM Row Management. In *Proceedings of the International Symposium on Memory Systems*, MEMSYS '18, page 137–147, New York, NY, USA, 2018. Association for Computing Machinery.

[36] Howard David, Chris Fallin, Eugene Gorbatov, Ulf R. Hanebutte, and Onur Mutlu. Memory Power Management via Dynamic Voltage/Frequency Scaling. In *Proceedings of the 8th ACM International Conference on Autonomic Computing*, ICAC '11, page 31–40, New York, NY, USA, 2011. Association for Computing Machinery.

[37] Jack Doweck, Wen-Fu Kao, Allen Kuan-yu Lu, Julius Mandelblat, Anirudha Rahatekar, Lihu Rappoport, Efraim Rotem, Ahmad Yasin, and Adi Yoaz. Inside 6th-Generation Intel Core: New Microarchitecture Code-Named Skylake. *IEEE Micro*, 37(2):52–62, 2017.

[38] Hadi Esmaeilzadeh, Emily Blem, Renee St. Amant, Karthikeyan Sankaralingam, and Doug Burger. Dark Silicon and the End of Multicore Scaling. In *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ISCA '11, page 365–376, New York, NY, USA, 2011. Association for Computing Machinery.

[39] David Yeager. Dynimize: The Big Picture. *Dynimize*, 2020.

[40] Gokcen Kestor, Roberto Gioiosa, Darren J. Kerbyson, and Adolfy Hoisie. Quantifying the energy cost of data movement in scientific applications. In *2013 IEEE International Symposium on Workload Characterization (IISWC)*, pages 56–65, 2013.

[41] Dhinakaran Pandiyan and Carole-Jean Wu. Quantifying the energy cost of data movement for emerging smart phone workloads on mobile platforms. In *2014 IEEE International Symposium on Workload Characterization (IISWC)*, pages 171–180, 2014.

[42] Amirali Boroumand, Saugata Ghose, Youngsok Kim, Rachata Ausavarungnirun, Eric Shiu, Rahul Thakur, Daehyun Kim, Aki Kuusela, Allan Knies, Parthasarathy Ranganathan, and Onur Mutlu. Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks. *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, 53(2):316–331, mar 2018.

[43] W.H. Kautz. Cellular Logic-in-Memory Arrays. *IEEE Transactions on Computers*, C-18(8):719–727, 1969.

[44] Harold S. Stone. A Logic-in-Memory Computer. *IEEE Transactions on Computers*, C-19(1):73–78, 1970.

[45] David Elliot Shaw, Salvatore J. Stolfo, Hussein Ibrahim, Bruce Hillyer, Gio Wiederhold, and JA Andrews. The NON-VON database machine: A brief overview. *IEEE Database Eng. Bull.*, 4(2):41–52, 1981.

[46] D.G. Elliott, W.M. Snelgrove, and M. Stumm. Computational Ram: A Memory-simd Hybrid And Its Application To Dsp. In *1992 Proceedings of the IEEE Custom Integrated Circuits Conference*, pages 30.6.1–30.6.4, 1992.

[47] Peter M. Kogge. EXECUBE-A New Architecture for Scaleable MPPs. In *1994 International Conference on Parallel Processing Vol. 1*, volume 1, pages 77–84, 1994.

[48] M. Gokhale, B. Holmes, and K. Iobst. Processing in memory: the Terasys massively parallel PIM array. *Computer*, 28(4):23–31, 1995.

[49] D. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Keeton, C. Kozyrakis, R. Thomas, and K. Yelick. A case for intelligent RAM. *IEEE Micro*, 17(2):34–44, 1997.

[50] M. Oskin, F.T. Chong, and T. Sherwood. Active Pages: a computation model for intelligent memory. In *Proceedings. 25th Annual International Symposium on Computer Architecture (Cat. No.98CB36235)*, pages 192–203, 1998.

[51] Yi Kang, Wei Huang, Seung-Moon Yoo, D. Keen, Zhenzhou Ge, V. Lam, P. Pattnaik, and J. Torrellas. FlexRAM: toward an advanced intelligent memory system. In *Proceedings 1999 IEEE International Conference on Computer Design: VLSI in Computers and Processors (Cat. No.99CB37040)*, pages 192–201, 1999.

[52] Basilio B. Fraguela, Jose Renau, Paul Feautrier, David Padua, and Josep Torrellas. Programming the FlexRAM Parallel Intelligent Memory System. In *Proceedings of the Ninth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '03, page 49–60, New York, NY, USA, 2003. Association for Computing Machinery.

[53] Jeff Draper, Jacqueline Chame, Mary Hall, Craig Steele, Tim Barrett, Jeff LaCoss, John Granacki, Jaewook Shin, Chun Chen, Chang Woo Kang, Ihn Kim, and Gokhan Daglikoca. The Architecture of the DIVA Processing-in-Memory Chip. In *Proceedings of the 16th International Conference on Supercomputing*, ICS '02, page 14–25, New York, NY, USA, 2002. Association for Computing Machinery.

[54] K. Mai, T. Paaske, N. Jayasena, R. Ho, W.J. Dally, and M. Horowitz. Smart Memories: a modular reconfigurable architecture. In *Proceedings of 27th International Symposium on Computer Architecture (IEEE Cat. No.RS00201)*, pages 161–171, 2000.

[55] Kimberly Keeton, David A. Patterson, and Joseph M. Hellerstein. A Case for Intelligent Disks (IDISKs). *SIGMOD Rec.*, 27(3):42–52, sep 1998.

[56] Anurag Acharya, Mustafa Uysal, and Joel Saltz. Active Disks: Programming Model, Algorithms and Evaluation. In *Proceedings of the Eighth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS VIII, page 81–91, New York, NY, USA, 1998. Association for Computing Machinery.

[57] Liu and Jino. Intelligent Magnetic Bubble Memories and Their Applications in Data Base Management Systems. *IEEE Transactions on Computers*, C-28(12):888–906, 1979.

[58] Doty, Greenblatt, and Stanley Y.W. Su. Magnetic Bubble Memory Architectures for Supporting Associative Searching of Relational Databases. *IEEE Transactions on Computers*, C-29(11):957–970, 1980.

[59] Bongiovanni and Luccio. Maintaining Sorted Files in a Magnetic Bubble Memory. *IEEE Transactions on Computers*, C-29(10):855–863, 1980.

[60] Yong-Bin Kim and T. Chen. Assessing merged DRAM/logic technology. In *1996 IEEE International Symposium on Circuits and Systems (ISCAS)*, volume 4, pages 133–136 vol.4, 1996.

[61] Joseph Gebis. Trends in merged DRAM-logic computing. In Sunny Bains and Leo J. Irakliotis, editors, *Critical Technologies for the Future of Computing*, volume 4109, pages 198 – 205. International Society for Optics and Photonics, SPIE, 2000.

[62] Damla Senol Cali, Gurpreet S. Kalsi, Zülal Bingöl, Can Firtina, Lavanya Subramanian, Jeremie S. Kim, Rachata Ausavarungnirun, Mohammed Alser, Juan Gomez-Luna, Amirali Boroumand, Anant Norion, Allison Scibisz, Sreenivas Subramoneyon, Can Alkan, Saugata Ghose, and Onur Mutlu. GenASM: A High-Performance, Low-Power Approximate String Matching Acceleration Framework for Genome Sequence Analysis. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 951–966, 2020.

[63] Junwhan Ahn, Sungpack Hong, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. A scalable processing-in-memory accelerator for parallel graph processing. In *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, pages 105–117, 2015.

[64] Qiuling Zhu, Tobias Graf, H. Ekin Sumbul, Larry Pileggi, and Franz Franchetti. Accelerating sparse matrix-matrix multiplication with 3D-stacked logic-in-memory hardware. In *2013 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–6, 2013.

[65] Seth H Pugsley, Jeffrey Jestes, Huihui Zhang, Rajeev Balasubramonian, Vijayalakshmi Srinivasan, Alper Buyuktosunoglu, Al Davis, and Feifei Li. NDC: Analyzing the impact of 3D-stacked memory+logic devices on MapReduce workloads. In *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 190–200, 2014.

[66] Dongping Zhang, Nuwan Jayasena, Alexander Lyashevsky, Joseph Greathouse, Lifan Xu, and Michael Ignatowski. TOP-PIM: Throughput-oriented programmable processing in memory. In *HPDC 2014 - Proceedings of the 23rd International Symposium on High-Performance Parallel and Distributed Computing*, 06 2014.

[67] Amin Farmahini-Farahani, Jung Ho Ahn, Katherine Morrow, and Nam Kim. NDA: Near-DRAM acceleration architecture leveraging commodity DRAM devices and standard memory modules. *2015 IEEE 21st International Symposium on High Performance Computer Architecture, HPCA 2015*, pages 283–295, 03 2015.

[68] Gabriel H Loh, Nuwan Jayasena, Mark Oskin, Mark Nutter, David Roberts, Mitesh Meswani, Dong Ping Zhang, and Mike Ignatowski. A processing in memory taxonomy and a case for studying fixed-function pim. In *Workshop on Near-Data Processing (WoNDP)*, pages 1–4, 2013.

[69] Kevin Hsieh, Eiman Ebrahimi, Gwangsun Kim, Niladrish Chatterjee, Mike O'Connor, Nandita Vijaykumar, Onur Mutlu, and Stephen Keckler. Transparent Offloading and Mapping (TOM): Enabling Programmer-Transparent Near-Data Processing in GPU Systems. *ACM SIGARCH Computer Architecture News*, 44:204–216, 06 2016.

[70] Ashutosh Pattnaik, Xulong Tang, Adwait Jog, Onur Kayiran, Asit K. Mishra, Mahmut T. Kandemir, Onur Mutlu, and Chita R. Das. Scheduling Techniques for GPU Architectures with Processing-In-Memory Capabilities. In *Proceedings of the 2016 International Conference on Parallel Architectures and Compilation*, PACT '16, page 31–44, New York, NY, USA, 2016. Association for Computing Machinery.

[71] Berkin Akin, Franz Franchetti, and James C. Hoe. Data Reorganization in Memory Using 3D-Stacked DRAM. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, ISCA '15, page 131–143, New York, NY, USA, 2015. Association for Computing Machinery.

[72] Kevin Hsieh, Samira Khan, Nandita Vijaykumar, Kevin Chang, Amirali Boroumand, Saugata Ghose, and Onur Mutlu. Accelerating pointer chasing in 3D-stacked memory: Challenges, mechanisms, evaluation. In *2016 IEEE 34th International Conference on Computer Design (ICCD)*, pages 25–32, 10 2016.

[73] Aurelia Augusta and Stratos Idreos. JAFAR: Near-Data Processing for Databases. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, page 2069–2070, New York, NY, USA, 2015. Association for Computing Machinery.

[74] Joo Lee, Jaewoong Sim, and Hyesoon Kim. BSSync: Processing Near Memory for Machine Learning Workloads with Bounded Staleness Consistency Models. In *2015 International Conference on Parallel Architecture and Compilation (PACT)*, pages 241–252, 10 2015.

[75] Mingyu Gao and Christoforos E. Kozyrakis. HRL: Efficient and flexible reconfigurable logic for near-data processing. *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 126–137, 2016.

[76] Ping Chi, Shuangchen Li, Cong Xu, Tao Zhang, Jishen Zhao, Yongpan Liu, Yu Wang, and Yuan Xie. PRIME: A Novel Processing-in-Memory Architecture for Neural Network Computation in ReRAM-Based Main Memory. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pages 27–39, 2016.

[77] Boncheol Gu, Andre S. Yoon, Duck-Ho Bae, Insoon Jo, Jinyoung Lee, Jonghyun Yoon, Jeong-Uk Kang, Moonsang Kwon, Chanho Yoon, Sangyeun Cho, Jaeheon Jeong, and Duckhyun Chang. Biscuit: A Framework for near-Data Processing of Big Data Workloads. In *Proceedings of the 43rd International Symposium on Computer Architecture*, ISCA '16, page 153–165. IEEE Press, 2016.

[78] Duckhwan Kim, Jaeha Kung, Sek Chai, Sudhakar Yalamanchili, and Saibal Mukhopadhyay. Neurocube: A Programmable Digital Neuromorphic Architecture with High-Density 3D Memory. In *Proceedings of the 43rd International Symposium on Computer Architecture*, ISCA '16, page 380–392. IEEE Press, 2016.

[79] Hadi Asghari-Moghaddam, Young Hoon Son, Jung Ho Ahn, and Nam Sung Kim. Chameleon: Versatile and Practical near-DRAM Acceleration Architecture for Large Memory Systems. In *The 49th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-49. IEEE Press, 2016.

[80] Vivek Seshadri, Thomas Mullins, Amirali Boroumand, Onur Mutlu, Phillip B. Gibbons, Michael A. Kozuch, and Todd C. Mowry. Gather-Scatter DRAM: In-DRAM Address Translation to Improve the Spatial Locality of Non-Unit Strided Accesses. In *Proceedings of the 48th International Symposium on Microarchitecture*, MICRO-48, page 267–280, New York, NY, USA, 2015. Association for Computing Machinery.

[81] Zhiyu Liu, Irina Calciu, Maurice Herlihy, and Onur Mutlu. Concurrent Data Structures for Near-Memory Computing. In *Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '17, page 235–245, New York, NY, USA, 2017. Association for Computing Machinery.

[82] Mingyu Gao, Grant Ayers, and Christos Kozyrakis. Practical near-data processing for in-memory analytics frameworks. In *2015 International Conference on Parallel Architecture and Compilation (PACT)*, pages 113–124, 10 2015.

[83] Qi Guo, Nikolaos Alachiotis, Berkin Akin, Fazle Sadi, Guanglin Xu, Tze Low, Larry Pileggi, James Hoe, and Franz Franchetti. 3D-Stacked Memory-Side Acceleration: Accelerator and System Design. In *WoNDP*, 12 2014.

[84] Zehra Sura, Arpith Jacob, Tong Chen, Bryan Rosenburg, Olivier Sallenave, Carlo Bertolli, Samuel Antao, Jose Brunheroto, Yoonho Park, Kevin O'Brien, and Ravi Nair. Data Access Optimization in a Processing-in-Memory System. In *Proceedings of the 12th ACM International Conference on Computing Frontiers*, CF '15, New York, NY, USA, 2015. Association for Computing Machinery.

[85] Amir Morad, Leonid Yavits, and Ran Ginosar. GP-SIMD Processing-in-Memory. *ACM Trans. Archit. Code Optim.*, 11(4), jan 2015.

[86] Syed Minhaj Hassan, Sudhakar Yalamanchili, and Saibal Mukhopadhyay. Near Data Processing: Impact and Optimization of 3D Memory System Architecture on the Uncore. In *Proceedings of the 2015 International Symposium on Memory Systems*, MEMSYS '15, page 11–21, New York, NY, USA, 2015. Association for Computing Machinery.

[87] Shuangchen Li, Cong Xu, Qiaosha Zou, Jishen Zhao, Yu Lu, and Yuan Xie. Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories. In *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, 2016.

[88] Mingu Kang, Min-Sun Keel, Naresh Shanbhag, Sean Eilert, and Ken Curewitz. An energy-efficient VLSI architecture for pattern recognition via deep embedding of computation in SRAM. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8326–8330, 05 2014.

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

[90] Ali Shafiee, Anirban Nag, Naveen Muralimanohar, Rajeev Balasubramonian, John Paul Strachan, Miao Hu, R. Stanley Williams, and Vivek Srikumar. ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pages 14–26, 2016.

[91] Vivek Seshadri, Yoongu Kim, Chris Fallin, Donghyuk Lee, Rachata Ausavarungnirun, Gennady Pekhimenko, Yixin Luo, Onur Mutlu, Phillip B. Gibbons, Michael A. Kozuch, and Todd C. Mowry. RowClone: Fast and Energy-Efficient in-DRAM Bulk Data Copy and Initialization. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-46, page 185–197, New York, NY, USA, 2013. Association for Computing Machinery.

[92] Kevin Chang, Prashant Nair, Donghyuk Lee, Saugata Ghose, Moinuddin Qureshi, and Onur Mutlu. Low-Cost Inter-Linked Subarrays (LISA): Enabling fast inter-subarray data movement in DRAM. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 568–580, 03 2016.

[93] Vivek Seshadri, Donghyuk Lee, Thomas Mullins, Hasan Hassan, Amirali Boroumand, Jeremie Kim, Michael A. Kozuch, Onur Mutlu, Phillip B. Gibbons, and Todd 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*, MICRO-50 '17, page 273–287, New York, NY, USA, 2017. Association for Computing Machinery.

[94] L. Nai, R. Hadidi, J. Sim, H. Kim, P. Kumar, and H. Kim. GraphPIM: Enabling Instruction-Level PIM Offloading in Graph Computing Frameworks. In *2017 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 457–468, Los Alamitos, CA, USA, feb 2017. IEEE Computer Society.

[95] Jeremie Kim, Damla Senol Cali, Hongyi Xin, Donghyuk Lee, Saugata Ghose, Mohammed Alser, Hasan Hassan, Oguz Ergin, Can Alkan, and Onur Mutlu. GRIM-Filter: Fast Seed Location Filtering in DNA Read Mapping Using Processing-in-Memory Technologies. *BMC Genomics*, 19, 05 2018.

[96] Shuangchen Li, Dimin Niu, Krishna T. Malladi, Hongzhong Zheng, Bob Brennan, and Yuan Xie. DRISA: A DRAM-Based Reconfigurable In-Situ Accelerator. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-50 '17, page 288–301, New York, NY, USA, 2017. Association for Computing Machinery.

[97] Gwangsun Kim, Niladrish Chatterjee, Mike O'Connor, and Kevin Hsieh. Toward standardized near-data processing with unrestricted data placement for gpus. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '17, New York, NY, USA, 2017. Association for Computing Machinery.

[98] Maciej Besta, Raghavendra Kanakagiri, Grzegorz Kwasniewski, Rachata Ausavarungnirun, Jakub Beránek, Konstantinos Kanellopoulos, Kacper Janda, Zur Vonarburg-Shmaria, Lukas Gianinazzi, Ioana Stefan, Juan Gómez-Luna, Jakub Golinowski, Marcin Copik, Lukas Kapp-Schwoerer, Salvatore Di Girolamo, Nils Blach, Marek Konieczny, Onur Mutlu, and Torsten Hoefler. SISA: Set-Centric Instruction Set Architecture for Graph Mining on Processing-in-Memory Systems. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 282–297, 10 2021.

[99] Shuangchen Li, Alvin Oliver Glova, Xing Hu, Peng Gu, Dimin Niu, Krishna T. Malladi, Hongzhong Zheng, Bob Brennan, and Yuan Xie. SCOPE: A Stochastic Computing Engine for DRAM-Based in-Situ Accelerator. In *Proceedings of the 51st Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-51, page 696–709. IEEE Press, 2018.

[100] Sayeef Salahuddin, Kai Ni, and Suman Datta. The era of hyper-scaling in electronics. *Nature Electronics*, 1(8):442–450, 2018.

[101] Fabian Oboril, Rajendra Bishnoi, Mojtaba Ebrahimi, and Mehdi B. Tahoori. Evaluation of Hybrid Memory Technologies Using SOT-MRAM for On-Chip Cache Hierarchy. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(3):367–380, 2015.

[102] Shimeng Yu and Pai-Yu Chen. Emerging Memory Technologies: Recent Trends and Prospects. *IEEE Solid-State Circuits Magazine*, 8(2):43–56, 2016.

[103] S. Ambrogio, S. Balatti, A. Cubeta, A. Calderoni, N. Ramaswamy, and D. Ielmini. Understanding switching variability and random telegraph noise in resistive RAM. In *2013 IEEE International Electron Devices Meeting*, pages 31.5.1–31.5.4, 2013.

[104] Alexander Hardtdegen, Camilla La Torre, Felix Cüppers, Stephan Menzel, Rainer Waser, and Susanne Hoffmann-Eifert. Improved switching stability and the effect of an internal series resistor in hfo2/tiox bilayer reram cells. *IEEE Transactions on Electron Devices*, 65(8):3229–3236, 2018.

[105] Tifenn Hirtzlin, Marc Bocquet, Bogdan Penkovsky, Jacques-Olivier Klein, Etienne Nowak, Elisa Vianello, Jean-Michel Portal, and Damien Querlioz. Digital Biologically Plausible Implementation of Binarized Neural Networks With Differential Hafnium Oxide Resistive Memory Arrays. *Frontiers in Neuroscience*, 13, 2020.

[106] Lei Xie, H.A. Du Nguyen, Jintao Yu, Ali Kaichouhi, Mottaqiallah Taouil, Mohammad AlFailakawi, and Said Hamdioui. Scouting Logic: A Novel Memristor-Based Logic Design for Resistive Computing. In *2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 176–181, 2017.

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

[108] Aayush Ankit, Izzat El Hajj, Sai Rahul Chalamalasetti, Geoffrey Ndu, Martin Foltin, R. Stanley Williams, Paolo Faraboschi, Wen-mei W Hwu, John Paul Strachan, Kaushik Roy, and Dejan S. Milojicic. PUMA: A Programmable Ultra-Efficient Memristor-Based Accelerator for Machine Learning Inference. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '19, page 715–731, New York, NY, USA, 2019. Association for Computing Machinery.

[109] Gokul Krishnan, Sumit K. Mandal, Chaitali Chakrabarti, Jae-Sun Seo, Umit Y. Ogras, and Yu Cao. Impact of On-Chip Interconnect on In-Memory Acceleration of Deep Neural Networks. *J. Emerg. Technol. Comput. Syst.*, 18(2), dec 2022.

[110] Lixue Xia, Boxun Li, Tianqi Tang, Peng Gu, Xiling Yin, Wenqin Huangfu, Pai-Yu Chen, Shimeng Yu, Yu Cao, Yu Wang, Yuan Xie, and Huazhong Yang. MNSIM: Simulation platform for memristor-based neuromorphic computing system. In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 469–474, 2016.

[111] Ankit, Aayush. PUMA Compiler. https://github.com/Aayush-Ankit/puma-compiler, 2019.

[112] Andi Drebes, Lorenzo Chelini, Oleksandr Zinenko, Albert Cohen, Henk Corporaal, Tobias Grosser, Kanishkan Vadivel, and Nicolas Vasilache. Tc-cim: Empowering tensor comprehensions for computing-in-memory. In *IMPACT 2020 workshop (associated with HIPEAC 2020)*, 2020. Informal proceedings.

[113] Mahdi Zahedi, Muah Abu Lebdeh, Christopher Bengel, Dirk Wouters, Stephan Menzel, Manuel Le Gallo, Abu Sebastian, Stephan Wong, and Said Hamdioui. MNEMOSENE: Tile Architecture and Simulator for Memristor-Based Computation-in-Memory. *J. Emerg. Technol. Comput. Syst.*, 18(3), jan 2022.

[114] Mahdi Zahedi, Geert Custers, Taha Shahroodi, Georgi Gaydadjiev, Stephan Wong, and Said Hamdioui. Sparsemem: Energy-efficient design for in-memory sparse-based graph processing. In *Design, Automation & Test in Europe Conference & Exhibition, DATE 2023, Antwerp, Belgium, April 17-19, 2023*, pages 1–6. IEEE, 2023.

[115] João Dinis Ferreira, Gabriel Falcao, Juan Gómez-Luna, Mohammed Alser, Lois Orosa, Mohammad Sadrosadati, Jeremie S. Kim, Geraldo F. Oliveira, Taha Shahroodi, Anant Nori, and Onur Mutlu. pLUTo: Enabling Massively Parallel Computation in DRAM via Lookup Tables. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 900–919, 2022.

[116] Taha Shahroodi, Gagandeep Singh, Mahdi Zahedi, Haiyu Mao, Joel Lindegger, Can Firtina, Stephan Wong, Onur Mutlu, and Said Hamdioui. Swordfish: A Framework for Evaluating Deep Neural Network-based Basecalling using Computation-In-Memory with Non-Ideal Memristors. In *56rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2023.

[117] Taha Shahroodi, Michael Miao, Mahdi Zahedi, Stephan Wong, and Said Hamdioui. RattlesnakeJake: A Fast and Accurate Pre-alignment Filter Suitable for Computation-in-Memory. In *Embedded Computer Systems: Architectures, Modeling, and Simulation*, pages 209–221. Springer Nature Switzerland, 2023.

[118] Taha Shahroodi, Michael Miao, Mahdi Zahedi, Stephan Wong, and Said Hamdioui. SieveMem: A Computation-in-Memory Architecture for Fast and Accurate Pre-Alignment. In *2023 IEEE 34th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 156–164. IEEE, 2023.

[119] Taha Shahroodi, Michael Miao, Stephan Wong, and Said Hamdioui. FilterFuse: A Computation-In-Memory Architecture for High-Performance Long-Read Pre-Alignment Filtering. In *Review Process of 2024 ACM/IEEE 51th Annual International Symposium on Computer Architecture (ISCA)*, 2024.

[120] Taha Shahroodi, Mahdi Zahedi, Can Firtina, Mohammed Alser, Stephan Wong, Onur Mutlu, and Said Hamdioui. Demeter: A Fast and Energy-Efficient Food Profiler Using Hyperdimensional Computing in Memory. *IEEE Access*, 10:82493–82510, 2022.

[121] Taha Shahroodi, Mahdi Zahedi, Abhairaj Singh, Stephan Wong, and Said Hamdioui. KrakenOnMem: A Memristor-Augmented HW/SW Framework for Taxonomic Profiling. In *Proceedings of the 36th ACM International Conference on Supercomputing*, ICS '22, New York, NY, USA, 2022. Association for Computing Machinery.

[122] Taha Shahroodi, Raphael Cardoso, Mahdi Zahedi, Stephan Wong, Alberto Bosio, Ian O'Connor, and Said Hamdioui. Lightspeed Binary Neural Networks using Optical Phase-Change Materials. In *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–2. IEEE, 2023.

[123] Taha Shahroodi, Raphael Cardoso, Stephan Wong, Alberto Bosio, Ian O'Connor, and Said Hamdioui. High-Performance Data Mapping for BNNs on PCM-based Integrated Photonics. In *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2024.

[124] Dhinakaran Pandiyan and Carole-Jean Wu. Quantifying the energy cost of data movement for emerging smart phone workloads on mobile platforms. In *2014 IEEE International Symposium on Workload Characterization (IISWC)*, pages 171–180, 2014.

[125] Gokcen Kestor, Roberto Gioiosa, Darren J. Kerbyson, and Adolfy Hoisie. Quantifying the energy cost of data movement in scientific applications. In *2013 IEEE International Symposium on Workload Characterization (IISWC)*, pages 56–65, 2013.

[126] Hoang Anh Du Nguyen, Jintao Yu, Muath Abu Lebdeh, Mottaqiallah Taouil, Said Hamdioui, and Francky Catthoor. A Classification of Memory-Centric Computing. *J. Emerg. Technol. Comput. Syst.*, 16(2), jan 2020.

[127] Mahdi Zahedi, Remon van Duijnen, Stephan Wong, and Said Hamdioui. Tile Architecture and Hardware Implementation for Computation-in-Memory. In *2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 108–113, 2021.

[128] Mahdi Zahedi, Taha Shahroodi, Geert Custers, Abhairaj Singh, Stephan Wong, and Said Hamdioui. System Design for Computation-in-Memory: From Primitive to Complex Functions. *2022 IFIP/IEEE 30th International Conference on Very Large Scale Integration (VLSI-SoC)*, pages 1–6, 2022.

[129] James M Tour and Tao He. The fourth element. *Nature*, 453(7191):42–43, 2008.

[130] Jamie Beckett. Demystifying the memristor: Proof of fourth basic circuit element could transform computing. *HPL. HP. COM*, 2008.

[131] Dirk Wouters. Resistive switching materials and devices for future memory applications. *Tutorial on 43 rd IEEE Semiconductor Interface Specialists Conference (SISC), San Diego*, 5, 2012.

[132] Heba Abunahla, Baker Mohammad, Maguy Abi jaoudé, and Mahmoud Al-Qutayri. Novel hafnium oxide memristor device: Switching behaviour and size effect. In *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4, 05 2017.

[133] Abu Sebastian, Manuel Gallo, Riduan Khaddam-Aljameh, and Evangelos Eleftheriou. Memory devices and applications for in-memory computing. *Nature Nanotechnology*, 15, 03 2020.

[134] Lizhou Wu, Siddharth Rao, Mottaqiallah Taouil, Erik Jan Marinissen, Gouri Sankar Kar, and Said Hamdioui. Characterization and Fault Modeling of Intermediate State Defects in STT-MRAM. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1717–1722, 2021.

[135] Sayeef Salahuddin, Kai Ni, and Suman Datta. The era of hyper-scaling in electronics. *Nature Electronics*, 1, 08 2018.

[136] Fabian Oboril, Rajendra Bishnoi, Mojtaba Ebrahimi, and Mehdi B. Tahoori. Evaluation of Hybrid Memory Technologies Using SOT-MRAM for On-Chip Cache Hierarchy. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(3):367–380, 2015.

[137] Pai-Yu Chen and Shimeng Yu. Technological Benchmark of Analog Synaptic Devices for Neuroinspired Architectures. *IEEE Design & Test*, 36(3):31–38, 2019.

[138] Xiaoxuan Yang, Brady Taylor, Ailong Wu, Yiran Chen, and Leon O. Chua. Research Progress on Memristor: From Synapses to Computing Systems. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 69(5):1845–1857, 2022.

[139] Shubham Jain, Abhronil Sengupta, Kaushik Roy, and Anand Raghunathan. RxNN: A Framework for Evaluating Deep Neural Networks on Resistive Crossbars. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 40(2):326–338, 2021.

[140] YeonJoo Jeong, Mohammed A. Zidan, and Wei D. Lu. Parasitic Effect Analysis in Memristor-Array-Based Neuromorphic Systems. *IEEE Transactions on Nanotechnology*, 17(1):184–193, 2018.

[141] Yaojun Zhang, Xiaobin Wang, Hai Li, and Yiran Chen. STT-RAM Cell Optimization Considering MTJ and CMOS Variations. *IEEE Transactions on Magnetics*, 47(10):2962–2965, 2011.

[142] Nicola Peserico, Bhavin J. Shastri, and Volker J. Sorger. Integrated Photonic Tensor Processing Unit for a Matrix Multiply: a Review, 2022.

[143] Xing Lin, Yair Rivenson, Nezih T. Yardimci, Muhammed Veli, Yi Luo, Mona Jarrahi, and Aydogan Ozcan. All-optical machine learning using diffractive deep neural networks. *Science*, 361(6406):1004–1008, 2018.

[144] Alexander N. Tait. Quantifying Power in Silicon Photonic Neural Networks. *Phys. Rev. Appl.*, 17:054029, May 2022.

[145] Raphael Cardoso, Clément Zrounba, Mohab Abdalla, Paul Jimenez, Mauricio Gomes de Queiroz, Benoît Charbonnier, Fabio Pavanello, Ian O'Connor, and Sébastien Le Beux. Towards a Robust Multiply-Accumulate Cell in Photonics using Phase-Change Materials. In *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–2, 2023.

[146] Johannes Feldmann, Nathan Youngblood, Maxim Karpov, Helge Gehring, Xuan Li, Maik Stappers, Manuel Le Gallo, Xin Fu, Anton Lukashchuk, Arslan Sajid Raja, et al. Parallel convolutional processing using an integrated photonic tensor core. *Nature*, 589(7840):52–58, 2021.

[147] David A. B. Miller. Attojoule Optoelectronics for Low-Energy Information Processing and Communications. *Journal of Lightwave Technology*, 35(3):346–396, 2017.

[148] Sandeep R Agrawal, Sam Idicula, Arun Raghavan, Evangelos Vlachos, Venkatraman Govindaraju, Venkatanathan Varadarajan, Cagri Balkesen, Georgios Giannikis, Charlie Roth, Nipun Agarwal, and Eric Sedlar. A Many-Core Architecture for in-Memory Data Processing. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-50 '17, page 245–258, New York, NY, USA, 2017. Association for Computing Machinery.

[149] Vinay Joshi, Manuel Le Gallo, Simon Haefeli, Irem Boybat, S. R. Nandakumar, Christophe Piveteau, Martino Dazzi, Bipin Rajendran, Abu Sebastian, and Evangelos Eleftheriou. Accurate deep neural network inference using computational phase-change memory. *Nature Communications*, 11, 2019.

[150] Tae-Youl Yang, Il-Mok Park, Byoung-Joon Kim, and Young-Chang Joo. Atomic migration in molten and crystalline Ge2Sb2Te5 under high electric field. *Applied Physics Letters*, 95(3), 07 2009. 032104.

[151] Wabe Koelmans, Abu Sebastian, Vara Jonnalagadda, Daniel Krebs, Laurent Dellmann, and Evangelos Eleftheriou. Projected phase-change memory devices. *Nature communications*, 6:8181, 09 2015.

[152] G. Snider. Computing with hysteretic resistor crossbars. *Applied Physics A*, 80:1165–1172, 03 2005.

[153] Muath Lebdeh, Uljana Reinsalu, Hoang Anh Du Nguyen, Stephan Wong, and Said Hamdioui. Memristive Device Based Circuits for Computation-in-Memory Architectures. In *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, 05 2019.

[154] Pierre-Emmanuel Gaillardon, Luca Amarú, Anne Siemon, Eike Linn, Rainer Waser, Anupam Chattopadhyay, and Giovanni De Micheli. The Programmable Logic-in-Memory (PLiM) Computer. In *Proceedings of the 2016 Conference on Design, Automation & Test in Europe*, DATE '16, page 427–432, San Jose, CA, USA, 2016. EDA Consortium.

[155] Catherine Graves, Can Li, Xia Sheng, Darrin Miller, Jim Ignowski, Lennie Kiyama, and John Paul Strachan. In-Memory Computing with Memristor Content Addressable Memories for Pattern Matching. *Advanced Materials*, 32:2003437, 08 2020.

[156] Daichi Fujiki, Scott Mahlke, and Reetuparna Das. In-Memory Data Parallel Processor. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '18, page 1–14, New York, NY, USA, 2018. Association for Computing Machinery.

[157] Linghao Song, Xuehai Qian, Hai Li, and Yiran Chen. Pipelayer: A pipelined reram-based accelerator for deep learning. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 541–552, 2017.

[158] Saransh Gupta, Mohsen Imani, and Tajana Rosing. FELIX: Fast and Energy-Efficient Logic in Memory. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–7, 2018.

[159] Nima TaheriNejad. SIXOR: Single-Cycle In-Memristor XOR. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 29(5):925–935, 2021.

[160] Yichen Shen, Nicholas C. Harris, Scott Skirlo, Dirk Englund, and Marin Soljačić. Deep learning with coherent nanophotonic circuits. In *2017 IEEE Photonics Society Summer Topical Meeting Series (SUM)*, pages 189–190, 2017.

[161] Daniel Sturm and Sajjad Moazeni. Scalable Coherent Optical Crossbar Architecture using PCM for AI Acceleration, 10 2022.

[162] Erwin Dijk, Yan Jaszczyszyn, Delphine Naquin, and Claude Thermes. The Third Revolution in Sequencing Technology. *Trends in Genetics*, 34, 06 2018.

[163] Simon Ardui, Adam Ameur, Joris Vermeesch, and Matthew Hestand. Single molecule real-time (SMRT) sequencing comes of age: Applications and utilities for medical diagnostics. *Nucleic acids research*, 46, 02 2018.

[164] Miten Jain, S Koren, Joshua Quick, AC Rand, Thomas Sasani, JR Tyson, Andrew Beggs, Alexander Dilthey, Ian Fiddes, S Malla, H Marriott, KH Miga, Thomas Nieto, Justin O'Grady, Hugh Olsen, BS Pedersen, Arang Rhie, H Richardson, Aaron Quinlan, and Matthew Loose. Nanopore sequencing and assembly of a human genome with ultra-long reads. *Nature Biotechnology*, 04 2017.

[165] Mehdi Kchouk, Jean-Francois Gibrat, and Mourad Elloumi. Generations of Sequencing Technologies: From First to Next Generation. *Biology and Medicine*, 09, 01 2017.

[166] Jason Weirather, Mariateresa de Cesare, Yunhao Wang, Paolo Piazza, Vittorio Sebastiano, Xiu-Jie Wang, David Buck, and Kin Au. Comprehensive comparison of Pacific Biosciences and Oxford Nanopore Technologies and their applications to transcriptome analysis. *F1000Research*, 6:100, 02 2017.

[167] Yunhao Wang, Yue Zhao, Audrey Bollas, Yuru Wang, and Kin Fai Au. Nanopore sequencing technology, bioinformatics and applications. *Nature Biotechnology*, 39:1348 – 1365, 2021.

[168] Marc Pagès-Gallego and Jeroen de Ridder. Comprehensive benchmark and architectural analysis of deep learning models for Nanopore sequencing basecalling, 2022.

[169] Ryan Wick, Louise Judd, and Kathryn Holt. Performance of neural network basecalling tools for Oxford Nanopore sequencing. *Genome Biology*, 20, 06 2019.

[170] Raquel Dias and Ali Torkamani. Artificial intelligence in clinical and genomic diagnostics. *Genome Medicine*, 11, 11 2019.

[171] Yao-zhong Zhang, Arda Akdemir, Georg Tremmel, Seiya Imoto, Satoru Miyano, Tetsuo Shibuya, and Rui Yamaguchi. Nanopore basecalling from a perspective of instance segmentation. *BMC Bioinformatics*, 21:136, 04 2020.

[172] Franka Rang, Wigard Kloosterman, and Jeroen De Ridder. From squiggle to basepair: Computational approaches for improving nanopore sequencing read accuracy. *Genome Biology*, 19, 07 2018.

[173] Oxford Nanopore Technologies Ltd. Developers. Flappie. https://github.com/nanoporetech/flappie, 2018.

[174] Oxford Nanopore Technologies Ltd. Developers. Scrappie. https://github.com/nanoporetech/scrappie, 2019.

[175] Oxford Nanopore Technologies Ltd. Developers. Bonito. https://github.com/nanoporetech/bonito, 2020.

[176] Oxford Nanopore Technologies Ltd. Developers. Metrichor. https://metrichor.com, 2017.

[177] Zhimeng Xu, Yuting Mai, Denghui Liu, Wenjun He, Xinyuan Lin, Chi Xu, Lei Zhang, Xin Meng, Joseph Mafofo, Walid Abbas Zaher, Ashish Koshy, Yi Li, and Nan Qiao. Fast-bonito: A faster deep learning based basecaller for nanopore sequencing. *Artificial Intelligence in the Life Sciences*, 1:100011, 2021.

[178] Gagandeep Singh, Mohammed Alser, Alireza Khodamoradi, Kristof Denolf, Can Firtina, Meryem Banu Cavlak, Henk Corporaal, and Onur Mutlu. A framework for designing efficient deep learning-based genomic basecallers, 2023.

[179] Qian Lou, Sarath Chandra Janga, and Lei Jiang. Helix: Algorithm/Architecture Co-Design for Accelerating Nanopore Genome Base-Calling. In *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques*, PACT '20, page 293–304, New York, NY, USA, 2020. Association for Computing Machinery.

[180] J Shendure and H.L. Ji. Next-generation DNa sequencing. *Nat. Biotechnol.*, 26:1135–1145, 01 2008.

[181] Michael L Metzker. Sequencing technologies—the next generation. *Nature reviews genetics*, 11(1):31–46, 2010.

[182] Mohammed Alser, Joel Lindegger, Can Firtina, Nour Almadhoun, Haiyu Mao, Gagandeep Singh, Juan Gomez-Luna, and Onur Mutlu. From molecules to genomic variations: Accelerating genome analysis via intelligent algorithms and architectures. *Computational and Structural Biotechnology Journal*, 20:4579–4599, 2022.

[183] Sean Eddy. What is dynamic programming? *Nature biotechnology*, 22:909–10, 08 2004.

[184] Saul B. Needleman and Christian D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, 1970.

[185] T.F. Smith and M.S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, 1981.

[186] Mohammed Alser, Taha Shahroodi, Juan Gómez-Luna, Can Alkan, and Onur Mutlu. SneakySnake: a fast and accurate universal genome pre-alignment filter for CPUs, GPUs and FPGAs. *Bioinformatics*, 36(22-23):5282–5290, 12 2020.

[187] Mohammed Alser, Hasan Hassan, Akash Kumar, Onur Mutlu, and Can Alkan. Shouji: a fast and efficient pre-alignment filter for sequence alignment. *Bioinformatics*, 35(21):4255–4263, 03 2019.

[188] Mohammed Alser, Onur Mutlu, and Can Alkan. MAGNET: Understanding and Improving the Accuracy of Genome Pre-Alignment Filtering. *arXiv*, 2017.

[189] Hongyi Xin, John Greth, John Emmons, Gennady Pekhimenko, Carl Kingsford, Can Alkan, and Onur Mutlu. Shifted Hamming distance: a fast and accurate SIMD-friendly filter to accelerate alignment verification in read mapping. *Bioinformatics*, 31(10):1553–1560, 01 2015.

[190] Jo Handelsman, Michelle R. Rondon, Sean F. Brady, Jon Clardy, and Robert M. Goodman. Molecular biological access to the chemistry of unknown soil microbes: a new frontier for natural products. *Chemistry & Biology*, 5(10):R245–R249, 1998.

[191] Michelle Rondon, Paul August, Alan Bettermann, Sean Brady, Trudy Grossman, Mark Liles, Kara Loiacono, Berkley Lynch, Ian Macneil, Charles Minor, Choi-Lai Tiong-Yip, Michael Gilman, Marcia Osburne, JON Clardy, Jo Handelsman, and Robert Goodman. Cloning the Soil Metagenome: a Strategy for Accessing the Genetic and Functional Diversity of Uncultured Microorganisms. *Applied and Environmental Microbiology*, 66, 07 2000.

[192] Nathan LaPierre, Mohammed Alser, Eleazar Eskin, David Koslicki, and Serghei Mangul. Metalign: Efficient alignment-based metagenomic profiling via containment min hash. *Genome biology*, 21:242, 09 2020.

[193] Derrick Wood, Jennifer Lu, and Ben Langmead. Improved metagenomic analysis with Kraken 2, 09 2019.

[194] Ana Pérez-Cobas, Laura Gomez-Valero, and Carmen Buchrieser. Metagenomic approaches in microbial ecology: An update on whole-genome and marker gene sequencing analyses. *Microbial Genomics*, 6, 07 2020.

[195] Nicola Segata, Levi Waldron, Annalisa Ballarini, Vagheesh Narasimhan, Olivier Jousson, and Curtis Huttenhower. Metagenomic microbial community profiling using unique clade-specific marker genes. *Nature methods*, 9:811–4, 06 2012.

[196] Duy Tin Truong, Eric Franzosa, Timothy Tickle, Matthias Scholz, George Weingart, Edoardo Pasolli, Adrian Tett, Curtis Huttenhower, and Nicola Segata. MetaPhlAn2 for Enhanced Metagenomic Taxonomic Profiling. *Nature Methods*, 12:902–903, 10 2015.

[197] Arthur Brady and Steven Salzberg. PhymmBL expanded: Confidence scores, custom databases, parallelization and more. *Nature methods*, 8:367, 05 2011.

[198] Ivan Gregor, Johannes Dröge, Melanie Schirmer, Christopher Quince, and Alice McHardy. PhyloPythiaS+: A self-training method for the rapid reconstruction of low-ranking taxonomic bins from metagenomes. *PeerJ*, 4, 06 2014.

[199] Folker Meyer, Saurabh Bagchi, Somali Chaterji, Wolfgang Gerlach, Ananth Grama, Travis Harrison, Tobias Paczian, Will Trimble, and Andreas Wilke. MG-RAST version 4-lessons learned from a decade of low-budget ultra-high-throughput metagenome analysis. *Briefings in bioinformatics*, 20, 09 2017.

[200] Daniel Huson, Sina Beier, Isabell Flade, Anna Górska, Mohamed El-Hadidi, Suparna Mitra, Hans-Joachim Ruscheweyh, and Rewati Tappu. MEGAN Community Edition - Interactive Exploration and Analysis of Large-Scale Microbiome Sequencing Data. *PLoS computational biology*, 12:e1004957, 06 2016.

[201] J. Dröge, Ivan Gregor, and Alice McHardy. Taxator-tk: Precise Taxonomic Assignment of Metagenomes by Fast Approximation of Evolutionary Neighborhoods. *Bioinformatics (Oxford, England)*, 31, 03 2014.

[202] Derrick E Wood and Steven L Salzberg. Kraken: ultrafast metagenomic sequence classification using exact alignments. *Genome biology*, 15(3):R46–R46, 2014.

[203] Rachid Ounit, Steve Wanamaker, Timothy Close, and Stefano Lonardi. CLARK: Fast and accurate classification of metagenomic and genomic sequences using discriminative k-mers. *BMC Genomics*, 16, 03 2015.

[204] Jennifer Lu, Florian Breitwieser, Peter Thielen, and Steven Salzberg. Bracken: Estimating species abundance in metagenomics data. *PeerJ Computer Science*, 3:e104, 01 2017.

[205] Fernando Meyer, Adrian Fritz, Zhi-Luo Deng, David Koslicki, Till Lesker, Alexey Gurevich, Gary Robertson, Mohammed Alser, Dmitry Antipov, Francesco Beghini, Denis Bertrand, Jaqueline Brito, C. Titus Brown, Jan Buchmann, Aydin Buluç, Bo Chen, Rayan Chikhi, Philip Clausen, Alexandru Cristian, and Alice McHardy. Critical Assessment of Metagenome Interpretation: the second round of challenges. *Nature Methods*, 19:1–12, 04 2022.

[206] Alexander Sczyrba, Peter Hofmann, Peter Belmann, David Koslicki, Stefan Janssen, Johannes Dröge, Ivan Gregor, Stephan Majda, Jessika Fiedler, Eik Dahms, Andreas Bremges, Adrian Fritz, Ruben Garrido-Oter, Tue Jørgensen, Nicole Shapiro, Philip Blood, Alexey Gurevich, Yang Bai, Dmitrij Turaev, and Alice McHardy. Critical Assessment of Metagenome Interpretation – a benchmark of computational metagenomics software. *Nature Methods*, 06 2017.

[207] Michael Roberts, Wayne Hayes, Brian Hunt, Steve Mount, and James Yorke. Reducing storage requirements for biological sequence comparison. *Bioinformatics (Oxford, England)*, 20:3363–9, 01 2005.

[208] Ben Langmead, Christopher Wilks, Valentin Antonescu, and Rone Charles. Scaling read aligners to hundreds of threads on general-purpose processors. *Bioinformatics*, 35, 07 2018.

[209] David J. Agnew, John Pearce, Ganapathiraju Pramod, Tom Peatman, Reg Watson, John R. Beddington, and Tony J. Pitcher. Estimating the Worldwide Extent of Illegal Fishing. *PLOS ONE*, 4(2):1–8, 02 2009.

[210] Robert Smith. Rural rogues: A case story on the 'smokies' trade. *International Journal of Entrepreneurial Behaviour & Research*, 10:277–294, 08 2004.

[211] Robert Smith. Documenting the UK "Black Fish Scandal" as a case study of criminal entrepreneurship. *International Journal of Sociology and Social Policy*, 35:199–221, 04 2015.

[212] Robin Kobus, José Abuín Mosquera, André Müller, Sören Hellmann, Juan Pichel, Tomas Pena, Andreas Hildebrandt, Thomas Hankeln, and Bertil Schmidt. A big data approach to metagenomics for all-food-sequencing. *BMC Bioinformatics*, 21, 03 2020.

[213] Fabian Ripp, Christopher Krombholz, Yongchao Liu, Mathias Weber, Anne Schaefer, Bertil Schmidt, René Köppel, and Thomas Hankeln. All-Food-Seq (AFS): A quantifiable screen for species in biological samples by deep DNA sequencing. *BMC Genomics*, 15, 07 2014.

[214] Chunyu Yuan and Sos S. Agaian. A comprehensive review of Binary Neural Network. *Artificial Intelligence Review*, pages 1–65, 2021.

[215] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. BinaryConnect: Training Deep Neural Networks with Binary Weights during Propagations. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, page 3123–3131, Cambridge, MA, USA, 2015. MIT Press.

[216] Haotong Qin, Ruihao Gong, Xianglong Liu, Xiao Bai, Jingkuan Song, and Nicu Sebe. Binary neural networks: A survey. *Pattern Recognition*, 105:107281, 2020.

[217] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[218] Jiawen Liu, Hengyu Zhao, Matheus A. Ogleari, Dong Li, and Jishen Zhao. Processing-in-Memory for Energy-Efficient Neural Network Training: A Heterogeneous Approach. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 655–668, 2018.

[219] Jia Zhan, Onur Kayıran, Gabriel H. Loh, Chita R. Das, and Yuan Xie. OSCAR: Orchestrating STT-RAM cache traffic for heterogeneous CPU-GPU architectures. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1–13, 2016.

[220] Sunil Shukla, Bruce Fleischer, Matthew Ziegler, Joel Silberman, Jinwook Oh, Vijayalakshmi Srinivasan, Jungwook Choi, Silvia Mueller, Ankur Agrawal, Tina Babinsky, Nianzheng Cao, Chia-Yu Chen, Pierce Chuang, Thomas Fox, George Gristede, Michael Guillorn, Howard Haynie, Michael Klaiber, Dongsoo Lee, Shih-Hsien Lo, Gary Maier, Michael Scheuermann, Swagath Venkataramani, Christos Vezyrtzis, Naigang Wang, Fanchieh Yee, Ching Zhou, Pong-Fei Lu, Brian Curran, Leland Chang, and Kailash Gopalakrishnan. A Scalable Multi-TeraOPS Core for AI Training and Inference. *IEEE Solid-State Circuits Letters*, 1(12):217–220, 2018.

[221] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, pages 525–542, Cham, 2016. Springer International Publishing.

[222] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1, 2016.

[223] Fabrice Devaux. The true Processing In Memory accelerator. In *2019 IEEE Hot Chips 31 Symposium (HCS)*, pages 1–24, 2019.

[224] Introduction to UPMEM PIM. Processing-in-memory (PIM) on DRAM Accelerator (White Paper). In *UPMEM, Grenoble, France*, 2018.

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

[226] Ankit, Aayush and Kim, Dong-Eun and Chakraborty, Indranil and Ali, Mustafa and Negi, Shubham. PUMA Functional Simulator. https://github.com/Aayush-Ankit/puma-functional-model, 2020.

[227] Dong, Xiangyu and Xu, Cong and Xie, Yuan and Jouppi, Norman P. 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, 2012.

[228] Matt Poremba and Yuan Xie. NVMain: An Architectural-Level Main Memory Simulator for Emerging Non-volatile Memories. In *2012 IEEE Computer Society Annual Symposium on VLSI*, pages 392–397, 2012.

[229] Naveen Muralimanohar, Rajeev Balasubramonian, and Norman Jouppi. Cacti 6.0: A tool to model large caches. *HP Laboratories*, 01 2009.

[230] Matthew Kay Fei Lee, Yingnan Cui, Thannirmalai Somu, Tao Luo, Jun Zhou, Wai Teng Tang, Weng-Fai Wong, and Rick Siow Mong Goh. A System-Level Simulator for RRAM-Based Neuromorphic Computing Chips. *ACM Trans. Archit. Code Optim.*, 15(4), jan 2019.

[231] Damla Senol Cali, Gurpreet S. Kalsi, Zülal Bingöl, Can Firtina, Lavanya Subramanian, Jeremie S. Kim, Rachata Ausavarungnirun, Mohammed Alser, Juan Gomez-Luna, Amirali Boroumand, Anant Norion, Allison Scibisz, Sreenivas Subramoneyon, Can Alkan, Saugata Ghose, and Onur Mutlu. GenASM: A High-Performance, Low-Power Approximate String Matching Acceleration Framework for Genome Sequence Analysis. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 951–966, 2020.

[232] Yatish Turakhia, Gill Bejerano, and William J. Dally. Darwin: A Genomics Co-Processor Provides up to 15,000X Acceleration on Long Read Assembly. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '18, page 199–213, New York, NY, USA, 2018. Association for Computing Machinery.

[233] Anirban Nag, C. N. Ramachandra, Rajeev Balasubramonian, Ryan Stutsman, Edouard Giacomin, Hari Kambalasubramanyam, and Pierre-Emmanuel Gaillardon. GenCache: Leveraging In-Cache Operators for Efficient Sequence Alignment. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '52, page 334–346, New York, NY, USA, 2019. Association for Computing Machinery.

[234] Daichi Fujiki, Arun Subramaniyan, Tianjun Zhang, Yu Zeng, Reetuparna Das, David Blaauw, and Satish Narayanasamy. GenAx: A Genome Sequencing Accelerator. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 69–82, 2018.

[235] Ying Wang, Yinhe Han, Cheng Wang, Huawei Li, and Xiao-Wei Li. RADAR: a case for retention-aware DRAM assembly and repair in future FGR DRAM memory. 06 2015.

[236] Shaahin Angizi, Jiao Sun, Wei Zhang, and Deliang Fan. PIM-Aligner: A Processing-in-MRAM Platform for Biological Sequence Alignment. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1265–1270, 2020.

[237] Shaahin Angizi, Jiao Sun, Wei Zhang, and Deliang Fan. AlignS: A Processing-In-Memory Accelerator for DNA Short Read Alignment Leveraging SOT-MRAM. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2019.

[238] Shaahin Angizi, Wei Zhang, and Deliang Fan. Exploring DNA Alignment-in-Memory Leveraging Emerging SOT-MRAM. In *Proceedings of the 2020 on Great Lakes Symposium on VLSI*, GLSVLSI '20, page 277–282, New York, NY, USA, 2020. Association for Computing Machinery.

[239] Yeseong Kim, Mohsen Imani, Niema Moshiri, and Tajana Rosing. GenieHD: Efficient DNA Pattern Matching Accelerator Using Hyperdimensional Computing. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 115–120, 2020.

[240] Mohsen Imani, Tarek Nassar, Abbas Rahimi, and Tajana Rosing. HDNA: Energy-efficient DNA sequencing using hyperdimensional computing. In *2018 IEEE EMBS International Conference on Biomedical & Health Informatics (BHI)*, pages 271–274, 2018.

[241] Farzaneh Zokaee, Mingzhe Zhang, and Lei Jiang. FindeR: Accelerating FM-Index-Based Exact Pattern Matching in Genomic Sequences through ReRAM Technology. In *2019 28th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 284–295, 2019.

[242] L. Jiang and F. Zokaee. EXMA: A Genomics Accelerator for Exact-Matching. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 399–411, Los Alamitos, CA, USA, mar 2021. IEEE Computer Society.

[243] Ann Franchesca Laguna, Hasindu Gamaarachchi, Xunzhao Yin, Michael Niemier, Sri Parameswaran, and X. Sharon Hu. Seed-and-Vote Based in-Memory Accelerator for DNA Read Mapping. In *Proceedings of the 39th International Conference on Computer-Aided Design*, ICCAD '20, New York, NY, USA, 2020. Association for Computing Machinery.

[244] Lingxi Wu, Rasool Sharifi, Marzieh Lenjani, Kevin Skadron, and Ashish Venkat. Sieve: Scalable In-situ DRAM-based Accelerator Designs for Massively Parallel k-mer Matching. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pages 251–264, 2021.

[245] Wenqin Huangfu, Xueqi Li, Shuangchen Li, Xing Hu, Peng Gu, and Yuan Xie. MEDAL: Scalable DIMM Based Near Data Processing Accelerator for DNA Seeding Algorithm. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '52, page 587–599, New York, NY, USA, 2019. Association for Computing Machinery.

[246] Wenqin Huangfu, Krishna T. Malladi, Shuangchen Li, Peng Gu, and Yuan Xie. NEST: DIMM Based near-Data-Processing Accelerator for K-Mer Counting. In *Proceedings of the 39th International Conference on Computer-Aided Design*, ICCAD '20, New York, NY, USA, 2020. Association for Computing Machinery.

[247] Ming Cheng, Lixue Xia, Zhenhua Zhu, Yi Cai, Yuan Xie, Yu Wang, and Huazhong Yang. TIME: A training-in-memory architecture for memristor-based deep neural networks. In *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, 2017.

[248] Yue Xi, Bin Gao, Jianshi Tang, An Chen, Meng-Fan Chang, Xiaobo Sharon Hu, Jan Van der Spiegel, He Qian, and Huaqiang Wu. In-memory Learning with Analog Resistive Switching Memory: A Review and Perspective. *Proceedings of the IEEE*, 109:14–42, 2021.

[249] Xiaoyang Liu and Zhigang Zeng. Memristor crossbar architectures for implementing deep neural networks. *Complex & Intelligent Systems*, 8:787–802, 07 2021.

[250] Fernando Corinto, Alon Ascoli, and Steve Kang Sung-Mo. Memristor-based neural circuits. In *2013 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 417–420, 2013.

[251] Amirali Amirsoleimani, Majid Ahmadi, and Arash Ahmadi. STDP-based unsupervised learning of memristive spiking neural network by Morris-Lecar model. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 3409–3414, 2017.

[252] Jihong Zhang and Xiaofeng Liao. Synchronization and chaos in coupled memristor-based FitzHugh-Nagumo circuits with memristor synapse. *AEU - International Journal of Electronics and Communications*, 75:82–90, 2017.

[253] S. Lashkare, S. Chouhan, T. Chavan, A. Bhat, P. Kumbhare, and U. Ganguly. PCMO RRAM for Integrate-and-Fire Neuron in Spiking Neural Networks. *IEEE Electron Device Letters*, 39(4):484–487, 2018.

[254] Maruan Al-Shedivat, Rawan Naous, Gert Cauwenberghs, and Khaled Nabil Salama. Memristors Empower Spiking Neurons With Stochasticity. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 5(2):242–253, 2015.

[255] Jafar Shamsi, Amirali Amirsoleimani, Sattar Mirzakuchaki, and Majid Ahmadi. Modular neuron comprises of memristor-based synapse. *Neural Computing and Applications*, 09 2015.

[256] Adnan Mehonic and Anthony J. Kenyon. Emulating the Electrical Activity of the Neuron Using a Silicon Oxide RRAM Cell. *Frontiers in Neuroscience*, 10, 2016.

[257] Jia-Qin Yang, Ruopeng Wang, Zhan-Peng Wang, Qin-Yuan Ma, Jing-Yu Mao, Yi Ren, Xiaoyang Yang, Ye Zhou, and Su-Ting Han. Leaky integrate-and-fire neurons based on perovskite memristor for spiking neural networks. *Nano Energy*, 74:104828, 2020.

[258] Zhenyu Zhao, Lianhua Qu, Lei Wang, Quan Deng, Nan Li, Ziyang Kang, Shasha Guo, and Weixia Xu. A memristor-based spiking neural network with high scalability and learning efficiency. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 67(5):931–935, 2020.

[259] Nan Zheng and Pinaki Mazumder. Online Supervised Learning for Hardware-Based Multilayer Spiking Neural Networks Through the Modulation of Weight-Dependent Spike-Timing-Dependent Plasticity. *IEEE Transactions on Neural Networks and Learning Systems*, 29(9):4287–4302, 2018.

[260] Nan Zheng and Pinaki Mazumder. Learning in Memristor Crossbar-Based Spiking Neural Networks Through Modulation of Weight-Dependent Spike-Timing-Dependent Plasticity. *IEEE Transactions on Nanotechnology*, 17(3):520–532, 2018.

[261] Yuan Zeng, Kevin Devincentis, Yao Xiao, Zubayer Ibne Ferdous, Xiaochen Guo, Zhiyuan Yan, and Yevgeny Berdichevsky. A Supervised Stdp-Based Training Algorithm for Living Neural Networks. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1154–1158, 2018.

[262] Giacomo Pedretti, Valerio Milo, S. Ambrogio, Roberto Carboni, S. Bianchi, Alessandro Calderoni, Nirmal Ramaswamy, A. Spinelli, and D. Ielmini. Memristive neural network for on-line learning and tracking with brain-inspired spike timing dependent plasticity. *Scientific Reports*, 7, 07 2017.

[263] Yu Nishitani, Yukihiro Kaneko, and Michihito Ueda. Supervised Learning Using Spike-Timing-Dependent Plasticity of Memristive Synapses. *IEEE Transactions on Neural Networks and Learning Systems*, 26:2999–3008, 2015.

[264] Yun Long, Taesik Na, and Saibal Mukhopadhyay. ReRAM-Based Processing-in-Memory Architecture for Recurrent Neural Network Acceleration. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 26(12):2781–2794, 2018.

[265] Kamilya Smagulova and A. James. A survey on LSTM memristive neural network architectures and applications. *The European Physical Journal Special Topics*, 228, 05 2019.

[266] H. Tsai, S. Ambrogio, C. Mackin, P. Narayanan, R. M. Shelby, K. Rocki, A. Chen, and G. W. Burr. Inference of Long-Short Term Memory networks at software-equivalent accuracy using 2.5M analog Phase Change Memory devices. In *2019 Symposium on VLSI Technology*, pages T82–T83, 2019.

[267] Can Li, Zhongrui Wang, Mingyi Rao, Daniel Belkin, Wenhao Song, Hao Jiang, Peng Yan, Yunning Li, Peng Lin, Miao Hu, Ning Ge, John William Strachan, Mark Barnell, Qing wu, Stan Williams, Jianhua Joshua Yang, and Qiangfei Xia. Long short-term memory networks in memristor crossbar arrays. *Nature Machine Intelligence*, 1, 01 2019.

[268] Kazybek Adam, Kamilya Smagulova, and Alex Pappachen James. Memristive LSTM network hardware architecture for time-series predictive modeling problems. In *2018 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*, pages 459–462, 2018.

[269] Kamilya Smagulova, Kazybek Adam, Olga Krestinskaya, and Alex Pappachen James. Design of CMOS-memristor Circuits for LSTM architecture. In *2018 IEEE International Conference on Electron Devices and Solid State Circuits (EDSSC)*, pages 1–2, 2018.

[270] Kamilya Smagulova, Olga Krestinskaya, and Alex Pappachen James. A Memristor-Based Long Short Term Memory Circuit. *Analog Integr. Circuits Signal Process.*, 95(3):467–472, jun 2018.

[271] Taha Shahroodi, Gagandeep Singh, Mahdi Zahedi, Haiyu Mao, Joel Lindegger, Can Firtina, Stephan Wong, Onur Mutlu, and Said Hamdioui. Swordfish: A framework for evaluating deep neural network-based basecalling using computation-in-memory with non-ideal memristors. *arXiv preprint arXiv:2310.04366*, 2023.

[272] Damla Senol Cali, Jeremie S Kim, Saugata Ghose, Can Alkan, and Onur Mutlu. Nanopore Sequencing Technology and Tools for Genome Assembly: Computational Analysis of the Current State, Bottlenecks and Future Directions. *Briefings in Bioinformatics*, 20(4):1542–1559, 04 2018.

[273] Mohammed Alser, Joel Lindegger, Can Firtina, Nour Almadhoun, Haiyu Mao, Gagandeep Singh, Juan Gomez-Luna, and Onur Mutlu. From molecules to genomic variations: Accelerating genome analysis via intelligent algorithms and architectures. *Computational and Structural Biotechnology Journal*, 20:4579–4599, 2022.

[274] Gagandeep Singh, Mohammed Alser, Alireza Khodamoradi, Kristof Denolf, Can Firtina, Meryem Banu Cavlak, Henk Corporaal, and Onur Mutlu. A Framework for Designing Efficient Deep Learning-Based Genomic Basecallers. *arXiv preprint arXiv:2211.03079*, 2022.

[275] Onur Mutlu and Can Firtina. Accelerating Genome Analysis via Algorithm-Architecture Co-Design. In *DAC*, pages 1–4, 07 2023.

[276] Amirali Boroumand, Saugata Ghose, Berkin Akin, Ravi Narayanaswami, Geraldo F. Oliveira, Xiaoyu Ma, Eric Shiu, and Onur Mutlu. Google Neural Network Models for Edge Devices: Analyzing and Mitigating Machine Learning Inference Bottlenecks. In *2021 30th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 159–172, Los Alamitos, CA, USA, sep 2021. IEEE Computer Society.

[277] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A. Alemi. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, page 4278–4284. AAAI Press, 2017.

[278] Benjamin C. Lee, Engin Ipek, Onur Mutlu, and Doug Burger. Architecting phase change memory as a scalable dram alternative. In *International Symposium on Computer Architecture*, 2009.

[279] Abhairaj Singh, Mahdi Zahedi, Taha Shahroodi, Mohit Gupta, Anteneh Gebregiorgis, Manu Komalan, Rajiv V Joshi, Francky Catthoor, Rajendra Bishnoi, and Said Hamdioui. Cim-based robust logic accelerator using 28 nm stt-mram characterization chip tape-out. In *2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pages 451–454. IEEE, 2022.

[280] Mirko Prezioso, Farnood Merrikh-Bayat, Brian Hoskins, Gina Adam, Konstantin Likharev, and Dmitri Strukov. Training and Operation of an Integrated Neuromorphic Network Based on Metal-Oxide Memristors. *Nature*, 521, 12 2014.

[281] Onur Mutlu, Saugata Ghose, Juan Gómez-Luna, and Rachata Ausavarungnirun. A Modern Primer on Processing in Memory. pages 171–243, 2022.

[282] Lingyun Shi, Guohao Zheng, Bobo Tian, Brahim Dkhil, and Chungang Duan. Research progress on solutions to the sneak path issue in memristor crossbar arrays. *Nanoscale Advances*, 2(5):1811–1827, 2020.

[283] Miao Hu, Catherine E Graves, Can Li, Yunning Li, Ning Ge, Eric Montgomery, Noraica Davila, Hao Jiang, R Stanley Williams, J Joshua Yang, et al. Memristor-based analog computation and neural network classification with a dot product engine. *Advanced Materials*, 30(9):1705914, 2018.

[284] Wenqiang Zhang, Xiaochen Peng, Huaqiang Wu, Bin Gao, Hu He, Youhui Zhang, Shimeng Yu, and He Qian. Design guidelines of rram based neural-processing-unit: A joint device-circuit-algorithm analysis. In *Proceedings of the 56th Annual Design Automation Conference 2019*, pages 1–6, 2019.

[285] VG Karpov, YA Kryukov, SD Savransky, and IV Karpov. Nucleation switching in phase change memory. *Applied physics letters*, 90(12):123504, 2007.

[286] Mohamad G. Moinuddin, Aijaz H. Lone, Shivangi Shringi, Srikant Srinivasan, and Satinder K. Sharma. Low-current-density magnetic tunnel junctions for stt-ram application using mgo $_x$ n $_{1-x}$ ($x = 0.57$) tunnel barrier. *IEEE Transactions on Electron Devices*, 67(1):125–132, 2020.

[287] Daniel Branton, David W Deamer, Andre Marziali, Hagan Bayley, Steven A Benner, Thomas Butler, Massimiliano Di Ventra, Slaven Garaj, Andrew Hibbs, Xiaohua Huang, et al. The potential and challenges of nanopore sequencing. *Nature biotechnology*, 26(10):1146–1153, 2008.

[288] Said Hamdioui, Lei Xie, Hoang Anh Du Nguyen, Mottaqiallah Taouil, Koen Bertels, Henk Corporaal, Hailong Jiao, Francky Catthoor, Dirk Wouters, Linn Eike, and Jan van Lunteren. Memristor based computation-in-memory architecture for data-intensive applications. In *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1718–1725, 2015.

[289] Oxford Nanopore Technologies Ltd. Developers. GridION. https://nanoporetech.com/products/gridion, 2017.

[290] Oxford Nanopore Technologies Ltd. Developers. PromethION. https://nanoporetech.com/products/promethion-2, 2018.

[291] Oxford Nanopore Technologies Ltd. Developers. MinION. https://nanoporetech.com/products/minion, 2019.

[292] Tim Dunn, Harisankar Sadasivan, Jack Wadden, Kush Goliya, Kuan-Yu Chen, David Blaauw, Reetuparna Das, and Satish Narayanasamy. SquiggleFilter: An Accelerator for Portable Virus Detection. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '21, page 535–549, New York, NY, USA, 2021. Association for Computing Machinery.

[293] Heng Li. Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences. *Bioinformatics*, 32(14):2103–2110, 2016.

[294] Rainer Waser, Regina Dittmann, Georgi Staikov, and Kristof Szot. Redox-Based Resistive Switching Memories – Nanoionic Mechanisms, Prospects, and Challenges. *Advanced Materials*, 21, 2009.

[295] Benjamin C. Lee, Engin Ipek, Onur Mutlu, and Doug Burger. Phase change memory architecture and the quest for scalability. *Communications of the ACM*, 53:99 – 106, 2010.

[296] Geethan Karunaratne, Manuel Gallo, Giovanni Cherubini, Luca Benini, Abbas Rahimi, and Abu Sebastian. In-memory hyperdimensional computing. *Nature Electronics*, 3, 06 2020.

[297] Qiangfei Xia and J. Joshua Yang. Memristive crossbar arrays for brain-inspired computing. *Nature Materials*, 18:309–323, 2019.

[298] Long Cheng, Yi Li, Kang-Sheng Yin, Si-Yu Hu, Yu-Ting Su, Miao-Miao Jin, Zhuorui Wang, Ting-Chang Chang, and Xiang shui Miao. Functional Demonstration of a Memristive Arithmetic Logic Unit (MemALU) for In-Memory Computing. *Advanced Functional Materials*, 29, 2019.

[299] Dmitri Strukov, Gregory Snider, Duncan Stewart, and Stan Williams. The Missing Memristor Found. *Nature*, 453:80–3, 06 2008.

[300] Mahdi Zahedi, Taha Shahroodi, Geert Custers, Abhairaj Singh, Stephan Wong, and Said Hamdioui. System Design for Computation-in-Memory: From Primitive to Complex Functions. In *2022 IFIP/IEEE 30th International Conference on Very Large Scale Integration (VLSI-SoC)*, pages 1–6, 2022.

[301] Mahdi Zahedi, Taha Shahroodi, Stephan Wong, and Said Hamdioui. Efficient signed arithmetic multiplication on memristor-based crossbar. *IEEE Access*, 11:33964–33978, 2023.

[302] Fabien Alibart, Ligang Gao, Brian D. Hoskins, and Dmitri B. Strukov. High precision tuning of state for memristive devices by adaptable variation-tolerant algorithm. *Nanotechnology*, 23, 2011.

[303] Ching-Yi Chen, Hsiu-Chuan Shih, Cheng-Wen Wu, Chih-He Lin, Pi-Feng Chiu, Shyh-Shyuan Sheu, and Frederick T. Chen. RRAM Defect Modeling and Failure Analysis Based on March Test and a Novel Squeeze-Search Scheme. *IEEE Transactions on Computers*, 64:180–190, 2015.

[304] Jung-Hoon Lee, Dong-Hyeok Lim, Hongsik Jeong, Huimin Ma, and Luping Shi. Exploring Cycle-to-Cycle and Device-to-Device Variation Tolerance in MLC Storage-Based Neural Network Training. *IEEE Transactions on Electron Devices*, 66:2172–2178, 2019.

[305] A. Vittal, L.H. Chen, M. Marek-Sadowska, Kai-Ping Wang, and S. Yang. Crosstalk in VLSI Interconnections. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18(12):1817–1824, 1999.

[306] A. Vittal and M. Marek-Sadowska. Crosstalk Reduction for VLSI. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 16(3):290–298, 1997.

[307] S. Ramachandran, J.W. Nicholson, S. Ghalmi, and M.F. Yan. Measurement of Multipath Interference in the Coherent Crosstalk Regime. *IEEE Photonics Technology Letters*, 15(8):1171–1173, 2003.

[308] Geoffrey W. Burr, Robert M. Shelby, Abu Sebastian, Sangbum Kim, Seyoung Kim, Severin Sidler, Kumar Virwani, Masatoshi Ishii, Pritish Narayanan, Alessandro Fumarola, Lucas L. Sanches, Irem Boybat, Manuel Le Gallo, Kibong Moon, J. Woo, Hyunsang Hwang, and Yusuf Leblebici. Neuromorphic computing using non-volatile memory. *Advances in Physics: X*, 2:124 – 89, 2017.

[309] Leibin Ni, Zichuan Liu, Hao Yu, and Rajiv V. Joshi. An Energy-Efficient Digital ReRAM-Crossbar-Based CNN With Bitwise Parallelism. *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits*, 3:37–46, 2017.

[310] Suyog Gupta, Ankur Agrawal, K. Gopalakrishnan, and Pritish Narayanan. Deep Learning with Limited Numerical Precision. In *International Conference on Machine Learning*, 2015.

[311] Swagath Venkataramani, Ashish Ranjan, Kaushik Roy, and Anand Raghunathan. AxNN: Energy-efficient neuromorphic systems using approximate computing. *2014 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pages 27–32, 2014.

[312] Hokchhay Tann, Soheil Hashemi, Iris Bahar, and Sherief Reda. Hardware-software codesign of accurate, multiplier-free Deep Neural Networks. *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, 2017.

[313] Skanda Koppula, Lois Orosa, A. Giray Yağlıkçı, Roknoddin Azizi, Taha Shahroodi, Konstantinos Kanellopoulos, and Onur Mutlu. EDEN: Enabling Energy-Efficient, High-Performance Deep Neural Network Inference Using Approximate DRAM. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '52, page 166–181, New York, NY, USA, 2019. Association for Computing Machinery.

[314] Song Han, Huizi Mao, and William J. Dally. Deep Compression: Compressing Deep Neural Networks With Pruning, Trained Quantization and Huffman Coding. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[315] Kodai Ueyoshi, Kota Ando, Kazutoshi Hirose, Shinya Takamaeda-Yamazaki, Junichiro Kadomoto, Tomoki Miyata, Mototsugu Hamada, Tadahiro Kuroda, and Masato Motomura. QUEST: A 7.49 TOPS Multi-Purpose Log-Quantized DNN Inference Engine Stacked on 96MB 3D SRAM Using Inductive-Coupling Technology in 40nm CMOS. In *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*, pages 216–218, 2018.

[316] Weiwen Jiang, Qiuwen Lou, Zheyu Yan, Lei Yang, Jingtong Hu, Xiaobo Sharon Hu, and Yiyu Shi. Device-circuit-architecture co-exploration for computing-in-memory neural accelerators. *IEEE Transactions on Computers*, 70(4):595–605, 2020.

[317] Oxford Nanopore Technologies Ltd. Developers. Dorado. https://github.com/nanoporetech/dorado, 2022.

[318] Yun Long, Xueyuan She, and Saibal Mukhopadhyay. Design of reliable DNN accelerator with un-reliable ReRAM. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1769–1774. IEEE, 2019.

[319] Beiye Liu, Hai Li, Yiran Chen, Xin Li, Qing Wu, and Tingwen Huang. Vortex: Variation-aware training for memristor X-bar. In *Proceedings of the 52nd Annual Design Automation Conference*, pages 1–6, 2015.

[320] Lerong Chen, Jiawen Li, Yiran Chen, Qiuping Deng, Jiyuan Shen, Xiaoyao Liang, and Li Jiang. Accelerator-friendly neural-network training: Learning variations and defects in RRAM crossbar. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pages 19–24. IEEE, 2017.

[321] Michael Klachko, Mohammad Reza Mahmoodi, and Dmitri Strukov. Improving noise tolerance of mixed-signal neural networks. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2019.

[322] Markus Fritscher, Johannes Knödtel, Daniel Reiser, Maen Mallah, Stefan Pechmann, Dietmar Fey, and Marc Reichenbach. Simulating large neural networks embedding MLC RRAM as weight storage considering device variations. In *2021 IEEE 12th Latin America Symposium on Circuits and System (LASCAS)*, pages 1–4. IEEE, 2021.

[323] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2(7), 2015.

[324] Gouranga Charan, Abinash Mohanty, Xiaocong Du, Gokul Krishnan, Rajiv V Joshi, and Yu Cao. Accurate inference with inaccurate rram devices: A joint algorithm-design solution. *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits*, 6(1):27–35, 2020.

[325] Beiye Liu, Hai Li, Yiran Chen, Xin Li, Tingwen Huang, Qing Wu, and Mark Barnell. Reduction and IR-drop compensations techniques for reliable neuromorphic computing systems. In *2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 63–70. IEEE, 2014.

[326] Miao Hu, Hai Li, Yiran Chen, Qing Wu, and Garrett S Rose. BSB training scheme implementation on memristor-based circuit. In *2013 IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA)*, pages 80–87. IEEE, 2013.

[327] Neng Huang, Fan Nie, Peng Ni, Feng Luo, and Jianxin Wang. Sacall: a neural network basecaller for oxford nanopore sequencing data based on self-attention mechanism. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2020.

[328] Meng-Yao Lin, Hsiang-Yun Cheng, Wei-Ting Lin, Tzu-Hsien Yang, I-Ching Tseng, Chia-Lin Yang, Han-Wen Hu, Hung-Sheng Chang, Hsiang-Pang Li, and Meng-Fan Chang. DL-RSIM: A simulation framework to enable reliable ReRAM-based accelerators for deep learning. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8. IEEE, 2018.

[329] Hiroki Konishi, Rui Yamaguchi, Kiyoshi Yamaguchi, Yoichi Furukawa, and Seiya Imoto. Halcyon: an accurate basecaller exploiting an encoder–decoder model with monotonic attention. *Bioinformatics*, 37(9):1211–1217, 2021.

[330] Wonjoo Kim, Anupam Chattopadhyay, Anne Siemon, Eike Linn, Rainer Waser, and Vikas Rana. Multi-state memristive tantalum oxide devices for ternary arithmetic. *Scientific reports*, 6(1):1–9, 2016.

[331] Synopsys, Inc. Synopsys Design Compiler. https://www.synopsys.com/support/training/rtl-synthesis/design-compiler-rtl-synthesis.html.

[332] Satyabrata Sarangi and Bevan Baas. DeepScaleTool: A Tool for the Accurate Estimation of Technology Scaling in the Deep-Submicron Era. In *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, 2021.

[333] MNEMOSENE partners. The MNEMOSENE project. http://www.mnemosene.eu/, 2020.

[334] AMD. AMD$^{\delta}$ EPYC$^{\delta}$ 7742 CPU. https://www.amd.com/en/products/cpu/amd-epyc-7742.

[335] NVIDIA. NVIDIA V 100. https://www.nvidia.com/en-us/data-center/v100/.

[336] Charlene Yang. Hierarchical roofline analysis: How to collect data using performance tools on intel CPUs and NVIDIA GPUs. *arXiv preprint arXiv:2009.02449*, 2020.

[337] Wick, Ryan. Raw fast5s. https://bridges.monash.edu/articles/dataset/Raw_fast5s/7676174.

[338] Wick, Ryan. Reference genomes. https://bridges.monash.edu/articles/dataset/Reference_genomes/7676135.

[339] Shubham Jain and Anand Raghunathan. Cxdnn: Hardware-software compensation methods for deep neural networks on resistive crossbar systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 18(6):1–23, 2019.

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

[341] Markus Fritscher, Johannes Knödtel, Maen Mallah, Stefan Pechmann, Emilio Perez-Bosch Quesada, Tommaso Rizzi, Christian Wenger, and Marc Reichenbach. Mitigating the Effects of RRAM Process Variation on the Accuracy of Artificial Neural Networks. In *International Conference on Embedded Computer Systems*, pages 401–417. Springer, 2022.

[342] Mengshi Zhang, Yuqun Zhang, Lingming Zhang, Cong Liu, and Sarfraz Khurshid. DeepRoad: GAN-based Metamorphic Testing and Input Validation Framework for Autonomous Driving Systems. In *2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 132–142, 2018.

[343] Jelena Kocić, Nenad Jovičić, and Vujo Drndarević. An End-to-End Deep Neural Network for Autonomous Driving Designed for Embedded Automotive Platforms. *Sensors*, 19(9), 2019.

[344] Zhong Li, Minxue Pan, Tian Zhang, and Xuandong Li. Testing DNN-based Autonomous Driving Systems under Critical Environmental Conditions. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 6471–6482. PMLR, 18–24 Jul 2021.

[345] Lerong Chen, Jiawen Li, Yiran Chen, Qiuping Deng, Jiyuan Shen, Xiaoyao Liang, and Li Jiang. Accelerator-friendly neural-network training: Learning variations and defects in RRAM crossbar. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pages 19–24. IEEE, 2017.

[346] Mohammed Alser, Hasan Hassan, Hongyi Xin, Oğuz Ergin, Onur Mutlu, and Can Alkan. GateKeeper: a new hardware architecture for accelerating pre-alignment in DNA short read mapping. *Bioinformatics*, 33(21):3355–3363, 05 2017.

[347] Genome Reference Consortium. Human reference genome GRCh38.p14. https://www.ncbi.nlm.nih.gov/assembly?term=GRCh38&cmd=DetailsSearch.

[348] Unknown. Homo sapiens (human). https://www.ebi.ac.uk/ena/data/view/ERR240727.

[349] Faraz Hach, Fereydoun Hormozdiari, Can Alkan, Farhad Hormozdiari, Inanc Birol, Evan E Eichler, and S Cenk Sahinalp. mrsFAST: a cache-oblivious algorithm for short-read mapping. *Nature methods*, 7(8):576–577, 2010.

[350] Martin Šošić and Mile Šikić. Edlib: A C/C++ Library for Fast, Exact Sequence Alignment Using Edit Distance. *Bioinformatics*, 33(9):1394–1395, 2017.

[351] D. R. Bentley, Shankar Balasubramanian, Harold P Swerdlow, Geoffrey Paul Smith, J Milton, Clive Brown, Kevin P Hall, Dirk J. Evers, Colin L. Barnes, Helen Bignell, Jonathan Mark Boutell, Jason Bryant, Richard J. Carter, R. Keira Cheetham, Anthony J. Cox, Darren J. Ellis, Michael R. Flatbush, Niall A. Gormley, Sean J. Humphray, Leslie J. Irving, Mirian Karbelashvili, Scott M. Kirk, Heng Li, Xiaohai Liu, Klaus Maisinger, Lisa J. Murray, Bojana Obradovic, Tobias William Barr Ost, Michael L. Parkinson, Mark Pratt, Isabelle Rasolonjatovo, Mark T. Reed, Roberto Rigatti, Chiara Rodighiero, Mark T. Ross, Andrea Sabot, Subramanian V. Sankar, Aylwyn Scally, Gary P. Schroth, Mark E. B. Smith, Vincent P. Smith, Anastassia Spiridou, Peta E. Torrance, Svilen Tzonev, Eric H. Vermaas, Klaudia Walter, Xiaolin Wu, Lu Zhang, Mohammed D. Alam, Carole Anastasi, I. N. C. Aniebo, David M. D. Bailey, Iain Bancarz, Saibal Banerjee, Selena G. Barbour, Primo A. Baybayan, Vincent Benoit, Kevin Benson, Claire Bevis, Phillip J. Black, Asha Boodhun, J. S. Brennan, John A. Bridgham, Rob C. Brown, Andrew Anand Brown, Dale H. Buermann, Abass A. Bundu, James C. Burrows, Nigel P. Carter, Nestor Castillo, Maria Chiara E. Catenazzi, Simon Chang, R. Neil Cooley, Natasha R. Crake, Olubunmi O. Dada, Konstantinos D. Diakoumakos, Belen Dominguez-Fernandez, David J. Earnshaw, Ugonna C. Egbujor, Dave Elmore, Sergey Etchin, Mark Ewan, Milan Fedurco, Louise J. Fraser, Karin V Fuentes Fajardo, W. Scott Furey, Dave George, Kimberley J. Gietzen, Colin P. Goddard, George Golda, Philip A. Granieri, David E. Green, David L. Gustafson, Nancy F. Hansen, Kevin Harnish, Christian D. Haudenschild, Narinder I. Heyer, Matthew M. Hims, Johnny T. Ho, Adrian Horgan,

Katya Hoschler, Steve Hurwitz, D. Vlasenko K. Ivanov, Maria Q. Johnson, Terena James, Thomas A. Jones, Gyoung-Dong Kang, Tzvetana H. Kerelska, Alan D. Kersey, Irina Khrebtukova, Alex Kindwall, Zoya Kingsbury, Paula I. Kokko-Gonzales, Anil Kumar, Marc Laurent, Cynthia Taylor Lawley, Sarah E. Lee, Xavier Lee, Arnold Liao, Jennifer A. Loch, Mitch Lok, Shujun J Luo, Radhika M. Mammen, John Wesley Martin, Patrick G. McCauley, Paul McNitt, Parul Ben D Mehta, Keith Moon, Joe W. Mullens, Taksina Newington, Zemin Ning, Bee Ling Ng, Sonia M. Novo, Michael J. O'neill, Mark A. Osborne, Andrew P. Osnowski, Omead Ostadan, Lambros L. Paraschos, Lea Pickering, Andrew C. Pike, Alger C. Pike, D. Chris Pinkard, Daniel P. Pliskin, Joe Podhasky, Victor J. Quijano, Come Raczy, Vicki H. Rae, Stephen R. Rawlings, Ana Chiva Rodriguez, Phyllida Roe, J. Rogers, M. Candelaria Rogert Bacigalupo, Nikolai Romanov, Anthony Romieu, Rithy K. Roth, Natalie J. Rourke, Silke T. Ruediger, Eli Rusman, Raquel M. Sanches-Kuiper, Martin R. Schenker, Jose Miguel Seoane, Richard Shaw, Mitch K. Shiver, Steven W. Short, Ning Leung Sizto, Johannes P. Sluis, Melanie Ann Smith, Jean-Ernest Sohna Sohna, Eric Spence, K. Stevens, Neil W. Sutton, Lukasz Szajkowski, Carolyn L. Tregidgo, Gerardo Turcatti, Stephanie vandeVondele, Yuli Verhovsky, Selene M. Virk, Suzanne Wakelin, Gregory C. Walcott, Jingwen Wang, Graham J. Worsley, Juying Yan, Ling Yau, Mike Zuerlein, Jane Rogers, James C. Mullikin, Matthew E. Hurles, Nick J. McCooke, John S. West, Frank L. Oaks, Peter L. Lundberg, David Klenerman, Richard Durbin, and Anthony J. Smith. Accurate Whole Human Genome Sequencing using Reversible Terminator Chemistry. *Nature*, 456:53 – 59, 2008.

[352] Shihui Yin, Zhewei Jiang, Jae-Sun Seo, and Mingoo Seok. XNOR-SRAM: In-Memory Computing SRAM Macro for Binary/Ternary Deep Neural Networks. *IEEE Journal of Solid-State Circuits*, 55(6):1733–1743, 2020.

[353] Zahid Ullah, Manish K. Jaiswal, and Ray C. C. Cheung. Z-TCAM: An SRAM-based Architecture for TCAM. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 23(2):402–406, 2015.

[354] Taha Shahroodi, Michael Miao, Joel Lindegger, Stephan Wong, Onur Mutlu, and Said Hamdioui. An in-memory architecture for high-performance long-read pre-alignment filtering. *arXiv preprint arXiv:2310.15634*, 2023.

[355] Michael A Quail, Iwanka Kozarewa, Frances Smith, Aylwyn Scally, Philip J Stephens, Richard Durbin, Harold Swerdlow, and Daniel J Turner. A large genome center's improvements to the Illumina sequencing system. *Nature methods*, 5(12):1005–1010, 2008.

[356] Bo Segerman. The most frequently used sequencing technologies and assembly methods in different time segments of the bacterial surveillance and RefSeq genome databases. *Frontiers in cellular and infection microbiology*, 10:527102, 2020.

[357] Edward J Fox, Kate S Reid-Bayliss, Mary J Emond, and Lawrence A Loeb. Accuracy of next generation sequencing platforms. *Next generation, sequencing & applications*, 1, 2014.

[358] Anthony Rhoads and Kin Fai Au. PacBio sequencing and its applications. *Genomics, proteomics & bioinformatics*, 13(5):278–289, 2015.

[359] Yukiteru Ono, Kiyoshi Asai, and Michiaki Hamada. Pbsim: Pacbio reads simulator—toward accurate genome assembly. *Bioinformatics*, 29(1):119–121, 2013.

[360] Taishan Hu, Nilesh Chitnis, Dimitri Monos, and Anh Dinh. Next-generation sequencing technologies: An overview. *Human Immunology*, 82(11):801–811, 2021.

[361] Miten Jain, Hugh E Olsen, Benedict Paten, and Mark Akeson. The Oxford Nanopore MinION: delivery of nanopore sequencing to the genomics community. *Genome biology*, 17(1):1–11, 2016.

[362] M. Amin, S. Skiena, and M. C. Schatz. NanoBLASTer: Fast alignment and characterization of Oxford Nanopore single molecule sequencing reads. In *2016 IEEE 6th International Conference on Computational Advances in Bio and Medical Sciences (ICCABS)*, pages 1–6, Los Alamitos, CA, USA, oct 2016. IEEE Computer Society.

[363] Ehsan Haghshenas, Süleyman Cenk Sahinalp, and Faraz Hach. lordFAST: sensitive and Fast Alignment Search Tool for LOng noisy Read sequencing Data. *Bioinformatics*, 35:20–27, 2018.

[364] Mark Chaisson and Glenn Tesler. Mapping single molecule sequencing reads using Basic Local Alignment with Successive Refinement (BLASR): Theory and Application. *BMC bioinformatics*, 13:238, 09 2012.

[365] Gregory G. Faust and Ira M. Hall. YAHA: fast and flexible long-read alignment with optimal breakpoint detection. *Bioinformatics*, 28(19):2417–2424, 07 2012.

[366] Bo Liu, Dengfeng Guan, Mingxiang Teng, and Yadong Wang. rHAT: fast alignment of noisy long reads with regional hashing. *Bioinformatics*, 32(11):1625–1631, 11 2015.

[367] Ivan Sović, Mile Šikić, Andreas Wilm, Shannon Nicole Fenlon, Swaine Chen, and Niranjan Nagarajan. Fast and sensitive mapping of nanopore sequencing reads with GraphMap. *Nature Communications*, 7, 2016.

[368] Nika Mansouri Ghiasi, Jisung Park, Harun Mustafa, Jeremie Kim, Ataberk Olgun, Arvid Gollwitzer, Damla Senol Cali, Can Firtina, Haiyu Mao, Nour Almadhoun Alserr, Rachata Ausavarungnirun, Nandita Vijaykumar, Mohammed Alser, and Onur Mutlu. GenStore: A High-Performance in-Storage Processing System for Genome Sequence Analysis. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '22, page 635–654, New York, NY, USA, 2022. Association for Computing Machinery.

[369] Haiyu Mao, Mohammed Alser, Mohammad Sadrosadati, Can Firtina, Akanksha Baranwal, Damla Senol Cali, Aditya Manglik, Nour Almadhoun Alserr, and Onur Mutlu. GenPIP: In-Memory Acceleration of Genome Analysis via Tight Integration of Basecalling and Read Mapping. In *MICRO-55: 55th Annual IEEE/ACM International Symposium on Microarchitecture*, 09 2022.

[370] Yufeng Gu, Arun Subramaniyan, Tim Dunn, Alireza Khadem, Kuan-Yu Chen, Somnath Paul, Md Vasimuddin, Sanchit Misra, David Blaauw, Satish Narayanasamy, and Reetuparna Das. GenDP: A Framework of Dynamic Programming Acceleration for Genome Sequencing Analysis. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*, ISCA '23, New York, NY, USA, 2023. Association for Computing Machinery.

[371] Yoongu Kim, Vivek Seshadri, Donghyuk Lee, Jamie Liu, and Onur Mutlu. A case for exploiting subarray-level parallelism (SALP) in DRAM. *Proceedings - International Symposium on Computer Architecture*, pages 368–379, 2012.

[372] Nastaran Hajinazar, Geraldo F. Oliveira, Sven Gregorio, João Dinis Ferreira, Nika Mansouri Ghiasi, Minesh Patel, Mohammed Alser, Saugata Ghose, Juan Gómez-Luna, and Onur Mutlu. SIMDRAM: A Framework for Bit-Serial SIMD Processing Using DRAM. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '21, page 329–345, New York, NY, USA, 2021. Association for Computing Machinery.

[373] Yukiteru Ono, Michiaki Hamada, and Kiyoshi Asai. PBSIM3: a simulator for all types of PacBio and ONT long reads. *NAR Genomics and Bioinformatics*, 4(4):lqac092, 12 2022.

[374] Ono, Yukiteru and Hamada, Michiaki and Asai, Kiyoshi. PBSIM3: a simulator for all types of PacBio and ONT long reads. https://github.com/yukiteruono/pbsim3, 2022.

[375] Heng Li. Minimap2: Pairwise alignment for nucleotide sequences. *Bioinformatics (Oxford, England)*, 34, 05 2018.

[376] Petr Danecek, James K Bonfield, Jennifer Liddle, John Marshall, Valeriu Ohan, Martin O Pollard, Andrew Whitwham, Thomas Keane, Shane A McCarthy, Robert M Davies, and Heng Li. Twelve years of SAMtools and BCFtools. *GigaScience*, 10(2), 02 2021. giab008.

[377] Alexa McIntyre, Rachid Ounit, Ebrahim Afshinnekoo, Robert Prill, Elizabeth Henaff, Noah Alexander, Samuel Minot, David Danko, Jonathan Foox, Sofia Ahsanuddin, Scott Tighe, Nur Hasan, Poorani Subramanian, Kelly Moffat, Shawn Levy, Stefano Lonardi, Nick Greenfield, Rita Colwell, Gail Rosen, and Christopher Mason. Comprehensive benchmarking and ensemble approaches for metagenomic classifiers. *Genome Biology*, 18:182, 09 2017.

[378] Florian P Breitwieser, Jennifer Lu, and Steven L Salzberg. A review of methods and databases for metagenomic classification and assembly. *Briefings in Bioinformatics*, 20(4):1125–1136, 09 2017.

[379] Lulu Ge and Keshab K. Parhi. Classification Using Hyperdimensional Computing: A Review. *IEEE Circuits and Systems Magazine*, 20:30–47, 2020.

[380] Denis Kleyko, Abbas Rahimi, Dmitri A. Rachkovskij, Evgeny Osipov, and Jan M. Rabaey. Classification and Recall With Binary Hyperdimensional Computing: Tradeoffs in Choice of Density and Mapping Characteristics. *IEEE Transactions on Neural Networks and Learning Systems*, 29(12):5880–5898, 2018.

[381] Intel. Intel VTune Amplifier 2019 User Guide. https://software.intel.com/en-us/vtune-amplifier-help, 2018.

[382] Minxuan Zhou, Lingxi Wu, Muzhou Li, Niema Moshiri, Kevin Skadron, and Tajana Simunic. Ultra Efficient Acceleration for De Novo Genome Assembly via Near-Memory Computing. *2021 30th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 199–212, 2021.

[383] Pentti Kanerva. Hyperdimensional Computing: An Introduction to Computing in Distributed Representation with High-Dimensional Random Vectors. *Cognitive Computation*, 1:139–159, 2009.

[384] Ross W. Gayler. Vector Symbolic Architectures answer Jackendoff's challenges for cognitive neuroscience. *ArXiv*, abs/cs/0412059, 2004.

[385] Dmitri A. Rachkovskij. Some approaches to analogical mapping with structure-sensitive distributed representations. *Journal of Experimental & Theoretical Artificial Intelligence*, 16(3):125–144, 2004.

[386] Serge Slipchenko and Dmitri Rachkovskij. Analogical mapping using similarity of binary distributed representations. *J. Information Theories and Applications*, 16:269–290, 01 2009.

[387] Dmitri Rachkovskij and Serge Slipchenko. Similarity-based retrieval with structure-sensitive sparse binary distributed representations. *Computational Intelligence*, 28:106 – 129, 03 2012.

[388] Fateme Rasti Najafabadi, Abbas Rahimi, Pentti Kanerva, and Jan M Rabaey. Hyperdimensional computing for text classification. In *Design, automation test in Europe conference exhibition (DATE), University Booth*, pages 1–1, 2016.

[389] Mohsen Imani, Abbas Rahimi, Deqian Kong, Tajana Rosing, and Jan M. Rabaey. Exploring Hyperdimensional Associative Memory. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 445–456, 2017.

[390] Denis Kleyko, Evgeny Osipov, Nikolaos Papakonstantinou, Valeriy Vyatkin, and Arash Mousavi. Fault detection in the hyperspace: Towards intelligent automation systems. In *2015 IEEE 13th International Conference on Industrial Informatics (INDIN)*, pages 1219–1224, 2015.

[391] Denis Kleyko, Evgeny Osipov, Ross W. Gayler, Asad I. Khan, and Adrian G. Dyer. Imitation of honey bees' concept learning processes using Vector Symbolic Architectures. *Biologically Inspired Cognitive Architectures*, 14:57–72, 2015.

[392] Abbas Rahimi, Simone Benatti, Pentti Kanerva, Luca Benini, and Jan M. Rabaey. Hyperdimensional biosignal processing: A case study for EMG-based hand gesture recognition. In *2016 IEEE International Conference on Rebooting Computing (ICRC)*, pages 1–8, 2016.

[393] P. Kanerva, J. Kristofersson, and A. Holst. Random Indexing of Text Samples for Latent Semantic Analysis. In L.R. Gleitman and A.K. Josh, editors, *Proceedings of the 22nd Annual Conference of the Cognitive Science Society*, volume 1036, Erlbaum, New Jersey, 2000.

[394] Behnam Khaleghi, Mohsen Imani, and Tajana Rosing. Prive-HD: Privacy-Preserved Hyperdimensional Computing. In *Proceedings of the 57th ACM/EDAC/IEEE Design Automation Conference*, DAC '20. IEEE Press, 2020.

[395] T.A. Plate. Holographic reduced representations. *IEEE Transactions on Neural Networks*, 6(3):623–641, 1995.

[396] Stephen I. Gallant and T. Wendy Okaywe. Representing Objects, Relations, and Sequences. *Neural Computation*, 25(8):2038–2078, 08 2013.

[397] Ross W. Gayler. Multiplicative Binding, Representation Operators & Analogy. In *Workshop Poster*, 1998.

[398] Stephen I. Gallant and Phil Culliton. Positional binding with distributed representations. In *2016 International Conference on Image, Vision and Computing (ICIVC)*, pages 108–113, 2016.

[399] D.A. Rachkovskij. Representation and processing of structures with binary sparse distributed codes. *IEEE Transactions on Knowledge and Data Engineering*, 13(2):261–276, 2001.

[400] Tony Plate. Holographic Reduced Representations. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 6:623–41, 02 1995.

[401] Dmitri A. Rachkovskij, Ernst M. Kussul, and Tatiana N. Baidyk. Building a world model with structure-sensitive sparse binary distributed representations. *Biologically Inspired Cognitive Architectures*, 3:64–86, 2013.

[402] Pentti Kanerva. *Sparse distributed memory*. MIT press, 1988.

[403] Denis Kleyko, Evgeny Osipov, Alexander Senior, Asad I. Khan, and Yaşar Ahmet Şekerciogğlu. Holographic Graph Neuron: A Bioinspired Architecture for Pattern Processing. *IEEE Transactions on Neural Networks and Learning Systems*, 28(6):1250–1262, 2017.

[404] Mohsen Imani, Saikishan Pampana, Saransh Gupta, Minxuan Zhou, Yeseong Kim, and Tajana Rosing. DUAL: Acceleration of Clustering Algorithms using Digital-based Processing In-Memory. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 356–371, 2020.

[405] Knut Reinert, Temesgen Hailemariam Dadi, Marcel Ehrhardt, Hannes Hauswedell, Svenja Mehringer, René Rahn, Jongkyu Kim, Christopher Pockrandt, Jörg Winkler, Enrico Siragusa, Gianvito Urgese, and David Weese. The SeqAn C++ template library for efficient sequence analysis: A resource for programmers. *Journal of Biotechnology*, 261:157–168, 2017. Bioinformatics Solutions for Big Data Analysis in Life Sciences presented by the German Network for Bioinformatics Infrastructure.

[406] Mark Harris et al. Optimizing parallel reduction in CUDA. *Nvidia developer technology*, 2(4):70, 2007.

[407] Nathan LaPierre, Serghei Mangul, Mohammed Alser, Igor Mandric, Nicholas Wu, David Koslicki, and Eleazar Eskin. MiCoP: microbial community profiling method for detecting viral and fungal organisms in metagenomic samples. *BMC Genomics*, 20:423, 06 2019.

[408] Project: PRJEB34001. Calibration sausages WGS containing mammalian and avian species. https://www.ebi.ac.uk/ena/browser/view/PRJEB34001.

[409] Project: PRJNA271645. Calibration sausages Metagenome. https://www.ebi.ac.uk/ena/browser/view/PRJNA271645.

[410] Miao Hu, Catherine Graves, Can Li, Yunning Li, Ning Ge, Eric Montgomery, Noraica Dávila, Hao Jiang, Stan Williams, Jianhua Joshua Yang, Qiangfei Xia, and John William Strachan. Memristor-Based Analog Computation and Neural Network Classification with a Dot Product Engine. *Advanced Materials*, 30, 01 2018.

[411] Mahdi Zahedi, Mahta Mayahinia, Muath Abu Lebdeh, Stephan Wong, and Said Hamdioui. Efficient Organization of Digital Periphery to Support Integer Datatype for Memristor-Based CIM. In *2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 216–221, 2020.

[412] Shinhyun Choi, Jong Hoon Shin, Jihang Lee, Patrick Sheridan, and Wei D. Lu. Experimental Demonstration of Feature Extraction and Dimensionality Reduction Using Memristor Networks. *Nano letters*, 17 5:3113–3118, 2017.

[413] Mahdi Zahedi, Remon van Duijnen, Stephan Wong, and Said Hamdioui. Tile Architecture and Hardware Implementation for Computation-in-Memory. In *2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 108–113, 2021.

[414] Gonzalo Diarce, Álvaro Celador, K. Martin, A. Urresti, Ane Garcia-Romero, and J.M. Sala. PA comparative study of the CFD modeling of a ventilated active fa ade including phase change materials. *Applied Energy*, 126:307–17, 08 2014.

[415] Simone Balatti, Stefano Ambrogio, Zhongqiang Wang, and Daniele Ielmini. True Random Number Generation by Variability of Resistive Switching in Oxide-Based Devices. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 5(2):214–221, 2015.

[416] Francesco Maria Puglisi°, Nicolò Zagni, Luca Larcher, and Paolo Pavan. A new verilog-A compact model of random telegraph noise in oxide-based RRAM for advanced circuit design. In *2017 47th European Solid-State Device Research Conference (ESSDERC)*, pages 204–207, 2017.

[417] R.G. Carvajal, J. Ramirez-Angula, and J. Tombs. High-speed high-precision voltage-mode MIN/MAX circuits in CMOS technology. In *2000 IEEE International Symposium on Circuits and Systems (ISCAS)*, volume 5, pages 13–16 vol.5, 2000.

[418] Junwhan Ahn, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. PIM-enabled instructions: A low-overhead, locality-aware processing-in-memory architecture. In *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, pages 336–348, 2015.

[419] J. Picorel, D. Jevdjic, and B. Falsafi. Near-Memory Address Translation. In *2017 26th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 303–317, Los Alamitos, CA, USA, sep 2017. IEEE Computer Society.

[420] ARM Holdings. Cortex-A8 Technical: Reference Manual, 2010.

[421] Intel Intel. and IA-32 architectures software developer's manual. *Volume 3A: System Programming Guide, Part*, 1(64):64, 64.

[422] Amirali Boroumand, Saugata Ghose, Minesh Patel, Hasan Hassan, Brandon Lucia, Kevin Hsieh, Krishna T. Malladi, Hongzhong Zheng, and Onur Mutlu. LazyPIM: An Efficient Cache Coherence Mechanism for Processing-in-Memory. *IEEE Computer Architecture Letters*, 16(1):46–50, 2017.

[423] Amirali Boroumand, Saugata Ghose, Minesh Patel, Hasan Hassan, Brandon Lucia, Rachata Ausavarungnirun, Kevin Hsieh, Nastaran Hajinazar, Krishna T. Malladi, Hongzhong Zheng, and Onur Mutlu. CoNDA: Efficient Cache Coherence Support for near-Data Accelerators. In *Proceedings of the 46th International Symposium on Computer Architecture*, ISCA '19, page 629–642, New York, NY, USA, 2019. Association for Computing Machinery.

[424] Daichi Fujiki, Scott Mahlke, and Reetuparna Das. Duality Cache for Data Parallel Acceleration. In *Proceedings of the 46th International Symposium on Computer Architecture*, ISCA '19, page 397–410, New York, NY, USA, 2019. Association for Computing Machinery.

[425] Ivan Merelli, Lucia Morganti, Elena Corni, Carmelo Pellegrino, Daniele Cesini, Luca Roverelli, Gabriele Zereik, and Daniele D'Agostino. Low-power portable devices for metagenomics analysis: Fog computing makes bioinformatics ready for the Internet of Things. *Future Generation Computer Systems*, 88:467–478, 2018.

[426] Daniele D'Agostino, Lucia Morganti, Elena Corni, Daniele Cesini, and Ivan Merelli. Combining Edge and Cloud computing for low-power, cost-effective metagenomics analysis. *Future Generation Computer Systems*, 90:79–85, 2019.

[427] Özge Eyice, Motonobu Namura, Yin Chen, Andrew Mead, Siva Samavedam, and Hendrik Schäfer. SIP metagenomics identifies uncultivated Methylophilaceae as dimethylsulphide degrading bacteria in soil and lake sediment. *The ISME journal*, 9(11):2336—2348, November 2015.

[428] Subho Sankar Banerjee, Mohamed El-Hadedy, Jong Bin Lim, Zbigniew T. Kalbarczyk, Deming Chen, Steven S. Lumetta, and Ravishankar K. Iyer. ASAP: Accelerated Short-Read Alignment on Programmable Hardware. *IEEE Trans. Comput.*, 68(3):331–346, mar 2019.

[429] Xia Fei, Zou Dan, Lu Lina, Man Xin, and Zhang Chunlei. FPGASW: Accelerating Large-Scale Smith–Waterman Sequence Alignment Application with Backtracking on FPGA Linear Systolic Array. *Interdisciplinary Sciences: Computational Life Sciences*, 10, 04 2017.

[430] Robin Kobus, Christian Hundt, André Müller, and Bertil Schmidt. Accelerating metagenomic read classification on CUDA-enabled GPUs. *BMC Bioinformatics*, 18, 01 2017.

[431] Derrick E Wood, Jennifer Lu, and Ben Langmead. Kraken 2 Open-Sourced Implementation. https://github.com/DerrickWood/kraken2, 2019.

[432] Qian Lou and Lei Jiang. BRAWL: A Spintronics-Based Portable Basecalling-in-Memory Architecture for Nanopore Genome Sequencing. *IEEE Computer Architecture Letters*, 17(2):241–244, 2018.

[433] D. Mayhew and V. Krishnan. PCI express and advanced switching: evolutionary path to building next generation interconnects. In *11th Symposium on High Performance Interconnects, 2003. Proceedings.*, pages 21–29, 2003.

[434] PCI-SIG. PCI-E Specification. https://pcisig.com/specifications.

[435] S. Rao, W. Kim, S. van Beek, S. Kundu, M. Perumkunnil, S. Cosemans, F. Yasin, S. Couet, R. Carpenter, B.J. O'Sullivan, S. H. Sharifi, N. Jossart, L. Souriau, L. Goux, D. Crotti, and G. S. Kar. STT-MRAM array performance improvement through optimization of Ion Beam Etch and MTJ for Last-Level Cache application. In *2021 IEEE International Memory Workshop (IMW)*, pages 1–4, 2021.

[436] Manu Komalan, Sushil Sakhare, Trong Huynh Bao, Siddharth Rao, Woojin Kim, Christian Tenllado, José Ignacio Gómez, Gouri Sankar Kar, Arnaud Furnemont, and Francky Catthoor. Cross-layer design and analysis of a low power, high density STT-MRAM for embedded systems. In *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4, 2017.

[437] Intel Corp. Intel® Performance Counter Monitor. https://www.intel.com/software/pcm, 2017.

[438] Marek Kokot, Maciej Długosz, and Sebastian Deorowicz. KMC 3: counting and manipulating k-mer statistics. *Bioinformatics (Oxford, England)*, 33, 01 2017.

[439] Ian Curtis. The Intel Skylake-X Review: Core i9 7900X, i7 7820X and i7 7800X Tested: Die Size Estimates and Arrangements. https://www.anandtech.com/show/11550/the-intel-skylakex-review-core-i9-7900x-i7-7820x-and-i7-7800x-tested/6, 2017.

[440] Lillian Pentecost, Alexander Hankin, Marco Donato, Mark Hempstead, Gu-Yeon Wei, and David Brooks. NVMExplorer: A Framework for Cross-Stack Comparisons of Embedded Non-Volatile Memories. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 938–956, 2022.

[441] Aaron Stillmaker and Bevan Baas. Scaling equations for the accurate prediction of CMOS device performance from 180nm to 7nm. *Integration*, 58:74–81, 2017.

[442] Yi-Fan Qin, Rui Kuang, Xiao-Di Huang, Yi Li, Jia Chen, and Xiang-Shui Miao. Design of High Robustness BNN Inference Accelerator Based on Binary Memristors. *IEEE Transactions on Electron Devices*, 67(8):3435–3441, 2020.

[443] Yiyang Zhao, Yongjia Wang, Ruibo Wang, Yuan Rong, and Xianyang Jiang. A Highly Robust Binary Neural Network Inference Accelerator Based on Binary Memristors. *Electronics*, 10(21), 2021.

[444] Tifenn Hirtzlin, Marc Bocquet, Bogdan Penkovsky, Jacques-Olivier Klein, Etienne Nowak, Elisa Vianello, Jean-Michel Portal, and Damien Querlioz. Digital biologically plausible implementation of binarized neural networks with differential hafnium oxide resistive memory arrays. *Frontiers in neuroscience*, 2020.

[445] Chung-Cheng Chou, Zheng-Jun Lin, Pei-Ling Tseng, Chih-Feng Li, Chih-Yang Chang, Wei-Chih Chen, Y. D. Chih, and Tsung-Yung Jonathan Chang. An N40 256K×44 embedded RRAM macro with SL-precharge SA and low-voltage current limiter to improve read and write performance. *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*, pages 478–480, 2018.

[446] Ruizhou Ding, Zeye Liu, Rongye Shi, Diana Marculescu, and R.D. (Shawn) Blanton. LightNN: Filling the Gap between Conventional Deep Neural Networks and Binarized Networks. In *Proceedings of the on Great Lakes Symposium on VLSI 2017*, GLSVLSI '17, page 35–40, New York, NY, USA, 2017. Association for Computing Machinery.

[447] Stefano Ambrogio, Nicola Ciocchini, Mario Laudato, Valerio Milo, Agostino Pirovano, Paolo Fantini, and D. Ielmini. Unsupervised Learning by Spike Timing Dependent Plasticity in Phase Change Memory (PCM) Synapses. *Frontiers in Neuroscience*, 10, 03 2016.

[448] Qi Liu, Bin Gao, Peng Yao, Dong Wu, Junren Chen, Yachuan Pang, Wenqiang Zhang, Yan Liao, Cheng-Xin Xue, Wei-Hao Chen, Jianshi Tang, Yu Wang, Meng-Fan Chang, He Qian, and Huaqiang Wu. 33.2 a fully integrated analog reram based 78.4tops/w compute-in-memory chip with fully parallel mac computing. In *2020 IEEE International Solid- State Circuits Conference - (ISSCC)*, pages 500–502, 02 2020.

[449] Yann Lecun, Leon Bottou, Y. Bengio, and Patrick Haffner. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86:2278 – 2324, 12 1998.

[450] Alex Krizhevsky. Learning Multiple Layers of Features from Tiny Images. *University of Toronto*, 05 2012.

[451] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. *Advances in neural information processing systems*, 29, 2016.

[452] Pan Zhou, Weiguang Zhao, Jianghui Li, Ang Li, Wei Du, and Shiping Wen. Massive Maritime Path Planning: A Contextual Online Learning Approach. *IEEE Transactions on Cybernetics*, 51(12):6262–6273, 2021.

[453] Gang Chen, Shengyu He, Haitao Meng, and Kai Huang. PhoneBit: Efficient Gpu-Accelerated Binary Neural Network Inference Engine for Mobile Phones. In *Proceedings of the 23rd Conference on Design, Automation and Test in Europe*, DATE '20, page 786–791, San Jose, CA, USA, 2020. EDA Consortium.

[454] Eriko Nurvitadhi, David Sheffield, Jaewoong Sim, Asit Mishra, Ganesh Venkatesh, and Debbie Marr. Accelerating Binarized Neural Networks: Comparison of FPGA, CPU, GPU, and ASIC. In *2016 International Conference on Field-Programmable Technology (FPT)*, pages 77–84, 2016.

[455] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, and Olivier Temam. Dadiannao: A machine-learning supercomputer. *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 609–622, 2014.

[456] Joao Ambrosi, Aayush Ankit, Rodrigo Antunes, Sai Rahul Chalamalasetti, Soumitra Chatterjee, Izzat El Hajj, Guilherme Fachini, Paolo Faraboschi, Martin Foltin, Sitao Huang, Wen-Mei Hwu, Gustavo Knuppe, Sunil Vishwanathpur Lakshminarasimha, Dejan Milojicic, Mohan Parthasarathy, Filipe Ribeiro, Lucas Rosa, Kaushik Roy, Plinio Silveira, and John Paul Strachan. Hardware-Software Co-Design for an Analog-Digital Accelerator for Machine Learning. In *2018 IEEE International Conference on Rebooting Computing (ICRC)*, pages 1–13, 11 2018.

[457] Raphael Cardoso, Lubna Arif, Clément Zrounba, Fabio Pavanello, Ian O'Connor, Laurent Vivien, Léopold Virot, and Sébastien Le Beux. Energy efficient on-chip optical broadcast with partial-absorption photodiodes. In *2022 20th IEEE Interregional NEWCAS Conference (NEWCAS)*, pages 198–202, 2022.

[458] Sajjad Moazeni, Sen Lin, Mark Wade, Luca Alloatti, Rajeev J. Ram, Miloš Popović, and Vladimir Stojanović. A 40-Gb/s PAM-4 Transmitter Based on a Ring-Resonator Optical DAC in 45-nm SOI CMOS. *IEEE Journal of Solid-State Circuits*, 52(12):3503–3516, 2017.

[459] Aditya Narayan, Yvain Thonnart, Pascal Vivet, and Kaan Coşkun. PROWAVES: Proactive Runtime Wavelength Selection for Energy-efficient Photonic NoCs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, PP:1–1, 11 2020.

[460] Masaki Wada, Taiji Sakamoto, Shinichi Aozasa, Ryota Imada, Takashi Yamamoto, and Kazuhide Nakajima. Full C-Band and Power Efficient Coupled-Multi-Core Fiber Amplifier. In *2020 Optical Fiber Communications Conference and Exhibition (OFC)*, pages 1–3, 01 2020.

[461] Ankit, Aayush and Silveira, Plinio and Aguiar, Glaucimar. PUMA Simulator. https://github.com/Aayush-Ankit/puma-simulator, 2019.

[462] Clément Zrounba, Sebastien Cueff, Sébastien Le Beux, Ian O'Connor, and Fabio Pavanello. Exploration of the optical behavior of phase-change materials integrated in silicon photonics platforms. In *2021 Conference on Lasers and Electro-Optics Europe & European Quantum Electronics Conference (CLEO/Europe-EQEC)*, pages 1–1, 2021.

[463] Robert Polster, Yvain Thonnart, Guillaume Waltener, Jose-Luis Gonzalez, and Eric Cassan. Efficiency Optimization of Silicon Photonic Links in 65-nm CMOS and 28-nm FDSOI Technology Nodes. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 24:1–10, 12 2016.

[464] Hanyu Zhang, Linjie Zhou, Liangjun Lu, Jian Xu, Ningning Wang, Hao Hu, B. M. A. Rahman, Zhiping Zhou, and Jianping Chen. Miniature Multi-Level Optical Memristive Switch Using Phase Change Material. *arXiv e-prints*, page arXiv:1905.03163, May 2019.

[465] Congming Gao, Xin Xin, Youyou Lu, Youtao Zhang, Jun Yang, and Jiwu Shu. ParaBit: Processing Parallel Bitwise Operations in NAND Flash Memory Based SSDs. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '21, page 59–70, New York, NY, USA, 2021. Association for Computing Machinery.

[466] Yu Cai, Erich F. Haratsch, Onur Mutlu, and Ken Mai. Error Patterns in MLC NAND Flash Memory: Measurement, Characterization, and Analysis. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '12, page 521–526, San Jose, CA, USA, 2012. EDA Consortium.

[467] H.-S. Philip Wong, Simone Raoux, SangBum Kim, Jiale Liang, John P. Reifenberg, Bipin Rajendran, Mehdi Asheghi, and Kenneth E. Goodson. Phase Change Memory. *Proceedings of the IEEE*, 98(12):2201–2227, 2010.

[468] Jun Yang and Bo Zhao. Improving phase change memory (pcm) and spin-torque-transfer magnetic-ram (stt-mram) as next-generation memories: a circuit perspective. 2013.

[469] Dmytro Apalkov, Alexey Khvalkovskiy, Steven Watts, Vladimir Nikitin, Xueti Tang, Daniel Lottis, Kiseok Moon, Xiao Luo, Eugene Chen, Adrian Ong, Alexander Driskill-Smith, and Mohamad Krounbi. Spin-Transfer Torque Magnetic Random Access Memory (STT-MRAM). *J. Emerg. Technol. Comput. Syst.*, 9(2), may 2013.

[470] Stuart S. P. Parkin and See-Hun Yang. Memory on the racetrack. *Nature nanotechnology*, 10 3:195–8, 2015.

[471] Abdullah Giray Yağlıkçı, Minesh Patel, Jeremie S. Kim, Roknoddin Azizi, Ataberk Olgun, Lois Orosa, Hasan Hassan, Jisung Park, Konstantinos Kanellopoulos, Taha Shahroodi, Saugata Ghose, and Onur Mutlu. BlockHammer: Preventing RowHammer at Low Cost by Blacklisting Rapidly-Accessed DRAM Rows. *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 345–358, 2021.

[472] Jawad Haj-Yahya, Jisung Park, Rahul Bera, Juan Gómez Luna, Efraim Rotem, Taha Shahroodi, Jeremie Kim, and Onur Mutlu. BurstLink: Techniques for Energy-Efficient Video Display for Conventional and Virtual Reality Systems. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '21, page 155–169, New York, NY, USA, 2021. Association for Computing Machinery.

[473] Rahul Bera, Konstantinos Kanellopoulos, Anant Nori, Taha Shahroodi, Sreenivas Subramoney, and Onur Mutlu. Pythia: A Customizable Hardware Prefetching Framework Using Online Reinforcement Learning. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '21, page 1121–1137, New York, NY, USA, 2021. Association for Computing Machinery.

[474] Minesh Patel, Taha Shahroodi, Aditya Manglik, A Giray Yaglikci, Ataberk Olgun, Haocong Luo, and Onur Mutlu. A case for transparent reliability in DRAM systems. *arXiv preprint arXiv:2204.10378*, 2022.

[475] Can Firtina, Jisung Park, Mohammed H. Alser, Jeremie S. Kim, Damla Senol Cali, Taha Shahroodi, Nika Mansouri Ghiasi, Gagandeep Singh, Konstantinos Kanellopoulos, Can Alkan, and Onur Mutlu. BLEND: a fast, memory-efficient and accurate mechanism to find fuzzy seed matches in genome analysis. *NAR Genomics and Bioinformatics*, 5, 2021.

[476] Can Firtina, Kamlesh R. Pillai, Gurpreet Singh Kalsi, Bharathwaj Suresh, Damla Senol Cali, Jeremie S. Kim, Taha Shahroodi, Meryem Banu Cavlak, Joel Lindegger, Mohammed H. Alser, Juan Gómez-Luna, Sreenivas Subramoney, and Onur Cezmi Mutlu. Aphmm: Accelerating profile hidden markov models for fast and energy-efficient genome analysis. *ArXiv*, abs/2207.09765, 2022.

[477] Mahdi Zahedi, Taha Shahroodi, Stephan Wong, and Said Hamdioui. Bcim: Efficient implementation of binary neural network based on computation in memory, 2022.

[478] Asif Ali Khan, Fazal Hameed, Taha Shahroodi, Alex K. Jones, and Jeronimo Castrillon. Efficient Memory Layout for Pre-Alignment Filtering of Long DNA Reads Using Racetrack Memory. *IEEE Computer Architecture Letters*, pages 1–4, 2024.

Curriculum Vitæ

Taha (Michael) Shahroodi

01-07-1995	Born in Tehran, Iran.

Education

Sep. 2018	B.Sc. degree in Computer Engineering Sharif University of Technology (SUT), Tehran, Iran
Oct. 2020	M.Sc. degree in Computer Science Eidgenössische Technische Hochschule Zürich (ETH Zürich) *Thesis:* Investigating the Opportunities for Improving the Efficiency and Accuracy of Metagenomics Profiling *Promotor:* Onur Mutlu
Mar. 2024	Ph.D. degree in Computer Science and Engineering Technische Universiteit Delft (TU Delft), Delft, The Netherlands *Thesis:* Computation-in-Memory for Modern Applications *Promotors:* Stephan Wong, Said Hamdioui

Main Publications

9. **T. Shahroodi**, M. Miao, J. Lindegger, S. Wong, O. Mutlu, and S. Hamdioui, *FilterFuse: A Computation-In-Memory Architecture for High-Performance Long-Read Pre-Alignment Filtering*, Submitted for publication to International Symposium on Computer Architecture (ISCA), 2024.

8. **T. Shahroodi**, R. Cardoso, S. Wong, A. Bosio, I. O'Connor, and S. Hamdioui, *High-Performance Data Mapping for BNNs on PCM-based Integrated Photonics*, Design, Automation & Test in Europe Conference & Exhibition (DATE), 2024.

7. **T. Shahroodi**, G. Singh, M. Zahedi, H. Mao, J. Lindegger, C. Firtina, S. Wong, O. Mutlu, and S. Hamdioui, *A Framework for Evaluating DNN-based Basecalling using Computation-In-Memory with Non-Ideal Memristors*, 56th IEEE/ACM International Symposium on Microarchitecture (MICRO), 2023.

6. **T. Shahroodi**, M. Miao, M. Zahedi, S. Wong, and S. Hamdioui, *SieveMem: A Computation-in-Memory Architecture for Fast and Accurate Pre-Alignment*, 34th IEEE International Conference on Application-specific Systems, Architectures, and Processors (ASAP), 2023.

5. **T. Shahroodi**, M. Miao, M. Zahedi, S. Wong, and S. Hamdioui, *RattlesnakeJake: A Fast and Accurate Pre-Alignment Filter Suitable for Computation-in-Memory*, International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS), 2023.

4. **T. Shahroodi**, S. Wong, and S. Hamdioui, A Case for Genome Analysis Where Genomes Reside, International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS), 2023.

3. **T. Shahroodi**, R. Cardoso, M. Zahedi, S. Wong, A. Bosio, I. O'Connor, and S. Hamdioui, *Lightspeed Binary Neural Networks using Optical Phase-Change Materials*, Design, Automation & Test in Europe Conference & Exhibition (DATE), 2023.

2. **T. Shahroodi**, M. Zahedi, A. Singh, S. Wong, and S. Hamdioui, *KrakenOnMem: A Memristor-Augmented HW/SW Framework for Taxonomic Profiling*, 36th ACM International Conference on Supercomputing (ICS), 2022.

1. **T. Shahroodi**, M. Zahedi, C. Firtina, M. Alser, S. Wong, O. Mutlu, and S. Hamdioui, *Demeter: A Fast and Energy-Efficient Food Profiler using Hyperdimensional Computing in Memory*, IEEE Access, 2022.

Other Publications

13. A.A. Khan, F. Hameed, **T. Shahroodi**, A.K. Jones, J. Castrillon, *Efficient Memory Layout for Pre-alignment Filtering of Long DNA Reads Using Racetrack Memory*, IEEE Computer Architecture Letters (CAL), 2024.

12. C. Firtina, K. Pillai, G.S. Kalsi, B. Suresh, D.S. Cali, J. Kim, **T. Shahroodi**, M.B. Cavlak, J. Lindegger, M. Alser, J. Gómez Luna, S. Subramoney, and O. Mutlu, *ApHMM: Accelerating Profile Hidden Markov Models for Fast and Energy-Efficient Genome Analysis*, ACM Transactions on Architecture and Code Optimization (TACO), 2023.

11. M. Zahedi, **T. Shahroodi**, S. Wong, and S. Hamdioui, *BCIM: Efficient Implementation of Binary Neural Network Based on Computation in Memory*, Submitted for publication to IEEE Transactions on Emerging Topics in Computing (TETC), 2023.

10. M. Zahedi, G. Custers, **T. Shahroodi**, G. Gaydadjiev, S. Wong, and S. Hamdioui, *SparseMEM: Energy-efficient Design for In-memory Sparse-based Graph Processing*, Design, Automation & Test in Europe Conference & Exhibition (DATE), 2023.

9. M. Zahedi, **T. Shahroodi**, S. Wong, and S. Hamdioui, *Efficient Signed Arithmetic Multiplication on Memristor-Based CrossbarProcessing*, IEEE Access 11, 2023.

8. C. Firtina, J. Park, M. Alser, J.S. Kim, D.S. Cali, **T. Shahroodi**, N.M. Ghiasi, G. Singh, K. Kanellopoulos, C. Alkan, and O. Mutlu, *BLEND: A Fast, Memory-Efficient and Accurate Mechanism to Find Fuzzy Seed Matches in Genome Analysis*, NAR Genomics and Bioinformatics, 2023.

7. M. Patel, **T. Shahroodi**, A. Manglik, G. Yaglikci, A. Olgun, H. Luo, and O. Mutlu, *A Case for Transparent Reliability in DRAM systems*, Submitted to arXiv preprint arXiv:2204.10378 (2022).

6. J.D. Ferreira, G. Falcao, J. Gómez Luna, M. Alser, L. Orosa, M. Sadrosadati, J.S. Kim, G.F. Oliveira, **T. Shahroodi**, A. Nori, and O. Mutlu, *pluto: Enabling Massively Parallel Computation in DRAM via Lookup Tables*, 55th IEEE/ACM International Symposium on Microarchitecture (MICRO), 2022.

5. M. Zahedi, **T. Shahroodi**, G. Custers, A. Singh, S. Wong, and S. Hamdioui, *System Design for Computation-in-Memory: From Primitive to Complex Functions*, IFIP/IEEE 30th International Conference on Very Large Scale Integration (VLSI-SoC), 2022.

4. A. Singh, M. Zahedi, **T. Shahroodi**, M. Gupta, A. Gebregiorgis, M. Komalan, R.V. Joshi, F. Catthoor, R. Bishnoi, and S. Hamdioui, *Cim-based Robust Logic Accelerator using 28 nm STT-MRAM Characterization Chip Tape-out*, IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS), 2022.

3. R. Bera, K. Konstantinos, A. Nori, **T. Shahroodi**, S. Sreenivas, and O. Mutlu, *Pythia: A Customizable Hardware Prefetching Framework using Online Reinforcement Learning*, 54th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2021.

2. J. Haj-Yahya, J. Park, R. Bera, G. Gómez Luna, E. Rotem, **T. Shahroodi**, J. Kim, and O. Mutlu, *BurstLink: Techniques for Energy-Efficient Video Display for Conventional and Virtual Reality Systems*, 54th Annual IEEE/ACM International Symposium on Microarchitecture, 2021.

1. A.G. Yağlikçi, M. Patel, J.S. Kim, R. Azizi, A. Olgun, L. Orosa, H. Hassan, J. Park, K. Kanellopoulos, **T. Shahroodi**, S. Ghose, and O. Mutlu, *BlockHammer: Preventing RowHammer at Low Cost by Blacklisting Rapidly-Accessed DRAM Rows*, IEEE International Symposium on High-Performance Computer Architecture (HPCA), 2021.