UC Santa Barbara

UC Santa Barbara Electronic Theses and Dissertations

Title

Towards Scalable High-Order Ising Machines with In-Memory Computing: From Algorithms and Architectures to Integrated Circuits

Permalink

https://escholarship.org/uc/item/1ht6683n

ISBN

9798276023656

Author

Bhattacharya, Tinish

Publication Date

2025-11-18

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

University of California
Santa Barbara

Towards Scalable High-Order Ising Machines with In-Memory Computing: From Algorithms and Architectures to Integrated Circuits

A dissertation submitted in partial satisfaction
of the requirements for the degree

Doctor of Philosophy
in
Electrical and Computer Engineering

by

Tinish Bhattacharya

Committee in charge:

Professor Dmitri Strukov, Chair
Professor Kerem Camsari
Professor Joao Hespanha
Professor John Paul Strachan, RWTH Aachen University, Forschungszentrum Juelich GmbH

December 2025

The Dissertation of Tinish Bhattacharya is approved.

Professor Kerem Camsari

Professor Joao Hespanha

Professor John Paul Strachan, RWTH Aachen University, Forschungszentrum Juelich GmbH

Professor Dmitri Strukov, Committee Chair

November 2025

Towards Scalable High-Order Ising Machines with In-Memory Computing: From Algorithms and Architectures to Integrated Circuits

Copyright © 2025

by

Tinish Bhattacharya

Dedicated to Mom, Papai, Chotai, Dia, Dadai, Thamma and Namrata

Acknowledgements

The past several years at UCSB have been an important phase of my life. One that has given me joy, sorrow, success, failure and lessons, that have ultimately shaped me as a person and researcher. During this period I have had the opportunity to work and play with some fantastic people and this is my humble attempt at thanking them for the positive influence they had on my research and my life.

I am grateful to my advisor Prof. Dmitri Strukov for the support and guidance that he provided throughout my doctoral studies. He allowed me to make mistakes, learn from them and then make a few more, for which I am grateful. For it is through our missteps that we learn our most valuable lessons. I thank him for always keeping his office door open for me to barge in and drag him into impromptu brainstorming sessions, to which he would never fail to oblige with enthusiasm, no matter what he was in the midst of. He taught me how to think out of the box, look past the smoke screen of low hanging fruits and strive for broader and impactful ideas. His kindness, creativity and unpretentious wit continues to humble and inspire me.

I am grateful to Prof. John Paul Strachan, whom I had the opportunity of collaborating with and also having as my dissertation committee member. The memristor crossbar test platform used to validate some of our ideas was previously developed under his supervision and I appreciate his guidance in interpreting and diagnosing the early results. I also thank him for his unassuming yet incisive feedback on my research, and for continually helping me broaden its scope. I would like to thank my committee member Prof. Kerem Camsari, for the insightful discussions and for always reminding me the bigger picture. I would like to thank Prof. Joao Hespanha for being in my committee and providing useful suggestions and advice.

I have been fortunate enough to work with some of the greatest scientists and re-

searchers around the world. I am thankful to Dr. Giacomo Pedretti and Dr. Thomas Van Vaerenbergh for being great mentors during my research internship at Hewlett Packard Labs, for the insightful discussions and fruitful collaboration. I am grateful for the discussions and opportunity to work with Dr. Arne Heittmann, Dr. Fabian Bohm, Dr. Ignacio Rozada, Dr. Xiangyi Zhang, and other members of the DARPA QuICC team. I am grateful to Dr. Raymond G. Beausoleil for spearheading the multi-institution DARPA QuICC collaboration and fostering an environment of innovation. His bold and forthright leadership has been inspiring, and offers much to learn from.

I want to thank Dr. Patrick H Mahan, Dr. Justin Lorentzen, Dr. Jill A Nolde, Dr. Edward H Aifer and Dr. Thomas J Walls of the Naval Research Laboratory for the insightful discussions and collaborative work as part of the in-sensor computing project.

I am grateful to my colleagues at Strukov Research Group, past and present, for their help and collaboration. I want to thank Dr. Shubham Pande and Sai Sukruth Bezugam for the late-night experiments and discussions during the neohebbian synapse project and Dr. Ewelina Wlazlak for fabricating the remarkable memristors. I thank George Higgins Hutchinson for collaborating on the several 30-day tapeout cycles (an observation only in retrospect), help with last-minute design signoffs and valuable discussions. I also thank Prof. Dongseok Kwon for the discussions and collaboration on the *gint1* chip. I want to thank Prof. Sanghyeon Choi and Daniel Espinosa for the discussions we shared and for being supportive colleagues. I am grateful to Dr. Mohammad Reza Mahmoodi for helping me resolve the numerous LVS and DRC issues during my first tapeout. Finally, I want to thank Prof. Shubham Sahay, who was a postdoctoral researcher in the group and the only person I personally knew when I first joined UCSB. He helped me find my footing when I was new to the country, introduced me to his friends, inspired me with his work, and entertained me with countless stories and gossip.

I would like to thank the Office of Naval Research, the U.S. Naval Research Labo-

ratory, NSF, DARPA, and all the taxpayers whose contributions directly or indirectly supported my research. I am grateful to the UC Santa Barbara, its faculty and exceptional staff, for providing the necessary infrastructure, excellent courses, and support whenever I needed it. I also thank all the student members of the union (UAW 4811) for their hard work, support, and advocacy in creating a more equitable environment.

I am deeply grateful to my friends in Santa Barbara, Colby, Ankit, Tanmay, Rahul, Jinesh, Aishwarya, Neha, Harsh, Ramya, Benu, and many others, who helped create a home away from home. I thank them for the many memories we shared, from mountain climbs, long road trips, cooking amazing dishes [1], driving lessons, hundred mile bike rides, car repairs, to the festive celebrations – for I shall always cherish them. I want to thank Sushant for always being there, for the six hour long video chats, and for keeping me sane throughout this journey. I want to thank him, Mayank, Shridu and all my friends from college for all the trips we took at home and abroad. I also want to thank my high school friends—especially Ishan, Rohit, Yash, Rudraneel, Arka, and others from our clan of extraordinary people who call ourselves *The Town*. I thank them for being part of this journey from the very beginning, back when we were preparing to join engineering college, for being there whenever I needed them, for the crazy exploits, and for the trips we took together[2].

I am deeply grateful to my parents for the immense sacrifices they made throughout their lives for me. They supported me through the highs and lows of life with unconditional love and in many ways are the true last authors [3] of the work that defines me. To Chotai (Kinjawl) for the selfless love and the countless exploits we shared together. To my grandparents dia, dadai, thamma, and to the Late Bimalendu Bhusan Bhattacharjee

[1] with me mostly on tasting duty

[2] that took a painstakingly long time to break state boundaries and finally become global

[3] In academic publishing especially in STEM fields, the last author is typically the senior author who provides overall direction, long-term vision, continual feedback and secures funding that enables the work

for their relentless support and unwavering faith in me. To Pomma, Nana, and the Late Sumona Chatterjee for their love and constant encouragement. To Fuchum, Amartya, Prakriti, Jhupu, Gullu, and all my cousins for filling my life with love, laughter, and warmth. I was fortunate to have Surjo, my cousin and a dear friend, living just two hours away in Los Angeles while also pursuing his PhD in Caltech. I am grateful to him for helping me stay sane as we both navigated the maze of doctoral studies, for the many trips we took together, and for our eternal pursuit of the green fairy.

Going back to my undergraduate days, I have been extremely lucky to find a friend and now my partner, Namrata who has been a part of this journey from the very beginning. I am grateful to her for being the most cheerful audience at my conference talks, for always supporting me through my darkest times, for exploring places around the world with me, for our long online conversations when we were thirteen time zones apart and for making the journey to finally be in the same time zone. I thank her for forgiving all the times I could not show up because of conference deadlines, and for always responding with an unwavering "*yes, whatever it takes*" to every "*is it possible*" question, in work and in life.

I am grateful to Dr. Ikbal Chowdhury for introducing me to the world of brain-inspired computing. Although I was already inclined toward scientific research, I am not sure I would have found my way into this field without his chance mention of it. I wish to thank my past advisor, Prof. Manan Suri, for giving me the opportunity to work in his lab, formally introducing me to research in unconventional computing, and training me in academic writing. I am grateful to him and to Prof. Arindam Basu for their mentorship and support during my undergraduate years. Lastly, I want to thank Prof. Mark Rodwell, Prof. Forrest Brewer, Prof. Mamidala Jagadesh Kumar, Prof. Basabi Bhaumik, Prof. Shouri Chatterjee, Supriyo Ghosh, Asimendu DasGupta and the countless other teachers whose teachings will stay with me for life.

"One should absorb the colour of life, but one should never remember its details. Details are always vulgar."

— Oscar Wilde, *The Picture of Dorian Gray*

"Maybe that's enlightenment enough: to know that there is no final resting place of the mind; no moment of smug clarity. Perhaps wisdom is realizing how small I am, and unwise, and how far I have yet to go."

— Anthony Bourdain

"The first principle is that you must not fool yourself and you are the easiest person to fool."

— Richard P. Feynman

"Thy right is to work only, but never to its fruits; let the fruit of action be not thy motive, nor let thy attachment be to inaction."

— Interpreted in english from the Bhagwad Gita, Chapter 2, Verse 47

Curriculum Vitæ

Tinish Bhattacharya

Education

2025	Ph.D. in Electrical and Computer Engineering (Expected), University of California, Santa Barbara.
2021	M.S. in Electrical and Computer Engineering, University of California, Santa Barbara.
2019	B. Tech. in Electrical Engineering, Indian Institute of Technology Delhi

Professional Employment

9/2019 - 12/2025	Graduate Student Researcher / Teaching Assistant, University of California Santa Barbara, CA.
7/2023 - 3/2024	Research Associate Intern, Hewlett Packard Enterprise, Milpitas, CA.
5/2018 - 7/2018	Research Intern, Nanyang Technological University, Singapore.

Honors and Awards

1. Recipient of the James V. & Beverly R. Zaleski Discovery Award of $94K for carrying out research in the field of integrated circuits for scalable optimization, awarded to a single student in the College of Engineering (COE) of UCSB in 2025-26.
2. ECE Dissertation Fellowship for $10K at UCSB in Spring 2025.
3. ECE Conference Travel Grant for $2.5K at UCSB in Spring 2025.
4. Academic Senate Doctoral Travel Grant for $1.5K at UCSB in 2025 (declined).
5. IEEE Computational Intelligence Society Travel Grant for IJCNN 2018, Rio de Janeiro, Brazil.
6. Design and Innovation Summer Award (DISA) 2016, IIT Delhi.
7. Recipient of the Institute Semester Merit Award for being in the top 7% of class, IIT Delhi, 2016.
8. Secured a national rank of 246 in IITJEE 2015 among 1.3 million students.
9. Awarded the KVPY Fellowship by the Department of Science & Technology, India, in the year 2014-2015 with a national rank of 125 among 50,000 students.

Selected Publications

1. **T. Bhattacharya**, G. H. Hutchinson, D. Kwon, D. Strukov. "PCIM-SAT: A 55nm Probabilistic K-SAT Solver with p-Bit-Based Parallel-Variable Update on a Mixed-Signal Compute-in-Memory Architecture," *2026 IEEE International Solid-State Circuits Conference (ISSCC)*, (2026). (accepted)
2. **T. Bhattacharya***, D. Kwon*, G. H. Hutchinson*, X. Zhang, I. Rozada and D. B. Strukov. "A Fully Integrated Mixed-Signal Compute-In-Memory Accelerator for Solving Arbitrary Order Boolean Satisfiability Problems". In *2025 IEEE Symposium on VLSI Technology and Circuits*, pp. 1-3, (2025).
3. **T. Bhattacharya**, D. Kwon, G. H. Hutchinson, X. Zhang, G. Pedretti, F. Bohm, J. P. Strachan, T. V. Vaerenbergh, R. Beausoleil, I. Rozada, D. Strukov. "KLIMA: Low-latency mixed-signal In-Memory Computing accelerator for solving arbitrary-order Boolean Satisfiability" *2025 IEEE Hot Chips 37 Symposium (HCS)*, pp. 1-7, (2025).
4. **T. Bhattacharya**, G. H. Hutchinson, G. Pedretti, X. Sheng, J. Ignowski, T. Van Vaerenbergh, R. Beausoleil, J. P. Strachan, and D. B. Strukov. "Computing High-Degree Polynomial Gradients in Memory" . *Nature Communications*, **15**, p. 8211, (2024).
5. **T. Bhattacharya***, S. S. Bezugam*, S. Pande*, E. Wlazlak, and D. Strukov, "ReRAM-Based NeoHebbian Synapses for Faster Training-Time-to-Accuracy Neuromorphic Hardware", In 2023 *International Electron Devices Meeting (IEDM)*, pp. 1-4. IEEE, (2023).
6. **T. Bhattacharya**, G. H. Hutchinson, G. Pedretti, and D. Strukov, "HO-FPIA: High-Order Field-Programmable Ising Arrays with In-Memory Computing", *2024 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 252-259. IEEE, (2024).
7. G. H. Hutchinson*, **T. Bhattacharya***, D. Kwon, F. Bohm, G. Pedretti, T. V. Vaerenbergh, A. Heitmann, J. P. Strachan, and D. B. Strukov. "Compressed High Order Ising Machine" *IEEE Journal on Emerging and Selected Topics in Circuits and Systems (JETCAS)*, (2025). (under review)
8. **T. Bhattacharya**, H. J. Kim and D. B. Strukov. "Memristor-Based Neuromorphic Readout Circuit for Event Cameras" *2025 59th Asilomar Conference on Signals, Systems, and Computers*, (2025).

9. G. Pedretti, F. Bohm, **T. Bhattacharya**, A. Heittmann, X. Zhang, M. Hizzani, G. H. Hutchinson, D. Kwon, J. Moon, E. Valiante, I. Rozada, C. E. Graves, J. Ignowski, M. Mohseni, J. P. Strachan, D. Strukov, R. Beausoleil, T. V. Vaerenbergh. "Solving Boolean satisfiability problems with resistive content addressable memories". *npj Unconventional Computing*, **2**, (2025).

10. G. H. Hutchinson, E. Sifferman, **T. Bhattacharya**, and D. Strukov, "FPIA: Field-Programmable Ising Arrays with In-Memory Computing", *2024 IEEEACM International Symposium on Low Power Electronics and Design (ISLPED)*, pp. 1-6, (2024).

11. S. S. Bezugam, **T. Bhattacharya**, H. Petschenig, S. Choi, G. Hutchinson, G. Pedretti, X. Sheng, J. Ignowski, T. Van Vaerenbergh, R. Beausoleil, R. Legenstein and D. Strukov "Controlling ReRAM's Switching Characteristics with Shadow Memory for Continual Learning", *2025 IEEE International Memory Workshop (IMW)*, Monterey, USA, 2025.

12. S. Choi, S. S. Bezugam, **T. Bhattacharya**, D. Kwon, D. Strukov."Wafer-scale fabrication of memristive passive crossbar circuits for brain-scale neuromorphic computing" . *Nature Communications*, **16**, p. 8757, (2025).

13. M. Hizzani, A. Heittmann, G. H. Hutchinson, D. Dobrynin, T. Vaerenbergh, **T. Bhattacharya**, A. Renaudineau, D. Strukov, J. P. Strachan. "Memristor-based hardware and algorithms for higher-order Hopfield optimization solver outperforming quadratic Ising machines." *2024 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1-5. IEEE, (2024).

14. G. Pedretti, F. Bohm, M. Hizzani, **T. Bhattacharya**, P. Bruel, J. Moon, S. Serebryakov et al. "Zeroth and higher-order logic with content addressable memories", In *2023 International Electron Devices Meeting (IEDM)*, pp. 1-4. IEEE, (2023).

15. **T. Bhattacharya**, S. Li, Y. Huang, W. Kang, W. Zhao & M. Suri, "Low-Power (1T1N) Skyrmionic Synapses for Spiking Neuromorphic Systems", *IEEE Access*, pp. 5034-5044. IEEE, (2018).

16. S. Dutta, **T. Bhattacharya**, N. R. Mohapatra, M. Suri, & U. Ganguly, "Transient Variability in SOI-Based LIF Neuron and Impact on Unsupervised Learning", *IEEE Transactions on Electron Devices (TED)*, 65(11), pp.5137-5144. IEEE, (2018).

17. M. Suri, **T. Bhattacharya**, and S. Chakraborty, "Experimental validation of cbram performance enhancement using fnw soft-technique", *2017 IEEE International Memory Workshop (IMW)*, Monterey, USA, 2017.

18. **T. Bhattacharya**, A. Gupta, S. Thoi, S. Roy and A. Prasad, "Robust Motion Artefact Resistant Circuit for Calculation of Mean Arterial Pressure from Pulse Transit Time", *2017 39th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, Jeju Island, South Korea, 2017.

19. S. Chakraborty, **T. Bhattacharya**, and M. Suri, "Current optimized coset coding for efficient rram programming", *IEEE Transactions on Very Large Scale Integration (TVLSI)*, 26(5), pp. 1000-1004. IEEE, (2018).

Service

- Journal reviewer for *Nature Communications* (2023, 2024), *Communications Engineering* (2024), *IEEE TCAS-I* (2024), *IEEE JXCDC* (2024 x2), *IEEE TED* (2025 x1), *IEEE TED* (2024 x2), *Journal of Physics D* (2019), *IEEE Access* (2020).
- Reviewer, *IEEE 2025 International Symposium on Circuits and Systems (ISCAS)*.
- Reviewer, *IEEE 2024 International Symposium on Circuits and Systems (ISCAS)*.

Abstract

Towards Scalable High-Order Ising Machines with In-Memory Computing: From Algorithms and Architectures to Integrated Circuits

by

Tinish Bhattacharya

Many of the hardest and most fascinating problems in science and engineering can be formulated as optimization tasks. Learning in biological systems and artificial neural networks, for example, is an optimization process, as are challenges in logistics, robotics, and biomedical applications that require finding exact or near-optimal ground states of discrete polynomial functions of arbitrary order. Many such problems are NP-Hard/Complete and remain computationally expensive to solve on conventional computers. This has driven the development of Ising machines that are hardware solvers that aim to find the absolute/approximate ground state of the Ising model, to which the objectives of the original optimization problem have been mapped.

Despite their success in solving optimizations with quadratic objective functions, most state-of-the-art Ising machines, being limited to natively implementing up to second-order interactions, are likely to incur considerable penalties in convergence times on higher-order optimization problems such as Boolean Satisfiability (K-SAT), Max-K-Cut and Max-K-SAT. This is due to an exponential increase in search space caused by the order reduction step that converts the high-order objective function to a quadratic one. Nevertheless, high-order problems are widespread in fields like electronic structure prediction, network routing, model verification, cryptography, and this necessitates the development of high-order Ising Machines. This dissertation attempts to address this need through advancements in algorithm, architecture, and integrated circuit design.

We present a new hardware paradigm that enables massively parallel computation of high-order gradients using mixed-signal in-memory computing circuits. It is shown for the first time, that a pair of back-to-back connected crossbar arrays with appropriate peripheral circuits can provide extremely fast solutions to high-order optimization problems with hardware complexity that scales linearly with the number of variables and terms in the objective function but independent of the order (K).

This approach is experimentally validated with an optimization chip (tuned to solving K-SAT), taped out in 55 nm CMOS technology. The integrated circuit features custom SRAM based in-memory computing arrays, analog-input Winner-Take-All circuits, and achieves one to two orders of magnitude speedup over prior solvers, despite implementing a legacy SAT-solving algorithm.

We introduce a new probabilistic SAT-solving algorithm that leverages the bipartite clause-variable relationship to update multiple variables per iteration, thereby improving navigational efficiency and outperforming several prior algorithms. A second chip demonstrating this algorithm is taped out in 55 nm CMOS technology. It improves upon the parallelization capabilities of the previous chip and incorporates dynamic comparator based Probabilistic-bits (p-bits) to eliminate the need for ADCs or analog-domain reduction circuits (a critical bottleneck in analog in-memory computing). The chip exhibits the fastest-to-date convergence times on small-scale K-SAT problems and outperforms prior hardware (including those designed in more advanced 28 nm) by 4-13 times.

Through hardware-aware simulations, we present a framework for implementing different flavors of gradient computing circuits in high-order ising machines, as well as extended circuit implementations for natively mapping and solving high-order XOR/parity-based constraints. Furthermore, we present a multi-tile architecture that enables scaling these solvers to handle large high-order optimization problems. The architecture consists of mixed-signal in-memory compute cores interconnected by a field programmable gate

array-inspired reconfigurable routing fabric. Analog computations are performed locally within each tile, whereas ADC-free peripheral circuits confine inter-tile communication to the digital domain.

Finally, an outlook to the future of such solvers is presented, in addition to progress and preliminary results on hardware architectures for accelerating complete SAT solvers and tape-out of a Floating-gate based analog in-memory computing macro for implementing high-order Ising machines with multi-bit weights.

Contents

Curriculum Vitae **x**

Abstract **xiv**

1 Introduction **1**
1.1 Combinatorial Optimization Problems 1
1.2 Ising Machines 4
1.3 In-Memory Computing 9
1.4 Thesis Outline 14
1.5 Permissions and Attributions 15

2 Computing High-Order Discrete Gradients In-Memory **17**
2.1 Background 17
2.2 Dual cross-coupled crossbar architecture 18
2.3 Experimental Demonstration with memristor crossbar 28
2.4 Impact of Crossbar Noise on Gradients 33
2.5 Hardware Modeling of Two High-Order Optimization Machines 39
2.6 Real-valued gradient computing hardware 46
2.7 Summary 52

3 A 55nm K-SAT Solver running WalkSAT Algorithm **54**
3.1 Background 54
3.2 Key Features 55
3.3 Architecture and Schematics 57
3.4 Measurement Results 62
3.5 Hybrid Mode Operation enabling other Algorithms 67
3.6 Summary and Outlook 73

4 A 55 nm K-SAT solver with probabilistic-bits based parallel variable update **75**
4.1 Background 75

4.2 Algorithm Design . . . 76
4.3 Key Features . . . 81
4.4 Architecture and Schematics . . . 83
4.5 Measurement Results . . . 89
4.6 Summary and Outlook . . . 91

5 Framework for High-Order Ising Machines with In-Memory Computing 94
5.1 Background . . . 94
5.2 Classification of gradient computing array . . . 96
5.3 Evaluating XOR constraints in-memory . . . 102
5.4 Modeling Results . . . 104
5.5 Summary and Outlook . . . 109

6 Towards multi-tile architectures and solving large-scale problems 111
6.1 Introduction . . . 111
6.2 Background . . . 115
6.3 High-Order Field Programmable Ising Array . . . 119
6.4 Results . . . 126
6.5 Summary and Outlook . . . 128

7 Outlook for High-Order Ising Machines 132
7.1 Summary of Contributions . . . 132
7.2 Preliminary results from ongoing work . . . 135
7.3 Future Directions . . . 139

A Supplementary Notes for Chapter 2 145
A.1 Partial derivatives and difference quotients . . . 145
A.2 Gradient computation in high-order polynomials . . . 147
A.3 Gradient computation in Conjunctive-Normal Form . . . 151
A.4 Single-step backward pass with three-terminal crossbar arrays . . . 157
A.5 3-SAT instance solved in the experimental demo . . . 161
A.6 3-SAT instance generation . . . 161
A.7 WalkSAT Algorithm . . . 163
A.8 Discrete-time High-Order Hopfield Neural Network . . . 164
A.9 Hardware and Algorithm Simulation Details . . . 165
A.10 Simulation Details of Real-Valued Gradient Computing . . . 168
A.11 Comparison with other approaches . . . 174

B Supplementary Notes for Chapter 5 178
B.1 Relation between PUBO and the Ising Model . . . 178
B.2 Relation between CNF and the Ising Model . . . 180

B.3 Relation between XNF and the Ising Model 183

C ReRAM-based NeoHebbain Synapse for fast online learning 186
C.1 Background 186
C.2 *E-Prop* Learning Algorithm 188
C.3 NeoHebbian Synapse 189
C.4 Application Modeling 196
C.5 Summary 199

D Memristor-Based Neuromorphic Readout Circuit for Event Cameras 200
D.1 Background 200
D.2 Results 203
D.3 Summary 208

Bibliography 209

Chapter 1

Introduction

1.1 Combinatorial Optimization Problems

Combinatorial optimization is a branch of mathematical optimization concerned with finding the best object from a finite set of objects, where the feasible objects are discrete or can be reduced to a discrete set. At the core of any such problem lies an objective function, which assigns a real-valued score to each feasible object, quantifying its quality or cost [1]. The goal of the optimization is to identify the object that either maximizes or minimizes this objective function, thereby achieving the best possible value according to the chosen criterion. If now one assumes that the object is a N-sized vector of variables where each variable can take one out of z different values, the set S of all possible objects is z^N. In the common case of binary-valued variables, the search space is 2^N. This simple fact highlights the exponential scaling of combinatorial search space with problem size (also popularly known as combinatorial explosion) and the potential difficulty of solving combinatorial optimization problems [2].

The roots of the interplay between computational complexity and combinatorial optimization can be traced back to Stephen Cook's seminal 1971 paper [3], where he for-

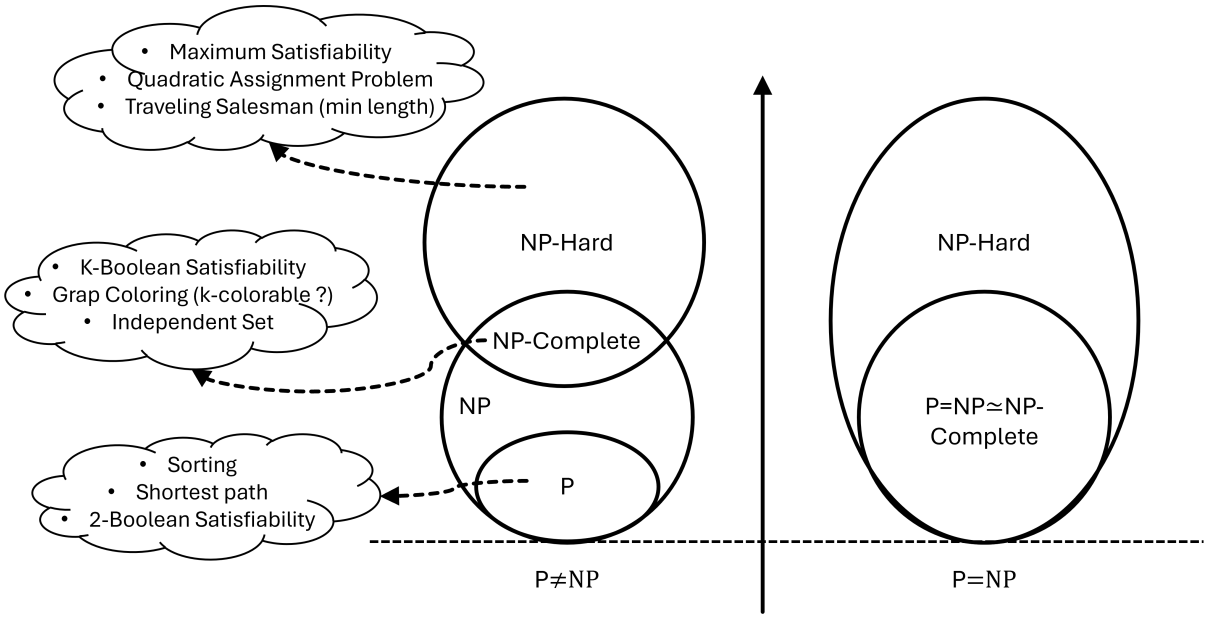


Figure 1.1: Euler diagram for P, NP, NP-Complete, and NP-Hard set of problems.

mally defined the complexity class NP (non-deterministic polynomial) and proved that propositional satisfiability is NP-Complete. Interestingly, satisfiability (also the primary combinatorial optimization problem of interest in this thesis) was the first ever problem (combinatorial or otherwise) to be shown to belong to the NP-Complete class. Subsequently, Richard Karp in 1972 [4] showed that 21 fundamental problems across graph theory, combinatorics, scheduling, and number theory are NP-Complete by demonstrating their polynomial-time reducability to propositional satisfiability.

Fig. 1.1 shows the relationship between problems in complexity classes P, NP, NP-Complete and NP-Hard using Euler diagrams. All problems in NP are decision-type problems, in that the solution to the underlying problem is a 'yes' or 'no'. Correctness of a candidate solution can be verified in polynomial-time using deterministic algorithms for problems in this class. Equivalently, a non-deterministic algorithm can be used to solve such a problem in polynomial-time. Problems in P are those that can be solved in polynomial-time using deterministic algorithms and therefore is a subset of NP. NP-

Complete constitutes the hardest problems in the NP class and are characterized by the property that any problem in NP can be reduced in polynomial-time to such problems. Problems in NP-Hard are at least as hard as the hardest problems in NP and may or may not be decision type problems. Readers are referred to [5] for detailed description of different complexity classes.

Classic examples of combinatorial optimization include the Traveling Salesman Problem [6], Knapsack Problem [7], Graph Coloring [8], Quadratic Assignment [9], Boolean Satisfiability (K-SAT) [3] and Maximum Satisfiability (MAX-SAT) [10]. Problems like K-SAT and MAX-SAT are particularly interesting because of the higher-order (K) constraints (i.e., constraints involving more than two variables) found in their objective functions (discussed later in Equation 1.6) that naturally leads to increased complexity. A Boolean Satisfiability (K-SAT) problem with N variables comprises conjunction (AND) of M clauses, where a clause (C) is a disjunction (OR) of as many as $k \leq K$ literals (with $k \equiv K$ for uniform random problems), i.e., normal or complementary Boolean variables (x_i's). The objective is to find a variable assignment that satisfies all clauses of the Boolean function. While K-SAT is a decision-type problem, MAX-SAT can be seen as an optimization version of the SAT problem with a similar structure. The goal here, is to find a variable assignment that maximizes the number (or the weighted sum) of satisfied clauses. These problems find widespread applications across diverse fields, including operations research [11, 12, 13], computer science (circuit design [14], compiler optimization [15], and routing), machine learning [16], and computational biology (sequence alignment and protein folding) [17, 18]. These problems also play an important role in the computation of Ramsey numbers [19], that represents one of the most challenging problems in extremal combinatorics due to their rapid growth and combinatorial complexity. SAT/MAX-SAT help tightening the estimates for Ramsey numbers by providing constructive lower bounds [20].

Their widespread prevalence has spurred development of diverse range of efficient algorithms and solver frameworks. Broadly, most approaches can be categorized into these five categories: (i) exact, (ii) approximate, (iii) heuristic, (iv) metaheuristic or (v) Learning based methods.

In the context of SAT/MAX-SAT problems, exact methods like Conflict-Driven Clause Learning (CDCL) guarantee optimality of solution but often exhibit high computational cost in conventional computers. Approximate methods provide formal guarantees on solution quality (not optimality) within reasonable runtime bounds. Heuristic methods like GSAT (Greedy SAT), WalkSAT, are often problem specific, do not guarantee optimality or solution quality but are computationally efficient and effective for many practical problem instances. Metaheuristic methods like simulated annealing and genetic algorithms provide a generic framework for solving a wide range of combinatorial optimization problems. These approaches often rely on underlying heuristics, but incorporate strategies to escape local optima and explore the search space more broadly, albeit without any formal guarantees of optimality. Finally, learning-based methods employ machine learning (or related) techniques to guide the search process more intelligently.

A notable solver framework within the metaheuristic category is the class of Ising machines [21], which has attracted significant attention in both research and industry over the past few decades. Their appeal stems from their versatility, high performance, inherent parallelism, and their close alignment with physics.

1.2 Ising Machines

Ising machines are hardware solvers that aim to find the absolute or approximate ground states of the Ising model from statistical physics, which describes interactions between spins in a magnetic system [21]. In the classical Ising model, each spin s_i

can take values in {-1,+1}, and the total energy of the system is determined by the Hamiltonian

$$E(\mathbf{s}) = -\sum_{i \neq j} J_{ij} s_i s_j - \sum_i h_i s_i \tag{1.1}$$

where J_{ij} represents pairwise interaction strengths between spins i and j, and h_i denotes external magnetic fields. The ground state of this system (configuration with minimal energy) corresponds to an optimal solution to the underlying Ising problem. Note that Ising models are closely related to the Quadratic Unconstrained Binary Optimization (QUBO) models [22] which are more commonly referenced in the mathematical literature. The latter is a quadratic discrete polynomial of the same form as shown in Equation 1.1 but over variables $x_i \in \{0,1\}$. Note that a polynomial in Ising over $s_i \in \{-1, +1\}$ can be expressed as a QUBO by substituting $s_i = 2x_i - 1$ and vice versa by substituting $x_i = (s_i + 1)/2$.

In 2014, Andrew Lucas's paper provided Ising formulations of Karp's 21 NP-Complete problems and several other NP-Hard problems [23]. This means that the objective function of any problem in complexity class NP can be translated to the Ising model. This means the development of a scalable Ising Machine could provide a generic framework for solving combinatorial optimization problems and have a considerable impact for practical applications. This and their close association with several physical phenomena has led to development of promising demonstration of Ising Machines using quantum tunneling in quantum annealers [24], thermal excitations in classical thermal annealing [25], minimum gain principle in coherent Ising machines [26] and attractors in dynamical system based solvers [27]. Readers are referred to [21] for a detailed review on Ising machines and their implementations on different physical substrates.

Computing gradients is essential across many applications [28, 22], such as training

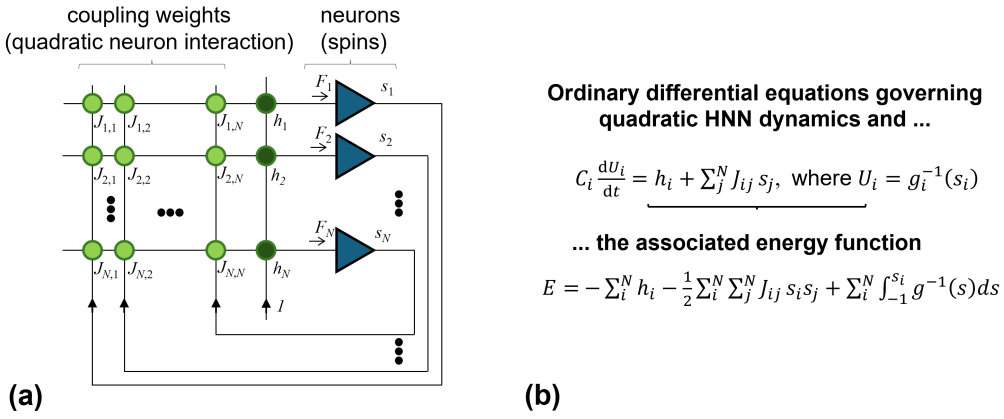


Figure 1.2: Illustration of (a) a quadratic Hopfield Neural Network and (b) its associated equations.

the weights in modern Deep Neural Networks (DNN) [29], and even in Ising machines (IMs) [21] or closely related approaches like Hopfield neural networks (HNNs) [30] and Boltzmann machines (BMs) [31]. A HNN can be seen as a neural-network based Ising Machine. A continuous-time second order HNN [32] consists of a recurrently connected network of pairwise symmetrically coupled graded-response neurons as shown in Fig. 1.2. The weight matrix in the network is set according to the matrix **J** corresponding to the coupling coefficients of the underlying objective function that is to be minimized. Neuron states are continuously updated to seek the minimum of the associated scalar energy function [32] according to Equation 1.2, where C_i, R_i are the capacitance and resistance associated with the leaky-integrating neurons, u_i is the internal neuron state often referred to as the membrane potential and g(.) is a nonlinear function like sigmoid or tanh. As a result, the inputs to the neurons are the continuously evolving partial-derivatives of the underlying objective function with respect to the neuron values. It has been shown that the given dynamics actually minimize an underlying Lyapunov function given by Equation 1.3, which would appear to be different from the Ising model given in Equation 1.1. However, by satisfying certain conditions and following a schedule that

gradually increases the steepness of the function g(.), the neuron states s_i relax to the bipolar values (-1,+1) and the ground state of Equation 1.1 is attained [33].

$$C_i \frac{du_i}{dt} = -\frac{u_i}{R_i} + \sum_j J_{ij} s_j + h_i, s_i = g(u_i) \tag{1.2}$$

$$E(\mathbf{s}) = -\frac{1}{2} \sum_{i \neq j} J_{ij} s_i s_j - \sum_i h_i s_i + \sum_i \int_{-1}^{s_i} g^{-1}(s)\, ds \tag{1.3}$$

Discrete-time solvers on the other hand compute the discrete gradients (also known as local field), given by Equation 1.4. It is the negative of the discrete partial derivative of the Ising model in Equation 1.1 with respect to spin s_i.

$$F_i = -\frac{\Delta E(s)}{\Delta s_i} = h_i + \sum_{j \neq i} J_{ij} s_j \tag{1.4}$$

The energy change $\Delta_{s_i} E$ given by $2 s_i F_i$, then corresponds to the change in energy when spin s_i is flipped. In a discrete-time HNN, spins are then updated as $s_i = sign(F_i)$ [30]. However, such a greedy update rule can result in the solver being stuck in a local minima. As a result, more performant Ising machine implementations incorporate simulated annealing along with sampling to guarantee asymptotic convergence to global minima of underlying objective function under certain conditions. One such implementation involving Glauber dynamics [34], would be to select a spin s_i at random (or cyclically), propose an update with probability equal to $\sigma(-\Delta_{s_i} E/T)$ (where $\sigma(.)$ is the sigmoid function) and flip the spin if accepted, while gradually reducing the algorithmic temperature T according to a certain schedule from a very high value initially to zero [25].

Many hard constrained satisfaction problems (CSPs) including Boolean Satisfiability (K-SAT), Maximum Satisfiability (MAX-SAT), often have objective functions that have

third or more order interactions that result in Ising models with higher order ($K > 2$) monomials as shown in Equation 1.5.

$$E(\mathbf{s}) = -\sum_{i} h_i s_i - \sum_{i<j} J_{ij} s_i s_j - \sum_{i_1<i_2<i_3} J_{i_1 i_2 i_3} s_{i_1} s_{i_2} s_{i_3} - \cdots - \sum_{i_1<i_2<\cdots<i_K} J_{i_1 i_2 \ldots i_K} \prod_{a=1}^{K} s_{i_a}. \tag{1.5}$$

Take for instance the CNF shown in Equation 1.6, where CNF refers to the Conjunctive Normal Form (the standard way of sharing and storing K-SAT problems) and $\mathrm{x}_i \in \{0, 1\}$.

$$\text{CNF} = (\neg x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_2 \vee x_3 \vee x_4) \wedge (x_3 \vee x_1 \vee \neg x_4) \tag{1.6}$$

One can then formulate the objective function depicting the number of unsatisfied clauses for any given variable assignment as shown in Equation 1.7. This is obtained by taking for each clause the product of a function of its constituent literals (where the function pertains to x_i or 1-x_i depending on whether the complementary or normal literal of that variable is present in that clause) and summing them all up. The resulting polynomial for this example has a maximum order of three, corresponding to the maximum clause order in the original CNF.

$$\begin{aligned} H &= x_1(1-x_2)x_3 + x_2(1-x_3)(1-x_4) + (1-x_3)(1-x_1)x_4 \\ &= x_1x_3 - x_1x_2x_3 + x_2 - x_2x_3 - x_2x_4 + x_2x_3x_4 + x_4 - x_3x_4 - x_1x_4 + x_1x_3x_4 \end{aligned} \tag{1.7}$$

Note that while Equation 1.7 corresponds to a Polynomial Unconstrained Binary Optimization (PUBO), it can be converted to the high-order Ising model (like in Equa-

tion 1.5) through variable substitution as described earlier. However, since most Ising Machines can only implement second-order interactions, a quadratization step is necessary for reducing the high-order terms to second-order. This quadratization step (often referred to as order reduction throughout this thesis) often increases the number of variables in the problem, as well as the required precision and range of the new second-order interaction weights compared to the original high-order ones [35]. The auxiliary variables increase the number of feasible points in the state space exponentially and assuming an exponential dependence of solution time on problem size, a solver can now possibly take exponentially longer time to solve the reduced quadratic problem compared to the original high-order one (exponentially or more generally super-polynomial at least in the worst case [1] is expected since the new problem also lies in the NP-Complete/Hard class) [37, 38]. This shows the importance of developing Ising machine hardware that can natively map and solve high-order optimization problems.

Among the different physical substrates available for implementing Ising Machines, those using digital, analog or mixed-signal techniques on CMOS or CMOS-compatible hardware [39, 40, 41, 42, 43, 44] are especially interesting because of the maturity of CMOS fabrication technology, scalability at cost, programmability and deployability. Among the different CMOS-based options, in-memory computing has become a key architectural concept in play that helps accelerate the core operations of an Ising machine.

1.3 In-Memory Computing

Technology scaling [45] has benefited the computing industry significantly over several decades to the effect that low-cost portable computing is now ubiquitious. While

[1] Note that for certain high-order problems, such as 3-Regular-3-XORSAT, it has been shown for a specific solver that the quadratic reformulation indeed exhibits worse convergence times compared to the original K=3 formulation. However, the degradation did not scale exponentially with problem size [36].

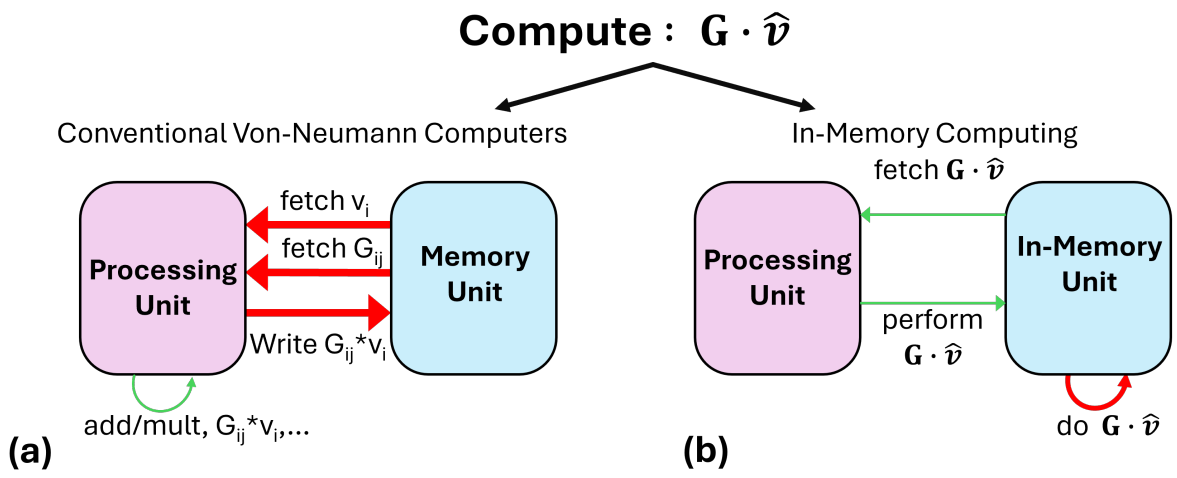
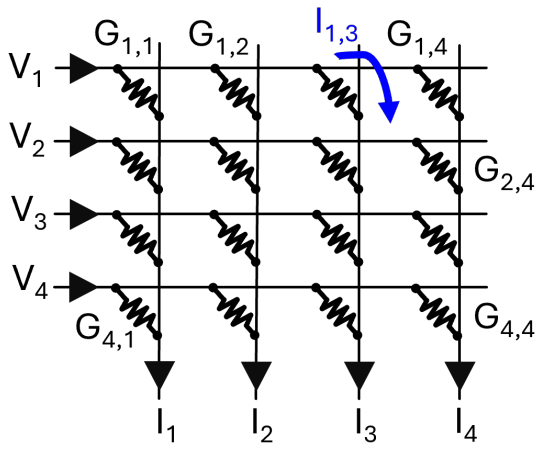


Figure 1.3: Illustration of performing vector-matrix multiplication in (a) conventional Von Neumann computers and (b) using In-Memory computer.

Moore's law has increased the density of integrated circuits and made computing blocks (like arithmetic logic units) significantly faster, the speed and throughput scaling of memory blocks (like SRAM arrays in cache, DRAM for main memory) has lagged behind [46]. Some notable factors behind this are increased refresh rates in DRAM [47], fundamental speed limits imposed by signal-to-noise ratio (SNR) requirements in analog sensing of bit line voltages in memory arrays that is further exacerbated by voltage scaling [48], growing memory array sizes leading to increased word line delays [49] and the ever growing separation between the memory and compute blocks in the traditional Von-Neumann architecture that has lead to increased power consumption in communicating larger chunks of data between the two [50]. This has led to what is popularly known as the memory wall (or also the Von-Neumann bottleneck) [46]. This has significantly effected data intensive workloads like machine learning, data analytics, graph analytics, scientific computing and optimization that demand a higher throughput communication between the compute and memory blocks.

This has led to the development of heterogenous computing platforms like Graphic



$$I_{i,j} = V_i G_{i,j} \quad \text{[Ohm's Law]}$$
$$I_j = \sum_{i=1}^{4} I_{i,j} \quad \text{[Kirchoff's Law]}$$
$$I_j = \sum_{i=1}^{4} V_i G_{i,j}$$

Figure 1.4: Illustration of vector-matrix multiplication being implemented with in--memory computing approach using a 4×4 crossbar of programmable resistive devices.

Processing Units (GPUs) [51], Tensor Processing Units (TPUs) [52] and other application specific accelerators [53, 54, 55, 56]. These use a combination of techniques like single-instruction-multiple-thread (SIMT) [51], systolic arrays, massive core- and thread-level parallelism and high-bandwidth memory interfaces among other things to accelerate vector, matrix and tensor computations that arise in workloads like deep learning, large language models and graphic processing. However, such accelerators do not fundamentally solve the memory wall problem, rather amortizes the data fetch costs with arithmetic-intensive operations [50]. On the other hand, Near-Memory Computing (NMC) [57] better co-locates memory and compute, and has significantly benefitted from advanced memory technologies like High Bandwidth Memory (HBM) [58], Hybrid Memory Cube (HMC) [59] and 3D integration technologies [60]. However, there still exists a physical seperation between the memory and compute.

In-Memory Computing (IMC) is an alternate and promising approach, where com-

putational tasks are performed within the memory array itself, by having pre-loaded a subset of the operands onto the memory array and subsequently exploiting the physical attributes of the memory elements, array organization, control circuitry and/or peripheral circuits to perform the desired computation [61] (see Fig. 1.3 for an illustration of the concept). Several memory devices like Dynamic Random Access Memory (DRAM) [62], Static-Random Access Memory (SRAM) bitcells [63], Floating-Gate cells (NOR Flash) [64], memristors like two terminal Resistive Random Access Memory (RRAM) [65] have been explored as potential candidates for In-Memory Computing, each with their own set of trade-offs. In this dissertation we take the opportunity to implement In-Memory Computing based optimization prototypes with each of these memory elements (though some more than others) excluding DRAM.

Besides improving the energy and latency associated with data transfer, In-Memory Computing can also reduce the computational complexity of certain tasks like for instance Matrix-Vector Multiplications (MVM). This is better explained using the example illustrated in Fig. 1.4. It shows a 4 x 4 array of resistors with the resistor in the ith row and jth column having a conductance of $G_{i,j}$. When a vector of voltages $\mathbf{V} = \{V_1, V_2, V_3, V_4\}$ is applied to the word lines (horizontally running lines), the resulting currents in the bit lines (vertical lines) are equal to the MVM operation between vector $\mathbf{V}$ and matrix $\mathbf{G}$ (denoting the conductances), owing to Ohm's and Kirchoff's laws in action. Since the result of the MVM operation is available immediately after the application of the input voltages (with additional delays incurred due to memory device and array parasitics), the computational complexity of such an operation is nearly O(1), whereas the same tends to be O(N^2) or more in the case of CPUs or GPUs (N is the size of the vector).

In the context of Ising machines, In-Memory Computing plays an important role in accelerating the computation of the discrete gradients (also known by other names like local field), given by Equation 1.4 and guides many optimization schedules by helping

identify energy minimizing spin updates. In the case where the external fields (h_i) are all equal to zero, the computation of local fields of all spins can be casted as a matrix-vector multiplication between the vector of spins ($\mathbf{s}$) and the matrix $\mathbf{J}$. This MVM can be executed using an In-Memory Computing array (like in Fig. 1.4), where the input voltages are proportional to the spin values ($V_i \in \{-V_r, +V_r\}$, where V_r is the read voltage of the memristor), the bit lines voltages are biased to virtual ground (= 0 V in this case) and the conductance of memory elements set proportional to the coupling coefficients ($G_{i,j} = \alpha J_{ij}$) of the Ising model. Such massively parallel local field/gradient computation is possible in the case of a QUBO objective function by setting the input voltage ranges accordingly ($V_i \in \{0, V_r\}$). Note that when external fields are non-zero, they can be either adjusted for in the peripheral sensing circuits or added directly by allocating an additional row in the crossbar array to source currents proportional to corresponding h_i's. While In-Memory computing approaches have shown promise for improving performance of second-order Ising Machines significantly, their potential for accelerating higher-order Ising Machines have remained unexplored, and this is one of the core contributions of this dissertation.

While In-Memory Computing exploiting the analog physical properties of memory devices show promise for alleviating the memory wall problem, they are also plagued with a number of issues like limited weight precision (1-8 bits, depending on the memory technology used), precision-hardware cost tradeoffs in driving and sensing peripheral circuits (accurate sensing for neural networks often require high-precision Analog-to-Digital Converters), fixed and time-varying variability, capacity-latency tradeoff (large crossbar arrays have higher latency, requiring tiling approaches for larger problems) [61, 66, 67, 68]. Many of these issues require mitigation techniques that are specific to the application. Throughout this thesis, we present hardware-software co-design techniques for mitigating many of these issues like precision, ADC-requirement, in addition to a multi-tile archi-

tecture for implementing scalable high-order Ising Machines with In-Memory computing.

1.4 Thesis Outline

This thesis is organized into seven chapters and four appendices. Chapter 1 serves as an introduction to the thesis and provides background necessary to discuss the future chapters of the thesis. This includes a brief introduction to combinatorial optimization problems, Ising Machines and In-Memory Computing. Chapter 2 and Appendix A presents our proposed cross-coupled dual in-memory crossbar array approach to computing the discrete gradients in High-Order Ising Machines in a massively parallel fashion. We present problem mapping techniques, array biasing conditions, experimental results from a prototype integrated circuit comprising metal oxide memristor crossbar arrays to validate the approach, sensitivity study, comparison with other works and discuss applicability of this approach to implement dynamical system based high-order Ising solvers.

Chapter 3 presents the first integrated circuit that was taped out in 55 nm CMOS technology, to demonstrate the applicability of our high-order gradient computing idea by solving small-scale K-SAT problems using a legacy SAT solving algorithm. We present detailed schematics, architecture, workflow, measurement results and comparison with prior works.

Chapter 4 presents a new probabilistic SAT solving algorithm that is suitable for parallelism and solves SAT problems with improved navigational efficiency. The design details and measurement results of a second integrated circuit demonstrating this new algorithm is also presented.

Chapter 5 and Appendix B presents a framework for implementing high-order ising machines with in-memory computing approaches in CMOS-based application-specific integrated circuits. We provide classifications of different high-order gradient comput-

ing schemes, extend circuit implementations to include native mapping and solving of constrained satisfaction problems with high-order XOR constraints and hardware-aware simulation results. Chapter 6 presents results on a FPGA-inspired multi-tile architecture for scaling up high-order Ising machines to solve large-scale problems. Chapter 7 presents a summary of the core contributions of this dissertation, some preliminary results and discussion on a third integrated circuit design comprising in-memory computing macro implemented with analog weight capable Floating-Gate (NOR Flash) cells, a digital in/near-memory computing approach to solving Boolean Satisfiability problems, and an in-memory computing based complete K-SAT solver, and finally concludes with an outlook to the future of these solvers.

Appendix C presents experimental and simulation results on a ReRAM-based neo-Hebbian synapse for accelerated online-training neuromorphic hardware, while Appendix D provides preliminary results from a hardware platform demonstrating near-sensor computing with event-based pixels, spike-based pattern recognition, and memristor-based in-memory computing. Both appendices cover topics that are unrelated to Ising machines and high-order combinatorial optimization.

1.5 Permissions and Attributions

1. The content of Chapter 2 and Appendix A has appeared previously in the journal Nature Communications [69]. It is reproduced here under the Creative Commons Attribution 4.0 International.

2. Part of the content of Chapter 3 has appeared previously in the proceedings of the 2025 Symposium on VLSI Technology and Circuits [70] ©2025 The Japan Society of Applied Physics. Reprinted with permission. Part of the content of Chapter 3 has also appeared in the proceedings of the 2025 IEEE Hot Chips Symposium

(HCS) [71] ©2025 IEEE. Reprinted with permission.

3. Part of the content of Chapter 4 will in the proceedings of the 2026 IEEE International Solid-State Circuits Conference (ISSCC) [72] ©2026 IEEE. Reprinted with permission.

4. Part of the content of Chapter 6 has appeared previously in the proceedings of the 2024 IEEE Computer Society Annual Symposium on VLSI (ISVLSI) [73] ©2024 IEEE. Reprinted with permission.

5. Portions of Chapter 7 are adapted from material included in a manuscript [74] submitted for review to the IEEE Journal on Emerging and Selected Topics in Circuits and Systems (JETCAS) as of August 2025.

6. The content of Appendix C has appeared previously in the proceedings of the 2023 IEEE International Electron Devices Meeting (IEDM) [75] ©2023 IEEE. Reprinted with permission.

7. The content of Appendix D will appear in the proceedings of the 2025 59th Asilomar Conference on Signals, Systems, and Computers [76] ©2025 IEEE. Reprinted with permission.

Chapter 2

Computing High-Order Discrete Gradients In-Memory

2.1 Background

Most promising Ising Machine (or Hopfield Neural Network) implementations rely on massively parallel spin/updates which in turn are dependent on fast gradient computation (as seen in Chapter 1) to rapidly converge. Most state-of-the-art Ising Machines capable of such degree of parallelism rely on in-memory computing architectures and are restricted to mapping and computing gradients of second-order Ising models [43, 44]. Though there have been proposals of higher-(K)-order Ising Machines/HNNs and their variants [37, 35, 77, 78] that rely on computing gradients of K-degree polynomials, an efficient hardware implementation is lacking, while the previous in-memory computing proposals do not readily extend to larger K. For example, a straightforward in-memory crossbar array implementation of K-order HNN with N neurons (e.g., N variables in energy function) requires N^K coupling weights [37], i.e., hardly practical for larger N and/or K. Meanwhile, functions of higher K can represent increasingly important and challenging

problems. For example, polynomial unconstrained binary optimization (PUBO) problems are described by K-degree polynomials and naturally arise in protein folding and other first-principle calculation methods [79, 80] and operations research. Notably, K grows linearly with the size of molecular system (molecular orbitals) in PUBO approaches for calculating electronic structure [80]. The well-known K–Boolean–satisfiability (K-SAT) problem goes from polynomial to expected exponential runtime as K increases from 2 to 3 [5]. While higher $K > 3$ can be mapped to the K=3 case, this requires a polynomial increase in the variables, potentially increasing the runtime with increasing K. Therefore, the operation of such networks relies on efficient computations of higher order polynomial gradients.

In this chapter, we present the development of a paradigm for computing gradients of arbitrary-degree polynomial functions in a massively parallel fashion. The proposed paradigm enables efficient hardware that can immediately impact the above described use cases and can be more broadly applied to accelerate gradient computation of arbitrary functions when using Taylor expansion approximation.

2.2 Dual cross-coupled crossbar architecture

We expound upon binary-variable function (H) gradient computations using a high degree polynomial consisting of four variables (x) and four terms (i.e., monomials) of varying degree, each with unique factor a (Fig. 2.1). Computing the gradient of such a function requires calculations of all its (pseudo) partial derivatives $\Delta H_{x_i} \equiv H_{x_i} \Delta x_i$, where H_{x_i} is the difference quotient [81] of H with respect to x_i and Δx_i is the change in the variable value (Appendix A.1). In our approach, pseudo partial derivatives are decomposed into "make" and "break" components. (This terminology is inspired by stochastic algorithms used for solving SAT problems [82, 83].) The make component

of pseudo derivative with respect to a given variable represents the contribution from monomials that would change their value from a zero to a non-zero one, i.e., corresponding a value, after flipping the variable state. Similarly, the break component represents the contribution to the derivative from monomials that were previously evaluated to non-zero value but would become zero after changing the variable state. The difference between the make and break values is related to the pseudo partial derivative as shown in Equation 2.1 - see an example for a specific variable assignment in Fig. 2.1.

$$\Delta H_{x_i} = (\text{make}_{x_i} - \text{break}_{x_i}) \tag{2.1}$$

We call the monomials that contribute to make and break value summations as, correspondingly, make and break monomials. Note that the sets of make and break monomials do not overlap and depend on the current variable assignment and that some monomials might be of neither make nor break type. Our goal is to design efficient hardware that can identify such make, break monomials, and then compute make and break values for each variable.

Before discussing in-memory computing implementation, it is convenient to visualize high-order polynomials as a bipartite graph with variables and monomials representing two sets of vertices (Fig. 2.2). Such a graph maps naturally to a crossbar memory array by assigning variables to one set of (say, vertical) crossbar wires while monomials to another (horizontal) set of wires by setting the array's binary coupling weight to the "on" state, i.e., $b_{ji} = 1$, if the i-th variable is present in a j-th monomial, while setting the weight to the "off" state ($b_{ji} = 0$) otherwise (Fig. 2.2).

All pseudo derivatives are computed in parallel in three steps. First, in a "forward pass", variable values are applied to the crossbar array, and dot products $\sum_i b_{ji}x_i$ corresponding to the sums of monomial variables are calculated at the monomial side of the

Considered Polynomial function & its pseudo partial derivatives

$$H = a_1x_1 + a_2x_1x_2 + a_3x_1x_2x_3 + a_4x_1x_2x_3x_4$$

$$\Delta H_{xi} \equiv H_{xi}\Delta x_i \equiv (\text{make}_{xi} - \text{break}_{xi}) = (\bar{x}_i\Sigma_i^{\text{make}} - \; x_i\Sigma_i^{\text{break}})$$

			make		break		
ΔH_{x1}	$=(a_1 + a_2x_2 + a_3x_2x_3 + a_4x_2x_3x_4)\Delta x_1 =$		0	$-$	a_1	$=$	$-a_1$
ΔH_{x2}	$=(\quad + a_2x_1 + a_3x_1x_3 + a_4x_1x_3x_4)\Delta x_2 =$		a_2+a_3	$-$	0	$=$	a_2+a_3
ΔH_{x3}	$=(\qquad\quad + a_3x_1x_2 + a_4x_1x_2x_4)\Delta x_3 =$		0	$-$	0	$=$	0
ΔH_{x4}	$=(\qquad\qquad\quad + a_4x_1x_2x_3)\Delta x_4 =$		0	$-$	0	$=$	0

ΔH_x for $\mathbf{x} = \{1, 0, 1, 0\}$

Figure 2.1: Considered 4th-degree polynomial function. Note that binary-variable high-order polynomials are multilinear, i.e., degree of any variable in a term is not more than one.

crossbar array (Fig. 2.3a). This information is then used to identify all make and break monomials (Fig. 2.3b,c). Specifically, monomials whose variable sum is equal to their maximum value, i.e., monomial degree K_j, are break monomials, while those whose sums are one short of their degree $(K_j - 1)$ are make monomials. For example, for the specific variable assignment of the considered function, there are two (2nd and 3rd degree) make monomials and one (1st degree) break monomial (Fig. 2.3b,c).

The data flow via the crossbar circuit is reversed in the next two steps. Unit inputs for the identified make monomials, denoted by indicator variable s_j, and zero inputs otherwise, are applied at the monomial side of the array to compute the number of make monomials that each variable is a member of at the variable side of the array. In a more general case shown in Fig. 2.3b, in a "backward" make pass, unit inputs are scaled according to the monomial factors so that the computed dot products $\sum_j b_{ji}a_j s_j$, called make variable sums, correspond exactly to variables' potential make values. "Potential" because computed values are only relevant for currently zero variables, and only flipping those variables can change a monomial value to a non-zero one. Therefore, the make val-

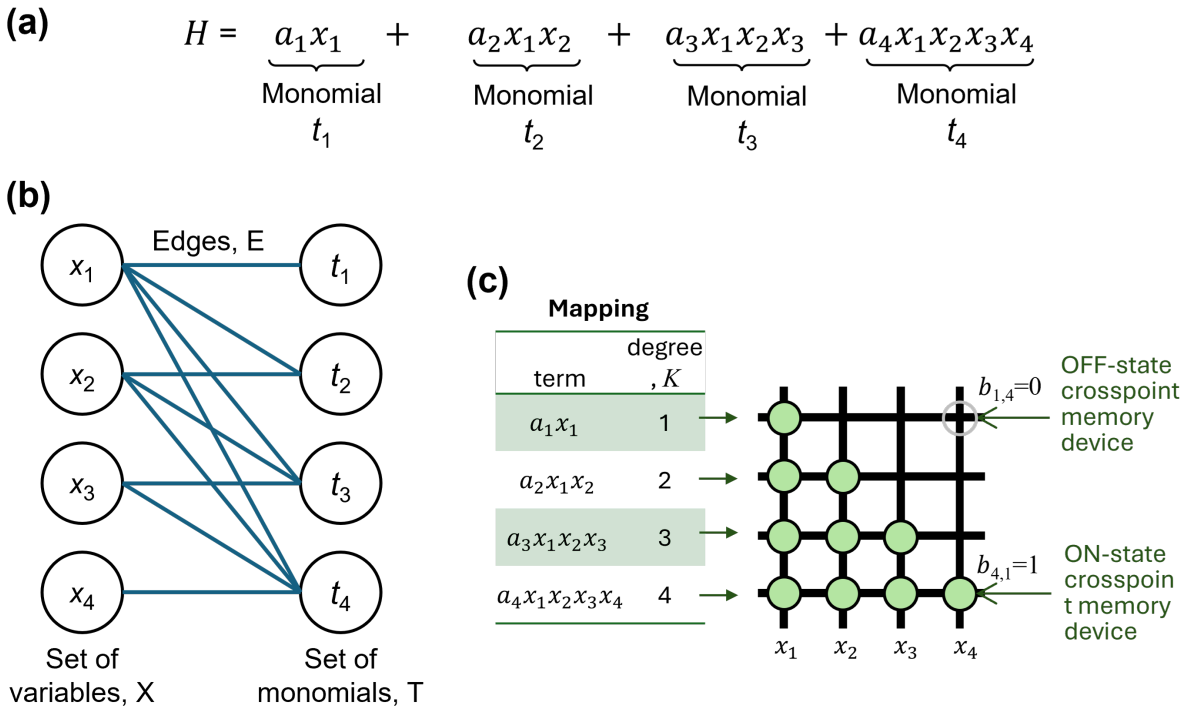


Figure 2.2: (a) Considered multi-linear polynomial example, (b) its bi-partite graph representation, and (c) corresponding crossbar circuit implementation.

ues are computed by multiplying the make variable sums at the crossbar array periphery by the inverted value of a variable, i.e., $\bar{x}_i \sum_j b_{ji} a_j s_j$, thus making the result zero for all variables that are currently one. Similar operations are performed in a "backward" break pass to compute break values, with the only difference that scaled unit inputs are applied according to the identified break monomials, denoted by indicator variable z_j, and the final peripheral multiplication is performed with normal variable values to compute the break values $x_i \sum_j b_{ji} a_j z_j$ (Fig. 2.3c). Appendix A.2 provides a more formal, rigorous framework for the proposed in-memory massively parallel computation of function pseudo gradient.

There are several important variations and extensions of the discussed approach. Monomial variable sums can be compared to fixed values (0 and 1) to identify make

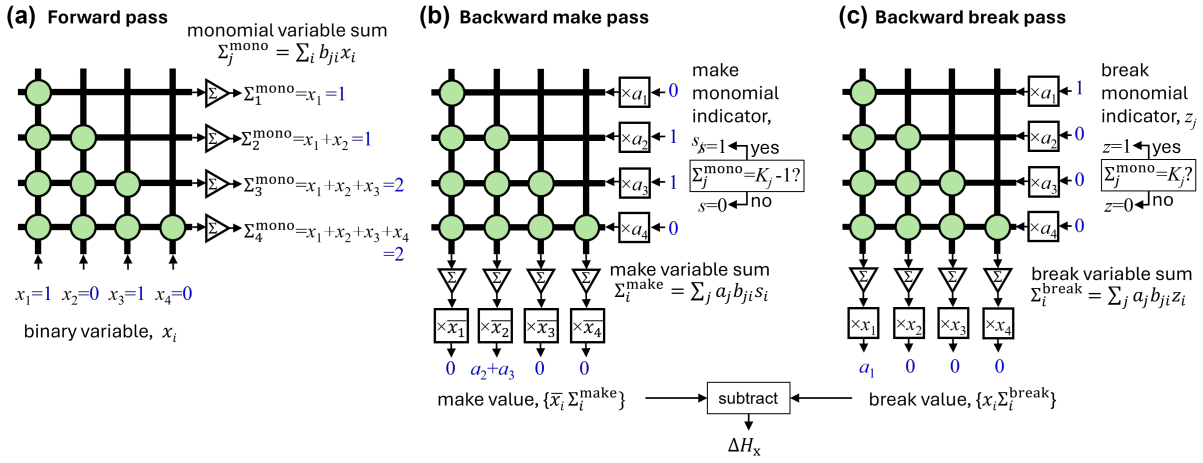


Figure 2.3: (a) Sums of monomial variables are first computed in the forward vector-by-matrix multiplication pass. These values are compared to monomial orders in backward steps (b) and (c) to identify break and make type monomials. Then, unit inputs, scaled by a monomial factor, are applied for the identified break and make monomials in the backward vector-by-matrix multiplication. Finally, the results are appropriately gated at the periphery to compute make and break terms of the pseudo partial derivatives for each variable. Values shown in blue correspond to specific variable assignment x_1=1, x_2=0, x_3=1, x_4=0.

and break monomials by applying inverted values of variables in the forward pass (Fig. 2.4a-c). This allows for simplifying peripheral circuitry of the backward passes by not requiring storing specific thresholds for each monomial of Fig. 2.3d,e approach. An input scaling in the backward passes can be implemented with more complex analog (multi-level) memory devices to simplify the periphery further (Fig. 2.4d). Also, backward pass computations can be implemented on separate crossbar arrays, which allows performing all three operations in parallel in a pipelined design to increase computational throughput (Fig. 2.4e).

More importantly for further discussion, the proposed approach is also suitable after minor modifications for computing pseudo gradients of Boolean logic functions expressed in conjunctive normal form (CNF), thus allowing solving Constraint Satisfaction Problems (CSPs) like SAT and MAXSAT, in native space without potentially time-consuming

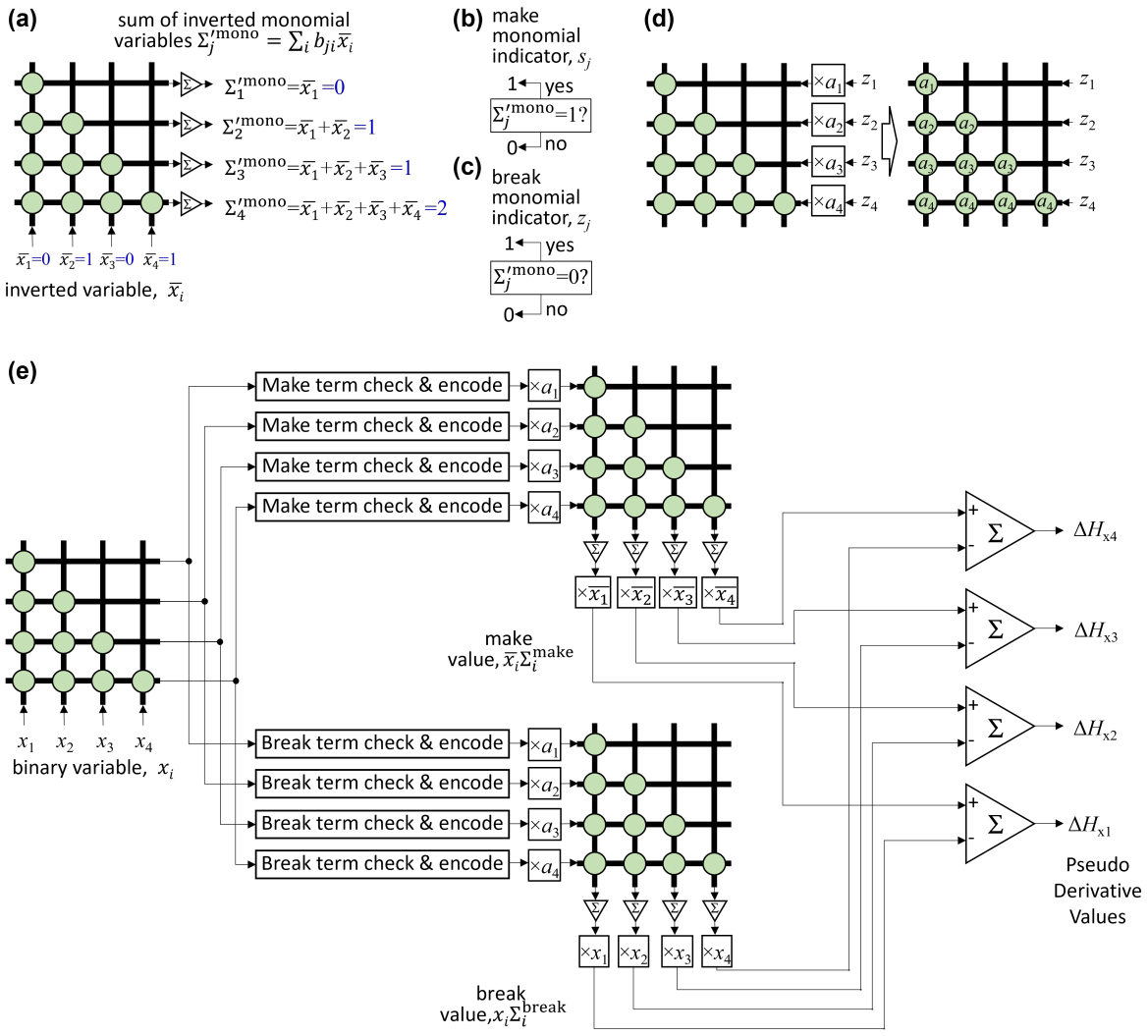


Figure 2.4: Simplified periphery and high-throughput extensions: (a-d) Simplifying peripheral circuitry by (a) inverting variable inputs in the forward pass and (d) implementing constant multiplication with multi-valued memory devices in the backward passes. With such modification, identifying make and break monomials with inverted variable inputs requires comparing dot products of the forward pass with fixed values of (b) 0 and (c) 1, correspondingly. (e) Increasing throughput by implementing backward passes with separate crossbar memory arrays. On panel a, values in blue correspond to the inverted version of the same variable assignment x_1=1, x_2=0, x_3=1, x_4=0 as that was used in Fig. 2.1. The polynomial mapped to the crossbars in all panels is the same as that in Fig. 2.1.

conversion to an equivalent PUBO form (see Equation 1.6 and related text for details on such conversion). CNF Boolean function comprises of conjunction (AND) of clauses, where a clause is a disjunction (OR) of literals, i.e., normal or complementary variables. The goal in the K-SAT problem, known as NP-hard for $K \geq 3$, is to find a variable assignment that satisfies all clauses of the given CNF function, with up to K literals per clause (see Chapter 1.1 for further details). Fig. 2.5 shows details of all steps for parallel computation of Boolean variable gain values. (A gain value is commonly used in SAT community to describe the change in the number of satisfied clauses after flipping a variable[42,43]. It is effectively the negative of the pseudo partial derivative with respect to that variable of an equivalent PUBO energy function.) In this case, clauses are mapped to crossbar array rows, while literals (Fig. 2.5b), are mapped to crossbar array columns. Forward (Fig. 2.5c) and backward (Fig. 2.5d,e) operations are similar to those of the monomial approach with inverted input variables (Fig. 2.4a). Appendix A.3 provides a more formal framework for the proposed in-memory parallel computation of CNF pseudo gradients.

Finally, another variation (Fig. 2.6) is due to the use of crossbar arrays based on active, three-terminal memory devices, such as 1-transistor-1-resistor (1T1R) or floating gate memory. An additional "gate" signal in such crossbar arrays (Fig. 2.6a) can be used to condition the dot-product terms, which in turn allows for computing the make and break backward passes using a single crossbar in a single step (instead of two as shown in Fig. 2.4e) for high-throughput gain computation of CNF-form Boolean functions (see Appendix A.4 for details). Figure. 2.7 shows how such a one-shot gradient computation technique can be extended for CNF-form Boolean functions using 1T1R devices.

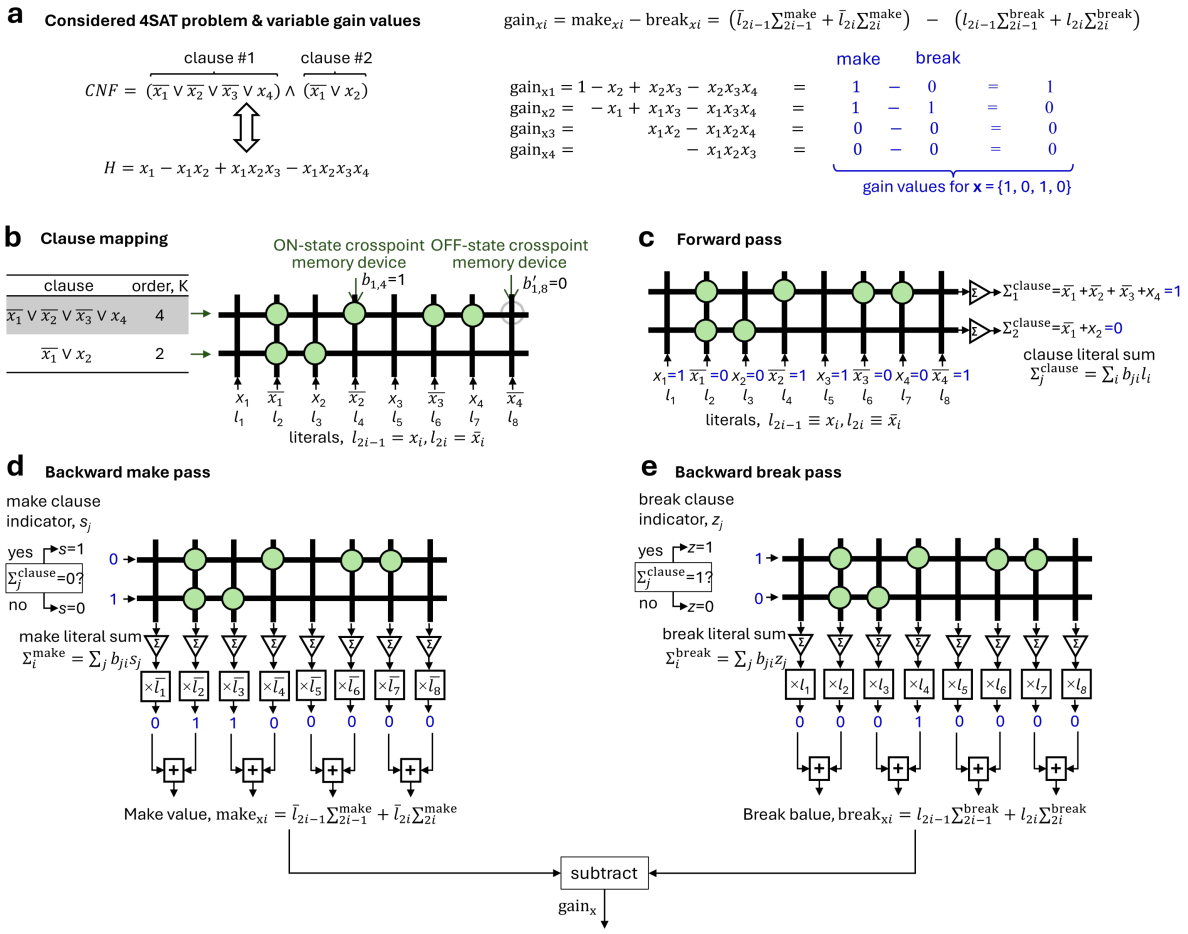


Figure 2.5: Conjunctive Normal Form gain computation. **a.** Considered 4th-degree CNF Boolean function, i.e., 4-SAT problem. **b–e.** The main idea of the gain computing showing (**b**) crossbar memory array implementation and (**c–e**) three in-memory computing operations. The approach is similar to the polynomial pseudo gradient computation. Specifically, sums of clause literals are first computed similarly to the inverted monomial approach (Fig. 2.4a) in the forward vector-by-matrix multiplication pass as shown in panel (**c**). The sum values are compared to clause orders in backward steps (**d**) and (**e**) to identify break and make type clauses. Unit inputs, that can be scaled by clause factor when clauses in CNF are weighted (e.g., used in weighted SAT problems), are applied for the identified break and make clauses in the backward vector-by-matrix multiplication. The results are then properly gated at the periphery to compute make and break values for each variable. The corresponding variable gains (i.e., negative pseudo partial derivatives) are found by subtracting the outputs of two backward passes as shown in panel (**a**). Values shown in blues correspond to $x_1 = 1$, $x_2 = 0$, $x_3 = 1$, $x_4 = 0$, i.e., the same example of assignment as in Fig. 2.1. Also note that the considered 4-SAT problem is equivalent to polynomial function in Fig. 2.1, when assuming $a_1 = 1$, $a_2 = -1$, $a_3 = 1$, and $a_4 = -1$.

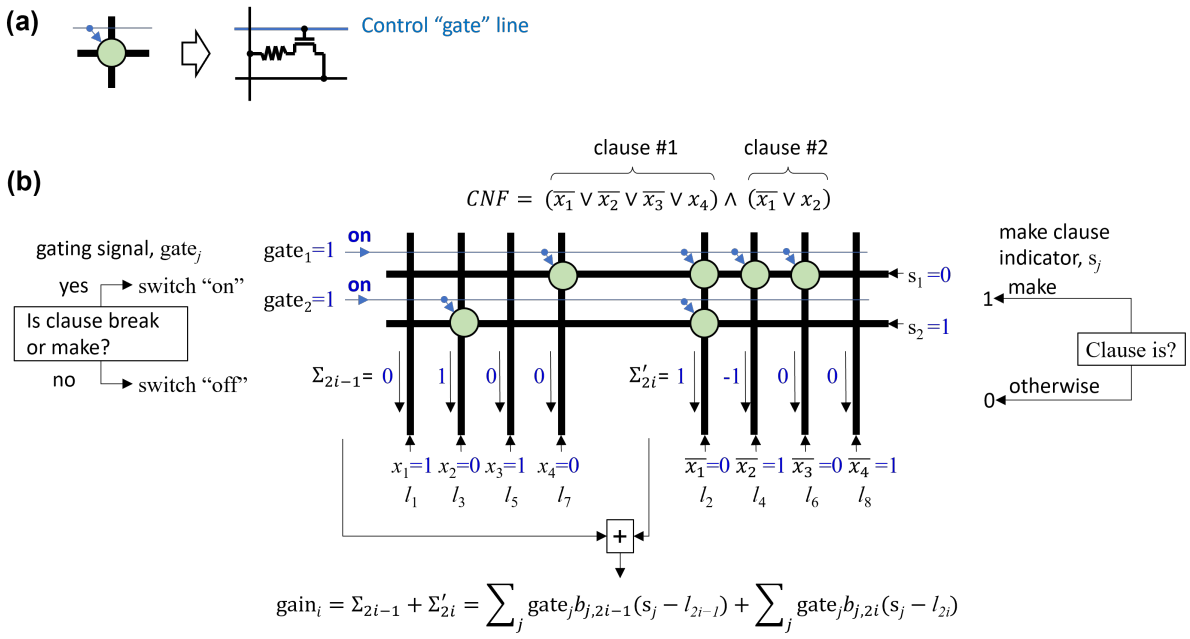


Figure 2.6: **Extension for three-terminal memory devices:** (**a**) A three-terminal memory device based on 1T1R memory. (**b**) Backward pass implementation to compute gain values using a single $M \times 2N$ crossbar array (instead of two such arrays as shown in Figs. 2.5d and e), where M is the number of clauses and N is the number of variables. Values in blue correspond to the specific variable assignment $x_1 = 1$, $x_2 = 0$, $x_3 = 1$, $x_4 = 0$. Σ_{2i-1} and Σ'_{2i} are sums of triple products along $(2i-1)$-th and $2i$-th columns corresponding to variable x_i's normal and complementary literals. Specifically, the triple product in Σ_{2i-1} (Σ'_{2i}) consists of terms $s_j - l_{2i-1}$ ($s_j - l_{2i}$), where s_j is the make clause indicator, the weight value $b_{j,2i-1}$ ($b_{j,2i}$), and gating signal $gate_j$. The gating signals are applied to 1T1R control lines and turn on only make or break type clauses. Variable gain values are obtained by adding Σ_{2i-1} and Σ'_{2i}.

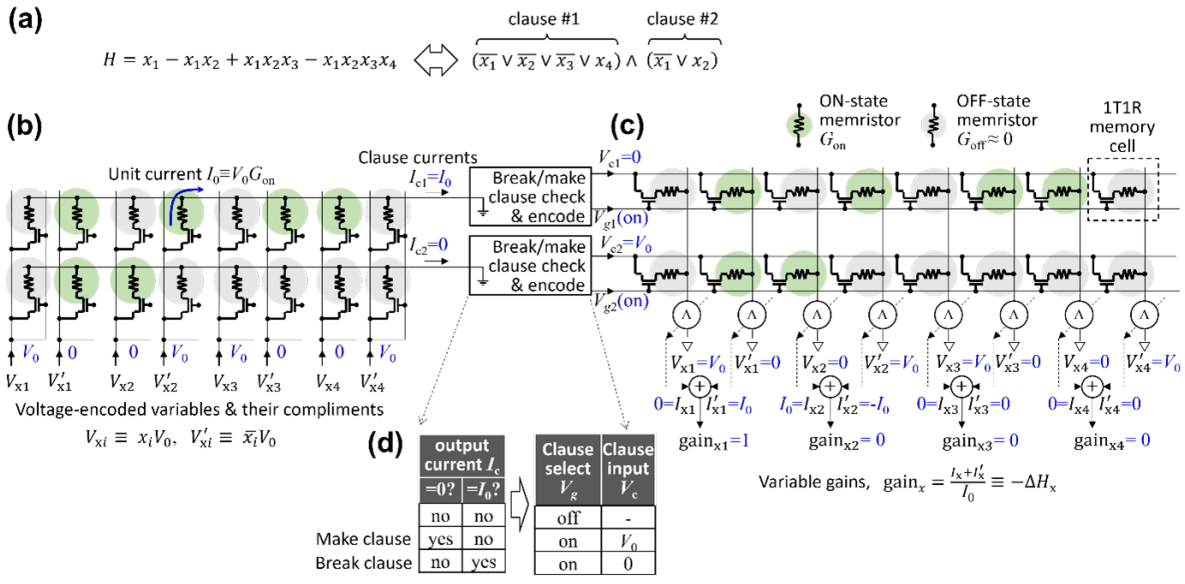


Figure 2.7: **A compact 1T1R implementation for CNF-form Boolean function gain calculation.** (**a**) The considered toy example of CNF function. (**b, c**) 1T1R circuit implementations of (**b**) forward and (**c**) backward computation of the proposed approach. (**d**) Peripheral logic functionality. The bit lines in the second array are tied to the literal values so that $I_x + I'_x$ is proportional to the $gain_{x_i} = \sum_j gate_j b_{j,2i-1}(s_j - l_{2i-1}) + \sum_j gate_j b_{j,2i}(s_j - l_{2i})$.

2.3 Experimental Demonstration with memristor crossbar

The key functionality of gain computation in CNF-form Boolean logic functions was experimentally validated by solving a high-order combinatorial optimization problem (Figs. 2.8, 2.9). Specifically, the studied optimization problem is a custom-generated uniform random 3-SAT with $N = 14$ variables and $M = 64$ clauses (Appendix A.5). The problem parameters were chosen to maximize the use of hardware resources and problem hardness [84]. The WalkSAT/SKC algorithm [83], a state-of-the-art local search heuristic, was implemented to solve a 3-SAT problem. Such an algorithm repeatedly flips variables using information on their break values to converge to the solution. An unsatisfied clause is first selected randomly out of all unsatisfied clauses determined in a forward pass. Break values for all variables within the selected clause are computed in parallel in the backward pass. Then, a specific variable is flipped according to the algorithm heuristics, see Appendix A.6 for more details on the 3-SAT instance generation and algorithm.

The experiments were performed on a prototype board that features several 1T1R TaO_x memristor crossbar arrays, back-end-of-the-line monolithically integrated with 180 nm CMOS circuits implementing driving, sensing, memory programming, and input/ output (analog-to-digital conversion) functions (Fig. 2.9). The forward and backward operations were tested on $M \times 2N$ sub-arrays of two crossbar arrays of a chip. Specifically, the conductance of on-state ($b_{ji} = 1$) memristors were first tuned to $G_{on} = 110$ μS with 15% tuning accuracy using the write-verify approach (Fig. 2.8c), while the conductance of all off-state ($b_{ji} = 0$) and outside of the utilized sub-array memristors were set to as small as possible (< 10 μS) values.

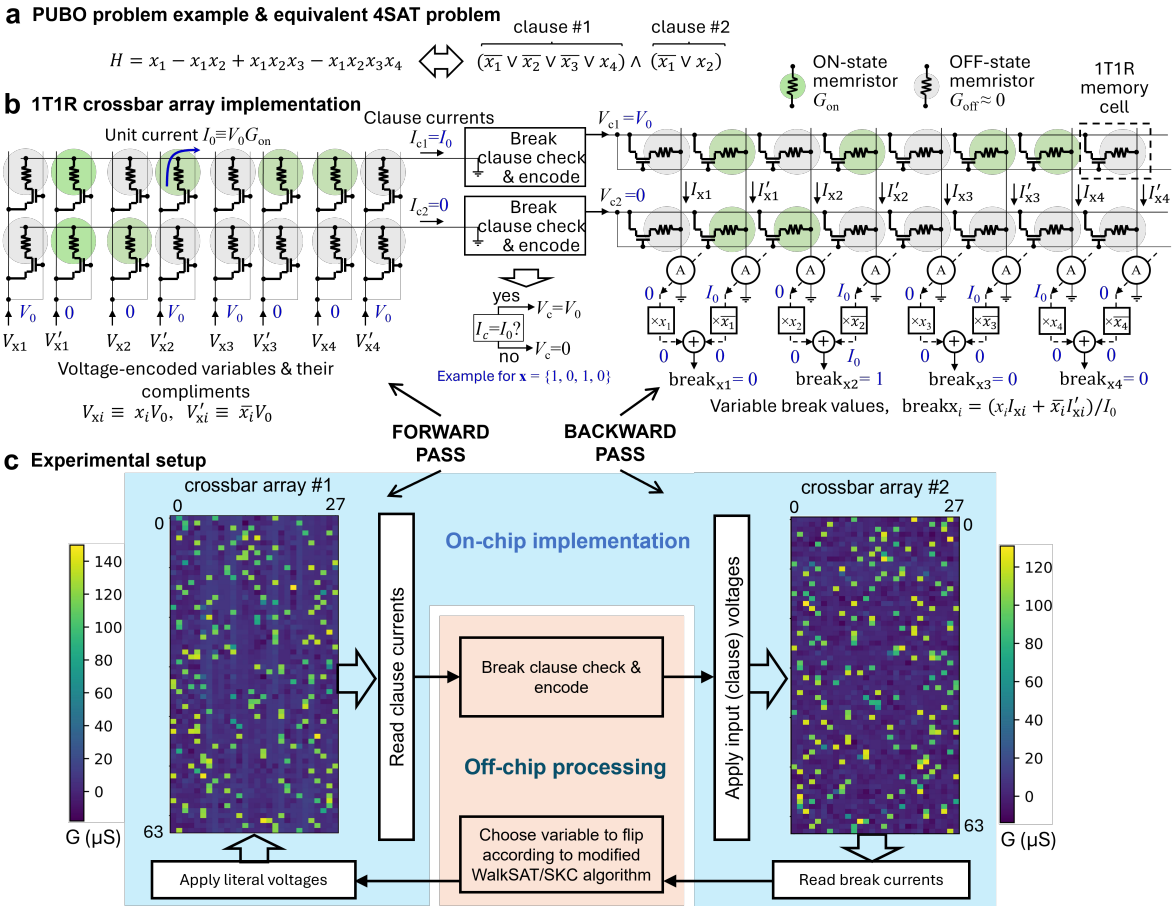


Figure 2.8: **In-memory computing hardware implementation.** (**a**) A toy example of CNF Boolean function with $M = 2$, $N = K = 4$, which is equivalent to the monomial in Fig. 1a with $a_1 = 1$, $a_2 = -1$, $a_3 = 1$, and $a_4 = -1$ when represented in a PUBO form. (**b**) A prospective in-memory 1T1R memristor circuit implementation of panel (**a**) problem for parallel computation of its break values. Each 1T1R memory cell is comprised of a select transistor coupled with a memristor. Note that the gate voltages are tied to the word lines in the utilized chip to suppress leakages through unselected memory cells. A text shown in blue corresponds to a specific variable assignment $x_1 = 1$, $x_2 = 0$, $x_3 = 1$, $x_4 = 0$. (**c**) Experimental setup details for solving the considered 14-variable, 64-clause 3-SAT problem. Two arrays represent 1T1R crossbar circuits of a chip in the integrated CMOS/memristor setup and are used for demonstrating forward and backward passes. The colormaps show the measured conductance of programmed memristors corresponding to the studied 3-SAT problem in the experiment.

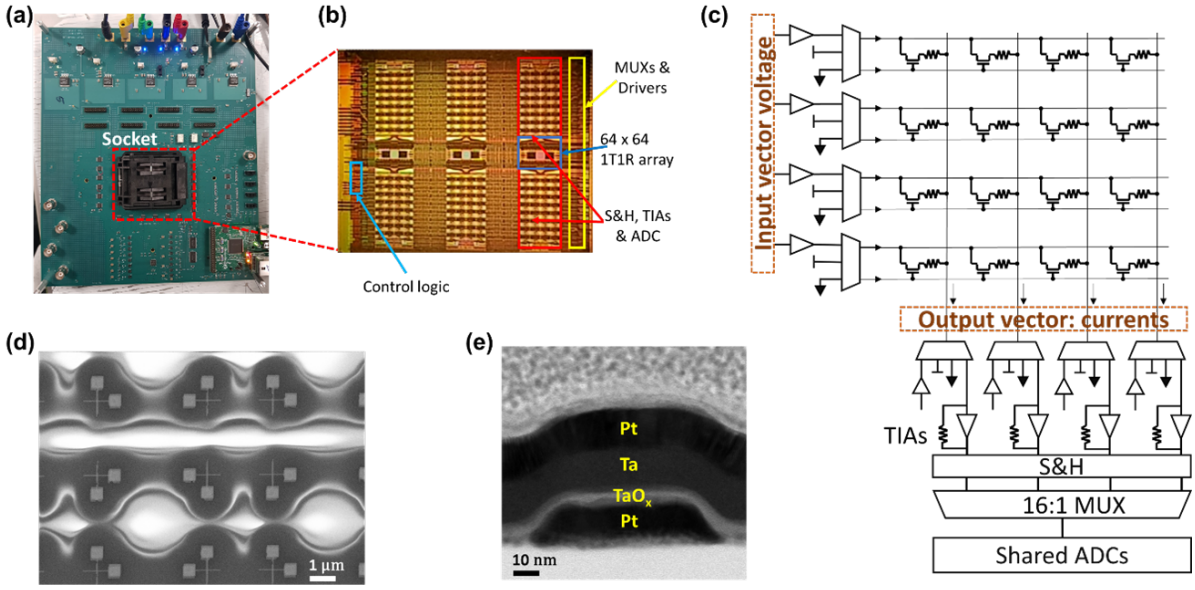


Figure 2.9: **Experimental setup** Hewlett Packard Enterprise's CMOS/memristor prototyping setup [85]. (**a**) Photo of the printed circuit board used for testing the chip. (**b**) Photo of a CMOS die hosting three 1T1R memristor crossbar arrays and peripheral circuitry. (**c**) Circuit schematic of a single crossbar and its peripherals from the CMOS/memristor chip. (**d**) Scanning Electron Microscope (SEM) image of the memristor devices and (**e**) cross-section Transmission Electron Microscopy (TEM) image of the Pt/Ta/TaO$_x$/Pt memristor stack.

A single iteration of the algorithm involves the application of digital voltages $V_{x_i} \equiv x_i V_0$ and $V'_{x_i} \equiv \bar{x}_i V_0$, with $V_0 = 0.2$ V, encoding literals $l_{2i-1} \equiv x_i$ and $l_{2i} \equiv \bar{x}_i$, correspondingly, to the word lines of the crossbar array for the forward computation (Figs. 2.8b and 2.5c). The bit-lines are tied to the ground so that the clause output currents I_{cj} are computed in memory according to Kirchhoff's and Ohm's laws and correspond to dot-products $I_0 \sum_{i=1}^{2N} l_i b_{ji}$, where $I_0 \equiv V_0 G_{on}$ is a unit current via on-state coupling weight while assuming negligible off-state currents ($G_{off} = 0$), and b_{ji} are coupling binary weights between literals l_i and the j-th clause.

In the backward computation step, the word line voltages (V_{cj}) are applied to the second array's rows corresponding to the identified break clauses, while biasing other rows to zero. The bit-line currents of the second array are multiplied by the corresponding

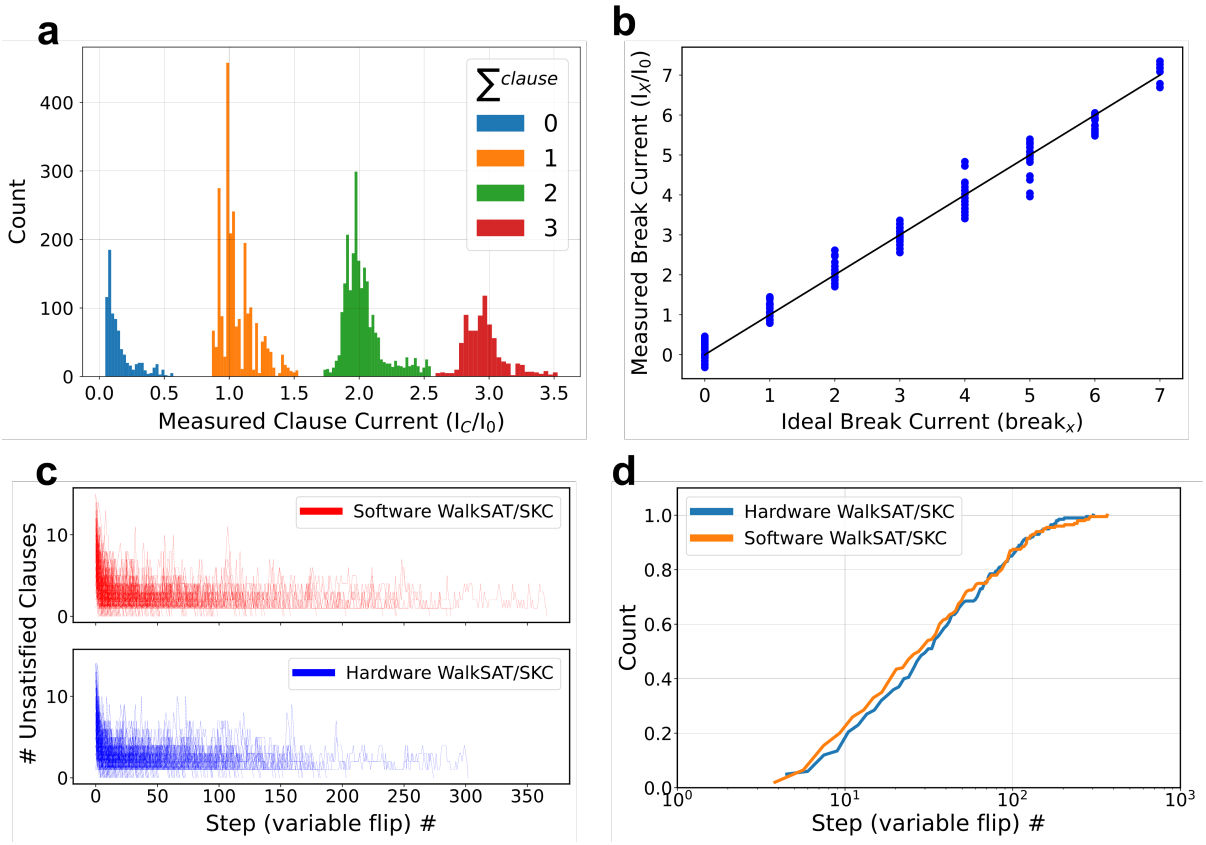


Figure 2.10: **Experimental results.** (**a**, **b**) Measured output (bit-line) currents for (**a**) forward and (**b**) backward passes. In panel (**a**), currents are grouped according to their ideal clause literal sum values. In panel (**b**), outputs are shown against calculated ideal break values. In both panels, data are collected during a single iteration of an algorithm. (**c**, **d**) Functional performance comparison between ideal software model and experiment, showing (**c**) evolution of the number of unsatisfied clauses, and (**d**) run-length distribution curves obtained across 200 iterations (restarts) of the algorithm, each time with new randomly initialized variable assignments. In all experiments, MAX_FLIPS = 10,000.

literal values and normalized to the unit current I_0 to compute break values of variable x_i (Fig. 2.8b and 2.5e), such that

$$break_{x_i} = \frac{x_i I_{x_i} + \bar{x}_i I'_{x_i}}{I_0} = l_{2i-1} \sum_j b_{j,2i-1} z_j + l_{2i} \sum_j b_{j,2i} z_j \tag{2.2}$$

Here, I_{x_i} and I'_{x_i} denote the currents flowing in the bit lines corresponding to the normal (l_{2i-1}) and complementary (l_{2i}) literals of variable x_i. Note that in the performed experiment, the bit-line currents are always converted to the corresponding digital voltages using on-chip circuitry and transmitted out of the chip via a serial peripheral interface so that break-clause checks and encode function (Fig. 2.8b) in the forward pass and heuristics for selecting a variable in the backward pass are performed on the personal computer (Fig. 2.8c).

The experimental demonstration was successful despite hardware nonidealities such as inaccurate tuning of on-state coupling weight conductances (Fig. 2.8c). While measured clause output currents (Fig. 2.10a) in the forward pass and break output currents (Fig. 2.10b) in the backward pass deviated from their ideal values, the margins between adjacent clause currents in the forward pass were sufficient to clearly distinguish break clauses—see, e.g., nonoverlapping histograms for 0, 1, and 2 clause current cases in Fig. 2.10a. The margins were slimmer for the backward pass (Fig. 2.10b), though still large enough for correct operation, in part due to the stochastic nature of the convergence (Fig. 2.10c). Notably, the run-time distribution, i.e., the cumulative probability of finding the solution over algorithm runtime [82] in hardware, follows closely the simulated one (Fig. 2.10d).

2.4 Impact of Crossbar Noise on Gradients

The performance of in-memory mixed-signal computing circuits might be affected by device and circuit non-idealities, especially those of memristors. In light of these concerns, we performed detailed SPICE simulations of a vast ensemble of analog computations using 1T1R crossbars to study the impact of varying maximum order of mapped K-SAT problem, clause-to-variable ratio, problem size, and crossbar dimensions on the accuracy of computed clause currents, break and gain values. The transistor in the 1T1R device was emulated using a linear resistor having resistance 950 Ω or 1 GΩ depending on whether the gate signal was turned ON or OFF respectively, for faster simulation convergence. Memristors were emulated using resistors with conductance values drawn from a normal distribution with mean set to $G_{\text{on}} = 100$ μS and $G_{\text{off}} = 1$ μS and standard deviations 3 μS and 0.25 μS depending on whether 1 and 0 are stored respectively, to reflect memristor programming errors. The standard deviation values were chosen based on experimentally realized programming errors at those conductance levels in [86]. Line resistance of 1 Ω per crosspoint device was considered for emulating bit-line and word-line resistances. The forward and backward crossbars were biased as described in Fig. 2.7 with the read voltage of memristors V_{on} being set to 0.2 V.

Fig. 2.11a-c shows the error (three times the standard deviation) in clause currents in the forward array when K-SAT problems of different maximum order K are mapped to the crossbar, and histograms for specific values of K. Uniform random SAT problems with 14 variables, 64 clauses and $K = 3, 4, 5, 6, 7, 8, 9, 10$ were generated as described in Appendix A.6 and were mapped to a crossbar of dimension 64×64. Literal values corresponding to 500 randomly generated variable assignments were applied as inputs to the crossbar. The number of valid clause current levels increases with K and so does the error at each level. However, since the errors in $\sum_{\text{clause}} = 0$ and $\sum_{\text{clause}} = 1$ and

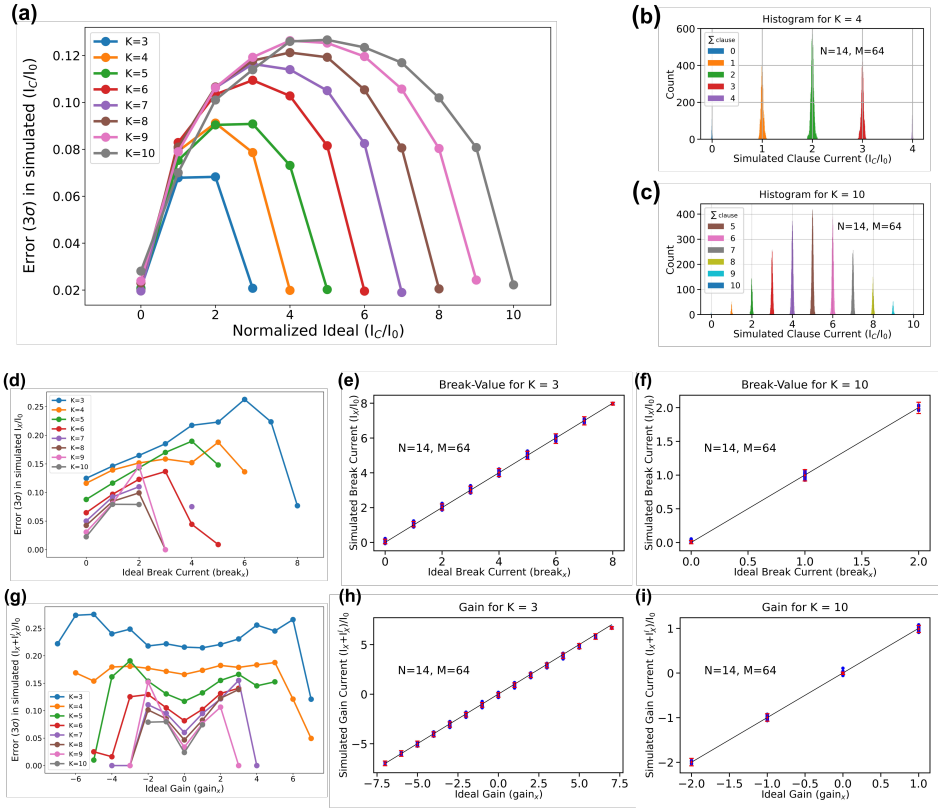


Figure 2.11: (**a**) Plot of three times standard deviation in the simulated normalized clause current for different values of ideal normalized clause current at different values of K (maximum order of SAT CNF mapped to the crossbar). (**b**–**c**) Histogram of the simulated clause current values for (**b**) $K = 4$ and (**c**) $K = 10$. Each K-SAT problem mapped to the 64×64 forward array crossbar has $N = 14$, $M = 64$, and was applied 500 randomly generated literal vector assignments. Memristor conductances were drawn from a normal distribution with mean set to $G_{\text{on}} = 100$ μS and $G_{\text{off}} = 1$ μS and standard deviations $\sigma_{G_{\text{on}}} = 3$ μS, $\sigma_{G_{\text{off}}} = 0.25$ μS. See Supplementary Note 6 for details on simulation setup.(**d, g**) Plot of three times standard deviation in the simulated normalized (**d**) break and (**g**) gain currents for different values of ideal normalized break and gain currents respectively at different values of K (maximum order of the SAT CNF mapped to the crossbar). (**e, f, h, i**) Simulated vs. ideal break current values for (**e**) $K = 3$ and (**f**) $K = 10$ and simulated vs. ideal gain current values for (**h**) $K = 3$ and (**i**) $K = 10$. Each K-SAT problem mapped to the 64×64 backward array crossbar has $N = 14$, $M = 64$. The crossbar received clause select (V_g) and input (V_c) signals (Fig. S10) corresponding to 500 randomly generated literal vector assignments. Memristor conductances were drawn from a normal distribution with mean set to $G_{\text{on}} = 100$ μS and $G_{\text{off}} = 1$ μS and standard deviations $\sigma_{G_{\text{on}}} = 3$ μS, $\sigma_{G_{\text{off}}} = 0.25$ μS.

their overlap region (as well as overlap of $\sum_{\text{clause}} = 1$ and $\sum_{\text{clause}} = 2$) is not significantly degraded, make and break clause indicator signals (determined by whether clause current is equal to zero and one respectively) can be reliably computed even as K increases for the considered problem size.

Fig. 2.11d-i shows the errors in measured break and gain values for problems with different maximum order and histograms for specific values of K. Error in make values is expected to follow a similar trend as break values, as both are subjected to equivalent underlying hardware conditions. Interestingly, the number of valid levels (in break and gain currents) as well as their error decreases with increasing K. The magnitudes of both these currents depend on the make and break literal sums ($\sum_{\text{make}}$ and $\sum_{\text{break}}$), which in turn are proportional to the number of clauses with make and break indicator signals (s_j and z_j) equal to one. Statistically, the probability of a clause having $s_j = 1$ is $\sim 1/2^K$ and $z_j = 1$ is $\sim K/2^K$. Therefore, the number of clauses that contribute to the break and gain currents decreases with increasing K, and hence the decrease in the number of their valid levels. This also results in reduced errors, as there are fewer leakage currents flowing in the bit lines (contributed by OFF-state crosspoint devices with activated word lines and equal to $V_{\text{on}}G_{\text{off}}$).

Another parameter that changes with different high-order optimization problems is the clause-to-variable ratio (M/N). Fig. 2.12 shows the errors in measured break and gain values for problems with different M/N values and histograms for specific values of M/N. Uniform random SAT problems with 64 clauses, $K = 3$ and varying number of variables were generated as described in the Methods section and were mapped to a crossbar of dimension 64×64. Increasing clause-to-variable ratio does not seem to have a significant effect on the number of valid levels or error.

Although all problems considered till now could be mapped to 64×64 crossbar arrays, larger problems would inevitably require larger arrays. Increasing crossbar dimensions

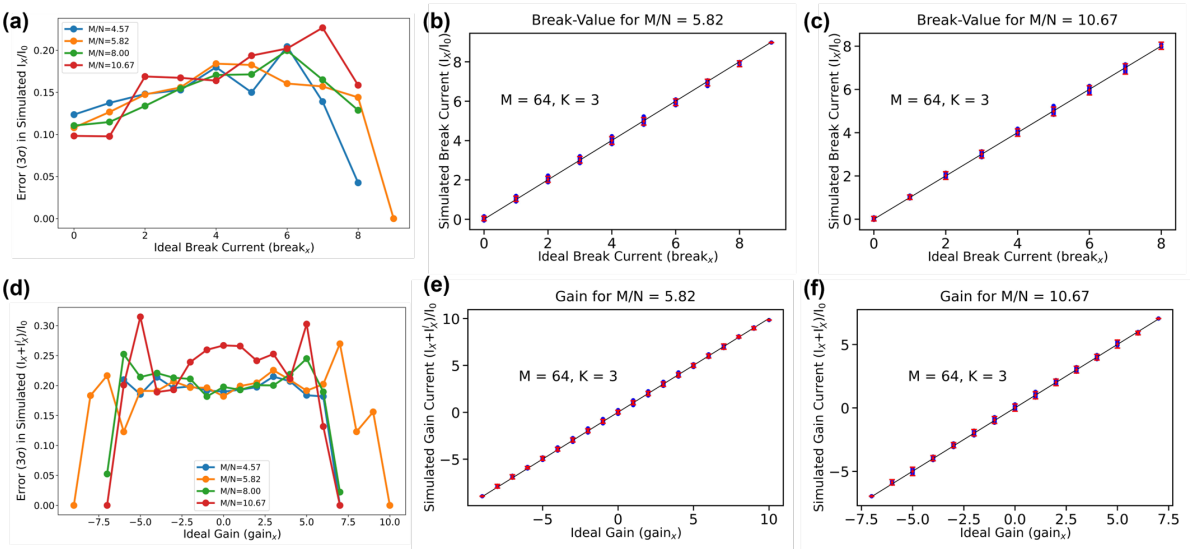


Figure 2.12: (**a, d**) Plot of three times standard deviation in the simulated normalized (**a**) break and (**d**) gain currents for different values of ideal normalized break and gain currents respectively at different values of clause-to-variable ratio (M/N) of the SAT CNF mapped to the crossbar. (**b, c, e, f**) Simulated vs. ideal break current values for (**b**) $M/N = 5.82$ and (**c**) $M/N = 10.67$ and simulated vs. ideal gain current values for (**e**) $M/N = 5.82$ and (**f**) $M/N = 10.67$. Each K-SAT problem mapped to the 64×64 backward array crossbar has $K = 3$ and $M = 64$. The crossbar received clause select (V_g) and input (V_c) signals corresponding to 500 randomly generated literal vector assignments. Memristor conductances were drawn from a normal distribution with mean set to $G_{\text{on}} = 100$ μS and $G_{\text{off}} = 1$ μS and standard deviations $\sigma_{G_{\text{on}}} = 3$ μS, $\sigma_{G_{\text{off}}} = 0.25$ μS.

leads to larger errors due to higher IR drops along the now longer word and bit lines and leakage currents. Fig. 2.13a-c shows the error in clause currents in the forward array for larger problem sizes and crossbar array dimensions and histograms for specific problem sizes. As expected, errors increase with increasing crossbar array and problem size, but overall accuracy of forward operation is not degraded as the thresholds for classifying the $\sum_{\text{clause}} = 0$ and $\sum_{\text{clause}} = 1$ levels are still well defined. Fig. 2.13d-i shows the increase in break and gain value computation error in the backward operation for increasing problem and crossbar sizes. However, errors in the backward operation are on average 3-5 times larger than those in the forward operation. This is because for the considered 3-SAT

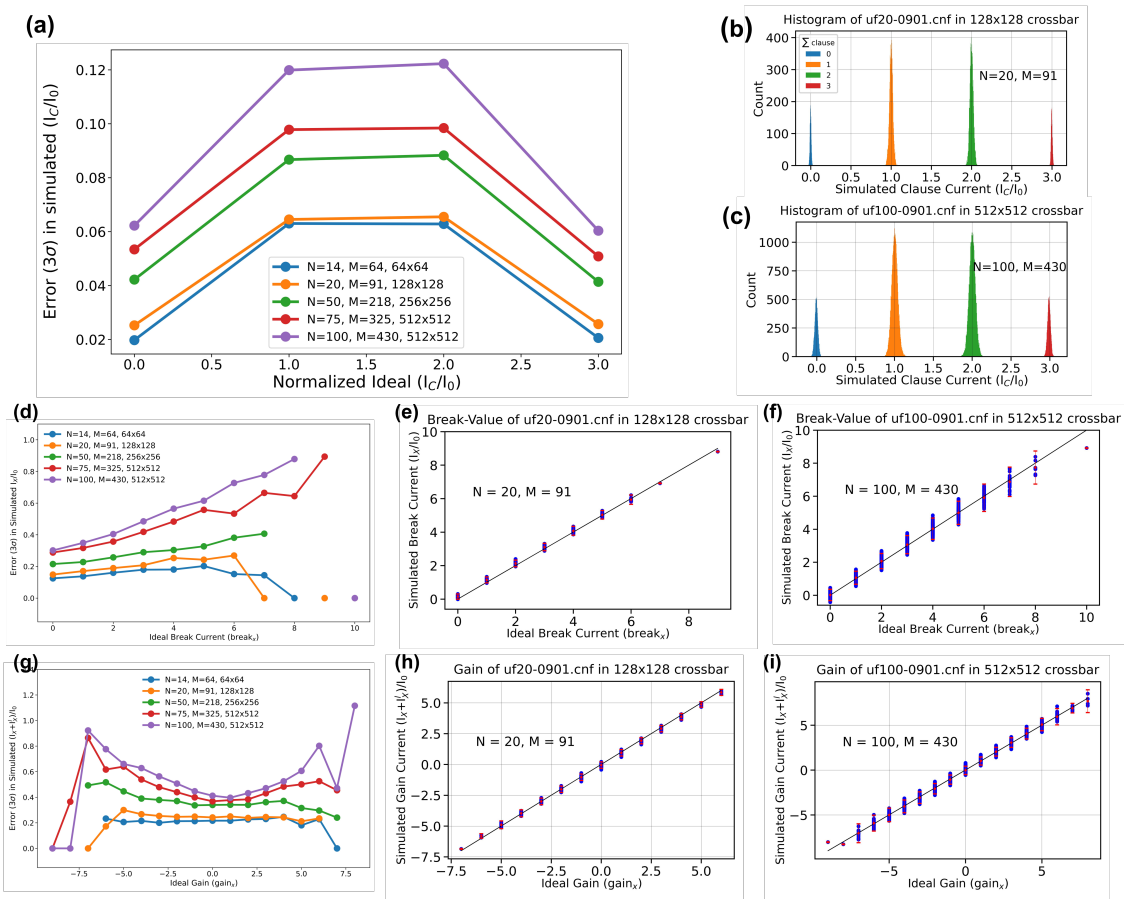


Figure 2.13: (**a**) Plot of three times standard deviation in the simulated normalized clause current for different values of ideal normalized clause current at different forward crossbar array sizes. (**b-c**) Histogram of the simulated clause current values for 3-SAT problems with (**b**) $N = 20$, $M = 91$ and (**c**) $N = 100$, $M = 430$ mapped to crossbar arrays with dimensions 128×128 and 512×512, respectively. For each 3-SAT problem, 200 randomly generated literal vector assignments were applied as inputs. Random 3-SAT instance used for experiments was mapped to the 64×64 crossbar, whereas instances `uf20-0901`, `uf50-0901`, `uf75-051` and `uf100-0901` from SATLIB benchmark [87] were mapped to crossbars with linear dimensions ranging from 128 to 512. (**d, g**) Plot of three times standard deviation in the simulated normalized (**d**) break and (**g**) gain currents for different values of ideal normalized break and gain currents respectively at different forward crossbar array sizes. (**e, f, h, i**) Simulated vs. ideal (**e, f**) break and (**h, i**) gain current values for 3-SAT problems with (**e, h**) $N = 20$, $M = 91$ and (**f, i**) $N = 100$, $M = 430$ mapped to crossbar arrays with dimensions 128×128 and 512×512, respectively. Crossbar arrays received clause select (V_g) and input (V_c) signals corresponding to 200 randomly generated literal vector assignments. Memristor conductances were drawn from a normal distribution with mean set to $G_{\text{on}} = 100$ μS and $G_{\text{off}} = 1$ μS and standard deviations $\sigma_{G_{\text{on}}} = 3$ μS, $\sigma_{G_{\text{off}}} = 0.25$ μS.

problems, each clause in the forward array has at most three ON-state memristors in their bit line, whereas each literal has a much higher number of ON-state memristors (5 to 10). This leads to higher leakage currents contributed by OFF-state memristors with activated word lines ($V_{\text{on}}G_{\text{off}}$). Arrays of size 512×512 that can accommodate uniform random 3-SAT problems of up to 100 variables show much higher errors, with significant overlap between neighboring gain/break current levels.

To study the impact of memristor variation on error in gain and break value computation, we performed backward array simulations with a 128×128 memristor crossbar array corresponding to a uniform random 3-SAT problem with 20 variables and 91 clauses, tuned with different levels of errors (see Fig. 2.14). Both gain and break-value errors are borderline (with maximum value of its three-times sigma close to 0.5) for up to 240% and 2.3% tuning precision (corresponding to $\sigma_{G_{\text{off}}} = 3$ μS and $\sigma_{G_{\text{on}}} = 3$ μS) of the OFF and ON-state memristors respectively.

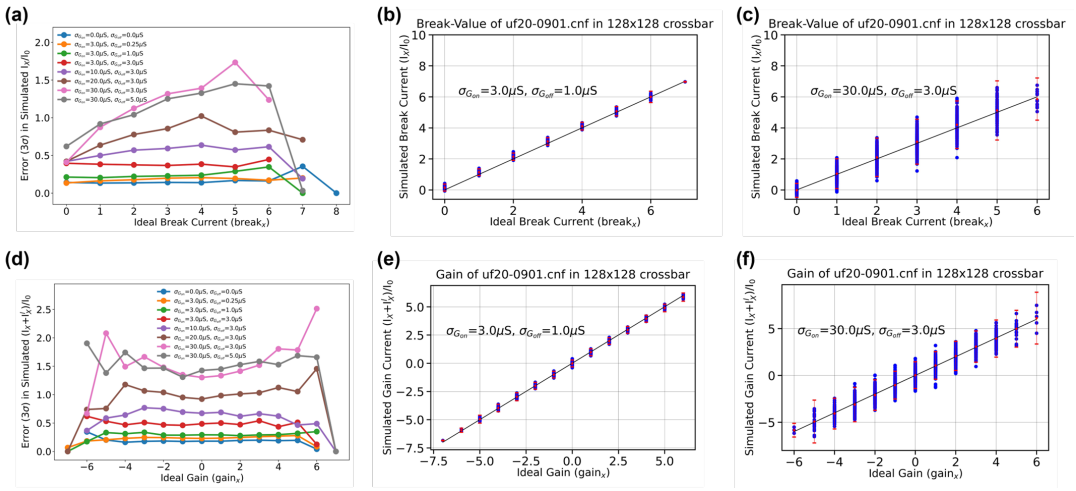


Figure 2.14: (**a, d**) Plot of three times standard deviation in the simulated normalized (**a**) break and (**d**) gain currents for different values of ideal normalized break and gain currents respectively at different levels of memristor programming error. (**b, c, e, f**) Simulated vs. ideal (**b, c**) break and (**e, f**) gain current values for 3-SAT problem mapped to a 128×128 crossbar array with (**b, e**) $\sigma_{G_{\text{on}}} = 3$ μS, $\sigma_{G_{\text{off}}} = 1$ μS and (**c, f**) $\sigma_{G_{\text{on}}} = 30$ μS, $\sigma_{G_{\text{off}}} = 3$ μS. Crossbar arrays received clause select (V_g) and input (V_c) signals corresponding to 200 randomly generated literal vector assignments.

To conclude, simulation results show that the forward operation is more robust compared to the backward one, which can be explained by the simpler functionality of the former, in which only reliable detection of zero and unit clause currents is required. In the backward operation, errors in break and gain values are significantly larger. However, the errors decrease with increasing K, due to the reduced number of clauses with s_j and z_j equal to one, whereas it remains relatively unaffected by increasing M/N. On the other hand, crossbar array dimensions and memristor tuning error significantly reduce dot-product accuracy and increase overlap between neighboring break and gain values. The degradation upon crossbar array scaling is mainly due to larger unwanted leakage currents from nominally OFF-state memristors (with $V_0 G_{\text{off}} \approx 0.2\ \mu$A from a single memristor). Hence, it can be addressed by improving the ON/OFF ratio of the memristor technology. The alternative, more scalable solution to increasing crossbar dimensions for solving larger problems is to rely on a multi-tile architecture with fixed-size smaller crossbar arrays, as discussed in Chapter 6.

2.5 Hardware Modeling of Two High-Order Optimization Machines

To highlight the benefits of high order gradient computation in practical high-order optimization machines, we performed hardware modeling of a WalkSAT/SKC solver (see Appendix A.7) and a High-Order Hopfield Neural Network (HO-HNN) (see Appendix A.8). Figure 2.15a shows the circuit blocks required for implementing WalkSAT/SKC solver hardware that is enabled by our proposed in-memory clause evaluation and break-value computation. Such an implementation follows the ideas laid down in Fig. 2.5 and Fig. 2.7. The forward and backward arrays are driven by $2N$ and M word-line

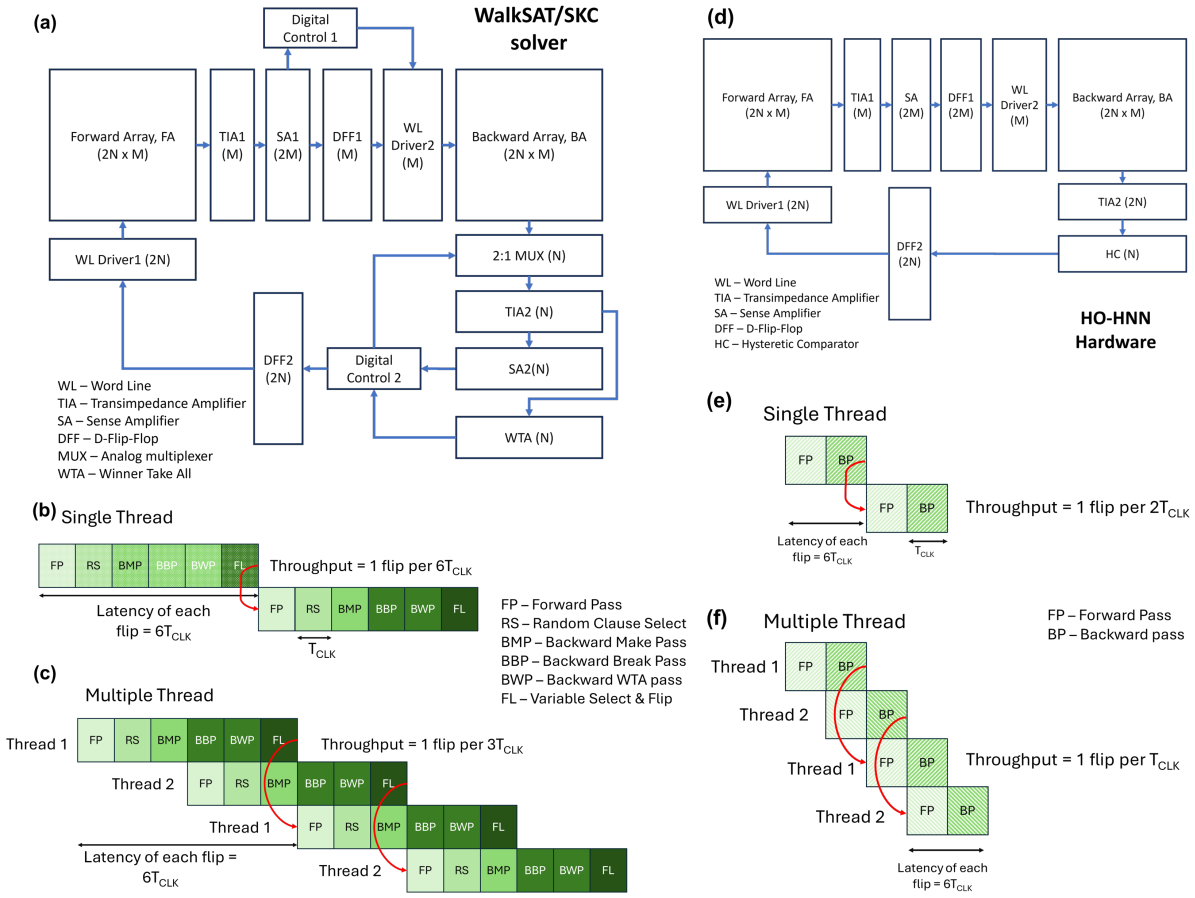


Figure 2.15: (a) Block level diagram of circuit components required for implementation of a WalkSAT/SKC solver. The number inside brackets denote the count/dimension of the circuit block. (b, c) Timing diagrams of the solver when single (b) and two threads (c) are run. Six pipeline stages are required for a single variable flip in any thread. (d) Block level diagram of circuit components required for implementation of a High Order HNN solver. The number inside brackets denote the count/dimension of the circuit block. (e, f) Timing diagrams of the solver when single (e) and two threads (f) are run. Two pipeline stages are required for a single variable flip in any thread.

drivers corresponding to literal and clause indicator inputs respectively. Transimpedance amplifiers are used to convert the bit line currents to voltage, that are then compared against fixed thresholds using sense amplifiers. For the forward array, there are two sense amplifiers per bit line (hence $2M$ total count for SA1s) to evaluate both make and break clause indicator signals in parallel. For the backward array, each variable has a 2-to-1 analog multiplexer to pass either its make or break current for voltage conversion. A voltage-mode analog winner-take-all circuit (see [88]) is implemented to choose the variable with the minimum break value. Register arrays (DFFs) are used to latch clause indicator signals and literal values in each cycle.

Figure 2.15b shows the timing diagram of the WalkSAT/SKC solver for a single thread (a thread corresponds to a unique random variable state initialization). Overall execution is divided into six pipeline stages: FP (Forward Pass), RS (Random Clause Select), BMP (Backward Make Pass), BBP (Backward Break Pass), BWP (Backward WTA Pass) and FL (Variable Select & Flip), where each stage takes one clock cycle to update. In FP, the forward array evaluation takes place as a result of which the make clause indicator signals (s_j) are sent to the digital control logic and the break clause indicator signals (z_j) are sent to be latched by the register array in the next cycle. In RS stage, the forward pass results are latched and one unsatisfied clause is randomly selected in the control logic. In BMP stage, only one word line drive corresponding to the selected clause is activated. The analog multiplexer is configured to pass make currents and the backward array peripherals are used to detect the member variables of the unsatisfied clause. In BBP stage, the z_j values latched in DFF1 are used to drive the backward array and bit line peripherals are configured to compute break values of the member variables. In BWP stage, the same voltage output of the TIAs as obtained in the BBP stage is used to detect the variable with minimum break value using the winner take all circuit. Finally in FL stage, the variable is chosen by the control logic based on the WalkSAT/SKC heuristic

and final values (post-flipping) are latched in the register array (DFF2).

While the backward array is operated during BMP, BBP and BWP stages, the remaining stages are idle. To increase hardware utilization, Fig. 2.15c shows the timing diagram for a pipelined dual-thread implementation of the solver. In this implementation, a second thread (with a different state initialization) is launched three clock cycles after the first is started. As a result, the second thread goes through the FP stage while the first goes through the BBP stage. The latency of each thread's flip is still the same (six clock cycles) but the throughput is doubled. The advantage for such multi-threaded implementations of stochastic local search solvers lies in the fact that the performances of such solvers also depend on the state at which they are initialized. Therefore, a second thread running in parallel increases the probability of converging in a given time by roughly two times (see [89]).

Unlike WalkSAT/SKC that uses only break-values, HO-HNN makes use of full gain information of all variables to update states. Moreover, in contrast to other implementations of hardware HNNs and simulated annealing that use gradient information of single or sub-groups of neurons, our implementations evaluate and uses the gain/gradient information of all neurons in parallel to update neuron states. Such massively parallel gain computation is possible using our proposed hardware paradigm comprising a forward and backward array. Fig. 2.15d shows the circuit blocks required for implementing a HO-HNN hardware that is enabled by in-memory gain computation. In this hardware, we are able to directly map the problem in CNF without converting it to a PUBO. However, problems whose cost function is natively present in the polynomial form, a similar hardware can be used with trivial modifications, where clauses and literals are replaced by monomials and variables respectively (see Fig. 2.2). In the backward array, Hysteretic Comparators (HC) [43] are used instead of sense amplifiers, to compare the gain value against a noisy threshold and guide evolution of the neuron states.

Fig. 2.15e shows the timing diagram of the HO-HNN solver for a single thread. Owing to the rather simpler structure of the HO-HNN algorithm, the execution is divided into just two pipeline stages: FP (Forward Pass) and BP (Backward Pass), where each stage takes one clock cycle to update. In FP, the forward array evaluation takes place and resultantly the make and break clause indicator signals (s_j and z_j) are latched by the $2M$ sized register array. In BP stage, the s_j and z_j values latched in DFF1 are used to drive the backward array and bit line peripherals are used to compute and compare gain values of all variables/neurons against dynamically changing threshold values and update neuron state. To increase hardware utilization, Fig. 2.15f shows the timing diagram for a pipelined dual-thread implementation of the solver. The latency of each thread's flip is two clock cycles (same as single thread implementation) but the throughput is doubled (one flip per clock cycle). A second thread running in parallel increases the probability of converging in a given time by roughly two times. The operating frequency of both solvers is set to 500 MHz ($T_{\text{clk}} = 2$ ns). See Appendix A.9 for details about the circuit modeling.

We performed hardware aware simulations of the two algorithms by modeling the break-value/gain computation error results for different crossbar sizes and memristor programming errors. Scaling trends in time and energy to solution were obtained by solving uniform random 3-SAT instances (see Appendix A.9 for details). Fig. 2.16a shows the scaling trend in batch TTS with problem size for the two heuristics when ideal and realistic memristor programming errors are considered, whereas Fig. 2.16b shows the impact of different levels of memristor programming error on batch TTS for 20-variable SAT problems. For larger problems ($N = 100$), there is 8% drop in batch TTS (HO-HNN heuristic) when memristors are programmed with 2.5% and 20% error margins for high and low conductance regimes respectively, compared to the case with 0% memristor programming error margins. For higher memristor programming error

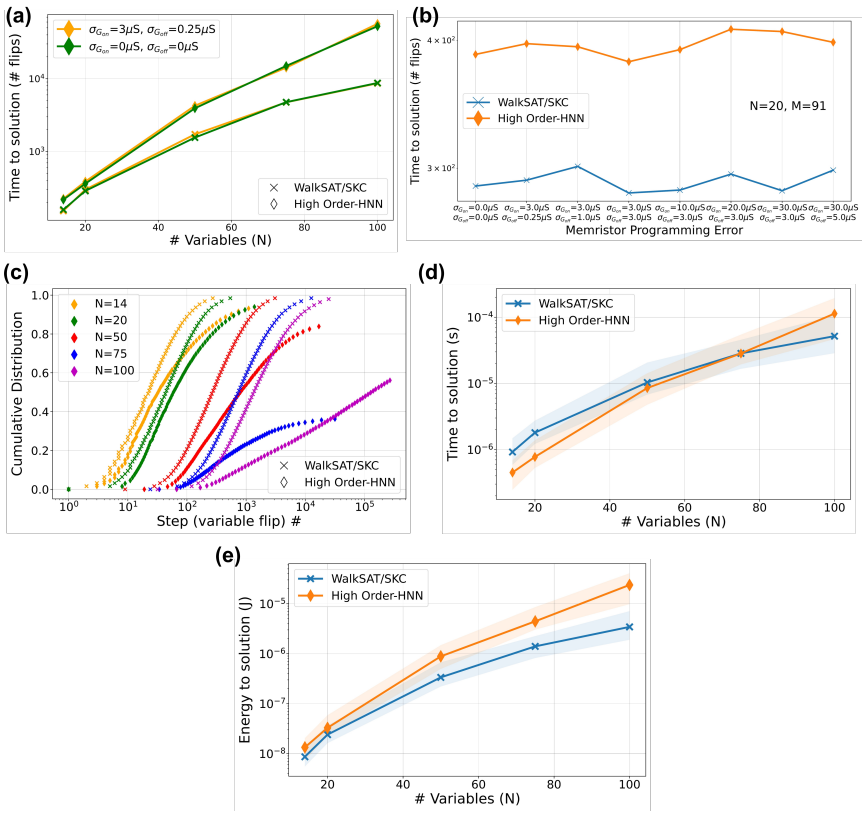


Figure 2.16: (a) Scaling trend of time to solution (in number of variable flips) for WalkSAT/SKC and High Order HNN algorithms with problem size, for ideal and non-ideal memristor programming. (b) Plot of time to solution for different levels of memristor programming error on 20-variable 91-clause uniform random 3-SAT problem instances. (c) Cumulative distribution function for the number of variable flips required by WalkSAT/SKC and high-order Hopfield Neural Network algorithms to reach global minimum across all iterations (up to MAX_ITER) for all instances of 3-SAT problems with varying size. The number of instances for problems with $N = 14, 20, 50, 75$ and 100 were $80, 80, 80, 50$ and 80 respectively. For both algorithms, MAX_ITER was set to 500. The total number of datapoints for each problem size and a specific algorithm used to generate the plot is equal to the number of instances of that problem size times MAX_ITER, which is 25,000 for $N = 75$ and 40,000 for all others. However, only 63 data points sampled regularly from the sorted list of run-length distribution of each problem size across all its instances and iterations per instance, are plotted for clarity. Note that datapoints correspond to successful iterations only. (d) Time and (e) energy to solution of the WalkSAT/SKC and high-order HNN solvers for random uniform 3-SAT problems. The shaded area in the plot shows interquartile range (25–75%) and the markers indicate the median across instances of that problem size. Hardware-aware algorithmic simulations are taking into accounts errors in break/gain values due to memristor tuning errors, with the assumed relative value of $\sim 2.4\%$ and $\sim 20\%$ (corresponding standard deviation for the conductance tuning of 3 μS and 0.25 μS) for the on- and off-state memristors, respectively.

margins (16% and 235% for high and low conductance respectively), there is up to 6% drop in batch TTS (HO-HNN heuristic) at even smaller problem sizes with $N = 20$. Both heuristics seem to be robust to computation errors, potentially due to requirement of binary weights (resulting in larger margins) and the inherent navigational properties of the heuristics themselves. For example, with WalkSAT/SKC, good performance has been observed to be associated with successfully being able to flip a variable with zero break-value when present as well as the accuracy of the step where the variable with minimum break-value is flipped. Although computation error in break-values increases with larger crossbar sizes and memristor programming tolerances, the probability that a variable with break-value = 0 is interpreted as that with break-value = 1 is relatively low. Moreover, even if it does fail to detect presence of variable with break-value = 0, the minimum break-value detection step might be able to flip the correct variable (since its computed break-value is still low compared to other variables).

All data points in Fig. 2.16a-c are represented in number of variable flips, without considering any hardware aspects like cycle time, pipeline stages and single/multiple thread implementations. The Time-to-Solution (TTS, see Appendix A.9 for definition and details) in number of variable flips can be converted to hardware TTS (in seconds) using equation 2.3, where t_{clk} is the cycle time, speedup_factor is equal to the number of threads and # of pipeline stages is set to six and two for WalkSAT/SKC and HO-HNN respectively.

$$\text{TTS (in seconds)} = \text{TTS (in \# of variable flips)} \times t_{\text{clk}} \times \frac{\text{\# of pipeline stages}}{\text{speedup\_factor}} \tag{2.3}$$

All data points in Fig. 2.16d represent hardware TTS (in seconds) with $t_{\text{clk}} = 2$ ns and speedup_factor = 2 (corresponding to multi-threaded execution). The expression for

Energy-to-Solution (ETS, in Joules) plotted in Fig. 2.16e is shown in equation (2.4), where E_{avg} is the average energy (in Joules) spent per clock cycle by the respective heuristics.

$$\text{ETS (in Joules)} = \text{TTS (in \# of variable flips)} \times \frac{\text{\# of pipeline stages}}{\text{speedup\_factor}} \times E_{\text{avg}} \quad (2.4)$$

The expression for E_{avg} for WalkSAT/SKC and HO-HNN are given by equations (2.5) and (2.6), respectively.

$$E_{\text{avg}}^{(\text{WalkSAT/SKC})} = \text{speedup\_factor} \times \frac{E_{\text{FP}} + E_{\text{RS}} + E_{\text{BMP}} + E_{\text{BBP}} + E_{\text{BWP}} + E_{\text{FL}}}{\text{\# of pipeline stages}} \quad (2.5)$$

$$E_{\text{avg}}^{(\text{HO-HNN})} = \text{speedup\_factor} \times \frac{E_{\text{FP}} + E_{\text{BP}}}{\text{\# of pipeline stages}} \quad (2.6)$$

The stage-specific energy values $(E_{\text{FP}}, E_{\text{RS}}, E_{\text{BMP}}, E_{\text{BBP}}, E_{\text{BWP}}, E_{\text{FL}})$ and $(E_{\text{FP}}, E_{\text{BP}})$ for WalkSAT/SKC and HO-HNN respectively, are obtained by summing up the energy spent in the different hardware components that are active in that stage scaled by the size/count of those components (see Appendix A.9).

2.6 Real-valued gradient computing hardware

The proposed approach can be extended to computing gradients of functions with real-valued variables (Fig. 2.17). Let's first note that in the simplest case of multi-linear polynomial functions, a partial derivative with respect to a given variable equals to a sum of monomials that such a variable is a member of, divided by the value of that variable

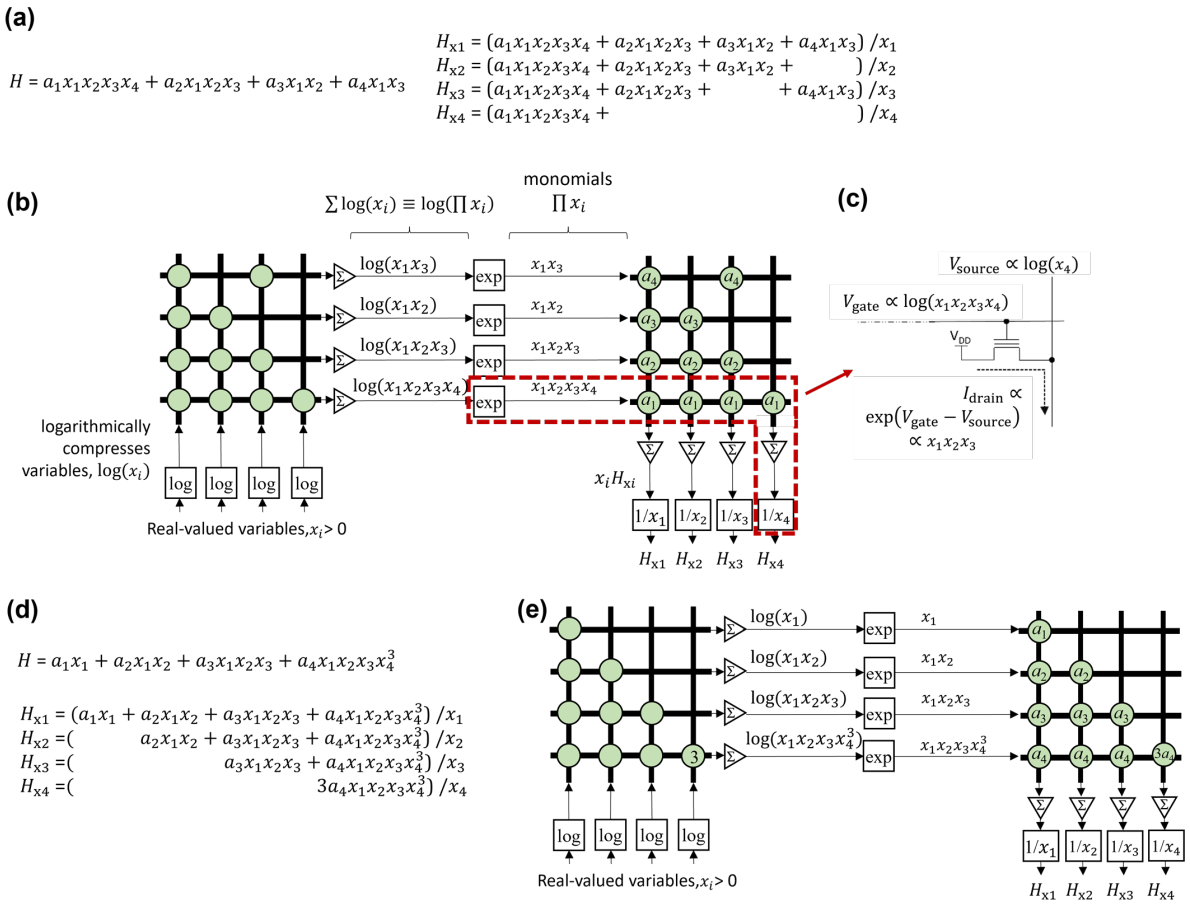


Figure 2.17: **Gradient computation of real-valued functions.** (a) Considered multi-linear polynomial energy function of real-valued variables and its partial derivatives. (b) The main idea and (c) circuit implementation with floating gate memories. In such an implementation, the exponentiation and division by variable $(1/x)$ blocks are replaced with coupling weights consisting of floating gate memories operating in a subthreshold regime. Logarithmically encoded monomial values from the first array and log-encoded variable values are directly applied to the second array's gate and source voltage lines. As a result, division and exponentiation operations are executed by virtue of the exponential dependence of cell current on the difference of applied gate and source voltages. (d) Energy function and its partial derivatives and (e) crossbar circuit implementation for a more general polynomial (non-multi-linear). Note that a term with x_4 in the third power of the energy function is implemented with multi-state weights in the first crossbar and factor of 3 in the second crossbar.

(a)

$$H = a_1 x_1 x_2 x_3 x_4 + a_2 x_1 x_2 x_3 + a_3 x_1 x_2 + a_4 x_1 x_3$$

(b)

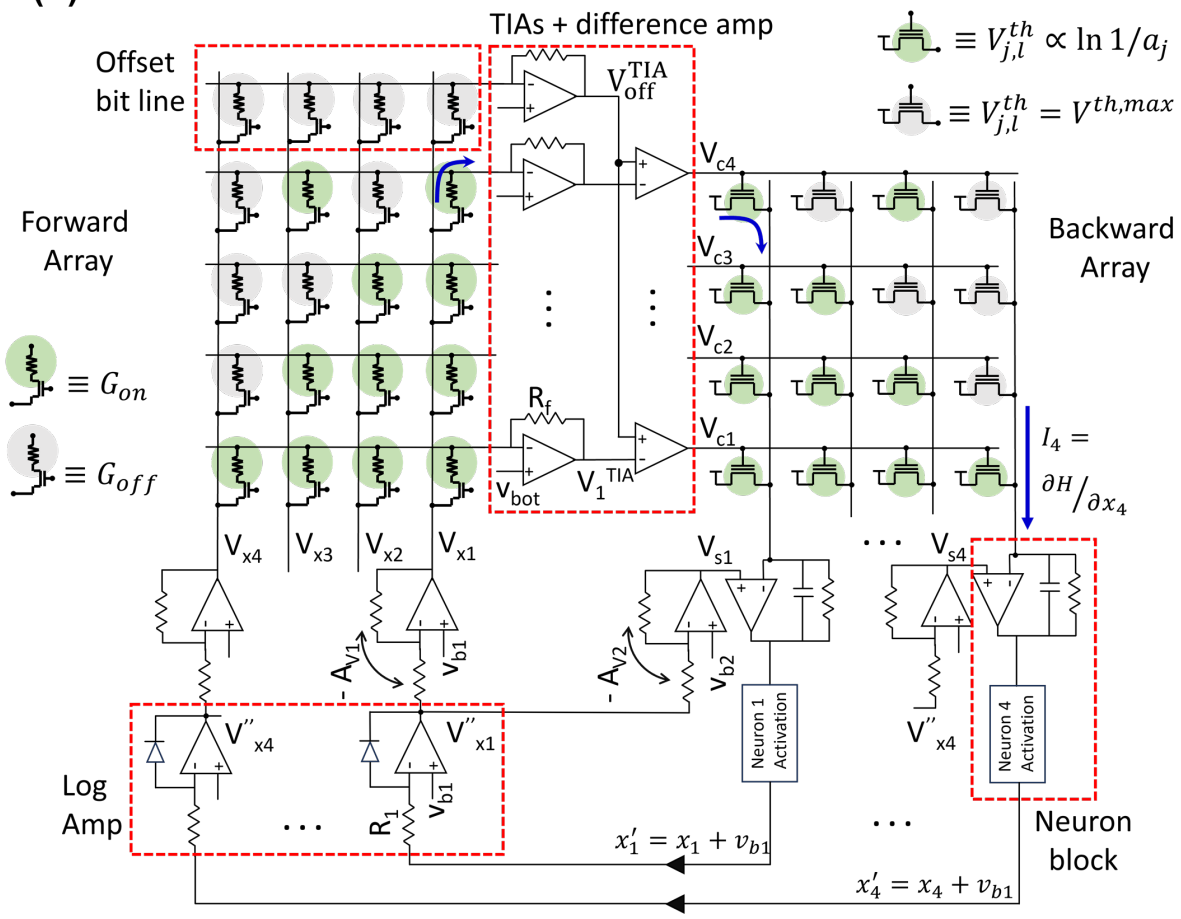


Figure 2.18: **Possible circuit implementation of real-valued gradient computing hardware.** (a) Considered multi-linear polynomial energy function of real-valued variables and (b) possible circuit implementation of its real-valued gradient computing hardware. Forward and backward arrays comprise of linear and exponential crosspoint devices: memristors operating in its linear regime and floating-gate memory cells operating in its subthreshold regime respectively.

(Fig. 2.17a). Partial derivatives are computed in parallel to follow these steps. Like the binary variable function, the first step involves an $N \times M$ crossbar array with similarly configured binary memory weights to compute in-memory monomial products. Due to real-valued variables, the products are computed differently by applying logarithmically encoded variables and exponentiating the outputs (Fig. 2.17b). Appropriately weighted and specific (to the considered variable) monomial terms are summed up in the second array based on the multi-bit memory devices. Finally, the outputs from the second array are divided by variable values. Naturally, the division operation requires the variable to be nonzero and may require a preprocessing step of shifting the variable ranges. Generalization beyond multi-linear function requires analog weights in the first array, and adequately adjusting weights in the second array (Fig. 2.17d,e). Figures 2.17c shows the main idea of floating gate memory device implementation of exponentiation, while Figure 2.18 shows a possible detailed circuit implementation of the real-valued gradient computing hardware. The forward and backward arrays comprise of crosspoint devices with linear and exponential input-to-output characteristics respectively. Memristors operating in their linear regime (with read voltage < 0.2 V) and floating-gate memory cells operating in their subthreshold regime are chosen for the forward and backward crosspoint devices respectively, for this specific case. Using floating-gate memory or any other device with exponential trend in their input-to-output characteristics, enables performing the exponent operation locally at each crosspoint device. Devices with linear trends would necessitate addition of an anti-log amplifier at the input side of the backward array. For the forward array, memristors are chosen for their rather simpler (enabled by Ohm's law) resistive behavior. However, it is noted that with some peripheral tweaks, even floating-gate cells can be used as linear crosspoint devices in the forward array (see [90]). Note that the neuron blocks (in Fig. 2.18b) are only shown for completeness and not explicitly studied in this work. In an Ising machine with continuous-time dynamics,

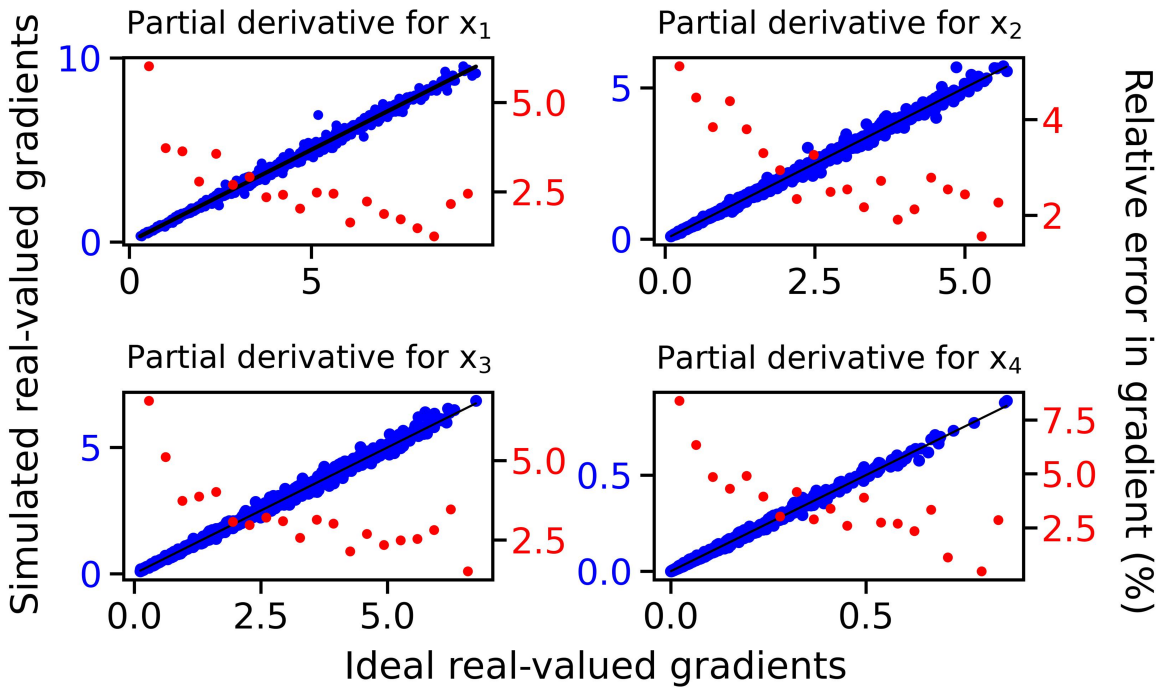


Figure 2.19: **Simulation result of real-valued gradient computation hardware.** The left axis shows plot of the simulated real-valued gradient depicted by the bit line current in the backward array vs ideal gradient values. The right axis shows plot of the relative error in the gradient values. The error is larger for smaller gradients mainly due to the assumed fixed absolute tuning error for floating gate memories, i.e., larger relative error for smaller weights in this modeling study.

these blocks determine the update mechanism/rule of the variables. One potential implementation of such a neuron block comprises of a leaky integrator followed by a nonlinear activation (see [33]).

The circuit operation analysis shows a trade-off between the maximum degree (K) and bit-precision of the monomial coefficients that can be implemented with the fixed dynamic range of the polynomial's variables (Appendix A.10).

Fig. 2.19 shows simulation results for real-valued partial-derivatives with respect to each variable in the polynomial described in Fig. 2.18a. The abscissa shows the ideal partial-derivative values whereas the left and right ordinates show the measured partial-

	Coherent Ising Machine (CIM)	D-Wave 2000Q	sparse Ising Machine (sIM)		mem-SO-HNN	Augmented Ising Machine (AIMs)	This Work	
			FPGA sIM	Nanodevice sIM			WalkSAT/SKC	HO-HNN
Spin representation	Coherent light	Superconducting qubits	Digital bits	CMOS-MTJ p-bit	Digital bits	Analog charge based	Digital bits	
Coupling representation	Coupling matrix in FPGA	Flux storage	Sparse coupling matrix in FPGA		Memristor crossbar	Custom CMOS coupling cell	Memristor crossbar	
Connectivity	All-to-all	Sparse	Sparse	Sparse	All-to-all	Sparse	All-to-all	
Dynamics	discrete-time	continuous-time	discrete-time	continuous-time	discrete-time	continuous-time	discrete-time	
Interaction	High order	Second order	Second order	Second order	Second order	Third order	High order	
Frequency	1GHz		30MHz	1GHz	500 MHz		500 MHz	
Time-to-Solution (TTS)	388.9us (N=100)	44ms (N=20)	*<L> = 2555.09s (N=100)	*<L> = 77.77s (N=100)	12.12ms (N=100)	73.8us (N=100)	1.8us (N=20) **51.8us** (N=100) ***<L>=22us**(N=100)	**0.77us** (N=20) 113.4us (N=100)
Power	50W	25kW	75W	**38.7mW**	317.2mW	300mW (N=500)	**13.35mW** (N=20) 66.4mW (N=100)	42.4mW (N=20) 209mW (N=100)
Energy-to-Solution (ETS)	19.445mJ (N=100)	1.1kJ	191.632kJ (N=100)	3J (N=100)	3.84mJ (N=100)		**24.03nJ** (N=20) **3.43uJ** (N=100) ***<E>=1.46uJ** (N=100)	32.6nJ (N=20) 23.7uJ (N=100)
Area	1 km fibre ring cavity	>10m^2 room			0.0224 mm^2	6.76 mm^2	0.0214 mm^2 (N=100)	**0.0197 mm^2** (N=100)
Solutions per second per watt	51.427	$9x10^{-4}$	$5.21x10^{-6}$	0.33	260.114		**$4.1x10^7$** (N=20) **$3x10^5$** (N=100)	$3x10^7$ (N=20) $4.2x10^4$ (N=100)
Solutions per second per mm^2					$3.7x10^3$	$2x10^3$	**$9x10^5$**	$4.5x10^5$

Figure 2.20: Comparison of proposed in-memory high-degree gradient computation hardware with current state-of-the-art accelerators. Other accelerators include Coherent Ising Machine [91], D-Wave's 2000Q quantum annealer [92], sparse Ising Machines (sIM) [93], memristor crossbar based second-order discrete-time HNNs (mem–SO-HNN) [43, 37] and the Augmented Ising Machine (AIMs) [78].

derivative and the relative percent error respectively. All parameter values used in the simulations are detailed in Fig. A.4. The simulations were done based on behavioral modeling of all hardware components described in Fig. 2.18b except for the neuron blocks. Error in the gradient computation is attributed to the modeled tuning errors of the memristors and floating-gate cells used in the forward and backward arrays respectively. Simulation results show relative real-valued gradient error well below 10%, even without circuit optimization (Fig. 2.19).

2.7 Summary

Fig. 2.20 and Appendix A.11 provide details of comparison with other approaches including the Coherent Ising Machine [91], D-Wave's 2000Q quantum annealer [92], sparse Ising Machines (sIM) [93], memristor crossbar based second-order discrete-time HNNs (mem-SO-HNN) [43, 37] and the Augmented Ising Machine (AIMs) [78]. WalkSAT/SKC outperforms high-order HNN in both hardware time and energy-to-solution metrics for larger problems. This is not surprising given native CNF encoding of the studied problems, which is more efficiently exploited by WalkSAT/SKC heuristics. Most importantly, solvers based on our proposed approach are at least $\sim$7.7 times faster than other discrete-time solvers and three orders of magnitude more energy efficient than the memristor-based second-order HNN, owing to compact hardware footprint and higher navigational efficiency of native high-order solvers. Moreover, the proposed hardware has at least two orders of magnitude higher throughput per watt and throughput per unit area compared to all other technologies.

Note that a direct comparison to a recent work on high-order IMs [35] is challenging because of the lack of hardware implementation details. Also, while AIMs [78] was specifically developed to solve 3-SAT problems, a crude analysis for extending it to support K-order problems with M clauses (by assuming an implementation with $M \times N$ array of unit cells, each hosting $K-1$ $N:1$ multiplexers) reveals inferior worst-case $\sim KN^2M$ complexity scaling (as opposed to $\sim NM$ of our approach). It is also worth noting that a forward pass described in Fig. 2.5c is similar to earlier work on clause evaluation with content addressable memory implementations [94, 95, 96]. Indeed, the critical operation in the content addressable memory computation is in-memory vector-by-matrix multiplication between binary weights and inputs that produce binary "match" outputs. However, our approach takes advantage of all the outputs, not just binary match values.

This feature and, more importantly, backward pass are key novelties of our approach, enabling for the first time massively parallel in-memory computing of make and break values of CNF-form Boolean functions and, more generally, pseudo-gradient and gradient computation in functions of binary and real-valued variables respectively.

In summary, the presented hardware paradigm of computing high-degree polynomials in memory shows great promise for various applications relying on optimization algorithms, especially combinatorial optimization. Its versatility and scalability could enable implementation of more advanced and powerful discrete-time heuristics, like G2WSAT [97] for solving constraint satisfaction problems (some examples of which will be presented in the subsequent chapters). Although its capability of computing real-valued gradients could pave the way towards implementing large-scale continuous-time continuous-state dynamics-based solver hardware potentially much faster than their discrete-time counterparts, we shall focus on the discrete-time implementations in the remainder of this thesis.

Chapter 3

A 55nm K-SAT Solver running WalkSAT Algorithm

3.1 Background

The sparse graph structure of most practical, hard K-SAT problems and constant data transfer between the compute and memory blocks incurred in SAT solvers make it difficult to solve large problems efficiently on conventional digital computers. This has led to increasing efforts towards designing dedicated hardware for SAT solving [96, 98, 93, 99]. A purely digital solver [96] implementing WalkSAT, a SOTA local search algorithm, used SRAM-based In-Memory Computing (IMC) array to accelerate parallel clause evaluation and map arbitrary-order K-SAT problems. However, it computes gradients (specifically the break-value) of variables in a sequential manner, thereby limiting its performance. Furthermore, updating multiple variables in a single step, either using chromatic block updates [98, 93] or gradient-informed stochastic parallel flips [100, 99], have been shown to improve solution times for parallel hardware. While solvers in [98, 100, 99] demonstrated multi-variable update, they were limited to solving only 3-SAT problems. Pre-processing

steps that reduce clause order by introducing additional variables and clauses can be implemented to map higher-order ($K > 3$) problems to such solvers. However, this comes at the cost of exponential increase in the search space and poor solution times and solvability [35]. Recently, in [69] (refer to Chapter 2 for details), a massively parallel approach to computing gradients of variables in high-degree polynomials was proposed. Building upon that, a fully integrated mixed-signal integrated circuit was designed and subsequently taped out. It incorporates the in-memory based parallel clause evaluation and the parallel break-value computing techniques to accelerate a polular SAT-solving algorithm known as WalkSAT [83] (also see Appendix A.7) and maps arbitrary-order K-SAT problems in their native high-order space.

3.2 Key Features

One of the key features of our integrated circuit is that it used the ideas presented in [69] to arbitrary-order SAT problem in its native form without order reduction. Variables and clauses are associated with columns and rows of crossbar array respectively (similar to Fig. 2.5). The crossbar memory element used is a custom 10-transistor SRAM cell. This cell enables decoupled and orthogonal access lines to the crossbar, that allows both the forward and backward passes (as defined according to Fig. 2.5) to be executed in the same physical array. For example, if a 8-transistor SRAM cell was to be used, it would allow only one of those to be executed in a single array comprising such a cell. This improves area efficiency by 1.6×. Voltage-mode sensing is implemented, motivated by their low peripheral overheads and that their nonlinearity is not a major cause of concern for SAT-solving. This is because of the relatively low relevance of high-valued break/make/clause values in uniform random SAT-solving (see Figures. 2.11,2.12,2.13 and 2.14). During the backward pass operation, the chip allows massively parallel break-

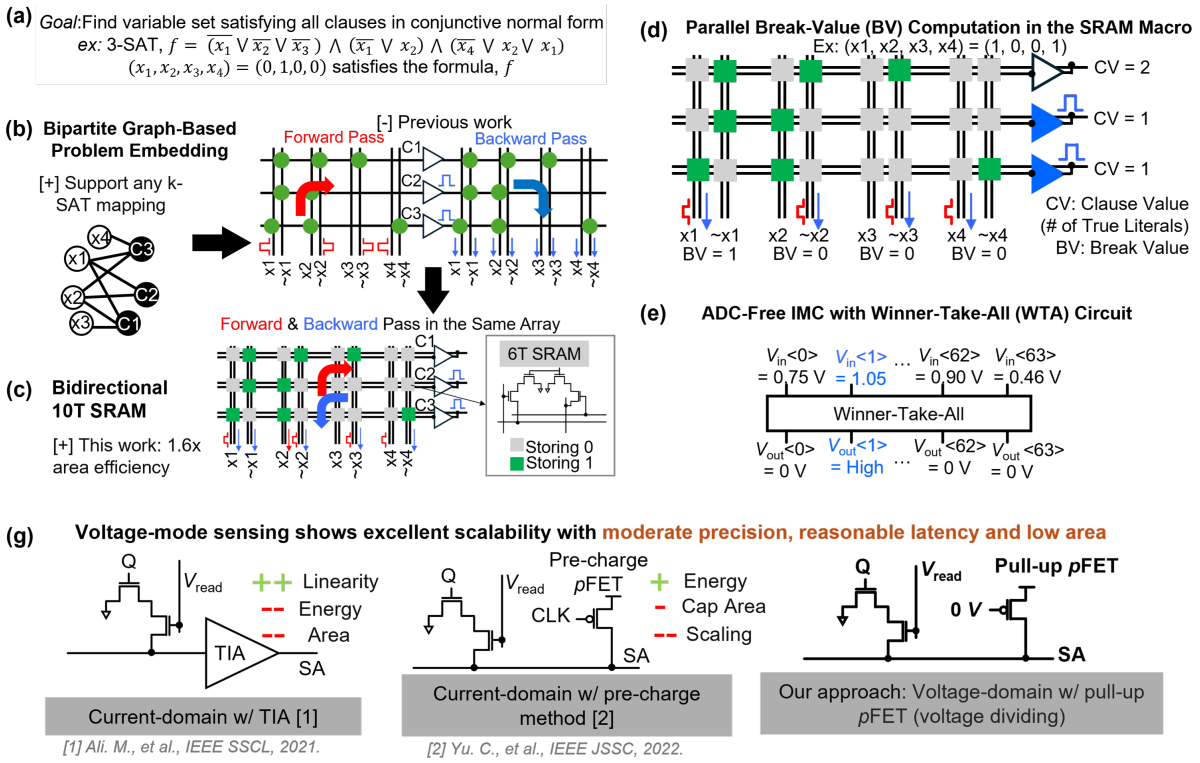


Figure 3.1: Summary of key features of our chip; (a) Example of SAT problem used for illustration, (b) Bipartite graph-based problem embedding that enables any-K-SAT mapping, (c) Bi-directional 10-SRAM cells allowing both forward and backward pass in single array, (d) parallel break-value computation, (e) Winner-Take-All circuit enabling ADC-free in-memory computing, (f) Voltage-mode sensing.

value computation, enabling fast implementation of the WalkSAT algorithm. Finally, it integrates analog-input Winner-Take-All (WTA) circuit for selecting promising variables, resulting in ADC-free in-memory computing. The key features of our chip is summarized in Fig. 3.1.

Our hardware platform allows operating the chip in two modes: (i) Integrated mode: where the control sequence for the WalkSAT algorithm is embedded in the main finite state machine of the chip; (ii) Hybrid mode: where off-chip ADCs and a host FPGA / PC is used for sensing and implementing the control sequence specific to a heuristic, respectively, but the chip's in-memory computing circuits are used for gradient compu-

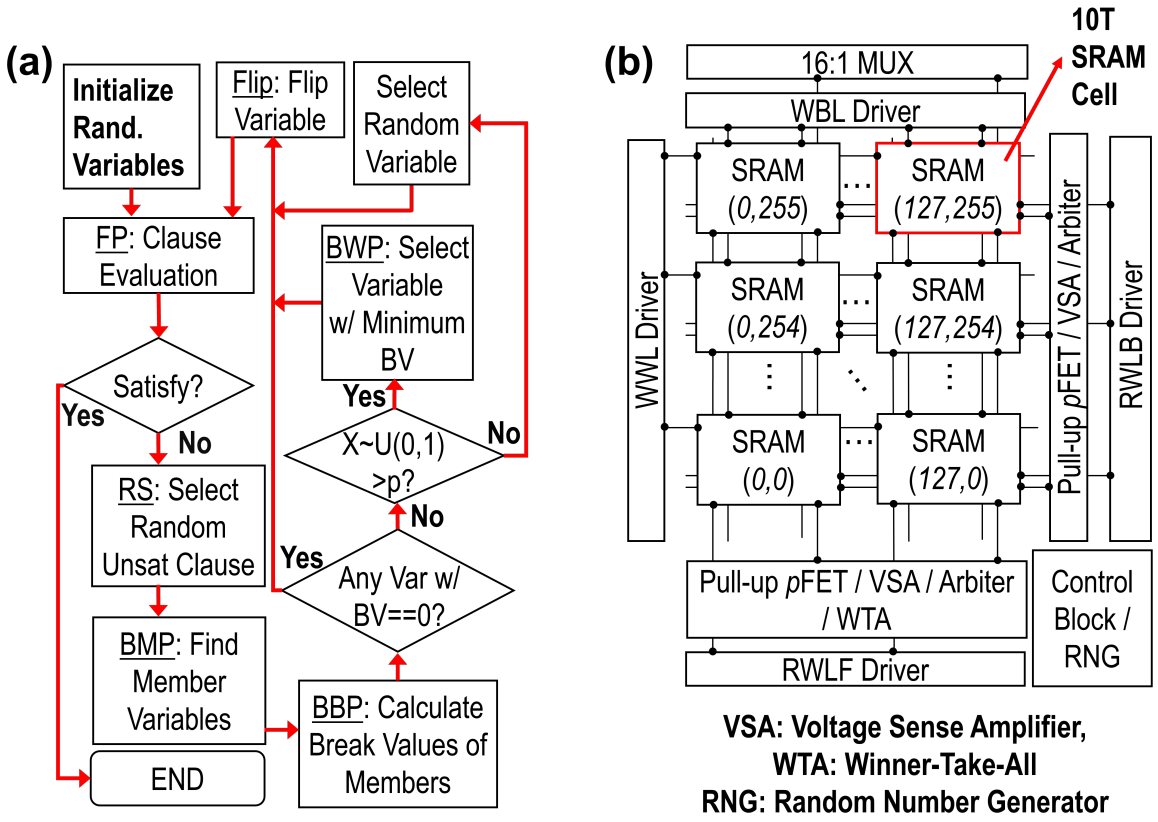


Figure 3.2: (a) Overall workflow of the algorithm implemented in our chip in the integrated mode, (b) Top-level architecture of the chip.

tation (the most expensive operation in optimization heuristics). This mode is primarily meant to explore and invent advanced heuristics that are compatible with our chip's architecture and can be incorporated into the integrated mode of a future prototype.

3.3 Architecture and Schematics

The integrated-mode's workflow (following the WalkSAT algorithm) is shown in Fig. 3.2a whereas the top-level architecture is shown in Fig. 3.2b. The first step is mapping the problem to the SRAM array, where columns represent literals and rows represent clauses. The SRAM cell at location $(2i, j)$ or $(2i{+}1, j)$ is written with a "1" when the

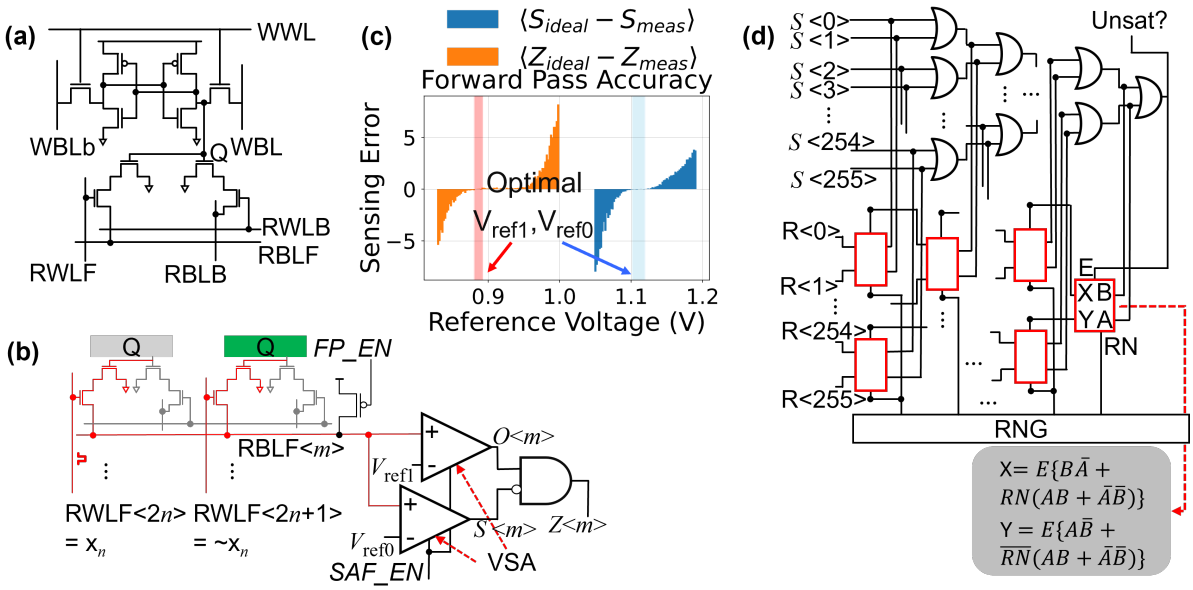


Figure 3.3: (a) 10-Transistor Custom SRAM cell schematic, (b) Forward pass operation highlighted in a row with relevant peripheral circuits, (c) experimental characterization of forward pass operation's sensing error and calibration of reference voltages V_{ref0} and V_{ref1}, (d) Arbiter circuit for selecting one element randomly out of N elements.

normal or complementary literal of a variable x_i is present in a clause C_j; otherwise, with a "0". Subsequently, all variables of the problem are initialized to random values, after which the control circuit iteratively updates/flips a single variable in a randomly selected unsatisfied clause, based on whether it is a promising variable (Break-Value equal to zero and/or least among other variables in that clause) or at random. This iterative process has six steps; Forward Pass (FP), where all clauses are evaluated in parallel using current variable assignments. If all clauses are satisfied, the solver terminates, otherwise continues to the next step; Random Select (RS), where an unsatisfied clause is selected at random; Backward Make-Pass (BMP), where the member variables of the randomly selected clause is obtained using a backward pass operation; Backward Break-Pass (BBP), where the break-values of all variables is computed using a backward pass operation; Backward Winner Pass (BWP), where the WTA circuit is used to choose the

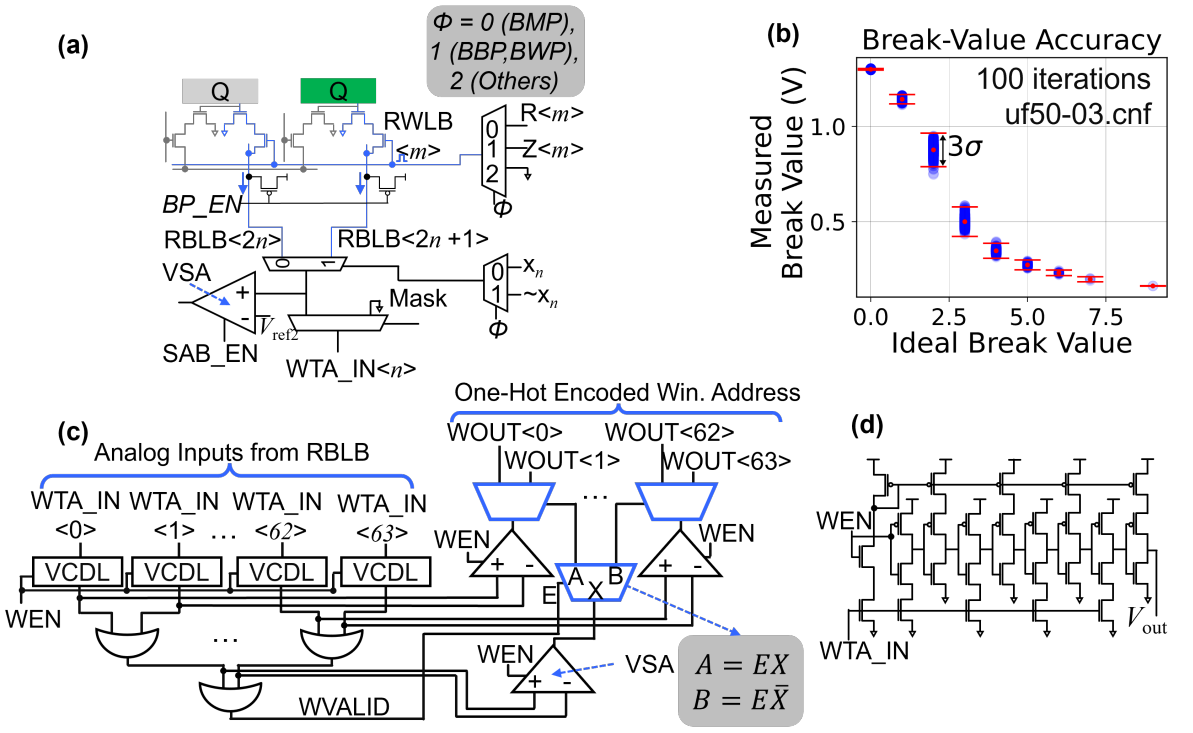


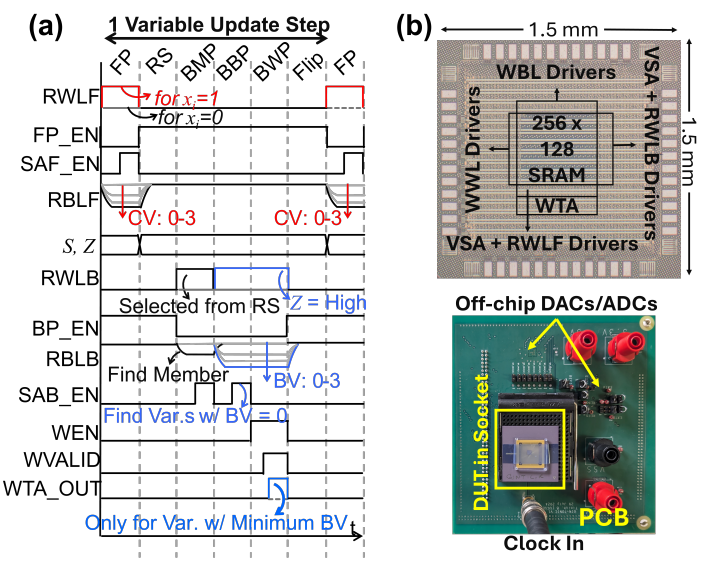
Figure 3.4: (a) Backward pass operation highlighted in a column along with relevant peripheral circuits, (b) experimental characterization of backward pass operation's transfer characteristics and error, (b) schematic of Winner-Take-All circuit and (c) Voltage-Controlled Delay Line (VCDL).

member-variable with the minimum break-value; Flip, where it is checked if there is any member variable with zero break-value, in which case such a variable is flipped (ties are broken randomly). Otherwise, with probability p and 1-p, a randomly chosen member variable or the minimum break-valued member variable is flipped, respectively.

The custom SRAM cell (Fig. 3.3a) includes the conventional 6T-SRAM cell along with two switches, each consisting of a pair of series connected nFETs. The switch associated with RWLF and RBLF enables forward pass (FP) operations, whereas the switch associated with RWLB and RBLB enables backward pass (BP) operations. Since the word lines and bit lines of the two switches are orthogonal to each other, both FP and BP can be executed in the same physical crossbar. The clauses are evaluated during FP

(Fig. 3.3b) by turning on pull-up pFETs connected to RBLFs and using RWLF drivers to apply variable-value-proportional voltages to RWLFs. For each clause j, the voltage divider between the pFET and activated SRAMs' forward switches results in a voltage on RBLF$\langle j \rangle$ that is inversely proportional to the number of true literals in clause C_j. StrongARM latch-based Voltage Sense Amplifiers (VSA) [101] with reference voltages V_{ref0} and V_{ref1} are enabled in the second half of the FP cycle to determine whether a clause has zero (S) or at most one true (O) literal, respectively (see Fig. 3.3c for reference voltage calibration). Additional digital logic is used to evaluate signal (Z), which denotes whether the clause has exactly one true literal. In the RS step, one unsatisfied clause is selected randomly by the reduction-tree-based arbiter circuit shown in Fig. 3.3c. This circuit consists of three main parts: a reduction tree of OR gates, a broadcast tree and a pseudo random number generator (PRNG). The input to the reduction tree is the vector S and it creates a heirarchical mapping of the location of unsatisfied clauses. Additionally its output determines whether the problem is unsatisfied. The digital logic of each node in the broadcast tree is shown in the inset of Fig. 3.3c. Each node accepts an input from the PRNG, another from one of its parent node's outputs and two from the outputs of OR-gates located at the same height in the reduction tree. Ultimately, the broadcast tree selects a random path from its root to one of its leaf nodes corresponding to an unsatisfied clause row (represented as the one-hot encoded output R).

In the Backward Make Pass (BMP), the arbiter's output (R) is applied to the RWLBs, made possible by the control circuit setting ϕ to 0. Since only one RWLB (corresponding to the chosen clause) is activated, a mask identifying the member variable indices is recorded by sensing the RBLBs with sense amplifiers on the variable side (Fig. 3.4a). Subsequently, in the Backward Break Pass (BBP), the RWLBs are set to (Z), made possible by the control circuit setting ϕ to 1. The resulting voltage at RBLBs of all one-valued literals depicts the Break-Value (BV) of corresponding variables (see Figures 2.5,



(c)

Benchmark Problems		
Hard	Uniform Random	100 instances per size/order
		20-var 3-SAT: ID #1-100 from SATLIB
		50-var 3-SAT: ID #1-100 from SATLIB
		Rest are custom generated
	Planted 4-SAT	50 instances per size/order
		All from W. Regli et al., 2025.
	Semi-prime	All from W. Regli et al., 2025.
Chip Details		
All measurements at 1.3 V, 180 MHz, @ RT		
Metrics		
200 repeats per instances		
Solvability		Percentage of instances solved across all repeats
Solution Time		Mean time to find a solution. Excludes programming and serial data transfer time.
Energy to Solution		Average power x solution time

Figure 3.5: (a) Timing diagram of our chip's integrated mode, highlighting the size main steps per update step, (b) chip micrograph and test setup photo, (c) details regarding benchmark problems, operation environment and measured metrics. Among uniform random problems, the 20 and 50-variable 3-SAT problems were obtained from [87], while all others were custom generated according to Appendix A.6. Planted 4-SAT instances were obtained from batches 11-12 and 25-28 of [102]. Semiprime instances were obtained from batch 7 of [102].

2.6 and 2.7 and Apendix A.3 for further details). While this step computes break-values of all variables in parallel, since the algorithm (see Fig. 3.2a) only focuses on member variables of an unsatisfied clause, the mask identified during BMP step is used to check for only those variables if any one has BV = 0. The check is performed by using a VSA to compare the voltage formed on RBLB with reference voltage V_{ref2}. Note that the same reference voltage is also used in BMP. Its value is set somewhere between the RBLB voltages expected for a logical zero and one output in the backward pass operation (see Fig. 3.4b). During BMP, a LOW output of the VSA identifies the member-variables, whereas during BBP, a HIGH output identifies the variables with zero break-value. In the Backward WTA Pass (BWP), the RWLBs are still set to Z in order to retain the break-values on the RBLBs and the winner-take-all (Fig. 3.4c) is enabled to identify the

member variable with minimum BV by gating its input multiplexers with the member-variable mask. Note that while the circuit described in Fig. 3.4c looks for the input index with maximum voltage, the inverse relation between the ideal and measured break-values (Fig. 3.4b) ensures that the one-hot encoded output of the WTA denotes the variable with minimum break-value. In the final stage (Flip), the member variable with BV = 0 is flipped if any; otherwise, with probability p, the variable with least BV (from BWP), and with probability $(1-p)$, a randomly selected variable using an arbiter on the variable side (not illustrated in the figures), is flipped. Additional bits from the PRNG is used for the p / $(1-p)$ bernoulli decision. Figure. 3.5a shows a representative timing diagram of our solver, highlighting the evolution of the critical signals through the six steps.

3.4 Measurement Results

The chip micrograph with its core functional blocks is shown in Fig. 3.5b. It also shows the test board consisting of the device under test (DUT) in a socket, port for relaying a clock signal from an external clock generator and off-chip digital to analog converters (DACs) and analog to digital converters (ADC). The DACs are used to provide the analog reference voltages V_{ref0}, V_{ref1} and V_{ref2}, whereas the ADC channels are used during the hybrid mode to implement additional heuristics.

The operation, benchmark problem and measured metric details are provided in Fig. 3.5c. Problems in the uniform random category with 20 and 50 variables, were obtained from the SATLIB library [87], whereas other problems were custom generated according to Appendix A.6. Other problems belong to the planted 4-SAT or semiprime category. They were obtained from batches 11-12, 25-28 and batch 7 of [102] respectively. Planted problems in Boolean Satisfiability refer to instances that are intentionally constructed by embedding one or more known solutions within a random or semi-random SAT for-

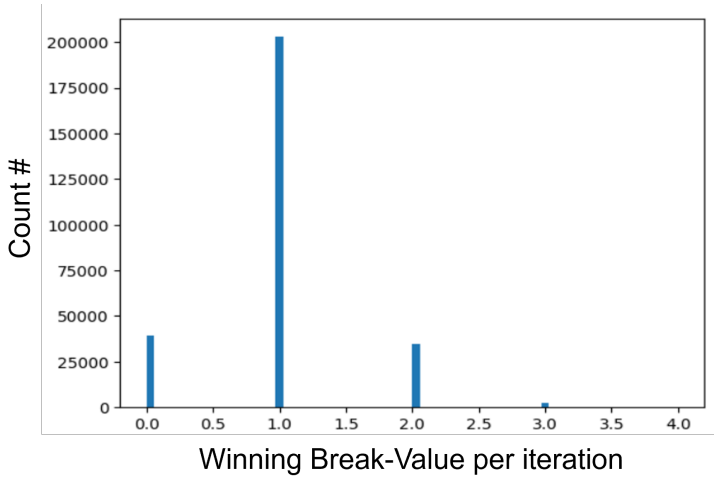


Figure 3.6: Histogram of the break-value of winning variable (the one that is ultimately flipped in each iteration) across several iterations of running the WalkSAT algorithm on uf50-900 instance from SATLIB with 500 independent restarts.

mula. Such instances provide an intermediate level of control between purely random benchmarks and structured real-world problems. Semiprime factorization in SAT solving refers to the process of encoding integer factorization problems as Boolean satisfiability (SAT) instances, particularly for semiprimes—numbers that are the product of exactly two prime numbers. In this formulation, the integer factorization task is transformed into a SAT instance whose satisfying assignments correspond directly to the prime factors of the given semiprime. Since semiprime factorization forms the basis of modern cryptographic security (notably RSA encryption), such benchmarks capture computational complexity that is highly relevant to practical cryptanalysis, thereby providing SAT instances of tangible real-world significance.

For quantifying the solver's performance we focus on metrics like solution time, solvability and the energy to solution. Here, solution time is the mean time to find a solution and is calculated by taking a mean of the different run-lengths (number of flips to reach solution) incurred across 200 independent repeats of the solver on a particular instance and then scaling it with 33 ns (time to perform a single flip, given by the time taken

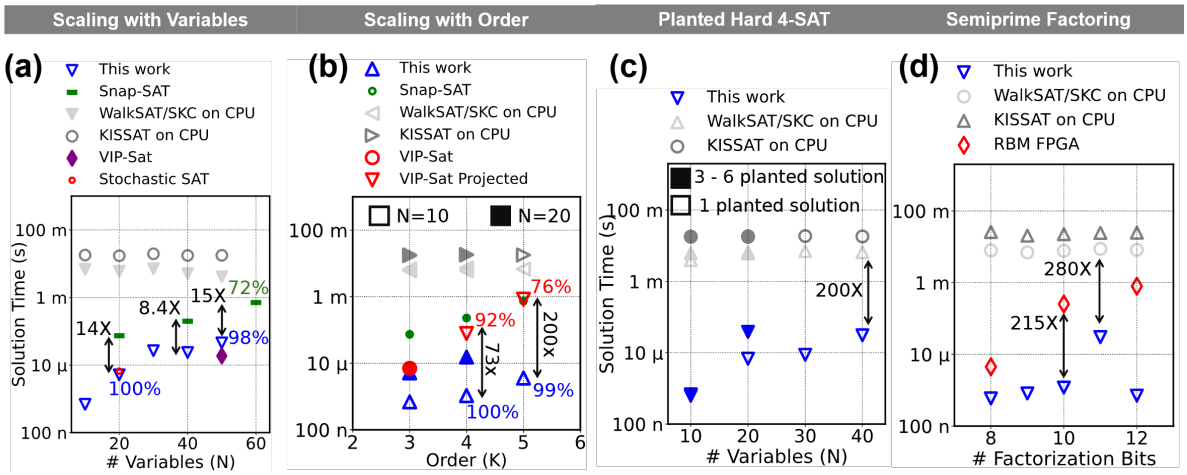


Figure 3.7: (a) Measured solution times for uniform random 3-SAT problems of different sizes and (b) K-SAT problems with different values of order (K) and variables (N) = 10, 20, respectively. (c) Measured solution times for hard 4-SAT problems of different sizes with different numbers of quietly planted solutions, (d) Measured solution times for semi-prime factorization problems. Measured results of prior solvers and their sources; Snap-SAT [96], VIP-Sat [98], Stochastic SAT [99], RBM FPGA [103].

to execute the six steps with 180 MHz operating frequency). The energy to solution is obtained by scaling the time to solution with the average power consumed while solving that problem. Solvability is the percentage of instances in a particular class of problem that are successfully solved across the 200 repeats. While Chapter 2 used a different definition of time-to-solution, most SAT-solving ASICs [96, 98, 99] report performance using mean solution time. For consistency and fair comparison with prior art, we adopt that convention here and in Chapter 4.

Figures. 3.3c and 3.4b already shows the measurement results related to FP sensing reliability and break-value transfer function observed during the backward pass operations. The nonlinearity observed in Fig. 3.4b is primarily due to the voltage-divider-based dot-product computation adopted in this work. Such a voltage-mode sensing requires relatively simple peripheral sensing circuit like the VSAs. This decision was further motivated by the fact that higher values of clause currents (during forward pass opera-

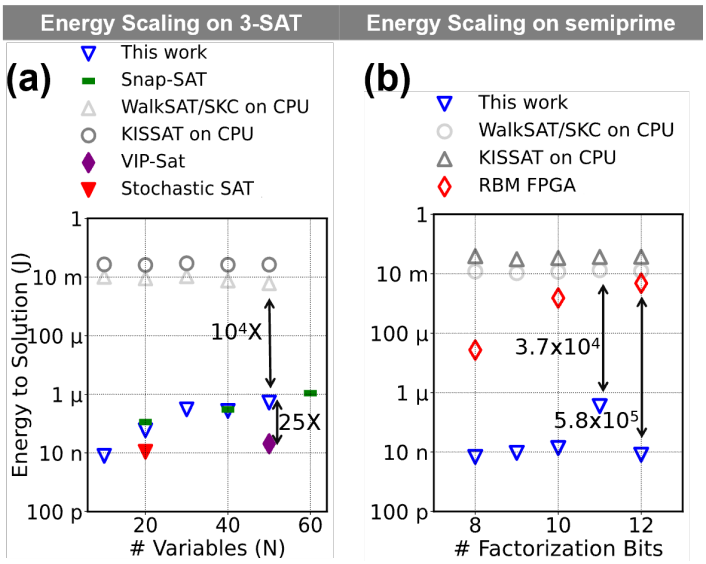


Figure 3.8: (a) Measured energy to solution on uniform random 3-SAT and (b) semi-prime factorization problems of varying size. Measured results of prior solvers and their sources; Snap-SAT [96], VIP-Sat [98], Stochastic SAT [99], RBM FPGA [103].

tions) and break-values (during backward pass operations) are rarely encountered when sampling across random variable assignments on uniform random K-SAT problems (see Figures. 3.6, 2.11,2.12,2.13 and 2.14).

Fig. 3.7a-b shows the measured solution times of our solver on uniform random SAT problems with varying size and order (percentage annotations depict solvability). The performance of our solver on hard 4-SAT with different numbers of quietly planted solutions and semi-prime factoring instances, is shown in Fig. 3.7c-d. The measured Energy-to-Solution is shown in Fig. 3.8. Figure. 3.9 provides a performance summary and comparison with prior work (only on uniform random 3-SAT, due to many prior SAT solvers being limited to implementing third order implementations only). Our solver's performance is compared with several solvers including application-specific integrated circuits (ASICs) like Snap-SAT [96], VIP-Sat [98], Stochastic SAT [99], and CMOS Ising solver chip [104], Restricted Boltzmann Machine (RBM)-based implementations on

Specifications	VLSI 2024	ISSCC 2024	Sci Rep 2024	ISSCC 2023	This work
Technology	65nm	65nm	65 nm	65nm	55nm
Architecture	Analog	Digital	Digital	Digital	Mixed
Memory	Stochas--tic CT	Near-Memory	Ring Oscillator	SRAM In/Near-Memory	SRAM In-Memory
Flips per iteration	Multiple	Multiple	Multiple	Single	Single
Chip Area (mm^2)	0.37	1.115	1.8	0.93	0.544
Maximum order	3	3	2	2 - 128	64
# Variables (N)	20	50	20	128	64
# Clauses (M)	91	218	91	1024	256
Solvability	100%‡	100%‡, 98%⇈	100%‡	72%◊	100%‡, 98%⇈
Solution Time	6.6 us‡	7 us‡, 18.7 us⇈	15.7 ms‡	70 us‡, 710 us◊	5.12 us‡, 45 us⇈
Solution Energy	11 nJ‡	20.8 nJ⇈	0.15 mJ‡	1098 nJ◊	59 nJ‡, 518 nJ⇈

‡N=20, ⇈N=50, ◊N=60, where N is # of variables.

Figure 3.9: Summary of the chip's features, performance and comparison with prior hardware SAT solvers on 3-SAT problems only.

FPGA [103] and in-house CPU implementations of WalkSAT and KISSAT [105] algorithms on a 2.8 GHz Intel Xeon CPU with 90 GB RAM and 64 logical cores. Chip-to-chip performance variation is shown in Fig. 11. While VIP-SAT [98] supports only 3-SAT instances, we projected its performance on higher-order K-SAT problems (Fig. 3.7b) by reducing high-order clauses to third-order [35], solving the resulting instances using the heuristics of [98], and scaling the algorithmic time-to-solution with physical time per iteration extrapolated for various problem sizes from [98].

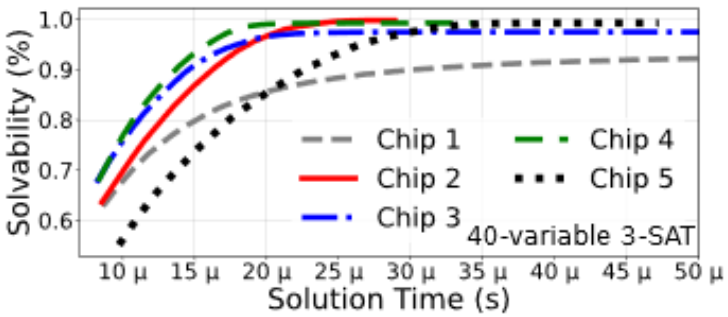


Figure 3.10: Measured chip-to-chip statistics.

Our hardware achieves nearly one order better and competitive solution times on uniform random 3-SAT, compared to other single- and multi-variable-update-based ASICs, respectively. It outperforms prior approaches, including both ASIC- and FPGA-based implementations, by 10÷200× on K-SAT problems with $K \geq 4$. Solution times are 73× and 200× better than the projected times of 3-SAT-specific solvers [98] on 4- and 5-SAT problems, respectively, at just 10 variables. This improvement results from the exponential increase in search space caused by order reduction and highlights the importance of natively high-order SAT solvers.

3.5 Hybrid Mode Operation enabling other Algorithms

The developed hardware platform also supports a hybrid mode of operation, in which off-chip peripherals such as analog-to-digital converters (ADCs) and a personal computer (PC) interface enable the implementation of algorithms beyond WalkSAT. This configuration allows exploration of the versatile capabilities of the dual cross-coupled in-memory crossbar macro integrated within our chip. In this mode, the control logic governing the sequence of operations (e.g., forward and backward passes) as well as the decision logic for variable updates can be implemented in Python on a host PC. Corresponding instructions for executing individual forward or backward passes with arbitrary input vectors are transmitted to the chip. The resulting outputs can be sensed either using on-chip peripherals such as sense amplifiers and the winner-take-all (WTA) circuit, or via off-chip ADCs and subsequently read by the PC. In this way, a wide range of existing and novel SAT-solving heuristics that depend on evaluating parameters such as clause values, make/break/gain metrics, can be implemented, demonstrating the flexibility and gener-

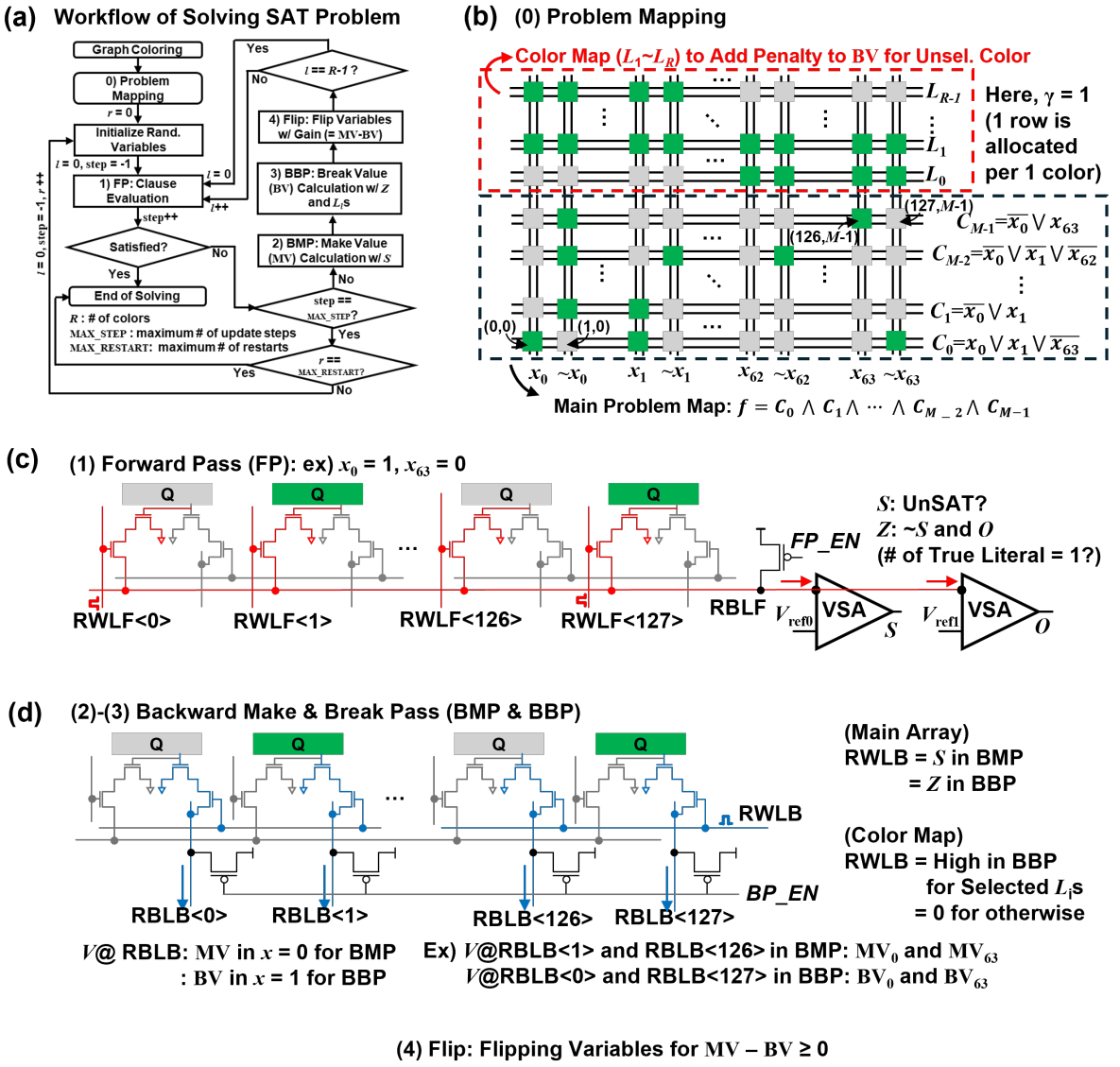


Figure 3.11: (a) Workflow of the chromatic gradient-based heuristic implemented in hybrid mode, (b) Mapping of the K-SAT problem as well as its chromatic group information to the crossbar, forward (c) and backward (d) pass operation details.

ality of the proposed architecture. Here, we introduce a simple gradient-based chromatic multi-variable update heuristic and use the hybrid mode to explore the potential of using our chip's architecture towards accelerating multi-variable update heuristics (something that is eventually designed and taped out in Chapter 4).

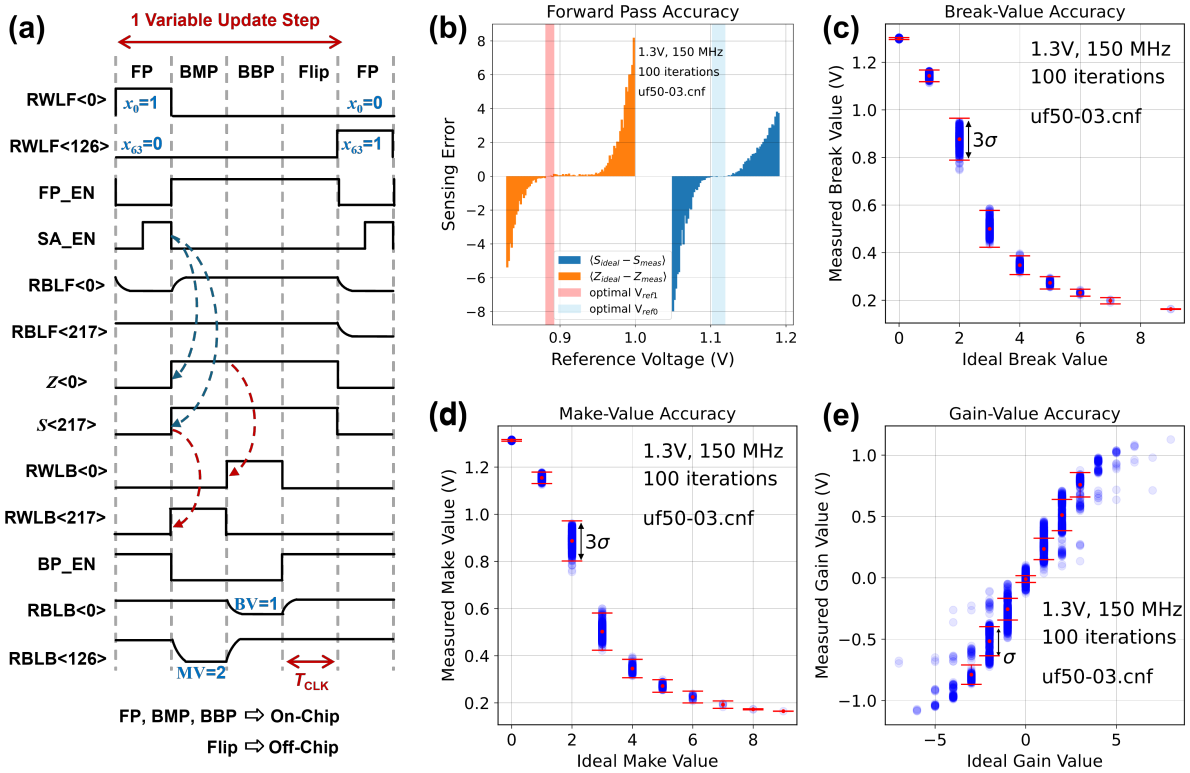


Figure 3.12: (a) Timing diagram of the gradient-based chromatic multi-variable update heuristic implemented using hybrid mode, (b) Measurement results of forward pass sensing error, Ideal versus measured break (c) and make (d) values during BBP (c) and BMP (d) operations, respectively, (d) Ideal vs Measured Gain values.

Fig. 3.11a illustrates the workflow of the proposed solving process with our macro. The first step is to perform greedy graph coloring (with polynomial time overhead) of the problem [98, 93]. This is used to exploit parallel updating of multiple uncorrelated variables sharing the same color. The second step is mapping the problem to the SRAM array, which is done in a manner similar to Chapter 3.3, where columns represent literals and rows represent clauses of the problem. The SRAM cell at location $(2i, j)$ or $(2i{+}1, j)$ is written with a "1" when the normal or complementary literal of a variable x_i is present in a clause C_j; otherwise, it is programmed to a "0". Variable colors are mapped to the unused part of the array. Each variable color is encoded with γ number of rows, so that γR

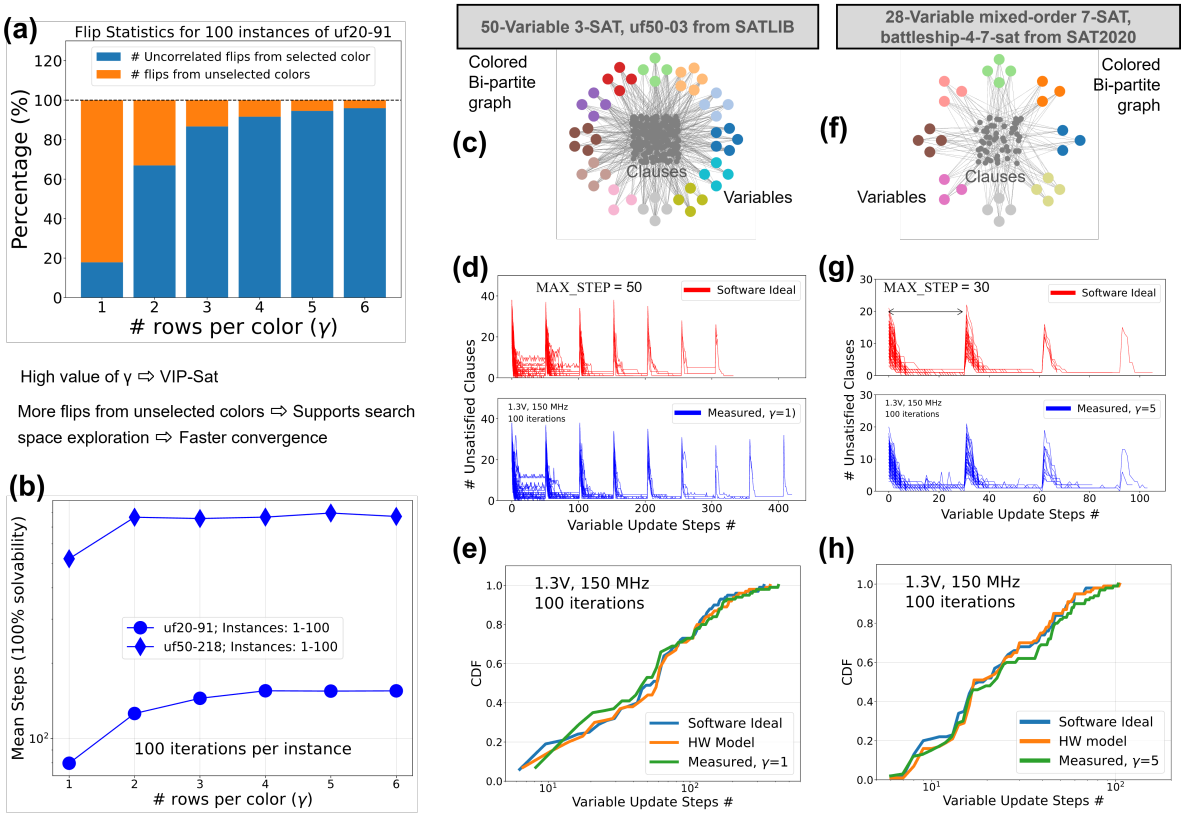


Figure 3.13: Simulation results showing the impact of the parameter γ on (a) the distribution of variable update type (uncorrelated flips versus correlated flips) and (b) the mean number of steps taken to find a solution. (c) and (f) show the bi-partite graph representing third and seventh order SAT problems, respectively. (d) and (g) show the evolution (measured and simulated) of clause satisfaction with update steps for a third and seventh order SAT problems respectively. (e) and (h) show the cumulative distribution (simulated and measured) of update steps for the two problems. Here VIP-Sat refers to the solver in [98].

rows of the SRAM array are allocated toward storing color-related information for a total of R unique colors. Any variable x_i that does not belong to color L_j has corresponding SRAM cells at $(2i, \gamma j+M+t)$ and $(2i+1, \gamma j+M+t)$, where $t = \{0, 1, \ldots, \gamma-1\}$, written with a "1"; otherwise, they are written with a "0".

Once all variables are initialized randomly, the solver proceeds through iterative variable-update cycles until convergence. Each iteration consists of four major stages; Forward Pass (FP), Backward Make Pass (BMP), Backward Break Pass (BBP), and Flip

(FP); that are executed sequentially (see Fig. 3.12a) while the solver advances through a fixed color schedule, activating one color per iteration. Clauses are evaluated during (1) FP, by turning on pull-up pFETs connected to RBLFs and using drivers to apply variable-value-proportional voltages to the RWLFs. The voltage divider formed between the pFET and activated SRAMs' read nFETs results in a voltage on RBLF$\langle j \rangle$ that is inversely proportional to the number of true literals in clause C_j. Two VSAs with reference voltages V_{ref0} and V_{ref1} are enabled in the second half of the FP cycle to determine whether a clause is unsatisfied (S) and whether it has at most one true literal (O), respectively. Additional digital logic ($\sim S$ AND O) is used to evaluate signal (Z), which denotes whether the clause contains exactly one true literal.

Subsequently, the RWLBs associated with unsatisfied clauses ($S\langle j \rangle = 1$) are activated during (2) Backward Make Pass (BMP) to compute Make-Values (MV), i.e., the number of unsatisfied clauses that a variable is part of. As a result, the voltage at RBLBs of all zero-valued literals, given by H_B(MV), where H_B is the BP transfer function, depicts the MV of corresponding variables.

The next step is the (3) Backward Break Pass (BBP) to compute Break-Values (BV), i.e., the number of satisfied clauses that a variable is part of, which will become unsatisfied (or break) if that variable is flipped. In this step, the RWLBs of the main array associated with clauses that contain one true literal ($Z\langle j \rangle = 1$), and the RWLBs of the color array associated with the currently selected color, are activated. The voltage at RBLBs of all one-valued literals, given by H_B(BV)/H_B(BV+γ), depicts the BV / (BV with γ) for corresponding variables that are part of the currently selected color / unselected colors, respectively.

Lastly, in (4) the voltage difference

$$G_{\text{eff}} = H_B(\text{BV}) - H_B(\text{MV}) \;/\; H_B(\text{BV}+\gamma) - H_B(\text{MV})$$

is computed for variables of selected/unselected colors, respectively. The difference is proportional to the gain (= MV−BV), i.e., the negative pseudo-gradient, and gain−γ, for variables of selected and unselected colors, respectively (note that H_B is a monotonically decreasing function). Variables whose $G_{\text{eff}} \geq 0$ are flipped, and the process returns to step (1).

Fig. 3.12 shows the measurement results related to the FP sensing reliability and backward pass transfer function H_B. The chip is operated at 1.3 V and 150 MHz. A 3-SAT instance (`uf50-03.cnf`) from the SATLIB benchmark with ($N = 50$, $M = 218$) is mapped to the SRAM array. Optimal VSA reference voltages $V_{\text{ref0}}/V_{\text{ref1}}$ that minimize the Hamming distance error between expected and measured S and Z values over 100 forward-pass operations with randomly initialized variable values are determined. One hundred iterations of BMP and BBP with realistic S and Z input vectors and $\gamma = 0$ are performed to measure the backward-pass transfer function H_B. The ideal and measured gain values are given by the difference between ideal make and break, and the difference between break and make- voltages measured during backward pass operations, respectively.

Fig. 3.13a-b shows simulation results on uniform random 3-SAT problems, highlighting how changes to the microarchitecture (by varying γ) can change the dynamics and performance of the heuristic. The offset γ is used to control the degree of correlated variable-update inhibition. If it is set to a high value, only uncorrelated variables with non-negative gains are flipped. Conversely, if it is set to zero, all variables with non-negative gains are flipped, which may lead to oscillations and non-convergence. However, setting it to an optimal value allows correlated variables with relatively high gain (important variables) to flip in addition to uncorrelated variables of the currently selected color. This leads to better navigational efficiency and faster solution times for certain SAT problems.

Fig. 3.13c-h shows the measurement results of solving two hard SAT problems using our macro. Each problem, along with its specific color map (see Fig. 3.13c,f), is mapped to the SRAM array. Optimal values of γ that lead to minimum solution times are obtained from software simulations. Each instance is solved 100 times. The measured Cumulative Distribution Function (CDF) of the variable-update steps required to find a solution follows closely that of the software simulations with the hardware-aware model (which incorporates the H_B transfer function and error distribution of BP) and the hardware-agnostic (ideal) model. The solver successfully finds a solution in all 100 trials for both instances. The mean solution-time for the 3-SAT and 7-SAT instances are $1.9\,\mu$s/$9.6\,\mu$s and $0.8\,\mu$s/$2.6\,\mu$s, respectively (assuming each of the four stages per update step can complete in 6.6 ns, corresponding to a clock frequency of 150 MHz). While FP, BMP, BBP stages are experimentally and individually demonstrated at 150 MHz, it is assumed that the last stage FP (which is executed in PC) can also complete in 6.6 ns. However, our experience with the integrated mode, where a more complicated variable-update logic is implemented in a smaller duration of 5.5 ns, suggest that this could be fair assumption.

3.6 Summary and Outlook

In summary, this chapter presented the design, implementation, and measurement results of a K-SAT solver chip fabricated in 55 nm CMOS technology. The solver can map arbitrary-order SAT problems and leverages the in-memory computation techniques introduced in Chapter 2 to successfully demonstrate solution to hard K-SAT instances with up to 50 variables. In its integrated mode, which runs the WalkSAT algorithm encoded within the on-chip control logic, the hardware achieves superior performance compared to prior solvers. In its hybrid mode, combining on-chip in-memory computation with off-chip ADCs and a host PC, the system demonstrates the versatility of the proposed

architecture. This flexibility opens the door to implementing advanced methods such as high-order Hopfield neural networks [69] and multi-variable-update solvers, which are further explored in Chapter 4. Experimental and modeling analyses confirm that local-search-based SAT solving remains robust to nonlinearities in break-value computation. Moreover, while the current prototype supports up to 64 variables, the adoption of an FPGA-inspired reconfigurable routing fabric could enable multi-tile solvers capable of handling thousands of variables [73], as discussed later in Chapter 6.

Chapter 4

A 55 nm K-SAT solver with probabilistic-bits based parallel variable update

4.1 Background

Boolean Satisfiability (K-SAT, $K \geq 3$) is an NP-complete problem that arises across nearly all domains of science and engineering like cryptography, computational biology, AI planning and EDA [106]. Solving SAT, i.e., finding satisfying assignment for all clauses, is computationally expensive on any hardware, which has motivated efforts towards designing SAT-solving hardware accelerators [107, 96, 70, 71, 98, 99]. Complete K-SAT solvers based on the DPLL algorithm [107] can prove unsatisfiability of instances, but their decision-tree search scales poorly with problem size compared to other algorithms. A fully digital WalkSAT-based solver [96] exploited SRAM-based In-Memory Computing (IMC) for parallel clause evaluation but computed break-values (B_i, refer to Chapter 2 for details) sequentially, thereby limiting its performance. A mixed-signal

in-memory computing solver [70, 71] improved parallelism by computing clause and break-values in one shot using SRAM-based in-memory computing, but updated only one variable per iteration, limiting navigational efficiency. Solvers updating multiple variables in parallel using chromatic block updates [98] or continuous-time (CT) dynamics [99, 108] have demonstrated fast convergence but are typically restricted to 3-SAT problems. Solving higher-order ($K > 3$)-SAT problems on such hardware [98, 99, 108], as well as other accelerators that are limited to second-order interactions [109, 110], requires introducing additional variables and clauses, which inflates the search space exponentially and degrades performance [35].

Here, we present a 55 nm mixed-signal K-SAT solver that implements a new parallel-variable update algorithm for fast convergence and a in-memory computing fabric that maps arbitrary-order K-SAT instances without preprocessing. It solves 50 variable 3-SAT (satisfiable) problems in 5.5 μs with 100 % solvability at 265 nJ. A custom 11-transistor SRAM cell based crossbar is used for one-shot analog gradient computation and ADC-less gradient-informed sampling with probabilistic-bit circuits that enable 4-13$\times$ speedup over prior solvers.

4.2 Algorithm Design

The dual cross-coupled in-memory architecture described in Chapter 2 faces scalability limits beyond a certain number of variables and clauses, motivating the development of multi-tile architectures. These limitations primarily stem from increasing parasitic delays and leakage currents in larger crossbar arrays, which degrade the signal-to-noise ratio of the readouts. One such multi-tile implementation is introduced in Chapter 6.

Regardless of the specific architecture, executing algorithms such as WalkSAT requires a global winner-take-all (WTA) circuit that aggregates analog inputs from multiple tiles

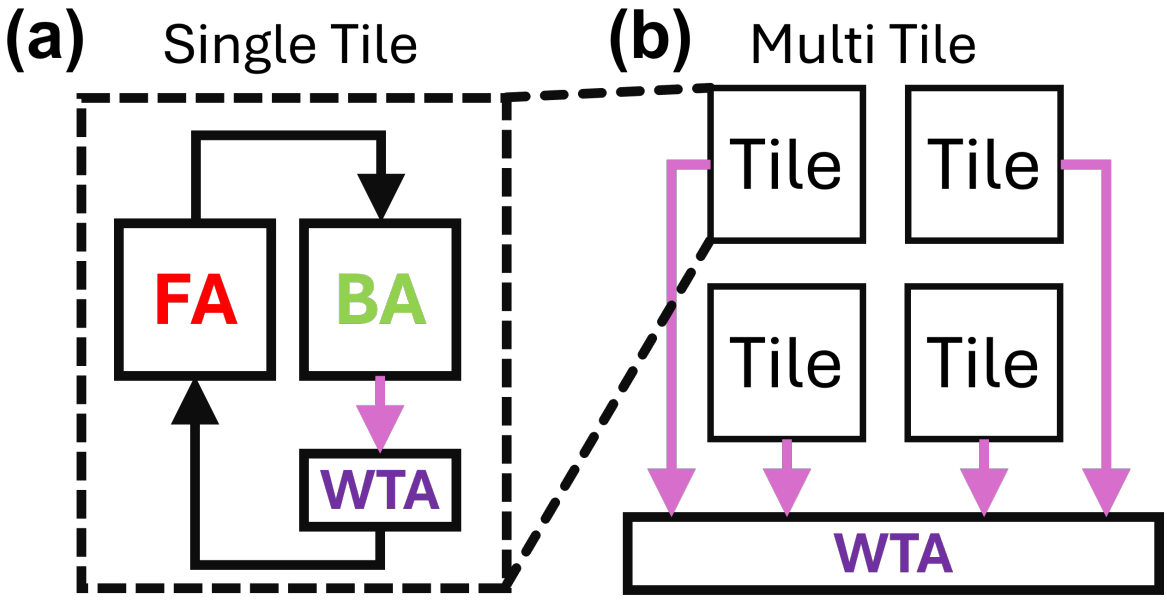


Figure 4.1: (a) A single tile consisting of the dual cross-coupled in-memory crossbar based tile, (b) illustration of an architecture where a global winner-take-all circuit is required to determine promising variables when more than one tile is present.

across the die to identify the most promising variable (see Fig. 4.1). However, the performance of such an analog reduction circuit deteriorates as the number of inputs (i.e., variables) increases, fundamentally constraining the scalability of such systems [111].

One workaround that avoids using large analog reduction circuits is to implement local WTAs within each tile and employ an arbitration scheme during the variable-update phase to select the tile whose local winner becomes the global variable to flip in that iteration. Simulations of a such a system with random-arbitration and local WTAs exhibited worse performance than the baseline WalkSAT algorithm, with degradation becoming more pronounced as problem size increased. The performance also varied with the specific graph partitioning implemented. Another solution could be to use analog-to-digital converters (ADCs) that can digitize the analog make/break or gain values in each tile, and using digital-domain reduction circuit to determine the global true winner.

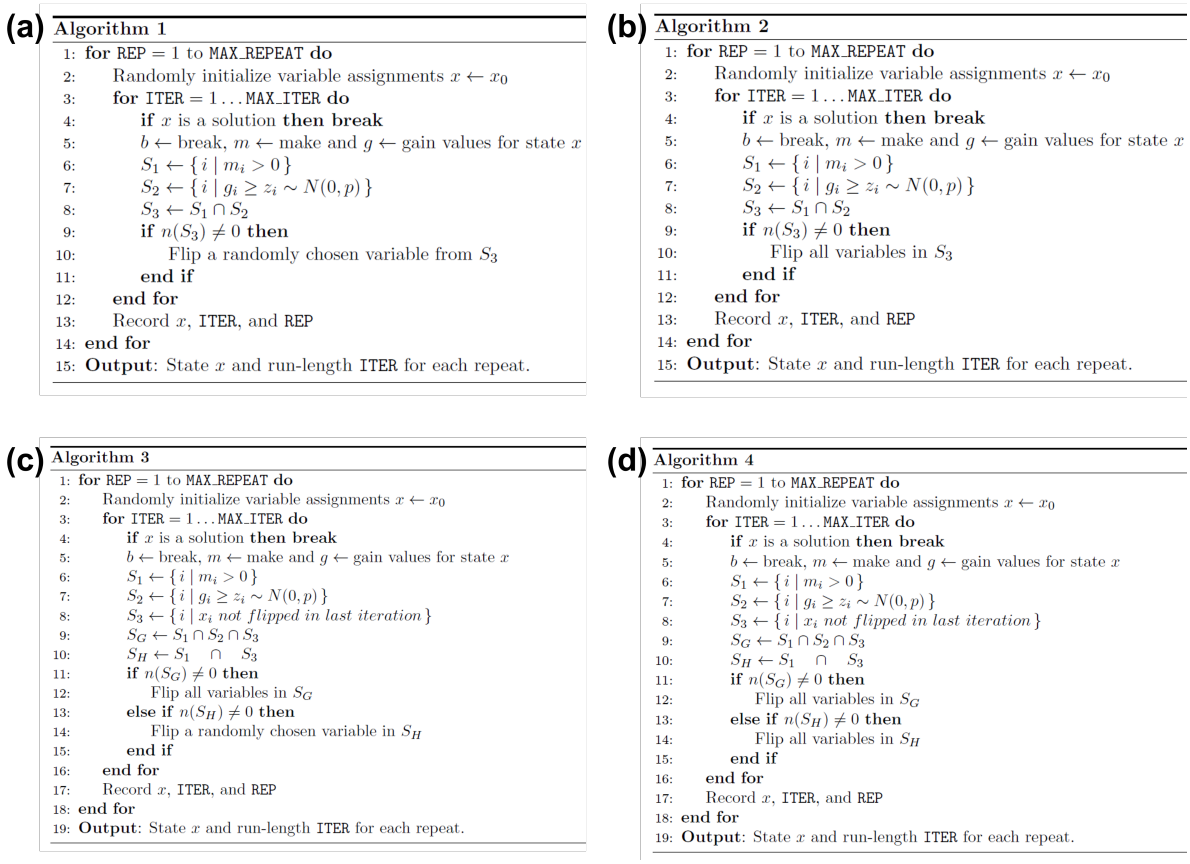


Figure 4.2: Pseudocode of the four different algorithms: (a) Algo 1, (b) Algo 2, (c) Algo 3, (d) Algo 4.

Although both analog and digital reduction circuits experience latency growth with input count, digital implementations can exploit pipelining to support multi-trial parallelism (as discussed in Fig. 2.15). Nevertheless, integrating ADCs incurs significant energy and area overheads. Therefore, there is a need for algorithms that does not require reduction cicuits in analog domain, or power and area hungry ADCs, performs at least as good as the WalkSAT algorithm and potentially takes advantage of the parallelism offered by the high-order gradient computing core and a multi-tile architecture.

Guided by these principles, we present four algorithms that for the lack of a fitting name, we simply refer to as Algo 1, Algo 2, Algo 3 and Algo 4 (see Fig. 4.2). The two

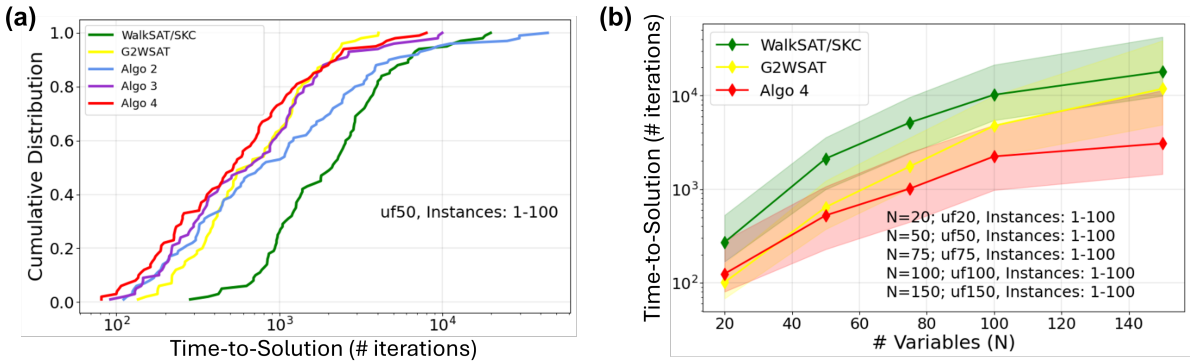


Figure 4.3: (a) Cumulative distribution of the algorithmic time-to-solution with 99% certainty (as defined in Appendix A.9) for Algo 2, Algo3, Algo4, WalkSAT and G2WSAT [97]. (b) Algorithmic time-to-solution (in iterations) with problem size for the different algorithms. Shaded area define the 25% to 75% percentile regions. All instances from SATLIB.

defining features of all these algorithms is (i) to only consider update of variables with non-zero make-values (denoted by the set S_1) and (ii) use of probabilistic-bits (p-bits) to propose update of variables (denoted by the set S_2) instead of using deterministic reduction circuits.

A variable that has non-zero make-value is necessarily present in at least one unsatisfied clause, by definition. Therefore, only variables in unsatisfied clauses are flipped, a key feature of the WalkSAT algorithm. However, unlike WalkSAT that focuses on the variables of a specific clause, these algorithms consider variables present across all unsatisfied clauses. This avoids an arbiter circuit for selecting a random clause and an additional pipeline stage to find member variables of the chosen clause (BMP stage in Fig. 3.5).

We refer to p-bits in this thesis as functions that outputs a 1 with probability that is proportional to its input, otherwise a 0. The input to the p-bits in the implemented algorithms is the variable gain and the transfer characteristic (that defined the relationship between the input and output probability) is a saturating nonlinear function (resembling

a sigmoid function). This action is mathematically described in Line 7 of the panels in Fig. 4.2. Now since the input of individual p-bits is a variable's gain value, and does not depend on the parameter of any other variable, analog reduction circuits are entirely avoided. Inter-tile communication remain restricted to digital exchanges of variable and clause statuses, as will be seen in Chapter 6.

Variables present in unsatisfied clauses probabilistically end up in the set S_3 in Algo 1 and Algo 2 (with the ones with higher gain having a higher probability). While Algo 1 selects one variable from this set S_3 at random, Algo 2 flips all variables in the set in parallel. Such a multi-variable update scheme might flip correlated variables (those sharing a common clause) and potentially result in an increase in the energy function (number of unsatisfied clauses in K-SAT problems). However, as we saw in Fig. 3.13a-b and will see in subsequent results, a certain degree of freedom with respect to flipping correlated variables can improve performance over algorithms that update multiple variables only from the same color (uncorrelated variables). The exact reason behind such an observation is unknown and a rigorous mathematical formulation of this phenomenon remains a topic of future work. One condition that is conjectured to be responsible for such a parallel update scheme to work is the sparsity of the problem graph.

Algo 3 and 4 incorporate two new features in addition to parallel variable updates; flip-history and rejection-free mechanism. The set S_3 in Algo 3 and Algo 4, tracks variables that were not flipped in the previous iteration (the flip-history). Considering variables that are also present in this set (along with S_1 and S_2) avoids updating variables that were very recently flipped, thereby limiting oscillations (a scenario when the same set of variables keep getting flipped back and forth). It implicitly introduces the idea of refractory period, a mechanism critical to neural networks [112, 113]. The rejection-free mechanism flips a single (in Algo 3) or all variables (Algo 4) that belong to any unsatisfied clause and was/were not flipped in the previous iteration, when there the set $S_1 \cap S_2 \cap S_3$

is empty. This reduces the number of iterations where no variable is flipped.

Figure. 4.3 summarizes the performance of the different algorithms (excluding Algo 1) and compares them against the WalkSAT and G2WSAT algorithm [97] (another competitive stochastic local search based SAT solving algorithm). While all algorithms perform better than WalkSAT on the easy to intermediate instances, Algo 2 performs poorly on the harder instances (marked by the tail on the right side in Fig. 4.3a). This could be attributed to the naive parallel updates that it implements that often leads to oscillations in harder instances. Both Algo 3 and Algo 4 on the other hand, perform better than WalkSAT across most instances and at par with the G2WSAT algorithm. Figure. 4.3b shows how the performance of different algorithms scale with problem size. In this figure we choose to focus on only Algo 4 out of the algorithms introduced in this section due to its relatively superior performance. Moreover, its advantage over WalkSAT and G2WSAT seems to increase with problem size. Note that hyperparameter optimization was done for each algorithm on the first 15 instances of the uf50-218 set from SATLIB and those parameters were kept the same across all problem sizes. The degradation in G2WSAT's performance may indicate that the chosen hyperparameter values are not optimal at larger sizes. It also displays the much lesser sensitivity of Algo 4 to carefully tuned hyperparameters.

In the next section, we present the design and measurement results of a 55 nm CMOS integrated circuit that implements Algo 4 to solve K-SAT problems.

4.3 Key Features

The probabilistic SAT-solving algorithm introdued in previous section (referred to as Algo 4 in 4.2) operates directly in the high-order K-SAT space and leverages the bipartite clause-variable relationship to update multiple variables per step with high

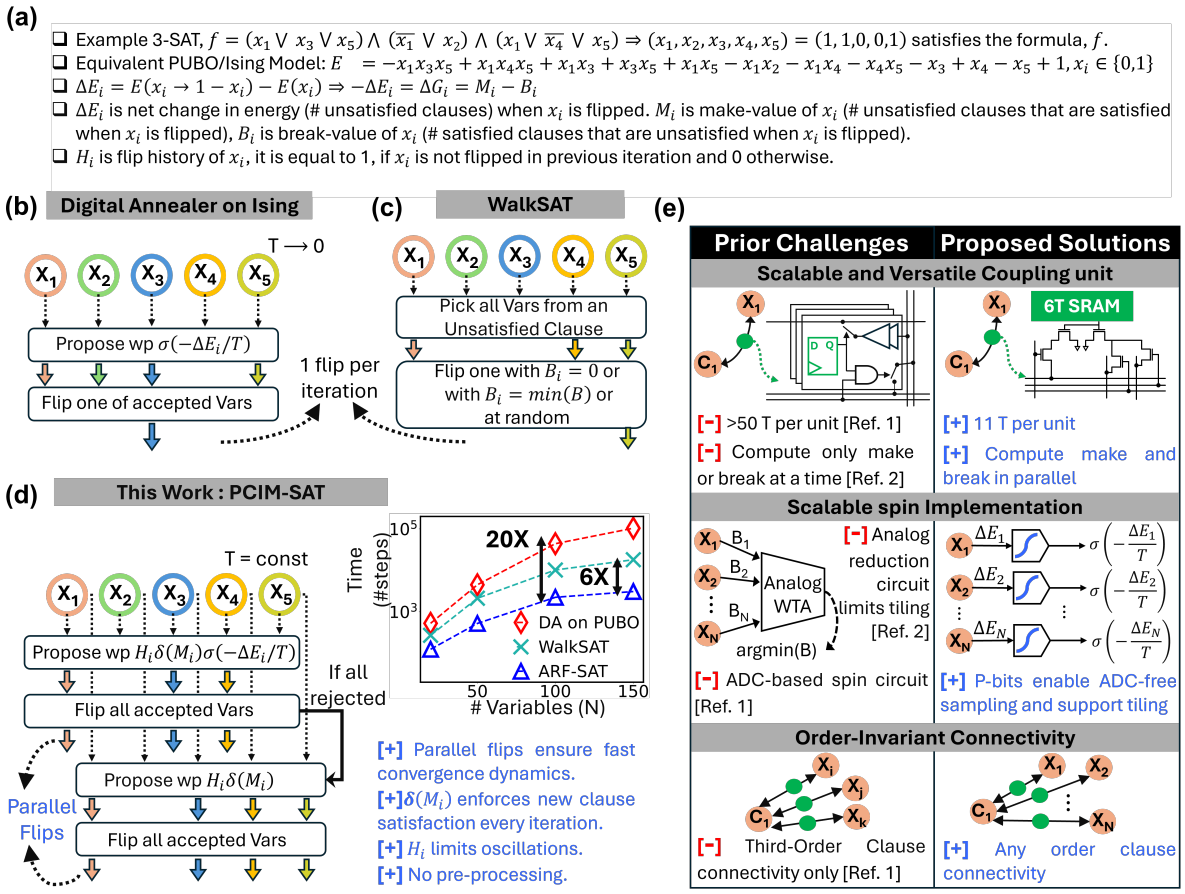


Figure 4.4: (a) Example of Boolean Satisfiability problem, its conversion to Ising/PUBO model and definition of other relevant concepts. Single variable update dynamics of (b) Digital Annealer and (c) WalkSAT algorithm. (d) Parallel variable update dynamics, key properties of proposed algorithm PCIM-SAT and comparison of its algorithmic performance with other solvers. (e) Challenges with respect to prior hardware solvers and the solutions proposed in current hardware for tackling those. Ref. 1 is [108] and Ref. 2 is [70].

navigational efficiency. It effectively modifies the sigmoidal probability function $\sigma(\cdot)$ of Gibbs sampling-based solvers (like the Glauber dynamics [34]) by scaling it with a Heaviside step function $\delta(\cdot)$ of the variable's make-value M_i, and the flip-history H_i (see Fig. 4.4a for definitions). This restricts updates to promising variables in unsatisfied clauses and limits oscillations. Variables accepted by this function are updated independently in parallel, while a rejection-free mechanism ensures progress by updating all

variables with non-zero M_i and H_i, if no candidate is accepted. This heuristic outperforms state-of-the-art (SOTA) algorithms such as WalkSAT and the Digital Annealer [40] (when simulated on third-order Ising formulation of 3-SAT [37]), with performance gains increasing with problem size (see Fig. 4.4b-d).

This work advances the field of SAT-solving hardware by making four key contributions: (1) a mixed-signal in-memory computing architecture that is order-invariant and implements a novel SAT solver capable of updating multiple variables in each time step without prior preprocessing (Fig. 4.4d-e), (2) a custom 11-transistor SRAM crossbar array that allows computation of both make and break-values of all variables (effectively the gradients) in a single cycle (Fig. 4.4e), (3) dynamic comparator-based probabilistic-bit (p-bit) circuits on the variable side, enabling massively parallel sampling informed by the analog gradients computed in the crossbar, thereby eliminating the need for ADCs or analog-domain reductions (Fig. 4.4e), and (4) a multi-threaded pipeline that exploits the decoupled orthogonal accesses to the crossbar array to launch three independent and parallel solvers, improving convergence by 1.7× at similar energy.

4.4 Architecture and Schematics

The algorithm implementation (Fig. 4.5b) involves the following circuit blocks (Fig. 4.5a): a 256 × 128 bi-directional 11T-SRAM array; clause evaluators; variable/clause registers and drivers; Satisfiability (SAT) evaluators; make-value filter; p-bits; a 128-bit Pseudo Random Number Generator (PRNG); and a 128-bit wide 4-stage RNG buffer chain. In the first step, the K-SAT problem is loaded to the memory array such that each row corresponds to a clause and each column to a literal. Depending on whether x_i or $\sim x_i$ is present in clause C_j, $\text{SRAM}_{j,2i}$ or $\text{SRAM}_{j,2i+1}$ is set to 1 respectively, otherwise to 0. In the second step, all variables are initialized to random binary values using the

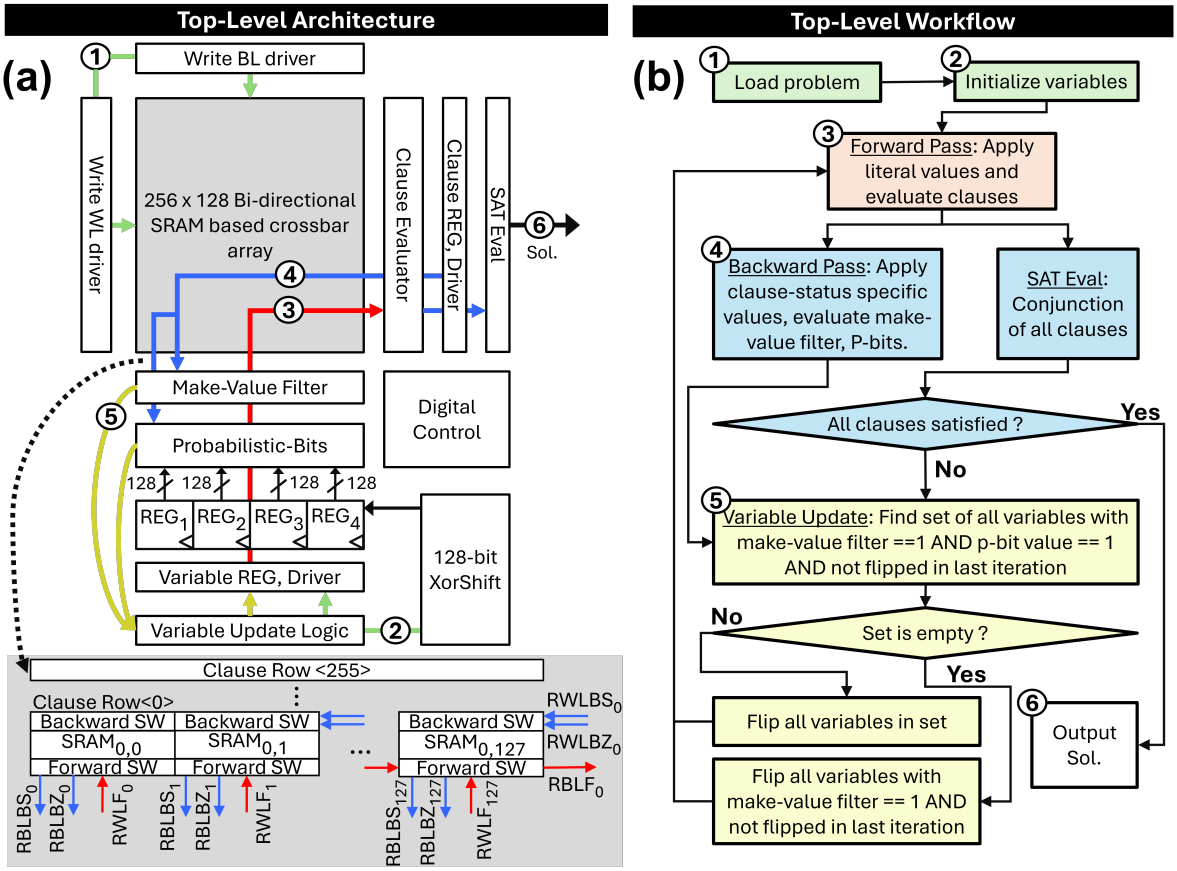


Figure 4.5: (a) Top-level architecture and (b) Workflow of the chip.

PRNG. Subsequently, the solver attempts to find the solution by iteratively executing the following three functions in sequence: (i) *Forward Pass (FP)*; where literal-values are applied to the crossbar and all clauses are evaluated in parallel, (ii) *Backward Pass (BP)*; where clause signals are applied to the crossbar array to compute the make and break values of all variables in parallel. The make-value filters detect the variables with $M_i > 0$, whereas the p-bits for each variable output a 1 with a probability proportional to the gain of the variable ($= M_i - B_i$). Simultaneously, the SAT evaluator checks if the problem is satisfied, (iii) *Variable Update (VU)*; where it is checked if there are variables with $M_i > 0$ and $H_i = 1$ and accepted by the p-bit (corresponding to $S_1 \cap S_2 \cap S_3$ in Fig.

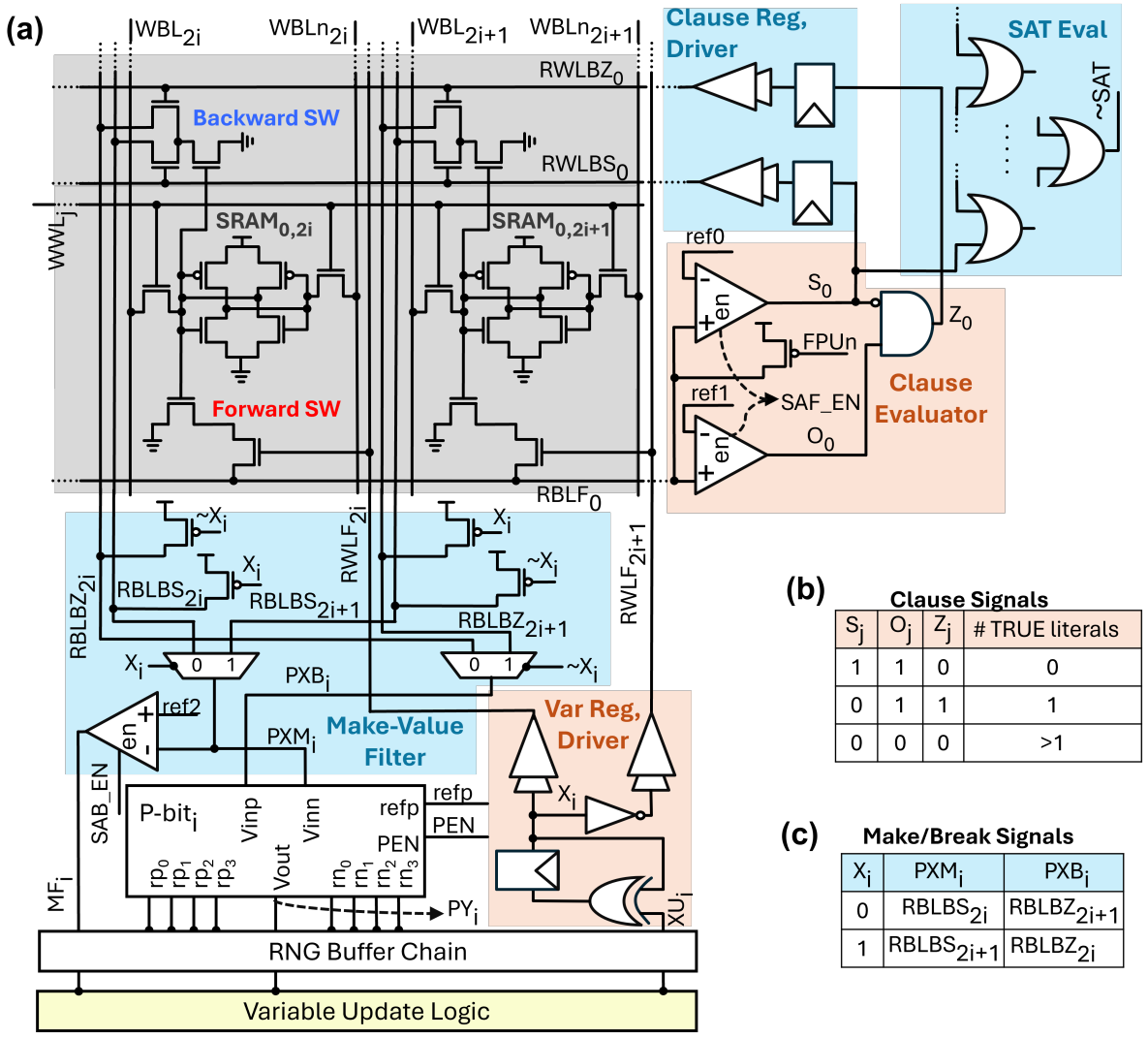


Figure 4.6: (a) Detailed schematic of the mixed-signal in-memory computing fabric and tables depicting values and relations of signals critical to (b) clause evaluation and (c) backward pass operation..

4.2d), in which case all such variables are flipped; otherwise, all with $M_i > 0$ and $H_i = 1$ (corresponding to $S_1 \cap S_3$ representing the candidates for rejection-free mechanism in Fig. 4.2d) are flipped. Existence of at least one variable with $M_i > 0$ is guaranteed at this stage, since no variable with $M_i > 0$ implies the solution has been reached.

On the schematic level (Fig. 4.6), each coupling cell in a row consists of a 6-transistor

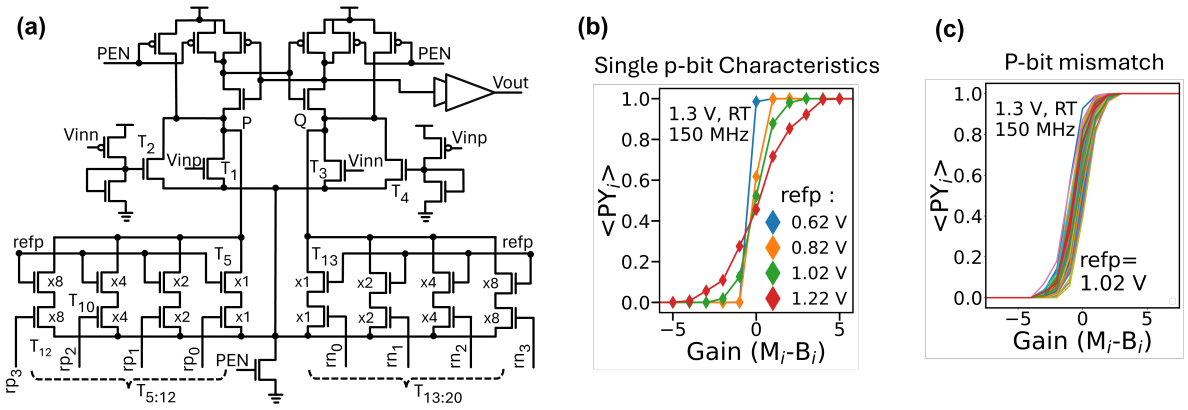


Figure 4.7: (a) Schematic of a probabilistic-circuit, (b) transfer function of a single p-bit for different values of refp and (c) transfer function of all p-bits in the chip for a fixed value of refp.

SRAM, 2-transistor forward switch, and 3-transistor backward switch. During the forward pass, drivers apply variable-proportional voltages to the RWLFs, while at the same time pull-up pFETs connected to the RBLFs on the clause side are turned on. The forward switches that are ON represent literals in respective clauses that have been set to 1 (TRUE). As a result, the voltage divider formed by the pFET and the nFETs in the forward switches generates a voltage on RBLF_j inversely proportional to the number of true literals in clause C_j. StrongARM latch-based voltage sense amplifiers (VSAs), referenced to off-chip analog reference voltages ref0 and ref1, are used to detect whether a clause has zero ($S_j = 1$) or at most one ($O_j = 1$) true literal. Additional logic derives signal Z, indicating clauses with exactly one true literal. The SAT evaluator uses a tree of OR gates to check if all clauses are satisfied.

In the backward pass, clause signals S_j and Z_j are applied to RWLB_{S_j} and RWLB_{Z_j} respectively. The make-value of variable x_i is evaluated by accumulating contributions from clauses with $S_j = 1$ and that contain its zero-valued literal, on PXM_i. The break-value of variable x_i is computed by accumulating contributions from clauses with $Z_j = 1$ and that contain its one-valued literal on PXB_i [69]. The peripheral pull-up

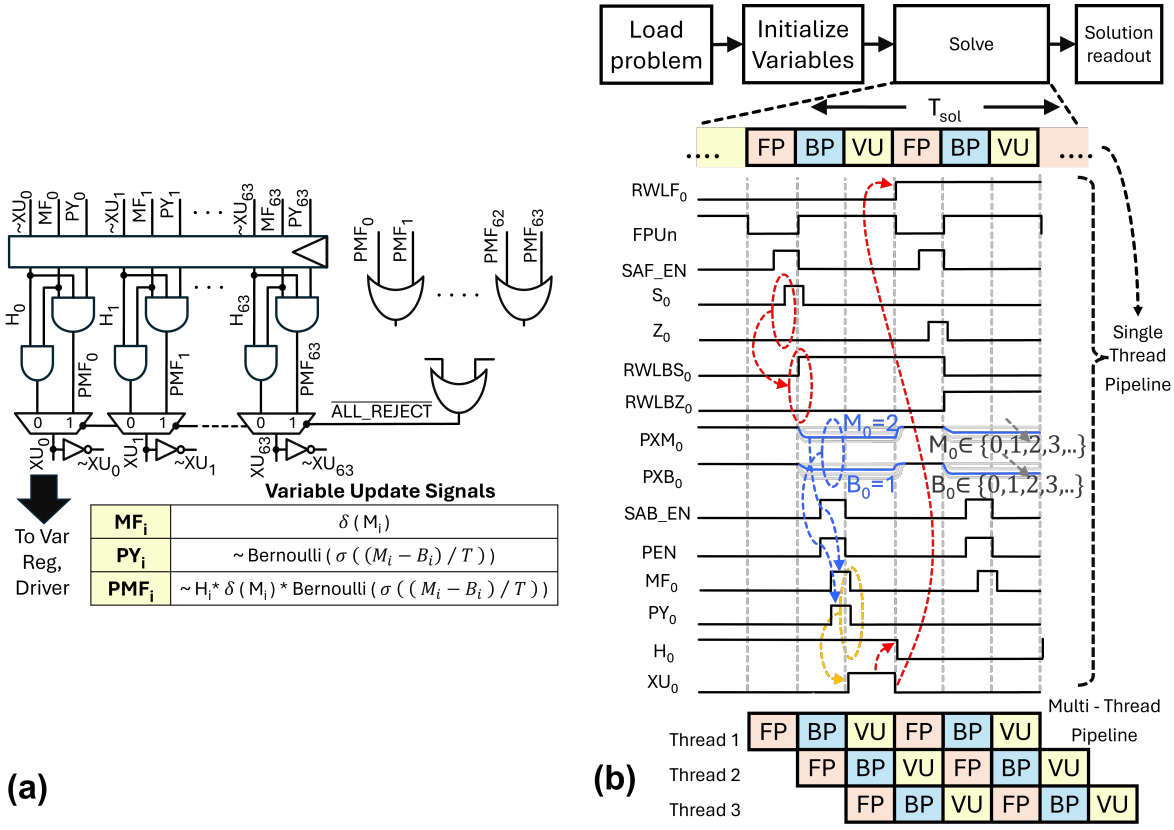


Figure 4.8: (a) Schematic of variable update logic and (b) illustration of the chip's timing diagram.

pFETs and multiplexers ensure for each variable that a voltage divider on only the $\mathrm{RBLB_S}$ ($\mathrm{RBLB_Z}$) line of its zero- (one-) valued literal is activated and its output is connected to PXM_i (PXB_i). VSAs referenced to ref2 are used to detect $M_i > 0$.

Probabilistic bits (p-bits) are implemented using a StrongARM latch (see Fig. 4.7a) with three differential pairs: (i) T1 and T3, (ii) T2 and T4, (iii) T5:12 and T13:20. While both (i) and (ii) discharge nodes P and Q during the evaluation phase (PEN = 1), with differential current proportional to $\mathrm{PXB}_i - \mathrm{PXM}_i$, the latter along with its p-type input common-source amplifiers extends support for a rail-to-rail input range. Each branch in (iii) is a 4-bit binary-weighted current-switching DAC, and the pair contributes a differential current proportional to $\mathrm{ref}_p \times (R_p - R_n)$, where R_p and R_n are the uniform

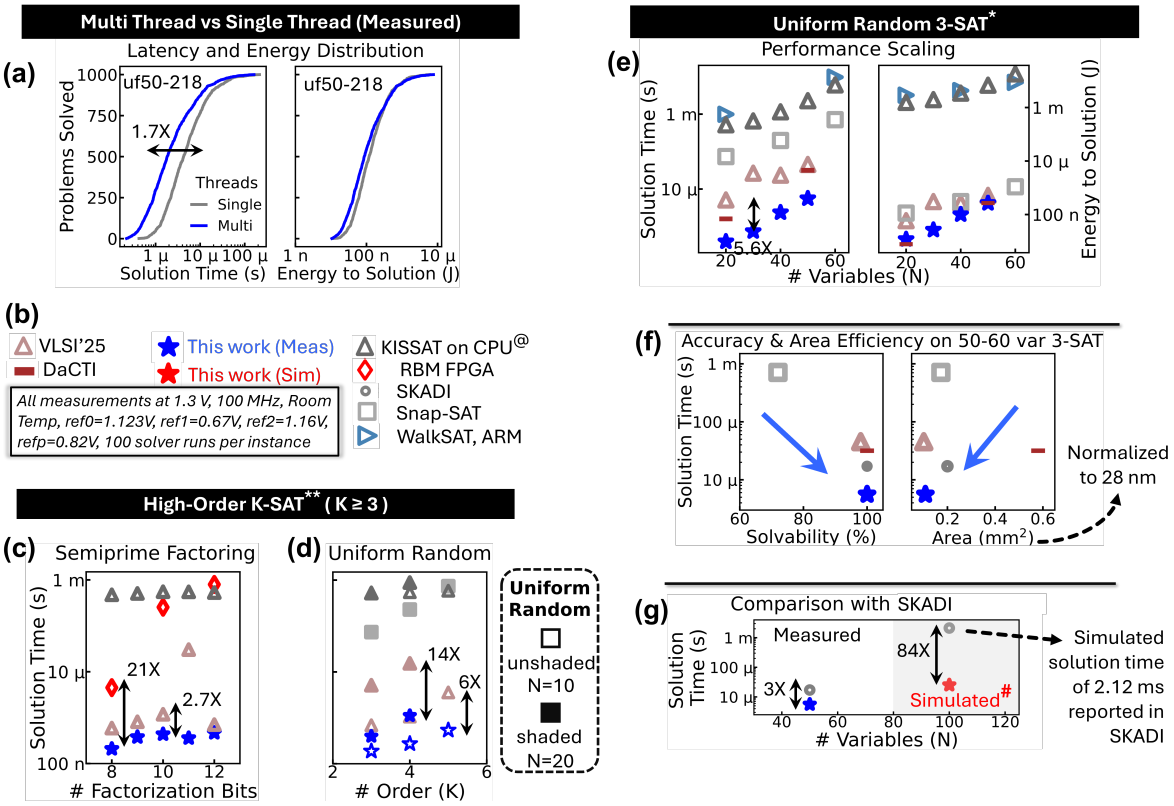


Figure 4.9: Measurement results of our chip on different SAT benchmarks and comparison with other solvers. (a) Solution time and energy-to-solution distribution across all 1000 instances of uf50-218 set in SATLIB [87], when the single- and multi-thread modes of the chip is selected. (b) Legend box and other experimental parameters for the measurements shown in this figure. Scaling of solution time with problem size and order on (c) semiprime factoring problems and (d) uniform random high--order K-SAT problems. (e) Scaling of solution time and energy with variables on uniform random 3-SAT instances. (f) Solution time, solvability and area comparison of selected solvers on 3-SAT problems with 50-60 variables. (g) Comparison of our solver's solution time with a DPLL solver. @KISSAT on CPU: C-language implementation of KISSAT on AMD EPYC7373X@3.0/3.73 GHz, *$\textit{Uniform Random 3-SAT}$: $N = 20, 50$ has 1000 instances each from `uf20-91/uf50-218` sets in SATLIB [87], $N = 30, 40$ has 1000 custom-generated instances each (according to A.6). **$\textit{High-Order K-SAT}$: (A) Semiprime Factoring: batch-07 in [102], (B) Uniform Random: 100 custom-generated instances for each (N, K) pair (same as in [70]). #Our solver's simulation done with macro extended to map $N = 100$, $M = 430$ size SAT problems. Other solvers include VLSI'25 [70], DaCTI [108], RBM on FPGA [103], SKADI [107], Snap-SAT [96] and WalkSAT, ARM from [96]. The multi-thread mode is implemented in (c)-(g).

random numbers corresponding to binary bits $\mathrm{rp}_{3:0}$ and $\mathrm{rn}_{3:0}$. Ignoring offset voltage and transconductance variations in the different pairs, the output of the p-bit is one if $(\mathrm{PXB}_i - \mathrm{PXM}_i) > \mathrm{ref}_p R_0$, where $R_0 = R_n - R_p$ is also a uniform random number. The differential voltage $(\mathrm{PXB}_i - \mathrm{PXM}_i)$ is a saturating nonlinear function of the variable's gain ($\Delta G_i = M_i - B_i$) (see Fig. 3.12), leading to an approximately sigmoidal dependence of the p-bit's output probability on the gain. Each p-bit receives eight random bits from the RNG buffer chain, with each of the four buffer stages supplying random bits for a total of sixteen p-bits. The circuit responsible for executing the variable update step and a detailed timing diagram of the solver is shown in Fig. 4.8.

4.5 Measurement Results

The experimental results for probabilistic-bits (Fig. 4.7b-c) reveal a relatively tight spread in the characteristics across all p-bits, arising primarily from the offset-voltage variations in the circuit's differential pairs. (The results also indicate that the slope of the p-bit function, i.e., the solver's temperature parameter T, can be dynamically adjusted by tuning the reference bias (ref_p), a feature that can be exploited in more advanced algorithms with annealing.) While prior SAT accelerators [70] have been restricted by structural hazards on the IMC fabric to single-thread operation, our circuit can interlace three independent solver threads for the same SAT problem in its three-stage pipeline (Fig. 4.8b). While the problem graph mapped to the SRAM array is reused, three sets of variable and clause registers are required for each thread to independently maintain its variable and clause statuses, respectively.

Similar to Chapter 3, we use the mean solution time metric to benchmark solver's latency. The solver's measured results on common SAT benchmarks show that multi-thread parallelism improves convergence by 1.7× while maintaining similar energy consump-

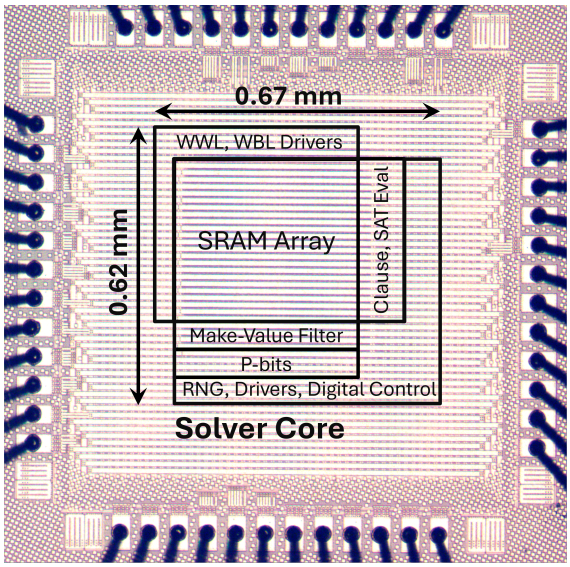
	ISSCC 2023 Snap-SAT		ISSCC 2024 VIP-SAT		VLSI 2025		ISSCC 2025 SKADI	ISSCC 2025 DaCTI		This Work	
Technology	65nm		65nm		55nm		28 nm	28 nm		55 nm	
Technique	Column wise In-memory OR		PE Array		WalkSAT, In-memory		DPLL Solver	Dynamic CT Injection		PCIM-SAT, In-memory	
Architecture	Digital		Digital		Mixed		Digital	Mixed		Mixed	
Memory Type (#transistors)	SRAM (6T)		Near-Memory		SRAM (10T)		SRAM (~20T)	Flip-Flop (>50T)		SRAM (11T)	
Chip Area (mm^2)	0.93		1.115		0.544		0.2	0.58		0.42	
Pre-processing	NO		YES		NO		NO	NO		NO	
Flips per iteration	Single		Multiple		Single		NA	Multiple		Multiple	
Arbitrary Order	Yes (up to 128)		No (3)		Yes (up to 64)		Yes (up to 50)	No (3)		Yes (up to 64)	
# Variables / Clauses	20/86	60/258	20/91	50/218	20/91	50/218	50/218	20/91	50/218	20/91	50/218
# Problems Tested	1000^D		100^C		100^C		1000^C	1000^C		1000^C	
Solvability[A] (%)	NA	72^H	100^H	98^H	100^H	98^H	100^G	100^H	100^H	100^H	100^H
Solution Time[B] (us)	70	713	7^E	18.7^E	5.12	45	17.1	1.6	31.7	0.39	5.5
Solution Energy[C] (nJ)	100	1098	2.1^E	20.8^E	59	518	58.0	7.8	268.9	11.8^F	265^F

[A] Number of SAT instances that can be solved in a given time, [B] Mean time/energy to find a solution across all instances in the set, [C] SATLIB Benchmark, [D] Benchmark not publicly available, [E] energy and latency incurred during off-chip preprocessing not taken into account, [F] (Mean Energy) = P x (Mean solution time), where P is the average power consumption of chip while solving SAT instances not including off-chip DACs used to supply reference biases ref0, ref1, ref2, refp, [G] Reported on satisfiable and unsatisfiable instances, [H] Reported on satisfiable instances. Incomplete/heuristic algorithms cannot prove unsatisfiability.

Figure 4.10: Features and performance comparison with prior SAT solver ASICs. Other solvers include ISSCC 2023 Snap-SAT [96], ISSCC 2024 VIP-SAT [98], VLSI 2025 [70], ISSCC 2025 SKADI [107] and ISSCC 2025 DaCTI[108].

tion (Fig. 4.9a). Independent threads parallelly explore different regions of the energy landscape, thereby increasing the ensemble's probability of reaching a solution early. Our chip's performance scaling is compared with several prior solvers not requiring pre-processing. Compared to prior state-of-the-art discrete-time incomplete solvers [70], our solver is 8–13× faster and 2–5× more energy efficient on uniform random 3-SAT problems, whereas it is 2–14× faster on practically relevant higher-order K-SAT problems. Compared to a continuous-time solver in a more advanced 28 nm node [108], our solver is 4–5.6× faster with comparable energy efficiency. Compared to a purely digital DPLL-based

(a)



(b)

Chip Summary	
Prototype	PCIM-SAT
Technology	55 nm CMOS
Core Area	0.42 mm^2
Supply	1.3 V
Frequency	100 MHz
Max Variables	64
Max Clauses	256
Max Order	64
Accuracy	100%@
Solvability	100%#
Mean Solution Time	0.39 us (20 var 3-SAT*)
	5.5 us (50 var 3-SAT**)
Mean Solution Energy	11.8 nJ (20 var 3-SAT*)
	265 nJ (50 var 3-SAT**)

@ Correctness of the solved problem, # Our solver achieves 100% solvability on all tested Uniform random 3-SAT (satisfiable) instances, * all instances from uf20-91 set in SATLIB, ** all instances from uf50-218 set in SATLIB.

Figure 4.11: (a) Die micrograph and (b) performance summary of prototype.

complete solver [107] also in 28 nm, our chip achieves a 3× speedup. Modeling in [107] further suggests an 84× speedup on larger problems.

4.6 Summary and Outlook

In summary, PCIM-SAT is the first silicon prototype (Fig. 4.11) to natively map arbitrary-order K-SAT problems and accelerate efficient multi-/parallel-variable update heuristics. Leveraging SRAM-based mixed-signal in-memory computing and a new probabilistic SAT-solving algorithm, it achieves 100% solvability on 50-variable 3-SAT (satis-

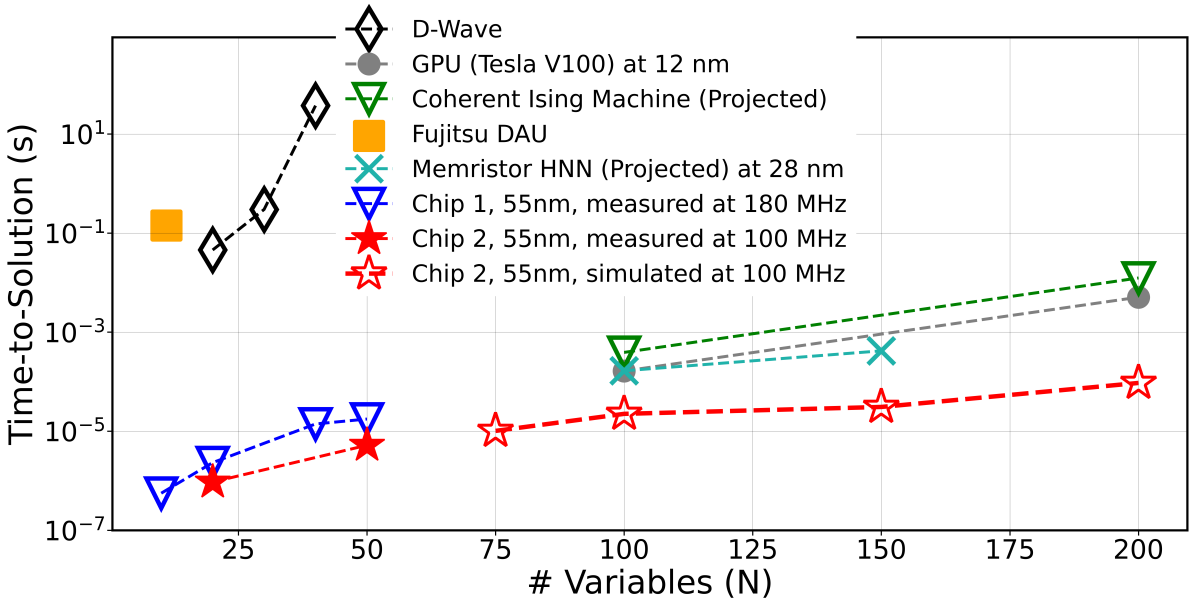


Figure 4.12: Scaling of Time-to-Solution (in seconds) with number of variables in uniform random 3-SAT instances. Definition of time-to-solution used here is according to Appendix A.9 and is the median TTS_{99} across instances of a particular size. Chip 1 refers to the chip presented in Chapter 3. Chip 2 (measured) corresponds to measured results presented in this chapter. Results of other solvers obtained from D-Wave [78], GPU and Coherent Ising Machine [114], Fujitsu Digital Annealer [115] and memristor HNN [37].

fiable) instances in 5.5 μs while consuming 265 nJ (Fig. 4.10, 4.11). The locally dense yet globally sparse structure of K-SAT problems, combined with the use of p-bits, enables seamless scalability to multi-tile architectures [73] (discussed later in Chapter 6), where all graph computations can be performed locally within each tile, eliminating the need for analog reductions across tiles and limiting inter-tile communication to simple digital exchanges of variable and clause values. Figure. 4.12 compares the performance of PCIM-SAT (as presented in this chapter and reffered to as Chip2 in the figure) and Chip 1 (as presented in previous Chapter 3) with other solvers including quantum annealing based D-Wave [78], GPU [114], Coherent Ising Machine [114], Fujitsu's Digital Annealer [115] and memristor-based Hopfield Neural Network [37]. The performance of our chip

is also projected for larger problems where the Algo 4 (see Fig. 4.3) is simulated within a multi-tile architecture framework(see [73] and Chapter 6), while assuming a clock frequency of 100 MHz and 55 nm CMOS technology node. It demonstrates the scaling prospects and superior performance of the presented hardware in the context of solving sparse Boolean Satisfiability problems.

Chapter 5

Framework for High-Order Ising Machines with In-Memory Computing

5.1 Background

High Order Ising Machines (HOIMs) are a generalization of the Ising Machine framework that can solve problems with higher order (k≥2) polynomial objective functions, that are widespread and arise in several scientific and industrial scenarios. Figure. 5.1 shows a top-level block diagram of the major hardware blocks that comprise a High-Order Ising Machine. Implementations of High-Order Ising Machines primarily differ in the design of their high-order gradient computation units (marked by FA and BA in Fig. 5.1) or the peripheral circuits that include the spin-based circuitry (marked by P1 and P2 in Fig. 5.1). While the latter enables implementations of different optimization solver dynamics and scales linearly with problem size, the former dominates the hardware footprint and has super-linear scaling.

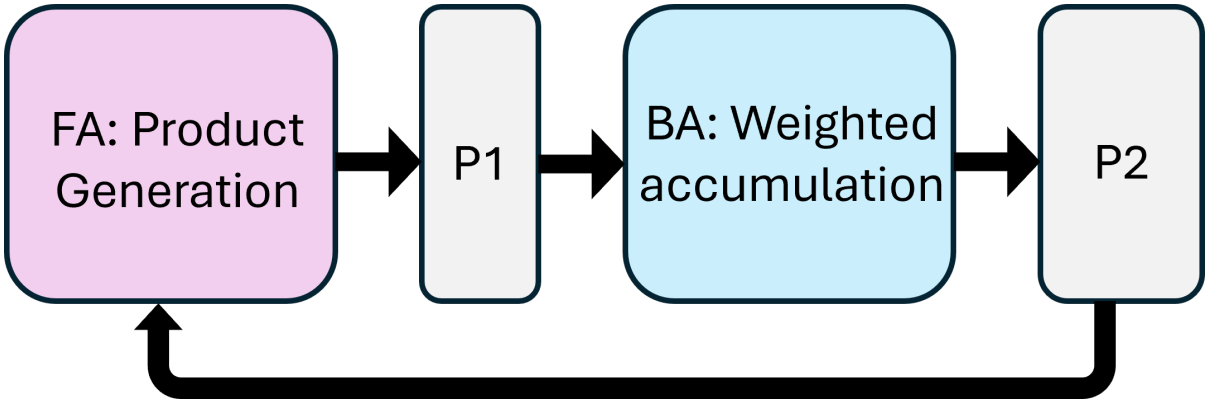


Figure 5.1: Block diagram of high-order Ising machines.

High-order optimization problems originate in different formulations; Polynomial Unconstrained Binary Optimization (PUBO) in molecular design and many-body physics, Conjunctive Normal Form (CNF) in Boolean/ Maximum Satisfiability, XNF comprising of XOR and/or disjunctive constraints and arising in cryptography applications, XOR-SAT. Depending on the formulation in which the problem is mapped and the specific way the high-order interactions are utilized to evaluate gradients, the size of such gradient computing circuitry could vary significantly. As crossbar array sizes scale super-linearly with problem size, such design decisions could considerably impact the hardware cost and overall scalability of high-order Ising machines. Furthermore, different input formulations impose distinct hardware requirements. For instance, XOR constraints can be evaluated using low-precision ADCs, whereas comparators are sufficient for evaluating high-order interactions arising in CNF/PUBO formulations. This leads to high-order Ising machines being tailored to specific problem classes, thereby sacrificing the universality of their second-order counterparts.

In this chapter, we provide a unifying framework in which all current implementations of high-order gradient computing circuits are classified into two distinct classes. We present a new mixed-mode circuitry that outperforms previous implementations in terms

of hardware costs. As part of our framework, we provide mapping and circuit schemes that allow any high-order discrete optimization problem to be mapped in their native formulation on to a reconfigurable and versatile hardware, that allows their gradients and other related metrics to be computed. This hardware then allows implementing any gradient minimization technique available for the Ising model, in their original formulation without requiring them to be converted to the standard high-order Ising model. Using competitive optimization heuristics, we show that area efficiency associated with such versatile hardware, could be exploited to minimize solution latencies, while consuming the same amount of energy.

5.2 Classification of gradient computing array

The gradient computing circuit introduced in Chapter 1, uses the forward array to evaluate all the original monomials (in the case of a PUBO) or clauses (in the case of CNF). We classify this implementation as monomial-style gradient computing. Figure. 5.2 a-b shows the expression of the pseudo partial-derivatives computed in this particular implementation for an exemplary CNF and PUBO. Note that the derivative equation requires evaluating each of the original clauses or monomials (here evaluation refers to sum of the normal or inverted variables in the clause/monomial followed by applying a Kronecker-delta function on the same). Figure. 5.2 c-d show the crossbar array arrangements for monomial-style gradient computing in a CNF and PUBO. The circles in forward array represent unit weights (determined by the presence of an edge in the bipartite graph formed by the variables/literals and monomials/clauses), those in the backward array denote the multi-bit weights of the respective clauses/monomials (as specified in the original problem), whereas absence of any circle represents a zero weight. The sums corresponding to the unit clauses/monomials (without their respective

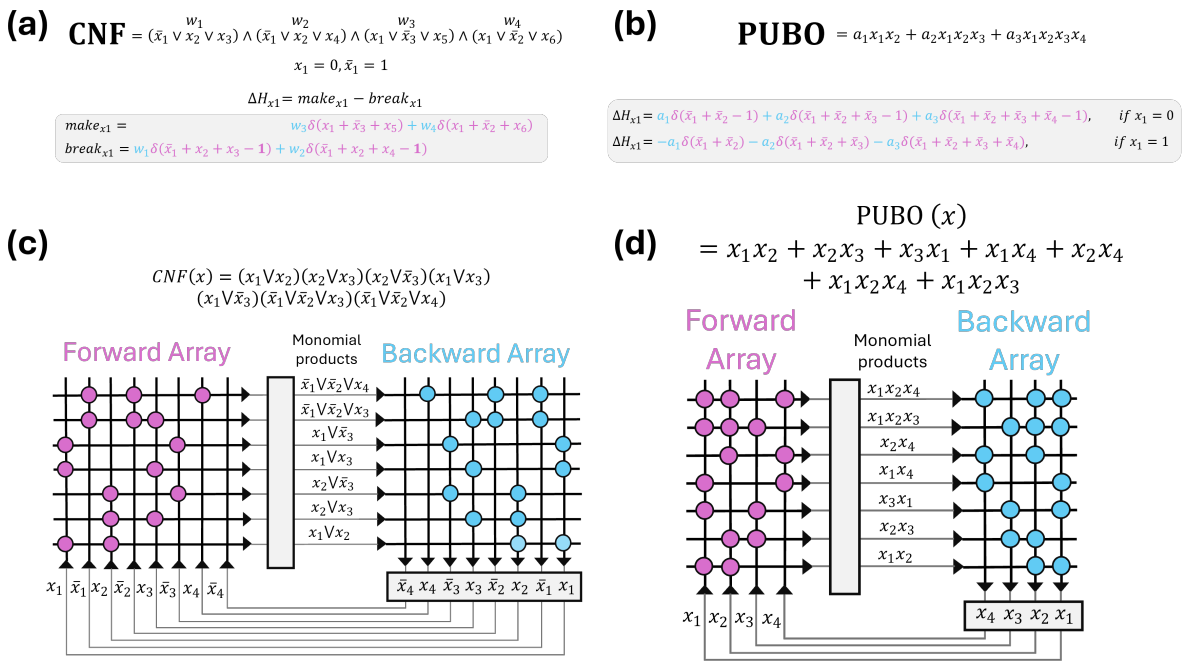


Figure 5.2: Example of a problem and expression of its pseudo partial-derivative using monomial-products, in (a) CNF and (b) in PUBO. Arrangement of the forward and backward arrays corresponding to the monomial-style implementation of a (c) CNF and (d) PUBO.

weights denoted by w_j) are evaluated in the forward array whereas the Kronecker-delta functions are implemented in the output-side periphery of the forward array. These unit clause/monomial-specific signals are then weighted by their coefficients/weights w_j's and accumulated for each variable/literal in the backward array whereas the output-side periphery of backward array implements circuits (not specified here) that determine the solver dynamics.

On the other hand, the first-order partial derivatives of a multilinear polynomial (over discrete or real valued variables) can be written as shown in Equation 5.1.

$$
\begin{aligned}
f(x,y,z) &= 4x + 3y - 2z + 5xy - yz + 7xz + 6xyz, \\
\frac{\partial f}{\partial x} &= 4 + 5y + 7z + 6yz, \\
\frac{\partial f}{\partial y} &= 3 + 5x - z + 6xz, \\
\frac{\partial f}{\partial z} &= -2 - y + 7x + 6xy
\end{aligned}
\tag{5.1}
$$

Each variable's partial derivative is the summation of the derivatives of the original monomials in which the variable is present. On applying a similar treatment to multi-linear arbitrary-order polynomials over discrete variables, the expressions of the pseudo partial derivatives of a CNF and PUBO can be expressed as shown in Fig. 5.3a-b. Note how the equation requires evaluating derivatives of the original clauses or monomials (here evaluation refers to sum of normal or inverted variables in the clause/monomial-derivative followed by applying a Kronecker-delta function on the same). As a result, we refer to this as the derivative-style gradient computing. Figure. 5.3 c-d show the crossbar array arrangements for derivative-style gradient computing in a CNF and PUBO. The circles in forward array represent unit weights, determined by the presence of an edge in the bipartite graph formed by the variables/literals and the first-order derivatives of monomials/clauses. All circles in a particular row of the backward array denote the multi-bit weights of the original clauses/monomials, whose derivative is assigned to that row. The sums corresponding to derivatives of unit clauses/monomials are evaluated in the forward array whereas the Kronecker-delta functions are implemented in the output-side periphery of the forward array. These clause/monomial-derivative signals are then weighted by their coefficients/weights and accumulated for each variable/literal in the backward array whereas the output-side periphery of backward array implements circuits (not specified here) that determine the solver dynamics.

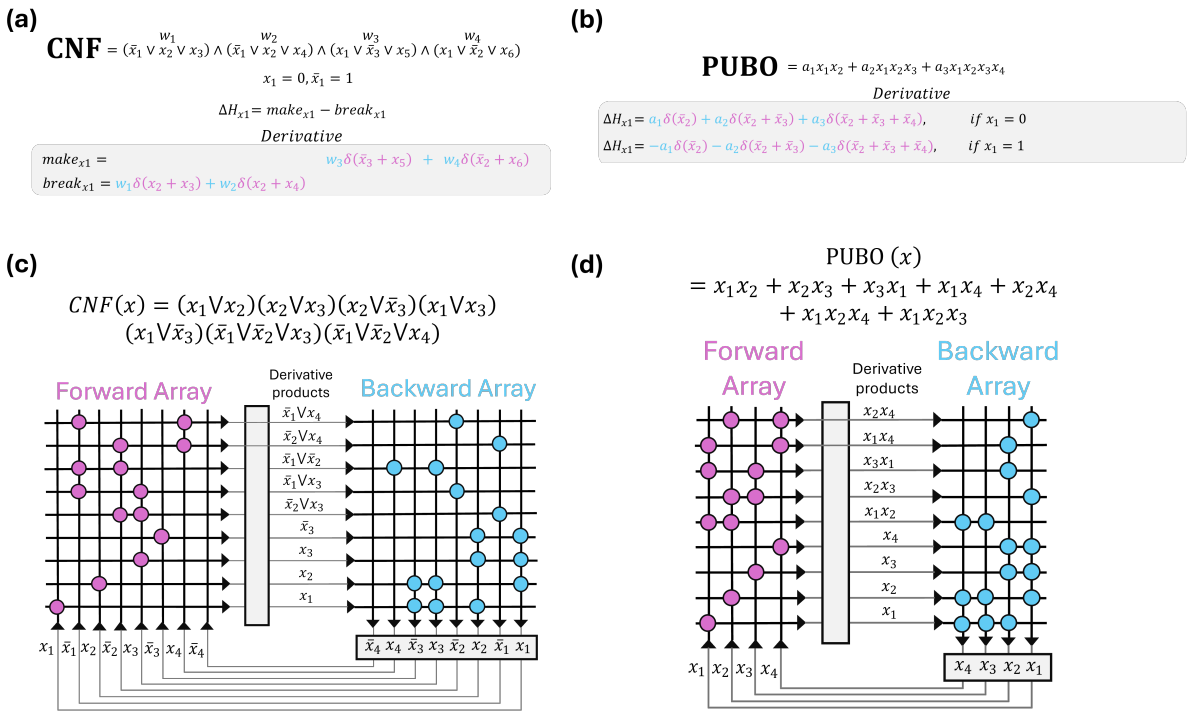


Figure 5.3: Example of a problem and expression of its pseudo partial-derivative using derivative-products, in (a) CNF and (b) in PUBO. Arrangement of the forward and backward arrays corresponding to the derivative-style implementation of a (c) CNF and (d) PUBO.

While both monomial-style and derivative-style circuits compute the pseudo gradients (change in objective function with respect to flipping the current state of the variables) required to guide solver dynamics, they differ primarily in the hardware complexity of the arrays. A derivative-style implementation requires a more compact hardware graph to map and solve a dense problem, compared to a monomial-style. A dense problem has many more monomials of a specific (or all) order than a sparse one. The number of all possible multilinear monomials of order K over N variables is given by $M^{max}_{monom} = {}^{N}C_{K}$, whereas the number of all possible derivatives of the same monomials is given by $M^{max}_{deriv} = {}^{N}C_{K-1}$. If N $\gg$ K (which is the case for many practical problems), then $M^{max}_{monom} > M^{max}_{deriv}$, which implies that the forward and backward array sizes (governed by M^{max}_{monom}, M^{max}_{deriv}) of

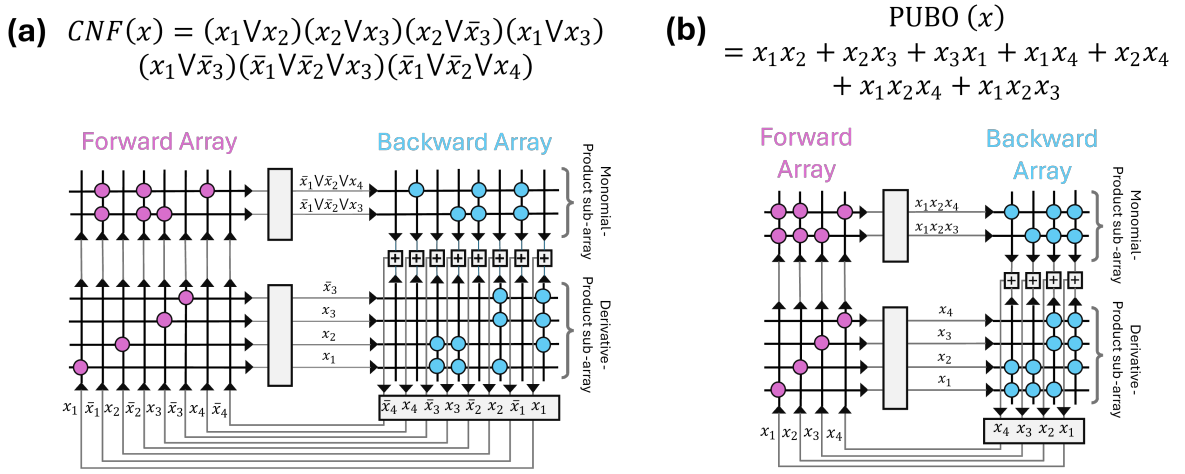


Figure 5.4: Hybrid-style gradient computing implementation in a (a) CNF and a (b) PUBO.

a derivative-style implementation would be lesser than that of a monomial-style one for denser problems. However, a problem can also be dense along a specific order and sparse along another. This leads to a mixed-mode style implementation that we refer to as the hybrid-style and crossbar arrangements for the same is shown in Fig. 5.4a-b. In this, for each order, a derivative-style or a monomial-style circuit is implemented depending on whether the polynomial is dense (number of monomials in that order greater than ${}^{N}C_{K-1}$) or sparse (number of monomials in that order lesser than ${}^{N}C_{K-1}$) in that order.

Figure. 5.5 shows the detailed circuit schematics of a hybrid-style gradient computing implementation with 1T1R crosspoint devices in CNF formulation. Although the forward and backward paths use separate crossbar arrays, each array embeds both the monomial-style graph and the derivative-style graph. This is achieved by carefully chosen transfer functions in the output-side periphery of the forward array and biasing scheme of the access lines of both arrays, as shown in Fig. 5.5. The gate and drain of all transistors (the lines vertically) in each column of the forward array are set to be proportional to the corresponding literal value. The source lines (horizontal lines) of forward array are all

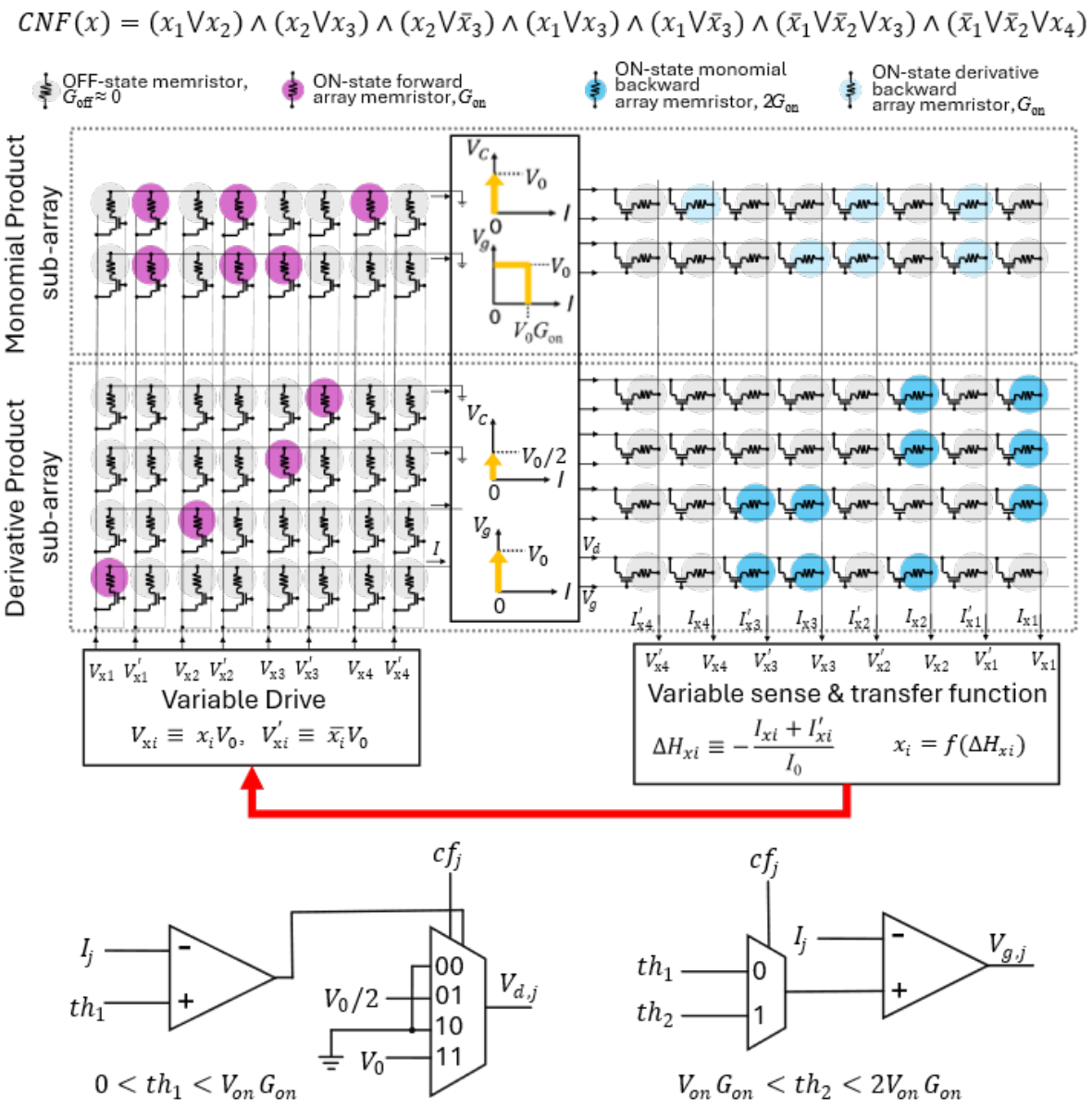


Figure 5.5: Circuit schematic of hybrid-style gradient computing implementation in a CNF with 1-transistor-1-resistor based crosspoint devices.

biased to a virtual ground of zero volts. This results in each source line carrying a current that is proportional to the sum of literals in each monomial (for the monomial-style sub-array) or derivative (for the derivative-style sub-array). This current is then passed to peripheral circuits consisting of comparator and multiplexers that determine the gate

and drain line voltages of the backward array. An additional row-wise configuration bit (denoted by cf_j) is required to denote whether the row corresponds to a monomial-product (cf_j=1) or derivative-product (cf_j=0). The overall biasing scheme of both arrays is similar to the one introduced in Fig. 2.7, except for the derivative-style rows in the backward array where both gate and drain lines in the backward array (running horizontally) are activated only when the derivative-product currents are zero. The source lines in the backward array (running vertically) are biased proportional to the literal values. The sum of currents in either source lines of each variable in the backward array is proportional to the pseudo partial derivative with respect to that variable.

5.3 Evaluating XOR constraints in-memory

Boolean Satisfiability problems are traditionally expressed in conjunctive normal form (CNF), where each clause is a disjunction of Boolean variables. However, many real-world applications naturally give rise to constraints expressed using exclusive-OR (XOR) operations—for example, channel decoding [116], model counting [117], and cryptographic decoding attacks [118]. Such problems are more naturally captured as hybrid XOR–CNF formulations that include both XOR and CNF clauses. Although XOR constraints can be converted into CNF using Tseitin transformations [119], this conversion substantially increases the number of variables and clauses, leading to degraded performance. As a result, hybrid XOR–CNF SAT solvers that directly support XOR clauses have been shown to significantly outperform solvers restricted to pure CNF [120].

However, unlike OR-type constraint (in CNFs) or AND-type constraints (in PUBOs), XOR is not a linearly separable function, which means the conventional affine transformation implemented in the forward array is not sufficient. Fig. 5.6 a and b describe two ways of evaluating XOR-type constraints. In type I (see Fig. 5.6a), each XOR

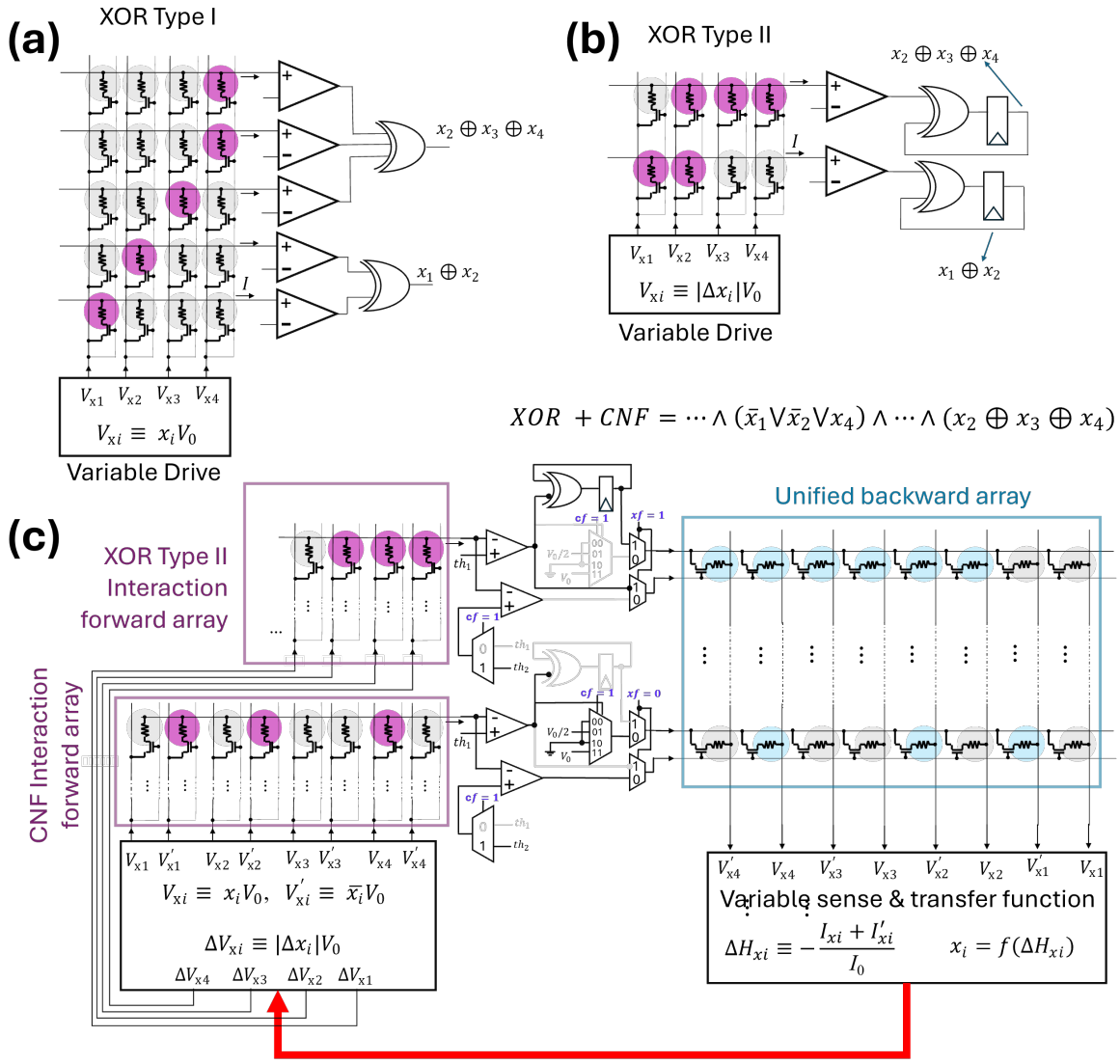


Figure 5.6: Evaluation of XOR constraints in-memory. Forward array circuit schematic for XOR constraint evaluation (a) type I and (b) type II. (c) Gradient computing circuit schematic of a hybrid XOR+CNF problem.

constraint has as many rows allocated to it as its order (K) and a comparator in each row followed by a K-input XOR gate is used to evaluate the constraint. The inputs to the array are set proportional to the variable values. However, this requires forward arrays with hardware complexity that scales with NMK, where N, M, K are the number of variables, XOR clauses and maximum order respectively. In type II (see Fig. 5.6b),

only a single row per constraint is required, but changes in variable values (since last iteration) are applied as inputs to the forward array. On the output-side periphery, a comparator, a 2-input XOR gate and a register is required. The register stores the value of the constraint from previous iteration and it is updated (flipped) if any of its constituent variables is flipped. Note that this works under the assumption that at most one variable per constraint is updated per iteration. Figure. 5.6c shows a detailed circuit schematic of a dual cross-coupled crossbar architecture implementing gradient computing in high-order hybrid XOR-CNF problem, with type-II circuit for evaluating the XOR constraints. While a single physical array is used in the backward path for accumulating gradient contributions from both XOR-type and OR-type constraints, two seperate physical arrays are required on the forward path for evaluating the two types of constraints. This is because the XOR constraint evaluation requires variable change as inputs whereas the OR constraint evaluation requires absolute variable values. A common output-side peripheral circuit is implemented for all constraints with an additional bit (xf) being used to differentiate between XOR and other constraints. Note that for each row of the backward sub-array allocated for XOR constraints, crosspoint devices of *both* literals of each variable (instead of the specific literal that is part of the constraint) are set to the weight (as specified in the problem) of the XOR constraint.

5.4 Modeling Results

We evaluated the hardware complexity of the different gradient computing circuits on the following benchmark problems: (i) Uniform random 3-SAT problems of different sizes from SATLIB [87]; (ii) uniform random (4,6,7)-SAT problems from [102]; (iii) select instances from SAT 2020 benchmark [121]; (iv) semiprime factoring problems formulated in CNF from (a) [122] and (b) [102]; (v) custom K-Regular-K-XORSAT instances for K

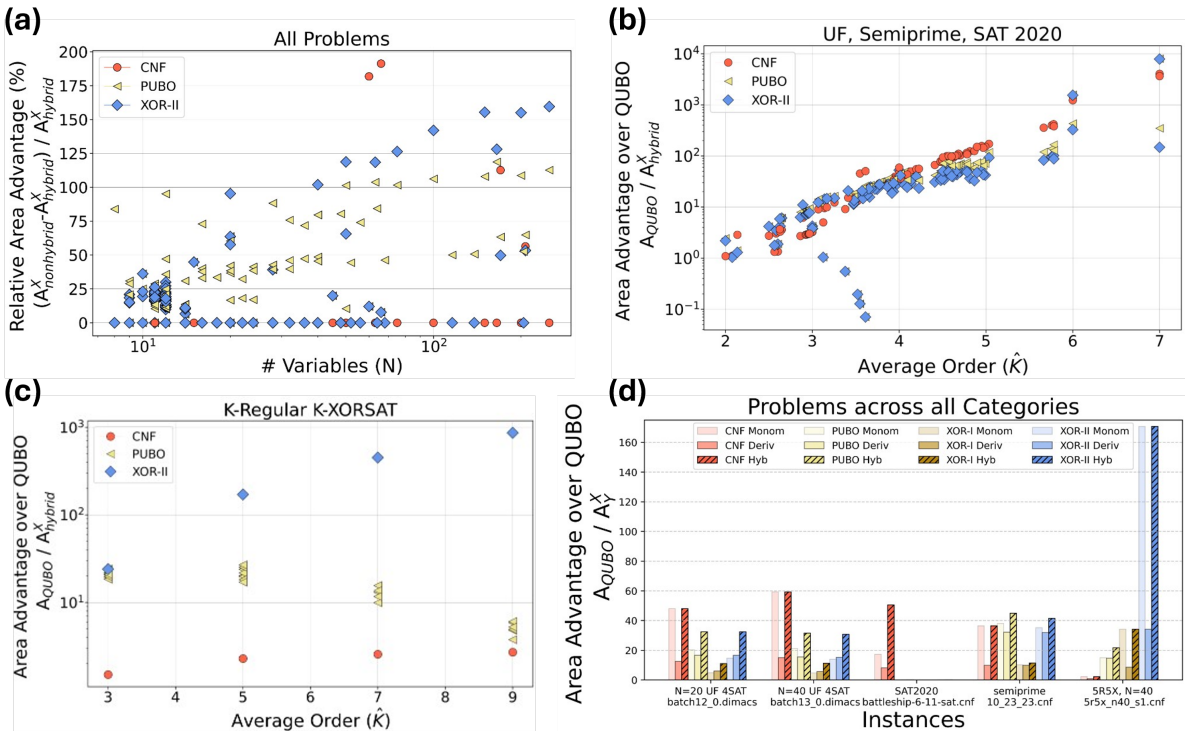


Figure 5.7: Comparison of the crossbar area given by total number of crosspoint devices, of different solvers on different problems across different formulations; (a) Relative area advantage of hybrid style circuit with respect to the most compact non-hybrid option; Scaling of the area advantage of hybrid style of high-order solver over a second-order solver with the average problem across instances from (b) uniform random, semiprime and SAT 2020 datasets and (c) K-Regular-K-XORSAT dataset; area advantage of high-order solver over second-order solver for selected instances in their different formulations and across different gradient computing circuit implementations.

= 3,5,7,9 generated following [123]. Problems in categories (i)-(iv) were natively present in CNF, whereas those in (v) were natively in XNF format (conjunction of XOR constraints). Furthermore, three different formulations of the objective function for each instance were considered: CNF, PUBO and XNF. All problems that were in CNF were also converted to PUBO (similar to Equation 1.7) and then to PUSO/XNF (see B.2,B.3), using variable substitution and polynomial expansion. Similarly problems in XNF were first converted to CNF with third order clauses using Tseytin transformation [119], which

was then converted to PUBO. All problem instances were also converted to QUBO from its PUBO formulation, in order to compare hardware costs of high-order circuits with second-order ones.

The total number of crosspoint devices required to implement the different flavours of gradient computing circuits for all three formulations of each instance was studied. Figure. 5.7a shows the relative area advantage of a hybrid style circuit with respect to the most compact non-hybrid style circuit (either monomial- or derivative-style). The hybrid style is at least as good as the most compact non-hybrid style circuit, while being up to 200% more compact on certain instances. From all subsequent analyses, we consider the hybrid style implementation due to its smaller hardware costs. Figure 5.7b,c illustrate the total crossbar area advantage achieved when mapping problems in their native high-order formulations using the hybrid-style gradient-computing hardware, compared to mapping their second-order formulations. The XOR formulation leads to the most compact hardware for the XORSAT problems compared to CNF or PUBO. This highlights the importance of hardware capable of handling XOR type constraints. Overall, the area advantage with respect to QUBO formulation seems to scale exponentially with the problem order. Figure. 5.7d gives a breakdown of the crossbar area advantage with respect to QUBO for few problem instances and their different formulations and gradient computing circuits.

The significant area advantages of efficient high-order solvers naturally raises questions as to whether area efficiency can be translated into improvements in convergence times. One way of achieving that is to launch multiple replicas of the solver as independent hardware solvers implemented in the spare die area. To that end, we compared convergence time of high-order solver with that of a second-order solver at iso-area. The number of replicas launched is given by $\left\lfloor \frac{A_{\text{QUBO}}}{A_{\text{High-Order}}} \right\rfloor$, where A_{QUBO} and $A_{\text{High-Order}}$ are the areas incurred in the crossbar arrays required for implementing the QUBO and hybrid-

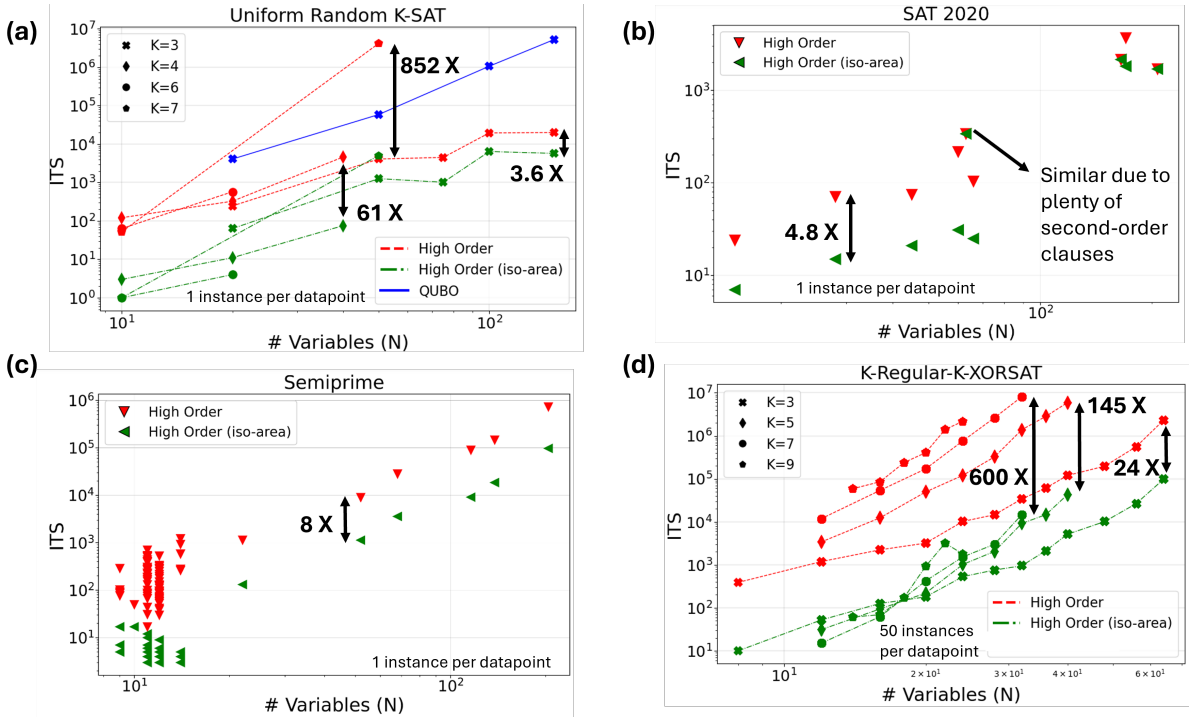


Figure 5.8: Scaling of iterations-to-solution with 99% certainty with problem size, of a hybrid-style high-order solver and its iso-area implementation with respect to a second-order solver, on (a) uniform random K-SAT problems, (b) SAT 2020 instances, (c) semiprime instances and (d) XORSAT instances.

style high-order solver. Figure. 5.8a shows the iterations-to-solution with 99% certainty (see Appendix A.9) of the high-order solver (in red), high-order solver at iso-area (in green) and a QUBO solver on uniform random K-SAT problems with different number of variables and order. The high-order solver dynamics implemented is the same as the one introduced in [124] whereas the second-order solver dynamics is the same as in Appendix A.8. Note that the results of the QUBO solver are only for K=3. Running multiple replicas seems to provide convergence advantages between 3.6 × to 852 ×, and the advantage seems to increase with the problem order. Similar trends are also observed for the other benchmark problems (see Figs. 5.8b-d).

The hardware complexity of all implementations so far was approximated by the

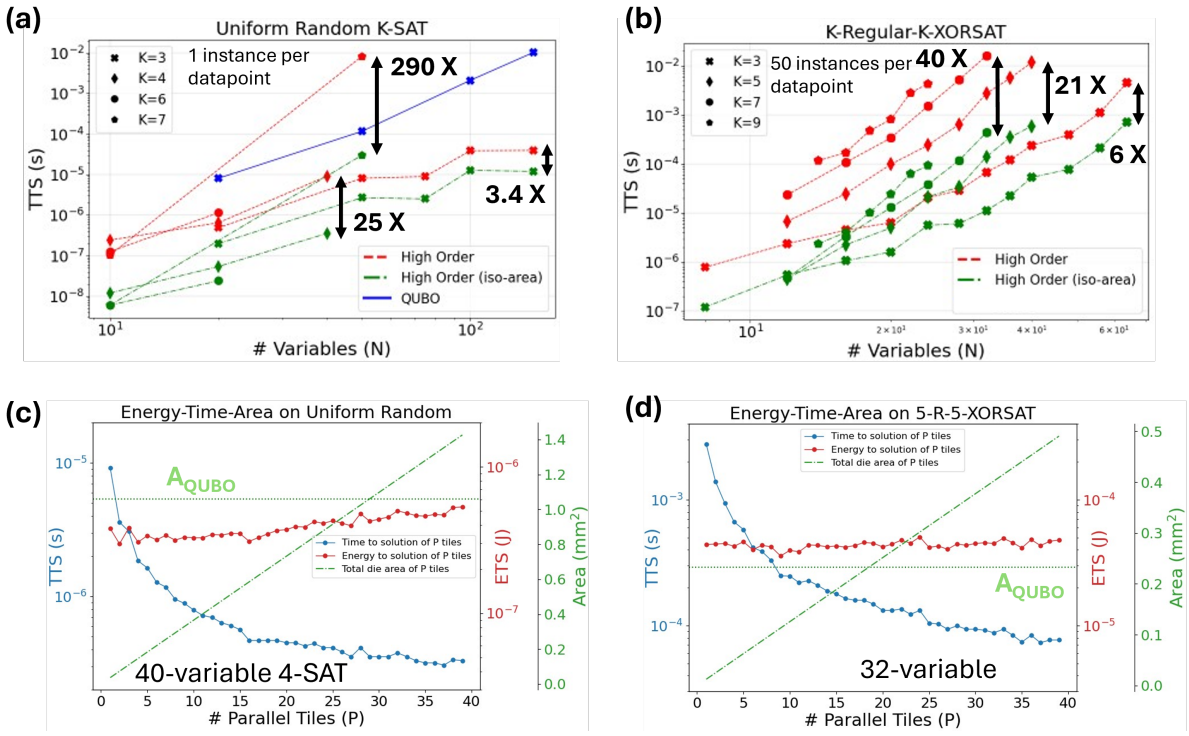


Figure 5.9: Scaling of physical time-to-solution with 99% certainty (in seconds) with problem size of a hybrid-style high-order solver and its iso-area implementation with respect to a second-order solver, on (a) uniform random K-SAT problems and (b) XORSAT instances. Scaling of the time-to-solution, energy-to-solution and area of high-order solver with the number of replicas for a (a) 40-variable uniform random 4-SAT problem and (d) a 32-variable 5-R-5-XORSAT problem.

crossbar area (given by total number of crosspoint devices). While the crossbar does occupy majority of the area in such solvers, the necessary peripheral circuits also occupy a non-trivial amount of area. Therefore, area models of the different circuit components (in 28 nm CMOS technology node) of a high-order solver and a second order solver were used to obtain realistic estimates of the different solvers[1]. Accounting for the area of the peripheral circuits reduces the overall area advantage between the high-order and second-order solvers (where the area advantage is still defined as $\left\lfloor \frac{A_{\text{QUBO}}}{A_{\text{High-Order}}} \right\rfloor$, but A_{QUBO}

[1]Energy, latency and area of all circuit components modeled in 28 nm CMOS technology node by Arne Heittmann, FZJ, Germany.

and $A_{\text{High-Order}}$ now correspond to the total solver area in mm^2, rather than only the number of crosspoints in the arrays). This, in turn, reduces the number of parallel replicas required to achieve iso-area. Figure. 5.9a,b reproduces the results from Fig. 5.8a,d but with realistic estimates of the number of replicas and plotting the physical time-to-solution in seconds (assuming clock periods that respects the latency estimates of different circuit components). While the convergence-time advantage under iso-area scaling does decrease, the benefits remain substantial, ranging from 25× to 290×. Figure. 5.9c,d show how the time, energy and area of the hig-order solver varies with increasing number of replicas (referred to by parallel tiles). The horizontal line in green denotes the area of the QUBO solver and the point where it intersects with the area of high-order solver, marks the iso-area sovler version. However, it is worth noting that even before iso-area conditions are met, the time-to-solution decreases substantially (by up to one order) at just about 10 replicas, while consuming similar energy as its non-replica version and still much less area than the QUBO solver.

5.5 Summary and Outlook

A framework for implementing gradient computation circuits in high order Ising machines was introduced, that classifies existing approaches into two categories. A new hybrid style gradient computing circuit was described that consistently achieves at par or reduced hardware costs compared to either category, across all considered high-order benchmark problems. Under reasonable assumptions, we demonstrate that arbitrary-order XOR interactions can be evaluated using a single comparator, thereby enabling ADC-free in-memory Ising machine implementations for any high-order discrete optimization problems in their native formulation. Using efficient optimization heuristics, we show how area efficient architecture can enable an even faster solution time at iso-energy,

by enabling a proportional number of parallel replicas to be run in the iso-area scenario. Such an iso-area comparison of HOIMs with second-order IMs reveals additional scaling advantages that increase with the problem's order. However, all analyses done till now was restricted to problems that could be mapped on to a single crossbar array. Larger more practical problems may not fit entirely in a single crossbar. Therefore, as a natural next step we look into ways of implementing multi-tile high-order Ising machines for solving large scale problems in the subsequent Chapter 6.

Chapter 6

Towards multi-tile architectures and solving large-scale problems

6.1 Introduction

Many large-sized hard combinatorial optimization problems (COPs) arising in practical and scientific scenarios [125, 126, 127] are difficult to solve on von-Neumann computers and could benefit from specialized accelerator hardware. Such problems are mapped to polynomial objective functions of order k with binary-valued variables ($x_i \in \{0, 1\}$) known as Polynomial Unconstrained Binary Optimization (PUBO) form or an equivalent Ising Hamiltonian with bipolar variables ($s_i \in \{-1, 1\}$) (see Chapter 1.2). Ising Machines (IMs) are promising dedicated hardware solvers for finding the global minima of such polynomial functions [21]. Although it is a common practice to first convert a PUBO ($k > 2$) to a Quadratic Unconstrained Binary Optimization (QUBO) form using known quadratization techniques [128] and then solve the optimization problem using Second-Order IMs (SOIMs), there is increasing evidence [37, 35, 69] that suggests solving the problem in PUBO using High-Order IMs (HOIMs) leads to orders of magnitude

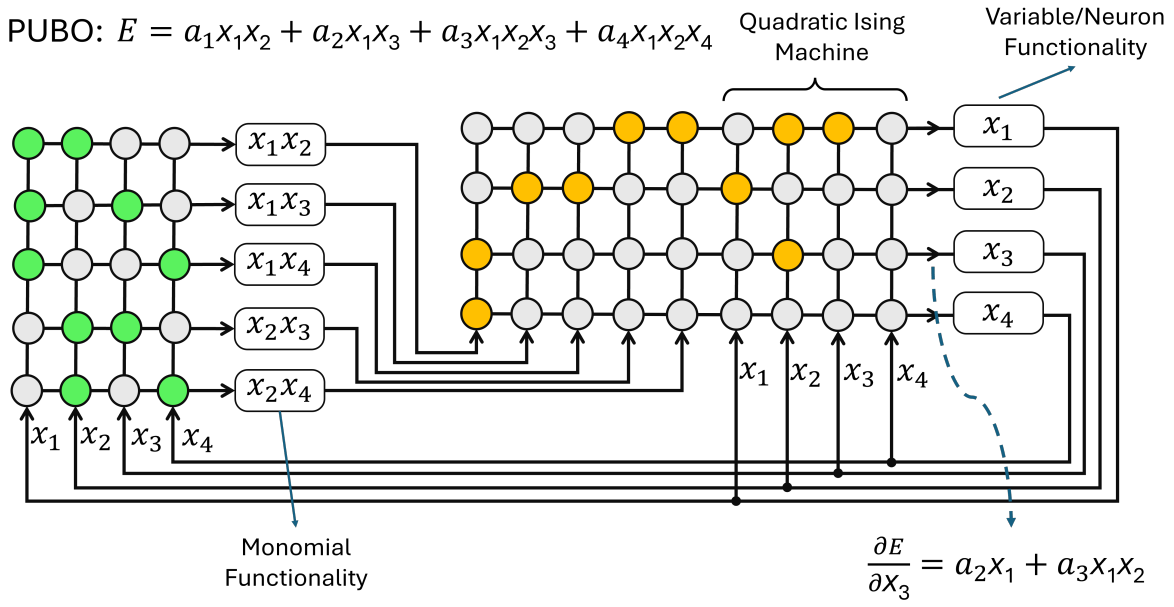


Figure 6.1: High-level implementation of a High-Order Ising Machine. The circles indicate locations of non-zero coupling weights corresponding to the considered PUBO example. For simplicity, this diagram does not show bias weights.

improvement in convergence time and hardware complexity.

A high-level schematic of a derivative-style HOIM (see Chapter 5 for definition of derivative-style) implemented using crossbar arrays is shown in Fig. 6.1. The array on the left computes derivative products by using Kirchoff's law to sum up the current contributions from respective member variables and then apply a nonlinear transfer function (e.g., step function for discrete-time Hopfield Neural Network [30], probabilistic-bit in Gibbs sampling-based solvers [34, 93], leaky-integrating neuron circuit for continuous-time Hopfield Neural Networks [33]). The output of this circuit and the variables are then summed up in the second array to finally compute the partial derivatives (partial pseudo-derivatives in the case of discrete-time systems) of the PUBO for each variable, which ultimately guides the dynamics of the HOIM.

Such dual-array-based implementations (presented in Chapter 2) have been shown to be more compact (with the area scaling with the product of number of variables and

monomials) than other proposals for HOIMs and respective SOIMs on a single crossbar [69]. However, most combinatorial optimization problems of practical relevance have thousands of variables that cannot be solved using a single crossbar, as array dimensions are limited due to IR-drop issues. Moreover, since in such hardware the number of physical resources scales with the problem size, the total area becomes an important metric to benchmark architectures. This is because in advanced processes, energy is largely dominated by data movement, and so energy consumption is expected to increase roughly proportional to linear circuit dimensions [66]. Moreover, a compact architecture could enable fitting the entire ising machine on a single chip, thereby reducing energy and latency-taxing inter-chip communication overheads.

A solution to this problem in neuromorphic accelerators is to break up a larger "logical" crossbar into multiple smaller physical crossbar circuits [66, 129]. For example, in the ISAAC architecture [66], a tile hosting a limited-size physical memory array generates analog partial dot-products. Partial product contributions from different tiles are summed in the digital domain after they undergo an Analog-to-Digital Conversion (ADC). However, PUBOs of most practical optimization problems are sparse, with the sparsity increasing with problem size [130]. They also have limited fan-in (number of monomials a variable is a part of), with the maximum fan-in being several orders smaller than the actual number of variables. Hence, all non-zero weights of a variable can be implemented in a single physical crossbar, with the dot-product being computed purely in the analog domain, thereby avoiding expensive ADCs. Such a sparsity-aware architecture was recently proposed for implementing second-order Ising machines [130].

However, such an architecture can only capture second-order interactions and is therefore limited to solving QUBOs. In this work, we report on a multi-tile architecture that is capable of implementing High-Order Ising Machines: the *High-Order Field Programmable Ising Array (HO-FPIA)*. It comprises in-memory computing tiles with two locally cross-

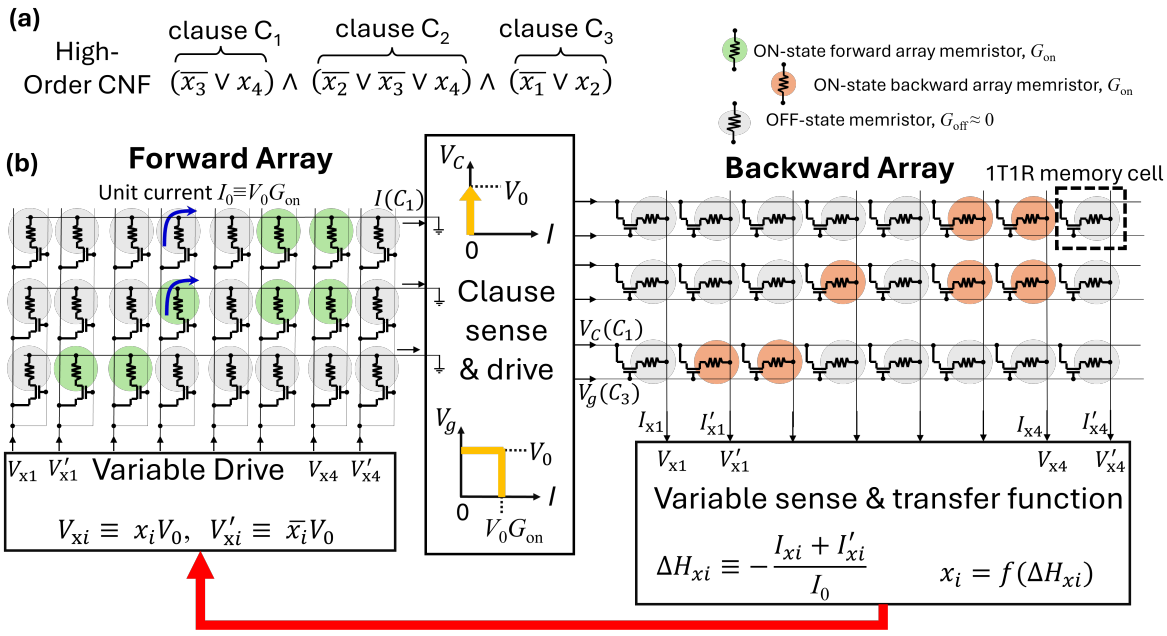


Figure 6.2: Parallel computation of gradients of high order polynomials encoded in Conjunctive Normal Form (CNF). (a) An example CNF consisting of three clauses and four variables. (b) Three terminal 1T1R crossbar circuit and peripheral functionality required for the gradient computation.

coupled crossbar arrays that compute partial derivatives of variables local to the tile and communicate with the other tiles using an Field Programmable Gate Array (FPGA)-like reconfigurable routing fabric. We leveraged open-source place-and-route tools developed for FPGA design to embed random, planned, and structured Boolean Satisfiability problems into the proposed architecture and compared its area efficiency with an ISAAC-like architecture and a sparsity-aware FPGA-inspired architecture limited to only second-order interactions. We considered both embedded flash [90] and a $6+2$T SRAM bit-cell as prospective candidates for the coupling cells in the crossbar arrays and their associated peripheral circuits for modeling the in-memory computing tiles. Our modeling results show that the proposed architecture is up to 116× and 43× more area-efficient compared to a naïve ISAAC-like architecture and the architecture in [130] that is capable of solving the QUBO version of the same problem, respectively. Further, we discuss how

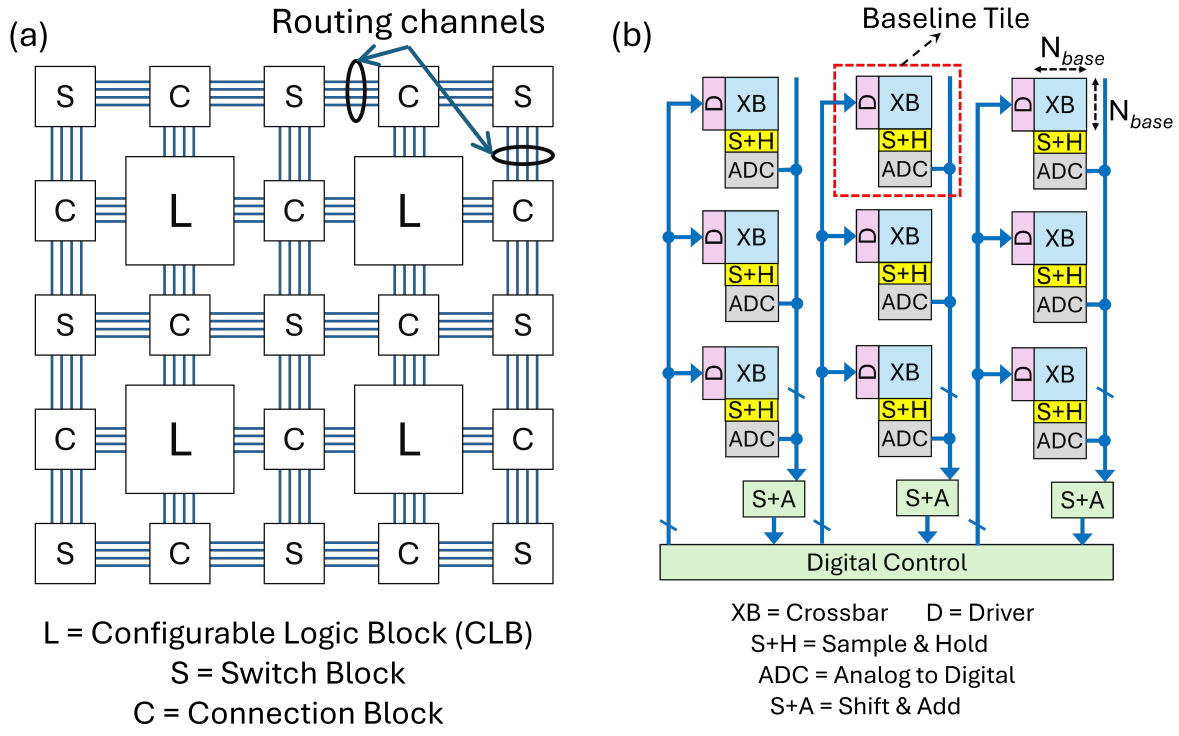


Figure 6.3: (a) Example island-style FPGA routing architecture and (b) baseline ISAAC-like architecture [66] for multi-tile crossbar-based neural network accelerators.

HO-FPIA can be several orders of magnitude faster and more energy-efficient than other architectures.

6.2 Background

6.2.1 Monomial-Style High-Order Ising Machines

k-Boolean Satisfiability, with $k > 2$, is an important class of hard combinatorial optimization problems that have a high-order objective function. They are natively present in the *Conjunctive Normal Form (CNF)*, which is a Boolean function comprising a conjunction (AND) of clauses (analogous to monomials in PUBO), where a clause is a

disjunction (OR) of literals, i.e., normal or complementary variables (see Fig. 6.2a and Chapter 1.1 for further details). The high-level implementation of a CNF-based HOIM is shown in Fig. 6.2b. A Voltage Drive (VD) circuit is used to drive the word-lines of the forward array with voltages proportional to the literal values. The current in the bit-lines (encoding the sum of literals in each clause) is sensed and compared against two fixed references, 0 and $V_{\text{on}}G_{on}$ (see inset in Fig. 6.2b), to generate clause-local voltages V_c and V_g, respectively, in the Clause Sense (CS) circuit. These clause-local signals are applied to the backward array by the Clause Drive (CD) circuit. For each pair of bit-lines corresponding to a variable, the Variable Sense (VS) circuit sets the bit-line bias proportional to the values of the literals and computes the partial derivative by sensing the sum of currents flowing in the lines. The variable's transfer function that determines the variable's next state based on the derivative information is also a part of this circuit.

Note that the high-order ising machine in Fig. 6.2b is a monomial-style one and that in Fig. 6.1 is a derivative-style. The former uses its crossbar arrays to map variables/literals to monomials/clauses, while the latter maps them to the first-order derivative products of the monomials/clauses in a PUBO/CNF, respectively (see Chapter 5 for more information). For the purposes of this work, we present all results on a CNF-based Monomial-style HOIM (as in Fig. 6.2b).

6.2.2 FPGA Architecture

A standard island-style Field Programmable Gate Array (FPGA) architecture is shown in Fig. 6.3a [131]. It comprises a large number of Configurable Logic Blocks (CLBs) that implement a small amount of digital logic and programmable routing that connects the CLB inputs and outputs to form a larger circuit. The logic blocks are groups of Look-Up Tables (LUTs) and Flip-Flops (FFs) along with local routing to intercon-

nect LUTs within a block. Global routing is achieved by vertical and horizontal routing channels surrounding the CLBs, where each contains several routing tracks (RCW). On these routing channels lie connection blocks, which connect the inputs and outputs of CLBs to the global routing tracks. Switch blocks are present at intersections of vertical and horizontal channels, which terminate or connect wire segments in one track to other tracks, in addition to buffering. The number of tracks each input and output pin of a CLB can connect to is defined by the architectural parameters F_I and F_O, respectively.

6.2.3 ISAAC-like Architecture

An exemplary multi-tile crossbar-based In-Memory Computing (IMC) accelerator architecture (inspired by the likes of ISAAC [66]) is shown in Fig. 6.3b. It comprises multiple tiles interconnected by a digital bus. Each tile is composed of a crossbar array (XB) receiving digital inputs from driver circuitry (D) and a sensing circuitry that is composed of Sample & Hold (S+H) and K Analog-to-Digital Converters (ADCs). To implement an HOIM in such an architecture, we assume that the full logical forward and backward matrices corresponding to the optimization problem's CNF are partitioned in a row-major or column-major order into the smaller crossbar arrays present in each tile. For example, a $P \times P$ array would be mapped to an array of $(P/N_{\text{base}}) \times (P/N_{\text{base}})$ tiles, each comprising $N_{\text{base}} \times N_{\text{base}}$ crossbar arrays. Such implementations lead to a simple address-based routing scheme and require little to no pre-processing.

The downside is that, since such a partition scheme does not guarantee full dot-product computation within the same array, ADCs are required to convert partial analog dot-products in each array's bit-lines to digital values, which are then aggregated in the digital domain. Usually, in such architectures, the crossbar arrays compute dot-products in one cycle, and the analog results are latched in the S+H circuit. Subsequently, the

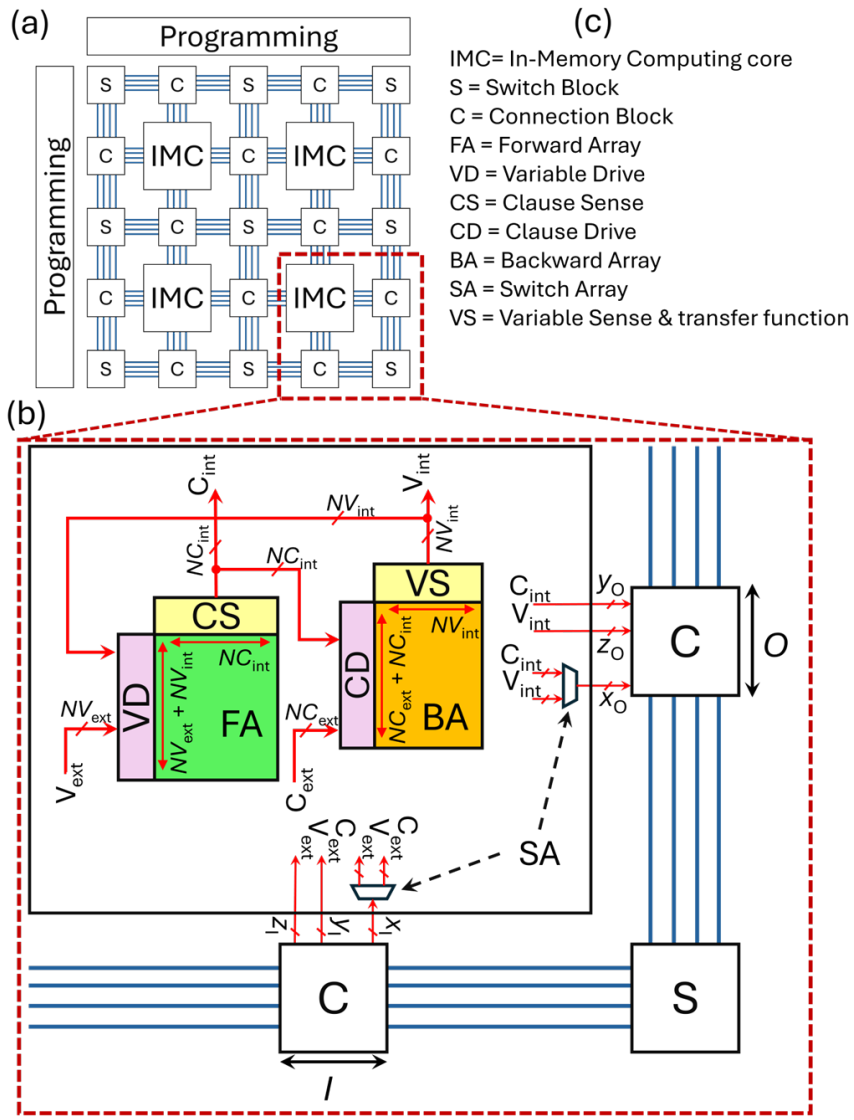


Figure 6.4: (a) Top level architecture of HO-FPIA, (b) zoomed in view of a tile showing in-memory computing tile consisting of mixed-signal blocks and neighboring routing fabric and (c) naming of different blocks.

N_{base} analog values are converted to digital using K parallel ADCs in a pipelined manner.

6.3 High-Order Field Programmable Ising Array

6.3.1 High-Order In-Memory Tile

The proposed High-Order Field Programmable Ising Array (HO-FPIA) architecture resembles an island-type FPGA [131], with the Configurable Logic Blocks (CLBs) replaced by mixed-signal In-Memory Computing (IMC) tiles and additional crossbar array programming circuitry on the periphery (Fig. 6.4). Block-level details of a single tile comprising the IMC tile and neighboring switch and connection blocks are shown in Fig. 6.4b. The tile comprises a Forward Array (FA) and a Backward Array (BA), along with their drive and sense circuits: Variable Drive (VD), Clause Drive (CD), and Variable Sense (VS), Clause Sense (CS), respectively (similar to Fig. 6.2).

The tile has I input pins and O output pins that are connected to a fraction (F_I and F_O, respectively) of the nearby routing tracks. Here, NV_{ext} and NC_{ext} are the numbers of variable and clause signals external to the tile (generated in other tiles) and together comprise the I input pins ($I = NV_{\text{ext}} + NC_{\text{ext}}$). Whereas NV_{int} and NC_{int} are the numbers of variable and clause signals that are generated inside the tile and passed to global routing through the O output pins ($O = NV_{\text{int}} + NC_{\text{int}}$). The FA and BA have NC_{int} and NV_{int} outputs, corresponding to the internally generated clause and variable signals, respectively. Both FA and BA have inputs arriving externally at the input pins (NV_{ext} and NC_{ext}), as well as local inputs that are outputs of the other array (NV_{int} and NC_{int}) in the same tile. As a result, FA and BA have dimensions of $(NV_{\text{ext}}+NV_{\text{int}})\times NC_{\text{int}}$ and $(NC_{\text{ext}} + NC_{\text{int}}) \times NV_{\text{int}}$, respectively.

As will be seen in later sections, not all inputs or outputs of the two arrays get assigned to valid variables or clauses, in which case all devices in those rows and/or columns are set to zero (off-state). Moreover, the resulting distribution of the types of signals (variables or clauses) arriving at the I/O pins varies between tiles. To accommodate such

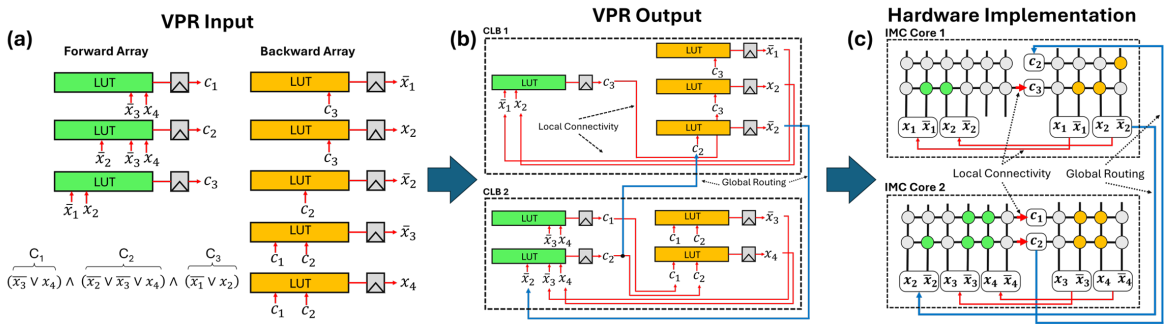


Figure 6.5: (a) Connectivity model represented using lookup tables and flip-flops of the example CNF shown in inset, that is provided as input to VPR, (b) high-level representation of the packed connectivity model provided as output by the packing stage of VPR, and (c) its crossbar-based hardware implementation.

configurability, the input (output) pins are connected to x_I (x_O) multiplexers that can dynamically assign the pin to either variable or clause wires, in addition to being connected to z_I (z_O) fixed variable wires and y_I (y_O) fixed clause wires. These multiplexers, along with the SRAM cells that store their configuration bits, comprise the Switch Array (SA).

6.3.2 Fan-In aware mapping with VPR

Due to the similarities with island-type architectures, many tools and algorithms for implementing circuits on FPGAs can be re-purposed to implement the proposed High-Order Field Programmable Ising Array (HO-FPIA). In this study, similar to [130, 132], we leverage the open-source Versatile Place and Route (VPR) tool [133] to pack, place, and route k-SAT problems in CNF onto our proposed architecture.

Figure 6.5 demonstrates the "pack" step of the VPR flow customized for HO-FPIA using a toy problem in CNF (Fig. 6.2a). Figure. 6.5a shows a graphical representation of the netlist corresponding to the CNF that is provided as input to VPR. Each clause and variable output of the Forward Array (FA) and Backward Array (BA) (in Fig. 6.2b), respectively, are mapped to a Flip-Flop (FF) to ensure a circuit without combinational

loops. The inputs to the FA and BA sharing non-zero coupling weights with the output nodes (in Fig. 6.2b) are mapped to respective equivalent-width Look-Up Tables (LUTs).

In the packing step, the goal is to pack connected LUTs together in the same block to minimize the number of signals that must be routed between blocks and to attempt to fill each block to its capacity in order to minimize the total number of blocks used. As a result of this optimization goal and the specific way of generating the input netlist, the majority of the trivial (zero-weight) connectivity is filtered out, and variables/clauses end up having all their input connections (fan-ins) in the same array. Figure 6.5b shows the packed output netlist file. Clauses 1 and 2 and literals $\overline{x}_3$ and x_4 have the highest number of shared inputs and therefore get clustered into the same block (CLB2). CLB1, on the other hand, requires clause 2 as an input, so it is globally routed from CLB2. Each of these CLBs is then mapped to an IMC tile with fixed FA and BA dimensions, as shown in Fig. 6.5c.

6.3.3 Modeling Framework

The table in Fig. 6.6 shows all architectural parameters relevant for VPR modeling and results. I, O, F_I, and F_O are free parameters that are varied to optimize the architecture for different instances or classes of problems. Due to the non-homogeneous fan-in spread of variables and clauses in k-SAT problems, the number of I/O pins of each tile that are of variable and clause types is not uniform across tiles. Therefore, NV_{ext} (NC_{ext}) and NV_{int} (NC_{int}) are set to the maximum number of input and output pins, respectively, of variable (clause) type occurring in any IMC tile. Subsequently,

$$x_I = \min(NC_{\text{ext}}, NV_{\text{ext}}) - I + \max(NC_{\text{ext}}, NV_{\text{ext}}), \qquad z_I = NV_{\text{ext}} - x_I, \qquad y_I = I - x_I - z_I.$$

	eFlash (opt.)	eFlash (pess.)	SRAM	Comments
A_{fetmin}	50	50	50	Minimum transistor area
A_{cell}	60	180	200**	Memory cell area (per weight)
A_{VD} / A_{CD}&&	1500	1500	1500	Variable/Clause Drive periphery (per word-line)
A_{VS} / A_{CS}&&	2500	2500	1250	Variable/Clause Sense periphery (per bit-line)
A_{wip}	11500	8200	1550	Input Programming periphery (per word-line)
A_{wop}	0	5300	775	Output Programming periphery (per bit-line)
S_{prog}	1000	1000	1000	Programming periphery time-multiplexing factor
A_{SA}	220	220	220	Switch array (per I/O pin)
N_{base}	256	256	256	Baseline array linear dimension
N_{windows}	-	-	-	Number of baseline tiles/windows
A_{ADC}	1.45E+06	1.45E+06	1.45E+06	8-bit ADC area in baseline
K	-	-	-	# of ADCs per baseline tile
$N_{\text{I,FA}}$	$\text{NV}_{\text{ext}} + \text{NV}_{\text{int}}$			Input dimension of FA
$N_{\text{O,FA}}$	NC_{int}			Output dimension of FA
$N_{\text{I,BA}}$	$\text{NC}_{\text{ext}} + \text{NC}_{\text{int}}$			Input dimension of BA
$N_{\text{O,BA}}$	NV_{int}			Output dimension of BA

Figure 6.6: Architectural parameters and model assumptions. Areas are expressed in units of F^2, where F denotes the technology minimum feature size. ** The SRAM cell area is smaller than that in [130] due to the CNF-monomial-based HOIM requiring only binary weights for k-SAT problems. &&Furthermore, in [130], the drive periphery area A_{VD} or A_{CD} is referred to as A_{rip}, and the sense periphery area A_{VS} or A_{CS} is referred to as A_{rop}. ADC model adapted from [134].

The output (x, y, z) parameters are set in a similar fashion. For the routing, we use Wilton topology switch blocks [131] and set 10% of all tracks in each channel to length = 1 ($L_{\text{span},1}$) to connect nearby tiles efficiently and the rest to length = 4 ($L_{\text{span},2}$) to connect tiles that are farther apart. The total number of tiles (M^2) and the optimized routing channel width (RCW) are extracted directly from the VPR output file.

In this study, we focus on three representative crossbar-compatible memory cell technologies: embedded flash (eFlash) with optimistic and pessimistic area scaling models (the latter corresponding to modifications accommodating single-bit erase [90]), and SRAM. For the latter, a 6+2T SRAM cell with scaled-width read transistors compatible with analog IMC [135] is considered. Parameters used for area and performance modeling are summarized in Fig. 6.6 and are similar to [130]. The total crossbar area in HO-FPIA

and the baseline architecture are estimated as in (6.1) and (6.2).

$$A_{\text{xbar,HO-FPIA}} = M^2 \left\{ N_{O,\text{FA}} N_{I,\text{FA}} + N_{O,\text{BA}} N_{I,\text{BA}} \right\} A_{\text{cell}} \tag{6.1}$$

$$A_{\text{xbar,base}} = N_{\text{windows}} N_{\text{base}}^2 A_{\text{cell}} \tag{6.2}$$

The linear crossbar dimension in baseline architecture (N_{base}) is set to 256. Although different linear crossbar dimensions are used in HO-FPIA, its maximum possible value is also set to 256. For a problem with N_V variables and N_C clauses, the number of baseline tiles (N_{windows}) can be simply calculated as

$$N_{\text{windows}} = \left\lceil \frac{4 N_V N_C}{N_{\text{base}}^2} \right\rceil .$$

The baseline architecture has its tiles arranged in a $\lceil \sqrt{N_{\text{windows}}} \rceil \times \lceil \sqrt{N_{\text{windows}}} \rceil$ square matrix, whereas the HO-FPIA has its IMC tiles in an $M \times M$ square matrix. The total die areas for HO-FPIA and baseline architectures are then given by (6.3) and (6.4).

$$\begin{aligned} A_{\text{HO-FPIA}} = A_{\text{xbar,HO-FPIA}} + M^2 \big\{ & A_{\text{VD}} N_{I,\text{FA}} + A_{\text{CS}} N_{O,\text{FA}} + A_{\text{CD}} N_{I,\text{BA}} + A_{\text{VS}} N_{O,\text{BA}} \\ & + (x_I + x_O) A_{\text{SA}} \big\} + A_{\text{routing}} \\ & + \frac{M \left[(N_{I,\text{FA}} + N_{I,\text{BA}}) A_{\text{wip}} + (N_{O,\text{FA}} + N_{O,\text{BA}}) A_{\text{wop}} \right]}{S_{\text{prog}}} \end{aligned} \tag{6.3}$$

$$\begin{aligned} A_{\text{base}} = A_{\text{xbar,base}} + N_{\text{windows}} & \left\{ (A_{\text{VD}} + A_{\text{VS}}) N_{\text{base}} + K A_{\text{ADC}} \right\} \\ & + \frac{\lceil \sqrt{N_{\text{windows}}} \rceil \, N_{\text{base}} (A_{\text{wip}} + A_{\text{wop}})}{S_{\text{prog}}} \end{aligned} \tag{6.4}$$

The first term on the right-hand side of (6.3) and (6.4) corresponds to the total area across all crossbar arrays as expressed in (6.1) and (6.2), respectively. The second term in

each equation represents the total area occupied by the peripheral circuits of the crossbar arrays across all IMC tiles or baseline tiles, including drive, sensing, switch arrays, and ADCs. The third term in (6.3) corresponds to the total area occupied by the components of the routing fabric, such as switch and connection blocks. This term is computed by VPR and is proportional to the total number of IMC tiles, the fraction of nearby routing tracks that each input and output pin of the IMC tile are connected to (F_I and F_O, respectively), and the number of routing tracks per channel (RCW). The final term in both (6.3) and (6.4) denotes the total area occupied by the peripheral circuits used to program the coupling cells.

Specifically, the programming circuit of a crossbar can be of two types: input programming periphery (A_{wip}, used to drive each word-line) and output programming periphery (A_{wop}, used to drive each bit-line). Owing to their large footprint, programming peripherals are often time-multiplexed (say, with a factor of S_{prog}), where the same input (output) programming circuit block is used to drive S_{prog} different word-lines (bit-lines) in S_{prog} consecutive time periods. For example, a crossbar in the baseline architecture with linear dimension N_{base} has input and output programming circuit areas equal to $N_{\text{base}}A_{\text{wip}}/S_{\text{prog}}$ and $N_{\text{base}}A_{\text{wop}}/S_{\text{prog}}$, respectively. In both HO-FPIA and the baseline architecture, the input (output) programming circuit is assumed to be shared by all crossbars arranged in a single row (column). As a result, the total programming circuit area also scales with the number of rows/columns of tiles or IMC tiles in the architecture.

The eFlash and SRAM area assumptions are based on the work in [90] and [136], respectively. Sensing, driving, and programming overhead assumptions for eFlash are similar to [66]. Periphery parameters for the SRAM implementation are taken from internal layouts of a related SRAM-based IMC chip [130, 70]. The number of ADCs per baseline tile (K) depends on the target operating frequency (f_{clk}). For $f_{\text{clk}} = 100\,\text{MHz}$ (a reasonable assumption considering up to 2.5 ns operation of crossbars comprising the

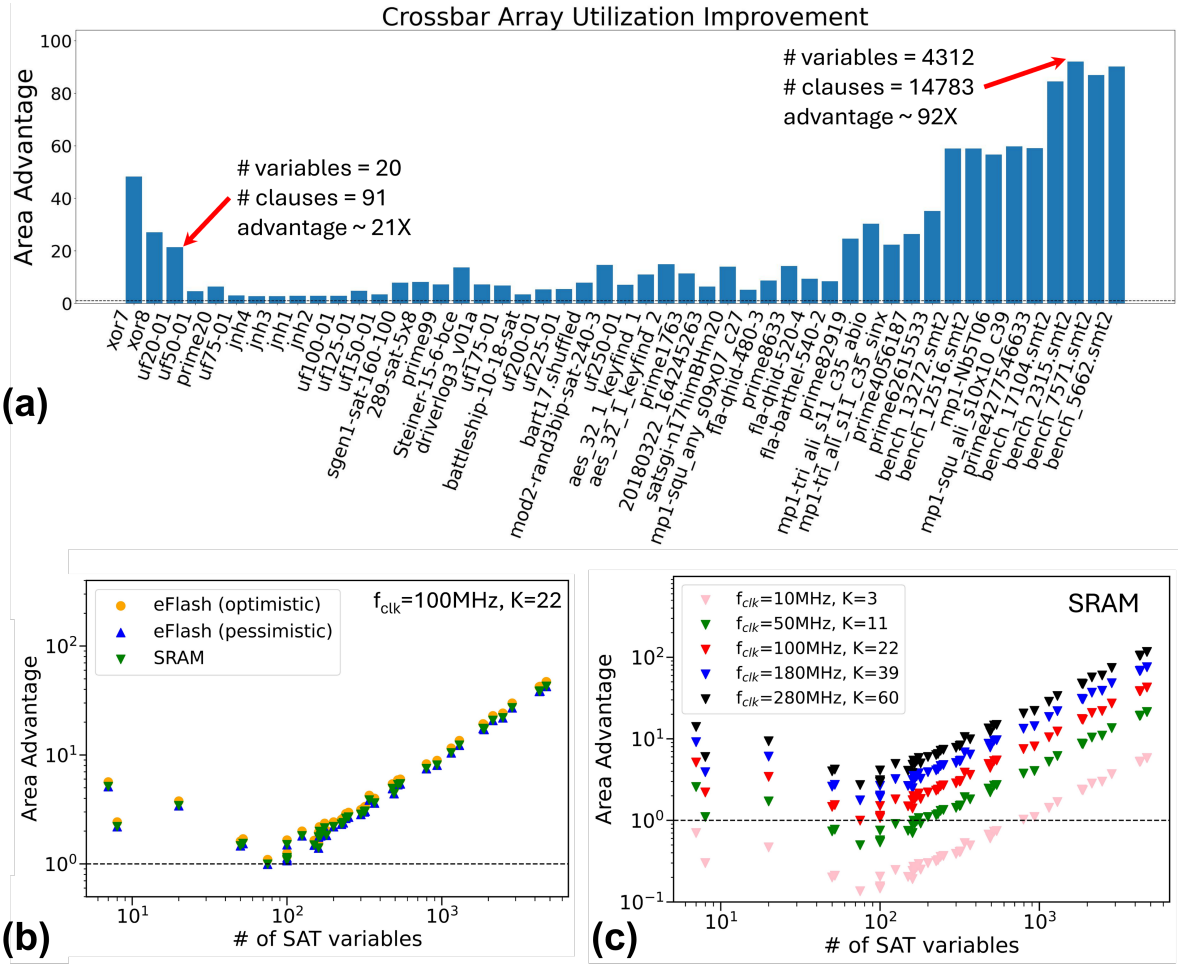


Figure 6.7: (a) Crossbar area advantage expressed as the ratio of total baseline crossbar area to total HO-FPIA crossbar area for all the considered benchmark problems, shown in ascending order with respect to their variable count. Total area advantage for (b) different coupling cell technologies and (c) operating frequency and K (which is the number of ADCs per baseline tile) vs number of variables of the considered benchmark problems.

considered coupling cell technologies [137, 138]) and a 1.2 Gsamples/s ADC [134], K must be 22 to convert all 256 analog dot-products in one cycle. We neglect overheads of digital processing and annealing circuitry (which can be packed in the unused corners of a tile) for both FPIA and baseline designs, as well as the (rather substantial) overhead of digital final summation in the baseline architecture.

For benchmarking, we consider random, planned, and structured k-SAT problems from publicly available datasets: SATLIB [87], SAT2020 [121], SAT2017 [139], JHN DIMACS [87], and custom-generated k-input XOR problems and semiprime factoring. Our primary figure of merit is the area advantage of HO-FPIA over the baseline, given by $A_{\text{base}}/A_{\text{HO-FPIA}}$. We use SMAC3 [140] to explore the architectural parameter space and optimize the area advantage across these benchmark problems.

6.4 Results

Fig. 6.7a shows the total crossbar area advantage ($A_{\text{xbar,base}}/A_{\text{xbar,HO-FPIA}}$) for the considered problems with the architectural free parameters optimized locally for each problem instance. The advantage grows as the problem size increases ($\sim 92\times$ for ~ 4000 variable problems vs $21\times$ for 20 variable problems). This result highlights the sparsity-aware feature of the packing step and how the sparsity (fraction of coupling weights in the large logical array that is zero) increases with problem size. This is generally true, for example in uniform random k-SAT problems, the sparsity is proportional to k/N_V, where k is the order of the problem. However, although this sets the motivation for using sparsity-aware techniques to implement HOIMs, it does not reflect the true area advantage that includes the area for routing and other periphery. Moreover, in a chip based on the proposed architecture, the architectural parameters (I, O, F_I, F_O, RCW) would all be fixed irrespective of the problem being solved.

Therefore, we next focus on a fixed HO-FPIA architecture based on fixed quasi-optimal architectural parameters: $I = 128$, $O = 13$, $F_I = 0.1$, $F_O = 0.3$, $N_{I,\text{FA}} = 254$, $N_{O,\text{FA}} = 13$, $N_{I,\text{BA}} = 141$, $N_{O,\text{BA}} = 20$, determined from previous modeling experiments, and the routing channel size RCW $= 266$ that is sufficient for successful mapping of all studied problems. Note that the selected $N_{I,\text{BA}}$ must not be smaller than the maximum

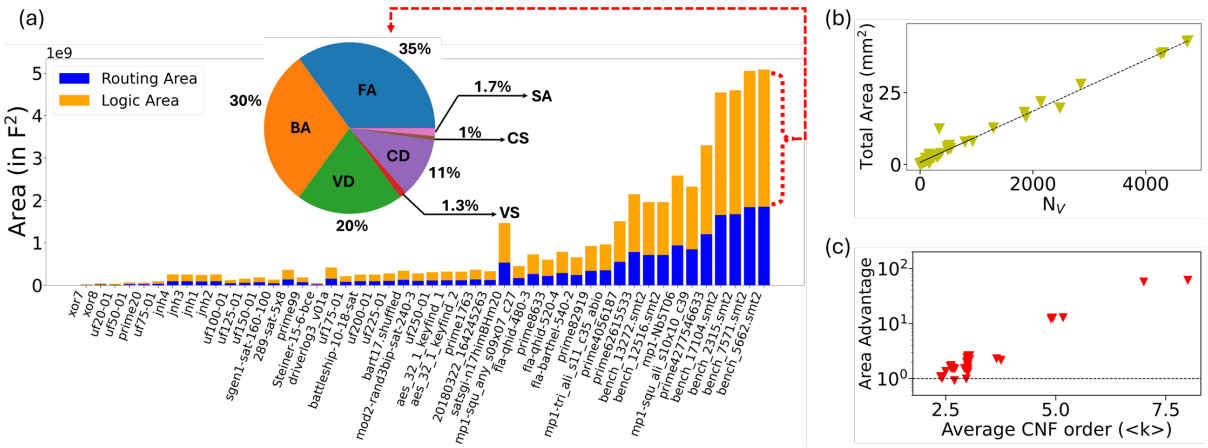


Figure 6.8: (a) Total area breakdown between routing fabric and in-memory computing tiles for the considered problems. Inset shows the area breakdown between the different mixed-signal blocks of an IMC tile for the largest SAT problem considered. (b) Total HO-FPIA chip area (in mm^2) after assuming 100% additional overhead in 65 nm CMOS technology for different problem sizes. (c) Area advantage of proposed HO-FPIA architecture with respect to quadratic FPIA [130] vs the average order of the SAT problem.

fan-in of the variables in the targeted benchmark problems. Fig. 6.7b shows the total area advantage ($A_{\text{base}}/A_{\text{HO-FPIA}}$) on a fixed architecture for the three different coupling cells with respect to N_V at $f_{\text{clk}} = 100\,\text{MHz}$. The maximum advantage obtained is $\sim 43\times$ at the problem instance with $N_V = 4734$. All three technologies have similar figures of merit with the eFlash (optimistic) case being on average $1.1\times$ more than the other two. However, there is room for further improvement. This is because for the considered problems, detailed critical path routing delays generated by VPR were observed to always be less than 1 ns. Since there are no ADCs, HO-FPIA can potentially operate at frequencies greater than 100 MHz, while the baseline requires more ADCs per tile (higher K) to compute all the N_{base} dot-products at lower cycle times. Fig. 6.7c shows the total area advantage at different f_{clk} and K with SRAM coupling cells. The advantage of HO-FPIA increases with increasing frequency, up to $116\times$ at $f_{\text{clk}} = 280\,\text{MHz}$.

Fig. 6.8a shows the area breakdown of HO-FPIA between IMC tile and routing fabric

with increasing problem size. Both routing and IMC area increase linearly with N_V, with the latter occupying approximately 62% of the total area on average across the considered problems. Fig. 6.8b shows the total HO-FPIA chip area (in mm^2) after assuming 100% additional overhead (for control logic, power delivery) in 65 nm technology with problem size. The pie chart in the inset shows the area breakdown between the different mixed-signal components inside an IMC tile. The crossbar array comprises the majority of the tile area ($\sim 65\%$) with the sensing and switch arrays comprising the least ($\sim 4\%$). We also compare the area advantage of the proposed HO-FPIA to a strictly second-order FPIA [130] that implements a second-order Ising machine to solve the QUBO version of the same high-order problem in Fig. 6.8c. Since the number of additional variables required in the QUBO of a k-SAT problem scale approximately as $\sum_{i=1}^{N_C} k_i$, where k_i is the order of clause i, the average order of the problem $\langle k \rangle = \sum_{i=1}^{N_C} k_i / N_C$ becomes a more important parameter. Fig. 6.8c shows that the area advantage of HO-FPIA increases with increasing average order of the problem, up to $\sim 43\times$ at $\langle k \rangle = 8$.

6.5 Summary and Outlook

Although HO-FPIA with fixed architectural parameters is more realistic, this leads to a sub-optimal area advantage for smaller problem sizes (Fig. 6.7b-c). This is mainly because RCW and the crossbar dimensions were chosen such that the problems with the largest fan-in and routing requirement can be implemented. This leads to underutilized crossbar ($\sim 75\%$) and routing area ($\sim 80\%$) when smaller (say 20 variable) problems are implemented. On the other hand, the same problem can fit in a single tile in the baseline architecture.

While both routing and crossbar areas increase with N_V, they also scale nearly linearly with the maximum fan-in of the problem. This means moderate-sized problems with large

	AIMS	FPIA	HO-FPIA
Ising Dynamics	CTDS	DTDS (extendable to CTDS/CTCS)*	DTDS (extendable to CTDS/CTCS)*
Ising Type	HOIM	SOIM	HOIM
Problem type	3-SAT#	QUBO	**k-SAT/PUBO**&
Area estimate	169 mm² at N_V=512, k=3	2.36 mm² at N_V=492, <k>=2.9 19.3 mm² at N_V=100, <k>=5.16	**0.98 mm²** at N_V=492, <k>=2.9 **1.5 mm²** at N_V=100, <k>=5.16
Navigational Efficiency	High	Low	**High**++

Figure 6.9: Comparison with other approaches. * Extendable to CTCS/CTDS with change in VS/CS circuitry and/or routing switches. #Architecture proposed in [78] is shown only for SAT problems with k=3. &SAT problems with arbitrary k can be mapped. Any PUBO can be also mapped by replacing clauses with monomials. ++It has been shown that SOIMs are up to 2 orders or magnitude slower than HOIMs for k-SAT problems [37]. Other approaches include AIMS [78] and FPIA [130].

maximum fan-in could have large area overheads. However, since the maximum fan-in is often much larger than the average fan-in, variable splitting techniques [130, 93] could be applied to reduce the maximum crossbar dimensions and channel widths required in the fixed architecture. Another way would be to employ heterogenous IMC tiles with two different crossbar dimensions, one that is sparsely available and with large linear dimensions for large fan-in variables, while the other is a more frequent one with smaller dimensions. A similar heterogeneity could be applied to routing channel widths.

The areas of some of the peripheral components (especially $A_{\mathrm{VS}}/A_{\mathrm{CS}}$) in the IMC tile could potentially deviate from our estimates depending on the HOIM dynamics implemented (whether it is continuous-time or discrete-time) and sensing mode (voltage or current mode). However, we believe that does not change the implication of our results as HO-FPIA area is dominated by crossbar arrays and routing. Indeed, for all consid-

ered problems, $A_{\mathrm{VS}} + A_{\mathrm{CS}}$ could be increased by 10× without more than 10% increase in the total area. We explored the proposed architecture for Discrete-Time-Discrete-State (DTDS) HOIM dynamics, but it is generalizable to Continuous-Time-Discrete-State (CTDS) with changes to the VS and CS circuits like in [78] and to Continuous-Time-Continuous-State (CTCS) by modifying buffered switches in the routing fabric to pass-gates in addition to sensing periphery changes.

The modeled footprint of our architecture is compared with AIMS [78] that can implement CTDS HOIM on large 3-SAT problems. The total area for 512-variable SAT problems is estimated to be $\sim 5.27\,\mathrm{mm}^2$ from the linear fitting (Fig. 6.8b) which is 32× more compact than that in [78] at the same technology node, even after adding 100% overhead to crudely account for possible swelling of the design when implementing all its features. The comparison for specific problems (prime8633 and jnh1) with locally optimized architectures shows an even larger advantage (table in Fig. 6.9). Table in Fig. 6.9 also shows more favorable density compared to quadratic FPIA though the latter is expected to be more area efficient for natively QUBO formulated problems that are not benchmarked in this work. On extrapolating the linear trend in total area with N_V, the proposed architecture can potentially implement up to 11 K variable problems in a $1\,\mathrm{cm}^2$ chip.

The operating frequency of the considered baseline architecture is primarily bottlenecked by the sampling speed of the ADCs and the number of such ADCs used. While such architectures can operate at iso-frequency with HO-FPIA, it does so by requiring multiple ADCs to process all the bit-line currents and therefore increased hardware cost. Fig. 6.7c shows how with increasing operating frequency, HO-FPIA area remains constant, but the baseline architecture area increases because of requiring more ADCs per tile. The increased number of ADCs also results in higher power consumption. For example, at $f_{\mathrm{clk}} = 100\,\mathrm{MHz}$, the power consumption of a HO-FPIA IMC tile comprising of

a 256×128 SRAM crossbar array, driving and sensing circuitry, multiplexers and digital control block is roughly 6 mW [130], whereas the total power consumption of just $K = 22$ ADCs in the baseline tile is 66 mW [134]. Note that this estimate for the baseline tile does not include the power consumption of other peripheral circuits and the crossbar array itself. As a result, at iso-operating frequency and assuming similar algorithmic latency of the Ising Machines implemented in the two architectures, HO-FPIA is at least one order more energy efficient than the baseline.

FPGA-inspired architectures like that in [130] have similar latency and power figures as HO-FPIA due to similar peripheral circuits and not requiring ADCs. However, they are restricted to implementing SOIMs that can solve only QUBOs. It has been shown that Ising Machines that can solve PUBO have orders of magnitude faster time-to-solution and lower energy-to-solution [37, 35] than those that solve QUBO of the same optimization problem. Therefore, HO-FPIA can be several orders faster and energy efficient than the architecture in [130] on natively high-order problems, just by virtue of superior navigational efficiency of HOIMs compared to SOIMs.

Finally, it has been shown in several studies that multiple parallelly running replicas/threads of heuristic solvers of optimization problems can converge to the optimal/quasi-optimal solution faster than if single replicas are run sequentially [43, 141] (something we also showed in Chapter 5). This is because the performance of such heuristic solvers depends partly on the randomly assigned initial state and therefore a replica that is initialized with a state closer to the global minima might converge faster than other replicas. As a result, the compact footprint of HO-FPIA can allow launching multiple physical threads of the HOIM, thereby leading to additional latency improvements at iso-area with respect to the other architectures.

Chapter 7

Outlook for High-Order Ising Machines

7.1 Summary of Contributions

This dissertation presents architectural techniques, concepts and algorithms, experimentally validated through fabricated integrated circuits, that we believe would comprise some of the key ingredients to building future scalable high-order Ising machines.

First, we introduce a hardware paradigm that enables massively parallel computation of high-order gradients (a core operation in high-order combinatorial optimization solvers incorporating non-complete techniques) using mixed-signal in-memory computing circuits (see Chapter 2). It is shown for the first time, that a pair of cross-coupled crossbars with appropriate peripheral circuits can provide fast solutions to high-order optimization problems with hardware complexity that scale linearly with the number of variables and terms in the objective function but independent of the order, thereby alleviating the trade-offs between parallelism and generalizability incurred in traditional solvers. The functionality of this approach is validated by experimentally solving a small-scale third-order

Boolean satisfiability (3-SAT) problem (an important class of high-order combinatorial optimization problems) on a general-purpose hardware prototype comprising integrated metal-oxide memristor crossbar circuits, with competitive heuristics algorithm. It is also shown that with appropriate crosspoint device selection and increased peripheral circuit overhead, such a paired crossbar approach could also enable continuous-valued gradient computing in continuous-state-continuous-time dynamical solvers.

The experimental demonstration with memristor crossbar had significant communication overhead between the PC and the general-purpose integrated circuit (that was not originally designed for the specific application), as well as substantial programming overhead of the problem to the crossbar due to iterative analog tuning of memristor states. Therefore, in order to demonstrate the true potential of the introduced architectural concepts, a fully integrated high-order optimization solver chip (tuned to solving arbitrary order K-SAT problems) is taped out in 55 nm CMOS technology node (see Chapter 3). The solver can map arbitrary-order SAT problems and leverages the introduced in-memory computation techniques to successfully demonstrate solution to hard K-SAT instances with up to 50 variables. The integrated circuit features a custom 10-transistor-SRAM based in-memory computing array. It performs high-degree gradient computation using a single physical array instead of two, by adding extra transistors in the standard 6T SRAM cell, thereby enabling decoupled orthogonal access to the array. Additionally, it uses analog-input Winner-Take-All circuits to act directly on the analog crossbar outputs and update the most promising variables. It achieves one to two orders of magnitude speedup over prior solvers despite implementing a legacy SAT-solving algorithm that updates only one variable per iteration.

While this demonstrated the potential of the paired-crossbar approach, the need for an analog reduction circuit (the winner-take-all block) to implement greedy moves, the restriction to at most one variable update per iteration, and the additional steps required to

identify variables within a single unsatisfied clause all posed challenges for scaling such a solver. Therefore, through careful hardware-software co-design, a new probabilistic SAT-solving algorithm is presented that leverages the bipartite clause-variable relationship to update multiple variables per iteration with high navigational efficiency. It outperforms a wide range of prior algorithms while the usage of probabilistic bits foregoes the need for reduction circuits and enables scalability. This is validated by a second mixed-signal chip also taped out in 55 nm CMOS (see Chapter 4). It uses an 11-transistor SRAM based crossbar array that not only improves upon the parallelization capabilities of the previous chip but also enables a multi-threaded pipeline, thereby further improving convergence times. Additionally, it uses dynamic comparator based Probabilistic-bit (p-bit) circuits, enabling massively parallel sampling and eliminating the need for ADCs or analog-domain reduction circuits. The chip exhibits the fastest-to-date solution times on 50 variable 3-SAT problems and outperforms prior ASICs (including those in more advanced 28 nm) by 4-13 times.

Furthermore, architectural concepts are introduced that classify existing high-order gradient computing circuits into two categories and introduces a new hybrid-style circuit implementation that achieves at par or reduced hardware costs compared to either category, across several high-order benchmark problems (see Chapter 5). Circuit techniques to handle XOR constraints in such paired crossbar architectures are presented. Using efficient optimization heuristics, it is shown that area efficient architectures can enable an even faster solution time at iso-energy, by enabling a proportional number of parallel replicas to be run in the iso-area scenario. Such an iso-area comparison of high-order Ising machines with second-order Ising machines reveals additional scaling advantages that increase with the problem's order. Through modeling effort it is shown that a Field Programmable Gate Array inspired architecture comprising tiles of paired crossbars, present as islands in a sea of routing fabric, can be adopted to solve larger

combinatorial optimization problems that cannot be mapped to a single pair of crossbar arrays (see Chapter 6).

7.2 Preliminary results from ongoing work

There are several ongoing work related to high-order combinatorial optimization problems that was not covered in the preceding chapters. One of them is an integrated circuit consisting of a 102×260 array of ESF-3 Floating-gate half cells [90]. Figure. 7.1a shows the die micrograph and a top-level diagram of the circuits is shown in Fig. 7.1b. A snippet of 8 adjacent half-cells (or 4 super-cells) is shown in Fig. 7.1c. The floating gate cells are arranged in a NOR-type architecture, where each cell has its drain connected to the bit line, thereby allowing random read access. The gate consists of two layers; a control gate and a floating gate. The floating gate can store varying amounts of charge that can shift the threshold voltage of the transistor proportionately (more charge store means higher threshold voltage). The amount of charge stored can be increased through a program operation and decreased using a erase operation. Programming in the current prototype is done through Channel Hot Electron injection (CHEI) (similar to [90]), whereas erase operation is done through Fowler–Nordheim tunneling between the floating gate and erase gate. In order to make it possible to erase each cell individually erase lines are made to run in parallel to the bit lines (similar to [90]). Due to their low footprint and ability to store analog weights through threshold-voltage modulation, floating-gate cells offer a more compact alternative to SRAM-based implementations of crossbar arrays, particularly for weighted MAXSAT and other PUBO formulations that require multi-bit weight representations. Unfortunately detailed measurement and performance benchmarking of this integrated circuit remains an ongoing project at the time of writing of this dissertation.

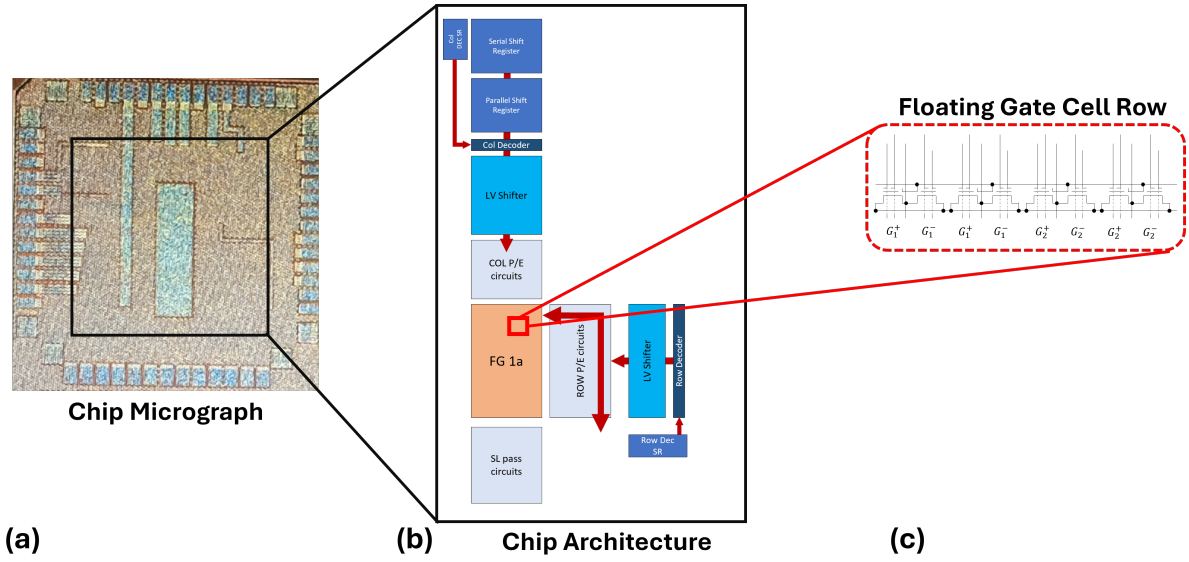


Figure 7.1: (a) Die micrograph of the Floating-gate array based analog In-Memory Computing macro chip, (b) top-level block diagram of the key components of the circuit and (c) arrangement of the floating-gate cells in a single column.

Another line of work explores digital in-/near-memory computing architectures that employ complex coupling cells to embed and solve high-order optimization problems with limited membership (i.e., the number of clauses in which a variable appears) and limited order (i.e., the number of variables in a clause) [74]. For problems that are provably or predictably sparse, such architectures can offer advantages over mixed-signal crossbar-based IMC solutions, including faster cycle times, higher hardware utilization, improved robustness to noise and variability, and shorter design and verification cycles. Figure 7.2a illustrates an example coupling cell implementation, while Fig. 7.2b shows the corresponding system-level architecture consisting of an array of these coupling cells along with the peripheral circuits responsible for evaluating the objective function, selecting the variable to update, and performing other required operations. The coupling cell array (shown in Fig. 7.2b) has TC rows and TR columns, where each row corresponds to a unique variable and each cell in that row corresponds to a derivative-product involved in

the pseudo partial derivative expression of that variable (see Equation 5.1 and Fig. 5.3 for reference). Each coupling cell computes a derivative-product using the K′-input AND gate (component #3 in Fig. 7.2a) and forwards the result to the multiply-accumulate (MAC) circuit (component #4 in Fig. 7.2a). The MAC circuit is responsible for multiplying the local unit derivative product with its corresponding weight (as specified by the problem's polynomial objective function), adding the result with partial sums contributed by upstream coupling cells and forwarding the new partial sum to downstream coupling cells. This way the partial-derivatives of all variables are computed in a massively parallel fashion in each row. Each coupling cell, stores a local (redundant) copy of one of its derivative-product's member variables and sources the other variables from neighboring cells. Due to this fixed and limited neighborhood connectivity, this type of architecture may not be able to embed moderately sparse-to-dense problems (and those with order greater than 5). However, the reduced capacitive loading along the signal paths responsible for the MAC operations used in partial-derivative computation, together with the smaller overall number of coupling cells, results in lower latency and improved hardware utilization. For additional details, readers are referred to a related manuscript that is under review at the time of writing and may appear in the literature as [74] in the future.

The final line of work explores opportunities of accelerating complete SAT solving algorithms with in-memory computing approaches. Unlike Ising machines, complete SAT solvers are guaranteed to determine whether the CNF is satisfiable (SAT) or unsatisfiable (UNSAT). Modern complete solvers are almost universally based on the DPLL (Davis–Putnam–Logemann–Loveland) search framework, exhaustively exploring the search space in a structured manner and ensuring correctness of the result [142]. Boolean Constraint Propagation (BCP), or unit propagation, is the inference mechanism through which modern complete SAT solvers deduce forced variable assignments whenever a clause becomes unit under the current partial assignment [143, 144]. BCP

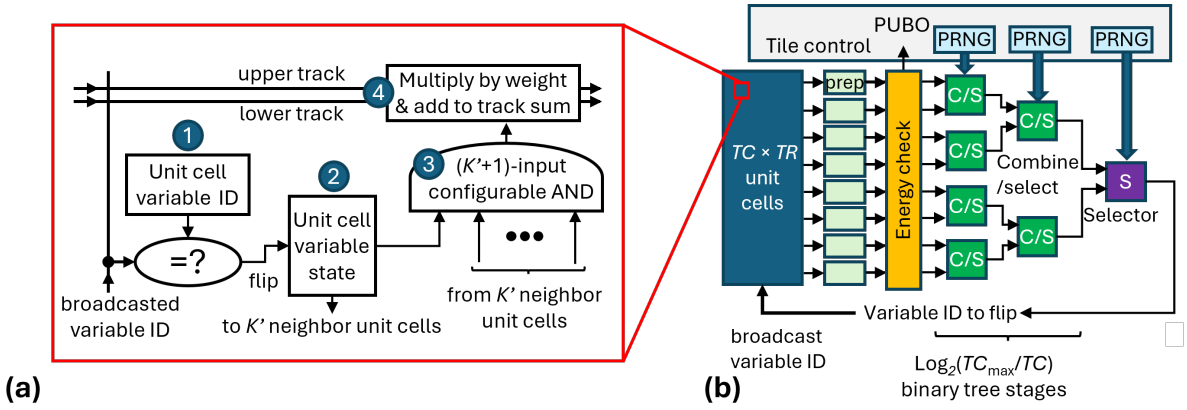


Figure 7.2: Top-level details of a compressed high-order Ising machine architecture using digital near-memory computing. (a) The coupling cell used to compute a derivative-product and (b) the coupling cell array and its associated peripheral circuit for mapping and solving K-SAT problems.

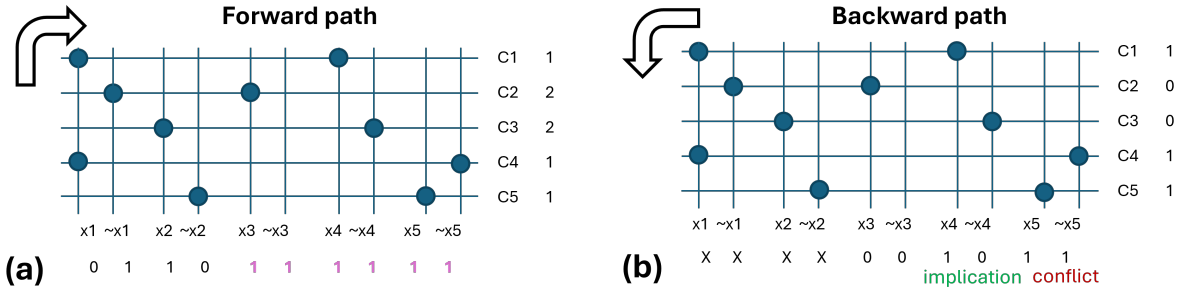


Figure 7.3: Key steps to accelerate Boolean constraint propagation in-memory. Forward path to find unit and/or conflict clauses and backward path to find implied literals and/or conflict literals.

constitutes the core computational kernel of such solvers, often dominating overall runtime. After each decision assignment, BCP recursively enforces all implied assignments, detects conflicts, backtracking, among other things. Figure. 7.3 shows the two key steps, a forward and a backward pass, using which BCP can be accelerated using in-memory computing crossbars. The K-SAT problem can be mapped on to the crossbar with each column representing a literal, each row representing a clause and the corresponding cross-point devices being set to their ON state whenever a particular literal is a member of a

clause (similar to Fig. 2.2). During a forward pass, the current literal values are applied to the column lines, with both literals of an unassigned variable being set to one (for the purposes of the forward pass). The objective of this step is to find clauses with one unassigned literal (a unit clause) or satisfied clauses with one true-valued literal. Such clauses have a value of one in the forward pass result. In the backward pass, only these clauses are activated. On the output side, the result in column lines corresponding to only unassigned variables is considered. Variables with one of its literals receiving a non-zero value and the other receiving a zero-value are considered to be implied at the current iteration. The values of these variables are then set accordingly (i.e., literal receiving the non-zero current is set to one, while its complementary literal is set to zero). If a variable receives non-zero values in the column lines of both of its literals, it means there exists unit clauses containing both of these literals. Such a condition signals a conflict, prompting the solver to either backtrack to a previous decision level or remain at the current level but flip the assignment of the most recent decision variable. Note that a conflict can also be detected if any clause receives a zero-value in the forward pass result. That implies the clause has no unassigned literals and all of its literals are set to False. Furthermore, note that although clauses with one True literal are also activated in the backward pass they don't effect literal implication or conflict detection. This is because they contribute currents to their member literals, all of whom are already assigned and only the backward pass outputs corresponding to unassigned literals are taken into consideration.

7.3 Future Directions

This dissertation presents experimentally validated concepts and techniques that could serve as key building blocks for future scalable high-order Ising machines. Nev-

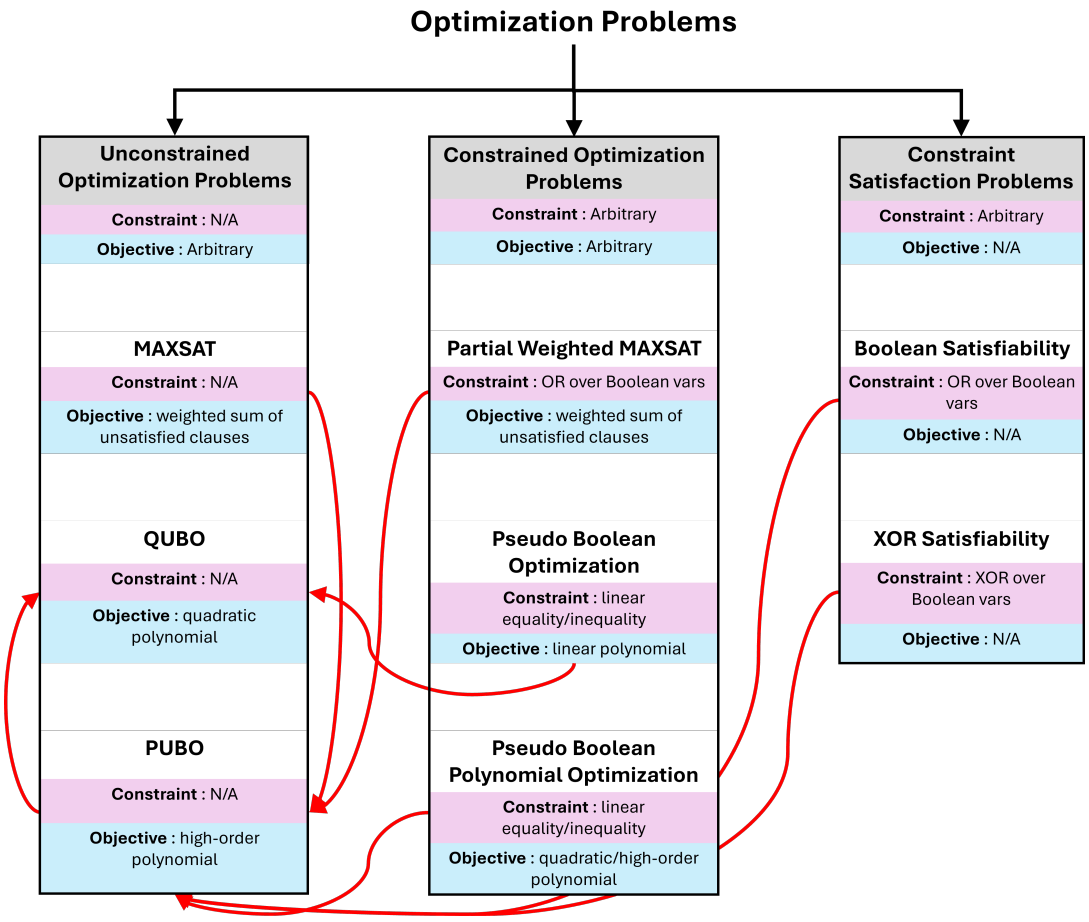


Figure 7.4: Categorization of optimization problems based on presence/absence of constraints and objective function. Unconstrained optimization problems like MAXSAT, QUBO and PUBO have an objective function that needs to be minimized but no hard constraints that must be satisfied. Constraint satisfaction problems only have hard constraints but no objective function as such. An objective function can be formalized that tracks the number of unsatisfied constraints but at any point in time any constraint is either satisfied or not and the global minima of the function is known and can be verified (it is the assignment of variables that satisfies all constraints). Constrained optimization problems on the other hand have both constraints and objective function. Pseudo boolean optimization (PBOs) has linear equality or inequality constraints and generally a linear polynomial (order equal to one) objective function. There are generalizations of PBOs with quadratic or higher-order objective functions that are known as Pseudo Boolean Polynomial Optimizations (PBPOs). Red arrows indicate existing relaxation techniques of problems in one form to another.

ertheless, several additional milestones must still be achieved and questions answered, before such systems can be fully realized.

A natural next step is to explore other optimization problems that share structural similarities with Boolean Satisfiability and to assess their compatibility with the proposed architectures, either directly or with adaptations. Broadly, combinatorial optimization problems can be classified into three categories based on the presence or absence of constraints and an objective function: Unconstrained optimization, constrained optimization and constraint satisfaction problems (see Fig. 7.4). Problems such as Partial Weighted MAXSAT (PMS) [145], Pseudo-Boolean Optimization (PBO), and Pseudo-Boolean Polynomial Optimization (PBPO) fall under the constrained optimization category; they involve both an objective function to be minimized or maximized and a set of hard constraints that must be satisfied. In contrast, constraint satisfaction problems such as Boolean Satisfiability, require only that all hard constraints be satisfied, with no explicit objective function. On the other hand, problems such as weighted MAXSAT or those formulated in QUBO/PUBO form belong to the unconstrained class, where the task is solely to optimize an objective function without any additional constraints. In this dissertation, we demonstrated how certain constraint satisfaction problems like K-SAT/XORSAT as well as unconstrained problems like those in PUBO or MAXSAT (see Fig. 2.2) could be implemented efficiently using in-memory computing hardware. However, it remains an open question how constrained optimization problems might be solved directly using similar architectures. Although such problems can be relaxed into unconstrained formulations by embedding constraints within the objective function using appropriate gadgets [22] (e.g., relaxing a PMS instance into a MAXSAT instance by assigning very large weights to constraint clauses), there is growing evidence that it may be more efficient (either in terms of solution navigation or hardware overhead) to retain explicit knowledge of the original constraints and to continuously track whether they are satisfied by the current assignment. Our hardware-software co-design revealed that an Ising-solver that tracked the status of all constraints and updated variables only within

satisfied constraints outperformed those that lack knowledge of the underlying constraints and operated on the unconstrained relaxation of the problem (see Chapter 4). Another work showed that it may be possible to retain such constraint specific knowledge in unconstrained formulations but at the cost of significant hardware costs [78]. This makes development of universal constrained optimization problem solvers by building upon the architectural and algorithmic developments introduced in this dissertation and other works [146, 95], an interesting future research topic.

While the standard Ising model has clear physical grounding as a model of ferromagnetism, it is unclear if constrained Ising models (especially those retaining the hard constraints in their original form without including them as monomials with large coefficients in the objective function) correspond directly to any natural physical material. Nonetheless, this lack of a direct physical analogue should not deter the design of Ising machines capable of efficiently handling such formulations. A useful parallel can be drawn from the evolution of neuromorphic engineering. Early neuromorphic systems placed strong emphasis on detailed biological fidelity [147], but it has become widely recognized that exact biophysical replication offers limited practical utility for building scalable and robust hardware. Instead, successful neuromorphic architectures increasingly combine bio-inspired principles with methods from linear algebra, optimization, and control theory [148, 149, 150, 151]. Hardware–software co-design, as demonstrated in Chapter 4, was enabled by a detailed understanding of both the capabilities of the proposed hardware architecture and the core algorithmic principles underlying state-of-the-art SAT solvers. In line with the argument of [152], that the hardware dictates the software in massively parallel unconventional computing platforms, progress in this area will require close collaboration between hardware architects, computer scientists, and experts in discrete mathematics and complexity theory. Such interdisciplinary efforts are essential for addressing interesting questions like: What forms of massively paral-

lel algorithms can efficiently solve constrained optimization problems without relying on analog reduction circuits? Should gradients associated with soft and hard constraints be treated differently [153], and if so, how? And to what extent can the core operations in optimization algorithms be reformulated in terms of matrix–vector operations so that they can benefit from the acceleration provided by in-memory computing hardware? A growing line of research explores the use of machine learning approaches to enhance the performance of combinatorial optimization algorithms [154], including local-search methods, complete SAT solvers [155], and metaheuristic techniques [156, 157]. Notably, recent works have demonstrated that large language models can be leveraged to automatically discover new heuristics that improve the efficacy of state-of-the-art complete SAT solvers [158, 159].

Another valuable step is a systematic cataloging of practically relevant combinatorial optimization problems (high-order or quadratic), along with the topological characteristics (like sparsity, average and maximum order, average and maximum degree of connectivity etc) of the underlying graph representing their objective function and constraints. This would be essential in guiding development of more efficient multi-tile architectures. For instance, the limited degrees, and clause orders, present in many high-order optimization problems (particularly those derived from Boolean Satisfiability) made it possible to partition large instances across multiple crossbars, where all crossbars are interconnected in a one-to-one manner through an FPGA-inspired routing fabric (see Chapter 6). It remains to be seen if similar architectures could be effectively extended to other problems. To improve hardware utilization, overall capacity, and applicability of Ising-based accelerators to a broader range of problem structures, heterogeneous architectures incorporating both sparse digital in-/near-memory tiles (as described in Chapter 7.2) and dense analog in-memory tiles (Chapter 6) represent a promising direction [160, 161]. While such tiling strategies may enable solving problems with hundreds of thousands of

variables on a single chip, scaling to exascale instances will ultimately require exploring multi-chip interconnect technologies [162], like 3D integration [163], high-bandwidth photonic links [164], and/or on-board address-event routing [165]. A systematic investigation into the overheads associated with large-scale Ising systems like problem loading, synchronization, energy tracking and solution readout is also necessary to ensure that the anticipated scaling benefits of such architectures are preserved.

Appendix A

Supplementary Notes for Chapter 2

A.1 Partial derivatives and difference quotients

The gradient of a multi-variate continuous, differentiable function $H : \mathbb{R}^N \rightarrow \mathbb{R}$ is given by equation A.1 where H_{x_i} is the partial derivative of H with respect to variable x_i and is evaluated as shown in equation A.2.

$$\nabla H = \left[\frac{\partial H}{\partial x_1}, \frac{\partial H}{\partial x_2}, \ldots, \frac{\partial H}{\partial x_N}\right]^T = \left[H_{x_1}, H_{x_2}, \ldots, H_{x_N}\right] \tag{A.1}$$

$$H_{x_1} = \lim_{h \rightarrow 0} \frac{H(x_1 + h, x_2, \ldots, x_N) - H(x_1, x_2, \ldots, x_N)}{h} \tag{A.2}$$

The partial derivative H_{x_i} tells the sign and the rate at which the function H will change when variable x_i is incremented or decremented by a certain amount. This guides gradient descent/ascent-based algorithms to iteratively update the respective variables with correct update direction/magnitude to decrease/increase the value of H. Consider a degree one bi-variate polynomial function $H = 2x_1 - 3x_2$, where $H_{x_1} = 2$ and $H_{x_2} = -3$. This implies that when variable x_1 is increased the function H increases with a rate of 2

whereas when variable x_2 is increased the function decreases with a rate of 3.

On the other hand, for discrete-valued functions like $H : \{a, a+h, \ldots, a+Kh\}^N \rightarrow \mathbb{R}$ (where $\{a, a+h, \ldots, a+Kh\}$ is the discrete domain of the function with a, K and h being integers), the usual definitions of gradients and partial derivatives are not useful. For such functions, the analogue to derivative is the partial difference quotient (see [81]) which is given by equation A.3. Note how at the limit $h \rightarrow 0$, the difference quotient becomes the partial derivative H_{x_i} in continuous space.

$$H_{x_1} = \frac{\Delta H}{\Delta x_1} = \frac{H(x_1 + h, x_2, \ldots, x_N) - H(x_1, x_2, \ldots, x_N)}{h} \tag{A.3}$$

Now for a binary valued function $H : \{0, 1\}^N \rightarrow \mathbb{R}$, the domain of each variable contains 0 and 1. So when $x_i = 0$, its only valid move is to change to 1 and vice versa. The difference quotients of H for $x_1 = 0$ and $x_1 = 1$ are shown in equation A.4 and A.5 (non-participating variables have been omitted for simplicity). Note that they are equal.

$$H_{x_1}\big|_{x_1=0} = \frac{\Delta H}{\Delta x_1}\big|_{x_1=0} = \frac{H(x_1 = 1) - H(x_1 = 0)}{1 - 0} = H(x_1 = 1) - H(x_1 = 0) \tag{A.4}$$

$$H_{x_1}\big|_{x_1=1} = \frac{\Delta H}{\Delta x_i}\big|_{x_1=1} = \frac{H(x_1 = 0) - H(x_1 = 1)}{0 - 1} = H(x_1 = 1) - H(x_1 = 0) \tag{A.5}$$

In the context of gradient descent inspired local search algorithms for solving combinatorial optimization problems that have such objective functions, the difference quotient provides information about the magnitude but not the sign/direction in which the function H will change if a variable undergoes a valid move. What is more important for such algorithms is the net change in function H when a variable x_i changes from 0 to 1

or from 1 to 0, where both the magnitude and the sign of the change are used to decide whether to accept that move or not. This is simply obtained by taking the product of the difference quotient with the difference in the final and initial variable values and is shown in equation A.6.

$$\Delta H_{x_i} = H_{x_i} \Delta x_i \tag{A.6}$$

Throughout the remainder of the supplementary and main text, we shall refer to ΔH_{x_i} in the context of binary valued functions H, as the "pseudo derivative of H w.r.t. x_i" or simply the "pseudo partial derivative". The pseudo derivative H_{x_i} is the net change in the function H when binary variable x_i is flipped, i.e., when it changes its value to $\bar{x}_i$. The vector of pseudo derivatives of a multivariate function H with respect to all its variables is referred to as the "pseudo gradient".

A.2 Gradient computation in high-order polynomials

Any K-order discrete multi-variate polynomial in variables $\{x_1, x_2, \ldots, x_N\} \in X$, can be expressed as a summation of monomials (terms) – see equation A.7, where t_j is the jth monomial and $t_j \in T$, the set of all monomials of H. Each of these monomials is a product of at least one variable (H might also include a constant offset term, but it is never relevant to computing derivatives, and is removed from T) and a real factor a_j. Further, let E be the set of all pairs $\{x_i, t_j\}$ such that degree of x_i in monomial t_j is non-zero (see equation A.7).

$$H = \sum_{j=1}^{M} t_j, \qquad t_j = a_j \prod_{(i \forall \{x_i, t_j\} \in E)}^{N} x_i \tag{A.7}$$

For such a polynomial, the pseudo derivative with respect to variable x_f (net change in the polynomial on flipping variable x_f, i.e., changing value from "pre" state 0 to "post" state 1 or vice versa) is given by equation A.8. The variable x_f is factored out and the rest is a sum of partial monomials. The variable x_f is factored out and the rest is a sum of partial monomials, with degree one less (in this case, removing x_f) than their respective original monomials.

$$\Delta H_{x_f} = \sum_{(j \forall \{x_f, t_j\} \in E)} \Delta t_j = \sum_{(j \forall \{x_f, t_j\} \in E)} (t_j^{post} - t_j^{pre}) = \Delta x_f \sum_{(j \forall \{x_f, t_j\} \in E)} a_j \prod_{\substack{(i \forall \{x_i, t_j\} \in E) \\ i \neq f}}^{N} x_i \tag{A.8}$$

Since the variables are binary, two cases arise — Case I: $x_f^{pre} = 1$ ($x_f^{post} = 0$), where initial value of x_f is 1 and Case II: $x_f^{pre} = 0$ ($x_f^{post} = 1$), where initial value of x_f is 0.

Case I ($x_f^{pre} = 1$):

The pseudo partial derivative with respect to the now-zero-valued variable is computed as:

$$\Delta H_{x_f} = \sum_{(j \forall \{x_f, t_j\} \in E)} (t_j^{post} - t_j^{pre}) = \sum_{(j \forall \{x_f, t_j\} \in E)} (-t_j^{pre}) = - \sum_{(j \forall \{x_f, t_j\} \in E)} a_j \prod_{(i \forall \{x_i, t_j\} \in E)}^{N} x_i \tag{A.9}$$

We define ancillary variable $z_j = \prod_{(i \forall \{x_i, t_j\} \in E)} x_i$ (local to monomial t_j and consistent with the z introduced in Fig. 2.3c of main text) and equal to the product of the variables in that term. Observing that the product of several binary variables is one if and only if each multiplicand is one, and that the order of the term is equal to the number of

multiplicands, the variables z_j can also be computed as in equation A.10, where K_j is the order of the term. This replaces a product with a linear transformation, which allows for its massively parallel in-memory computation. We refer to this variable as the "break monomial indicator" as it indicates if the monomial's value would change from non-zero to zero on flipping of any of its member variables.

$$z_j = \begin{cases} 1, & \text{if } \sum_{(i \forall \{x_i, t_j\} \in E)} x_i = K_j \\ 0, & \text{otherwise} \end{cases} \tag{A.10}$$

We define B as the $M \times N$ adjacency matrix of the graph G, where $B = (\hat{b}_1, \hat{b}_2, \ldots, \hat{b}_N)$ such that $\hat{b}_i = (b_{1i}, b_{2i}, \ldots, b_{Mi})^T$ and $b_{ji} = 1$ if $\{x_i, t_j\} \in E$ and 0 otherwise. A vector-matrix multiplication between this adjacency matrix and the variables can then be used to compute the sum of variables in each term (see equation A.11 below), where $\hat{\sum}_j^{mono} = (\sum_1^{mono}, \sum_2^{mono}, \ldots, \sum_M^{mono})$ is the vector representing the sum of variables in each monomial (referred to as the monomial variable sum in Fig. 2.3a) and computed in the forward pass (see Fig. 2.3a of main text) and $\hat{x} = (x_1, x_2, \ldots, x_N)$ is the variable vector. The expression for z_j can then be re-written as shown in equation A.12.

$$\hat{\sum}_j^{mono} = \hat{x} B^T \tag{A.11}$$

$$z_j = \begin{cases} 1, & \text{if } \sum_j^{mono} = K_j \\ 0, & \text{otherwise} \end{cases} \tag{A.12}$$

Substituting the break-monomial indicator in the pseudo derivative expression (of equation A.9) and considering the monomial factors, one observes equation A.13 where $\hat{\sum}_f^{break}$ is referred to as the "break variable sum" with respect to variable x_f. It is the

number of monomials weighted by their respective factors that will become zero from non-zero when a one-valued variable x_f is flipped to zero.

$$\Delta H_{x_f} = -\hat{\sum}_f^{break} = -\sum_{j=1}^{M} a_j z_j b_{jf} \tag{A.13}$$

Case II ($x_f^{pre} = 0$):

The pseudo partial derivative with respect to the previously-zero-valued variable is computed as shown in equation A.14.

$$\Delta H_{x_f} = \sum_{(j \forall \{x_f, t_j\} \in E)} (t_j^{post} - t_j^{pre}) = \sum_{(j \forall \{x_f, t_j\} \in E)} t_j^{post} = x_f^{post} \sum_{(j \forall \{x_f, t_j\} \in E)} a_j \prod_{\substack{(i \forall \{x_i, t_j\} \in E) \\ i \neq f}}^{N} x_i \tag{A.14}$$

Similarly, when $x_f^{pre} = 0$ ($x_f^{post} = 1$), the monomial values pre-variable-flipping (t_j^{pre}) are equal to zero. Since $x_f^{post} = 1$, it can be factored out and set to 1. Note that since $x_f^{pre} = 0$, the product in the right-most side of equation A.14 is equal to one only when all but one variable (x_f) in the monomial t_j have a value of 1. To evaluate this condition using linear transformation, we define a second ancillary variable s_j (local to term t_j) (see equation A.15) which is 1 when the sum of variables in the monomial is exactly equal to one less than the order of that term (K_j) and 0 otherwise. We refer to this variable as the "make monomial indicator" (see Fig. 2.3b of main text) as it indicates if the monomial's value will change from zero to non-zero on flipping of one of its member variables. The pseudo derivative for the $x_f^{pre} = 0$ case, can then be succinctly written down as in equation A.16, where $\hat{\sum}_f^{make}$ is the make variable sum with respect to variable x_f. It is the number of monomials weighted by their respective factors that will become non-zero from zero when a zero-valued variable x_f is flipped to one.

$$s_j = \begin{cases} 1, & \text{if } \sum_j^{mono} = K_j - 1 \\ 0, & \text{otherwise} \end{cases} \tag{A.15}$$

$$\Delta H_{x_f} = \hat{\sum}_f^{make} = \sum_{(j \forall \{x_f, t_j\} \in E)} a_j \prod_{\substack{(i \forall \{x_i, t_j\} \in E) \\ i \neq f}}^{N} x_i = \sum_{j=1}^{M} a_j s_j b_{jf} \tag{A.16}$$

Since the pseudo derivative with respect to one-valued variables is given by the break-variable sum (equation A.13) and that with respect to zero-valued variables is given by the make-variable sum (equation A.16, both can be combined as shown in equation A.17 to provide a generalized expression for the pseudo derivative of all the variables, that is equal to the difference between the make and break values.

$$\hat{\Delta H} = (1 - \hat{x}) \circ \hat{\sum}^{make} - \hat{x} \circ \hat{\sum}^{break} = ((\hat{a} \circ \hat{s})B) \circ (1 - \hat{x}) - ((\hat{a} \circ \hat{z})B) \circ \hat{x} \tag{A.17}$$

A.3 Gradient computation in Conjunctive-Normal Form

Any Constraint Satisfaction Problem (CSP) in CNF can be represented by a bipartite graph $G \in \{L, C, E\}$, where one set of vertices L denote the literals $\{l_1, l_2, \ldots, l_{2N-1}, l_{2N}\}$ such that $l_{2i-1} = x_i$ (positive literal of variable x_i), $l_{2i} = \bar{x}_i$ (negated literal of variable x_i), the other set C the clauses $\{c_1, c_2, \ldots, c_M\}$ and E the set of edges defined by $\{l_i, c_j\}$ such that literal l_i is a member of clause c_j. As such, there are N variables, $2N$ literals and M clauses in the CNF. Let B denote the forward binary $M \times 2N$ adjacency matrix of G, where $B = (\hat{b}_1, \hat{b}_2, \ldots, \hat{b}_{2N})$ such that $\hat{b}_i = (b_{1i}, b_{2i}, \ldots, b_{Mi})^T$ and $b_{ji} = 1$ if $\{l_i, c_j\} \in E$ and 0 otherwise.

The objective function in such CNF-based problems is the total number of unsatisfied clauses weighed by the clause weights w_j, where a clause is satisfied if the OR of its member literals is true and unsatisfied otherwise (see equation A.18). The goal of the optimization problem is then to minimize H. Therefore, the definition of pseudo derivatives here is the net change in the sum of weights of unsatisfied clauses when a variable is flipped. A popular score metric known as the gain is used in optimization algorithms designed for solving CSPs that are present in CNF. It is the net increase in the number of satisfied clauses weighed by their respective clause weights and can be expressed as the negative of the CNF pseudo derivative (see equation A.19).

$$H = \sum_{j=1}^{M} w_j U_j, \quad U_j = \begin{cases} 0, & \text{if } c_j \text{ is satisfied} \\ 1, & \text{if } c_j \text{ is unsatisfied} \end{cases} \tag{A.18}$$

In a CNF, when a literal is flipped not only do the clauses that contain that literal change but also the ones that contain its complement. Therefore, one needs to track the change caused by both literals associated with a given variable. The CNF gain when a variable x_f is flipped can then be expressed as the sum of gains with respect to its two literals l_{2f-1} and l_{2f}.

$$\begin{aligned} gain(x_f) &= gain(l_{2f-1}) + gain(l_{2f}) \\ &= - \sum_{(j \,\forall\, \{l_{2f-1}, c_j\} \in E)} w_j \Delta U_j - \sum_{(j \,\forall\, \{l_{2f}, c_j\} \in E)} w_j \Delta U_j \end{aligned} \tag{A.19}$$

Note that out of the two literals, one of them is true (one) and the other is false (zero) in any given configuration, so we evaluate the gradient contributions of the two separately. We define $pl(x_f)$ as the literal of x_f that is currently 1 and $nl(x_f)$ as the literal of x_f that is 0. Equation A.19 can then be re-written as:

$$gain(x_f) = gain(nl(x_f)) + gain(pl(x_f)) \tag{A.20}$$

Interestingly, $gain(pl(x_f))$ is always a negative value and its absolute value is popularly known as the break-value ($break_{x_f}$, analogous to but not the same as break-values that we defined in context of polynomial pseudo derivatives). This is because a clause (in CNF, a disjunction) is satisfied when any one of its literals is one. So, when a literal that is currently 1 is flipped to 0, it changes the clause status from satisfied to unsatisfied or remains the same. Therefore, break-value depicts the number of clauses that changed from satisfied to unsatisfied. Similarly, $gain(nl(x_f))$ is always a positive value and is known as the make-value ($make_{x_f}$, analogous to but not the same as make-values that we defined in context of polynomial pseudo derivatives). It depicts the number of clauses that changed from unsatisfied to satisfied. Before going ahead with the gain calculation, the linear transformation in the forward-step (a matrix-vector multiplication between the literal-vector and the adjacency matrix) that computes the sum of the literal-values in each clause is defined in equation A.21.

$$\hat{\sum}^{clause} = \hat{l}B^T \tag{A.21}$$

Case I (for the zero-valued literal, $nl(x_f)$): The gain with respect to the zero-valued literal $nl(x_f)$ can be written as shown in equation A.22.

$$\begin{aligned} gain(nl(x_f)) &= - \sum_{(j \forall \{nl(x_f), c_j\} \in E)} w_j \Delta U_j \\ &= - \sum_{(j \forall \{nl(x_f), c_j\} \in E)} w_j (U_j^{post} - U_j^{pre}) \end{aligned} \tag{A.22}$$

The variable U_j can be substituted with the Kronecker-delta function as in equation A.23. U_j equates to one when the sum of its member variables is zero (clause

is unsatisfied) and equates to zero when the sum of its member variables is non-zero (clause is satisfied).

$$U_j = \delta\left[\sum_{(i\,\forall\,\{l_i,c_j\}\in E)} l_i\right] = \delta[\sum_{j}^{clause}] \tag{A.23}$$

Since equation A.22 evaluates the CNF gain for literals that are currently 0, all clauses containing that literal would become satisfied post-flipping, i.e., $U_j^{post} = 0$ (see equation A.24).

$$gain(nl(x_f)) = -\sum_{(j\,\forall\,\{nl(x_f),c_j\}\in E)} w_j(0 - U_j^{pre}) = \sum_{(j\,\forall\,\{nl(x_f),c_j\}\in E)} w_j\delta[\sum_{j}^{clause}] \tag{A.24}$$

A clause-local variable s_j (also referred to as the make-clause indicator in Fig. 2.5d) as defined in equation A.25 can then be substituted in equation A.24 to yield the following expression for the gain as shown in equation A.26. Here $\sum_j w_j b_{j,nl(x_f)} s_j$ is the output that is computed in the backward make pass VMM (see Fig. 2.5d) and is referred to as the make-literal sum (see equation A.27) of literal $nl(x_f)$. Since $nl(x_f)$ is the zero-valued literal, its make-literal sum is also the make-value of variable x_f. The make-value of a variable is therefore the make literal sum (backward make pass output) of its zero-valued literal. To generalize the expression in equation A.26 and write it without exact information about the zero-valued literal, we write the make-value of x_f as the sum of make-literal sums scaled by the inverted literal value of both its literals as shown in equation A.28.

$$s_j = \begin{cases} 1, & \text{if } \sum_j^{clause} = 0 \\ 0, & \text{otherwise} \end{cases} \tag{A.25}$$

$$gain(nl(x_f)) = \sum_{(j \forall \{nl(x_f), c_j\} \in E)} w_j s_j = \sum_{j=1}^{M} w_j b_{j,nl(x_f)} s_j \tag{A.26}$$

$$\sum_{nl(x_f)}^{make} = \sum_{j=1}^{M} w_j b_{j,nl(x_f)} s_j \tag{A.27}$$

$$\begin{aligned} gain(nl(x_f)) &= (1 - l_{2f-1}) \sum_{l_{2f-1}}^{make} + (1 - l_{2f}) \sum_{l_{2f}}^{make} \\ &= (1 - l_{2f-1}) \sum_{j=1}^{M} w_j b_{j,2f-1} s_j + (1 - l_{2f}) \sum_{j=1}^{M} w_j b_{j,2f} s_j \end{aligned} \tag{A.28}$$

Case II (for the one-valued literal, $pl(x_f)$): The gain with respect to the one-valued literal $pl(x_f)$ can be written as shown in equation A.29.

$$\begin{aligned} gain(pl(x_f)) &= - \sum_{(j \forall \{pl(x_f), c_j\} \in E)} w_j (U_j^{post} - U_j^{pre}) \\ &= - \sum_{(j \forall \{pl(x_f), c_j\} \in E)} w_j (U_j^{post} - 0) \end{aligned} \tag{A.29}$$

Since we are considering the gain for $pl(x_f)$ (the literal that is currently one), U_j^{pre} can be set to 0 in equation A.29, since clause j is satisfied. Subsequently, U_j^{post} can be replaced with $\delta[\sum_j^{clause} - 1]$, where $\sum_j^{clause}$ denotes the sum of literals pre-flipping and the "1" is subtracted from it to reflect the literal $pl(x_f)$ changing from 1 to 0.

$$gain(pl(x_f)) = - \sum_{(j \forall \{pl(x_f), c_j\} \in E)} w_j \delta[\sum_{j}^{clause} - 1] \tag{A.30}$$

Additionally, we define another clause-local variable z_j known as the break-clause indicator (see Fig. 2.5e) as shown in equation A.31.

$$z_j = \begin{cases} 1, & \text{if } \sum_j^{clause} -1 = 0 \\ 0, & \text{otherwise} \end{cases} = \begin{cases} 1, & \text{if } \sum_j^{clause} = 1 \\ 0, & \text{otherwise} \end{cases} \tag{A.31}$$

Substituting this new variable in the gain expression, the final gain can then be written as shown in equation A.32, where $\sum_{j=1}^{M} w_j b_{j,pl(x_f)} z_j$ is the output that is computed in the backward break pass VMM and is known as the break-literal sum of literal $pl(x_f)$ denoted by $\sum_{pl(x_f)}^{break}$ (see Fig. 2.5e).

$$gain(pl(x_f)) = - \sum_{(j \forall \{pl(x_f), c_j\} \in E)} w_j z_j = - \sum_{j=1}^{M} w_j b_{j,pl(x_f)} z_j = - \sum_{pl(x_f)}^{break} \tag{A.32}$$

Since $pl(x_f)$ is the one-valued literal, its break-literal sum is also the break-value of variable x_f. The break-value of a variable is therefore the break literal sum (backward break pass output) of its one-valued literal. To generalize the expression in equation A.32 and write it without exact information about the one-valued literal, we write the gain of $pl(x_f)$ as the sum of break-literal sums scaled by the non-inverted literal value of both literals of x_f as shown in equation A.33. The absolute value of the expression in equation A.33 is the break-value of variable x_f.

$$\begin{aligned} gain(pl(x_f)) &= -l_{2f-1} \sum_{l_{2f-1}}^{break} -l_{2f} \sum_{l_{2f}}^{break} \\ &= -l_{2f-1} \sum_{j=1}^{M} w_j b_{j,2f-1} z_j - l_{2f} \sum_{j=1}^{M} w_j b_{j,2f} z_j \end{aligned} \tag{A.33}$$

The net gain of the variable is the sum of gains associated with its positive and negated literals or in other words the difference between make and break-values (see equation A.20) and therefore it can be written as shown in equations A.34 and A.35.

The clause-weight vector $\hat{w}$ is unitary for K-SAT problems.

$$gain(x_f) = \{(1 - l_{2f-1}) \sum_{l_{2f-1}}^{make} + (1 - l_{2f}) \sum_{l_{2f}}^{make}\} - \{l_{2f-1} \sum_{l_{2f-1}}^{break} + l_{2f} \sum_{l_{2f}}^{break}\} \tag{A.34}$$

$$gain(x_f) = \{(1-l_{2f-1})(\hat{w} \circ \hat{s}) \cdot \hat{b}_{2f-1} + (1-l_{2f})(\hat{w} \circ \hat{s}) \cdot \hat{b}_{2f}\} - \{l_{2f-1}(\hat{w} \circ \hat{z}) \cdot \hat{b}_{2f-1} + l_{2f}(\hat{w} \circ \hat{z}) \cdot \hat{b}_{2f}\} \tag{A.35}$$

A.4 Single-step backward pass with three-terminal crossbar arrays

When passive cross point devices (like 0T1R memristors) are replaced with three-terminal devices (like 1T1R memristor bit-cells), the backward step can be executed in the same crossbar array in a single cycle, instead of using two separate arrays like shown in Fig. 2.4e. To demonstrate this, consider the gradient computation in polynomial forms and CNF separately as described below.

Case A (Polynomials):

Using equation A.17, one can write down the gradient with respect to a single variable x_f as follows (we assume all monomial factors to be one for simplicity):

$$\Delta H_{x_f} = (1 - x_f) \sum_{f}^{make} - x_f \sum_{f}^{break} = \hat{s} \cdot \hat{b}_f (1 - x_f) - \hat{z} \cdot \hat{b}_f x_f = (-\hat{z}\, x_f + \hat{s} - \hat{s} x_f) \cdot \hat{b}_f \tag{A.36}$$

Note that we can rewrite $\hat{s} = \hat{s} \circ \hat{s}$ (as elementwise product with itself) without loss

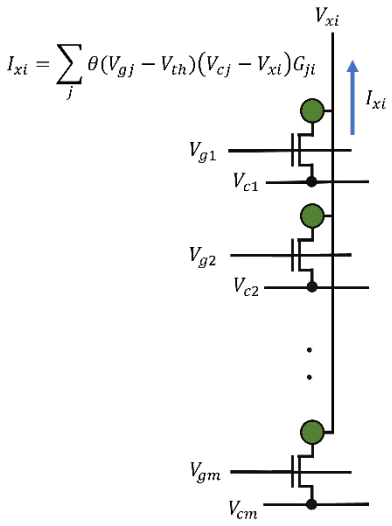


Figure A.1: The transistor in each cell acts like a switch, mathematically represented by the shifted Heaviside step function $\theta(V_{gj} - V_{th})$. When the switch is closed ($V_{gj} \geq V_{th}$), a current proportional to the product of the voltage difference across the cell and the conductance of the memristor flows. When the switch is open ($V_{gj} < V_{th}$), zero current flows.

of generality since it is a binary-valued vector. Also, based on definitions of $\hat{s}$ and $\hat{z}$ (in equations A.12 and A.15), one can see that both vectors cannot have their entries at the same position to be equal to one. This means $s_j z_j$ is always equal to zero. This allows us to re-write equation A.36 as shown below:

$$\Delta H_{x_f} = (\hat{s} \circ \hat{z} - \hat{z}\, x_f + \hat{s} \circ \hat{s} - \hat{s}\, x_f) \cdot \hat{b}_f = ((\hat{s} + \hat{z}) \circ (\hat{s} - x_f)) \cdot \hat{b}_f \tag{A.37}$$

The same equation can then be expanded using a summation as seen below:

$$\Delta H(x_f) = \sum_{j=1}^{M} (s_j + z_j)(s_j - x_f) b_{fj} \tag{A.38}$$

The expression inside the summation of equation A.38 denotes a triple product and is identical to the expression (equation A.39) for current flowing in a bit line of a 1T1R crossbar array (see Fig. A.1).

$$I_f = \sum_{j=1}^{M} \theta(V_{gj} - V_{th})(V_{Cj} - V_{Xf})G_{jf} \tag{A.39}$$

Since the signal $s_j + z_j$ is binary ("+" denoting arithmetic addition here, not logical disjunction, as s_j and z_j cannot be unity at the same given time) it acts as the gating signal $gate_j$. Depending on whether $gate_j$ is 0 or 1, the voltage V_{gj} applied to the gate of the transistor in Fig. A.1 is chosen to turn the switch off or on respectively ($\theta(V_{gj} - V_{th}) = 1$ when $V_{gj} \geq V_{th}$ and $= 0$ when $V_{gj} < V_{th}$). Voltage V_{Cj} proportional to s_j ($=s_j V_0$) is applied to the other terminal of the transistor from the monomial side (along the rows) and V_{x_i} proportional to variable value x_f ($=x_f V_0$) is applied along the column lines from the variable side. The normalization factor between equations A.38 and A.39 is the unit current via on-state coupling weight, $I_0 \equiv V_0 G_{on}$.

Case B (CNFs):

One can compactly re-write equation A.35 as shown in equation A.40 (we assume all clause weights to be one).

$$gain(x_f) = \sum_{k=0}^{1} (\hat{s}(1 - l_{2f-k}) - \hat{z} l_{2f-k}) \cdot \hat{b}_{2f-k} \tag{A.40}$$

After doing manipulations like what was introduced in equation A.37 (in Case A), we get equation A.41.

$$\begin{aligned} gain(x_f) &= \sum_{k=0}^{1} (\hat{s} - \hat{s} l_{2f-k} - \hat{z} l_{2f-k}) \cdot \hat{b}_{2f-k} \\ &= \sum_{k=0}^{1} (\hat{s} \circ \hat{z} + \hat{s} \circ \hat{s} - \hat{s} l_{2f-k} - \hat{z} l_{2f-k}) \cdot \hat{b}_{2f-k} \\ &= \sum_{k=0}^{1} ((\hat{s} + \hat{z}) \circ (\hat{s} - l_{2f-k})) \cdot \hat{b}_{2f-k} \end{aligned} \tag{A.41}$$

Re-writing the same in the summation form we get equations A.42 and A.43.

$$gain(x_f) = \sum_{j=1}^{M} \sum_{k=0}^{1} (s_j + z_j)(s_j - l_{2f-k}) b_{j,2f-k} \tag{A.42}$$

$$gain(x_f) = \sum_{j=1}^{M} (s_j + z_j)(s_j - x_f) b_{j,2f-1} + \sum_{j=1}^{M} (s_j + z_j)(s_j - \bar{x}_f) b_{j,2f} \tag{A.43}$$

$$I_{x_f}^{total} = I_{x_f} + I'_{x_f} = \sum_{j=1}^{M} \theta(V_{gj} - V_{th})(V_{cj} - V_{x_f}) G_{j,2f-1} + \sum_{j=1}^{M} \theta(V_{gj} - V_{th})(V_{cj} - V'_{x_f}) G_{j,2f} \tag{A.44}$$

Like Case A, the two inner summations expressed in equations A.42–A.43 can be mapped to the current flowing through a pair of bit-lines (corresponding to the two literals of a variable) of a 1T1R crossbar array that is summed at the periphery as shown in equation A.44 (see Fig. 2.7). Depending on whether the binary-valued signal $gate_j = s_j + z_j$ is 0 or 1, the transistor gate voltage V_{gj} (see Fig. 2.7c) is set such that it turns the transistors in that row off or on respectively ($\theta(V_{gj} - V_{th}) = 1$ when $V_{gj} \geq V_{th}$ and $= 0$ when $V_{gj} < V_{th}$). The other set of voltages V_{cj} applied from the clause side (along the rows in Fig. S10) is proportional to the make clause indicator signal s_j ($=s_j V_0$), while the voltages V_{x_f} and V'_{x_f} applied along the columns (in Fig. 2.7) from the literal side are proportional to the literal values x_f ($=x_f V_0$) and $\bar{x}_f$ ($=\bar{x}_f V_0$) respectively. The $b_{j,2f-k}$ values are mapped to the conductance $G_{j,2f-k}$ of the crosspoint devices. The normalization factor between equations A.43 and A.44 is the unit current via on-state coupling weight, $I_0 \equiv V_0 G_{on}$.

A.5 3-SAT instance solved in the experimental demo

```
p cnf 14  64
 5 10 13 0
3 -11 -12 0
7 -8 -12 0
0 -2 -7 0
-6 -7 13 0
2 -8 -11 0
5 7 -9 0
-6 -7 -11 0
-5 -7 8 0
0 -6 -12 0
2 4 9 0
1 -7 9 0
7 -8 -10 0
-1 5 12 0
-3 -6 13 0
0 1 -12 0
-1 6 10 0
1 5 -10 0
-1 -11 -13 0
2 4 -8 0
5 -6 -10 0
1 -6 13 0
9 10 13 0
0 5 10 0
2 5 11 0
-1 -2 5 0
0 -8 9 0
-8 -11 -12 0
10 -11 12 0
0 -2 13 0
0 -4 -6 0
-4 -5 -12 0
1 2 -11 0
2 -3 -13 0
11 12 -13 0
0 7 13 0
2 6 -8 0
-2 -6 7 0
-2 7 -12 0
0 -4 -10 0
-4 -6 -11 0
1 -2 -13 0
3 8 -11 0
-11 12 -13 0
-1 -4 8 0
2 -4 -11 0
-3 -8 10 0
-1 6 13 0
1 6 11 0
4 -10 11 0
0 -10 12 0
0 2 -11 0
9 11 -13 0
-1 -4 10 0
0 4 -10 0
4 9 -10 0
0 3 8 0
0 -1 -13 0
0 9 -12 0
1 11 -12 0
-4 8 11 0
2 -5 6 0
-1 5 -7 0
3 5 13 0
%
0
```

Figure A.2: 3-SAT instance solved in the experimental demo.

A.6 3-SAT instance generation

Random uniform 3-SAT problems with M clauses and N variables are generated by initializing an empty list of clauses and repeatedly adding "valid" clauses to that list,

one at a time, until the number of clauses reaches the target value of M. Specifically, a "candidate" clause is generated by randomly sampling three out of $2N$ literals without replacement. Each of the $2N$ literals has an equal probability of being chosen. Two criteria are then checked to determine if the candidate clause is valid: (i) if both literals of a variable are present in the candidate clause; and (ii) if the candidate clause is already present in the clause list. If both criteria are false, such a clause is added to the list. Once the entire list of clauses is ready, a SAT solver is run on the generated instance to check if it is satisfiable.

3-SAT problems with $M = 64$ clauses are considered. This number matches the linear dimension of the memristor crossbar arrays and hence corresponds to the largest SAT problems in terms of the number of clauses that can be implemented with the experimental setup without employing time-consuming time-multiplexing techniques. Prior work [84] shows that satisfiable randomly generated uniform 3-SAT problems with 14 variables, corresponding to a clause-to-variable ratio $M/N \approx 4.57$, are among the hardest. This was confirmed by generating and solving multiple instances of 3-SAT problems with $M = 64$ and N in the range from 12 to 21.

A.7 WalkSAT Algorithm

Throughout the paper, we have used WalkSAT/SKC and WalkSAT interchangeably.

Algorithm 1 WalkSAT/SKC Algorithm

Require: 3-SAT CNF formula, MAX_FLIPS, MAX_ITER, probability p

Ensure: "true" if a satisfying assignment is found; "false" otherwise

```
for t = 1 to MAX_ITER do
    Randomly initialize variable assignments X = X_0
    for f = 1 to MAX_FLIPS do
        if X is a solution then
            return true
        end if
        Randomly choose an unsatisfied clause c
        Calculate set of break values (BV) for all member variables of c
        if min(BV) = 0 then
            Flip variable with zero break value (break ties randomly)
        else
            With probability p, select and flip a random variable in c
            With probability 1−p, select and flip variable with smallest break value (break
            ties randomly)
        end if
    end for
end for
return false
```

A.8 Discrete-time High-Order Hopfield Neural Network

Algorithm 2 Digital-Annealer-like Update dynamics in a High-Order Hopfield Neural Network

Require: High-order energy function H in PUBO form, MAX_FLIPS, MAX_ITER, temperature decrease rate r, Initial temperature T_0, and offset_increase_rate

Ensure: "true", if a satisfying assignment is found, "false" otherwise

for $t = 1$ to MAX_ITER **do**
  Randomly initialize variable assignments $X = X_0$
  $E_{\text{offset}} = 0$
  for $f = 1$ to MAX_FLIPS **do**
    Update temperature as $T(f) = T_0 e^{-rf}$
    for each variable j **do**
      Propose new state as:

$$x_j = \begin{cases} 1, & \text{if } \dfrac{\partial H}{\partial x_j} + E_{\text{offset}}(2x_j - 1) < \eta_j \\ 0, & \text{else} \end{cases}$$

      If new state resulted in a variable flip, record
    end for
    if at least one flip accepted **then**
      Choose one flip uniformly at random amongst them and update the state
      $E_{\text{offset}} = 0$
    else
      $E_{\text{offset}} = E_{\text{offset}} + \text{offset\_increase\_rate}$
    end if
  end for
end for
return false

Here, $\eta_j \sim \mathcal{N}[0, \sqrt{2\pi}\, T(f)]$ is a random variable sampled from a normal distribution with zero mean and standard deviation proportional to the temperature at that step.

Note that this algorithm equivalently implements classical Hopfield neural network algorithms [33] if offset_increase_rate is set to zero (therefore fixing E_{offset} to zero). In a

physical implementation, a variable x_j encodes a neuron state in the network, $\frac{\partial H}{\partial x_j}$ corresponds to the weighted feedback accumulated at each neuron in a particular step, while η_j represents the noise added to the accumulated feedback. On the other hand, inclusion of E_{offset} and considering all variable flips in parallel are two aspects borrowed from the digital annealing algorithm [40] and has been shown to improve baseline high-order Hopfield neural network performance by up to an order [37].

A.9 Hardware and Algorithm Simulation Details

The memristors in the crossbar arrays are assumed to have $G_{\text{ON}} = 100$ μS and $G_{\text{OFF}} = 1$ μS, whereas the memristor cell size is assumed to be $12F^2$ ($F = 32$ nm). The read voltage (V_{ON}) applied across the memristors is assumed to be 0.2 V. The estimates for the word line drivers, transimpedance amplifier and register arrays are adopted from [43], whereas those for the sense amplifier, analog multiplexer and winner take all circuits are based on SPICE simulations and physical layouts of the respective modules in 65 nm technology node and then scaling down to 32 nm. Based on the latency estimates of the circuit blocks and the timing diagram, the critical path delay in any stage (excluding the WTA) is approximately 1.4 ns. Assuming an additional slack of 0.6 ns to achieve a reasonable dynamic range in the WTA and to account for additional overheads associated with full chip design, we set the operating frequency to 500 MHz ($T_{\text{clk}} = 2$ ns), similar to [43]. The estimates for the hysteretic comparators are adopted from [43].

Fig. A.3 shows the description, count/dimension, energy, latency, and area estimates for all circuit blocks required for the two modeled hardware solvers.

For the purposes of this paper, the term Time-to-Solution (TTS) is used to refer to Time-to-99% certainty, or the time taken for the solver/heuristic to reach the optimal solution of the optimization problem with 99% certainty. Two types of TTS definitions,

(a)

Component (Description)	Phase (Shorthand)	Count / Dimension	Latency (ns)	Energy (pJ per unit of component per cycle)	Area (μm^2 per unit of component)
Forward/Backward word line driver	WL1 / WL2	2N / M	0.501	0.27	2.73
Forward/Backward Array	FA/BA	2N x M	<0.5	0.008(ON), 0.00008(OFF)	0.0123
Forward/Backward transimpedance amplifier	TIA1/TIA2	M / N	0.3	0.26	10.5135
Forward/Backward sense amplifier	SA1/SA2	2M / N	<0.09	0.0023	8.058
Forward/Backward register array	DFF1/DFF2	M / 2N	<0.1	0.023	1.915
2-to-1 Multiplexer	MUX	N	0.0073	0.00023	2.084
Winner-take-all circuit	WTA	N	0 - 2	0.037	29.01

(b)

Phase (Shorthand)	Phase (Name)	Active Components
FP	Forward Pass	WL1(2N), FA, TIA1(M), SA1(2M)
RS	Random Clause Select	DFF1(M)
BMP	Backward Make Pass	WL2(1), BA, MUX(N), TIA2(N), SA2(N)
BBP	Backward Break Pass	WL2(M), BA, MUX(N), TIA2(k), SA2(k)
BWP	Backward WTA Pass	BA, TIA2(k), WTA(N)
FL	Variable Select & Flip	DFF2(2N)

(c)

Component (Description)	Phase (Shorthand)	Count / Dimension	Latency (ns)	Energy (pJ per unit of component)	Area (μm^2 per unit of component)
Forward/Backward word line driver	WL1 / WL2	2N / M	0.501	0.27	2.73
Forward/Backward Array	FA/BA	2N x M	<0.5	0.008(ON), 0.00008(OFF)	0.0123
Forward/Backward transimpedance amplifier	TIA1/TIA2	M / 2N	0.3	0.26	10.5135
Sense amplifier	SA	2M	<0.09	0.0023	8.058
Forward/Backward register array	DFF1/DFF2	2M / 2N	<0.1	0.023	1.915
Hysteresis Comparator	HC	N	0.025	0.25	3.4865

(d)

Phase (Shorthand)	Phase (Name)	Active Components
FP	Forward Pass	DFF2(2N), WL1(2N), FA, TIA1(M), SA1(2M)
BP	Backward Pass	DFF1(2M), WL2(M), BA, TIA2(2N), HC(N)

Figure A.3: (a) Area and energy breakdown for circuit components of WalkSAT/SKC hardware based on 32nm technology node, (b) Pipeline stage breakdown and active components of WalkSAT/SKC hardware, (c) area and energy breakdown for circuit components of high-order HNN hardware based on 32nm technology node, (d) pipeline stage breakdown and active components of high-order HNN hardware.

namely instance-wise TTS and batch TTS are considered. Instance-wise TTS as the name suggests is the TTS of the heuristic when run on a specific instance, whereas the batch TTS refers to the median of the instance-wise TTS values across all instances belonging to that problem size.

For a heuristic (either WalkSAT/SKC or high-order HNN) that is run with a certain value of MAX_FLIPS and MAX_ITER, we first measure Run-Length Distribution (RLD) $\hat{P}$, which is the cumulative distribution function of the number of variable flips required to find a solution (run-length) during successful iterations only [82]. It is defined as

$$\hat{P}(\text{run\_length} \leq j) = \frac{|\{t \mid \text{run\_length}(t) \leq j\}|}{\text{MAX\_ITER}}, \tag{A.45}$$

where run_length(t) is the run-length of the t^{th} iteration that was successful and $|\cdot|$ denotes the set cardinality function. We then compute the probability of success as

$$\text{success\_rate} = \frac{\text{\# of successful iterations}}{\text{MAX\_ITER}}. \tag{A.46}$$

Subsequently, the instance-wise TTS is computed using

$$\text{TTS}_i = \begin{cases} \text{MAX\_FLIPS} \times \dfrac{\log(1-0.99)}{\log(1-\text{success\_rate})}, & \text{if success\_rate} < 0.99, \\ \hat{P}^{-1}(0.99), & \text{if success\_rate} \geq 0.99, \end{cases} \tag{A.47}$$

where TTS_i is the TTS of instance i and $\hat{P}^{-1}(0.99)$ is the inverse of the function $\hat{P}$ at 0.99.

For all algorithm simulations, 80 generated instances of 14-variable 64-clause 3-SAT problems were used and instances `uf20-(921--1000)`, `uf50-(921--1000)`, `uf75-(51--100)` and `uf100-(921--1000)` from SATLIB were used for 3-SAT prob-

lems with $N = 20, 50, 75$ and 100 respectively. Optimal values of hyperparameters p, MAX_FLIPS for WalkSAT/SKC and r, T_0, MAX_FLIPS for HO-HNN were obtained separately for each problem size. Grid search was implemented to find the set of hyperparameters that led to minimum batch TTS. The grid search was performed on a different set of instances, specifically `uf20-(901--920)`, `uf50-(901--920)`, `uf75-(51--60)` and `uf100-(901--920)` for 3-SAT problems with $N = 20, 50, 75$ and 100 respectively. Due to similarity in number of variables, the hyperparameters for solving the 14-variable 3-SAT instances were set to the optimal hyperparameters for the 20-variable 3-SAT instances.

A.10 Simulation Details of Real-Valued Gradient Computing

The variables are assumed to be within the range $[x_{\min}, x_{\max}]$, where $x_{\min} > 0$ and $x_{\max} \leq 1$. The analog voltages corresponding to the variables are denoted by x_i (where $i \in \{1, 2, 3, 4\}$ in this example). For the version studied here, the mapping between the physical analog voltages depicting the polynomial's variables and their values in arbitrary units is 1 V/a.u. Shifted version of these voltages given by $x'_i = (x_i + v_{b1})$, undergo a log transformation, using an op-amp with a diode connected between its output and inverting input to generate the internal voltages V''_i (see equation A.48, where m_1 is the product of thermal voltage and the subthreshold factor of the diode). This is then followed by an amplification stage with gain $(-A_{V1})$ to generate voltages V_{xi} (see equation A.49) that are then applied to the word lines of the forward crossbar array. The gain A_{V1} is required to amplify the log compressed into the dynamic range of the memristor input read voltage.

$$V''_i = v_{b1} - m_1 \ln\left(\frac{x'_i - v_{b1}}{R_1 I_S}\right) \tag{A.48}$$

$$V_i = v_{b1} + A_{V1} m_1 \ln\left(\frac{x_i' - v_{b1}}{R_1 I_S}\right) \tag{A.49}$$

The bit line voltage of the forward array is equal to v_{bot} (see equation A.50) and is set by the bias voltage applied to the non-inverting end of the TIAs.

$$V_{\text{bot}} = v_{b1} - A_{V1} m_1 \ln(R_1 I_S) \tag{A.50}$$

The size of the forward array is $N \times (M+1)$, where N and M denote the number of variables and monomials in the polynomial respectively. The extra bit line is to compute an offset current that is subtracted from the other bit line currents (see equation A.51).

$$V_{\text{off}}^{\text{TIA}} = v_{b1} - R_f G_{\text{off}} A_{V1} m_1 \ln \prod_{i=1}^{N} (x_i' - v_{b1}) \tag{A.51}$$

This offset current captures the leakage currents flowing through OFF-state memristors with activated word lines in each bit line. Note that such an offset correction could potentially be used in the gradient computing hardware for binary-valued variables but is not necessary due to wider margins. On the other hand, in the real-valued gradient computing hardware, currents in the forward array are log-compressed and vary continuously in their valid range, resulting in narrower margins. The TIAs at the output of the forward crossbar array convert the bit line currents to proportional voltages (see equation A.52, where t_j denotes the j-th monomial and E the set of all edges connecting monomials with their member variables) that are subtracted from the voltage of the offset bit line's TIA output using difference amplifiers. The difference amplifier outputs constitute control-gate voltages for the floating-gate cells in the backward array and are

proportional to the log of monomial products (see equation A.53).

$$\begin{aligned} V_j^{\text{TIA}} = v_{b1} &- R_f G_{\text{on}} A_{V1} m_1 \ln \prod_{i\,\forall\{x_i,t_j\}\in E} (x_i' - v_{b1}) \\ &- R_f G_{\text{off}} A_{V1} m_1 \ln \prod_{i\,\forall\{x_i,t_j\}\notin E} (x_i' - v_{b1}) \end{aligned} \tag{A.52}$$

$$V_{cj} = V_{\text{off}}^{\text{TIA}} - V_j^{\text{TIA}} = R_f (G_{\text{on}} - G_{\text{off}}) A_{V1} m_1 \ln \prod_{i\,\forall\{x_i,t_j\}\in E} (x_i' - v_{b1}) \tag{A.53}$$

The source line voltage of the floating-gate cells in the backward array, denoted by V_{sl}, are given by expression in equation A.54 (generated by passing the internal voltages V_l'' through an amplification stage with gain A_{V2} and bias voltage v_{b2}).

$$V_{sl} = (1 + A_{V2}) v_{b2} - A_{V2} v_{b1} + A_{V2} m_1 \ln\left(\frac{x_l' - v_{b1}}{R_1 I_S}\right) \tag{A.54}$$

On setting $A_{V2} = R_f (G_{\text{on}} - G_{\text{off}}) A_{V1}$, the net control-gate to source voltage drop across the floating-gate cells and the subthreshold current through them are given by equations A.55 and A.56 respectively. Here $V_{(j,l)}^{\text{th}}$ is the threshold voltage of the cell sharing the j-th control gate and l-th source line and m_2 is the product of subthreshold factor and thermal voltage of the device.

$$\begin{aligned} V_{(j,l)}^{\text{cgs}} &= V_{cj} - V_{sl} \\ &= R_f (G_{\text{on}} - G_{\text{off}}) A_{V1} m_1 \ln\left(\frac{\prod_{i\,\forall\{x_i,t_j\}\in E} (x_i' - v_{b1})}{x_l' - v_{b1}}\right) \\ &\quad - (1 + A_{V2}) v_{b2} + A_{V2} v_{b1} + A_{V2} m_1 \ln(R_1 I_S) \end{aligned} \tag{A.55}$$

$$I_{(j,l)} = I_0 \exp\left(\frac{V_{(j,l)}^{\text{cgs}} - V_{(j,l)}^{\text{th}}}{m_2}\right) \tag{A.56}$$

The threshold voltage of the cells is set proportional to the coefficient (a_j) of the j-th monomial and is given by equation A.57, where V_{t0} is the threshold of the cells that correspond to monomials with coefficient equal to one. Monomials with higher coefficient values ($a_j > 1$) have lower threshold voltages, resulting in higher subthreshold currents.

$$V^{\text{th}}_{(j,l)} = V_{t0} - m_2 \ln(a_j) \tag{A.57}$$

On setting $R_f(G_{\text{on}} - G_{\text{off}})A_{V1}m_1 = m_2$, substituting equations A.54, A.56 and A.57 into A.55 and rearranging the terms, one obtains an expression for the current given by equation A.58, where the constant I''_0 is given by equation A.59.

$$\begin{aligned} I_{(j,l)} &= I_0 \exp\left[\frac{m_2 \ln\left(\frac{\prod_{i\,\forall\{x_i,t_j\}\in E}(x'_i - v_{b1})}{x'_l - v_{b1}}\right) - \left(1 + \frac{m_2}{m_1}\right)v_{b2} + \frac{m_2}{m_1}v_{b1}}{m_2}\right. \\ &\qquad \left. + \frac{m_2 \ln(R_1 I_S) - V_{t0} + m_2 \ln(a_j)}{m_2}\right] \\ &= I''_0 \frac{a_j \prod_{i\,\forall\{x_i,t_j\}\in E} x_i}{x_l} \end{aligned} \tag{A.58}$$

$$I''_0 = I_0 \exp\left(\frac{-(1 + m_2/m_1)v_{b2} + (m_2/m_1)v_{b1} + m_2 \ln(R_1 I_S) - V_{t0}}{m_2}\right) \tag{A.59}$$

$$\frac{\partial H}{\partial x_l} \propto I_l = \sum_{j=1}^{M} I_{(j,l)} = I''_0 \sum_{j=1}^{M} \frac{a_j \prod_{i\,\forall\{x_i,t_j\}\in E} x_i}{x_l} \tag{A.60}$$

The expression of the current in equation A.58 is proportional to the partial derivative of the j-th monomial with respect to variable x_l (provided they are in the range $[x_{\min}, x_{\max}]$ V). The currents from all the cells in a column get summed up (according to Kirchhoff's law) and this sum is proportional to the partial derivative of the high-degree polynomial

function with respect to the variable corresponding to that column (see equation A.60). It is worth noting that while the cost/energy function of most combinatorial optimization problems are defined with variables $x \in [0, 1]$, a new energy function can be defined with variables $x_{\text{new}} \in [x_{\min}, x_{\max}]$, by substituting $x = (x_{\text{new}} - x_{\min})/(x_{\max} - x_{\min})$ in the original function. Furthermore, while the current hardware description is only valid for positive monomial coefficients ($a_j \geq 0$), a differential-mode implementation with two bit lines per variable in the backward array can enable both positive and negative monomial coefficients.

Each threshold voltage that is programmed to each floating-gate cell has the following range: $V_{\text{th,min}} \leq V^{\text{th}}_{(j,l)} \leq V_{t0} < V_{\text{th,max}}$. The threshold voltage $V^{\text{th}}_{(j,l)} = V_{t0}$ when $a_j = 1$ (see equation A.57). The absolute maximum value ($V_{\text{th,max}}$) is determined by the device properties, and $V^{\text{th}}_{(j,l)} = V_{\text{th,max}}$ if $\{x_i, t_j\} \notin E$ (i.e., variable x_i is not present in monomial t_j). Equation A.61 describes the relationship between the coefficient bit-precision (δ, in number of bits) and the threshold voltage limits when tuning error is assumed to be zero.

$$\delta = \frac{V_{t0} - V_{\text{th,min}}}{m_2 \ln 2} \tag{A.61}$$

To maintain subthreshold operation regime, the control-gate to source voltage applied across each floating-gate cell is limited by $0 < V^{\text{cgs}}_{(j,l)} < V_{\text{th,min}}$. According to equation A.55, the maximum limit is achieved for the case shown in equation A.62 (when the corresponding monomial comprises of two variables, with both equal to the maximum value, $x_{\max}$).

$$V^{\text{cgs}}_{(j,l)} = m_2 \ln(x_{\max}) - (1 + m_2/m_1)v_{b2} + (m_2/m_1)v_{b1} + m_2 \ln(R_1 I_S) = V_{\text{th,min}} \tag{A.62}$$

On rearranging the terms in equation A.62, it gives an expression for a minimum value

for the bias voltage v_{b2}, as described in equation A.63.

$$v_{b2} \geq \frac{m_2 \ln(x_{\max}) + (m_2/m_1)v_{b1} + m_2 \ln(R_1 I_S) - V_{\text{th,min}}}{1 + m_2/m_1} \tag{A.63}$$

The minimum limit for $V^{\text{cgs}}_{(j,l)}$ is reached when all the variables in the monomial with the highest degree (K) are set to the minimum value, $x_{\min}$ (see equation A.64). Substituting for the minimum value for the bias voltage v_{b2} and rearranging the terms, one gets the expression in equation A.65.

$$V^{\text{cgs}}_{(j,l)} = m_2(K-1)\ln(x_{\min}) - (1 + m_2/m_1)v_{b2} + (m_2/m_1)v_{b1} + m_2 \ln(R_1 I_S) = 0 \tag{A.64}$$

$$K \leq 1 + \frac{\ln(x_{\max}) - V_{\text{th,min}}/m_2}{\ln(x_{\min})} \tag{A.65}$$

For a given range for the variables $(x_{\min}, x_{\max})$, this highlights a trade-off in the maximum order of the monomial with the bit-precision of the monomial coefficients. Note that for $x_{\max} \leq 1$, the denominator in equation A.65 is negative and therefore for larger values of the maximum degree K of the polynomial, higher values of $V_{\text{th,min}}$ are required. Higher values for $V_{\text{th,min}}$ mean lesser bit-precision for monomial coefficients. Furthermore, for a fixed value of $V_{\text{th,min}}$, there exists a trade-off between the maximum order and the dynamic range of the variables ($\zeta = x_{\max}/x_{\min}$). Larger values of K are possible with smaller values of ζ, which also results in smaller swing in the forward crossbar array input voltages, V_{xi}. Therefore, for smaller values of ζ, higher values of the gain A_{V1} are required. This results in higher power consumption and susceptibility to PVT variations.

Circuit and device parameters relevant to real-valued gradient computing are detailed in Fig. A.4.

Parameter	Value	Parameter	Value
x_{min}	0.028V	m_2	**130mV
x_{max}	1.0V	A_{V2}	4.66
v_{b1}	0.1V	v_{b2}	-0.93V
R_1	10kΩ	I_0	#100nA
I_S	0.03fA	V_{t0}	2.5V
m_1	*27.88mV	$V^{th,min}$	2.0V
A_{V1}	2	$V^{th,max}$	##3.5V
G_{on}	100μS	σ_{Gon}	$3μS
G_{off}	1μS	σ_{Goff}	$0.25μS
R_f	23.5kΩ	Δv_{th}	^^5.2mV

*$m_1=\eta(k_BT/q)$, where η =1.07 and k_BT/q = 26 mV. **$m_2=\eta(k_BT/q)$, where η=5[15] and k_BT/q = 26 mV. #I_0 is set to 100nA which corresponds to the highest analog state within in the subthreshold regime of eFlash cells. ##$V^{th,max}$ is set to 3.5V which is the highest threshold voltage. ^^ measure of flash threshold voltage tuning precision and corresponds to roughly 2% precision. Threshold voltages are sampled from a uniform distribution with bounds [v_{th}- Δv_{th}, v_{th}+ Δv_{th}], where v_{th} is the desired threshold voltage.

Figure A.4: Values of parameters used in simulations for real-valued gradient computation. eFlash cell data obtained from [90] and standard deviations in memristor tuning is the same as that used in Chapter 2.4.

A.11 Comparison with other approaches

In Fig. 2.20, performance of the proposed hardware is compared with other technologies in solving high-order combinatorial optimization problems, specifically the uniform random 3-SAT problem. The other technologies include the Coherent Ising Machine [91], D-Wave's 2000Q quantum annealer [92], sparse Ising Machines (sIM) [93], memristor crossbar based second-order discrete-time HNNs (mem-SO-HNN) [43, 37] and the Augmented Ising Machine (AIMs) [78]. All results pertaining to our hardware are based on experimentally grounded simulations (see Appendix A.9 and Fig. 2.16), whereas the results for D-Wave and FPGA-based sIM are based on experimental results. The results for CIM, MTJ p-bit based sIM, mem-SO-HNN and AIMs are based on modeling results.

For CIM, the TTS for the SAT-CFC [114] algorithm on 100-variable uniform random 3-SAT instances (in number of round trips) was obtained from [114] and multiplied with the single round trip time (2.5N ns) of NTT-CIM system [91] to obtain the hardware TTS

in seconds. A lower bound for the power consumption of the system was estimated by considering the power consumption of the FPGAs and is roughly equal to 50W (two Xilinx Virtex-7 FPGAs are used in the NTT-CIM system, see [91, 166]). The ETS was obtained by taking the product of the power with TTS. For the D-Wave system, experimental TTS values were obtained from [78] for 20-variable SAT problem. The results for the sparse Ising Machine have two parts: FPGA-based and Nanodevice-based. Results pertaining to the former are based on experimental results from FPGA, while the latter is based on modeling results where spin and random number generation circuitry is assumed to be replaced with CMOS + Magnetic Tunnel Junction (MTJ) based p-bits. Note that although [93] reports something called Time-to-Solution, it is different from how it is defined in this paper. For example, TTS_{99} in [93] is the time taken to satisfy 99% of the clauses of the 3-SAT problem, whereas TTS_{99} in our paper is the time taken to fully solve the problem with 99% certainty. Since we do not measure time taken to satisfy a fraction of the clauses, we chose to compare TTS_{100} values from [93] (which refers to time taken to satisfy 100% of the clauses, or in other words the entire SAT instance) with the mean time to solve 3-SAT problems using our hardware. We denote it as $\langle L \rangle$ to distinguish with the definition of TTS used in this paper. The TTS_{100} (or $\langle L \rangle$) for the FPGA-sIM (with k=4 configuration and 30 MHz clock frequency) on 100-variable 3-SAT problems was directly obtained from [93]. The throughput for this problem size is $5.5\text{x}10^{10}$ flips-per-second (fps). When nanodevices are used, the throughput increases to approximately $1.8\text{x}10^{12}$ fps (see [93]). If the dynamics of the Ising Machine remain the same and therefore the number of effective flips taken to solve the problem, the $\langle L \rangle$ for a Nanodevice-sIM on 100-variable SAT problem is obtained by scaling the $\langle L \rangle$ of the FPGA-sIM with the ratio of their throughputs. The power consumption of the FPGA-sIM is 75W (equal to the power of the AMD Ultrascale+ FPGA used in the work, see [93, 167]). The power consumption of Nanodevice-sIM is calculated by assuming 20W per p-bit based spin,

with a total of 1935 p-bits in the sparse Ising Machine. We also consider a memristor crossbar based HNN that can implement only second-order interactions, referred to as SO-HNN. A single memristor crossbar array with dimensions equal to (N+M) x 2(N+M) is used to implement the second-order coupling matrix of the QUBO form of the 3-SAT problem, using analog memristor conductance. Factor 2 arises from the requirement of positive and negative weights. Apart from the increased crossbar size, the requirement of analog-tunable memristors is another key feature that sets such SO-HNNs apart from the HO-HNN or WalkSAT/SKC hardware proposed in this work, that only require binary memristors. The remaining peripherals of the crossbar array include the appropriate number of WL drivers, register arrays, TIAs and Hysteretic Comparators (discussed in Appendix A.9). Due to the similarity in hardware components, we assume an optimistic 2 ns clock period to provide fair comparison with the proposed hardware, even though the crossbar sizes required by SO-HNN are larger for the same problem sizes. The TTS (in number of flips) for solving a 100-variable SAT problem is obtained from Ref. 14 (QUBO, parallel updates) and subsequently scaled with the clock period to obtain hardware TTS (in seconds). Energy per cycle for the SO-HNN hardware is calculated by adding the energy contributions of the constituent peripherals and crossbar array (using values from Fig. A.3c) and is then scaled up by the TTS (in number of flips) to obtain ETS. The TTS of Augmented Ising Machine (AIMs) to solve 100-variable SAT problems is also considered. A crude power estimate for N=500 is provided in [100], but it is unclear how that number scales with problem size. The area for an N=500 AIMs chip is given to be 13x13 mm^2. Based on our previous analysis, the area for AIMs seems to scale with N^2 and therefore a rough estimate for such a system with N=100 would be 6.76 mm^2. Note that AIMs implement continuous-time dynamics, whereas our modeled hardware implements discrete-time dynamics in its current version. Despite that, the performance in terms of TTS is at par. Continuous-time systems are supposed to have much higher

navigational efficiency compared to their discrete-time parts, resulting in faster TTS. The peripherals of the proposed high-degree gradient computation hardware can be modified to implement continuous-time dynamics and potentially achieve even faster TTS. Our hardware has at least 7.7x and 1.4x faster TTS when compared to discrete (CIM) and continuous-time systems (AIMs). Even at iso-frequency with FPGA-sIM (assuming the operation frequency of our hardware is scaled down to 30 MHz), the resultant mean time to solve 100-variable problems is six orders of magnitude faster, owing to the capability of implementing all-to-all connectivity and solving 3-SAT problems in their native form using advanced heuristics like WalkSAT/SKC. It has three orders of magnitude lower ETS when compared to a memristor based SO-HNN for 100-variable SAT problems, owing to compact high-order hardware footprint and higher navigational efficiency of native high-order solvers. The proposed hardware has at least two orders of magnitude higher throughput per watt and throughput per unit area compared to all other technologies. A higher throughput per unit area indicates that at iso-area multiple replicas of the Ising Machine can be launched in parallel to attain even faster TTS.

Appendix B

Supplementary Notes for Chapter 5

B.1 Relation between PUBO and the Ising Model

Polynomial objective functions over discrete variables commonly arise in several combinatorial optimization problems. Formally they are defined as shown in equation B.1, where $X = \{x_1, x_2, \ldots, x_N\}$ is the set of all variables, $T = \{t_1, t_2, \ldots, t_M\}$ is the set of all monomials in the function and E is the set of all edges in the bi-partite graph formed by non-overlapping vertices in X and T. An edge formed by $\{x_i, t_j\}$ exists in E if degree of x_i in monomial t_j is non-zero. Constant terms in H, are not considered here due to their absence in the gradient.

$$H = \sum_{j=1}^{M} t_j \ , \qquad t_j = a_j \prod_{i \ \forall \{x_i, t_j\} \in E}^{N} x_i \tag{B.1}$$

Such a polynomial is referred to as Polynomial Unconstrained Binary Optimization (PUBO) model when the variables are binary, i.e., $x_i \in \{0, 1\}$. It is referred to as the Ising model or Polynomial Unconstrained Spin Optimization (PUSO) model when variables are bipolar, i.e., $x_i \in \{-1, 1\}$. When the maximum order of any monomial is two, they are

referred to as Quadratic-UBO/USO (QUBO/QUSO, or the second-order Ising model). A problem in PUBO defined over $x_i \in \{0, 1\}$, can be converted to PUSO/Ising defined over $s_i \in \{-1, 1\}$, by substituting, $x_i = (1 + s_i)/2$ (see equation B.2). Similarly, the opposite can be achieved by substituting $s_i = 2x_i - 1$. The pseudo derivative of the PUBO w.r.t. a binary variable x_f is expressed in equation B.3. The pseudo derivative of the PUSO w.r.t. the same variable's spin counterpart is expressed in equation B.4. Note that the factor of 1/2 follows from the definition of difference quotient of discrete domain functions. On substituting, $\Delta s_f = 2\Delta x_f$, and re-substituting $x_i = (1 + s_i)/2$, the pseudo-derivative of PUSO equates exactly to that of PUBO (see equation B.5).

$$\text{PUBO}(\hat{x}) = H(x_1, x_2, \ldots, x_N), \qquad \text{PUSO}(\hat{s}) = H\left(\frac{1+s_1}{2}, \frac{1+s_2}{2}, \ldots, \frac{1+s_N}{2}\right) \tag{B.2}$$

$$\begin{aligned}\Delta\text{PUBO}_{x_f} = H_{x_f}\Delta x_f = \{&H(x_1, x_2, \ldots, x_f = 1, \ldots, x_N) \\ &- H(x_1, x_2, \ldots, x_f = 0, \ldots, x_N)\}\Delta x_f\end{aligned} \tag{B.3}$$

$$\Delta\text{PUSO}_{s_f} = H_{s_f}\Delta s_f = \left\{\frac{H\left(\frac{1+s_1}{2}, \ldots, \frac{1+(s_f=1)}{2}, \ldots, \frac{1+s_N}{2}\right) - H\left(\frac{1+s_1}{2}, \ldots, \frac{1+(s_f=-1)}{2}, \ldots, \frac{1+s_N}{2}\right)}{2}\right\}\Delta s_f \tag{B.4}$$

$$\begin{aligned}\Delta\text{PUSO}_{s_f} &= \{H\left(\frac{1+s_1}{2},\ldots,\frac{1+(s_f=1)}{2},\ldots,\frac{1+s_N}{2}\right)- \\ &\quad H\left(\frac{1+s_1}{2},\ldots,\frac{1+(s_f=-1)}{2},\ldots,\frac{1+s_N}{2}\right)\}\Delta x_f \\ &= \{H(x_1,x_2,\ldots,x_f=1,\ldots,x_N)-H(x_1,x_2,\ldots,x_f=0,\ldots,x_N)\}\Delta x_f \\ &= \Delta\text{PUBO}_{x_f}\end{aligned} \tag{B.5}$$

Since most discrete optimization techniques are guided by the pseudo partial derivatives, this result implies (an already well-known fact) that any optimization performed on the PUBO (or QUBO) model would be equivalent to that performed on the PUSO /Ising model (or QUSO).

B.2 Relation between CNF and the Ising Model

Many Constrained Satisfaction Problems (CSPs) including Boolean Satisfiability (K-SAT) and Maximum Satisfiability (MAXSAT) result in objective functions that are natively defined in the Conjunctive Normal Form (CNF) or Weighted CNF (WCNF). Note that in the rest of this paper we shall refer to both formulations as CNF. A CNF formula comprises of conjunction of clauses (from a set $C = \{c_1, c_2, \ldots, c_M\}$ of M clauses) where a clause is a disjunction of literals (from a set $L = \{l_1, l_2, \ldots, l_{2N-1}, l_{2N}\}$ s.t. $l_{2i-1} = x_i$ (positive literal of variable x_i), $l_{2i} = \bar{x}_i$ (negated literal of variable x_i) of $2N$ literals arising out of N variables) or in other words AND of ORs (see Equation B.6). Similar to PUBOs, E here is the set of edges defined by $\{l_i, c_j\}$ such that literal l_i is a member of clause c_j. In its most general form, each clause is also associated with a positive valued weight w_j. The goal in such problems is to find the literal assignments that satisfies all clauses (like in K-SAT where $w_j = 1,\ \forall j \in [1, M]$) or maximizes the sum of weights

of satisfied clauses (MAXSAT). The underlying objective natively encoded in the CNF representation in such problems can be translated to a PUBO as shown in Equation B.7. The merit function H denotes the weighted sum of unsatisfied clauses, and the objective is to minimize it.

$$\text{CNF} = \bigwedge_{j=1}^{M} c_j, \qquad c_j = \bigvee_{i \ \forall\{l_i,c_j\}\in E}^{2N} l_i \quad \text{and has weight } w_j \tag{B.6}$$

$$H = \sum_{j=1}^{M} w_j U_j, \qquad U_j = \begin{cases} 0 & \text{if } c_j \text{ is satisfied} \\ 1 & \text{if } c_j \text{ is unsatisfied} \end{cases} \tag{B.7}$$

The variable U_j associated with each clause can be expressed as a product of linear functions of its member variables as shown in equation B.8. When the expressions of all U_j's are expanded and the resultant algebraic expression reduced, it results in the PUBO formulation (see Equation B.9). However, the number of monomials M' is not necessarily the same as the number of clauses M. Consequently, the PUBO's bipartite graph described by the edge set E' is also different than that of the original CNF. The PUBO obtained in Equation B.9 can then be translated to PUSO/Ising model using variable substitution discussed in Supplementary Note 2.

$$U_j = \prod_{\substack{i \ \forall\{l_{2i-1},c_j\}\in E \\ \text{or } \{l_{2i},c_j\}\in E}}^{N} g(x_i, c_j), \qquad g(x_i, c_j) = \begin{cases} x_i & \text{if } \{\bar{x}_i, c_j\} \in E \\ 1 - x_i & \text{if } \{x_i, c_j\} \in E \end{cases} \tag{B.8}$$

$$H = \sum_{j=1}^{M} w_j \prod_{\substack{i \ \forall\{l_{2i-1},c_j\}\in E \\ \text{or } \{l_{2i},c_j\}\in E}}^{N} g(x_i, c_j) = \sum_{j=1}^{M'} a_j \prod_{i \ \forall\{x_i,t_j\}\in E'}^{N} x_i = \text{PUBO}(\hat{x}) \tag{B.9}$$

The clause-specific term U_j can be substituted with the Kronecker-delta function as in equation B.10. U_j equates to one when the sum of its member variables is zero (clause

is unsatisfied) and equates to zero when the sum of its member variables is non-zero (clause is satisfied).

$$U_j = \delta \left[\sum_{i \ \forall \{l_i, c_j\} \in E} l_i \right] \tag{B.10}$$

$$\begin{aligned} \Delta H_{x_f} &= \{H(x_1, x_2, \ldots, x_f = 1, \ldots, x_N) - H(x_1, x_2, \ldots, x_f = 0, \ldots, x_N)\} \Delta x_f \\ &= \left\{ \sum_{j=1}^{M} w_j \, \delta \left[\sum_{i \ \forall \{l_i, c_j\} \in E} l_i \;\middle|\; l_{2f-1} = 1, \ l_{2f} = 0 \right] \right. \\ &\left. - \sum_{j=1}^{M} w_j \, \delta \left[\sum_{i \ \forall \{l_i, c_j\} \in E} l_i \;\middle|\; l_{2f-1} = 0, \ l_{2f} = 1 \right] \right\} \Delta x_f \end{aligned} \tag{B.11}$$

The pseudo-derivative of the original CNF objective function (in Equation B.7) with this new substitution w.r.t. variable x_f is shown in Equation B.11. This shows that the objective function's gradients can be computed by retaining the original clause-literal structure (implicitly encoded in the edge set E) and evaluating clause specific terms U_j. It would be equivalent to computing the gradients of the PUBO function (in Equation B.9), that is represented by a different edge set E', where monomial-specific terms (t_j) would need to be computed. As already outlined in Supplementary Note 2, this further indicates that the gradients computed in CNF would also be equivalent to those in the PUSO formulation of the same problem.

Take for example the CNF considered in equation B.12, the objective function can then be expanded as in equation B.13 & B.14:

$$\text{CNF} = (\bar{x}_1 \vee x_2 \vee \bar{x}_3) \ \wedge \ (\bar{x}_2 \vee x_3 \vee x_4) \ \wedge \ (x_3 \vee x_1 \vee \bar{x}_4) \tag{B.12}$$

$$H = x_1(1-x_2)x_3 + x_2(1-x_3)(1-x_4) + (1-x_3)(1-x_1)x_4 \tag{B.13}$$

$$\begin{aligned} H = {} & x_1x_3 - x_1x_2x_3 + x_2 - x_2x_3 - x_2x_4 + x_2x_3x_4 \\ & + x_4 - x_3x_4 - x_1x_4 + x_1x_3x_4 \end{aligned} \tag{B.14}$$

Note how three clauses result in 10 terms in the resulting higher order polynomial. Such a conversion can potentially lead to $O(2^k)$ many terms in the polynomial arising out of each clause (with clause size k) in the worst case.

B.3 Relation between XNF and the Ising Model

Many Constrained Satisfaction Problems (CSPs) in coding theory, cryptographic analysis, result in objective functions with XOR constraints instead of OR constraints like in CNFs. This results in the Exclusive Normal Form (XNF). A XNF formula comprises conjunction of clauses (from a set $C = \{c_1, c_2, \ldots, c_M\}$ of M clauses) where a clause is a XOR of variables (from a set $X = \{x_1, x_2, \ldots, x_N\}$) (see Equation B.14). It can be represented by a bipartite graph $G = (X, C, E)$, where E is the set of edges defined by $\{x_i, c_j\}$ such that variable x_i is a member of clause c_j. In its most general form, each clause is also associated with a positive valued weight w_j. The goal is to find the variable assignment that satisfies all clauses (in decision-type problems) or maximizes the sum of weights of satisfied clauses (in optimization-type problems). The underlying objective natively encoded in the XNF representation can then be formulated algebraically as shown in Equation B.16.

$$\text{XNF} = \bigwedge_{j=1}^{M} c_j, \qquad c_j = \bigoplus_{i \ \forall\{x_i,c_j\}\in E}^{N} x_i, \qquad \text{and has weight } w_j, \text{ where } x_i \in \{0,1\} \quad \text{(B.15)}$$

$$H = \sum_{j=1}^{M} w_j U_j, \qquad U_j = \begin{cases} -1 & \text{if } c_j \text{ is satisfied} \\ 1 & \text{if } c_j \text{ is unsatisfied} \end{cases} \quad \text{(B.16)}$$

Here, the clause-specific term U_j is defined differently than in the CNF case (+1/-1 for unsatisfied/satisfied clauses rather than 1/0). This is because U_j can be easily expressed as (-1) raised to the power of the XOR clause itself. Since XOR operation over binary variables computes the parity function, it can be further re-written as shown in Equation B.17.

$$\begin{aligned} U_j &= (-1)^{\bigoplus_i^N \forall\{x_i,c_j\}\in E \, x_i} \\ &= (-1)^{\sum_i^N \forall\{x_i,c_j\}\in E \, x_i} \\ &= \prod_{i \ \forall\{x_i,c_j\}\in E}^{N} (-1)^{x_i} \end{aligned} \quad \text{(B.17)}$$

Now if the original binary variables x_i are translated to the bipolar variables s_i, using the relation shown in Equation B.18, then U_j can be re-written as the product of these bipolar variables (Equation B.19). This results in the PUSO formulation over these bipolar variables (see Equation B.20).

$$x_i = \frac{1 - s_i}{2}, \qquad \text{where } s_i \in \{-1,1\} \ \Rightarrow \ s_i = (-1)^{x_i} \quad \text{(B.18)}$$

$$U_j = \prod_{i \ \forall\{x_i,c_j\}\in E}^{N} s_i \quad \text{(B.19)}$$

$$\text{PUSO} = \sum_{j=1}^{M} w_j \prod_{i \ \forall\{x_i,c_j\}\in E}^{N} s_i \tag{B.20}$$

It is easy to verify that the pseudo gradients of PUSO w.r.t. any s_f is exactly equal to the pseudo gradients of H w.r.t. x_f. This implies optimizations done on XNF formulation is equivalent to those done on the Ising formulation. The topology and size of the bipartite graph representing the PUSO is also the same as that of XNF. As a result, the number of variables, high-order interaction terms (XOR clauses in XNF and monomials in PUSO) and the edge set E, remain the same in both formulations. This is unlike the case when a CNF is translated to PUBO.

Appendix C

ReRAM-based NeoHebbain Synapse for fast online learning

C.1 Background

Deep learning has been breaking records in both image recognition and natural language processing [168, 169]. However, to make linear improvements in performance one needs exponentially many computational resources and hence energy consumption and money [170]. For instance, OpenAI's 175-billion-parameter GPT-3 reportedly consumed about 1,287 MWh during training, while DeepMind's 280-billion-parameter model required roughly 1,066 MWh, equivalent to the annual electricity consumption of around 100 average U.S. households [171]. In contrast, the human brain operates with far greater density and energy efficiency, making brain-inspired or neuromorphic computing an attractive alternative that could help mitigate the exponentially rising computational costs. Four key features of such computing paradigms include, (i) asynchronous spike-based communication achieved with address-event representation [172]; (ii) massively parallel computation, that can be achieved using analog in-memory computing with crossbar ar-

rays of memristive devices; (iii) Recurrent connectivity; and (iv) online local learning. Even though all these features have already been demonstrated in theory and hardware individually by several SOTA neuromorphic chips by academia and industry, it has been mostly difficult to implement the last two features together in the same substrate. This is mainly because up until recently, learning in recurrent networks has been done by the Backpropagation through time (BPTT) algorithm [173]. The algorithm applies the backpropagation technique on the unrolled recurrent neural network U ($\sim$ up to 1000) times for U-step-long input sequences, and hence enormous energy and time resources are spent for training such effectively U-layer-deep network.

Recently suggested *e-Prop* learning algorithm addresses this inefficiency by taking inspiration from biologically-plausible three-factor rules [174]. *E-prop* synaptic weights are adjusted using locally stored information from pre- and post-synaptic neurons, avoiding resource-taxing network unrolling. Hence, such an algorithm is naturally suited for (in-memory, analog) neuromorphic hardware that could drastically reduce training time and energy. Furthermore, it enables online mode learning while approaching BPTT accuracy for supervised, unsupervised, and reinforcement learning tasks [174]. This work presents ReRAM based "neoHebbian" synapses, that are essential for an efficient e-Prop algorithm hardware implementation [75, 175]. Such synapses feature two state variables – a neuron coupling weight and an "eligibility trace" for updating the weight. To implement these features, the first proposed design utilizes a pair of ReRAM devices, each with unique conductance range, and connects them in series to form a voltage divider for controlling a weight update, and in parallel for the remaining operations. The second, a denser design, utilizes a heater to modulate ReRAM weight update. Benchmark simulation results have shown that the proposed devices could dramatically reduce training time and energy consumption for temporal data modeling applications.

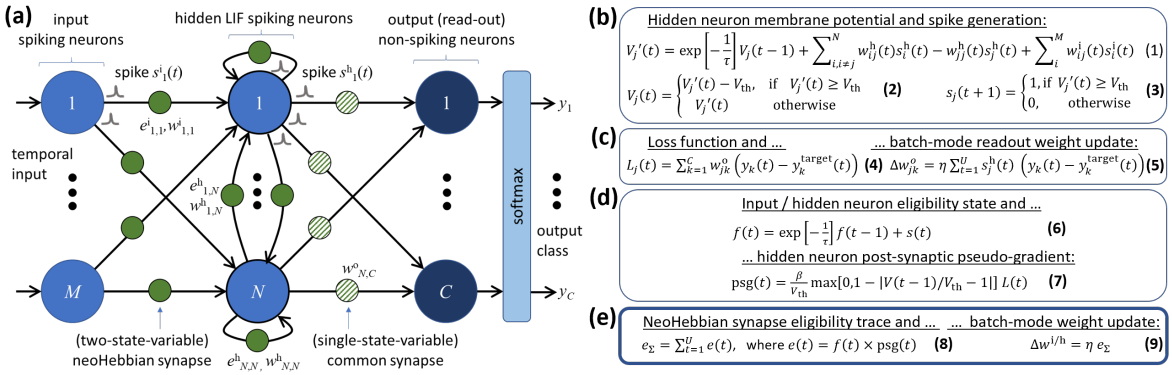


Figure C.1: Hardware friendly e-Prop learning algorithm: (a) Key elements of the network and (b-e) equations governing its operation. Synaptic eligibility traces $e(t)$ and neuron states (membrane potentials $V(t)$, eligibilities $f(t)$, and pseudo-gradients $\mathrm{psg}(t)$) are reset to 0 before applying a new batch of training data (i.e., at $t = 0$). Readout (w^o), and input (w^i) / recurrent (w^h) neoHebbian weights are only updated at the end of the batch. Eq. 7 is different (simpler to implement) compared to [176] without loss of accuracy. Membrane threshold $V_{\mathrm{th}} = 0.615$, leakage constant $\tau = 200$, learning rates $\eta = 0.1 \div 1$, $\beta = 0.3$.

C.2 *E-Prop* Learning Algorithm

E-Prop learning algorithm is based on a fully-connected recurrent spiking neural network (RSNN) that consists of a hidden layer of leaky-integrate-and-fire (LIF) spiking neurons, connected recurrently and to input neurons with neoHebbian synapses, and a readout layer of non-spiking neurons coupled to the hidden layer with common (Hebbian) synapses (Fig. C.1a). (For simplicity, neurons with spike frequency adaptation are not considered in this work). A neoHebbian synapse displays short-term hidden dynamics and long-term plasticity that are represented with synaptic "eligibility trace" e and "coupling weight" w state variables, respectively. e is modulated via the common (two-factor) Hebbian rule. It acts as an additional gating signal, that together with pre- and post-synaptic activities, affects long-term plasticity, i.e., w.

Fig. C.1b-e shows the considered equations for updating network states at discrete

time steps t. Spike generation, neuron membrane potential, and readout layer updates are performed using typical rules (Eqs. 1–5). Eqs. 6–9 are specific to hardware-friendly *e-Prop* [176]. Specifically, the training is performed in batches, with data in a single batch (a dataframe) applied sequentially over the T steps. The training sequence involves computing eligibility state (Eq. 6) and pseudo-gradients (Eq. 7) at pre- and post-synaptic neurons, the accumulation of eligibility contributions over the timeframe at the synapse level (Eq. 8), and updating coupling weight at the end of the timeframe according to (Eq. 9). The readout layer weights are separately trained using the batch-mode stochastic gradient descent approach (Eq. 5). The focus of this work is on hardware implementation of Eqs. 8 and 9 neoHebbian functionality with ReRAM devices.

C.3 NeoHebbian Synapse

Metal oxide memristors were fabricated similarly to previously developed recipe [65], e.g., using a combination of etch-down process and UV lithography for patterning, DC-mode magnetron sputtering for electrode deposition, and using thermal annealing in forming gas for adjusting non-stoichiometry. Figure. C.2 show measured switching characteristics that are important for the proposed synapse implementation and fitted models used in application modeling studies.

Fig. C.3a describes the main idea of proposed voltage-divider based synapse. It comprises of two memristors (M_e and M_w) whose conductances G_e and G_w represent e and w respectively. During a dataframe, switch 'S' is open which decouples M_e and M_w. Memristor M_w acts as a coupling weight and transmits weighted current spikes from pre-neuron i to post-neuron j. Simultaneously, appropriate voltage pulses are applied to M_e (Fig. C.3d) that results in conductance change proportional to the product of $f_i(t)$ and $\text{psg}_j(t)$, enabled by the exponential dependence of ΔG_e on applied voltage, see Fig. C.2a.

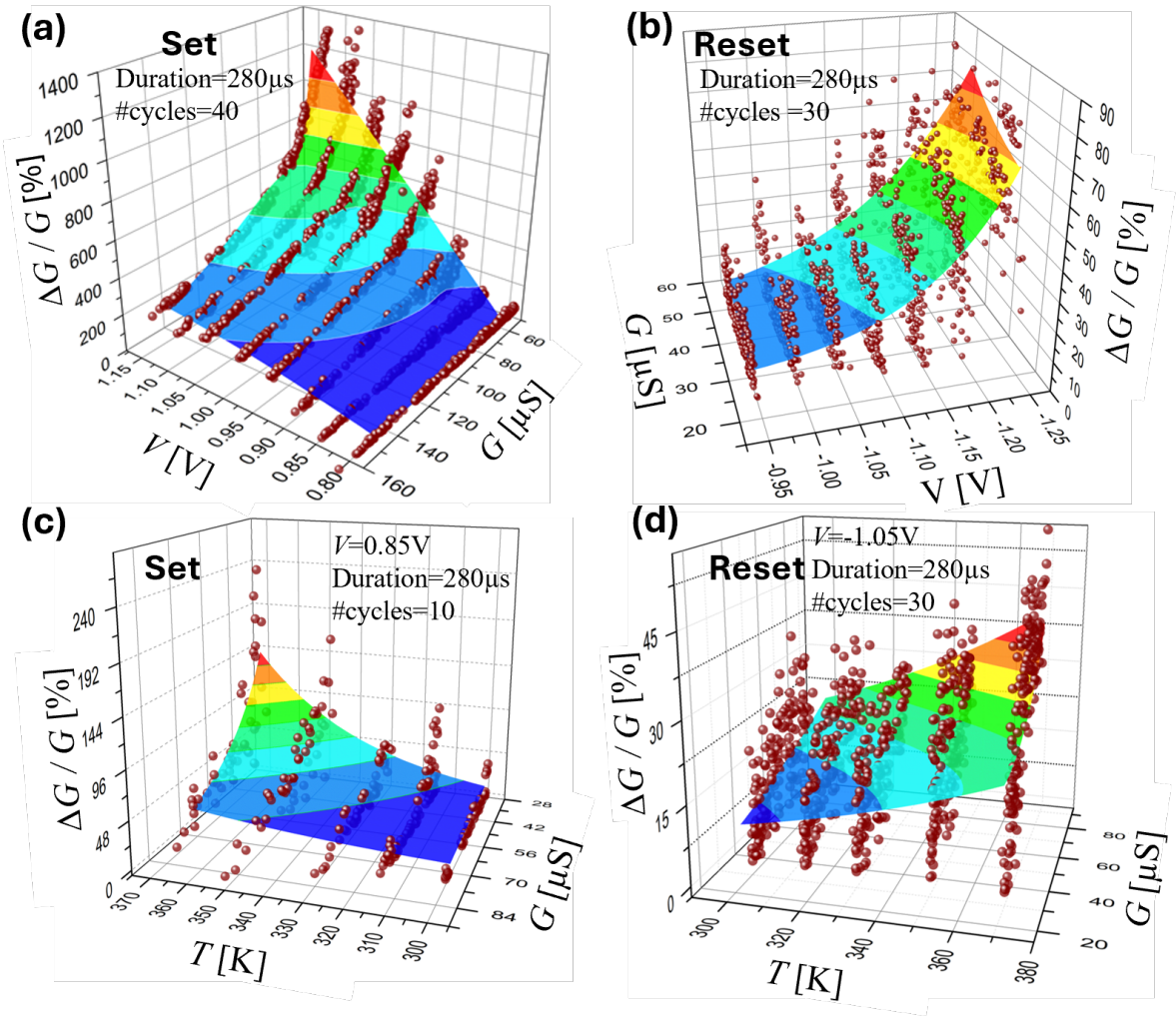


Figure C.2: (a-d) Characterization of switching dynamics, shown as normalized percentage conductance change, as a function of the initial conductance and (a, b) applied voltage amplitude, or (c,d) ambient temperature. Panels (a,b) are relevant to voltage–divider synapse design, while (c,d) for the one with temperature-encoded eligibility traces. All panels show experimental data obtained from several devices by repeating experiment the specified #cycles for each set of inputs. Conductances are measured at non-disturbing 0.1V. The color-mapped surfaced in (a-d) are fits utilized for benchmark application modeling.

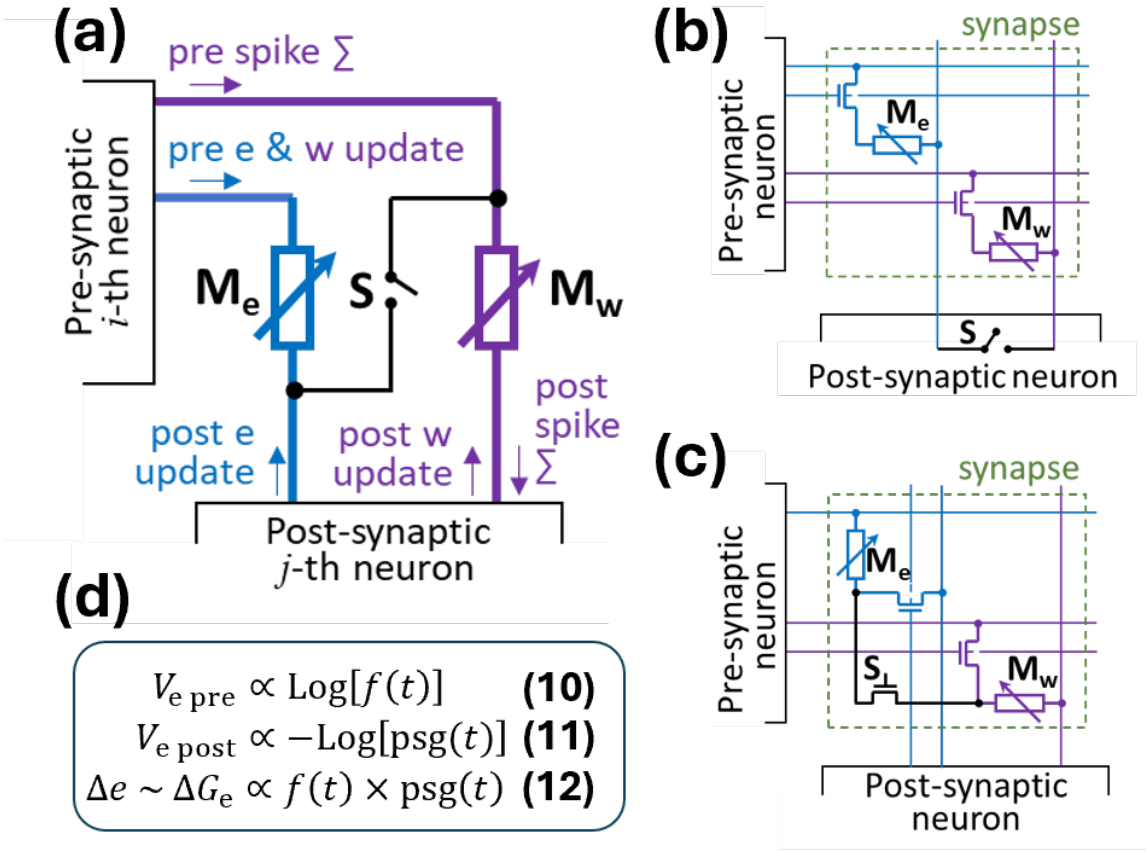


Figure C.3: Voltage-divider neoHebbian synapse: (a) Main idea, (b) 2T2M and (c) 3T2M implementations, and (d) equations for the eligibility trace update. For simplicity, only one synapse is shown, while many such synapses are connected via crossbar to pre-/post-synaptic neurons in an actual circuit. This work focuses on the 2T2M implementation.

This happens at every timestep thereby implementing Eq. 8. At the end of dataframe, 'S' is closed to form a voltage-divider circuit and a programming pulse is applied across the series resulting in a voltage drop across M_w (that impacts ΔG_w) proportional to G_e to approximate Eq. 9.

However, the dependence of ΔG_e on G_w is undesirable. An ad-hoc modeling (with experimentally matched parameters) shows that for the same range of G_e, decreasing G_w leads to higher M_w voltage and lower M_e voltage (Fig. C.4a). Therefore, cross-

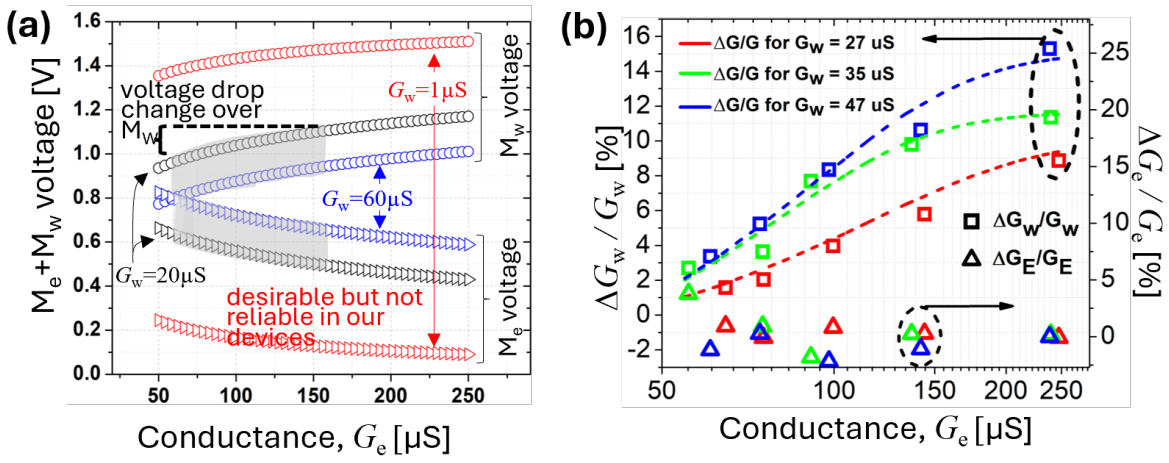


Figure C.4: Conductance range optimization in voltage-divider design: (a) Simulated voltage drop across serially connected M_W & M_E with applied constant $-1.6\,\mathrm{V}$. Shaded regions show selected G_E and G_W ranges. (b) Measured normalized percentage conductance change in serially connected M_W & M_E, when $-1.6\,\mathrm{V}$, $280\,\mu\mathrm{s}$ pulses are applied, for different G_E (denoting different accumulated eligibility trace values) and at three different initial G_W. Points and lines represent experiment and simulation, respectively.

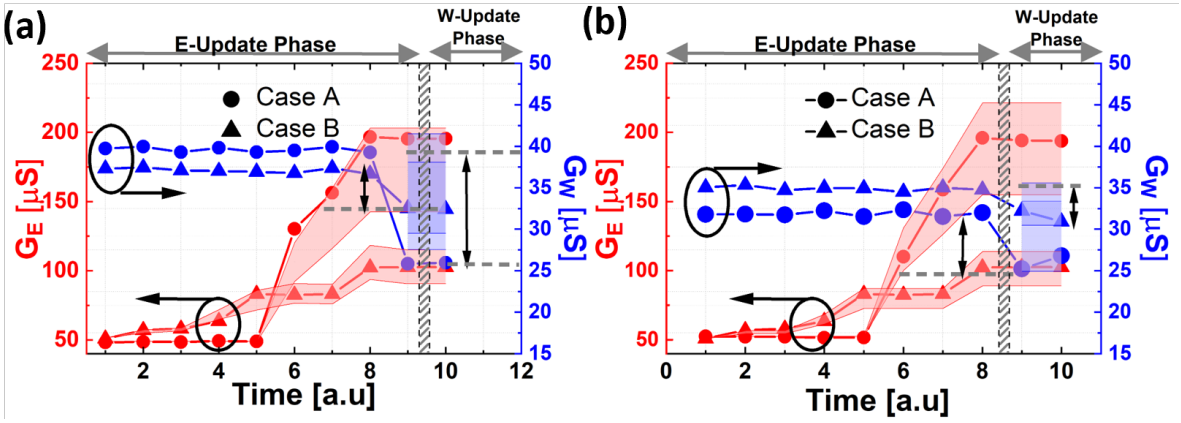
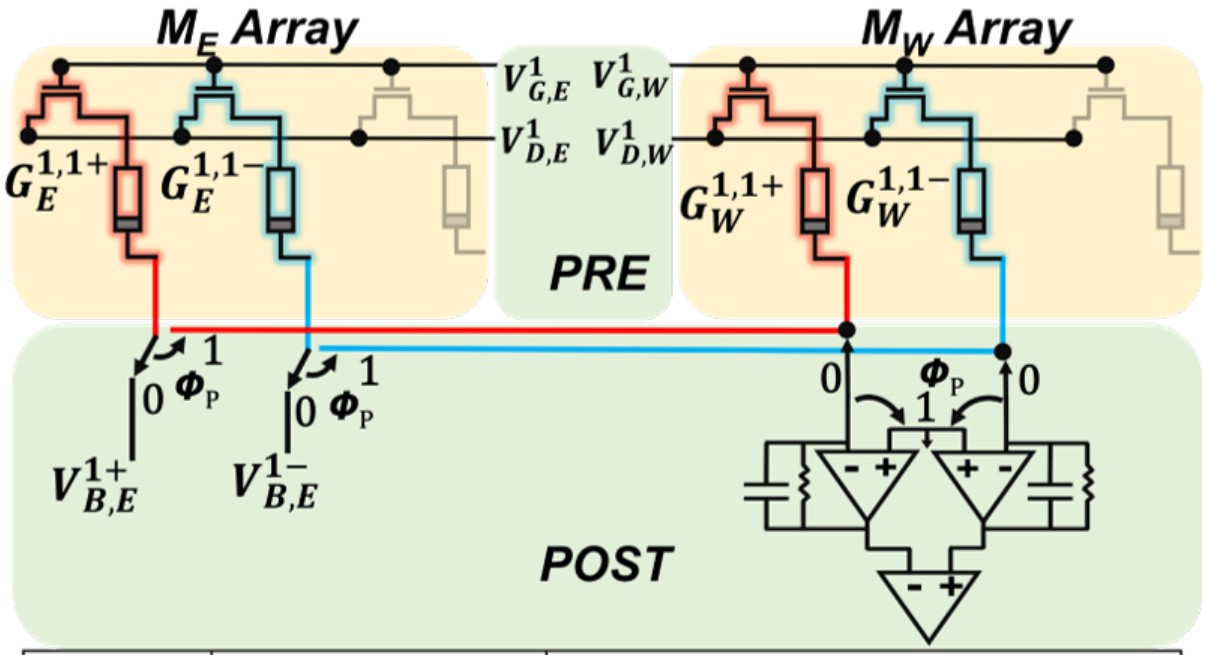


Figure C.5: Experimental demonstration of voltage-divider neoHebbian synapse: (a,b) Evolution of G_E and G_W for two initial conductance ranges of G_W. In the E-update phase, G_W (representative of weight) remains constant and G_E (eligibility) evolves with applied SET pulses to device M_E in two ways: rapid and gradual evolution shown by Case A and Case B, respectively. In the W-update phase, devices M_E and M_W are connected serially and $-1.6\,\mathrm{V}$, $280\,\mu\mathrm{s}$ pulse is applied across the two to change G_W proportional to accumulated net G_E. For a comparison, shaded regions show simulation results obtained using fitted model with added C2C variations.



	$\Phi_P = 0$	$\Phi_P = 1$
$V^i_{G,E}$ / $V^i_{G,W}$	V_{ON} / $z_i(t)$	V_{ON} for selected I; 0V for rest
$V^i_{D,E}$ / $V^i_{D,W}$	$\log(f_i(t))$ / V_{read}	0 V / V_{PROG}
$V_{B,E}^{j\mp}$	$-\log\lvert psg'_j(t)\rvert$ or 0V	0V

Figure C.6: Crossbar implementation of voltage-divider synapse design with differential eligibility and weight. Red & Blue lines show the connectivity of just one pair of positive and negative 2T2M synapses, respectively, when the first row is selected for weight update. Table shows voltage biases during different phases.

dependence can be mitigated by engineering different operating conductance regimes in M_e and M_w. This idea is validated by applying a reset pulse to serially connected memristors with $\sim 4\times$ separation between mean of their dynamic ranges (Fig. C.4b). The results show the desired monotonic dependence of $\Delta G_w/G_w$ on G_e and negligible changes in G_e. Furthermore, the complete synapse functionality (Eq. 8 & 9) is demonstrated by observing proportional weight-update ΔG_w to G_e at the end of toy-model dataframe across the studied G_w ranges (Fig. C.5).

Fig. C.3b, C.3c shows two (2T2M & 3T2M) designs of voltage-divider synapses. 2T2M

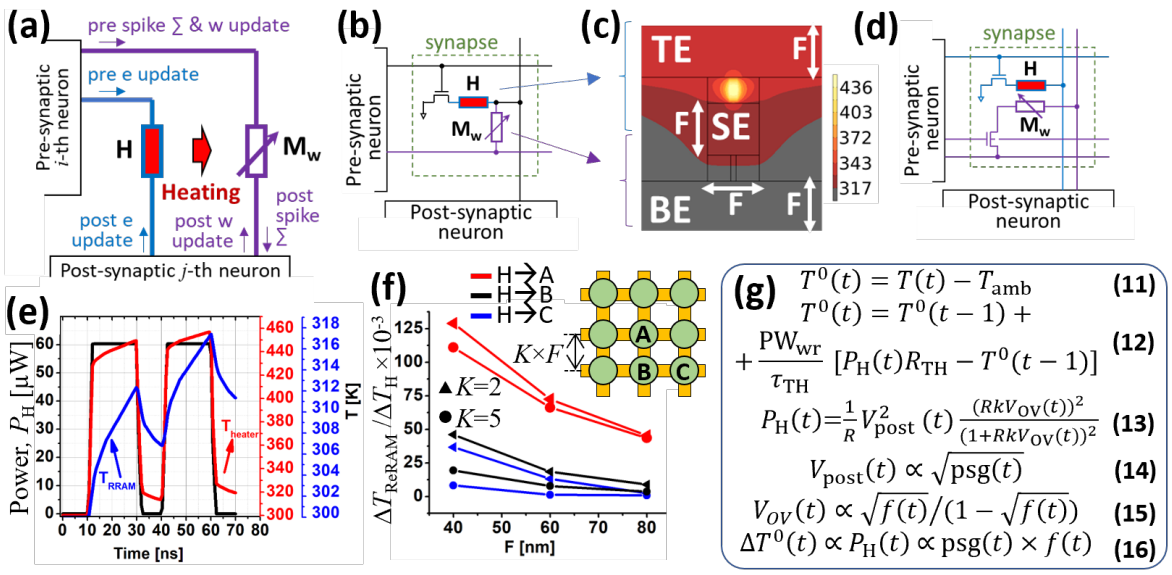
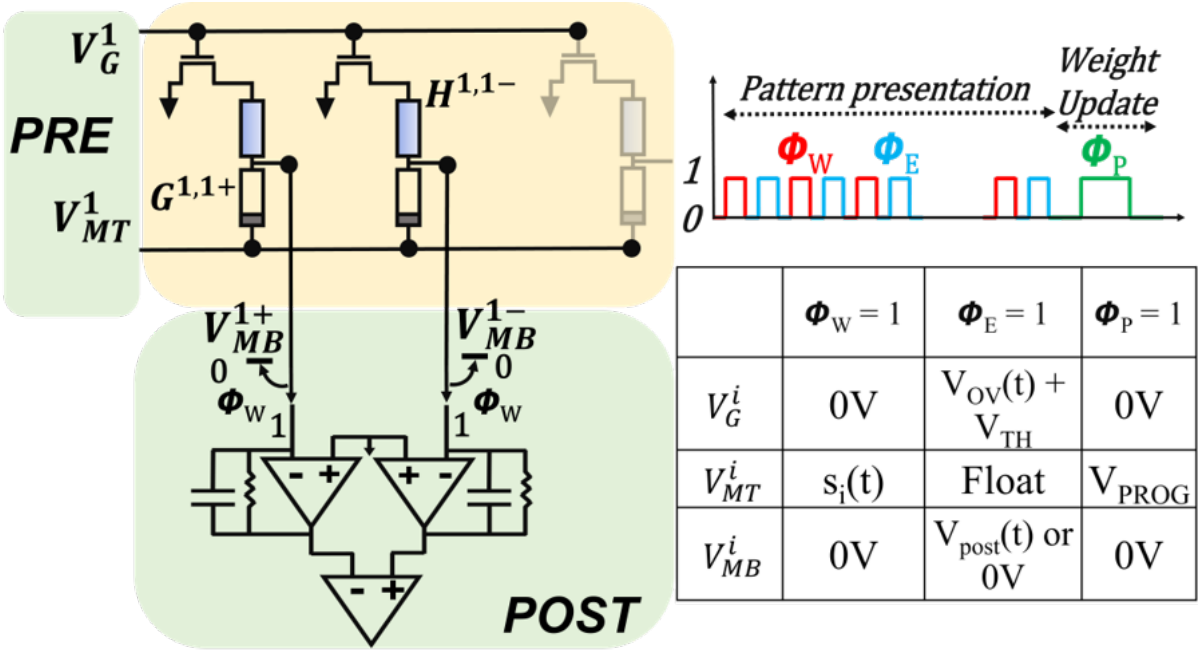


Figure C.7: NeoHebbian synapse with temperature-encoded eligibility: (a) The main idea, (b) 1T1H1M design and (c) its modeled geometry with overlapped temperature contours after 60 ns for $F = 60$ nm, (d) 2T1H1M design, (e) heater and ReRAM temperature evolution depicting the Φ_E phase, (f) Thermal cross-talk dependence on minimum feature size (F) and crossbar pitch (K) modeled at $t = 20$ ns, and (g) equations related to synapse in (b) during e-update where T_{amb}: ambient temperature, R_{TH}: heater thermal resistance, τ_{TH}: thermal time constant, PW: heating pulse-width, V_{post}: post-neuron signal, V_{OV}: overdrive voltage of transistor, R: heater electrical resistance, $k = \mu C_{\text{OX}} W/L$ of transistor. This work focuses on 1T1H1M design.

is denser but at the cost of moving the switch "S" to the neuron side and applying row-by-row w updates (vs. fully parallel updates in 3T2M). Fig. C.6 further explains this by showing an array-level implementation with differential 2T2M synapses allowing for both negative and positive e and w values. Two separate crossbars (for M_e and M_w) share peripheral circuits for pre/post-synaptic neurons. During a dataframe ($\Phi_p = 0$), current integration and e-update take place simultaneously via M_w and M_e arrays respectively. w-update takes place row-wise at the end of dataframe ($\Phi_p = 1$).

Fig. C.7a shows the main idea of second synapse design, in which e is encoded in the



	$\Phi_W = 1$	$\Phi_E = 1$	$\Phi_P = 1$
V_G^i	0V	$V_{OV}(t) + V_{TH}$	0V
V_{MT}^i	$s_i(t)$	Float	V_{PROG}
V_{MB}^i	0V	$V_{post}(t)$ or 0V	0V

Figure C.8: Crossbar implementation based on temperature synapse design. Table shows voltage biases during different phases. Inset shows the phase diagram.

form of temperature rise at synapse due to Joule heating at the heater. The temperature rise is proportional to the dissipated power, offering several options for hardware implementation of Eq. 7 (Fig. C.7b, C.7d). In one such implementation, the use of additional transistor and appropriate voltage signals (Fig. C.7b) ensure that the power dissipated across the heater and, consequently, the temperature rise at each time step t is proportional to $f(t) \times \mathrm{psg}(t)$. Due to thermal coupling, memristor local temperature follows the same proportionality. At the end of dataframe, a fixed amplitude programming pulse induces ΔG_w, proportional to accumulated local temperature rise at the synapse, as shown in Fig. C.2c, C.2d. Nonidealities specific to temperature-based synapses are local temperature decay in the absence of input and thermal crosstalk between the neighbouring synapses, both potentially limiting integration density. To investigate the latter issue, Fig. C.7f shows crosstalk dependence on minimum feature size and crossbar full pitch.

A similar to Fig. C.6, differential-synapse crossbar array implementation is suitable for

temperature-based design with dedicated select transistor (Fig. C.7d) for weight update. Denser option (Fig. C.7b) requires time-multiplexing between current integration ($\Phi_w = 1$) and e-update ($\Phi_E = 1$) phases (Fig. C.8).

C.4 Application Modeling

The accuracy and robustness of RSNN network with proposed NeoHebbian synapses is evaluated on TIMIT phoneme classification task (Fig. C.9) – see [174] for more details on the benchmark. For simplicity, one-directional SET e-update and RESET w-updates are utilized for voltage-divider synapses. On the other hand, temperature-based synapse is modeled with more practical approach of applying alternating SET / RESET w-updates.

The proposed approaches perform on par with ideal software-based model utilizing floating-point precision for the baseline 200-hidden neuron network (Fig. C.9b). The accuracy improves with network size, outperforming several previously proposed architectures, which suggests that nonideal behavior in the proposed synapses, e.g., super-linear w updates (Fig. C.2a-d), can be tolerated by increasing network size (Fig. C.9c). Figs. C.9d, C.9e elaborate on the accuracy sensitivity to the main sources of nonidealities. The modeled network is robust to cycle-to-cycle variations (Fig. C.9e), most likely due to the intrinsic redundancy in the baseline network and variation-aware training. The results show that network accuracy drop is negligible with up to 33% temperature cooling after 20 timesteps and reasonable thermal coupling (Fig. C.9d). The spatial synapse locations for the latter estimate are taken from $N \times M$ and $M \times M$ crossbar implementation for the input and recurrent layers, respectively (Fig. C.1a).

The energy and latency are crudely estimated assuming 10 ns spike integration, and $V_w = 3.2\,\mathrm{V}$ for 2.1 V write pulses for two synapse designs, projected from experimental data to achieve $\mathrm{PW}_w = 10\,\mathrm{ns}$ write time (i.e., $\approx V_w^2 G_w \mathrm{PW}_w \sim$ pJ-scale write en-

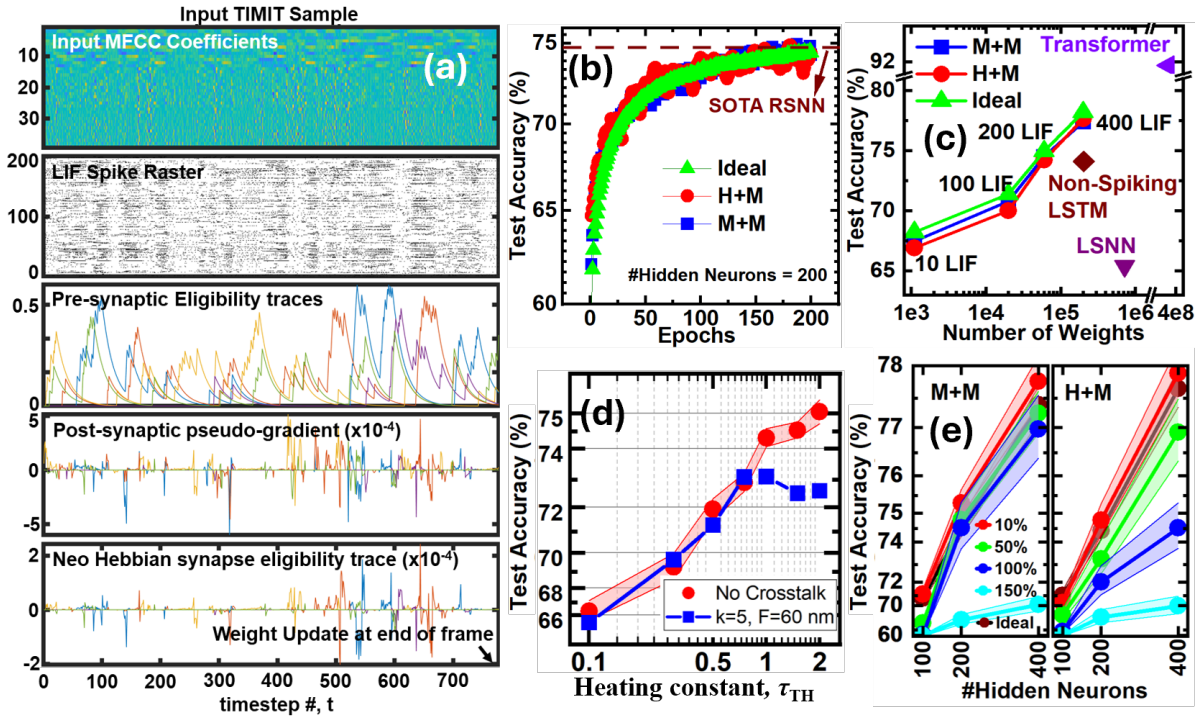
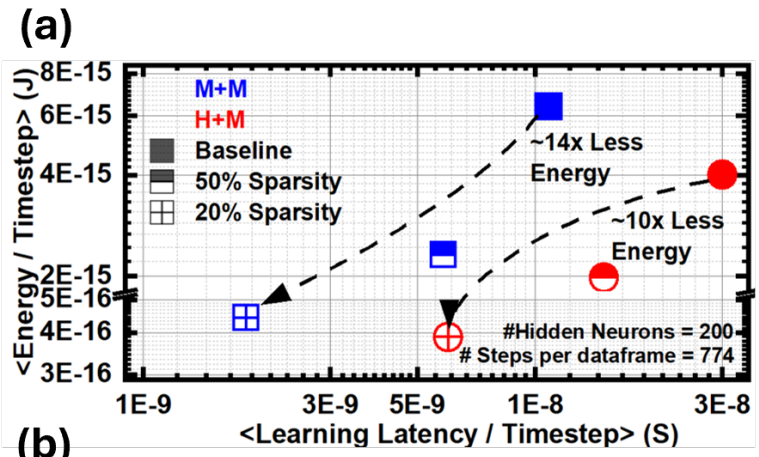


Figure C.9: Benchmark modeling results: (a) A sample from the TIMIT data set, demonstrating the classification of phonemes. (b,c) Accuracy across studied cases as a function of (b) training time and (c) the utilized hidden neurons, assuming no variations. (d,e) Accuracy sensitivity to (d) the thermal constant and crosstalk for temperature-based synapses, and (e) as a function of #hidden neurons when adding C2C variations, shown as percentages in coefficient of variations (CV) to ReRAM conductance change, for both designs. 200 epochs of training were used for panels c–e. C2C variability is modeled by adding normally-distributed noise scaled proportionally to the ideal ΔG values. Comparison done with LSNN [174], transformer [177].

ergy) for ReRAM devices, justified by previous work [65, 179]. The results show the temperature-based approach is slower due to the time-division multiplexing operation (Fig. C.10a, C.10b). To decrease energy consumption, sparsification technique with randomly selected skipped e-updates and w-updates was studied. The optimal percentage of skipped updates is $\sim 80\%$ (or $\sim 20\%$ sparsity denoting the proportion of skipped learning steps and % of learning updates omitted per step spatially, in e & w updates) with further sparsity increase balanced by longer training times to reach the same accuracy.



(c)

Approach	Density	Energy per update	Speed	Accuracy	Maturity
Voltage-divider (this work)	+ (1)	+	+(4)	+	+
Temperature (this work)	++(2)	+	+(4)	-	-
PCM	- (3)	- (3)	- (3)	- (5)	++
CMOS	--	--	--	++	+++

Figure C.10: Modeling results summary: (a, b) Energy and delay comparison of the studied approaches, and (c) qualitative comparison with previous work. Average values are shown to account for update sparsity. Note: (1) Limited by transistors; (2) 3D-scalable with 1D1H heater; (3) Only w is PCM-implemented; (4) Comparable when using similar array-level implementations; (5) Low accuracy on sine wave modeling task [178]. Other works include PCM [178] and CMOS [176].

C.5 Summary

We proposed and experimentally verify ReRAM neoHebbian synapses and model representative application using scalable *e-Prop* learning algorithm. Voltage-divider design shows more robust functionality but less dense. Temperature-based design features additional nonidealities, but potentially denser, especially passive version of 1T1H1M in which T is replaced with diode. This and additional reserves for improvement require further investigation to address thermal coupling and decay. Crucially, the proposed synapses are much denser, more energy efficient and faster, compared to earlier work based on CMOS and PCM devices [176, 178] (Fig. C.10c).

Appendix D

Memristor-Based Neuromorphic Readout Circuit for Event Cameras

D.1 Background

In conventional cyber-physical system architectures, sensing and processing components are deliberately separated: sensing operates in the noisy analog domain, while processing takes place in the digital domain, with data conversion bridging the two. However, the exponential growth of sensor networks and the vast volume of redundant raw data they generate [180, 181] have created critical bottlenecks, both in energy consumption and latency, due to the overhead of moving information across this interface [182]. To mitigate this issue, in/near-sensor (INS) computing, where computation is performed directly at or near the sensor, has been proposed [182]. At the same time, bio-inspired approaches to data processing also offer advantages. Mimicking the brain's co-location of memory and computation, as well as its spike-based signaling, has given rise to paradigms such as analog in-memory computing (IMC) and spiking neural networks (SNNs) [183]. Real-world systems leveraging these techniques have demonstrated substantial gains in both energy

efficiency and latency reduction [184, 185]. In parallel, event-based sensing, inspired by the human retina, has led to the development of dynamic vision sensors (DVS), which offer significant advantages over traditional frame-based cameras through more efficient information encoding [186]. Each of these concepts – INS, IMC, and SNN/DVS – has long been recognized for its potential, with significant academic and industrial efforts advancing them across a range of applications [55, 187, 188, 189, 190, 191, 192, 193, 194]. The overarching goal of this work is to demonstrate the advantages of integrating these technologies into a single chip. This chapter presents preliminary results toward that vision.

This paper presents preliminary results on the development of a memristor-based neuromorphic readout circuit for event cameras [76]. The prototype performs classification of images captured by focal plane array (FPA) using a one-layer feedforward spiking neural network (SNN). It comprises a 4×4-pixel image sensor integrated with dynamic vision sensor (DVS) circuitry, 16 input and 2 output leaky integrate-and-fire neurons, and 32 differential-pair synaptic weights. The synapses are implemented using titanium oxide memristors passively integrated into a 64×64 crossbar array. All other components are realized with conventional integrated circuits (ICs), distributed across two custom printed circuit boards (PCBs) and managed via field-programmable gate array (FPGA) boards and a personal computer. As a proof of concept, we emulated FPA operation and experimentally demonstrated the classification of synthetic 4×4 spatiotemporal patterns into two categories using an SNN trained ex-situ with the ReSuMe algorithm. The prototype embodies key principles of near/in-sensor computing, analog in-memory computing, event-based sensing, and neuromorphic processing. It reflects our vision for enhancing the energy efficiency of cyber-physical systems through close integration of sensing and computation to minimize costly analog-to-digital conversions and enable low-level operations with sensed data directly in the native analog domain. Our approach is well-suited

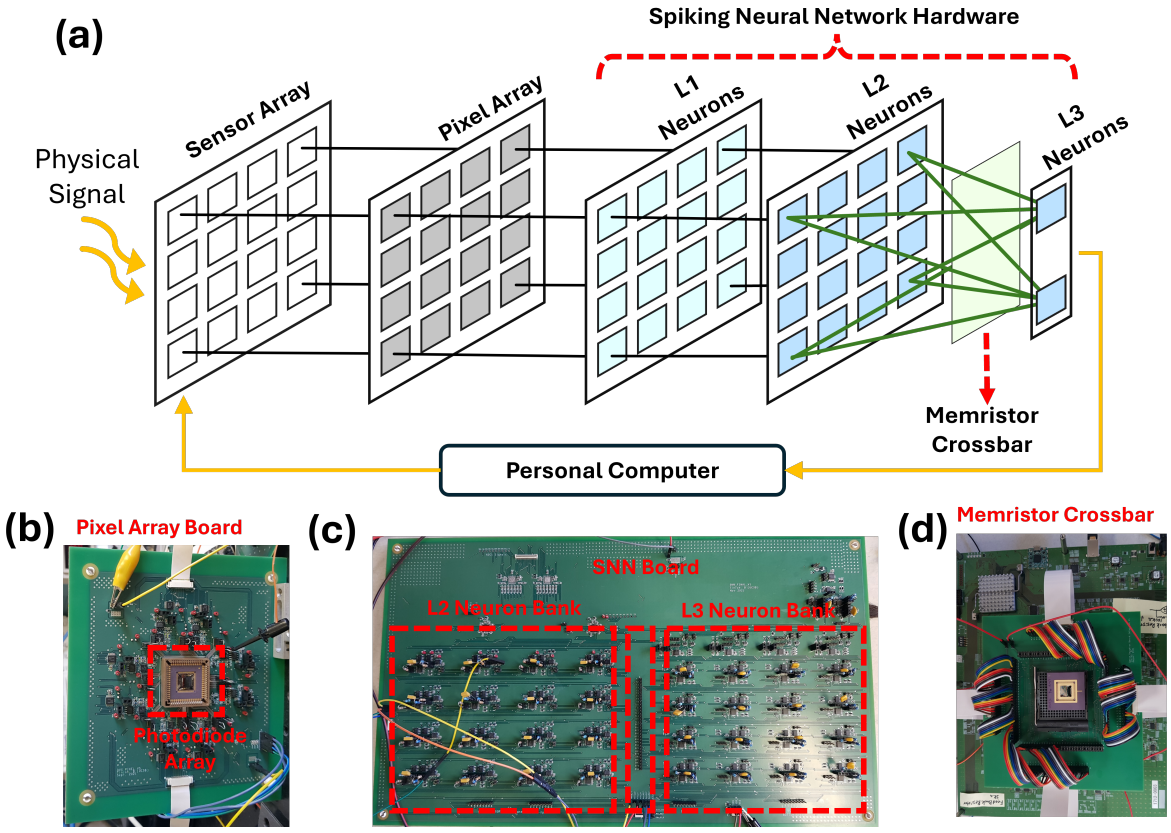


Figure D.1: High-level overview of the prototype: (a) system level diagram of pipeline implemented on our hardware platform, (b) the spiking readout integrated circuit (SROIC) board featuring the dynamic-vision sensor (DVS) circuitry and a socket for the photodiode (PD) array, (c) the spiking neural network (SNN) PCB implementing leaky spike integration (TR) and output (L2) leaky integrate-and-fire (LIF) neurons. The input (L1) LIF neurons are implemented on an FPGA board, (d) a PCB hosting the memristive crossbar circuits. A second FPGA board and a personal computer (PC) are used to control the experiment.

for advanced heterogeneous technology platforms that tightly integrate emerging memory devices with complementary metal-oxide-semiconductor (CMOS) and image-sensor circuitry.

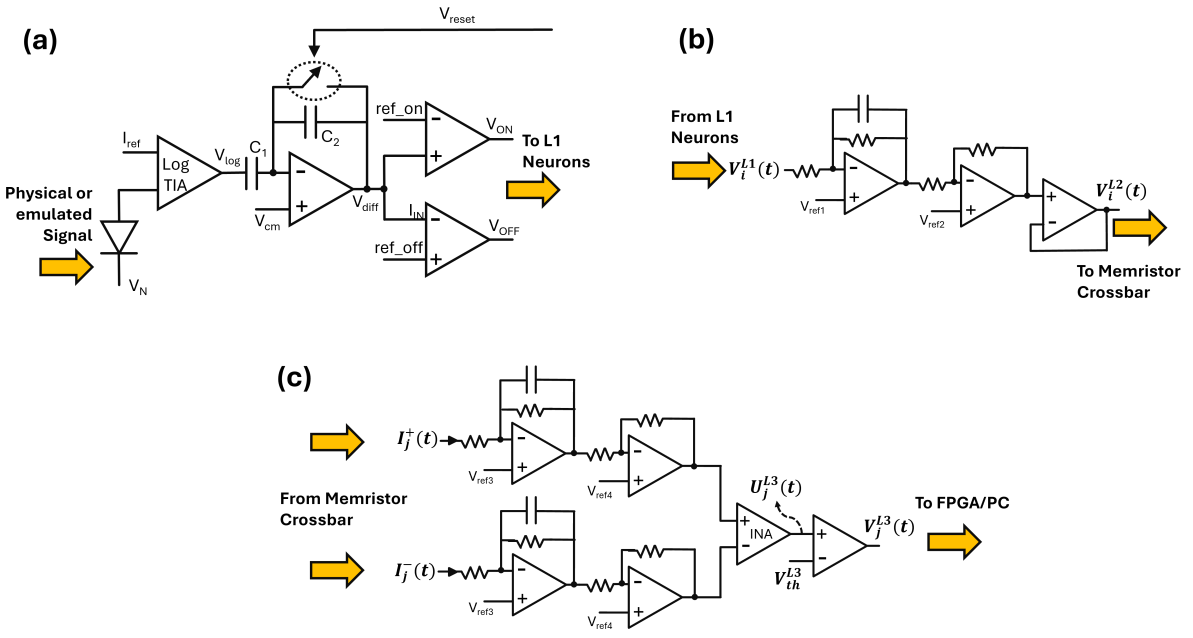


Figure D.2: Circuit schematics for (a) DVS, (b) leaky-spike integrator L2 neuron, and (c) LIF L3 neuron.

D.2 Results

Figure D.1 illustrates the major components of the developed prototype. The spiking readout integrated circuit (SROIC) board connects each pixel of a 4×4-pixel photodiode (PD) array placed in socket to a DVS-inspired pixel circuit (Fig. D.2a). The SROIC board is specifically designed to operate with infrared focal plane arrays (FPAs) [195]. It supports a wide range of input currents, from 1 nA to several milliamperes, and allows control over the photodiodes' responsivity and dark current by biasing their n-side using the SROIC's onboard digital-to-analog converter (DAC). Each pixel's DVS circuit generates a spike upon detecting a change in illuminance (Fig. D.3a). This circuit, designed similarly to the one described in Ref. [186], integrates the PD current using an operational amplifier (opamp)-based integrator. When the accumulated charge reaches the pixel's temporal contrast threshold, which is set by onboard DACs, the output voltage

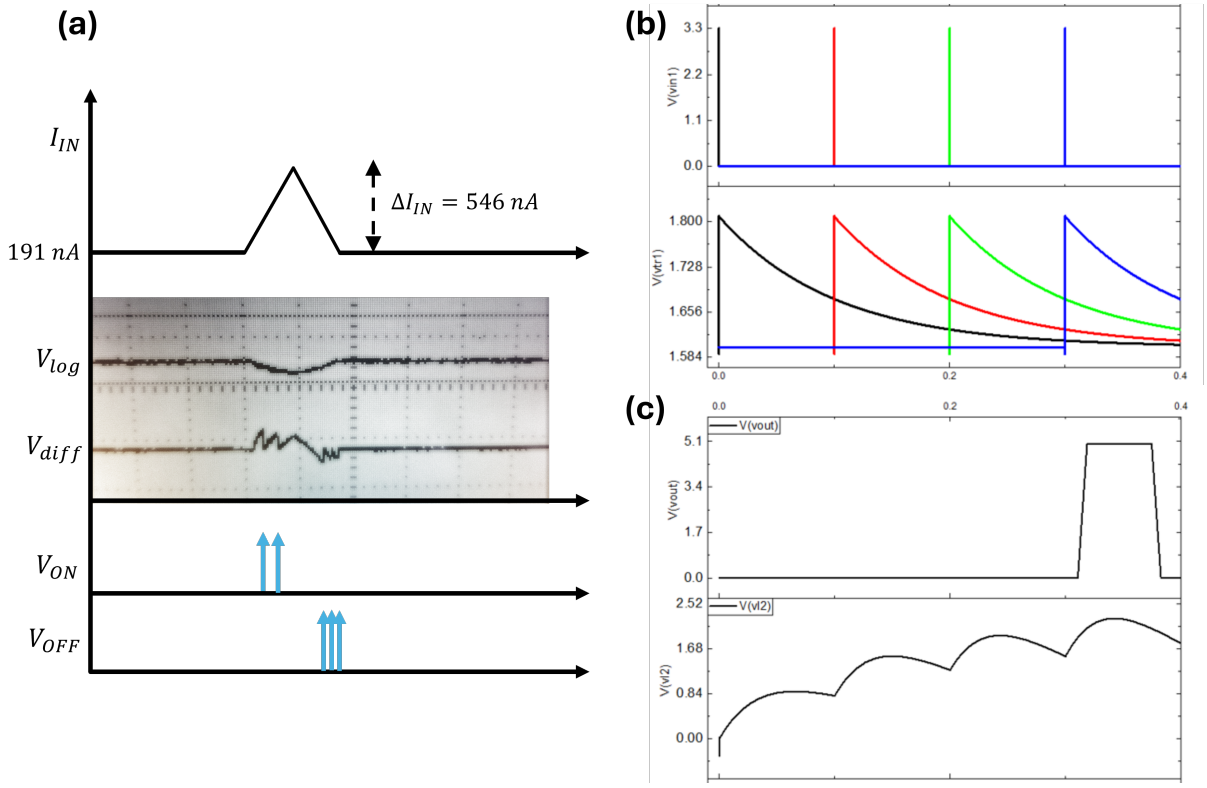


Figure D.3: Response of the DVS pixel circuit to a triangular current pulse provided by a current DAC. Illustration of the (b) L1, L2 neuron outputs and (c) L3 neuron's membrane potential and spiking output.

is reset via a FPGA board.

The digital spike packets generated by the SROIC board are transmitted in a one-to-one manner, to digital hardware LIF input neurons (L1 neurons in Fig. D.1a) implemented on the FPGA's reconfigurable fabric. The input neurons act as low-pass filters, sparsifying the dense spike streams emitted by the pixels. Additionally, they are equipped with a homeostasis mechanism, i.e., they dynamically adjust their threshold voltages at runtime in response to varying input spike densities. The spike outputs from the input neurons are handled asynchronously by the FPGA and transmitted to the L2 neurons on the Spiking Neural Network (SNN) board (Fig. D.2b shows circuit schematic of a single L2 neuron). These L2 neurons, perform leaky integration to convert incoming digital

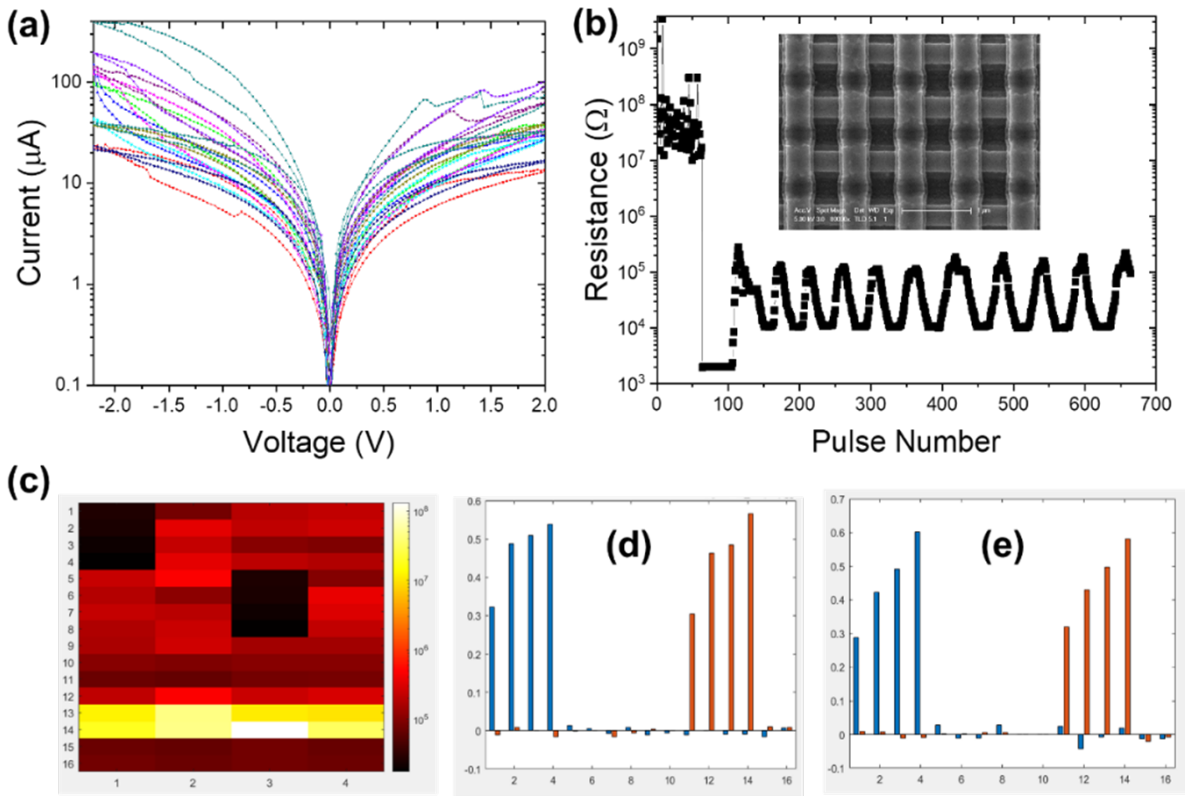


Figure D.4: Memristive devices: (a) Current-voltage switching characteristics for several devices. (b) Analog switching behavior between ON and OFF states, demonstrated by applying write voltage pulses; resistance is measured at a non-disturbing 0.2V bias. The inset shows a top-view scanning electron micrograph of a section of the crossbar array (scale bar: 1 μm). (c–e) Tuning of memristor conductance values: (c) color map and (e) histogram of conductance values (measured at 0.2V) after tuning, corresponding to (d) their target levels.

(square shaped) spikes into analog spikes with exponentially decaying shapes (Fig. D.3b). The resulting signals are then applied to the memristor crossbar circuit (Fig. D.1d). This exponential shaping captures more temporal information from the spike trains, such as inter-spike intervals and frequency. Specifically, the L2 neurons receives voltage spike inputs and integrates them by accumulating charge on a capacitor C_1, which leaks through a resistor R_2 with a defined time constant (Fig. D.2b). Two operational amplifiers are used to achieve the required gain and produce output analog voltages in the 0–200 mV

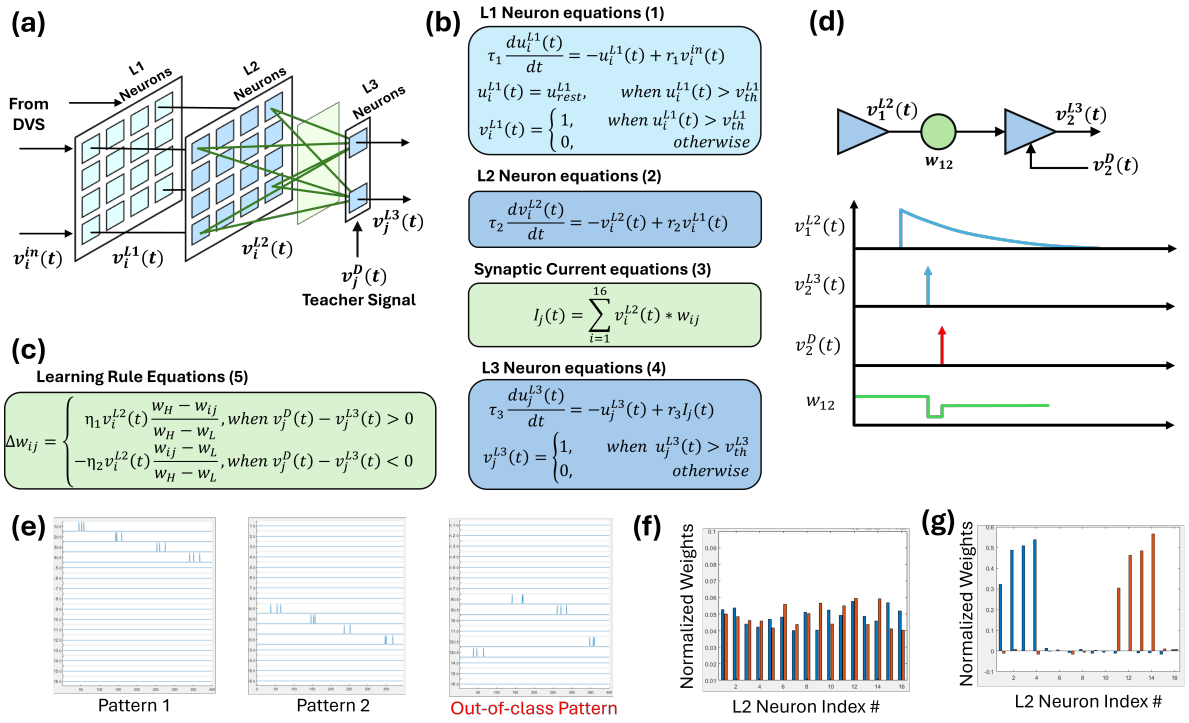


Figure D.5: Pattern classification task: (a) Diagram of the neural network and its associated parameters used in the study, and (b) equations defining neuron behaviour and synaptic current accumulation, (c) modified ReSuMe learning algorithm, (d) illustrative example of the learning rule effecting synaptic depression and potentiation in a single synapse and (e) examples of the two spatiotemporal input patterns (Pattern 1 and Pattern 2) the SNN is trained to recognize and an out-of-class pattern that the SNN should not recognize. (f, g) Visualization of synaptic weights: (f) initial and (g) final weights after training, with blue and red indicating the magnitudes of positive and negative weights in each (numbered) differential pair, respectively.

Inference Setup			Overall Accuracy	In-Class Accuracy	Out-of-Class Accuracy
Input	Weights	Neurons			
Software	Software	Software	99.025	99.95	98.1
Software	Fixed Resistors	Analog/Digital HW	96.7	97.5	95.8
DVS Pixel Array	Fixed Resistors	Analog/Digital HW	96.625	97.1	96.15
DVS Pixel Array	Memristor Crossbar	Analog/Digital HW	95.39	97.39	93.4

Figure D.6: Classification Accuracy Results

range, which is suitable for non-destructive (read) operation with the memristors.

A passively integrated 64×64 memristive crossbar circuits (Fig. D.4), with approximately 400 nm half-pitch dimensions, were fabricated in-house following a process similar to that described in Ref. [65]. The structure consists of Ti/Al/TiN-based top and bottom electrodes, and an Al_2O_3/TiO_{2-x} switching layer. The bottom and top electrodes are approximately 175 nm and 125 nm thick, respectively, while the titanium oxide switching layer is about 30 nm thick. The oxide layers are deposited via reactive sputtering, and a 15 nm aluminum oxide layer is formed using atomic layer deposition. Planarization of the bottom electrode is achieved through a combination of chemical-mechanical polishing and etch-back techniques. All crossbar features are patterned using photolithography and etching. After forming step, the fabricated memristive devices exhibit analog switching behavior with a dynamic range exceeding $10\times$ (Fig. D.4a, D.4b). Memristors implement synaptic weights, with negative weights realized through a differential pair configuration [196].

A 16×4 section of the crossbar array is used in the experiment, with outputs of L2 neurons connected to the 16 top electrodes and 4 bottom electrodes connected to differential inputs of the two L3 neurons. The L3 neurons are implemented on the SNN board (Fig. D.1a,c). The frontend of the L3 neurons is designed to convert virtual ground currents through positive and negative current branches, representing the weighted sums of the corresponding input spikes, into voltages using opamps. The accumulated voltages in the positive and negative branches are then subtracted from each other by an instrumentation amplifier (INA). The output of the INA represents the true accumulated voltage in response to the current flowing through the crossbar, effectively comprising both positive and negative weights. This voltage is compared against a comparator, with the threshold set by onboard DACs. The resulting output, corresponding to the classification result, is transmitted to a FPGA and further passed to the PC.

The SNN was initially trained in software (using high-precision arithmetic) with a supervised learning algorithm to classify synthetic 4×4-pixel spatiotemporal patterns into two categories (Fig. D.5). A modified ReSuMe algorithm [197, 198] was employed, incorporating classical network operation equations (Eqs. 1–4 in Fig. D.5b) along with modified weight update equations (weight normalization factors in Eq. 5 in Fig. D.5c) to account for the finite dynamic range of the memristors. Once the desired weights were determined, the memristors were tuned to the corresponding conductances using a write-verify algorithm [196] (Fig. D.4c-e). The classification results were then measured by emulating the FPA, stimulating each DVS pixel using a current-source DAC.

Figure D.6 summarizes the in-class, out-class, and overall accuracy of the network under different run conditions. The first row represents the original (high-precision) software simulation, while the last row displays the final hardware demonstration accuracy. For comparison, the table also presents results using fixed-value resistors for synaptic weights, both with and without the use of the SROIC and simulation results with discretized voltage spikes.

D.3 Summary

In summary, this paper presents preliminary results on the development of memristor-based neuromorphic readout circuits that integrate in-memory computing, neuromorphic processing, and dynamic vision sensing. The work specifically focuses on the classification of synthetic spatiotemporal images using an emulated focal plane array, demonstrating the robustness of this task to variations in input images. By leveraging energy-saving techniques and the potential for tight, monolithic integration of memristors [199] and image sensors [200] with CMOS circuitry, this approach offers a promising, energy-efficient solution for unifying sensing and computation in future cyber-physical systems.

Bibliography

[1] C. H. Papadimitriou and K. Steiglitz, *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1998.

[2] D. S. Johnson and M. R. Garey, *Computers and intractability: A guide to the theory of NP-completeness*. WH Freeman, 1979.

[3] S. A. Cook, *The complexity of theorem-proving procedures*, in *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, pp. 151–158, ACM, 1971.

[4] R. M. Karp, *Reducibility among combinatorial problems*, in *Complexity of Computer Computations* (R. E. Miller and J. W. Thatcher, eds.), pp. 85–103. Springer, Boston, MA, 1972.

[5] C. H. Papadimitriou, *Computational complexity*, in *Encyclopedia of computer science*, pp. 260–265. 2003.

[6] E. L. Lawler, *The traveling salesman problem: a guided tour of combinatorial optimization*, *Wiley-Interscience Series in Discrete Mathematics* (1985).

[7] H. Kellerer, U. Pferschy, and D. Pisinger, *Multidimensional knapsack problems*, in *Knapsack problems*, pp. 235–283. Springer, 2004.

[8] T. R. Jensen and B. Toft, *Graph coloring problems*. John Wiley & Sons, 2011.

[9] E. Cela, *The quadratic assignment problem: theory and algorithms*, vol. 1. Springer Science & Business Media, 2013.

[10] D. S. Johnson, *Approximation algorithms for combinatorial problems*, in *Proceedings of the fifth annual ACM symposium on Theory of computing*, pp. 38–49, 1973.

[11] H. A. Kautz, B. Selman, *et. al.*, *Planning as satisfiability.*, in *ECAI*, vol. 92, pp. 359–363, 1992.

[12] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, Englewood Cliffs, NJ, 1993.

[13] R. Asín Achá and R. Nieuwenhuis, *Curriculum-based course timetabling with sat and maxsat*, *Annals of Operations Research* **218** (2014), no. 1 71–91.

[14] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu, *Symbolic model checking without bdds*, in *International conference on tools and algorithms for the construction and analysis of systems*, pp. 193–207, Springer, 1999.

[15] S. Dai, G. Liu, and Z. Zhang, *A scalable approach to exact resource-constrained scheduling based on a joint sdc and sat formulation*, in *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 137–146, 2018.

[16] D. Bertsimas and R. Shioda, *Classification and regression via integer optimization*, *Operations Research* **55** (2007), no. 2 252–271.

[17] E. Althaus, S. Canzar, and H.-P. Lenhof, *Integer linear programming in computational biology*, in *Better Algorithms for Bioinformatics: Dagstuhl Seminar Proceedings*, 2008.

[18] A. Perdomo-Ortiz, N. Dickson, M. Drew-Brook, G. Rose, and A. Aspuru-Guzik, *Finding low-energy conformations of lattice protein models by quantum annealing*, *Scientific reports* **2** (2012), no. 1 571.

[19] R. L. Graham, B. L. Rothschild, and J. H. Spencer, *Ramsey Theory*. Wiley, New York, 2 ed., 1990.

[20] G. Exoo and D. Ismailescu, *New lower bounds for classical ramsey numbers*, *Electronic Journal of Combinatorics* **21** (2014), no. 1 P1.40.

[21] N. Mohseni, P. L. McMahon, and T. Byrnes, *Ising machines as hardware solvers of combinatorial optimization problems*, *Nature Reviews Physics* **4** (2022), no. 6 363–379.

[22] F. Glover, G. Kochenberger, and Y. Du, *A tutorial on formulating and using qubo models*, *arXiv preprint arXiv:1811.11538* (2018).

[23] A. Lucas, *Ising formulations of many np problems*, *Frontiers in physics* **2** (2014) 5.

[24] P. Hauke, H. G. Katzgraber, W. Lechner, H. Nishimori, and W. D. Oliver, *Perspectives of quantum annealing: Methods and implementations*, *Reports on Progress in Physics* **83** (2020), no. 5 054401.

[25] S. Kirkpatrick, C. D. Gelatt Jr, and M. P. Vecchi, *Optimization by simulated annealing*, *science* **220** (1983), no. 4598 671–680.

[26] Z. Wang, A. Marandi, K. Wen, R. L. Byer, and Y. Yamamoto, *Coherent ising machine based on degenerate optical parametric oscillators*, *Physical Review A* **88** (2013), no. 6 063853.

[27] J. J. Hopfield and D. W. Tank, *"neural" computation of decisions in optimization problems*, *Biological cybernetics* **52** (1985), no. 3 141–152.

[28] S. P. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[29] S. Ruder, *An overview of gradient descent optimization algorithms*, *arXiv preprint arXiv:1609.04747* (2016).

[30] J. J. Hopfield, *Neural networks and physical systems with emergent collective computational abilities.*, *Proceedings of the national academy of sciences* **79** (1982), no. 8 2554–2558.

[31] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski, *A learning algorithm for boltzmann machines*, *Cognitive science* **9** (1985), no. 1 147–169.

[32] J. J. Hopfield, *Neurons with graded response have collective computational properties like those of two-state neurons.*, *Proceedings of the national academy of sciences* **81** (1984), no. 10 3088–3092.

[33] G. Joya, M. Atencia, and F. Sandoval, *Hopfield neural networks for optimization: study of the different dynamics*, *Neurocomputing* **43** (2002), no. 1-4 219–237.

[34] R. J. Glauber, *Time-dependent statistics of the ising model*, *Journal of mathematical physics* **4** (1963), no. 2 294–307.

[35] C. Bybee, D. Kleyko, D. E. Nikonov, A. Khosrowshahi, B. A. Olshausen, and F. T. Sommer, *Efficient optimization with higher-order ising machines*, *Nature Communications* **14** (2023), no. 1 6033.

[36] S. Nikhar, S. Kannan, N. A. Aadit, S. Chowdhury, and K. Y. Camsari, *All-to-all reconfigurability with sparse and higher-order ising machines*, *Nature Communications* **15** (2024), no. 1 8977.

[37] M. Hizzani, A. Heittmann, G. Hutchinson, D. Dobrynin, T. Van Vaerenbergh, T. Bhattacharya, A. Renaudineau, D. Strukov, and J. P. Strachan, *Memristor-based hardware and algorithms for higher-order hopfield optimization solver outperforming quadratic ising machines*, in *2024 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, IEEE, 2024.

[38] D. Dobrynin, A. Renaudineau, M. Hizzani, D. Strukov, M. Mohseni, and J. P. Strachan, *Energy landscapes of combinatorial optimization in ising machines*, *Physical Review E* **110** (2024), no. 4 045308.

[39] S. Matsubara, M. Takatsu, T. Miyazawa, T. Shibasaki, Y. Watanabe, K. Takemoto, and H. Tamura, *Digital annealer for high-speed solving of combinatorial optimization problems and its applications*, in *2020 25th Asia and South Pacific design automation conference (ASP-DAC)*, pp. 667–672, IEEE, 2020.

[40] M. Aramon, G. Rosenberg, E. Valiante, T. Miyazawa, H. Tamura, and H. G. Katzgraber, *Physics-inspired optimization for quadratic unconstrained problems using a digital annealer*, *Frontiers in Physics* **7** (2019) 48.

[41] M. Yamaoka, C. Yoshimura, M. Hayashi, T. Okuyama, H. Aoki, and H. Mizuno, *A 20k-spin ising chip to solve combinatorial optimization problems with cmos annealing*, *IEEE Journal of Solid-State Circuits* **51** (2015), no. 1 303–309.

[42] Y. Su, H. Kim, and B. Kim, *31.2 cim-spin: A 0.5-to-1.2 v scalable annealing processor using digital compute-in-memory spin operators and register-based spins for combinatorial optimization problems*, in *2020 IEEE International Solid-State Circuits Conference-(ISSCC)*, pp. 480–482, IEEE, 2020.

[43] F. Cai, S. Kumar, T. Van Vaerenbergh, X. Sheng, R. Liu, C. Li, Z. Liu, M. Foltin, S. Yu, Q. Xia, *et. al.*, *Power-efficient combinatorial optimization using intrinsic noise in memristor hopfield neural networks*, *Nature Electronics* **3** (2020), no. 7 409–418.

[44] M. Mahmoodi, M. Prezioso, and D. Strukov, *Versatile stochastic dot product circuits based on nonvolatile memories for high performance neurocomputing and neurooptimization*, *Nature communications* **10** (2019), no. 1 5113.

[45] G. E. Moore, *Cramming more components onto integrated circuits*, *Proceedings of the IEEE* **86** (1998), no. 1 82–85.

[46] W. A. Wulf and S. A. McKee, *Hitting the memory wall: Implications of the obvious*, *ACM SIGARCH computer architecture news* **23** (1995), no. 1 20–24.

[47] Y. Kim, V. Seshadri, D. Lee, J. Liu, and O. Mutlu, *A case for exploiting subarray-level parallelism (salp) in dram*, *ACM SIGARCH Computer Architecture News* **40** (2012), no. 3 368–379.

[48] S. M. Kang and Y. Leblebici, *CMOS digital integrated circuits*. MacGraw-Hill New York, NY, USA:, 2003.

[49] M. Horiguchi, *Redundancy techniques for high-density drams*, in *1997 Proceedings Second Annual IEEE International Conference on Innovative Systems in Silicon*, pp. 22–29, IEEE, 1997.

[50] M. Horowitz, *1.1 computing's energy problem (and what we can do about it)*, in *2014 IEEE international solid-state circuits conference digest of technical papers (ISSCC)*, pp. 10–14, IEEE, 2014.

[51] J. Nickolls and W. J. Dally, *The gpu computing era*, *IEEE micro* **30** (2010), no. 2 56–69.

[52] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, *et. al.*, *In-datacenter performance analysis of a tensor processing unit*, in *Proceedings of the 44th annual international symposium on computer architecture*, pp. 1–12, 2017.

[53] Y.-H. Chen, J. Emer, and V. Sze, *Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks*, *ACM SIGARCH computer architecture news* **44** (2016), no. 3 367–379.

[54] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, *Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning*, *ACM SIGARCH Computer Architecture News* **42** (2014), no. 1 269–284.

[55] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, *Shidiannao: Shifting vision processing closer to the sensor*, in *Proceedings of the 42nd annual international symposium on computer architecture*, pp. 92–104, 2015.

[56] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, *Eie: Efficient inference engine on compressed deep neural network*, *ACM SIGARCH Computer Architecture News* **44** (2016), no. 3 243–254.

[57] 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** (2002), no. 2 34–44.

[58] J. S. Kim and Y. H. Kim, *HBM: Memory solution for bandwidth-hungry processors*, in *2014 IEEE Hot Chips 26 Symposium (HCS)*, pp. 1–24, IEEE, 2014.

[59] J. T. Pawlowski, *Hybrid Memory Cube (HMC)*, in *2011 IEEE Hot Chips 23 Symposium (HCS)*, pp. 1–24, IEEE, 2011.

[60] M. Farooq, J. S. Im, P. S. Andry, P. W. Coteus, J. U. Knickerbocker, W. E. McMahon, A. S. Rathore, K. Sakuma, E. J. Sprogis, J.-h. Su, and S. L. Wright, *3d copper tsv integration, testing and reliability*, in *2011 IEEE International Electron Devices Meeting (IEDM)*, pp. 7.1.1–7.1.4, IEEE, 2011.

[61] A. Sebastian, M. Le Gallo, R. Khaddam-Aljameh, and E. Eleftheriou, *Memory devices and applications for in-memory computing*, *Nature Nanotechnology* **15** (2020), no. 7 529–544.

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

[63] A. Biswas and A. P. Chandrakasan, *Conv-sram: An energy-efficient sram with in-memory dot-product computation for low-power convolutional neural networks*, *IEEE Journal of Solid-State Circuits* **54** (2018), no. 1 217–230.

[64] X. Guo, F. Merrikh-Bayat, M. Bavandpour, M. Klachko, M. R. Mahmoodi, M. Prezioso, K. K. Likharev, and D. B. Strukov, *Fast, energy-efficient, robust, and reproducible mixed-signal neuromorphic classifier based on embedded nor flash memory technology*, in *2017 IEEE International Electron Devices Meeting (IEDM)*, pp. 6.5.1–6.5.4, IEEE, 2017.

[65] H. Kim, M. Mahmoodi, H. Nili, and D. B. Strukov, *4k-memristor analog-grade passive crossbar circuit*, *Nature communications* **12** (2021), no. 1 5198.

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

[67] T. Gokmen and Y. Vlasov, *Acceleration of deep neural network training with resistive cross-point devices: Design considerations*, *Frontiers in neuroscience* **10** (2016) 333.

[68] S. Ambrogio, P. Narayanan, H. Tsai, R. M. Shelby, I. Boybat, C. Di Nolfo, S. Sidler, M. Giordano, M. Bodini, N. C. Farinha, *et. al.*, *Equivalent-accuracy accelerated neural-network training using analogue memory*, *Nature* **558** (2018), no. 7708 60–67.

[69] T. Bhattacharya, G. H. Hutchinson, G. Pedretti, X. Sheng, J. Ignowski, T. Van Vaerenbergh, R. Beausoleil, J. P. Strachan, and D. B. Strukov, *Computing high-degree polynomial gradients in memory*, *Nature Communications* **15** (2024), no. 1 8211.

[70] T. Bhattacharya, D. Kwon, G. H. Hutchinson, X. Zhang, I. Rozada, and D. Strukov, *A fully integrated mixed-signal compute-in-memory accelerator for solving arbitrary order boolean satisfiability problems*, in *2025 Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*, pp. 1–3, IEEE, 2025.

[71] T. Bhattacharya, D. Kwon, G. Hutchinson, X. Zhang, G. Pedretti, F. Bohm, J. P. Strachan, T. Van Vaerenbergh, R. Beausoleil, I. Rozada, *et. al.*, *Klima:*

Low-latency mixed-signal in-memory computing accelerator for solving arbitrary-order boolean satisfiability, in *2025 IEEE Hot Chips 37 Symposium (HCS)*, pp. 1–7, IEEE Computer Society, 2025.

[72] T. Bhattacharya, G. H. Hutchinson, D. Kwon, and D. Strukov, *Pcim-sat: A 55nm probabilistic K-SAT solver with p-bit-based parallel-variable update on a mixed-signal compute-in-memory architecture*, in *Proceedings of the IEEE International Solid-State Circuits Conference (ISSCC)*, 2026. To appear.

[73] T. Bhattacharya, G. H. Hutchinson, G. Pedretti, and D. Strukov, *Ho-fpia: High-order field-programmable ising arrays with in-memory computing*, in *2024 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 252–259, IEEE, 2024.

[74] G. H. Hutchinson, T. Bhattacharya, D. Kwon, F. Bohm, G. Pedretti, T. Van Vaerenbergh, R. Beausoleil, A. Heitmann, J. P. Strachan, and D. B. Strukov, *Compressed high-order ising machines*, *IEEE Journal on Emerging and Selected Topics in Circuits and Systems (JETCAS)* (2025). under review.

[75] T. Bhattacharya, S. Bezugam, S. Pande, E. Wlazlak, and D. Strukov, *Reram-based neohebbian synapses for faster training-time-to-accuracy neuromorphic hardware*, in *2023 International Electron Devices Meeting (IEDM)*, pp. 1–4, IEEE, 2023.

[76] T. Bhattacharya, H. J. Kim, and D. Strukov, *Memristor-based neuromorphic readout circuit for event cameras*, in *Proceedings of the 2025 59th Asilomar Conference on Signals, Systems, and Computers*, 2025. to appear.

[77] J. L. Johnson, *A neural network approach to the 3-satisfiability problem*, *Journal of Parallel and Distributed Computing* **6** (1989), no. 2 435–449.

[78] A. Sharma, M. Burns, A. Hahn, and M. Huang, *Augmenting an electronic ising machine to effectively solve boolean satisfiability*, *Scientific Reports* **13** (2023), no. 1 22858.

[79] W. E. Hart and S. Istrail, *Robust proofs of np-hardness for protein folding: general lattices and energy potentials*, *Journal of Computational Biology* **4** (1997), no. 1 1–22.

[80] R. Xia, T. Bian, and S. Kais, *Electronic structure calculations and the ising hamiltonian*, *The Journal of Physical Chemistry B* **122** (2017), no. 13 3384–3395.

[81] K. Jordán, *Calculus of finite differences*, vol. 33. American Mathematical Soc., 1965.

[82] H. H. Hoos and T. Stʊtzle, *Stochastic local search*, in *Handbook of approximation algorithms and metaheuristics*, pp. 297–307. Chapman and Hall/CRC, 2018.

[83] B. Selman, H. A. Kautz, B. Cohen, *et. al.*, *Noise strategies for improving local search*, in *AAAI*, vol. 94, pp. 337–343, 1994.

[84] B. Selman, D. G. Mitchell, and H. J. Levesque, *Generating hard satisfiability problems*, *Artificial intelligence* **81** (1996), no. 1-2 17–29.

[85] C. Li, J. Ignowski, X. Sheng, R. Wessel, B. Jaffe, J. Ingemi, C. Graves, and J. P. Strachan, *Cmos-integrated nanoscale memristive crossbars for cnn and optimization acceleration*, in *2020 IEEE International Memory Workshop (IMW)*, pp. 1–4, IEEE, 2020.

[86] X. Sheng, C. E. Graves, S. Kumar, X. Li, B. Buchanan, L. Zheng, S. Lam, C. Li, and J. P. Strachan, *Low-conductance and multilevel cmos-integrated nanoscale oxide memristors*, *Advanced electronic materials* **5** (2019), no. 9 1800876.

[87] H. H. Hoos and T. Stützle, *Satlib: An online resource for research on sat*, *Sat* **2000** (2000) 283–292.

[88] C. Mu, Y. Wang, J. Zheng, S. Liu, K. Zhou, S. Tang, C. Chen, and Q. Liu, *A 200m-query-vector/s computing-in-rram adc-less k-nearest-neighbor accelerator with time-domain winner-takes-all circuits*, in *2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pp. 222–225, IEEE, 2022.

[89] K. Kanazawa and T. Maruyama, *An approach for solving large sat problems on fpga*, *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* **4** (2010), no. 1 1–21.

[90] X. Guo, F. M. Bayat, M. Prezioso, Y. Chen, B. Nguyen, N. Do, and D. B. Strukov, *Temperature-insensitive analog vector-by-matrix multiplier based on 55 nm nor flash memory cells*, in *2017 IEEE Custom Integrated Circuits Conference (CICC)*, pp. 1–4, IEEE, 2017.

[91] T. Inagaki, Y. Haribara, K. Igarashi, T. Sonobe, S. Tamate, T. Honjo, A. Marandi, P. L. McMahon, T. Umeki, K. Enbutsu, *et. al.*, *A coherent ising machine for 2000-node optimization problems*, *Science* **354** (2016), no. 6312 603–606.

[92] D. Willsch, M. Willsch, C. D. G. Calaza, F. Jin, H. De Raedt, M. Svensson, and K. Michielsen, *Benchmarking advantage and d-wave 2000q quantum annealers with exact cover problems*, *arXiv preprint arXiv:2105.02208* (2021).

[93] N. A. Aadit, A. Grimaldi, M. Carpentieri, L. Theogarajan, J. M. Martinis, G. Finocchio, and K. Y. Camsari, *Massively parallel probabilistic computing with sparse ising machines*, *Nature Electronics* **5** (2022), no. 7 460–468.

[94] S. Park, J.-W. Nam, and S. K. Gupta, *Hw-bcp: A custom hardware accelerator for sat suitable for single chip implementation for large benchmarks*, in *Proceedings of the 26th Asia and South Pacific Design Automation Conference*, pp. 29–34, 2021.

[95] G. Pedretti, F. Böhm, M. Hizzani, T. Bhattacharya, P. Bruel, J. Moon, S. Serebryakov, D. Strukov, J. Strachan, J. Ignowski, *et. al.*, *Zeroth and higher-order logic with content addressable memories*, in *2023 International Electron Devices Meeting (IEDM)*, pp. 1–4, IEEE, 2023.

[96] S. Xie, M. Yang, S. A. Lanham, Y. Wang, M. Wang, S. Oruganti, and J. P. Kulkarni, *Snap-sat: A one-shot energy-performance-aware all-digital compute-in-memory solver for large-scale hard boolean satisfiability problems.*, in *ISSCC*, pp. 420–421, 2023.

[97] C. M. Li and W. Q. Huang, *Diversification and determinism in local search for satisfiability*, in *International Conference on Theory and Applications of Satisfiability Testing*, pp. 158–172, Springer, 2005.

[98] C. Shim, J. Bae, and B. Kim, *30.3 vip-sat: A boolean satisfiability solver featuring 5× 12 variable in-memory processing elements with 98% solvability for 50-variables 218-clauses 3-sat problems*, in *2024 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 67, pp. 486–488, IEEE, 2024.

[99] Q. Zhang, S. Su, Z. Liu, H.-C. Cheng, Z. Qiu, M. Palaria, J. Ye, D. Meng, B. Chen, S. Hossain, *et. al.*, *A stochastic analog sat solver in 65 nm cmos achieving 6.6 μs average solution time with 100% solvability for hard 3-sat problems*, in *Proceedings of the 2024 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*, pp. 1–2, IEEE, 2024.

[100] A. Sharma, M. Burns, and M. C. Huang, *Combining cubic dynamical solvers with make/break heuristics to solve sat*, in *26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2023)*, pp. 25–1, Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2023.

[101] B. Razavi, *The strongarm latch [a circuit for all seasons]*, *IEEE Solid-State Circuits Magazine* **7** (2015), no. 2 12–17.

[102] W. Regli, G. Mossi, M. T. Hajiaghayi, H. Munoz Bauza, I. J. Whitehouse, K. Banihashem, P. Jabbarzade, and T. H. Paul, "Sat benchmarks to assess quantum-inspired solvers." `https://github.com/UMD-ARLIS/QuICC-SAT-Datasets.git`, 2025. Licensed under CC BY 4.0.

[103] S. Patel, P. Canoza, and S. Salahuddin, *Logically synthesized and hardware-accelerated restricted boltzmann machines for combinatorial optimization and integer factorization*, *Nature Electronics* **5** (2022), no. 2 92–101.

[104] H. Cılasun, Z. Zeng, R. S, A. Kumar, H. Lo, W. Cho, W. Moy, C. H. Kim, U. R. Karpuzcu, and S. S. Sapatnekar, *3sat on an all-to-all-connected cmos ising solver chip*, *Scientific reports* **14** (2024), no. 1 10757.

[105] A. Biere, T. Faller, K. Fazekas, M. Fleury, N. Froleyks, and F. Pollitt, *CaDiCaL, Gimsatul, IsaSAT and Kissat entering the SAT Competition 2024*, in *Proc. of SAT Competition 2024 – Solver, Benchmark and Proof Checker Descriptions* (M. Heule, M. Iser, M. Järvisalo, and M. Suda, eds.), vol. B-2024-1 of *Department of Computer Science Report Series B*, pp. 8–10, University of Helsinki, 2024.

[106] J. Marques-Silva, *Practical applications of boolean satisfiability*, in *2008 9th International Workshop on Discrete Event Systems*, pp. 74–80, IEEE, 2008.

[107] Z. Wu, X. Tang, T. Zhang, L. Lin, H. Luo, B. Xu, Z. Wu, J. Song, Y. Liang, X. Bo, *et. al.*, *Skadi: A 28-nm complete k-sat solver featuring bidirectional in-memory deduction and incremental updating*, *IEEE Journal of Solid-State Circuits* (2025).

[108] E. Dikopoulos, Y.-T. Hsu, L. Wormald, W. Tang, Z. Zhang, and M. P. Flynn, *25.1 a physics-inspired oscillator-based mixed-signal optimization engine for solving 50-variable 218-clause 3-sat problems with 100% solvability and 31.7 μs solution time*, in *2025 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 68, pp. 01–03, IEEE, 2025.

[109] S. Tsukamoto, M. Takatsu, S. Matsubara, and H. Tamura, *An accelerator architecture for combinatorial optimization problems*, *Fujitsu Sci. Tech. J* **53** (2017), no. 5 8–13.

[110] A. Grimaldi, L. Sánchez-Tejerina, N. Anjum Aadit, S. Chiappini, M. Carpentieri, K. Camsari, and G. Finocchio, *Spintronics-compatible approach to solving maximum-satisfiability problems with probabilistic computing, invertible logic, and parallel tempering*, *Physical Review Applied* **17** (2022), no. 2 024052.

[111] N. Kumar, P. O. Pouliquen, and A. G. Andreou, *Device mismatch limitations on performance of a hamming distance classifier*, in *Proceedings of 1993 IEEE International Workshop on Defect and Fault Tolerance in VLSI Systems*, pp. 327–334, IEEE, 1993.

[112] C. Weistuch, L. R. Mujica-Parodi, and K. Dill, *The refractory period matters: unifying mechanisms of macroscopic brain waves*, *Neural Computation* **33** (2021), no. 5 1145–1163.

[113] S. A. Moosavi, A. Montakhab, and A. Valizadeh, *Refractory period in network models of excitable nodes: self-sustaining stable dynamics, extended scaling region and oscillatory behavior*, *Scientific reports* **7** (2017), no. 1 7107.

[114] S. Reifenstein, T. Leleu, T. McKenna, M. Jankowski, M.-G. Suh, E. Ng, F. Khoyratee, Z. Toroczkai, and Y. Yamamoto, *Coherent sat solvers: a tutorial*, *Advances in Optics and Photonics* **15** (2023), no. 2 385–441.

[115] C. Münch, F. Schinkel, S. Zielinski, and S. Walter, *Transformation-dependent performance-enhancement of digital annealer for 3-sat*, *arXiv preprint arXiv:2312.11645* (2023).

[116] A. Nandi, S. Chakrabartty, and C. S. Thakur, *Margin propagation based xor-sat solvers for decoding of ldpc codes*, *IEEE Transactions on Communications* (2024).

[117] M. Soos, S. Gocht, and K. S. Meel, *Tinted, detached, and lazy cnf-xor solving and its applications to counting and sampling*, in *International Conference on Computer Aided Verification*, pp. 463–484, Springer, 2020.

[118] E. Bellini, A. De Piccoli, R. Makarim, S. Polese, L. Riva, and A. Visconti, *New records of pre-image search of reduced sha-1 using sat solvers*, in *Proceedings of the Seventh International Conference on Mathematics and Computing: ICMC 2021*, pp. 141–151, Springer, 2022.

[119] G. S. Tseitin, *On the complexity of derivation in propositional calculus*, in *Automation of reasoning: 2: Classical papers on computational logic 1967–1970*, pp. 466–483. Springer, 1983.

[120] W. Nawrocki, Z. Liu, A. Fröhlich, M. J. Heule, and A. Biere, *Xor local search for boolean brent equations*, in *International Conference on Theory and Applications of Satisfiability Testing*, pp. 417–435, Springer, 2021.

[121] T. Balyo, N. Froleyks, M. J. Heule, M. Iser, M. Järvisalo, and M. Suda, *Proceedings of sat competition 2020: Solver and benchmark descriptions*, .

[122] P. Purdom and A. Sabry, "Cnf generator for factoring problems." Web page, 2014. Available: `https://cgi.luddy.indiana.edu/~sabry/cnf.html` [Accessed: insert date].

[123] I. Hen, *Equation planting: a tool for benchmarking ising machines*, *Physical Review Applied* **12** (2019), no. 1 011003.

[124] G. Pedretti, F. Böhm, T. Bhattacharya, A. Heittmann, X. Zhang, M. Hizzani, G. Hutchinson, D. Kwon, J. Moon, E. Valiante, *et. al.*, *Solving boolean satisfiability problems with resistive content addressable memories*, *npj Unconventional Computing* **2** (2025), no. 1 7.

[125] J. K. Lenstra, A. R. Kan, and P. Brucker, *Complexity of machine scheduling problems*, in *Annals of discrete mathematics*, vol. 1, pp. 343–362. Elsevier, 1977.

[126] P. Kramer, J. Van Leeuwen, *et. al.*, *Wire routing in np-complete*, .

[127] S. McArdle, S. Endo, A. Aspuru-Guzik, S. C. Benjamin, and X. Yuan, *Quantum computational chemistry*, *Reviews of Modern Physics* **92** (2020), no. 1 015003.

[128] I. G. Rosenberg, *Reduction of bivalent maximization to the quadratic case.*, .

[129] M. Bavandpour, M. R. Mahmoodi, and D. B. Strukov, *Acortex: An energy-efficient multipurpose mixed-signal inference accelerator*, *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits* **6** (2020), no. 1 98–106.

[130] G. H. Hutchinson, E. Sifferman, T. Bhattacharya, D. Kwon, and D. B. Strukov, *Fpia: Field-programmable ising arrays with in-memory computing*, in *Proceedings of the 29th ACM/IEEE International Symposium on Low Power Electronics and Design*, pp. 1–6, 2024.

[131] V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD for deep-submicron FPGAs*, vol. 497. Springer Science & Business Media, 2012.

[132] A. Mondal and A. Srivastava, *Ising-fpga: A spintronics-based reconfigurable ising model solver*, *ACM Transactions on Design Automation of Electronic Systems (TODAES)* **26** (2020), no. 1 1–27.

[133] V. Betz and J. Rose, *Vpr: A new packing, placement and routing tool for fpga research*, in *International Workshop on Field Programmable Logic and Applications*, pp. 213–222, Springer, 1997.

[134] L. Kull, T. Toifl, M. Schmatz, P. A. Francese, C. Menolfi, M. Braendli, M. Kossel, T. Morf, T. M. Andersen, and Y. Leblebici, *A 3.1 mw 8b 1.2 gs/s single-channel asynchronous sar adc with alternate comparators for enhanced speed in 32 nm digital soi cmos*, *IEEE Journal of Solid-State Circuits* **48** (2013), no. 12 3049–3058.

[135] A. Jaiswal, I. Chakraborty, A. Agrawal, and K. Roy, *8t sram cell as a multibit dot-product engine for beyond von neumann computing*, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* **27** (2019), no. 11 2556–2567.

[136] S. Mittal, J. S. Vetter, and D. Li, *A survey of architectural approaches for managing embedded dram and non-volatile on-chip caches*, *IEEE Transactions on Parallel and Distributed Systems* **26** (2014), no. 6 1524–1537.

[137] M. R. Mahmoodi and D. Strukov, *An ultra-low energy internally analog, externally digital vector-matrix multiplier based on nor flash memory technology*, in *Proceedings of the 55th Annual Design Automation Conference*, pp. 1–6, 2018.

[138] S. Yin, Z. Jiang, J.-S. Seo, and M. Seok, *Xnor-sram: In-memory computing sram macro for binary/ternary deep neural networks*, *IEEE Journal of Solid-State Circuits* **55** (2020), no. 6 1733–1743.

[139] T. Balyo, A. Biere, M. J. H. Heule, and M. J. Iser, "Sat competition 2017: Benchmark datasets." `http://satcompetition.org/2017/`, 2017.

[140] M. Lindauer, K. Eggensperger, M. Feurer, A. Biedenkapp, D. Deng, C. Benjamins, T. Ruhkopf, R. Sass, and F. Hutter, *Smac3: A versatile bayesian optimization package for hyperparameter optimization*, *Journal of Machine Learning Research* **23** (2022), no. 54 1–9.

[141] D. J. Earl and M. W. Deem, *Parallel tempering: Theory, applications, and new perspectives*, *Physical Chemistry Chemical Physics* **7** (2005), no. 23 3910–3916.

[142] M. Davis, G. Logemann, and D. Loveland, *A machine program for theorem-proving*, *Communications of the ACM* **5** (1962), no. 7 394–397.

[143] J. P. Marques-Silva and K. A. Sakallah, *Grasp: A search algorithm for propositional satisfiability*, *IEEE Transactions on Computers* **48** (2002), no. 5 506–521.

[144] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, *Chaff: Engineering an efficient sat solver*, in *Proceedings of the 38th annual Design Automation Conference*, pp. 530–535, 2001.

[145] J. Marques-Silva and J. Planes, *Algorithms for maximum satisfiability using unsatisfiable cores*, in *Proceedings of the conference on Design, automation and test in Europe*, pp. 408–413, 2008.

[146] S. Sassa, K. Kanazawa, S. Cai, and M. Yasunaga, *An fpga solver for partial maxsat problems based on stochastic local search*, *ACM SIGARCH Computer Architecture News* **44** (2017), no. 4 32–37.

[147] M. Carver, *Analog VLSI and neural systems*. Addison-Wesley, 1989.

[148] C. Eliasmith and C. H. Anderson, *Neural engineering: Computation, representation, and dynamics in neurobiological systems*. MIT press, 2003.

[149] G. Indiveri and S.-C. Liu, *Memory and information processing in neuromorphic systems*, *Proceedings of the IEEE* **103** (2015), no. 8 1379–1397.

[150] K. Boahen, *Neuromorphic microchips*, *Scientific American* **292** (2005), no. 5 56–63.

[151] S. Furber, *Large-scale neuromorphic computing systems*, *Journal of neural engineering* **13** (2016), no. 5 051001.

[152] J. Laydevant, L. G. Wright, T. Wang, and P. L. McMahon, *The hardware is the software*, *Neuron* **112** (2024), no. 2 180–183.

[153] C. Luo, S. Cai, K. Su, and W. Huang, *Ccehc: An efficient local search algorithm for weighted partial maximum satisfiability*, *Artificial Intelligence* **243** (2017) 26–44.

[154] Y. Bengio, A. Lodi, and A. Prouvost, *Machine learning for combinatorial optimization: a methodological tour d'horizon*, *European Journal of Operational Research* **290** (2021), no. 2 405–421.

[155] M. Shirokikh, I. Shenbin, A. Alekseev, and S. Nikolenko, *Machine learning for sat: Restricted heuristics and new graph representations*, *arXiv preprint arXiv:2307.09141* (2023).

[156] M. Karimi-Mamaghan, M. Mohammadi, P. Meyer, A. M. Karimi-Mamaghan, and E.-G. Talbi, *Machine learning at the service of meta-heuristics for solving combinatorial optimization problems: A state-of-the-art*, *European Journal of Operational Research* **296** (2022), no. 2 393–422.

[157] D. Dobrynin, M. Mohseni, and J. P. Strachan, *Nonlocal monte carlo via reinforcement learning*, *arXiv preprint arXiv:2508.10520* (2025).

[158] Y. Sun, F. Ye, Z. Chen, K. Wei, and S. Cai, *Automatically discovering heuristics in a complex sat solver with large language models*, *arXiv preprint arXiv:2507.22876* (2025).

[159] C. Yu, R. Liang, C.-T. Ho, and H. Ren, *Autonomous code evolution meets np-completeness*, *arXiv preprint arXiv:2509.07367* (2025).

[160] S. Liu, C. Mu, H. Jiang, Y. Wang, J. Zhang, F. Lin, K. Zhou, Q. Liu, and C. Chen, *Hardsea: Hybrid analog-reram clustering and digital-sram in-memory computing accelerator for dynamic sparse self-attention in transformer*, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* **32** (2023), no. 2 269–282.

[161] A. Sridharan, F. Zhang, J.-S. Seo, and D. Fan, *Sp-imc: A sparsity aware in-memory-computing macro in 28nm cmos with configurable sparse representation for highly sparse dnn workloads*, in *2024 IEEE Custom Integrated Circuits Conference (CICC)*, pp. 1–2, IEEE, 2024.

[162] A. Sharma, R. Afoakwa, Z. Ignjatovic, and M. Huang, *Increasing ising machine capacity with multi-chip architectures*, in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, pp. 508–521, 2022.

[163] L. Witters, S. Van Huylenbroeck, S. Kang, P. Zhao, S. Chew, K. D'havé, S. Iacovo, M. Stucchi, B. Zhang, S. Dewilde, *et. al.*, *High-density wafer level connectivity using frontside hybrid bonding at 250nm pitch and backside through-dielectric vias at 120nm pitch after extreme wafer thinning*, in *2025 Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*, pp. 1–3, IEEE, 2025.

[164] Y. Demir, Y. Pan, S. Song, N. Hardavellas, J. Kim, and G. Memik, *Galaxy: A high-performance energy-efficient multi-chip architecture using photonic interconnects*, in *Proceedings of the 28th ACM international conference on Supercomputing*, pp. 303–312, 2014.

[165] J. Park, T. Yu, S. Joshi, C. Maier, and G. Cauwenberghs, *Hierarchical address event routing for reconfigurable large-scale neuromorphic systems*, *IEEE transactions on neural networks and learning systems* **28** (2016), no. 10 2408–2422.

[166] Xilinx, Inc., *7 series fpga power benchmark design summary*, tech. rep., Xilinx, Jan, 2015. Accessed: 2025-11-16.

[167] T. Prickett Morgan, *Boosting the clock for high performance fpga inference*, *The Next Platform* (Oct, 2018). Accessed: 2025-11-16.

[168] Y. LeCun, Y. Bengio, and G. Hinton, *Deep learning*, *nature* **521** (2015), no. 7553 436–444.

[169] J. Devlin, M.-W. Chang, K. Lee, and K. 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)*, pp. 4171–4186, 2019.

[170] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, *Scaling laws for neural language models*, *arXiv preprint arXiv:2001.08361* (2020).

[171] A.-S. Lesiv and S. Maini, "How much energy will it take to power ai?." `https://research.contrary.com/foundations-and-frontiers/ai-inference`, July, 2024. Accessed: ¡insert date you accessed it¿.

[172] R. Serrano-Gotarredona, M. Oster, P. Lichtsteiner, A. Linares-Barranco, R. Paz-Vicente, F. Gomez-Rodriguez, H. Kolle Riis, T. Delbruck, S.-C. Liu, S. Zahnd, *et. al.*, *Aer building blocks for multi-layer multi-chip neuromorphic vision systems*, *Advances in neural information processing systems* **18** (2005).

[173] P. J. Werbos, *Backpropagation through time: what it does and how to do it*, *Proceedings of the IEEE* **78** (2002), no. 10 1550–1560.

[174] G. Bellec, F. Scherr, A. Subramoney, E. Hajek, D. Salaj, R. Legenstein, and W. Maass, *A solution to the learning dilemma for recurrent networks of spiking neurons*, *Nature communications* **11** (2020), no. 1 3625.

[175] S. Pande, S. S. Bezugam, T. Bhattacharya, E. Wlazlak, A. Chakaravorty, B. Chakrabarti, and D. Strukov, *Neohebbian synapses to accelerate online training of neuromorphic hardware*, *arXiv preprint arXiv:2411.18272* (2024).

[176] C. Frenkel and G. Indiveri, *Reckon: A 28nm sub-mm2 task-agnostic spiking recurrent neural network processor enabling on-chip learning over second-long timescales*, in *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 65, pp. 1–3, IEEE, 2022.

[177] A. Baevski, Y. Zhou, A. Mohamed, and M. Auli, *wav2vec 2.0: A framework for self-supervised learning of speech representations*, *Advances in neural information processing systems* **33** (2020) 12449–12460.

[178] T. Bohnstingl, A. Šurina, M. Fabre, Y. Demirağ, C. Frenkel, M. Payvand, G. Indiveri, and A. Pantazi, *Biologically-inspired training of spiking recurrent neural networks with neuromorphic hardware*, in *2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pp. 218–221, IEEE, 2022.

[179] H.-S. P. Wong, H.-Y. Lee, S. Yu, Y.-S. Chen, Y. Wu, P.-S. Chen, B. Lee, F. T. Chen, and M.-J. Tsai, *Metal–oxide rram*, *Proceedings of the IEEE* **100** (2012), no. 6 1951–1970.

[180] T. P. Truong, H. T. Le, and T. T. Nguyen, *A reconfigurable hardware platform for low-power wide-area wireless sensor networks*, in *Journal of Physics: Conference Series*, vol. 1432, p. 012068, IOP Publishing, 2020.

[181] M. Arnott and J. Morrish, *Global iot forecast report 2024–2034*, industry research report, Transforma Insights, Jun, 2025. Accessed: 2025-11-02.

[182] F. Zhou and Y. Chai, *Near-sensor and in-sensor computing*, *Nature Electronics* **3** (2020), no. 11 664–671.

[183] D. Strukov, G. Indiveri, J. Grollier, and S. Fusi, *Building brain-inspired computing*, *Nature Communications* (2019), no. 10 4838–2019.

[184] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, *et. al.*, *A million spiking-neuron integrated circuit with a scalable communication network and interface*, *Science* **345** (2014), no. 6197 668–673.

[185] F. Merrikh-Bayat, X. Guo, M. Klachko, M. Prezioso, K. K. Likharev, and D. B. Strukov, *High-performance mixed-signal neurocomputing with nanoscale floating-gate memory cell arrays*, *IEEE transactions on neural networks and learning systems* **29** (2017), no. 10 4782–4790.

[186] C. Brandli, R. Berner, M. Yang, S.-C. Liu, and T. Delbruck, *A 240× 180 130 db 3 μs latency global shutter spatiotemporal vision sensor*, *IEEE Journal of Solid-State Circuits* **49** (2014), no. 10 2333–2341.

[187] M. F. Amir, J. H. Ko, T. Na, D. Kim, and S. Mukhopadhyay, *3-d stacked image sensor with deep neural network computation*, *IEEE Sensors Journal* **18** (2018), no. 10 4187–4199.

[188] R. LiKamWa, Y. Hou, J. Gao, M. Polansky, and L. Zhong, *Redeye: analog convnet image sensor architecture for continuous mobile vision*, *ACM SIGARCH Computer Architecture News* **44** (2016), no. 3 255–266.

[189] N. Cottini, M. Gottardi, N. Massari, R. Passerone, and Z. Smilansky, *A 33μw 42 gops/w 64x64 pixel vision sensor with dynamic background subtraction for scene interpretation*, in *Proceedings of the 2012 ACM/IEEE international symposium on Low power electronics and design*, pp. 315–320, 2012.

[190] T. J. Koickal, A. Hamilton, S. L. Tan, J. A. Covington, J. W. Gardner, and T. C. Pearce, *Analog vlsi circuit implementation of an adaptive neuromorphic olfaction chip*, *IEEE Transactions on Circuits and Systems I: Regular Papers* **54** (2007), no. 1 60–73.

[191] M. M. Shulaker, G. Hills, R. S. Park, R. T. Howe, K. Saraswat, H.-S. P. Wong, and S. Mitra, *Three-dimensional integration of nanotechnologies for computing and data storage on a single chip*, *Nature* **547** (2017), no. 7661 74–78.

[192] F. Zhou, Z. Zhou, J. Chen, T. H. Choy, J. Wang, N. Zhang, Z. Lin, S. Yu, J. Kang, H.-S. P. Wong, *et. al.*, *Optoelectronic resistive random access memory for neuromorphic vision sensors*, *Nature nanotechnology* **14** (2019), no. 8 776–782.

[193] L. Mennel, J. Symonowicz, S. Wachter, D. K. Polyushkin, A. J. Molina-Mendoza, and T. Mueller, *Ultrafast machine vision with 2d material neural network image sensors*, *Nature* **579** (2020), no. 7797 62–66.

[194] S. Qin, F. Wang, Y. Liu, Q. Wan, X. Wang, Y. Xu, Y. Shi, X. Wang, and R. Zhang, *A light-stimulated synaptic device based on graphene hybrid phototransistor*, *2D Materials* **4** (2017), no. 3 035022.

[195] E. R. Fossum and B. Pain, *Infrared readout electronics for space-science sensors: state of the art and future directions*, *Infrared Technology XIX* **2020** (1993) 262–285.

[196] F. Alibart, L. Gao, B. D. Hoskins, and D. B. Strukov, *High precision tuning of state for memristive devices by adaptable variation-tolerant algorithm*, *Nanotechnology* **23** (2012), no. 7 075201.

[197] M. Prezioso, Y. Zhong, D. Gavrilov, F. Merrikh-Bayat, B. Hoskins, G. Adam, K. Likharev, and D. Strukov, *Spiking neuromorphic networks with metal-oxide memristors*, in *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 177–180, IEEE, 2016.

[198] W. Wang, G. Pedretti, V. Milo, R. Carboni, A. Calderoni, N. Ramaswamy, A. S. Spinelli, and D. Ielmini, *Learning of spatiotemporal patterns in a spiking neural network with resistive switching synapses*, *Science advances* **4** (2018), no. 9 eaat4752.

[199] B. Chakrabarti, M. A. Lastras-Montaño, G. Adam, M. Prezioso, B. Hoskins, M. Payvand, A. Madhavan, A. Ghofrani, L. Theogarajan, K.-T. Cheng, *et. al.*, *A multiply-add engine with monolithically integrated 3d memristor crossbar/cmos hybrid circuit*, *Scientific reports* **7** (2017), no. 1 42429.

[200] S.-G. Wuu, H.-L. Chen, H.-C. Chien, P. Enquist, R. M. Guidash, and J. McCarten, *A review of 3-dimensional wafer level stacked backside illuminated cmos image sensor process technologies*, *IEEE Transactions on Electron Devices* **69** (2022), no. 6 2766–2778.