Mapping Spiking Neural Networks to Heterogeneous Crossbar Architectures using Integer Linear Programming

Devin Pohl
Georgia Institute of Technology
dpohl@gatech.edu

Aaron Young
Oak Ridge National Lab
youngar@ornl.gov

Kazi Asifuzzaman
Oak Ridge National Lab
asifuzzamank@ornl.gov

Narasinga Rao Miniskar
Oak Ridge National Lab
miniskarnr@ornl.gov

Jeffrey S. Vetter
Oak Ridge National Lab
vetter@ornl.gov

Abstract*—Advances in novel hardware devices and architectures allow Spiking Neural Network (SNN) evaluation using ultra-low power, mixed-signal, memristor crossbar arrays. As individual network sizes quickly scale beyond the dimensional capabilities of single crossbars, networks must be mapped onto multiple crossbars. Crossbar sizes within modern Memristor Crossbar Architectures (MCAs) are determined predominately not by device technology but by network topology; more, smaller crossbars consume less area thanks to the high structural sparsity found in larger, brain-inspired SNNs. Motivated by *continuing* increases in SNN sparsity due to improvements in training methods, we propose utilizing heterogeneous crossbar sizes to further reduce area consumption. This approach was previously unachievable as prior compiler studies only explored solutions targeting homogeneous MCAs. Our work improves on the state-of-the-art by providing Integer Linear Programming (ILP) formulations supporting arbitrarily heterogeneous architectures. By modeling axonal interactions between neurons, our methods produce better mappings while removing inhibitive a priori knowledge requirements. We first show a 16.7–27.6% reduction in area consumption for square-crossbar homogeneous architectures. Then, we demonstrate 66.9–72.7% *further* reduction when using a reasonable configuration of heterogeneous crossbar dimensions. Next, we present a new optimization formulation capable of minimizing the number of inter-crossbar routes. When applied to solutions already near-optimal in area, an 11.9–26.4% routing reduction is observed without impacting area consumption. Finally, we present a profile-guided optimization capable of minimizing the number of runtime spikes between crossbars. Compared to the best-area-then-route optimized solutions, we observe a further 0.5–14.8% inter-crossbar spike reduction while requiring 1–3 orders of magnitude less solver time.

I. Introduction

The forefront of hardware-accelerated machine learning is currently experiencing a pivotal moment. The near-universal application of neural networks has outgrown traditional models (DNNs, CNNs, etc.) under expanding application spaces and problem complexity, with Spiking Neural Networks (SNNs) gaining popularity thanks to comparable accuracy despite significantly lower neuron counts [1]–[5]. Simultaneously, co-design with resistive-RAM technologies has contributed to material, device, and architecture-level improvements to Memristor Crossbar Architectures (MCAs). These developments together enable remarkably power- and area-efficient neuromorphic computing [6]–[11]. For the first time, co-design between MCAs and SNNs is nearing a full-stack solution for accessible, ultra-low-power inference of general-purpose, highly-accurate machine learning models.

However, dominant network structures of SNNs have shifted in recent years. Between advances in initial network generation [12]–[14], training [15], [16], and pruning methods [13], [17]–[19], networks have evolved drastically increased structural sparsity. Where previously, architectures only needed many small crossbars to support ever-growing networks, today's architectures require ever more ingenious tricks to fully take advantage of increasing sparsity [8]–[11].

Yet, these architectural improvements outpace compiler technology. Partitioning SNNs and mapping neuron clusters to various crossbars remains an open problem, with many recent approaches scaling to increasing network sizes [20]–[23]. However, these approximate solutions fail to support the level of target architecture heterogeneity required for leveraging structural sparsity to improve area and power metrics. To the best of our knowledge, SpikeHard [24] is the first study with potential to jump the gap, utilizing Integer Linear Programming (ILP) for improved area optimization.

However, two main limitations are present with this work: (1) it *requires* an initial solution and (2) it produces solutions demonstrably sub-optimal in area consumption. Furthermore, the base ILP constraints presented in SpikeHard prohibit the optimization of more useful heuristics.

This work improves on the state-of-the-art in SNN to MCA mapping through the following contributions:

- Providing ILP formulations that remove the need for an a priori known-valid solution while allowing for production of truly area-optimal solutions.
- Providing ILP formulations to minimize the number of required network routes between crossbars.
- Providing ILP formulations leveraging prior inference spike profiles for minimizing runtime network packets.

The rest of this text is organized as follows: Section II discusses background material, section III explains opportunities for improvement over the state-of-the-art, section IV shows the contributed ILP formulations and their meaning, and section V experimentally shows the effectiveness and tradeoff characteristics of these techniques.

978-3-9826741-0-0/DATE25

II. Background

This section summarizes key concepts and trends including spiking neural networks and training methods, memristor crossbar architectures and scaling efforts, use of integer linear programming in mapping, and profile-guided optimization.

A. Spiking Neural Networks

Neuromorphic computing involves utilizing bio-plausible artificial neural networks for computation. While this can include emulating neural components beyond just neurons and axons [1], the approach most relevant to applied machine learning is the Spiking Neural Network (SNN) with simple integrate-and-fire neurons. Instead of executing every neuron per input window like in DNNs, GNNs, etc, neurons in SNNs accumulate charge and, upon reaching a threshold, fire discrete "spikes" of information along axons which may delay the signals. With appropriate hardware, SNNs achieve much higher energy efficiency without significant loss in accuracy [2]–[5].

Training SNNs is still an open problem with fervent development [25]. While one class of training methods focuses on converting other types of networks (mainly CNNs, but also DNNs) into SNNs through various processes [17], [26]–[29], literature suggests that training SNNs "from scratch" produces networks with higher amounts of certain properties (e.g., gradient sparsity) [13], [14]. Continuing research on pruning, compression, and re-structuring SNNs show increasing ability to take advantage of these properties to produce more structurally sparse networks [13], [17]–[19].

B. Memristor Crossbar Architectures

Resistive-RAM (ReRAM) offers non-volatile random access memory while also supporting analog compute. While ReRAM research is not new [6], recent advances in memristor technology and architecture [30] rapidly approach production-ready systems for accelerating general-purpose SNNs. Analog ReRAM crossbar-based accelerators enable energy-efficient matrix multiplication but face non-idealities that limit crossbar dimensions.

Scaling architectures to network size (neuron count) is achieved by minimizing connection overhead via methods such as mixed-signal accumulation and hierarchical networking [8]–[10]. Scaling to density (edge count) requires more intricate tricks, such as re-purposing crossbar bit-lines for metadata [11]. Yet, the corresponding compiler technology to fully exploit such architectures remains notably absent.

C. Integer Linear Programming

The architectural advantages of high crossbar counts come with significant compiler challenges; after all, complex architectures are only useful if compilers can effectively leverage their complexity. For NP-hard problems like SNN-to-MCA mapping, traditional wisdom favors approximate, polynomial-time algorithms [20]–[23]. While these methods provide *adequate* solution quality, they fall short in supporting emerging heterogeneous crossbar architectures.

An alternative approach is Integer Linear Programming (ILP), a type of constraint programming requiring only mathematical descriptions of valid solutions. Off-the-shelf solvers then find satisfying and/or optimal solutions within generally tolerable time limits, despite NP-hard problem complexity. While recent work leverages ILP to map SNNs to MCAs with impressive reductions in area consumption [24], we will show further opportunities for reducing area consumption and inter-crossbar communication.

D. Profile Guided Optimization

The methods in this paper, in part, leverage Profile Guided Optimization (PGO) to improve average-case performance. PGO involves sampling execution to identify frequently activated components and optimizing them more aggressively.

In the context of spiking networks, certain synapses consistently experience more spikes across varying input patterns [31]–[33]. By clustering these "hot" synapses within single crossbars, expensive inter-crossbar communication is only required for infrequently utilized routes.

III. Related Work

Previous approaches to this mapping problem include block clustering [20], spectral clustering [23], exclusive sum-of-product mapping [21], and sum-of-cut-cost partitioning [22]. While effective for scaling to large network sizes, these methods produce sub-optimal mappings [24] and only support homogeneous MCAs. Modifying such algorithms for heterogeneous MCA support is yet to be attempted.

To our knowledge, only one work has employed ILP as an alternative method: SpikeHard [24]. This approach groups neurons into *Minimally Connected Components* (MCCs) and uses their aggregate dimension requirements for bin-packing. The greatest limitation is that it requires an initial solution to form MCCs—forming single-neuron MCCs is disastrous for optimization and cannot be worked around via multiple SpikeHard applications, shown empirically in Section V-D.

The second limitation is that area-optimized results are sub-optimal. This approach does not model axon-sharing, where a single word-line supplies input to multiple bit-lines in a crossbar. Consequently, placing two MCCs in the same crossbar may incorrectly require additional input lines. This effect is shown in Fig. 1 and is addressed in our approach.

The final limitation with SpikeHard is its lack of support for more complex optimizations. With input axons counted incorrectly, neither inter-crossbar connections nor network weights can be modeled with reasonable accuracy.

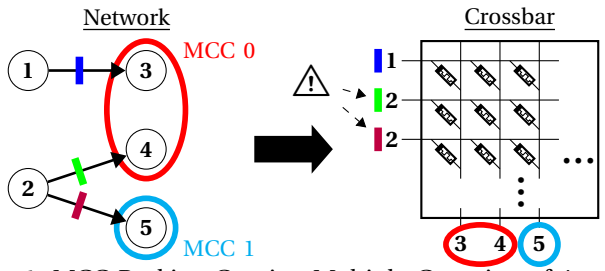


Fig. 1: MCC Packing Causing Multiple Counting of Axons

IV. Approach

This work moves away from traditional approximate solutions which sacrifice either optimality [22] or accuracy [34]. We first modify ILP constraints from [24] to support axon sharing, yielding more optimal area consumption and enabling advanced optimizations. We then apply an optimization for minimizing the count of inter-crossbar networking routes. Lastly, we introduce a profile-guided optimization to reduce runtime packets across chip routers.

A. Formulation of Constraints

The root cause of SpikeHard's shortcomings—incorrectly counting shared axons as in Fig. 1—is overcome with additional neuron placement variables. The high-level method of constructing the set of axons mapped as input to a given crossbar j is as follows:

$$\text{Inputs}_j = \bigcup_{i \in \text{Outputs}_j} \text{InputEdgesOfNode}(i) \tag{1}$$

This can be re-written imperatively in boolean logic by introducing x_{ij} for neuron placement and s_{kj} as axon placement. With graph edge definitions stored in m_{ik}:

$$s_{kj} = \bigvee_i x_{ij} \wedge m_{ik} \tag{2}$$

With the above explaining how we model axon sharing, this formula may be converted to ILP, yielding a formal definition of our solution. The following indicator variables are used:

$$\forall i, k \in \{1, ..., \#\text{ Neurons}\}$$

$$\forall j \in \{1, ..., \#\text{ Crossbars}\}$$

$$x_{ij}, s_{kj}, m_{ik}, y_j \in \{1, 0\}$$

Where:

- $x_{ij} = 1 \iff$ Neuron i is mapped to crossbar j (output)
- $m_{ik} = 1 \iff$ Neuron i takes input from neuron k
- $s_{kj} = 1 \iff$ Crossbar j takes neuron k as axonal input
- $y_j \iff$ Crossbar j is used *at all* in the design
- N_j = The number of available outputs on crossbar j
- A_j = The number of available inputs on crossbar j

N, A, and m are known a priori. Finally, solutions obey:

$$\forall i. \quad \sum_j x_{ij} = 1 \tag{3}$$

$$\forall j. \quad \sum_i x_{ij} \leq y_j N_j \tag{4}$$

$$\forall k, j. \quad s_{kj} \leq \sum_i x_{ij} m_{ik} \tag{5}$$

$$\forall i. \quad s_{kj} \geq x_{ij} m_{ik} \tag{6}$$

$$\forall j. \quad \sum_k s_{kj} \leq y_j A_j \tag{7}$$

Constraint 3 ensures each neuron outputs to one crossbar, while 4 prevents exceeding crossbar output capacity. Constraints 5 and 6 model synapse sharing as expressed in 2. Finally, constraint 7 limits input capacity. Together, N_j and A_j describe all available crossbar dimensions. These constraints form the foundation of our mapping algorithm.

B. Optimization for Area

The formulations in constraints 3–7 together describe a valid mapping of an SNN described in m_{ik} for a given MCA described in A_j and N_j. Now, utilizing the solution variables y_j, this approach moves from finding *some* valid solution to finding the *best* solution. The variables y_j describe whether or not crossbar j is "enabled," or has any neurons mapped to it. Minimizing area is achieved by minimizing the weighted sum of enabled crossbars; a constant area approximation factor C_j is included to consider non-linear area scaling of overhead hardware. This idea is expressed as the objective:

$$min\left(\sum_j y_j C_j\right) \tag{8}$$

This objective, together with the earlier constraints, may be passed to an ILP solver to produce solutions optimal in area consumption. Because of the added complexity of axon sharing and more solution variables ($i \in \{1, ..., \#\text{ Neurons}\}$ instead of $i \in \{1, ..., \#\text{ MCCs}\}$), optimization will occur more slowly at first compared to SpikeHard. However, given that SpikeHard can only further improve by being applied iteratively with successively larger MCCs, our approach will overtake it in an acceptable amount of solver time as demonstrated empirically in Section V-D.

C. Static Optimization for Number of Routes

Modeling axon sharing is not only useful for lower area consumption; accurate counts of axons also allows analyzing the connections between crossbars. Minimizing the number of these routes has a direct impact on energy consumption, network congestion, and router capability requirements. The heuristic preferring *local routes* over *global routes* is termed *Static Network Utilization* (SNU), as it provides a static approximation of (chip router) network utilization.

Utilizing the framework provided by earlier constraints, the number of expected total (local+global) network packets is trivial to optimize for:

$$min\left(\sum_{i,j} s_{ij}\right) \tag{9}$$

Extending this to count only global routes requires the new variables:

$$b_{kj} \in \{0, 1\}$$

Where b_{kj} is 1 if and only if neuron k is used as both output and input on crossbar j. This is realized by the following constraints:

$$\left.\begin{aligned} b_{kj} &\geq s_{kj} + x_{kj} - 1 \\ b_{kj} &\leq s_{kj} \\ b_{kj} &\leq x_{kj} \end{aligned}\right\} \quad i.e.\ b_{ij} = x_{ij} \wedge s_{ij} \tag{10}$$

Minimization of global route count is then achieved by counting the number of total routes and subtracting the number of local routes:

$$min\left(\sum_{i,j} s_{ij} - b_{ij}\right) \tag{11}$$

D. Profile-Guided Optimization for Number of Packets

While SNU minimizes the number of network *routes*, it only approximates the number of network *packets*—the key factor in network congestion and energy use. By leveraging regularities in SNN structure and application behavior [31]–[33], Profile Guided Optimization (PGO) can be used to penalize frequently used routes more than seldom-used ones. The result is an improvement in *average* case execution, targeting anticipated network packet count.

Efficiently implementing PGO within an ILP solver for this task requires both simulator support (for dumping profile data) and architectural support. The following calculations assume the architecture sends only one network packet per crossbar target per neuron fire. Specifically, networking must respect axon sharing: if neuron X targets both neurons Y and Z within crossbar j, only one packet should be generated per spike of X. With this hardware assumption, we introduce the statically known ILP variable W_i representing the profile count of neuron i's spikes during testing. Thus, anticipated chip router traffic (dynamic network utilization) is minimized by the following objective:

$$min\left(\sum_{i,j} s_{ij} \cdot W_i - b_{ij} \cdot W_i\right) \tag{12}$$

This heuristic will not only perform better by prioritizing frequently used routes but also solve faster. Since many neurons never fire within the profile data, their terms are removed from the above heuristic, enabling the solver to converge towards an optimal solution much more quickly.

V. Experimental Results

A. Selection of Networks

To emphasize the need for heterogeneous architectures, we selected practical SNNs with high structural sparsity. These pre-trained SNNs are gathered from recent research [35] identifying properties of particle tracks from high-energy particle collision simulations recorded by next-generation pixel detectors [36]. This topic is of great importance in high-energy physics research and serves as a realistic test case. Sensor data is converted into spike train format for SNN inference. Then, a recent version [37] of Evolutionary Optimization for Neuromorphic Systems (EONS) [38] is used to generate and train SNNs within the TENNLab framework [39]. Finally, we added simulation support for multi-crossbar neuromorphic processors to the framework. Table I summarizes attributes of the networks used hereafter.

TABLE I: Attributes of Networks used in Experimentation

Network	Node Count	Edge Count	Max Fan-In	Edge Density	Sparsity Index [40]	
					Incoming	Outgoing
A	229	464	11	0.0088	0.6889	0.6764
B	257	464	10	0.0070	0.6411	0.6304
C	148	487	15	0.0222	0.5744	0.6067
D	253	499	13	0.0078	0.6431	0.6541
E	150	446	11	0.0198	0.5876	0.6229

TABLE II: Utilized Crossbar Dimensions

Base Dimension	Multi-Macro 2x	Multi-Macro 4x	Multi-Macro 8x
4x4	8x4	16x4	32x4
8x8	16x8	32x8	–
16x16	32x16	–	–
32x32	–	–	–

B. Selection of Crossbar Sizes

Crossbar size choice is critical to practical optimization, as a natural tradeoff between area and SNU exists at differing crossbar sizes. For this study, we assume power-of-two square crossbars from 4x4 to 32x32 as supported by [41]–[43]. We reference the multi-macro vertical stacking technique from [11] to supply rectangular crossbars (we assume our square crossbars have the necessary additional capacity to enable this technique). Crossbars above 32 input channels are excluded, as optimal solutions never included them in preliminary testing. The total set of allowed crossbar dimensions is shown in Table II.

C. Experimental Setup

We tie the TENNLab framework [39] to Google's OR-Tools [44], utilizing the `SAT_INTEGER_PROGRAMMING` solver. Importantly, Google OR-Tools exposes *deterministic* timing results reflecting only the number, type, and complexity of each solver operation. Deterministic timing closely approximates wall clock time if unlimited resources (cores, memory, etc) are present. To evaluate our method independent of scaling potential, we provide only deterministic timing.

Crossbar sizes are either 16x16 (the smallest power-of-two size capable of fitting the most fan-in intense networks from Table I) for homogeneous experiments or pulled from Table II for heterogeneous experiments.

D. Area Comparison

Fig. 2 reports area consumption reduction per utilized solver runtime. Although adding per-unit area overhead is supported, we only consider memristor count to focus on the effectiveness of our method absent of hardware specifics. Four configurations are tested: with MCC or axon-sharing, targeting homogeneous or heterogeneous configurations. Improvement is relative to each network's best result under MCC targeting a homogeneous MCA. SpikeHard was applied repeatedly until convergence was achieved.

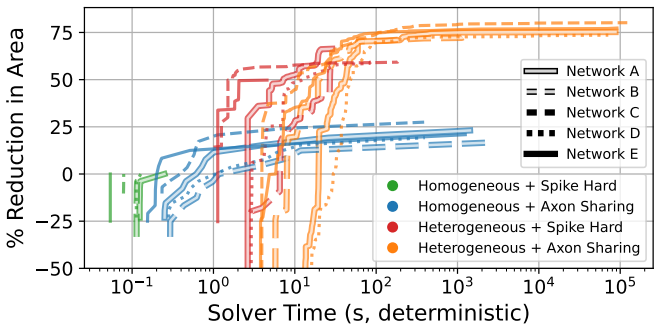
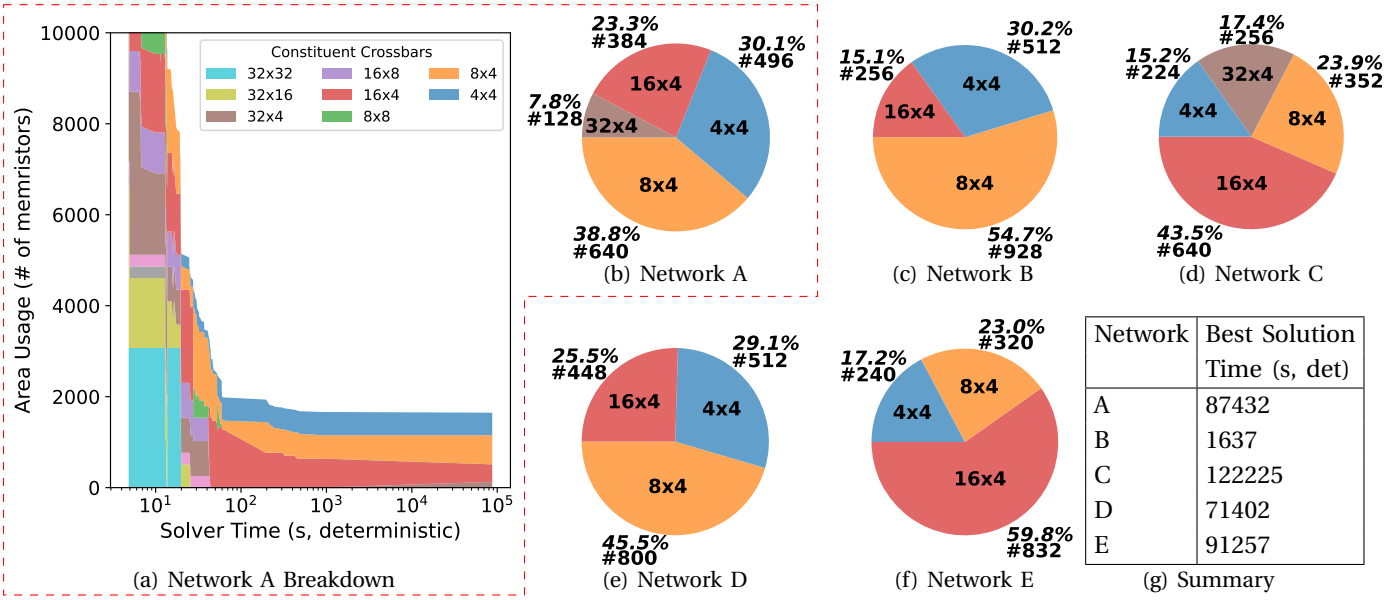


Fig. 2: Relative Improvements in Area Optimization



(a) Network A Breakdown (b) Network A (c) Network B (d) Network C (e) Network D (f) Network E

Network	Best Solution Time (s, det)
A	87432
B	1637
C	122225
D	71402
E	91257

(g) Summary

Fig. 3: Area optimization targeting reasonable heterogeneous architecture: Dimension (In x Out), Area% and #Count
The ILP solver did not terminate for these tasks; the best solutions found within a 5 hour limit are reported

The results indicate that modeling axon sharing reduces area 16.7–27.6% more than SpikeHard for homogeneous MCAs, though 2.5–13.2x additional solver time is needed to break even. A clear preference for heterogeneous MCAs is seen, with axon sharing providing 66.9–72.7% *further* area reduction. Overhead is also lower in the heterogeneous case, requiring just 0.15–3.73x more solver time to break even.

E. Area Breakdown

By plotting every intermediate solution, we explore *how* OR-Tools refines solutions. Fig. 3 shows such results for all networks in the study, focusing on one particular network in subfigure 3a. Although the plot is clipped for early values, preferred crossbar sizes were clearly identified quickly before solutions were slowly refined. Despite subfigure 3g showing high *best* solution times, all networks exhibited the same trend of finding *near-best* solutions quickly. Subfigures 3b–3f summarize the best solutions found. Despite the availability of larger crossbar sizes, a clear trend towards taller crossbars emerged due to the structural sparsity of the input SNNs.

While not explored further in this paper, these lessons could guide further research toward finding optimal solutions more quickly. For example, the iterative swapping approach in [22] is validated with our data.

F. Static Network Utilization

To explore how the SNU optimization from Section IV-C minimizes the number of routes, we took the area-optimal solutions from the previous experiment, restricted the set of enabled crossbars to not increase area, and optimized for SNU. Fig. 5 shows results for the homogeneous case, and Fig. 6 for the heterogeneous case. Both cases show similar improvements: 9.2–26.9% for homogeneous and 11.9–26.4% for heterogeneous. Improvement is reported relative to the most area-optimal solution found by the solver.

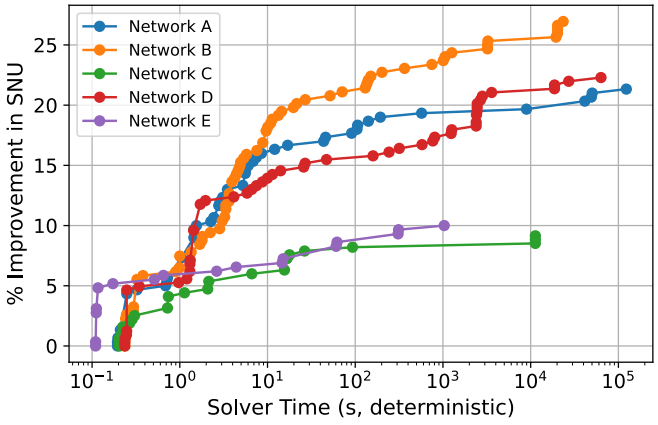


Fig. 5: Optimization of Routes over Already Area-Optimal Solutions for Homogeneous Architecture

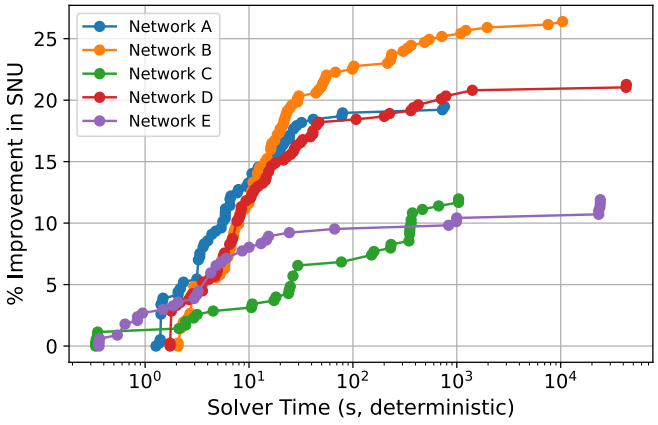


Fig. 6: Optimization of Routes over Already Area-Optimal Solutions for Heterogeneous Architecture

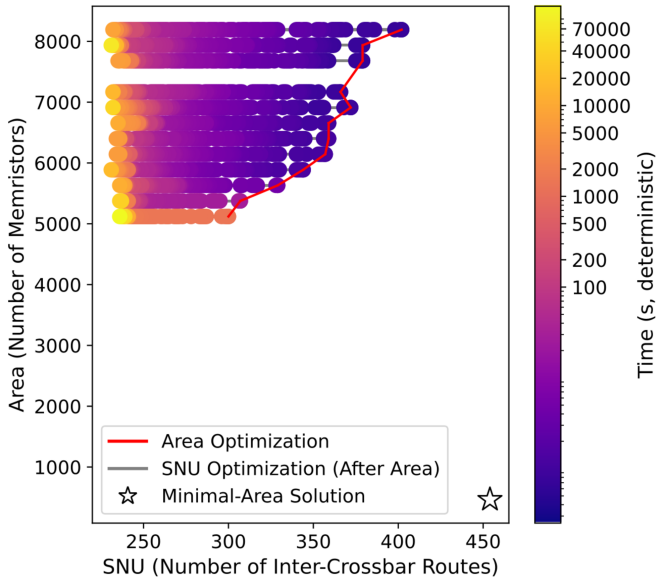


Fig. 7: Area/SNU Evolution for Network A Targeting Homogeneous MCA

G. Area-SNU Evolution

While the area and SNU optimization results from the previous section are promising, they do not fully capture the trade-off between the two optimizations. To illustrate this, we selected one network for further analysis. In Fig. 7, area optimization for the homogeneous architecture was performed, with every intermediate solution forming the basis for SNU optimization. The *total* solver time is reported, along with a mark indicating where a hypothetical minimal-area solution of one neuron per minimally sized crossbar would fall. While not achievable in any target architecture of this study, this point communicates a bound on the solution space.

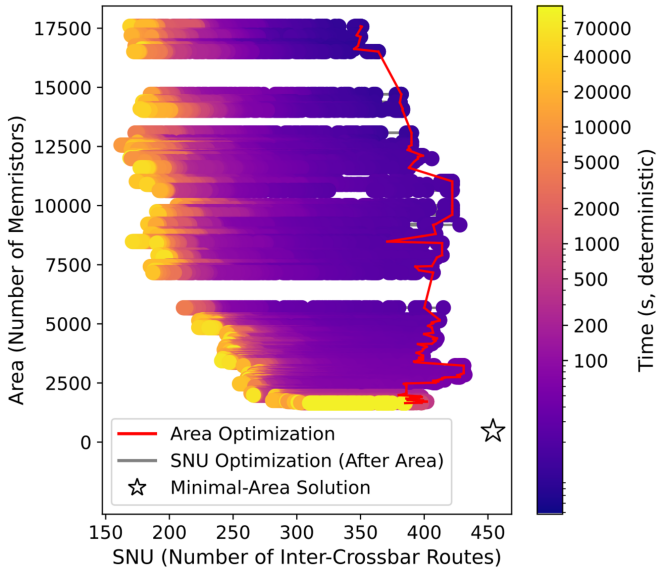


Fig. 8: Area/SNU Evolution for Network A Targeting Heterogeneous MCA

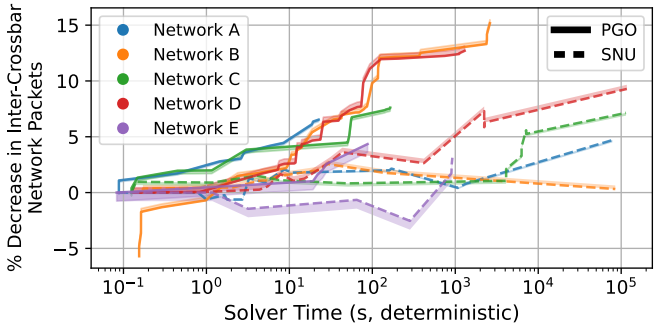


Fig. 9: Profile-Guided vs Static Optimization

Similarly, Fig. 8 shows the results for the heterogeneous case. Although early solutions are less optimal due to added hardware complexity, uniform improvements over the homogeneous case in area, power, and solver time are made quickly. At the optimization limit, a trade-off emerges between the two metrics; this trend is sensitive to the target architecture and consistent across all networks in this study.

H. Dynamic Network Utilization

This final experiment briefly showcases the profile-guided version of SNU. Instead of minimizing the number of routes, the number of network packets are minimized according to spike profile data. This data includes SmartPixel simulations of high-energy particle collisions [36]—the same data used to train and evaluate the networks in this study. A randomly-selected 1% sample of the data (51MB) was used for PGO, with optimization results shown in Fig. 9. This figure includes error bands indicating spike count under execution of the other 99% of the data (5.0GB) within the same application.

The reported results indicate 0.5–14.8% decrease in spike count compared to the best SNU-optimized networks while requiring 1–3 orders of magnitude less solver time. Additionally, due to the low error, the results confirm that spiking activity is regular enough to benefit from PGO.

VI. Conclusion

This paper addresses growing sparsity in Spiking Neural Networks (SNNs) and the resulting opportunity to reduce area consumption through heterogeneous Memristor Crossbar Architectures (MCAs). By developing Integer Linear Programming (ILP) formulations supporting heterogeneously sized crossbars and optimizing axonal interactions, we significantly outperform previous methods in area efficiency. We show a 16.7–27.6% reduction in area for homogeneous MCAs and a substantial 66.9–72.7% *further* reduction for heterogeneous MCAs. Additionally, we introduce an optimization to minimize inter-crossbar routing, achieving an 11.9–26.4% reduction without increasing area. Finally, we propose a profile-guided approach to reduce inter-crossbar spike count 0.5–14.8% more than the best route-minimized solutions, while requiring 1–3 orders of magnitude less solver time. These contributions demonstrate the potential of heterogeneous MCAs in SNN acceleration, enabling progress toward more efficient and scalable neuromorphic hardware.

References

[1] Y. Irizarry-Valle and A. C. Parker, "An astrocyte neuromorphic circuit that influences neuronal phase synchrony," *IEEE transactions on biomedical circuits and systems*, vol. 9, no. 2, p. 175—187, April 2015.

[2] A. Tavanaei, M. Ghodrati, S. R. Kheradpisheh, T. Masquelier, and A. Maida, "Deep learning in spiking neural networks," *Neural networks: the official journal of the International Neural Network Society*, vol. 111, p. 47—63, March 2019.

[3] P. Blouw, X. Choo, E. Hunsberger, and C. Eliasmith, "Benchmarking keyword spotting efficiency on neuromorphic hardware," in *Proceedings of the 7th annual neuro-inspired computational elements workshop*, 2019, pp. 1–8.

[4] N. Getty, T. Brettin, D. Jin, R. Stevens, and F. Xia, "Deep medical image analysis with representation learning and neuromorphic computing," *Interface Focus*, vol. 11, no. 1, p. 20190122, 2021.

[5] R. Shukla, M. Lipasti, B. Van Essen, A. Moody, and N. Maruyama, "Remodel: rethinking deep CNN models to detect and count on a neurosynaptic system," *Frontiers in neuroscience*, vol. 13, p. 4, 2019.

[6] M. Hu *et al.*, "Memristor crossbar-based neuromorphic computing system: A case study," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, no. 10, pp. 1864–1878, 2014.

[7] M. A. Zidan, J. P. Strachan, and W. D. Lu, "The future of electronics based on memristive systems," *Nature Electronics*, vol. 1, no. 1, pp. 22–29, 2018.

[8] J. Yue *et al.*, "15.2 a 2.75-to-75.9TOPS/W computing-in-memory NN processor supporting set-associate block-wise zero skipping and ping-pong cim with simultaneous computation and weight updating," in *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 64, 2021, pp. 238–240.

[9] J. Yue *et al.*, "14.3 a 65nm computing-in-memory-based CNN processor with 2.9-to-35.8TOPS/W system energy efficiency using dynamic-sparsity performance-scaling architecture and energy-efficient inter/intra-macro data reuse," in *2020 IEEE International Solid-State Circuits Conference - (ISSCC)*, 2020, pp. 234–236.

[10] Z. Chen, X. Chen, and J. Gu, "15.3 a 65nm 3T dynamic analog RAM-based computing-in-memory macro and CNN accelerator with retention enhancement, adaptive analog sparsity and 44TOPS/W system energy efficiency," in *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 64, 2021, pp. 240–242.

[11] S. Kim *et al.*, "Neuro-CIM: ADC-less neuromorphic computing-in-memory processor with operation gating/stopping and digital–analog networks," *IEEE Journal of Solid-State Circuits*, vol. 58, no. 10, pp. 2931–2945, 2023.

[12] Y. Shang, Y. Li, F. You, and R. Zhao, "Conversion-based approach to obtain an SNN construction," *International Journal of Software Engineering and Knowledge Engineering*, vol. 30, no. 11n12, pp. 1801–1818, 2020.

[13] J. Shen *et al.*, "ESL-SNNs: An evolutionary structure learning strategy for spiking neural networks," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, no. 1, pp. 86–93, Jun. 2023.

[14] Y. Li, F. Zhao, D. Zhao, and Y. Zeng, "Directly training temporal spiking neural network with sparse surrogate gradient," *Neural Networks*, vol. 179, p. 106499, 2024.

[15] H. Markram, W. Gerstner, and P. J. Sjöström, "A history of spike-timing-dependent plasticity," *Frontiers in synaptic neuroscience*, vol. 3, p. 4, 2011.

[16] J. D. Nunes, M. Carvalho, D. Carneiro, and J. S. Cardoso, "Spiking neural networks: A survey," *IEEE Access*, vol. 10, pp. 60 738–60 764, 2022.

[17] Z. Shen, S. Zhang, P. Zheng, and Y. Huang, "NeuroEvoSparse: A biologically plausible framework for efficient," in *2023 20th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP)*, 2023, pp. 1–5.

[18] Y. Gong, T. Chen, S. Wang, S. Duan, and L. Wang, "Lightweight spiking neural network training based on spike timing dependent backpropagation," *Neurocomputing*, vol. 570, p. 127059, 2024.

[19] B. Han, F. Zhao, W. Pan, and Y. Zeng, "Adaptive sparse structure development with pruning and regeneration for spiking neural networks," *Information Sciences*, vol. 689, p. 121481, 2025.

[20] C.-x. Li, S. Zhu, X. Hu, M. Dou, and S. Xiong, "Block-clustering on neural networks for large-scale memristor-based implementation," in *2021 International Conference on Neuromorphic Computing (ICNC)*. IEEE, 2021, pp. 373–385.

[21] D. Bhattacharjee, Y. Tavva, A. Easwaran, and A. Chattopadhyay, "Crossbar-constrained technology mapping for ReRAM based in-memory computing," *IEEE Transactions on Computers*, vol. 69, no. 5, pp. 734–748, 2020.

[22] A. Balaji *et al.*, "Mapping spiking neural networks to neuromorphic hardware," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 1, pp. 76–86, 2019.

[23] A. Ankit, A. Sengupta, and K. Roy, "TraNNsformer: Neural network transformation for memristive crossbar based neuromorphic system design," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2017, pp. 533–540.

[24] J. Clair, G. Eichler, and L. P. Carloni, "SpikeHard: Efficiency-driven neuromorphic hardware for heterogeneous systems-on-chip," *ACM Trans. Embed. Comput. Syst.*, vol. 22, no. 5s, sep 2023.

[25] C. D. Schuman *et al.*, "Opportunities for neuromorphic computing algorithms and applications," *Nature Computational Science*, vol. 2, no. 1, pp. 10–19, 2022.

[26] L. Deng *et al.*, "Comprehensive SNN compression using ADMM optimization and activity regularization," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 34, no. 6, pp. 2791–2805, 2023.

[27] A. Kugele, T. Pfeil, M. Pfeiffer, and E. Chicca, "Efficient processing of spatio-temporal data streams with spiking neural networks," *Frontiers in Neuroscience*, vol. 14, 2020.

[28] B. Rueckauer, I.-A. Lungu, Y. Hu, M. Pfeiffer, and S.-C. Liu, "Conversion of continuous-valued deep networks to efficient event-driven networks for image classification," *Frontiers in Neuroscience*, vol. 11, 2017.

[29] Y. Cao, Y. Chen, and D. Khosla, "Spiking deep convolutional neural networks for energy-efficient object recognition," *International Journal of Computer Vision*, vol. 113, pp. 54–66, 2015.

[30] Y. Chen, "ReRAM: History, status, and future," *IEEE Transactions on Electron Devices*, vol. 67, no. 4, pp. 1420–1433, 2020.

[31] D. J. Amit and G. Mongillo, "Spike-driven synaptic dynamics generating working memory states," *Neural Computation*, vol. 15, no. 3, pp. 565–596, 2003.

[32] E. Pastorelli *et al.*, "Scaling of a large-scale simulation of synchronous slow-wave and asynchronous awake-like activity of a cortical model with long-range interconnections," *Frontiers in Systems Neuroscience*, vol. 13, p. 33, 2019.

[33] J. Bartram *et al.*, "Parallel reconstruction of the excitatory and inhibitory inputs received by single neurons reveals the synaptic basis of recurrent spiking," *bioRxiv*, pp. 2023–01, 2023.

[34] J. Lin, Z. Zhu, Y. Wang, and Y. Xie, "Learning the sparsity for ReRAM: Mapping and pruning sparse neural network for ReRAM based accelerator," in *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, 2019, pp. 639–644.

[35] S. R. Kulkarni *et al.*, "On-sensor data filtering using neuromorphic computing for high energy physics experiments," in *Proceedings of the 2023 International Conference on Neuromorphic Systems*, ser. ICONS '23. New York, NY, USA: Association for Computing Machinery, 2023.

[36] J. Yoo *et al.*, "Smart pixel sensors: towards on-sensor filtering of pixel clusters with deep learning," *Machine Learning: Science and Technology*, vol. 5, no. 3, p. 035047, 2024.

[37] C. D. Schuman, J. P. Mitchell, R. M. Patton, T. E. Potok, and J. S. Plank, "Evolutionary optimization for neuromorphic systems," in *Proceedings of the 2020 Annual Neuro-Inspired Computational Elements Workshop*, 2020, pp. 1–9.

[38] C. D. Schuman, J. S. Plank, A. Disney, and J. Reynolds, "An evolutionary optimization framework for neural networks and neuromorphic architectures," in *2016 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2016, pp. 145–154.

[39] J. S. Plank, C. D. Schuman, G. Bruer, M. E. Dean, and G. S. Rose, "The TENNLab exploratory neuromorphic computing framework," *IEEE Letters of the Computer Society*, vol. 1, no. 2, pp. 17–20, 2018.

[40] S. Goswami, C. Murthy, and A. K. Das, "Sparsity measure of a network graph: Gini index," *Information Sciences*, vol. 462, pp. 16–39, 2018.

[41] H. Zhou, S. Li, K.-W. Ang, and Y.-W. Zhang, "Recent advances in in-memory computing: exploring memristor and memtransistor arrays with 2D materials," *Nano-Micro Letters*, vol. 16, no. 1, p. 121, 2024.

[42] Y. Huang *et al.*, "Memristor-based hardware accelerators for artificial intelligence," *Nature Reviews Electrical Engineering*, pp. 1–14, 2024.

[43] F. Aguirre *et al.*, "Hardware implementation of memristor-based artificial neural networks," *Nature communications*, vol. 15, no. 1, p. 1974, 2024.

[44] L. Perron and F. Didier, "OR-Tools CP-SAT v9.11," 2023.