

CIM-Tuner: Balancing the Compute and Storage Capacity of SRAM-CIM Accelerator via Hardware-mapping Co-exploration

Jinwu Chen^{1,2}, Yuhui Shi¹, He Wang¹, Zhe Jiang^{1,2}, Jun Yang^{1,2}, Xin Si^{1,2*} and Zhenhua Zhu^{3*}

¹School of Integrated Circuits, Southeast University, China ²National Center of Technology Innovation for EDA, China

³Department of Electronic Engineering, Tsinghua University

Email: chenjinwu@seu.edu.cn, 220236525@seu.edu.cn, wanghe020608@gmail.com, zhejiang.arch@gmail.com, dragon@seu.edu.cn, xinsi@seu.edu.cn, zhuzhenhua@mail.tsinghua.edu.cn

* Corresponding authors.

Abstract—As an emerging type of AI computing accelerator, SRAM Computing-In-Memory (CIM) accelerators feature high energy efficiency and throughput. However, various CIM designs and under-explored mapping strategies impede the full exploration of compute and storage balancing in SRAM-CIM accelerator, potentially leading to significant performance degradation. To address this issue, we propose CIM-Tuner, an automatic tool for hardware balancing and optimal mapping strategy under area constraint via hardware-mapping co-exploration. It ensures universality across various CIM designs through a matrix abstraction of CIM macros and a generalized accelerator template. For efficient mapping with different hardware configurations, it employs fine-grained two-level strategies comprising accelerator-level scheduling and macro-level tiling. Compared to prior CIM mapping, CIM-Tuner’s extended strategy space achieves $1.58\times$ higher energy efficiency and $2.11\times$ higher throughput. Applied to SOTA CIM accelerators with identical area budget, CIM-Tuner also delivers comparable improvements. The simulation accuracy is silicon-verified and CIM-Tuner tool is open-sourced at <https://github.com/champloo2878/CIM-Tuner.git>.

Index Terms—SRAM Computing-In-Memory, DNN Accelerator, Hardware-mapping Co-exploration

I. INTRODUCTION

SRAM Computing-in-Memory (SRAM-CIM) has garnered significant attention as a highly energy-efficient matrix multiplication engine. By enabling data movement with shorter distances and larger bandwidth, alongside customized designs for data precision and sparsity, SRAM-CIM achieves notable improvements in both energy and throughput. These advantages are validated at both the macro level [5]– [9] and accelerator level [10]– [17], positioning SRAM-CIM as a compelling alternative to digital computing arrays.

Current SRAM-CIM accelerator optimizations primarily target CIM computation itself, neglecting the compute and storage resources balancing and corresponding mapping strategy matching. However, this neglect may significantly degrade the inference performance of these DNN accelerators. Back to the conventional digital accelerators, the size of the on-chip buffer has a significant impact. Figure 1 shows the performance of a systolic array under different compute and storage allocations of fixed area budget. It demonstrates that

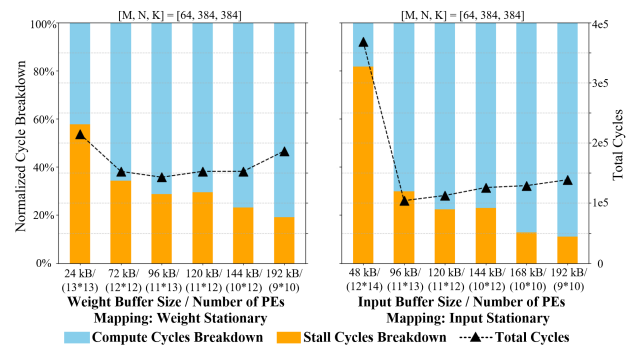


Fig. 1. Latency breakdown of matrix multiplication on systolic accelerators. Evaluated using scale-sim [1]. Left: varying weight buffer in weight stationary mapping. Right: varying input buffer in input stationary mapping.

stall cycles decrease as the buffer size increases, revealing that larger buffer leverages data locality better and improves data reuse to alleviate external bandwidth pressure. However, an oversized buffer inevitably reduces the area available for the computation array, leading to lower computation throughput and an overall latency increase.

For SRAM-CIM based accelerator, due to tightly coupled compute and storage units, exploring the compute and storage balancing demands more fine-grained modeling and mapping analysis compared to conventional digital arrays. Such exploration requires delving into specific SRAM-CIM macro designs. Previous optimization works like AutoDCIM [18] focused only on analyzing the storage-compute balance in a specific digital CIM macro template, lacking support for other CIM types and accelerator-level optimizations. To address this gap, these two challenges must be tackled: (1) **Adaptation to various designs:** Diverse SRAM-CIM implementations vary in data precision, accuracy, and compute mechanisms (e.g., bit-cell design, multi-bit expansion method, simultaneous compute/update capability and so on). This diversity increases the complexity of the problem and may push any solution towards being highly design-specific. (2) **Underexplored mapping strategies:** Different storage and compute ratio require

distinct optimal mapping strategies for accelerators. As a novel architecture, SRAM-CIM accelerator lacks comprehensive exploration of matrix multiplication mapping. Especially as current neural network operators vastly exceed on-chip buffer capacity, traditional static mapping proves inefficient, necessitating dynamic and fine-grained strategies to account for frequent parameter updating.

To address these challenges, we propose CIM-Tuner: an automated and open-sourced tool for balancing compute and storage capacity in SRAM-CIM accelerators. Within given CIM macro configurations and network workloads, CIM-Tuner simultaneously determines the optimal hardware sizing and operators mapping under area budget constraint and optimization target. The contributions include:

- First, we introduce a matrix abstraction for SRAM-CIM macros and establish a universal accelerator template to decouple diverse circuit implementations from architectural exploration. Unified evaluation formulas adaptable to any SRAM-CIM design are provided.
- Second, we propose the fine-grained two-level mapping strategies for better hardware resource exploration: **At the accelerator level**, both spatial and temporal scheduling methods are proposed, encompassing the mapping strategies introduced in previous works. **At the macro level**, novel Accumulation-First (AF) tiling and Parallel-First (PF) tiling strategies exploiting CIM-specific **storage-compute ratio (SCR)** to optimize CIM utilization while expanding the mapping strategy design space.
- Finally, we explore the hardware-mapping co-design space via simulated annealing. To accelerate exploration, the hardware design space is heuristically pruned and the layers with same size are gathered to reduce the mapping choices. Experimental results demonstrate that our expanded mapping strategy achieves $1.58\times$ higher energy efficiency and $2.11\times$ higher throughput compared to previous CIM mapping methodologies under the same co-exploration process. Also, evaluated on SOTA CIM accelerators [10], [16] under identical area budgets, CIM-Tuner delivers comparable improvements of $1.34\sim 2.31\times$ in energy efficiency and $1.03\sim 2.88\times$ in throughput.

The simulation accuracy of CIM-Tuner tool is silicon-verified and CIM-Tuner is open-sourced for the purpose of better early stage SRAM-CIM accelerator design decisions.

II. BACKGROUND AND MOTIVATION

A. SRAM-CIM Basis and Storage-Compute Ratio

Various CIM technologies leverage different types of memory. Among them, SRAM features high access speed, nearly unlimited endurance and seamless integration with advanced CMOS technology. Thus, SRAM-CIM has been widely researched and utilized in many DNN accelerators [10]– [17]. Typically, SRAM-CIM operates in two modes: compute and weight update. During compute, input vectors are projected into the CIM to perform multiply-accumulate (MAC) operations with stored weights. The update mode resembles conventional memory write operations.

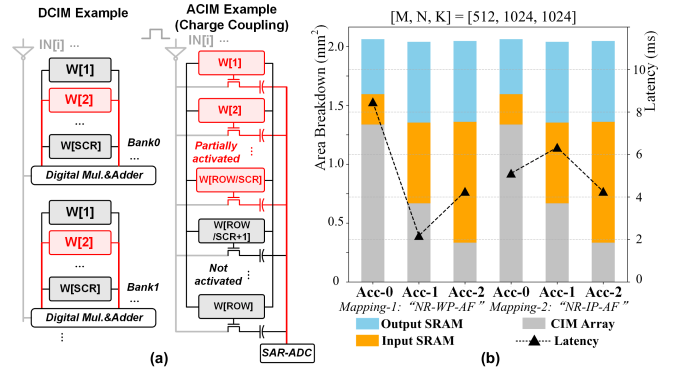


Fig. 2. (a) The SCR mechanism in both digital and analog CIM. (b) The latency of matrix multiplication across different hardware proportions and mapping strategies. See evaluation setup in Section IV.

As shown in Figure 2 (a), the storage-compute ratio (*SCR*) in CIM is defined as the ratio of memory cells to compute units within a CIM macro. In digital CIMs (DCIMs), multiple memory cells typically share one set of multipliers and adders within a local bank to maintain storage density and layout compactness [9]. Here, *SCR* equals the number of cells per local bank. For analog CIMs (ACIMs), *SCR* represents the ratio of total column cells to activated cells during computation, ensuring sufficient compute-line signal margin [17]. In SRAM-CIM accelerators, weight data is typically stored in the CIM array. Input data from different channels reuses weights, generating sums for different output channels.

B. Motivation for Compute and Storage Balancing

Currently, with the explosive growth in model parameters, the size of a single matrix multiplication operator (\sim MB level) often far exceeds the SRAM capacity of the on-chip SRAM-CIM accelerator (\sim KB level) [2]– [4]. This requires frequent updating of input data, weight, and output data during computation. Therefore, under limited area budget, selecting the correct ratio of compute and storage capacity, as well as the fine-grained mapping strategies, is particularly crucial. As shown in Figure 2 (b), assigning excessive compute capacity may lead to significant cycles wasted on data movement. Conversely, allocating excessive storage capacity can result in low computation throughput, also causing high latency. More than $4\times$ worse latency is observed in our motivation demonstration. Furthermore, different mapping strategies observed under the same hardware configuration may yield vastly different performance. In Figure 2, the input-priority update strategy demonstrates totally different latency variation with the weight-priority update strategy under the same hardware. This demonstrates that both the compute and storage balancing and mapping strategies matching can severely impact the performance of CIM accelerators.

However, existing SRAM-CIM accelerator designs [10]– [17] mainly focuses on optimizing computation paths leveraging techniques such as data sparsity, mixed-precision computation, and operator fusion. Meanwhile, critical decisions

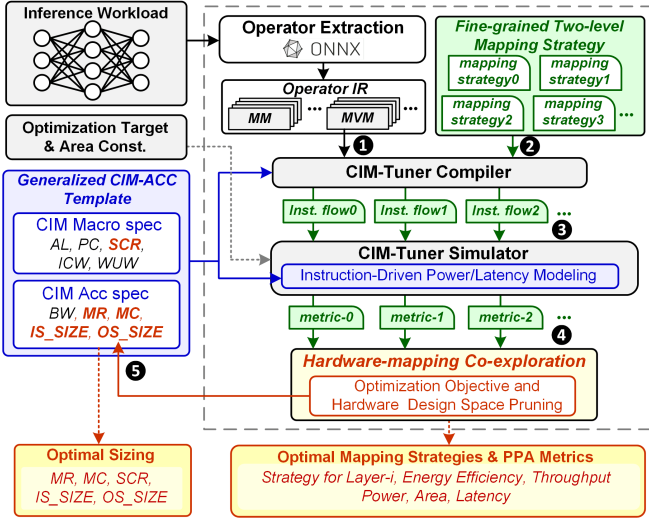


Fig. 3. Overview of CIM-Tuner’s hardware-mapping co-exploration.

concerning the size of on-chip buffers and the storage-compute ratio of CIM macros remain largely reliant on designers’ empirical intuition. This ad hoc approach often fails to achieve optimal system performance. Moreover, previous CIM mapping strategy [19] is under-explored, leaving the opportunity for higher performance under various hardware configurations. Therefore, this gap in both hardware balancing and mapping strategy matching motivates us to propose CIM-Tuner, an automated tool for optimal hardware sizing and mapping strategies under given macro configuration and area budget.

III. CIM-TUNER FRAMEWORK

A. Overview of CIM-Tuner

Figure 3 illustrates the workflow of CIM-Tuner, where mapping exploration operates as a sub-process of hardware exploration. For mapping exploration, first, target workload operators are represented through intermediate representations (IR) to extract matrix dimensions. Second, fine-grained two-level operator mapping strategies are explored. These strategies are converted into unified instruction flows via the CIM-Tuner compiler. Third, using the CIM-Tuner simulator’s instruction-driven power and latency models, cycle-accurate performance and power consumption simulations are performed for different mapping strategies. This enables selection of the optimal mapping strategy and generates PPA metrics.

Outside the mapping, the hardware exploration can be easily conducted by modifying the hardware configuration, with the corresponding notations detailed in Table I.

B. Matrix Abstraction and Generalized Accelerator Template

Matrix Abstraction of CIM Macro. To circumvent the complex taxonomy of SRAM-CIM micro-architectures, we abstract CIM Macros as a mathematical matrix, as shown in Figure 4. We observe that all SRAM-CIM variants fundamentally perform the same atomic operation: a vector-matrix projection between an input vector of accumulation length

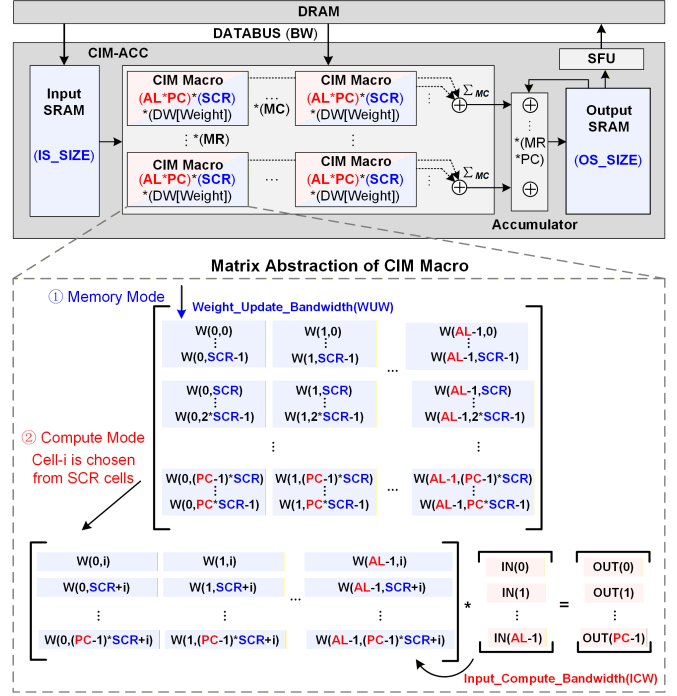


Fig. 4. Generalized three-stage pipelined CIM accelerator (top) and proposed matrix abstraction of SRAM-CIM macro (bottom).

TABLE I
NOTATIONS OF GENERALIZED ACCELERATOR TEMPLATE

Macro Parameters	Notation	Acc. Parameters	Notation
Accumulation Length	AL	Macro Cols	MC
Parallel Channel	PC	Macro Rows	MR
Storage-Compute Ratio	SCR	Bus Bandwidth	BW
Input-Compute Bandwidth	ICW	Input SRAM	IS_SIZE
Weight-Update Bandwidth	WUW	Output SRAM	OS_SIZE

(AL) and a weight matrix of $AL \times$ parallel channel (PC) stored in SRAM-CIM, generating a partial sum vector of length PC . The Storage-Compute Ratio (SCR) mechanism dynamically selects one from SCR available $AL \times PC$ weight matrices for computation. Besides, to standardize computation and weight update latencies across various CIM designs, CIM-Tuner introduces two key parameters: (1) Input computing bandwidth (ICW) representing processable input data bits per cycle. (2) Weight update bandwidth (WUW) representing updated weight data bits per cycle.

These two variables enable the characterization of data input mechanism across CIM implementations. For example, in DCIM, ICW typically correlates with the number of input bitline of one local computing unit, whereas in ACIM, ICW is generally associated with the Digital-to-Analog Converter (DAC) precision of one analog input driver:

$$ICW_{DCIM} = AL \times N_{InputBitline} \quad (1)$$

$$ICW_{ACIM} = AL \times Precision_{DAC} \quad (2)$$

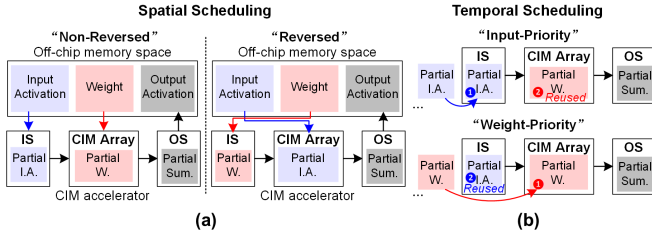


Fig. 5. Accelerator-level scheduling strategies, IS and OS are short for Input SRAM and Output SRAM, respectively: (a) Non-Reversed (NR) and Reversed (R) spatial scheduling, (b) Input-Priority (IP) and Weight-Priority (WP) temporal scheduling.

The latency of single CIM computation is:

$$Latency_{DCIMCompute} = \frac{Datawidth[Input]}{N_{InputBitline}} \times \frac{1}{freq} \quad (3)$$

$$Latency_{ACIMCompute} = \frac{Datawidth[Input]}{Precision_{DAC}} \times \frac{1}{freq} \quad (4)$$

The WUW is associated with the specific bitcell design and multi-bit extension method. The latency of single CIM update can be characterized by:

$$Latency_{update} = \frac{AL \times Datawidth[Weight]}{WUW} \times \frac{1}{freq} \quad (5)$$

$freq$ represents the operating clock frequency. Thus the matrix abstraction bridges hardware design specifics and element-wise arithmetic operations. By incorporating the reported energy efficiency, area, and frequency, various SRAM-CIM designs can be accurately modeled. Compared with previous fixed macro design search [18], matrix abstraction effectively supports complex circuit implementation details.

Generalized Accelerator Template. Unlike traditional digital computing arrays that utilize scratchpad memory to distribute and manage input, weight, and output data, SRAM-CIM accelerators typically follow a three-stage pipeline: (1) Buffering input data in Input SRAM, (2) Storing weights and computing in CIMs, and (3) Accumulating and buffering partial sums in Output SRAM. Consequently, we constructed the Generalized Accelerator Template shown in Figure 4 and extracted key parameters for compute-storage balancing. The size of the input SRAM and output SRAM are denoted as (IS_SIZE) and (OS_SIZE) respectively. The CIMs array is organized in a grid format of macro rows (MR) by macro columns (MC). Outputs are accumulated along the row direction, while inputs are broadcast along the column direction. Besides, the accelerator has a data transfer bandwidth to external memory of Bandwidth (BW) bits per cycle.

C. Fine-grained Two-level Mapping Strategy

For accelerator-level scheduling, we synthesize common strategies from previous CIM mapping works, encompassing both spatial and temporal approaches. As shown in Figure 5 (a), for spatial scheduling, CIM-Tuner offers two options determining whether input activation or weight data should be

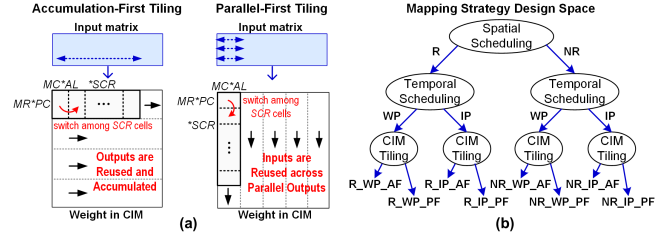


Fig. 6. (a) Accumulation-first (AF) tiling and parallel-first (PF) tiling during CIM computation. (b) Total mapping strategy design space.

stored as CIM weights. We divide spatial scheduling into Non-Reversed (NR) and Reversed (R), where NR denotes storing the input activation in Input SRAM and R denotes storing the activation in CIM array. These correspond to the weight stationary and input stationary dataflow referenced in previous work [19]. Meanwhile, previous temporal scheduling strategy typically follows the conventional input-priority update (IP) approach, as shown in Figure 5 (b), where input SRAM is updated first to maximize data reuse in CIMs. CIM-Tuner introduces an alternative temporal scheduling strategy called weight-priority update (WP). WP prioritizes updating data in SRAM-CIM to enhance data reuse of input SRAM.

For macro-level tiling, CIM-Tuner proposes two novel tiling strategies: accumulation-first tiling (AF) and parallel-first tiling (PF), in order to choose the optimal SCR of CIM macro. As illustrated in Figure 6 (a), The AF strategy prioritizes mapping SCR CIM data blocks within the same output channel. This approach allows partial sums (Psum) generated in consecutive cycles to be reused and accumulated, though requiring distinct input vectors. Conversely, the PF tiling prioritizes mapping SCR blocks within the same input channel. This benefits from reusing input vectors across consecutive cycles but necessitates temporarily storing different partial sums in the output SRAM. These partial sums are later fetched and accumulated. The AF and PF strategies capture the inherent trade-off between CIM input and output, thereby expanding the overall mapping strategy space and unlocking further optimization potential. The mapping strategy space comprising scheduling and tiling is shown in Figure 6 (b).

D. Optimization Objective and Hardware Space Pruning

Since fine-grained mapping strategies are explored for each operator, the strategy space for the entire network grows exponentially with the number of operators. To address this challenge, our hardware-mapping co-exploration is designed to be operator-size-aware. This allows operators of the same size to be merged, significantly reducing the strategy space that the network must handle.

When considering practical hardware implementation constraints, the continuous-valued hardware design space can also be pruned. First, to align with the address decoding, key hardware parameters including SCR , IS_SIZE , and OS_SIZE must be constrained to powers of 2. Second, to ensure efficient utilization of the accelerator’s external bandwidth, hardware

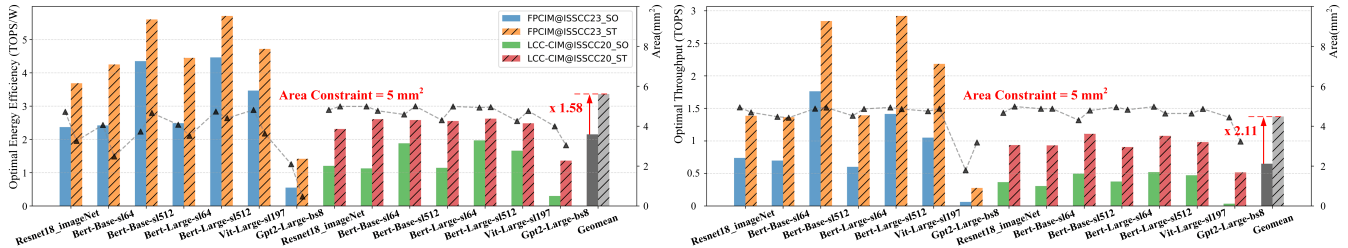


Fig. 7. Comparison with previous CIM mapping in [19]. Optimal metrics are obtained through identical hardware-mapping co-exploration under same area budget of 5 mm^2 . SO stands for spatial scheduling only [19] and ST stands for both scheduling and tiling, i.e., mapping strategies in CIM-Tuner.

designs where internal bandwidth (either *ICW* or *WUW*) falls below the specified *BW* are eliminated.

IV. EXPERIMENT

A. Experiment Setup

For the hardware template, we first implemented the proposed parameterized accelerator template using Verilog HDL. In this implementation, the CIM Macro is replaced with a parameterized behavior model. Next, the area and instruction power of several parameter configurations under the 28nm technology node can be obtained using DC Synthesize and Prime Time PX (PTPX) power analysis tools. Within the simulator, linear programming was applied to fit an instruction-level power model and the corresponding area model. To handle different CIM designs, we performed conversions in the simulator based on equivalent formulas from Matrix Abstraction. This approach ensures the adaptability and normalizes all performance metrics to the same technique node.

Regarding the mapping strategy, we developed the CIM-Tuner Compiler in C language, which compiles different mapping strategies into instruction flows based on the generalized accelerator template. A validation script is also provided to verify the functional correctness of these compiled instruction flows. For the co-exploration, we implemented the CIM-Tuner Simulator and simulated annealing algorithm in Python. The simulator analyzes instruction flows to assess the performance of the hardware accelerator across different operators, determining the optimal mapping strategy. Through the simulated annealing algorithm, hardware configurations are iteratively adjusted, enabling a co-exploration process that ultimately achieves an optimal hardware-mapping co-design.

B. Mapping Strategy Comparison

To demonstrate the effectiveness of our mapping strategy, we selected SOTA CIM optimization work [19] as the baseline. In [19], three optimizations for CIM accelerators are proposed: (1) selecting weight/input stationary mapping strategies; (2) adjusting macro dimensions; and (3) modifying the macros floorplan. Here, its mapping strategy corresponds to the spatial scheduling in CIM-Tuner. The latter two hardware modifications can be fully addressable through our hardware parameters exploration. Thus, for reproducing [19], we utilized the same CIM-Tuner co-exploration framework but restricted

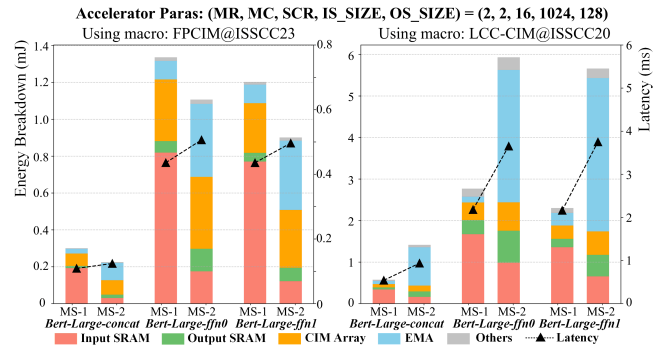


Fig. 8. Energy breakdown for Bert [4] operators under different CIM designs and Mapping Strategies. MS-1 stands for “NR-IP-AF” strategy and MS-2 stands for “NR-IP-PF” strategy.

the mapping strategy to spatial scheduling. As shown in Figure 7, across experiments on seven networks, CIM-Tuner demonstrated an average energy efficiency and throughput improvement of $1.58\times$ and $2.11\times$ compared to the baseline.

Moreover, to analyze the gains of tiling strategy, a breakdown study is conducted on three matrix multiplication operators from the Bert-large [4] network across two CIM macros. The accelerator parameters (*MR*, *MC*, *SCR*, *IS_SIZE*, *OS_SIZE*) are fixed at (2, 2, 16, 1024, 128). We selected “NR-IP-AF” and “NR-IP-PF” mapping strategies for comparison to demonstrate the impact of AF and PF strategies. As shown in Figure 8 and consistent with our earlier analysis, the AF strategy reduces Output SRAM overhead at the cost of increased Input SRAM overhead, whereas the PF strategy exhibits the opposite behavior. Given the limited 128 kB Output SRAM capacity, the PF strategy requires more external memory access (EMA) to store partial sums, significantly increasing the energy consumption. Compared to FPCIM, LCC-CIM’s shorter accumulation length generates more partial sums for the same operator, resulting in more severe EMA penalties.

C. Improvement on SOTA accelerators

To better and more directly demonstrate CIM-Tuner’s ability to adjust the compute and storage capacity of the accelerator, we instantiated two state-of-the-art SRAM-CIM accelerators, TranCIM [10] and TP-DCIM [16]. We extracted their corresponding CIM macro configurations and accelerator template

TABLE II
CIM-TUNER APPLIED ON SOTA ACCELERATORS

Name	(<i>MR</i> , <i>MC</i> , <i>SCR</i> , <i>IS_SIZE</i> , <i>OS_SIZE</i>)	Energy Eff.(TOPS/W)	Throughput(GOPS)	Area(mm ²)	Improve
TranCIM-Base	(3, 1, 1, 64, 128)	2.54	1002.3	3.52	/
TranCIM-EE.	(2, 1, 16, 4, 64)	3.40	734.3	2.99	× 1.34
TranCIM-Th.	(3, 1, 2, 2, 32)	3.05	1028.9	3.51	× 1.03
TP-DCIM-Base	(2, 4, 1, 16, 16)	1.89	460.9	2.23	/
TP-DCIM-EE.	(3, 2, 4, 2, 2)	4.36	1324.8	2.04	× 2.31
TP-DCIM-Th.	(3, 2, 4, 4, 256)	4.25	1326.7	2.16	× 2.88

*EE. is for optimal energy efficiency and Th. is for optimal throughput. Other hardware parameters are fixed during exploration.

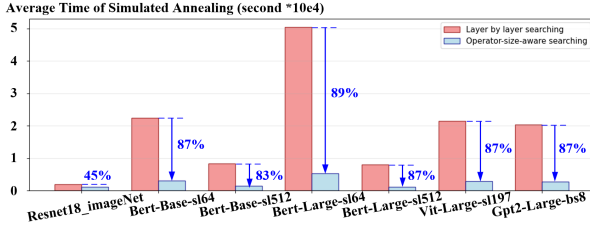


Fig. 9. Runtime illustration of simulated annealing.

parameters to serve as the baseline. We deliberately ignored their computational optimizations, focusing exclusively on the storage-compute ratio aspect of the accelerators.

As shown in Table II, we selected Bert-Large as the workload and performed co-exploration under different optimization targets. The area budgets are fixed as the baseline area. After co-exploration, the target computational capacity parameterized as *MR* and *MC*, along with storage capacity parameterized as *SCR*, *IS_SIZE*, and *OS_SIZE*, are all adjusted. These adjustments demonstrated significant improvements. This outcome proves that the original accelerator design’s storage-compute ratio allocation has room for enhancement, and the optimal mapping strategies for different operators across network should be modified accordingly.

We note that the performance metrics here show notable differences from the original accelerators’ results. This divergence arises because the original metrics incorporated various computational optimizations and were tested under extreme conditions like low voltage and frequency. In contrast, CIM-Tuner’s accelerator performance relies on our generalized template verification at standard test conditions. The power accuracy and functionality of our accelerator template are silicon-verified. Our approach prioritizes the performance gain through optimized compute and storage capacity balancing, while excluding specific computational optimizations.

D. Runtime Overhead and Acceleration

Figure 9 shows the runtime of CIM-Tuner on seven workloads. The time is estimated as the average time of macro [9] and [5] targeting energy efficiency and throughput under the area budget of 5 mm². By incorporating the operator-size-aware objective function, an average reduction of over 80% in execution time is achieved. Furthermore, compared to the vanilla hardware search method in [19], our approach incor-

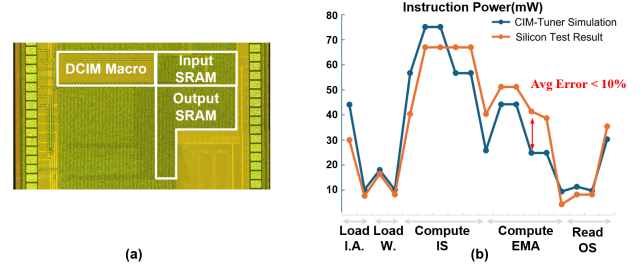


Fig. 10. Power accuracy verification on silicon results. (a) Die photo of instantiated SRAM-CIM accelerator. (b) Tested power of supported instructions.

porating hardware pruning and bandwidth constraints reduces the total hardware design space by over 35%.

E. CIM-Tuner Accuracy

For the latency verification, a verification script is provided to examine the instruction flow of CIM-Tuner compiler. By analyzing the generated memory access address trace, the matrix multiplication flow can be directly verified. For the power and functionality, a 28nm prototype accelerator of (*MR*, *MC*, *SCR*, *IS_SIZE*, *OS_SIZE*) = (1, 1, 16, 16, 16) is silicon verified as the Figure 10 shows. A vanilla DCIM macro is applied with parameters of (*AL*, *PC*, *SCR*, *ICW*, *WUW*) = (64, 8, 8, 512, 128). Compared to simulation results, a relative error of less than 10% is observed, which is considered very accurate as the simulation and silicon difference.

V. CONCLUSION

The various CIM implementations and under-explored mapping strategy impede the SRAM-CIM accelerators hardware balancing exploration. CIM-Tuner tackles these challenges by introducing a generalized accelerator template and fine-grained two-level mapping strategies. Via hardware-mapping co-exploration, optimal performance can be achieved over SOTA CIM accelerators.

ACKNOWLEDGMENT

This work was supported in part by the National Natural Science Foundation of China (62522403, 92264203, 62325405, U24B6015, 62504139); in part by the Key Research and Development Program of Jiangsu Province (BE2023020–1) and Beijing Natural Science Foundation (L257010); and in part by the Fundamental Research Funds for the Central Universities.

REFERENCES

- [1] A. Samajdar, J. M. Joseph, Y. Zhu, P. Whatmough, M. Mattina and T. Krishna, "A Systematic Methodology for Characterizing Scalability of DNN Accelerators using SCALE-Sim," *2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Boston, MA, USA, 2020. pp. 58-68
- [2] K. Sehoon *et al.*, "Full stack optimization of transformer inference: a survey." arXiv preprint arXiv:2302.14017 (2023).
- [3] A. Vaswani *et al.*, "Attention is All you Need," In *NeurIPS*, 2017.
- [4] J. Devlin *et al.*, "Bert: Pre-training of deep bidirectional transformers for language understanding." In *Proceedings of naacL-HLT*, Vol. 1. 2019.
- [5] X. Si *et al.*, "15.5 A 28nm 64Kb 6T SRAM Computing-in-Memory Macro with 8b MAC Operation for AI Edge Chips," In *2020 IEEE International Solid-State Circuits Conference - (ISSCC)*, 2020, pp. 246-248.
- [6] P. -C. Wu *et al.*, "A 22nm 832Kb Hybrid-Domain Floating-Point SRAM In-Memory-Compute Macro with 16.2-70.2TFLOPS/W for High-Accuracy AI-Edge Devices," In *2023 IEEE International Solid-State Circuits Conference (ISSCC)*, 2023, pp. 126-128.
- [7] Z. Jiang, S. Yin, J. -S. Seo and M. Seok, "C3SRAM: In-Memory-Computing SRAM Macro Based on Capacitive-Coupling Computing," In *IEEE Solid-State Circuits Letters*, vol. 2, no. 9, pp. 131-134, Sept. 2019.
- [8] J. -W. Su *et al.*, "A 8-b-Precision 6T SRAM Computing-in-Memory Macro Using Segmented-Bitline Charge-Sharing Scheme for AI Edge Chips," In *IEEE Journal of Solid-State Circuits*, vol. 58, no. 3, pp. 877-892, March 2023.
- [9] A. Guo *et al.*, "A 28nm 64-kb 31.6-TFLOPS/W Digital-Domain Floating-Point-Computing-Unit and Double-Bit 6T-SRAM Computing-in-Memory Macro for Floating-Point CNNs," In *2023 IEEE International Solid-State Circuits Conference (ISSCC) 2023*, pp. 128-130.
- [10] F. Tu *et al.*, "TranCIM: Full-Digital Bitline-Transpose CIM-based Sparse Transformer Accelerator With Pipeline/Parallel Reconfigurable Modes," In *IEEE Journal of Solid-State Circuits*, vol. 58, no. 6, pp. 1798-1809, June 2023.
- [11] F. Tu *et al.*, "16.1 MuITCIM: A 28nm 2.24μJ /Token Attention-Token-Bit Hybrid Sparse Digital CIM-Based Accelerator for Multimodal Transformers," *2023 IEEE International Solid-State Circuits Conference (ISSCC)*, San Francisco, CA, USA, 2023, pp. 248-250
- [12] J. Yue *et al.*, "A 5.6-89.9TOPS/W Heterogeneous Computing-in-Memory SoC with High-Utilization Producer-Consumer Architecture and High-Frequency Read-Free CIM Macro," In *2023 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*, 2023, pp. 1-2.
- [13] S. Liu *et al.*, "16.2 A 28nm 53.8TOPS/W 8b Sparse Transformer Accelerator with In-Memory Butterfly Zero Skipper for Unstructured-Pruned NN and CIM-Based Local-Attention-Reusable Engine," In *2023 IEEE International Solid-State Circuits Conference (ISSCC) 2023*, pp. 250-252.
- [14] M. -E. Shih *et al.*, "20.1 NVE: A 3nm 23.2TOPS/W 12b-Digital-CIM-Based Neural Engine for High-Resolution Visual-Quality Enhancement on Smart Devices," In *2024 IEEE International Solid-State Circuits Conference (ISSCC)*, 2024.
- [15] Y. Jing *et al.*, "AIG-CIM: A Scalable Chiplet Module with Tri-Gear Heterogeneous Compute-in-Memory for Diffusion Acceleration", In *Proceedings of the 61th ACM/IEEE Design Automation Conference (DAC)*, 2024.
- [16] Junwoo Park, Kyeongho Lee, and Jongsun Park. 2025. TP-DCIM: Transposable Digital SRAM CIM Architecture for Energy-Efficient and High Throughput Transformer Acceleration. *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*. Association for Computing Machinery, New York, NY, USA, Article 219, 1-8.
- [17] J. Yue *et al.*, "STICKER-IM: A 65 nm Computing-in-Memory NN Processor Using Block-Wise Sparsity Optimization and Inter/Intra-Macro Data Reuse," in *IEEE Journal of Solid-State Circuits*, vol. 57, no. 8, pp. 2560-2573, Aug. 2022
- [18] J. Chen *et al.*, "AutoDCIM: An Automated Digital CIM Compiler," *2023 60th ACM/IEEE Design Automation Conference (DAC)*, San Francisco, CA, USA, 2023, pp. 1-6
- [19] X. Sun *et al.*, "Efficient Processing of MLPerf Mobile Workloads Using Digital Compute-In-Memory Macros," In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 4, pp. 1191-1205, April 2024.
- [20] Cong Wang, Zeming Chen, and Shanshi Huang. 2025. MICSim: A Modular Simulator for Mixed-signal Compute-in-Memory based AI Accelerator. In *Proceedings of the 30th Asia and South Pacific Design Automation Conference (ASPDAC '25)*.
- [21] L. Chang *et al.*, "IPOCIM: Artificial Intelligent Architecture Design Space Exploration With Scalable Ping-Pong Computing-in-Memory Macro," In *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 32, no. 2, pp. 256-268, Feb. 2024.
- [22] Z. Zhu *et al.*, "MNSIM 2.0: A Behavior-Level Modeling Tool for Processing-In-Memory Architectures," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 11, pp. 4112-4125
- [23] Y. -C. Lo and R. -S. Liu, "Morphable CIM: Improving Operation Intensity and Depthwise Capability for SRAM-CIM Architecture," *2023 60th ACM/IEEE Design Automation Conference (DAC)*, San Francisco, CA, USA, 2023, pp. 1-6
- [24] J. Lee, A. Lu, W. Li and S. Yu, "NeuroSim V1.4: Extending Technology Support for Digital Compute-in-Memory Toward 1nm Node," in *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 71, no. 4, pp. 1733-1744