EasyACIM: An End-to-End Automated Analog CIM with Synthesizable Architecture and Agile Design Space Exploration

Haoyi Zhang[1], Jiahao Song[1], Xiaohan Gao[3],
Xiyuan Tang[1,2], Yibo Lin[1,4,5*], Runsheng Wang[1,4,5], Ru Huang[1,4,5]
[1]School of Integrated Circuits [2]Institute for Artificial Intelligence, Peking University
[3]School of Computer Science [4]Beijing Advanced Innovation Center for Integrated Circuits
[5]Institute of Electronic Design Automation, Peking University, Wuxi, China
hy.zhang@stu.pku.edu.cn, yibolin@pku.edu.cn

ABSTRACT

Analog Computing-in-Memory (ACIM) is an emerging architecture to perform efficient AI edge computing. However, current ACIM designs usually have unscalable topology and still heavily rely on manual efforts. These drawbacks limit the ACIM application scenarios and lead to an undesired time-to-market. This work proposes an end-to-end automated ACIM based on a synthesizable architecture (EasyACIM). With a given array size and customized cell library, EasyACIM can generate layouts for ACIMs with various design specifications end-to-end automatically. Leveraging the multi-objective genetic algorithm (MOGA)-based design space explorer, EasyACIM can obtain high-quality ACIM solutions based on the proposed synthesizable architecture, targeting versatile application scenarios. The ACIM solutions given by EasyACIM have a wide design space and competitive performance compared to the state-of-the-art (SOTA) ACIMs.

1 INTRODUCTION

With the emergence of AI technology, the demand for computility has increased dramatically and the memory-wall effect is becoming more and more evident. Computing-in-Memory (CIM) is a popular solution for AI accelerators addressing the memory bottleneck. Exploiting the structural alignment between a dense 2D array of bit cells and the dataflow in matrix-vector multiplication, CIM has unique advantages in energy and throughput over other solutions [1]. The mainstream CIM can be categorized into two groups ACIM and Digital CIM (DCIM). Although DCIM has better robustness, ACIM still has great potential because of high energy efficiency and high density at lower computing precision [2]. Based on this unique feature, ACIM is able to occupy a niche in AI edge computing.

Traditional ACIM research has focused on the pursuit of extreme performance such as high energy efficiency [3, 4], high area efficiency [5, 6], high accuracy [4, 7] or high throughput [8]. As Figure 1 shows, these designs often have an unscalable topology with fixed array height H, array width W, and ADC bits B_{ADC}. These fixed parameters lead to the gap between the unscalable CIM macro and different application scenarios. For example, a transformer for large language model (LLM) and a convolution neural network (CNN) for image identification are likely to have different accuracy requirements. A particular CIM macro may not be accurate enough for the transformer but has energy waste

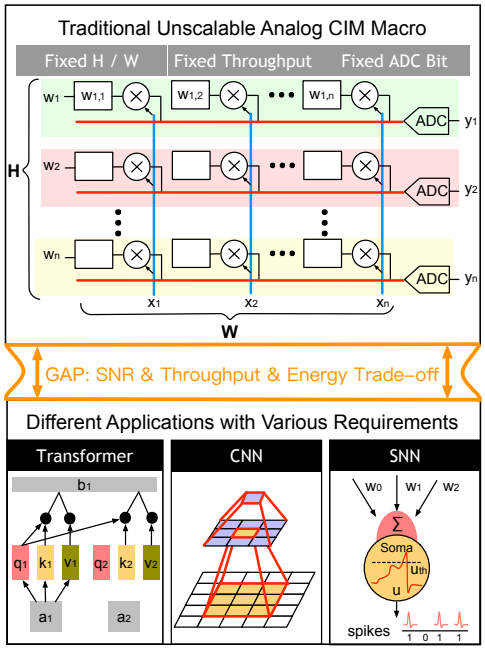

Figure 1: Unscalable ACIM macro and various scenarios.

for CNN due to the redundant accuracy. While some works have flexible ADC bits [5, 9, 10], it is difficult for such reconfigurable designs to eliminate all the overhead caused by redundant precision, including area, energy consumption, and throughput.

Beyond the ACIM circuit design itself, the design efficiency is also very significant. As the electronic design automation (EDA) technology evolves, some studies have emerged to help CIM circuits benchmarking [11, 12] and modeling [13, 14]. Furthermore, inspired by end-to-end automated flow for SRAM [15] and SAR ADC [16], AutoDCIM [17] proposed the first end-to-end automated flow for DCIM. However, the end-to-end automated flow for ACIM is still a blank slate since ACIM has a more sophisticated signal-noise ratio (SNR), energy, area, and throughput trade-off strategies than DCIM. Therefore, a complete end-to-end flow for ACIM should automatically optimize these trade-offs rather than leaving them up to users, as is the case with AutoDCIM [17].

In this work, we propose EasyACIM, an end-to-end automated ACIM with a fully synthesizable ACIM architecture and agile exploration of design specifications. To narrow the gap between the CIM macro and different application scenarios, EasyACIM proposes a novel ACIM architecture that can easily be implemented with different H, W, B_{ADC}, and throughput. EasyACIM constructs an estimation model for the particular architecture and leverages the genetic algorithm to automatically explore the Pareto frontier for the synthesizable architecture with a given array size. Such an approach further improves the design efficiency which is ignored in AutoDCIM [17]. EasyACIM integrates a

*Corresponding author

DAC '24, June 23–27, 2024, San Francisco, CA, USA

ACM ISBN 979-8-4007-0601-1/24/06
https://doi.org/10.1145/3649329.3656229

template-based hierarchical placement and routing framework to generate the final layouts for the ACIM with an agile exploration of design specifications. The main contributions of this paper can be summarized as follows:

- We propose a novel synthesizable ACIM architecture leveraging the local compute array and the reusable capacitors that can be used as CDAC capacitors in SAR ADCs, which is easily implemented into versatile application scenarios.
- We treat the determination of ACIM parameters as a multi-objective optimization problem, build estimation models for the proposed ACIM, and obtain the Pareto frontier by a MOGA-based (NGSA-II) design space explorer.
- We integrate a template-based hierarchical placement and routing framework into the EasyACIM, in order to generate the final layouts according to the Pareto-frontier design specifications.
- As illustrated in the results, EasyACIM can generate ACIMs with SOTA performance for various applications with a wide design space where the energy efficiency ranges from 50TOPS/W to 750TOPS/W and the area ranges from 1500F^2/bit to 7500F^2/bit.

The rest of the paper is organized as follows. Section 2 describes the background; Section 3 explains the detailed implementation; Section 4 demonstrates the results; Section 5 concludes the paper.

2 PRELIMINARIES

This section will review the background for the ACIM compute model, Pareto optimization, and layout automation for Analog and Mixed Signal (AMS) design, respectively.

2.1 ACIM Compute Model

Much research on ACIM has emerged in recent years. Most of them employ following three in-memory compute models (Figure 2): (a) charge summing (QS) [18]; (b) current summing (IS) [19]; (c) charge redistribution (QR) [20]. QS and QR are both charge-domain compute models that are insensitive to the process-voltage-temperature (PVT). However, such an approach stores information in the form of electrical charges and requires additional metal capacitance, resulting in additional area overhead. In more detail, the QR model leverages the redistributing charge between storage units which is more flexible and extensible for different computing applications. The QS model generates the results by summing the charge from the storage units which is more difficult to support different applications. IS is a current-domain compute model that usually has higher density but is sensitive to PVT. The information is stored in the electric current which is also difficult to be adaptive with various applications. For the consideration of robustness and extensibility, EasyACIM selects QR as the compute model for synthesizable architecture.

2.2 Pareto Optimization

The trade-off among SNR, energy, throughput, and area in ACIM is a typical muti-objective optimization problem [21]. It is difficult to find a single optimal solution, especially when faced with different application scenarios. Therefore, obtaining the Pareto-frontier set for the ACIM is a feasible solution. The Pareto-frontier set is made up of the solution vectors that are not dominated by other vectors. Formally, a solution vector $\boldsymbol{u} = [u_1, u_2, \ldots, u_P]^T$ is said to pareto-dominate [22] the solution vector $\boldsymbol{v} = [v_1, v_2, \ldots, v_P]^T$, in a minimization context, if and only if:

$$\begin{aligned} &\forall i \in \{1, \ldots, N\}, f_i(\boldsymbol{u}) \leq f_i(\boldsymbol{v}) \\ &\text{and } \exists j \in \{1, \ldots, N\} : f_j(\boldsymbol{u}) < f_j(\boldsymbol{v}) \end{aligned} \tag{1}$$

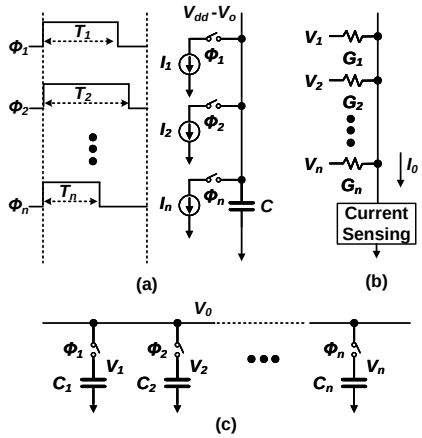

Figure 2: In-memory compute models: (a) QS (b) IS (c) QR.

The function values of the Pareto-frontier set form the Pareto frontier for a particular multi-objective optimization problem.

2.3 Layout Automation for AMS Design

The layout design of ACIM is more similar to analog and mixed-signal (AMS) circuits than digital circuits since the ACIM includes versatile AMS blocks such as SAR ADC, sense amplifier (SA), and CMOS switch. Such AMS blocks prevent the designers from using commercial digital layout automation tools to generate ACIM layouts. Therefore, the layout automation tools for AMS circuits are more suitable when tackling ACIM layouts.

Much research has been done on the placement and routing problems of AMS circuit designs. ALIGN [23] and MAGICAL [24] both construct a complete framework for AMS designs including placement and routing. These frameworks already have the ability to generate decent layouts for AMS designs. After that some independent placement and routing targeting better performance have emerged, such as SAGERoute [25, 26] and hierechical AMS placement [27]. All of these placement and routing methodologies are based on the partitioned grids, as Figure 3 shows. Since the grid-based method is easier to extend with versatile scenarios and honors different constraints in AMS design layouts. In practice, more constraints such as symmetry, alignment, etc. should be considered in addition to the basic half-perimeter wire length (HPWL).

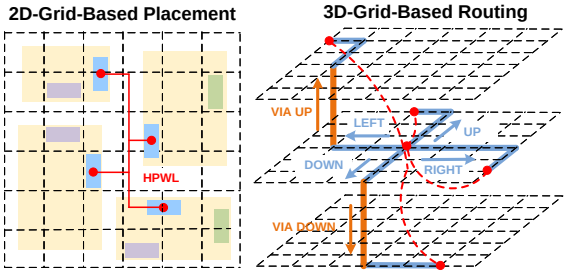

Figure 3: Basic grid-based placement and routing.

Although academia is booming in AMS layout automation, the automatically-generated layouts may still be unsatisfactory in some extreme cases. For example, the SRAM cell in ACIM is very dense, and the routing track is often well-designed by experienced designers. It is difficult for a fully automated tool to meet all the requirements. Therefore, we develop a template-based placement and routing method along with automated approaches to generate high-quality layout solutions for ACIM utilizing manually designed layout cells.

3 EASYACIM FRAMEWORK

An overview of the EasyACIM framework is depicted in Figure 4. The whole framework takes a customized cell library, synthesizable architecture, and technology files as input. The customized cell library includes netlists of all the components of ACIM (e.g. SAR logic, SA, 8T SRAM cell) and layouts of critical components of ACIM (e.g. SA, 8T SRAM cell). The synthesizable architecture determines the rules for combining these components. The technology files contain the necessary information for layout generation (e.g. DRC rules, layer map).

The MOGA-based design space explorer can generate a Pareto-frontier set at a user-defined array size leveraging NSGA-II, a classic MOGA. Each solution vector in the Pareto-frontier set contains four components including array height (H), array width (W), local array size (L), and ADC precision bits (B_{ADC}). After the automatic exploration, the users can remove undesired solutions from the Pareto-frontier set according to their requirements. Via this agile interaction, the Pareto-frontier set can be further refined to match the desired application scenarios. Then the template-based netlist generator as well as template-based hierarchical placer and router will be conducted in sequence for each solution of the Pareto-frontier set. Finally, high-quality ACIM layouts can be generated, ensuring Pareto-frontier design specifications that align with the user's requirements.

3.1 Synthesizable Architecture Design

Figure 6 demonstrates the overview of the proposed synthesizable ACIM architecture as well as the basic operating states. One column of the proposed ACIM is detailed in Figure 6. Inspired by a novel design [4], we reuse the compute capacitors C_{F} as the capacitors in CDAC during the SAR ADC conversion. This is achieved by dividing them into distinct SAR groups following a ratio of $1{:}1{:}2{:}4{:}\cdots{:}2^n$, aligning with the capacitance ratio in the CDAC. This approach greatly reduces the ADC area overhead in the ACIM designs. However, if each 8T SRAM cell is furnished with an individual capacitor and its corresponding control circuit, the area overhead will remain substantial. Therefore, we combine L 8T-SRAM cells into a local array [20]. The L 8T-SRAM cells in a particular local array share the same compute capacitor and control circuits. Selecting an appropriate L introduces a trade-off between area and throughput.

The proposed ACIM architecture has two operating states: 1) multiply-accumulate (MAC) state, and 2) ADC conversion state. In

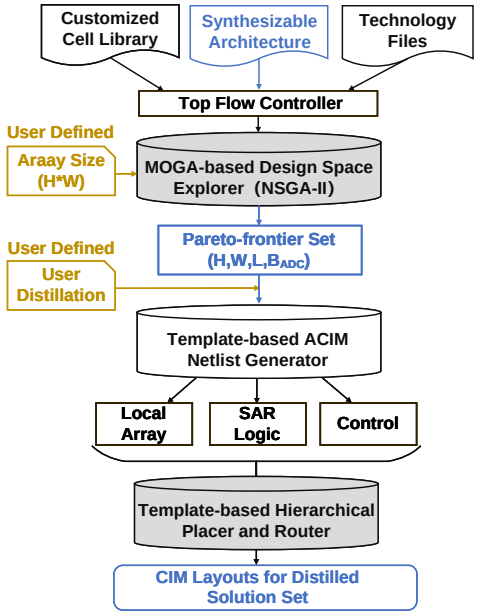

Figure 4: Overview of EasyACIM.

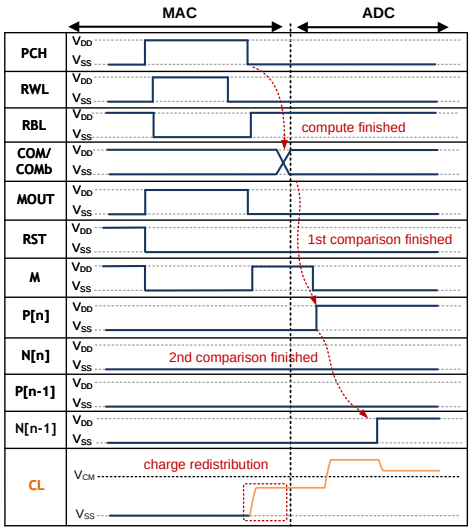

Figure 5: Timing diagram of the synthesizable ACIM.

the MAC state, both ends of the capacitor C_{F} are reset to the V_{CM} at first. Then, as Figure 5 shows, the RWL turns to V_{dd}, the RST turns to V_{ss}, and the MAC operation starts. After the MAC operation, the top plate of the capacitor will be changed to either V_{dd} or V_{ss}, representing the computation result. In the ADC conversion state, the top plate will be reset to V_{CM} again and the charge will redistribute in the bottom plate of the capacitor. After charge redistribution, the final accumulation result V_{x} is stored on the RBL. Then the SAR logic starts the switching procedure to get the final digital result. The P[n] and N[n] is the switching control signal based on the comparison results of each round. After B_{ADC} rounds comparison, the final MAC result with the precision of B_{ADC} bits can be obtained. When dealing with a different B_{ADC}, a CMOS switch will be inserted in the appropriate position of the RBL. The CMOS switch will remain closed until the charge redistribution is complete. Then the CMOS switch will be opened to separate the redundant large capacitance, thus saving energy during the ADC conversion process.

3.2 MOGA-based Design Space Explorer

The MOGA-based design space explorer consists of two parts: 1) ACIM performance estimation model and 2) NSGA-II-based optimization. While exploring ACIM design specifications, constructing the ACIM performance estimation model is more important than the optimization algorithm itself. The primary focus lies in developing an accurate and efficient ACIM estimation model, which is our main concentration. With a robust ACIM estimation model, the requirements of the algorithm can be appropriately relaxed. We select NSGA-II as the exploration algorithm due to its adeptness in maintaining a superior balance, facilitating smooth convergence, and preserving diversity within solutions [21].

3.2.1 ACIM Performance Estimation Model. The ACIM can be evaluated from various perspectives, such as SNR, energy, area, and throughput. EasyACIM adopts a QR mode computation with bottom-plate charge redistribution. Reference to literature [14], a customized estimation model is constructed for EasyACIM and is detailed as follows.

The total SNR ($\mathrm{SNR_T}$) is shown in Equation 2, where $\mathrm{SNR_{pre}}$ indicates the SNR before ADC, and $\mathrm{SQNR_y}$ indicates the SNR for quantization noise of ADC.

$$\mathrm{SNR_T} = \left[\frac{1}{\mathrm{SNR_{pre}}} + \frac{1}{\mathrm{SQNR_y}}\right]^{-1} \tag{2}$$

The $\mathrm{SNR_{pre}}$ can be breakdown into $\mathrm{SNR_a}$ and $\mathrm{SQNR_i}$. The $\mathrm{SNR_a}$ is the noise caused by the analog circuits and $\mathrm{SQNR_i}$ is the output referred

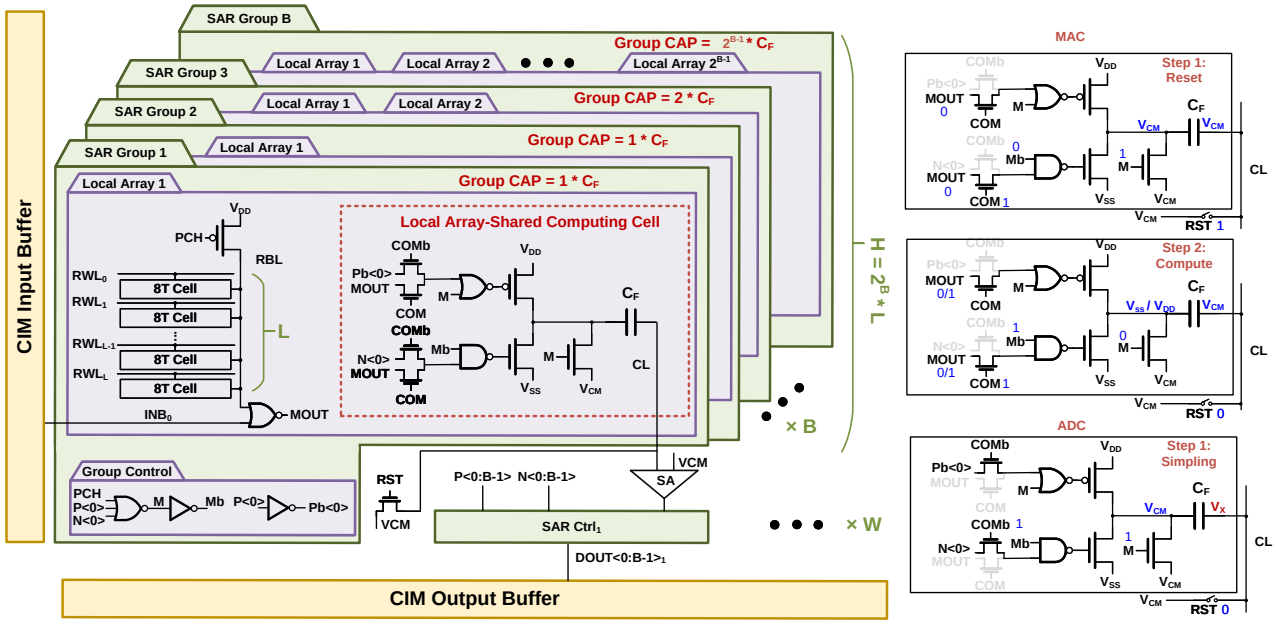

Figure 6: The synthesizable architecture and operating states.

SQNR due to input (weight and activation) quantization.

$$\mathrm{SNR}_{\mathrm{pre}} = \frac{\sigma_{y_o}^2}{\sigma_{q_i}^2 + \sigma_{\eta_a}^2} = \left[\frac{1}{\mathrm{SNR_a}} + \frac{1}{\mathrm{SQNR_i}}\right]^{-1} \tag{3}$$

In equation 3, $\sigma_{y_o}^2$ is the variance of the output and $\sigma_{y_o}^2 = N\sigma_w^2\mathbb{E}\left[x^2\right]$. $\sigma_{q_i}^2$ is the variance of input quantization noise, $\sigma_{\eta_e}^2$ is the variance of analog nonlinearity. Their definitions are detailed in 4 and 5, respectively. Table 1 gives definitions of some basic symbols.

$$\sigma_{q_i}^2 = \frac{1}{12}N\Delta_x^2\sigma_w^2 + \frac{1}{12}N\Delta_w^2\mathbb{E}\left[x^2\right] \tag{4}$$

In Equation 4 $\Delta_w = w_{\mathrm{m}}2^{-B_w+1}$, $\Delta_x = x_{\mathrm{m}}2^{-B_x}$

$$\sigma_{\eta_e}^2 = \frac{2}{3}\left(1 - 4^{-B_w}\right)N\left(\frac{\mathbb{E}\left[x^2\right]\sigma_{C_0}^2}{C_{\mathrm{o}}^2} + \frac{2\sigma_{\theta,\mathrm{o}}^2}{V_{\mathrm{dd}}^2} + \sigma_{\mathrm{inj}}^2\right) \tag{5}$$

In Equation 5, C_{o} is the compute capacitor with standard deviation $\sigma_{C_{\mathrm{o}}} = \kappa\sqrt{C_{\mathrm{o}}}$. κ is a layout and technology dependent mismatch coefficient [28]. $\sigma_{\theta,\mathrm{o}} = \sqrt{\frac{kT}{C_{\mathrm{o}}}}$ is the themeral noise caused by C_{o}, k is the Boltzmann constant, T is the temperature in Kelvin. σ_{inj}^2 indicates the noise caused by charge injection which is almost eliminated by the bottom-plate charge redistribution technique and can be ignored in the following calculations. $\mathrm{SQNR_y}$ is detailed as follows, where $\zeta_x = x_m/\sigma_x$, $\zeta_w = w_m/\sigma_w$.

$$\begin{aligned}\mathrm{SQNR}_{\mathrm{y(dB)}} &= 10\log_{10}\left(\frac{\sigma_{y_o}^2}{\sigma_{q_y}^2}\right) \\ &= 6B_y + 4.8 - \left[\zeta_{x(\mathrm{dB})} + \zeta_{w(\mathrm{dB})}\right] - 10\log_{10}(N)\end{aligned} \tag{6}$$

The throughput can be described as Equation 7. t_{com} is the computation delay which is much less than ADC's delay. The delay of ADC can be broken down into setup time t_{set} and 1-bit conversion time t_{conv}. t_{set} should satisfy $t_{\mathrm{set}} > 0.69\tau B_{\mathrm{ADC}}$, where τ is the time constant. t_{conv} can be estimated by $t_{\mathrm{conv}} = t_{\mathrm{conv/bit}} \cdot B_{\mathrm{ADC}}$

$$T = \frac{H}{L} \cdot W/(t_{\mathrm{com}} + t_{\mathrm{set}} + t_{\mathrm{conv}}) \tag{7}$$

The total average energy for one-bit computing can be defined as Equation 8. The E_{compute} and E_{control} are almost constant for different design specifications in a given architecture. The power consumption of ADC with different precision really makes the difference.

$$E = E_{\mathrm{compute}} + E_{\mathrm{control}} + \frac{E_{\mathrm{ADC}}}{H/L} \tag{8}$$

The E_{ADC} has an empirical formula [29] described as Equation 9, where k_1 and k_2 are empirical parameters which can be obtained from post-layout simulation.

$$E_{\mathrm{ADC}} = k_1 \cdot \left(B_{\mathrm{ADC}} + \log_2 V_{\mathrm{DD}}\right) + k_2 \cdot 4^{B_{\mathrm{ADC}}} \cdot V_{\mathrm{DD}}^2 \tag{9}$$

The average area of ACIM is demonstrated in Equation 10, where A_{SRAM} is the 8T-SARM cell area, A_{LC} is the area of local array-shared computing cell, A_{COMP} is the area of the dynamic comparator and A_{DFF} is the area of a single dynamic D-type Flip Flop (DFF) in the SAR logic.

$$A = A_{\mathrm{SRAM}} + \frac{1}{L} \cdot A_{\mathrm{LC}} + \frac{1}{H} \cdot A_{\mathrm{COMP}} + \frac{1}{H} \cdot B_{\mathrm{ADC}} \cdot A_{\mathrm{DFF}} \tag{10}$$

3.2.2 NSGA-II-based optimization. Based on the proposed ACIM performance estimation model, we obtain four objective functions f_{SNR}, f_{T}, f_{E}, f_{A}. Equation 7, 8, 10 clearly demonstrates f_{T}, f_{E}, f_{A}, respectively. The f_{SNR} can be obtained by simplifying Equations 2-6. The simplified f_{SNR} is depicted in Equation 11, where k_3 and k_4 are constant coefficients related to the data distribution, C_{o} is the compute capacitor.

$$\mathrm{SNR}_{\mathrm{(dB)}} = 6B_{\mathrm{ADC}} - 10\log 10\frac{H}{L} - 10\log 10\frac{k_3}{C_{\mathrm{o}}} + k_4 \tag{11}$$

Based on the previous analysis, the multi-objective optimization problem of ACIM performance can be formulated as Equation 12. The negative sign in front of f_{SNR} and f_{T} means that it is required to solve for the maximum value. The constraint $H - L \geq 0$ guarantees that local array size L can not be larger than the array height H and $\frac{H}{L} - 2^{B_{\mathrm{ADC}}}$ indicate that the ADC precision is limited by the available capacitors

Table 1: Notation

Symbol	Description
N	dot product length
B	precision in bits
x, w, and y	inputs, weights, and outputs
x_{m}, w_{m}, and y_{m}	the corresponding maximum
σ_x, σ_w	standard deviation of input and weight

The constraint $H \cdot W = Arraysize$ guarantees the final array size is exactly equal to the user-defined array size. Finally, a classic NSGA-II algotithm [21] is performed and a high-quality Pareto-frontier set can be obtained efficiently.

$$\begin{aligned} \min_x \quad & F(H, W, L, B_{\text{ADC}}) = [-f_{\text{SNR}}, -f_{\text{T}}, f_{\text{E}}, f_{\text{A}}] \\ & \text{s.t. } \frac{H}{L} - 2^{B_{\text{ADC}}} \geq 0 \\ & H - L \geq 0 \\ & H \cdot W = Arraysize \end{aligned} \tag{12}$$

3.3 Template-based Hierechical Placer and Router

After obtaining the user-distilled Pareto-frontier set, the template-based ACIM netlist generator generates netlists for each solution in the user-distilled Pareto-frontier set. Due to the page limit, we omit the details on the netlist generator, which follows a straightforward engineering process. Then, EasyACIM performs template-based hierarchical placement and routing for the netlists to generate the final layouts.

The fundamental of the proposed placer and router is the classic grid-based algorithm [25–27]. However, the fully automated layouts often fail to meet strict design requirements. Therefore, EasyACIM extends the classic algorithm to support manually designed cells in the layout automation framework shooting for better performance. EasyACIM leverages the hierarchical framework to facilitate this extension. Figure 7 depicts the strategy of template-based hierarchical placer and router. The "Std" layout cell indicates either a real standard cell or PCell in PDK, or a manually designed cell. In each hierarchy, the placement and routing inside the "Std" layout cell or subcircuit will be kept, only the inter-connection routing and over-cell placement are conducted. For example, in Hierarchy 1 only the interconnection among C1, C2, S1, and S2 will be routed and these "Std" layout cells or subcircuits will be placed as a whole. Finally, following a bottom-up strategy, the final layout can be generated with manual-designed cells.

4 EXPERIMENTAL RESULTS

We perform experiments on a Linux server with an Intel Xeon Gold 6230 CPU @ 2.10GHz. EasyACIM is implemented on the TSMC28 PDK with 1bx1b computation. The agile design exploration for a particular array size can be finished in 30 minutes. The layout generation for a particular solution in the Pareto-frontier set can be done in a few minutes thanks to the customized cell library and pre-defined routing tracks for critical nets including power nets and SAR logic control nets.

The main differences between EasyACIM and other design flows are shown in Table 2. Compared to the traditional flow, EasyACIM can dramatically accelerate the design cycle and generate design layouts automatically. In contrast to the AutoDCIM [17], EasyACIM automatically determines the design parameters (e.g. H, W, L, B_{ADC}) and performs

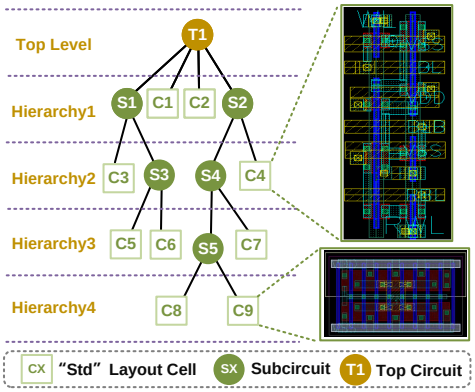

Figure 7: Strategy of template-based hierarchical placer and router.

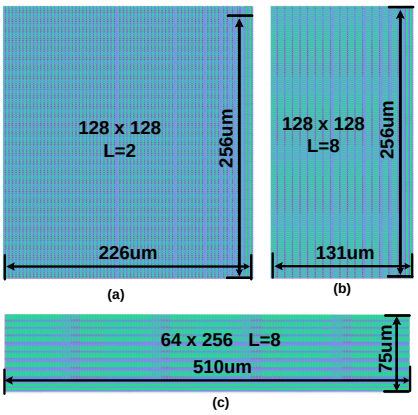

Figure 8: The layouts of 16kb ACIM with various design specifications.

Table 2: Comparison with Other CIM Design Flow.

Entry	Traditional Flow	AutoDCIM [17]	EasyACIM
Design type	Analog or Digital	Digital	Analog
Design of layout	Manual	Automatic	Automatic
Design time	1-2 months	NA	Several hours
Design space	Fixed	Unoptimized	Pareto frontier
Determination of design parameters	Manual	User-defined	Automatic

agile design space exploration to uncover the Pareto frontier, while AutoDCIM [17] only takes the user-defined design parameters and conducts design space exploration without any optimization.

Figure 8 demonstrates the final layout results of a 16kb ACIM with 3-bit ADC precision. Figure 8(a) shows the situation where H=128, L=2 shooting for high throughput (3.277TOPS) but at the expense of area (4504F^2/bit). Figure 8(b) depicts a design with a more balanced performance (throughput=0.813TOPS,area=2610F^2/bit). Compared to Figure 8(b), Figure 8(c) achieved higher SNR and the same throughput at the expense of area (area=2977F^2/bit).

A holistic analysis of EasyACIM design space is shown in Figure 9. During the design space exploration, B_{ADC} is set within 8 bits and L is limited to between 2 and 32 to avoid extreme results. Figure 9(a)(b) shows the overall design space of EasyACIM. Figure 9(c)(d), Figure 9(e)(f), and Figure 9(g)(h) illustrate the impact of different parameters H, L, B_{ADC} on the design space with a given array size. In Figure 9(a)(b) it can be seen that larger arrays present the potential to achieve higher SNR and throughput, while smaller arrays prioritize energy efficiency and area. Figure 9(c)(d) illustrates that a smaller H can lead to a higher throughput. However, this comes with limitations in SNR and an increase in area overhead. Figure 9(e)(f) depicts that reducing L leads to higher throughput and an increased upper bound of SNR, but incurs additional area overhead. As shown in Figure 9(g)(h), reducing B_{ADC} enhances energy efficiency, yet it notably diminishes the SNR as well.

The most common evaluation metrics of ACIM are energy efficiency and area. In Figure 10, we compare the design space with the SOTA ACIM designs. Design A [4], design B [5], design C [8] are SOTA ACIMs from JSSC/ISSCC in recent years. The Pareto frontier based on energy efficiency and area is highlighted with blue dashed lines in Figure 10. It can be seen that EasyACIM can generate high-quality ACIM solutions with competitive performance to SOTA ACIMs. The ACIM solutions generated by EasyACIM also have wide design space with energy efficiency ranging from 50TOPS/W to 750TOPS/W and area ranging from 1500F^2/bit to 7500F^2/bit.

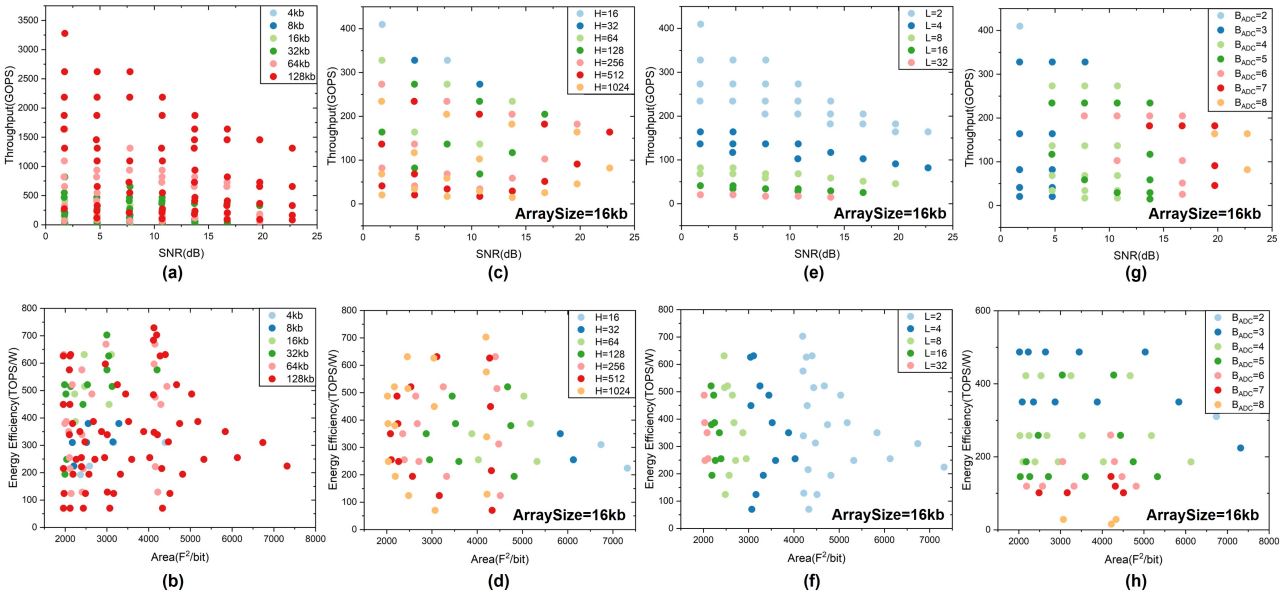

Figure 9: Design Space of EasyACIM: (a) (b) Design space categorized by array size; (c) (d) Design space categorized by H with 16kb array size; (e) (f) Design space categorized by L with 16kb array size; (g) (h) Design space categorized by B_{ADC} with 16kb array size.

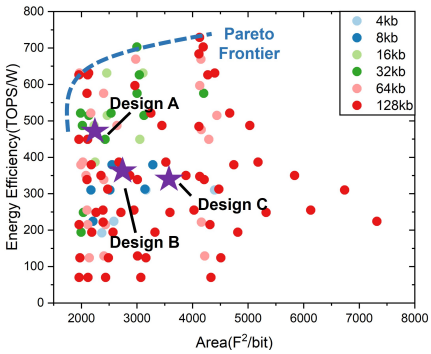

Figure 10: Comparasion between EasyACIM and SOTA ACIMs.

5 CONCLUSION

In this paper, we propose EasyACIM, the first end-to-end automated ACIM. Based on a novel synthesizable architecture, EasyACIM can be easily implemented in various applications with different requirements. Leveraging the MOGA-based Pareto-frontier explorer and template-based hierarchical layout placer and router, EasyACIM can generate high-quality ACIM solutions with competitive performance to SOTA ACIMs and wide design space where the energy efficiency ranges from 50TOPS/W to 750TOPS/W and area ranges from 1500F^2/bit to 7500F^2/bit. Validated in TSMC28, the experimental results demonstrate the robustness and benefits of EasyACIM.

6 ACKNOWLEDGEMENT

This work was supported in part by the National Science Foundation of China (Grant No. 62141404, 62125401), the Natural Science Foundation of Beijing, China (Grant No. Z230002), and the 111 project (B18001).

REFERENCES

[1] N. Verma *et al.*, "In-Memory Computing: Advances and Prospects," *IEEE Solid-State Circuits Magazine*, vol. 11, pp. 43–55, 2019.
[2] C.-J. Jhang *et al.*, "Challenges and Trends of SRAM-Based Computing-In-Memory for AI Edge Devices," *TCAS-I*, vol. 68, pp. 1773–1786, 2021.
[3] S. Cheon *et al.*, "A 2941-TOPS/W Charge-Domain 10T SRAM Compute-in-Memory for Ternary Neural Network," *TCAS-I*, vol. 70, pp. 2085–2097, 2023.
[4] C.-Y. Yao *et al.*, "A Fully Bit-Flexible Computation in Memory Macro Using Multi-Functional Computing Bit Cell and Embedded Input Sparsity Sensing," *JSSC*, vol. 58, pp. 1487–1495, 2023.
[5] C. Yu *et al.*, "A 65-nm 8T SRAM Compute-in-Memory Macro With Column ADCs for Processing Neural Networks," *JSSC*, vol. 57, pp. 3466–3476, 2022.
[6] A. Biswas *et al.*, "An area-efficient 6t-sram based compute-in-memory architecture with reconfigurable sar adcs for energy-efficient deep neural networks in edge ml applications," in *CICC*, 2022, pp. 1–2.
[7] B. Yan *et al.*, "A 1.041-Mb/mm ^{2} 27.38-TOPS/W Signed-INT8 Dynamic-Logic-Based ADC-less SRAM Compute-in-Memory Macro in 28nm with Reconfigurable Bitwise Operation for AI and Embedded Applications," in *ISSCC*. IEEE, 2022, pp. 188–190.
[8] Q. Dong *et al.*, "15.3 A 351TOPS/W and 372.4GOPS Compute-in-Memory SRAM Macro in 7nm FinFET CMOS for Machine-Learning Applications," in *ISSCC*. IEEE, 2020.
[9] C. Yu *et al.*, "A Logic-Compatible eDRAM Compute-In-Memory With Embedded ADCs for Processing Neural Networks," *TCAS-I*, vol. 68, pp. 667–679, 2021.
[10] M. Ali *et al.*, "A 65 nm 1.4-6.7 TOPS/W Adaptive-SNR Sparsity-Aware CIM Core with Load Balancing Support for DL workloads," in *CICC*. IEEE, 2023, pp. 1–2.
[11] P.-Y. Chen *et al.*, "NeuroSim: A Circuit-Level Macro Model for Benchmarking Neuro-Inspired Architectures in Online Learning," *TCAD*, vol. 37, pp. 3067–3080, 2018.
[12] J. Sun *et al.*, "Analog or Digital In-memory Computing? Benchmarking through Quantitative Modeling," *ICCAD*, 2023.
[13] S. K. Gonugondla *et al.*, "Fundamental limits on the precision of in-memory architectures," in *ICCAD*. ACM, 2020, pp. 1–9.
[14] S. K. Gonugondla *et al.*, "Fundamental Limits on Energy-Delay-Accuracy of In-Memory Architectures in Inference Applications," *TCAD*, vol. 41, pp. 3188–3201, 2022.
[15] S. Kamineni *et al.*, "MemGen: An Open-Source Framework for Autonomous Generation of Memory Macros," in *CICC*. IEEE, 2021, pp. 1–2.
[16] M. Liu *et al.*, "OpenSAR: An Open Source Automated End-to-end SAR ADC Compiler," in *ICCAD*. IEEE, 2021, pp. 1–9.
[17] J. Chen *et al.*, "Autodcim: An automated digital cim compiler," in *DAC*, 2023, pp. 1–6.
[18] M. Kang *et al.*, "A Multi-Functional In-Memory Inference Processor Using a Standard 6T SRAM Array," *JSSC*, vol. 53, pp. 642–655, 2018.
[19] Z. Jiang *et al.*, "Xnor-sram: In-memory computing sram macro for binary/ternary deep neural networks," in *2018 IEEE Symposium on VLSIT*, 2018, pp. 173–174.
[20] A. Biswas and A. P. Chandrakasan, "Conv-ram: An energy-efficient sram with embedded convolution computation for low-power cnn-based machine learning applications," in *ISSCC*, 2018, pp. 488–490.
[21] J. L. J. Pereira *et al.*, "A Review of Multi-objective Optimization: Methods and Algorithms in Mechanical Engineering Problems," *Archives of Computational Methods in Engineering*, vol. 29, pp. 2285–2308, 2022.
[22] P. Ngatchou *et al.*, "Pareto Multi Objective Optimization," in *Intelligent Systems Application to Power Systems*. IEEE, 2005, pp. 84–91.
[23] K. Kunal *et al.*, "Invited: Align – open-source analog layout automation from the ground up," *DAC*, pp. 1–4, 2019.
[24] H. Chen *et al.*, "MAGICAL: An Open- Source Fully Automated Analog IC Layout System from Netlist to GDSII," *IEEE Design & Test*, vol. 38, pp. 19–26, 2021.
[25] H. Zhang *et al.*, "Sageroute: Synergistic analog routing considering geometric and electrical constraints with manual design compatibility," in *DATE*, 2023, pp. 1–6.
[26] H. Zhang *et al.*, "Sageroute2.0: Hierarchical analog and mixed signal routing considering versatile routing scenarios," in *DATE*, 2024, pp. 1–6.
[27] K. Zhu *et al.*, "Hierarchical Analog and Mixed-Signal Circuit Placement Considering System Signal Flow," *TCAD*, vol. 42, pp. 2689–2702, 2023.
[28] V. Tripathi and B. Murmann, "Mismatch Characterization of Small Metal Fringe Capacitors," *TCAS-I*, vol. 61, pp. 2236–2242, 2014.
[29] B. Murmann, "Mixed-signal computing for deep neural network inference," *TVLSI*, vol. 29, pp. 3–13, 2021.