Hardware-Aware Compilation and Simulation for In-Memory Computing

Asif Ali Khan[†], Hadjer Benmeziane[§], Hamid Farzaneh[†], João P. C. Lima[†*],William Simon[§], Yiyu Shi[‡], Zheyu Yan[¶], Abu Sebastian[§],X. Sharon Hu[‡], Jeronimo Castrillon[†*], Corey Lammie[§]
[†]TU Dresden, Dresden, Germany
[§]IBM Research Europe, Rüschlikon, Switzerland
[‡]University of Notre Dame, Notre Dame, USA
[¶]Zhejiang University, Hangzhou, China
[*]Center for Scalable Data Analytics and Artificial Intelligence (ScaDS.AI), Dresden, Germany
{asif_ali.khan, jeronimo.castrillon}@tu-dresden.de, ase@zurich.ibm.com, shu@nd.edu, corey.lammie@ibm.com

Abstract

This brief presents an overview of recent tools and research efforts aimed at enhancing the programmability and reliability of In-Memory Computing (IMC)-based systems. We discuss hardware-aware training techniques that improve model resilience to analog device imperfections, and explore mapping strategies that balance accuracy and performance for heterogeneous IMC-based accelerators. Additionally, we examine a compiler framework that abstracts hardware complexities and enables seamless integration of these accelerators into existing deployment pipelines. By combining these approaches with advanced simulation tools, we propose an end-to-end workflow that facilitates the practical deployment and optimization of IMC technologies across diverse memory types and architectural designs.

Keywords

In-Memory Computing, Hardware-Aware Training, Heterogeneous Mapping, Compiler and Simulation Frameworks

1 Introduction

In-Memory Computing (IMC) systems have been demonstrated to outperform traditional Von Neumann architectures by orders of magnitude in both performance and energy efficiency, attracting significant research interest and relevance as of late [1]. IMC eliminates the need for frequent data movement between memory and compute units by performing computation directly within the memory. IMC can be realized in both *analog* (AIMC) and *digital* (DIMC) domains, with a variety of underlying mechanisms including analog crossbars, content-addressable memories (CAMs), and bulk-bitwise logic—each offering distinct trade-offs in terms of scalability, precision, and robustness.

However, despite significant technological progress, the widespread adoption and efficient utilization of these technologies remain a challenge. This is primarily due to two reasons: (i) IMC is often performed in the analog domain using non-volatile memory devices, and thus, is subject to circuit non-idealities and device noise, e.g., temporal variations, some of which are inherently stochastic [2, 3]. (ii) IMC systems typically offer only low-level programming models, making them accessible mainly to hardware experts [4, 5]. Furthermore, as IMC systems continue to evolve, selecting the appropriate memory technology, choosing a suitable IMC system architecture, and efficiently mapping applications onto these systems all have a significant impact on overall system performance, energy efficiency, and computational accuracy [6, 7].

2 Hardware-Aware Training

Similar to reduced-precision Deep Learning (DL) inference accelerators, IMC-based accelerators require weights and activations of floating-point networks to be adapted before their deployment. For some digital IMC accelerators, Quantization-Aware Training (QAT) and Post-Training Quantization (PTQ) can be employed.

For inherently non-deterministic AIMC accelerators, more sophisticated hardware-aware training techniques are required, which can be broadly categorized as inference- or chip-in-the-loop-based. Inference-based techniques [3, 8–10] use a representative software model of the system and usually inject Gaussian noise during training to improve the robustness of the network to analog noise. These "noisy" weights are then programmed to equivalent conductance states. Chip-in-the-loop-based techniques [11] update the conductance states of the devices on the chip using voltage pulses and use their response to directly govern training behavior iteratively. Finally, post-placement calibration techniques [12] can be used to refine scales and parameters after device programming.

To simulate DL workloads on IMC-based accelerators, a number of different frameworks [13] have been developed. A subset of these are specialized for hardware-aware training [14].

3 Accuracy- and Performance-Aware Mapping

To sustain high energy efficiency and throughput, while being sufficiently flexible to support end-to-end DL workloads, IMC-based accelerators must complement weight-stationary IMC-based processing elements with Digital Processing Units (DPUs), heterogeneously. These can vary in complexity, reconfigurability, and localization [15, 16]. DPUs play a crucial role in handling operations that IMC tiles cannot execute, such as activation functions and attention computations for transformer-based models. Additionally, DPUs

CASES '25, Taipei, Taiwan

ACM ISBN 979-8-4007-1991-2/2025/09
https://doi.org/10.1145/3742872.3758333

can extend the effective model size by managing portions of the model that exceed IMC's weight capacity, or be used to accelerate small MVM kernels when execution on IMC would be slower, or for AIMC specifically, supporting MVMs where analog noise sensitivity is a concern. As mapping of different network components to IMC tiles and DPUs affects both accuracy and performance, careful consideration must be taken. Several simulation frameworks have been proposed to effectively negotiate this trade-off [6, 17, 18] using different proxy models for accuracy and performance.

4 Compilation and Simulation

IMC-based systems typically expose low-level device-specific APIs to enable fine-grained control. This limits their accessibility and widespread adoption, restricting their use largely to device experts. To overcome this challenge, several technologies and architecture-specific compiler frameworks have been developed.

For AIMC systems, the Open CIM Compiler (OCC), based on the MLIR framework, abstracts hardware-level details from the programmer and generates optimized code that accounts for key device constraints, such as expensive write operations and fixed array dimensions [19]. In the context of CAM-based accelerators, frameworks such as C4CAM analyze high-level applications to identify CAM-suitable search patterns. When needed, C4CAM rewrites operations to enable offloading onto CAM hardware [20]. Similarly, for logic-based IMC accelerators, compiler frameworks have been developed that generate efficient code by incorporating device properties into optimization strategies [7].

Building on these technology-specific approaches, recent work has focused on generalizing compiler support across different IMC technologies. One such unified framework is Cinnamon, which is built on MLIR and introduces both high-level, device-agnostic and low-level, device-specific abstractions [4]. Cinnamon is designed to be extensible, allowing integration of (i) device-specific cost and performance models, such as [21], for guiding mapping and optimization decisions, and (ii) custom analysis, scheduling, and mapping strategies, including those discussed in Section 3. Cinnamon supports multiple backend targets. For AIMC systems in particular, it can be integrated with domain-specific simulators such as NeuroSim [22] for neural network workloads, or with general-purpose full-system simulators like ALPINE [23].

5 Proposed end-to-end workflow

To deploy a pre-trained floating-point PyTorch network to a heterogeneous AIMC accelerator, we describe one potential workflow.[1] First, hardware-aware training and heterogeneous mapping is performed using the IBM AIHWKIT [14] and LionHeart [6] frameworks. As a proxy for performance, to avoid computationally expensive system-level simulations, the analog MAC ratio (proportion of MAC operations performed using IMC tiles) can be used. Then, the compilation step is performed using Cinnamon, after lowering the heterogeneous network representation to MLIR's `linalg` abstraction. During compilation, a number of different transforms and lowering steps are applied, which are reconfigurable, e.g., the fusing of batch normalization parameters and tiling of weights. Finally, code generation is performed for a number of different backends, e.g., ALPINE [23], and the final performance of the system can be evaluated. Future extensions include support for dynamic workloads and tighter hardware-software co-design.

[1]This has been developed and will be presented for a hands-on tutorial on this topic, co-located with the ESWEEK consortium, 2025.

Acknowledgments

This work is partially funded by ScaDS.AI (01IS18026A-D), DFG (grants 502388442 and 528378584) and the Semiconductor Research Corporation (SRC) Logic and Memory Devices Program (LMD).

References

[1] A. A. Khan *et al.*, "The landscape of compute-near-memory and compute-in-memory: A research and commercial overview," *arXiv:2401.14428*, 2024.
[2] J. P. C. de Lima *et al.*, "All-in-memory stochastic computing using reram," *arXiv preprint arXiv:2504.08340*, 2025.
[3] M. J. Rasch *et al.*, "Hardware-aware training for large-scale and diverse deep learning inference workloads using in-memory computing-based accelerators," *Nature communications*, vol. 14, no. 1, p. 5282, 2023.
[4] A. A. Khan *et al.*, "Cinm (cinnamon): A compilation infrastructure for heterogeneous compute in-memory and compute near-memory paradigms," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'25)*, Mar. 2025.
[5] C. Lammie *et al.*, "Deep learning software stacks for analogue in-memory computing-based accelerators," *Nature Reviews Electrical Engineering*, 2025.
[6] ——, "Lionheart: A layer-based mapping framework for heterogeneous systems with analog in-memory computing tiles," *IEEE Transactions on Emerging Topics in Computing*, 2025.
[7] H. Farzaneh *et al.*, "SHERLOCK: Scheduling efficient and reliable bulk bitwise operations in NVMs," in *Proceedings of the 61th Annual Design Automation Conference (DAC'24)*, ser. DAC '24. ACM, Jun. 2024.
[8] A. Kazemi *et al.*, "Achieving software-equivalent accuracy for hyperdimensional computing with ferroelectric-based in-memory computing," *Scientific Reports*, vol. 12, no. 1, 2022.
[9] Z. Yan *et al.*, "Improving realistic worst-case performance of nvcim dnn accelerators through training with right-censored gaussian noise," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, 2023.
[10] J. P. C. de Lima and L. Carro, "Quantization-aware in-situ training for reliable and accurate edge ai," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2022, pp. 1497–1502.
[11] M. J. Rasch *et al.*, "Fast and robust analog in-memory deep neural network training," *Nature Communications*, vol. 15, no. 1, 2024.
[12] C. Lammie *et al.*, "Improving the accuracy of analog-based in-memory computing accelerators post-training," in *IEEE Int. Symp. on Circuits and Systems*, 2024.
[13] ——, "Modeling and simulating in-memory memristive deep learning systems: An overview of current efforts," *Array*, vol. 13, p. 100116, 2022.
[14] M. Le Gallo *et al.*, "Using the ibm analog in-memory hardware acceleration kit for neural network training and inference," *APL Machine Learning*, vol. 1, no. 4, 11 2023. [Online]. Available: https://doi.org/10.1063/5.0168089
[15] E. Ferro *et al.*, "A precision-optimized fixed-point near-memory digital processing unit for analog in-memory computing," in *2024 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2024.
[16] I. Boybat *et al.*, "Heterogeneous embedded neural processing units utilizing pcm-based analog in-memory computing," in *IEDM*, 2024.
[17] P. Behnam *et al.*, "Harmonica: Hybrid accelerator to overcome imperfections of mixed-signal dnn accelerators," in *2024 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2024, pp. 619–630.
[18] M. Risso *et al.*, "Precision-aware latency and energy balancing on multi-accelerator platforms for dnn inference," in *2023 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, 2023.
[19] A. Siemieniuk *et al.*, "Occ: An automated end-to-end machine learning optimizing compiler for computing-in-memory," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2021.
[20] H. Farzaneh *et al.*, "C4cam: A compiler for cam-based in-memory accelerators," in *Proc. of the Int. Conf. on Architectural Support for Programming Languages and Operating Systems, Volume 3*, ser. ASPLOS 2024, 2024.
[21] A. F. da Silva *et al.*, "LearnCNM2Predict: Transfer learning-based performance model for cnm systems," in *International Conference on Embedded Computer Systems: Architectures Modeling and Simulation (SAMOS)*. IEEE, Jul. 2025.
[22] X. Peng *et al.*, "Dnn+ neurosim: An end-to-end benchmarking framework for compute-in-memory accelerators with versatile device technologies," in *2019 IEEE international electron devices meeting (IEDM)*. IEEE, 2019, pp. 32–5.
[23] J. Klein *et al.*, "Alpine: Analog in-memory acceleration with tight processor integration for deep learning," *IEEE Transactions on Computers*, vol. 72, no. 7, pp. 1985–1998, 2022.