Approximate Computing in Memristive Processing-In-Array (PIA)

Fabian Seiler[+*] and Nima TaheriNejad[+*]
[*]Technische Universität Wien (TU Wien), Austria, [+]Heidelberg University, Germany
fabian.seiler@ziti.uni-heidelberg.de, nima.taherinejad@ziti.uni-heidelberg.de

***Abstract*—Memristor-based in-memory computing has been the topic of much research and development in recent years. Among them, Processing in Array (PIA) boasts of high energy and delay efficiency, since in this type of memory-centric computing, the data never leaves the array for being processed. PIA literature is divided into two major areas; i. analog computing (mainly for matrix multiplication) and ii. digital logic for general-purpose computing. The latter, which is conducted using commonly known stateful logics, despite its versatility and high parallelization ability, requires many steps for each (sub)function. In this work, we present approximate PIA for memristive stateful logic, and how it can significantly reduce the number of steps required for each function and consequently decrease the energy consumption, delay, and potentially the number of required memristors for computation. Approximate computing introduces errors into computations, which, as we show, can be tolerated in error-resilient applications such as image processing and machine learning.**

I. Introduction

In modern computer architectures, data movement between computational areas and memory elements is a major bottleneck for energy consumption and latency [1]. This has motivated a shift from traditional processor-centric architectures (i.e., von Neumann architectures) toward memory-centric computing. Memory-centric approaches appear in different forms, commonly referred to as In-Memory Computation (IMC)/Processing in Memory (PIM) and Near-Memory Computation (NMC)/Processing Near Memory (PNM). However, the distinction between "near" and "in" memory is often interpreted differently across the literature, which can lead to ambiguous classifications. To provide clearer terminology for the discussion in this paper, we use the memory-centric taxonomy shown in Figure 1, which categorizes computing approaches based on where computation occurs relative to the memory array. This taxonomy is independent of the specific memory technology and memory hierarchy level (e.g., storage or cache). The main considerations are the distance data must travel to be processed and the bandwidth limitations along that path. In this framework, Processing in Array (PIA) refers to computation performed directly within the memory array, implying minimal data movement and high bandwidth availability. Moving from PIA through Processing in array Border (PIB), Processing in Chip/package (PIC), Processing in (memory) boarD (PID), and finally Processing in (memory) Exterior (PIE) (traditional architectures), the distance between data and computation generally increases. For simple atomic operations (e.g., logical operations such as `AND` or `OR`), approaches closer to the memory array can provide higher efficiency. However, for more complex computations involving distributed data, different architectural levels may offer advantages, and multiple approaches can coexist in practical systems.

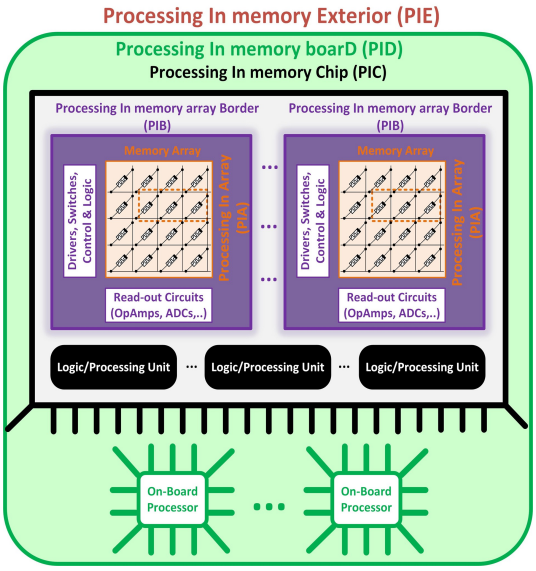

Fig. 1: Memory-centric taxonomy of computer architecture.

Since memristors are energy-efficient, compact, and capable of storing data non-volatilely, they are strong candidates for memory-centric computing [2]. Among memristive memory-centric methods, stateful logic stands out as efficient PIA solutions [2], [3], which is the focus of this work. While they are versatile and offer strong parallelism, the high latency of their (sub)functions makes them prime targets for optimization, since improvements can propagate through higher levels.

Approximate Computing (AxC) is another emerging computer paradigm that may provide possible solutions to computing problems, in that it compromises accuracy for gains in energy efficiency, speed, or area. As we show here, using AxC, the number of steps necessary in stateful processes can be drastically reduced, consequently improving energy efficiency and, in some cases, area usage [4]–[6]. The trade-off for these gains is the reduction of accuracy. Therefore, AxC can only be applied to inherently error-resilient applications such as image/video processing, machine learning, and related fields. Approximation techniques for stateful logic deviate significantly from those applied in CMOS-based designs. In PIA, logical

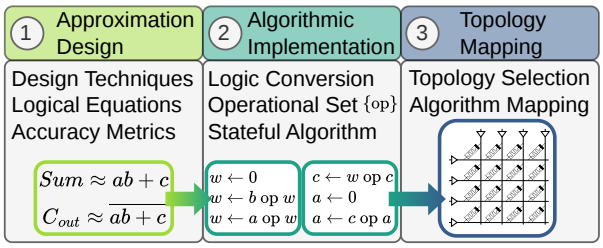

Fig. 2: Design Method for Approximate memristive PIA.

operations are first expressed algorithmically and subsequently implemented as memristive circuits (topologies) within the crossbar array. For example, in Material Implication (IMPLY)-logic [2], these implementations can take several competing forms: serial, parallel, semi-serial, and semi-parallel, each offering distinct trade-offs in performance and efficiency [4]. Rather than introducing new approximation circuits, this work systematizes and evaluates existing designs within a unified framework that separates logic, algorithm, and topology, and enables consistent cross-layer circuit- and application-level analysis to reveal practical design trade-offs.

II. Design Process of Approximate PIA Circuits

The most common design approach for approximating circuits is to redefine logic based on exact functions by omitting, adding, or modifying certain elements. Approximate functions in CMOS are synthesized using standardized methods, which makes it easy to compare their performance metrics [7], [8]. The flexibility of memristive crossbars enables a large design space, much of which remains underexplored compared to CMOS-based technologies [4]. Here, we outline the design process for PIA approximations and discuss the key design-space options and constraints involved. An overview of the design process for approximated circuits is illustrated in Figure 2 and will be explained in the following sections. Although sections can be evaluated sequentially, true optimization requires leveraging the dependencies between them. As shown in [6], the approximations were crafted to support an efficient algorithmic implementation using `IMPLY` operations.

A. PIA Adders as the showcase

As addition is one of the most critical operations in modern computing, extensive research has focused on the design of approximate adders [7]. Among approximate adder designs based on memristive PIA circuits, `IMPLY` logic has emerged as the most widely adopted approach [4]–[6], [9]–[11].

In [6], four adders (SIAFA 1-4) based on the serial topology were introduced, which tried to reduce the number of steps with an approximate truth table suitable for `IMPLY` operations. In [9], an approximation based on `NAND` operations was introduced, as they can be represented with only two `IMPLY` operations. Four adders (SSAXA 1-4) in the semi-serial topology were proposed in [5]. The approximations utilize the parallelization capabilities of the structure and try to reduce the number of steps while minimizing the propagated error. In [10], two serial adders were introduced that either disregard (SAID 1) or reduce (SAID 2) the dependency of the carry-in. To further reduce the number of steps, they also implemented the algorithm from SAID 2 in the semi-parallel topology (SPAID). Adders that completely disregard carry propagation (NoCarry/NoCarry+) were proposed in [4]. They utilized the unique properties of all four `IMPLY`-based topologies and introduced eight different implementations that use specific mapping schemes to optimize each approach. Finally, a derivative of [4] was proposed in [11], which adaptively truncates the adder if all higher bits are '0'. Here, we analyze the design process of approximated algorithms for memristive PIA, compare State-of-the-Art (SoA) approximated adders on circuit-level metrics, and evaluate them in image processing & machine learning tasks to extract the Pareto front.

B. Methodology

1) Design Approximation: The design process often starts with a behavior-level analysis of the exact logic and its potential approximation. The approximated functions can be represented either as truth tables or Boolean equations. They share this property and the resulting error with CMOS approaches. Therefore, the designs of the vast literature on CMOS-based approximations [7], [8] can be utilized for this step. However, to avoid inefficient approximations, it is important to consider the following steps when approximating the exact logic. The actual implementation of these approximations for PIA differs significantly and requires additional considerations, as discussed in the next two sections. The designs from [5], [6], [9] use similar approximations, where one C_{out} is erroneous (at different placements) and the relation $Sum \approx \overline{C_{out}}$ is utilized. In [4], [10], [11], the carry-out is not calculated at all and is either disregarded or an input is propagated instead.

2) Algorithmic Implementation: The logical equations that represent the approximate function, must now be converted to an equivalent sequence of atomic operations. The available set of operations $\mathcal{S}$ depends on the logic form. For example, with `IMPLY` logic, the set of atomic operations is constrained to $\mathcal{S} = \{\rightarrow, \bot\}$, consisting only of `IMPLY` and False operations. To allow the processing of general logical functions, the atomic operation set $\mathcal{S}$ must be functionally complete. The sequence of operations is described as a stateful algorithm that underlies constraints such as the number of available memristors (a topological constraint) or unique properties only present within stateful logic. A common example of constraints in stateful logic schemes is that storing results directly within a memristor inherently overwrites its original state information. Additionally, certain operations—such as the `NOR` and `OR` gates in MAGIC [3] and the `NOT` operation in `IMPLY` logic—require the output memristor to be preconditioned (set or reset) to ensure correct functionality. E.g., the algorithms from [4]–[6], [9], [10] require reset work-memristors to store inversions of inputs, which are used to calculate the basic logical connectives `OR` ($\overline{a} \rightarrow b \equiv a \vee b$) and `NAND` ($a \rightarrow \overline{b} \equiv \overline{a \wedge b}$).

3) Topology Mapping: The stateful algorithm (sequence of operations) must then be mapped to topologies (circuits) that can execute it. There may be various topologies that are compatible with the same equation. The difference lies within

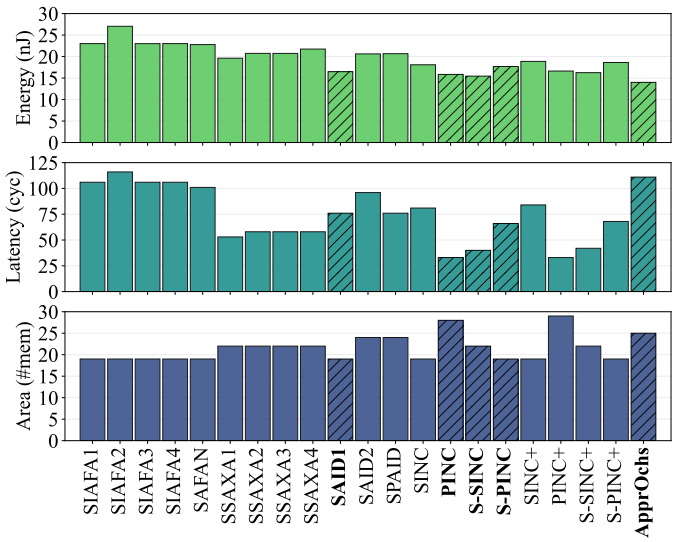

Fig. 3: Comparison of circuit-level metrics for SoA approaches with 5/8 adder approximated. Hatches indicate Pareto designs.

the resulting circuit-level metrics that depend on this mapping. In [4], the same two approximate algorithms are mapped to four different topologies, which result in drastically different metrics. The algorithms from [5], [6] are also very similar, but the choice of topology alone leads to stark differences. These can differ in the number of available memristors and resistors, the number of computational areas that can be parallelized, and other constraints such as connectivity between processing sections through switches [4]. Hence, it is meaningful to consider both top-down and bottom-up designs, where choices are affected by all three layers of the final solution (i.e., design, algorithm, and topology).

III. Comparison of SoA Approximations

In this section, we review and compare SoA approximate adders on circuit-level metrics to assess what approximate computing can contribute to memristive PIA. For a fair comparison, all algorithms from [4]–[6], [9]–[11] were re-implemented and simulated in SPICE using the same memristor model and control logic parameters as in [4]; this includes approaches previously co-authored by the present authors. No published performance numbers were reused.

Real memristor devices are subject to variability in switching thresholds, limited write endurance, and fabrication inconsistencies, which can affect the reliability of stateful logic operations in practical deployments [12]. For each SoA design, we evaluated energy consumption, step count, and area. The approximate adders were inserted in the lower bits of an Ripple Carry Adder (RCA), while the upper bits were computed exactly using the same topology. We denote the ratio of approximated to total bits as Approximation Degree (AxD); for example, $5/8$ refers to an 8-bit RCA with five lower bits approximated [6], [9]. A direct comparison with CMOS-based approximate adders is omitted due to fundamental differences in computation models, which make latency, energy, and area not directly comparable.

We present the circuit-level metrics for all SoA approaches with an AxD of $5/8$ in Figure 3. In this evaluation, switches are excluded from the metric because the circuits can be realized within a 1T1M crossbar array; doing so only increases the complexity of the control logic, as the switches reside beneath the memristors. The Pareto front, highlighted using hatching, consists of five algorithms corresponding to the most efficient designs within each topology. Unlike CMOS-based synthesis, memristive PIA supports multiple circuit and topology realizations for the same approximation, often yielding markedly different circuit-level outcomes. Within this space, the PINC and S-SINC algorithms [4] in the parallel and semi-serial topologies achieve the lowest step count and high energy efficiency. The SAID-1 [10] and S-PINC [4] designs in the serial and semi-parallel topologies minimize area. Finally, the ApprOchs adder [11] delivers the highest overall energy efficiency. To represent the different circuit-level criteria with a single value, we introduce the Energy-Steps-Area (ESA) score

$$ESA(m_E, m_S, m_A) = \frac{10^{-4}}{m_e \cdot m_S \cdot m_A}, \tag{1}$$

where m_E, m_S, and m_A denote the total energy consumption (Joule), number of steps, and area (number of memristors) of a partially approximated RCA. The normalization factor 10^{-4} improves readability. ESA equally weights energy, latency, and area and thus serves as a neutral first-order metric. Since different applications may prioritize these metrics differently, ESA is not intended as an application-optimized score; instead, Pareto-front analysis should be used to select designs under specific constraints. The corresponding exact serial, parallel, semi-serial, and semi-parallel `IMPLY`-based RCAs achieve ESA scores of 0.77, 1.60, 1.76, and 1.00, respectively. The ESA results for SoA designs at AxD $5/8$ are shown in Figure 4, where the semi-serial and parallel algorithms from [4] outperform others by at least 30% due to their fast and energy-efficient implementation.

This analysis yields a practical design principle: error-localized approximations [4], [11] align well with stateful logic constraints, whereas irregular carry corruption [5], [6], [9], [10] increases cost and reduces robustness, directly guiding future approximate PIA design.

IV. Application in Error-resilient Applications

We analyze and compare the results of the approximate adder from [4]–[6], [9]–[11] at the application-level. We first employed them in four commonly used image processing applications: image addition, image subtraction, grayscale filtering, and average pooling. Unlike most related work, which evaluates only one or a few images (risking input-dependent bias), we use the full datasets from [4] for a fair comparison, reporting mean Peak Signal-to-Noise Ratio (PSNR) and Mean Structural Similarity Index Metric (MSSIM) per task. In the literature, a PSNR of over 30dB and a MSSIM above 0.8 is deemed acceptable quality. We show mean PSNR and MSSIM versus ESA-Score for all adders at AxD $5/8$ in Figure 4, since all existing solutions fall short of the 30dB PSNR threshold at higher degrees. For image addition and subtraction, the approaches from [4], [11] retain the highest quality, which improves over other adders by up to 30% and 59% in terms of PSNR and MSSIM. In the grayscale filter and average pooling

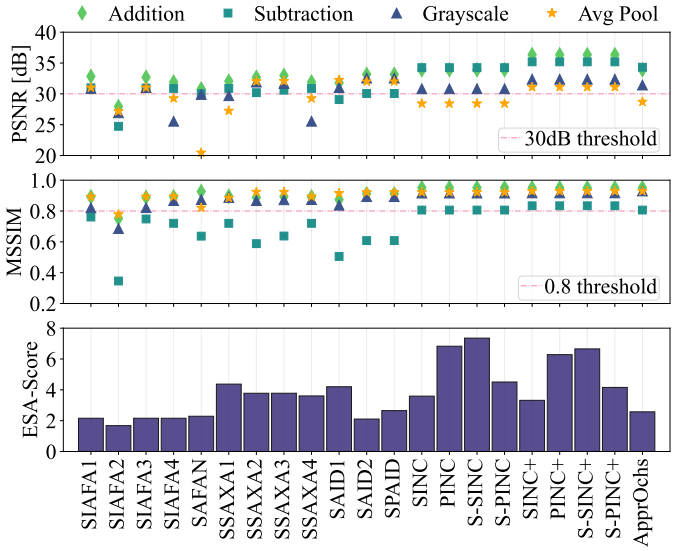

Fig. 4: Average image quality results of four applications over ESA-Score for SoA approaches with 5/8 adders approximated.

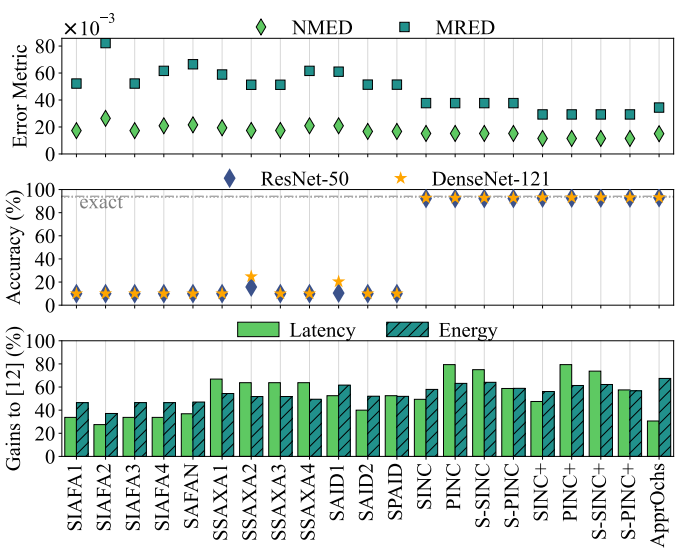

Fig. 5: Comparison of error metrics, the accuracy on CIFAR-10, and gains to [12] using partially approximate networks.

applications, the SAID2 and SPAID adder from [10] achieve the highest PSNR by 1-21%. When considering the ESA score, the PINC(+) and S-SINC(+) variants presented in [4] offer the most effective balance between output quality and significant reductions in latency and energy consumption.

To demonstrate the applicability of approximate adders in more complex workloads, we integrated them into array multipliers (configured with an AxD of $5/8$) and evaluated their performance on the CIFAR-10 dataset using the 8-bit quantized ResNet-50 & DenseNet-121 architectures. This evaluation aims to broaden the understanding of which PIA-based approximations can be effectively employed in machine learning applications, an aspect that has not been addressed in prior work. We initialized the networks with pretrained weights and replaced each multiplication with the proposed partially approximate array multipliers. The models were then fine-tuned for 50 epochs using an approximation-aware training strategy. The resulting error metrics, accuracies, and corresponding improvements in energy efficiency and latency relative to the most optimized serial exact adder [12] are summarized in Figure 5. The results reveal that ML suitability is largely determined by the structural nature of the approximation. To better understand this behavior, we also report common arithmetic error metrics, which quantify the magnitude and distribution of arithmetic errors in approximate adders [7]. The approaches from [5], [6], [9], [10] corrupt carry propagation or introduce non-monotonic behavior, causing error amplification across multiply-accumulate operations and leaving accuracy near 10–25%, far below usable performance. In contrast, the adders from [4], [11], which bound errors to low-significance bits, allow the network to compensate through fine-tuning, yielding only minor accuracy losses of 0.66 to 1.45 percentage points. This modest accuracy loss is offset by up to 62% energy savings and 77% latency reduction, underscoring the efficiency of approximate PIA.

While deeper networks may exhibit increased robustness due to redundancy, cascading effects remain workload-dependent, motivating future evaluation on larger-scale datasets and architectures to further study error propagation behavior.

V. Conclusion

In this work, we used a taxonomy to structure memory-centric computing approaches and to better position the type of designs and circuits studied in this paper. We provided an overview of the active research field of approximate memristive PIA and discussed the corresponding design process. We compared SoA approximated adder on circuit-level, extracted the Pareto front, and proposed a new evaluation metric. To showcase the applicability of these adders, we evaluated them in the four most common image-processing applications and extracted the most efficient trade-offs from the SoA. We applied the approximations in machine learning, where we analyzed approximate networks on CIFAR-10. To provide a comprehensive overview of the field, we ran several independent experiments to fill the gap that exists in the literature due to incomplete or incomparable numbers in certain works of the literature. We hope that this helps colleagues in the field with their future research on this topic.

References

[1] N. TaheriNejad. In-memory computing: Global energy consumption, carbon footprint, technology, and products status quo. pp. 1–6, 2024.

[2] J. Borghetti *et al.* 'Memristive' switches enable 'stateful' logic operations via material implication. *Nature*, 464:873–876, April 2010.

[3] S. Kvatinsky et al. MAGIC; memristor-aided logic. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 61(11):895–899, Nov 2014.

[4] F. Seiler and N. TaheriNejad. Accelerated image processing through imply-based nocarry approximated adders. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 71(11):5141–5154, 2024.

[5] F. Seiler and N. TaheriNejad. Efficient image processing via memristive-based approximate in-memory computing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 43(11):3312–3323, 2024.

[6] S. E. Fatemieh *et al.* Fast and compact serial imply-based approximate full adders applied in image processing. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 13:175–188, 2023.

[7] E. Napoli *et al.* Approximate full-adders: A comprehensive analysis. *IEEE Access*, 12:136054–136072, 2024.
[8] H. Jiang *et al.* Approximate arithmetic circuits: A survey, characterization, and recent applications. *Proceedings of the IEEE*, 108(12):2108–2135, 2020.
[9] S. Asgari *et al.* Energy-efficient and fast imply-based approximate full adder applying nand gates for image processing. *Comput. Electr. Eng.*, 113:109053, 2024.
[10] N. Kaushik *et al.* High-speed serial and semi-parallel imply-based approximate adders through memristors for in-memory computing. *2024 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, 2024.
[11] D. Ochs *et al.* Approchs: A memristor-based in-memory adaptive approximate adder. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 15(1):105–119, 2025.
[12] F. Seiler *et al.* An efficient robust serial imply-based in-memristor adder. In *2025 Cross-Disciplinary Conference on Memory-Centric Computing (CCMCC)*, pp. 1–7, 2025.

Fabian Seiler is currently pursuing a PhD in the Faculty of Engineering at Heidelberg University, Germany. His research interests include memristors, in-memory computing, and approximate computing.

Nima TaheriNejad is a full professor in the Faculty of Engineering at Heidelberg University, Germany. His areas of work include emerging computing paradigms, cyber-physical systems, and smart health.