RRAM-ECC: Improving Reliability of RRAM-Based Compute In-Memory

Zishen Wan*[1], Brian Crafton*[1], Samuel Spetalnick[1], Jong-Hyeok Yoon[2], Arijit Raychowdhury[1]
[1]School of Electrical and Computing Engineering, Georgia Institute of Technology, Atlanta, USA
[2]Department of Electrical Engineering and Computer Science, DGIST, Daegu, Korea
{zwan63, bcrafton3, sspetalnick3}@gatech.edu, jonghyeok.yoon@dgist.ac.kr, arijit.raychowdhury@ece.gatech.edu
*Equal Contributions

***Abstract*—Compute in-memory (CIM) is an exciting technique that minimizes data transport, maximizes memory throughput, and performs computation on the bitline of memory sub-arrays. This is especially interesting for machine learning applications, where increased memory bandwidth and analog domain computation offer improved area and energy efficiency. Unfortunately, CIM faces new challenges traditional CMOS architectures have avoided. In this work, we explore the impact of device variation (calibrated with measured data on foundry RRAM arrays) and propose a new class of error correcting codes (ECC) for hard and soft errors in CIM. We demonstrate single, double, and triple error correction offering up to 16,000× reduction in bit error rate over a design without ECC and over 427× over prior work, while consuming only 29.1% area and 26.3% power overhead.**

I. Introduction

RRAM is a promising candidate for compute in-memory (CIM) applications due to its multiply-and-accumulate structure in 1T-1R bitcell, non-volatile, high density, and process compatibility [1]. These properties seek to advance machine learning applications with higher throughput and bit-density. Despite these benefits, both CIM and RRAM face several challenges not before faced by traditional CMOS designs. Due to process, temperature, and write-to-write variations, the resistive state of RRAM undergoes both spatial and temporal variations, which is further exacerbated by the desired low voltage. CIM read out multiple rows simultaneously by accumulating current on the same bitline (BL). Therefore, CIM inherently has higher bit error rate (BER) than traditional memory arrays reading one wordline (WL) at a time. Traditional ECC cannot protect CIM because errors in multi-level outputs cannot be localized. Existing efforts to reduce CIM error impact use iterative write-verify [2] that requires high write energy with reduced endurance, or variation-aware training [3] that has low generality with high training cost, or arithmetic codes [4] that protect CIM additive syndromes but cannot identify error location.

To mitigate the impact of RRAM CIM errors, our paper makes three major contributions that will likely have long-term impact on both academia and industry. (1) We propose a new class of ECC scheme, *CIM-SECDED*, for hard and soft errors in CIM and compatible with any number of parallel row accesses. (2) We propose a general technique *Successive Correction* for error correction. Combined with *CIM-SECDED*, we achieve single, double, and triple error correction in CIM for the first time, and demonstrate good scalability. (3) We evaluate the proposed ECC scheme on a 40nm foundry RRAM test-chip array, and achieve over 16,000× BER reduction while consuming only 29.1% area and 26.3% power overhead.

All the details can be found in [5] and [6].

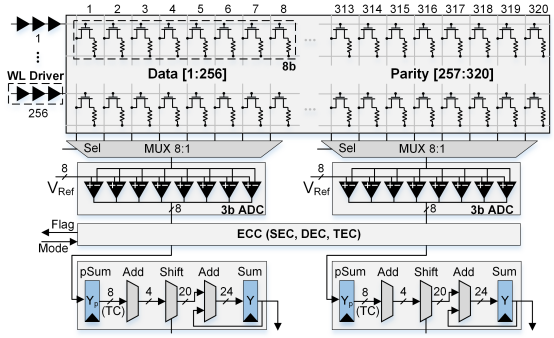

Fig. 1: 256×320 RRAM array. 8 adjacent cells form an 8-bit weight and share a 3-bit ADC through a 8-to-1 multiplexer. Shift and add logic accumulate partial sums and apply corresponding magnitude.

II. Characterization and Design Methodology

Characterizing RRAM Device Variation. The key obstacle to enabling CIM is cell-to-cell variation. Multi-row read significantly increases error chances due to accumulated variation from the same BL. To quantify this error rate, we collect data from a recent 40nm foundry RRAM test-chip [7]. We randomly program the cells with uniform value distribution (0-8 LRS) and perform multi-row read. We observe that when #LRS cell is low (<4) the ADC output is always correct in 8192 samples. When more LRS cells are read, errors occur with increasing frequency. However, we note that errors are always constrained to ± 1 errors. This observed property has special implications for both error correction and detection. Like traditional SECDED, a ± 1 can be detected and localized using a hamming code.

CIM-SECDED Scheme. Based on the characterization, we propose *CIM-SECDED*, a single error correction double error detection (SECDED) scheme compatible with CIM for any number of parallel row accesses. Fig. 2a shows the encoding, decoding, and correction steps for *CIM-SECDED*. The example uses a (4, 3) hamming code for each row and requires 2 bits for double error detection and sign detection (± 1). Compared to traditional SECDED, *CIM-SECDED* requires only 1 additional bit for sign detection. After the error is localized and checked, the sign of the error is computed using the checksum. The sign is obtained using a LUT (off-chip) containing various possible outcomes. Lastly, the address is decoded and the error is applied to the victim ADC output. For most CIM applications encoding can be done off-chip, and only decoding is required on-chip.

Successive Correction by Detection. To effectively correct errors while maintaining high performance, we propose *Successive Correction by Detection*. Upon error detected during CIM, we re-read the WLs leading to the error at fewer WLs/cycle. Fig. 2b shows an example of *Successive Correction*. We pad

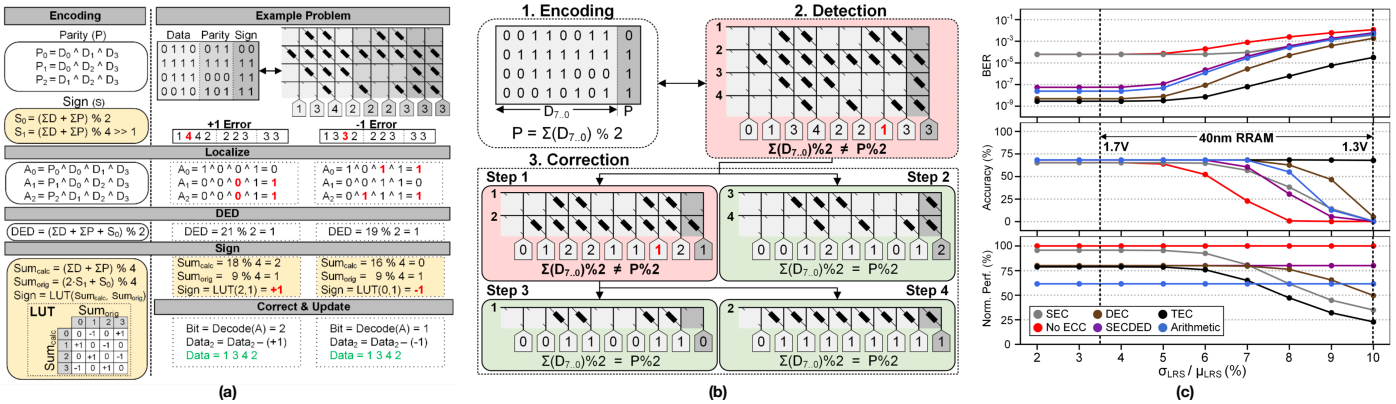

Fig. 2: (a) Example of *CIM-SECDED*. (b) Example of *Successive Correction*. (c) Evaluation of classification accuracy, BER, and normalized performance versus cell-to-cell variation ($\sigma_{\text{LRS}}/\mu_{\text{LRS}}$) for ResNet18 on ImageNet.

each word with a parity bit that is read and accumulated on WL during CIM. The error is detected by comparing parity bit with ADC outputs. Upon detecting the error, correction is performed by re-reading WLs where we recursively break down WLs in 2 sets. Soft errors are typically corrected after the first recursion. Hard errors can be quickly localized along one branch and minimize WLs re-reading time. This technique is especially powerful when many WLs are enabled ($\geq$32).

Double Error Correction. To further enhance the correction capability, *Successive Correction* is paired with *CIM-SECDED* to enable both double error correction (DEC) and triple error correction (TEC) after detection. Upon double error detection during *CIM-SECDED*, *Successive Correction* is performed. At each recursive step, SECDED is performed and continues until only 1 WL is enabled. This scheme can perform DEC for all 2-error syndromes. By enabling more detection capability, *Successive Correction* can provide greater correction capability.

Triple Error Correction. Triple error detection (TED) can be used to implement TEC with *Successive Correction*. Because SECDED has a hamming distance of 4, it can also be used for TED. Using *Successive Correction*, this code can be used to correct 3 errors. Instead of decoding syndromes as 1-error or 2-error like SECDED, TED simply detects any non-zero syndrome as an error. During CIM, SECDED is used to perform TED. Upon error detection, *Successive Correction* is performed. At each recursive step, TED is again performed and the process continues until only 1 WL is enabled. If only 1 WL is enabled, then SECDED is performed so errors can be corrected. DEC and TEC share the same decoder in hardware implementation.

III. Implementation and Evaluation Results

Hardware Implementation. To architect our ECC permutations, we consider the 256×256 RRAM macro [7] used for characterization (Fig. 1). Using measured area and power from RRAM macro as baseline, we evaluate each ECC scheme by synthesizing RTL implementation. Arithmetic code [4] incurs 70.9% area and 66.1% power overhead. In contrast, our proposed SEC, CIM-SECDED (DEC), TEC incurs 3.3%, 29.0%, 29.1% area overhead and 3.2%, 26.3%, 26.2% power overhead.

Simulation Framework. To evaluate the performance, BER, and accuracy of each ECC scheme, we construct a cycle-accurate simulator capable of executing tensor operations using CIM with non-idealities based on our measured data. We embed performance counters in ADC and sub-array objects to track cycles, errors, and ECC events. As output, the simulator produces a table with performance counters and all intermediate layer activations that are verified against a TensorFlow design.

Soft Error and Hard Error Evaluation. For soft error, we evaluate over the measured LRS variation (Fig. 2c). DEC and TEC yield the lowest BER for all cases. Arithmetic codes [4] yield lower performance and higher BER than DEC and TEC due to its high overhead on 8-bit operations. At 4% LRS variation (1.7V), TEC achieves 14.9× BER reduction and 27.8% speedup over arithmetic codes. At 6% LRS variation (1.5V), TEC achieves 427.1× BER reduction and 23.2% speedup over arithmetic code. For hard error, we sweep over Stuck-at-Fault (SAF) rate. For lower SAF rates (10^{-6}), DEC and TEC yield 58.9× and 280.4× better lower BER than arithmetic code while achieving 29% better performance. ECC can greatly reduce BER and thus we can enable more parallel WLs while achieving the same BER. We find that TEC can enable 32 WLs at lower BER than 8 WLs with no ECC. At 3.5% variation, we observe a 2.32× speedup and >200× reduction in BER. Thus TEC can achieve higher peak performance for a given BER over the range of variation on our 40nm foundry RRAM array.

Generability and Scalability. Our *CIM-SECDED* + *Successive Correction* approach can be generalized in other memory and error distribution. Beyond RRAM, it can be used for PCM, MRAM-based CIM, and results will depend on SAF and variation. Beyond normal distribution in our measured RRAM data, our method can be applied to other error distributions (e.g., log-normal). Our approach is scalable in that can enable any #WLs, correct >3 errors, and protect arbitrary sized data blocks like standard SECDED codes (e.g., 32/7, 64/8).

References

[1] S. Yu *et al.*, "Compute-in-memory chips for deep learning: Recent trends and prospects," *IEEE Circuits and Systems Magazine*, 2021.

[2] J. Wu *et al.*, "A 40nm low-power logic compatible phase change memory technology," in *IEDM*, 2018.

[3] Y. Long *et al.*, "Design of reliable dnn accelerator with un-reliable reram," in *DATE*, 2019.

[4] B. Feinberg *et al.*, "Making memristive neural network accelerators reliable," in *HPCA*, 2018.

[5] B. Crafton *et al.*, "Cim-secded: A 40nm 64kb compute in-memory rram macro with ecc enabling reliable operation," in *A-SSCC*, 2021.

[6] B. Crafton *et al.*, "Improving compute in-memory ecc reliability with successive correction," in *DAC*, 2022.

[7] S. Spetalnick *et al.*, "16.2 a 40nm 64kb 26.56tops/w 2.37mb/mm2 rram binary/compute-in-memory macro with 4.23× improvement in density and >75% use of sensing dynamic range," in *ISSCC*, 2022.