Enabling Error Correcting Code (ECC) Support in PIMeval_PIMbench

Michael Shires, Kelly Farran, Derek Hansen, Kevin Skadron

2026 OSCAR Workshop

Department of Computer Science

Background

- **Processing-in-memory (PIM)** reduces the cost of data movement by performing computation close to or within DRAM

- Server-class environments maintain strict reliability requirements and long-running workloads
    - Support for Error Correction Codes (ECC) essential for maintaining data integrity

- **The problem:** Existing PIM simulation frameworks model performance and energy, but do not model ECC or its associated overheads

Proposed Solution

- PIMeval-PIMbench* is an open-source framework for modelling performance and energy of PIM architectures and workloads

- We extend PIMeval to model ECC overheads throughout the PIM hierarchy

- Our extension includes:
    - Configurable ECC schemes
    - Support for easy addition of new ECC schemes
    - Reliability-aware modelling of performance and energy tradeoffs

* Siddique *et al.*, *Architectural Modeling and Benchmarking for Digital DRAM PIM*, IISWC 2024

Proposed Contribution

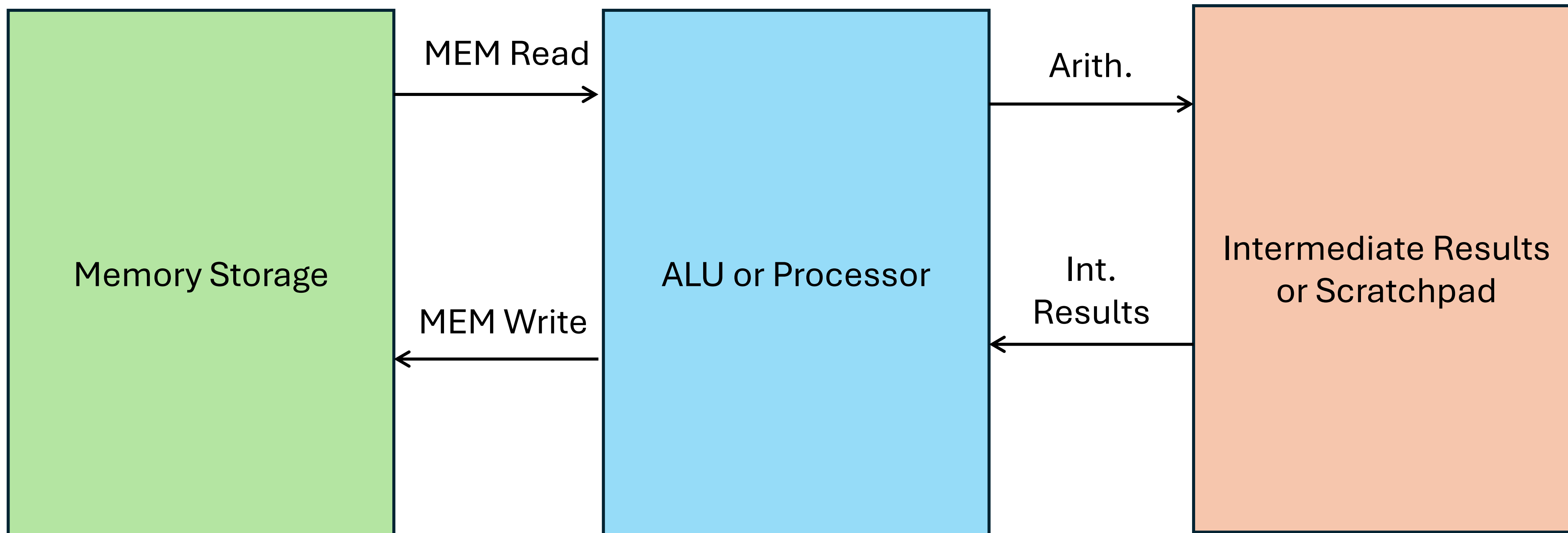
- Our project's proposed contribution is twofold:

Propose ECC methodologies for PIM and evaluate their performance and tradeoffs.

Provide a clean, flexible API for PIMeval-PIMbench to allow end-users to easily include ECC into proposed PIM architectures.

Methodology

- Key question – where to perform ECC, and at what granularity?
  - We propose Read, from memory and scratchpad
- Read, Write, intermediate results, and arithmetic



UNIVERSITY of VIRGINIA

Features

- 128+8 On-Die ECC IAW JEDEC (Enable ON/OFF)
- Controller Configurable ECC at user-provided granularity
  - Can stack ECC granularities, i.e. 64+8 and 256+16
  - Select from several error correction codes: SEC/DED, SECDED, RS, CRC32

ECC Methods

SECDED	CRC32	Reed-Solomon
k+1 ECC bits (k = data width)	32 ECC bits per CRC application	8 ECC bits for every 64 bits of data
Can both detect and correct; more lightweight	Faster output and more energy efficient, but can only detect	Can both detect and correct; higher protection from burst-error correction

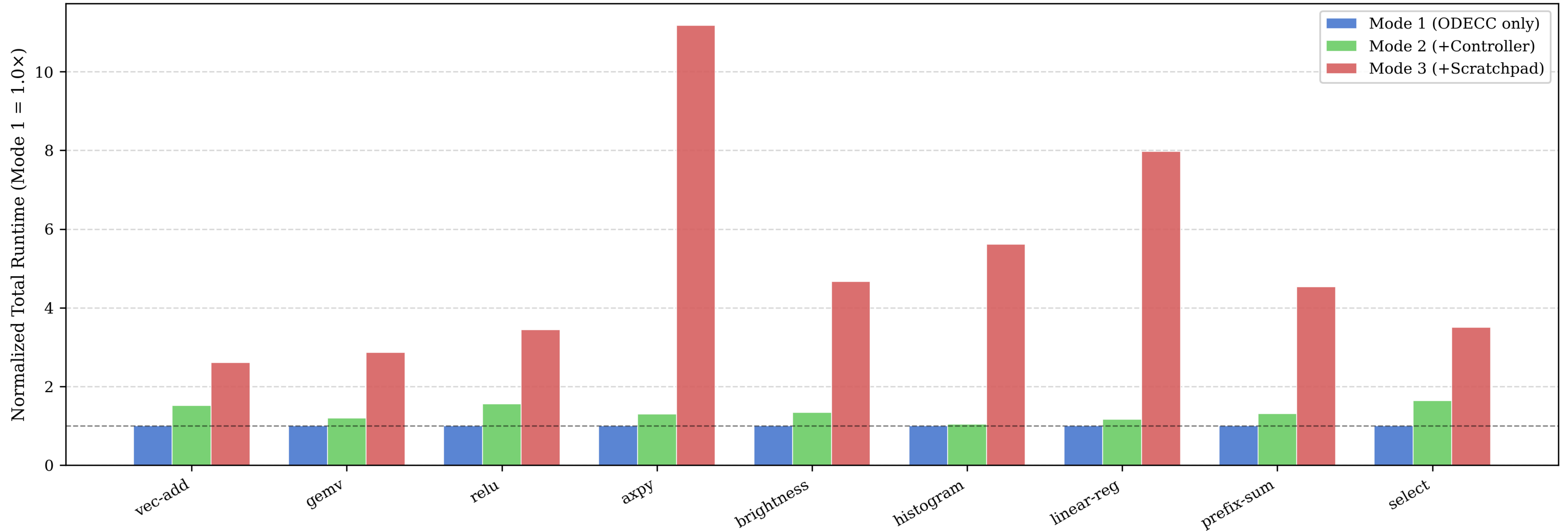
Features

- Two modes of running benchmarks – Analytical and Simulated
  - Analytical – energy and latency overhead are added in as a statistic
  - Simulated – ECC is performed

- InjectBitError and InjectBurstError API added for Simulated mode
  - Both follow a Poisson distribution that user's can modify execution rate.

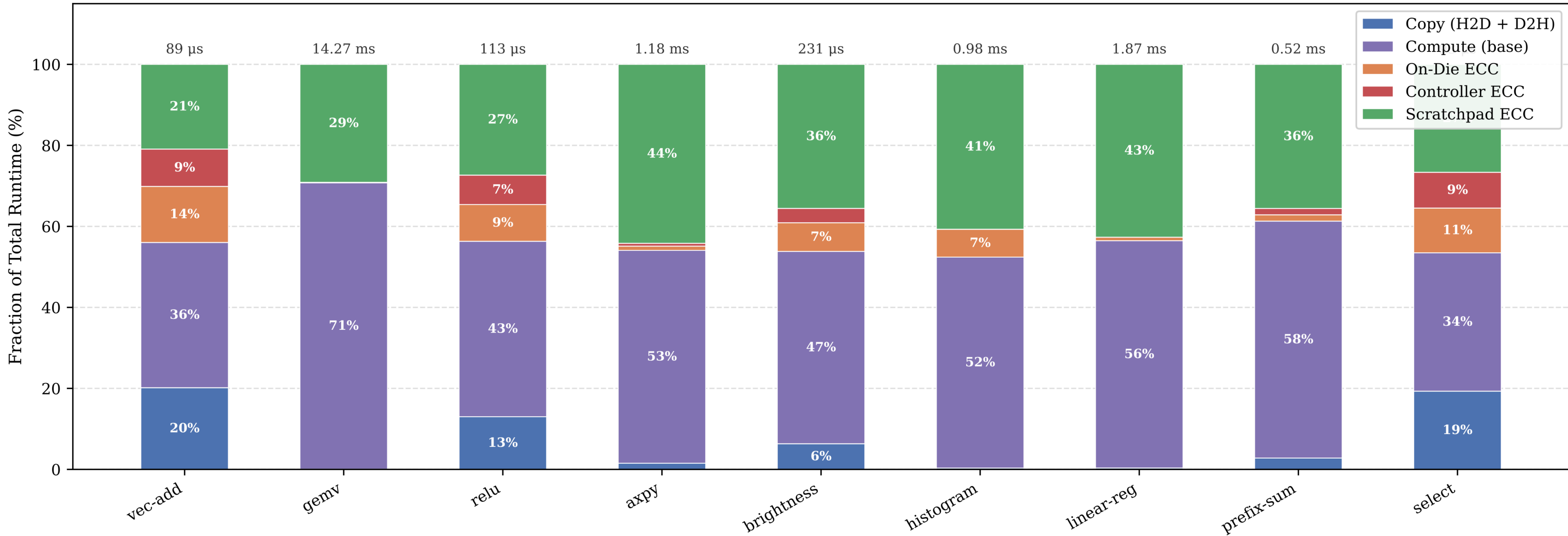
UNIVERSITY of VIRGINIA

Results – Simulated Runtime with different ECC Configurations



Results – Runtime Distribution of Workloads with All ECC Enabled

UNIVERSITY of VIRGINIA



Complementary Contribution

Monte Carlo ECC Tradeoff Tool

UNIVERSITY of VIRGINIA

- Explore reliability, area, and latency tradeoffs of ECC for stateful components in PIM units before full-system simulation
  - *Register file, scratchpad, cache (optional)*

- Configurable component sizes, FIT rates, and ECC schemes

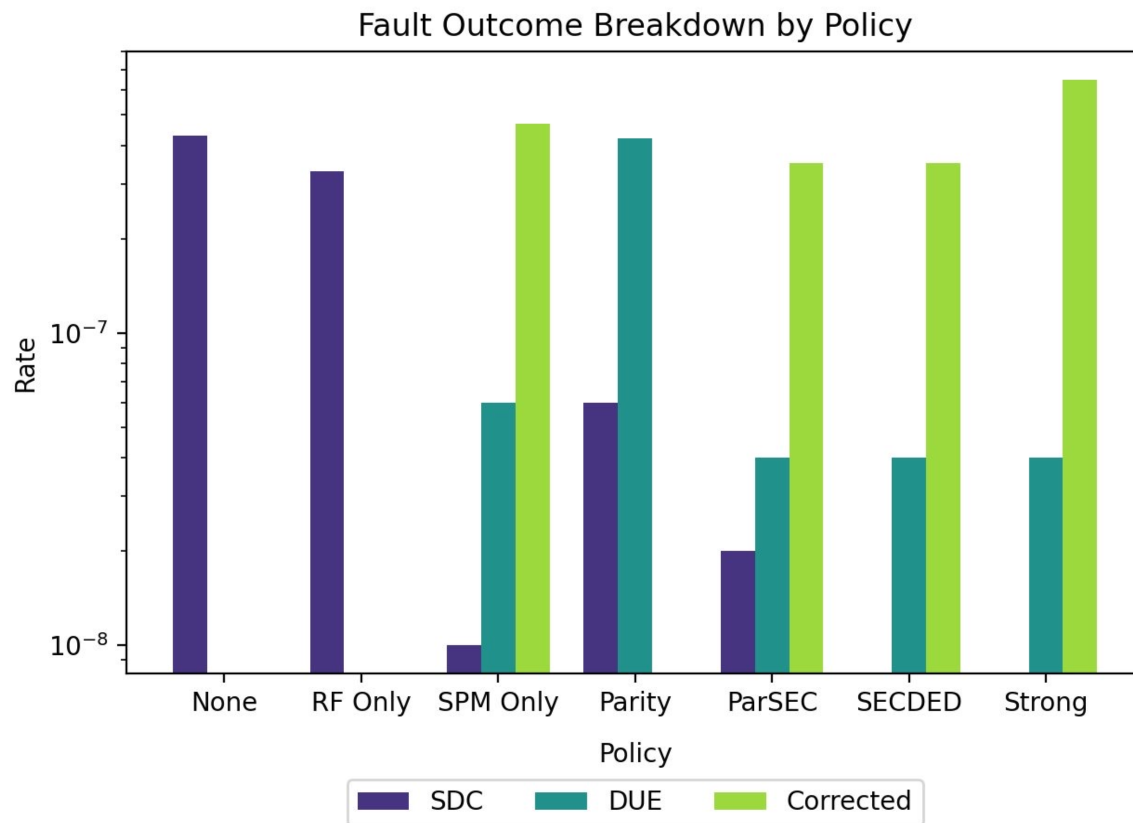
- Estimates corrected, detected unrecoverable (DUE), and silent data corruption (SDC) rates using Monte Carlo simulation

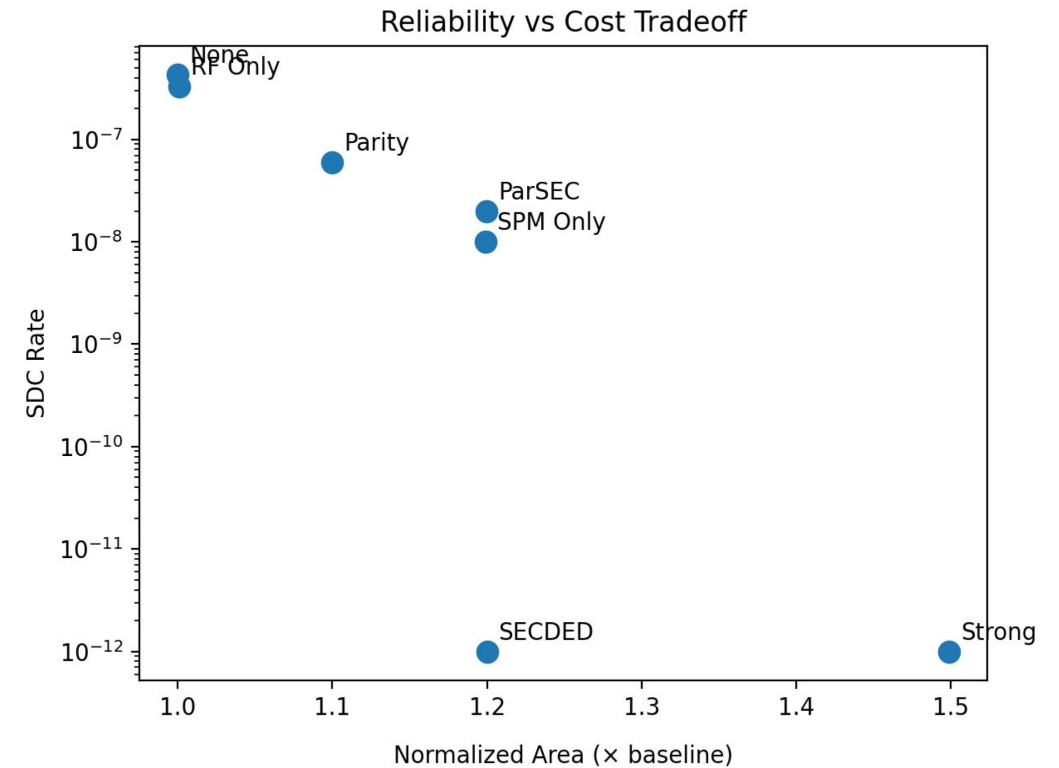
Tool link: https://github.com/klfarran/pim-ecc.git

Complementary Contribution

Monte Carlo ECC Tradeoff Tool

UNIVERSITY of VIRGINIA





Key Takeaways

- ECC modelling is absent in existing PIM simulators, limiting realistic evaluation of reliability-aware PIM-enabled systems
- We extend PIMeval-PIMbench with an ECC-aware modelling layer that enables configurable protection schemes and granularities across the memory hierarchy
- Our contributions enables system-level analysis of how ECC integrates with PIM performance, energy, and reliability tradeoffs

PIMeval + ECC Demonstration

```
# 1. Clone and build
git clone https://github.com/UVA-LavaLab/PIMeval-PIMbench
cd PIMeval-PIMbench
./setup.sh
make

# 2. Build the ECC benchmarks and tests
make -C tests/ecc perf

# 3. Run the demonstration script
./demo_ecc_features.sh
```

Setup – Config File or CLI Arguments

Example Configuration File

```
# Tier 1 — on-die DRAM ECC
odecc=1
odecc_data_width=128
odecc_parity_width=8

# Tier 2 — controller ECC
ecc=1
ecc_type=secded # secded | crc32 | rs
ecc_granularity=64

# Tier 3 — scratchpad
scratchpad=1
scratchpad_ecc=1
scratchpad_word_bits=32
```

Different Execution Options

```
# A) config file
./vec-add.out --pim-
sim_config=my_pim_ecc.cfg -l 65536

# B) inline CLI flags
./vec-add.out --pim-ecc=1 --pim-
ecc_type=secded -l 65536

# C) env vars (works with ANY benchmark)
PIMEVAL_ECC=1 PIMEVAL_SCRATCHPAD_ECC=1
./vec-add.out -l 65536
```

Questions?