Sisyphus: Cross-Layer Efficiency Across NVM Technologies in Compute-in-Memory Architectures

Ali Nezhadi[†*], Odysseas Chatzopoulos[‡*], Mahta Mayahinia[†], George Papadimitriou[‡],
Mehdi Tahoori[†] and Dimitris Gizopoulos[‡]
[†]Karlsruhe Institute of Technology, Karlsruhe, Germany, Email: {ali.nezhadi, mahta.mayahinia, mehdi.tahoori}@kit.edu
[‡]University of Athens, Athens, Greece, Email: {od.chatzopoulos, georgepap, dgizop}@di.uoa.gr
[*]*Equally contributing first authors.*

***Abstract*—Compute-in-Memory (CiM) employing Non-Volatile Memory (NVM) technology is an emerging paradigm that promises higher power efficiency for important data-intensive computations. The performance, power, and resilience properties of emerging NVM technologies determine the efficiency of architectures built around processors and computational memories, and affect design decisions. Thus, fast exploration of the broad design space is necessary to assist decision-making. We present *Sisyphus*, the first cross-layer framework built to facilitate computer architecture research when such an exploration is required. Sisyphus incorporates detailed technology information for various CiM circuit designs based on STT-MRAM, ReRAM, and PCM technologies and integrates them in fast microarchitecture level system models in gem5 to evaluate performance, power, and resilience (through fault injection) across a large space of design options. Sisyphus' holistic modeling enables the comprehensive evaluation of all efficiency aspects during the execution of actual workloads on the CPU-CiM architecture. This allows for comparisons to a baseline CPU-only system. In our experimental evaluation, we demonstrate how Sisyphus can derive conclusions regarding the prevalence of one NVM type over another, depending on the prioritized optimization aspect(s).**

I. Introduction

In modern computing systems, only the CPU handles data processing, while other components, such as main and cache memories, are dedicated solely to data storage. This CPU-centric design, commonly referred to as the von Neumann architecture, results in significant data movement between the CPU and main memory, which results in substantial energy consumption and performance degradation—a phenomenon known as the *memory wall* issue. This data transfer accounts for 40% to 60% of total system energy use [1]. Moreover, accessing memory consumes 100 to 800 times more energy than performing integer or floating-point addition operations [2].

To address the memory wall challenge and balance computation energy with data movement energy, there has been considerable interest in *computation-in-memory* (CiM) architectures, particularly those leveraging emerging resistive non-volatile memory (NVM) technologies, allowing energy-efficient computation near the data location. NVMs are devices that can have at least two stable resistive states: low and high (LRS and HRS). Spin transfer torque magnetic RAM (STT-MRAM), redox-based RAM (ReRAM), and phase change memory (PCM) are the promising technologies that have reached the level of production maturity by major foundries and design houses [3], [4]. Particularly for CiM, NVMs can enable analog computing and further improve the energy efficiency. Using CiM, many data-intensive applications (e.g., bitmap-indexed databases, database searches, DNA sequencing, data initialization, image processing, etc.) can be parallelized and hence, accelerated.

Despite these features, NVM-CiM comes along with reliability problems, stemming from their stochastic nature and susceptibility to variation. Therefore, a comprehensive assessment of the NVM-CiM in terms of performance and energy efficiency also requires a quantitative reliability analysis.

Such comprehensive analysis of the NVM-CiM, requires system simulators enabled with CIM capabilities. To this end, several CiM simulators have been proposed, which are qualitatively compared in Table I and further detailed in Section II-E. However, to the best of our knowledge, as demonstrated in Table I, no existing CiM simulator encompasses all the critical features necessary for simulating a modern system in real-world environments. These essential features include (among others) full-system simulation support, extensive configurability of both technology and architecture, and, most importantly, robust reliability assessment. To fill the gap, we propose *Sisyphus* that uniquely fulfills these requirements, establishing itself as the most advanced and state-of-the-art solution available for assessing cross-layer efficiency and reliability of CiM-based architectures.

Sisyphus allows researchers to thoroughly evaluate any new protection or detection scheme from device-level to algorithmic and application-level solutions, and provide reliable solutions for these emerging technologies. The major contributions of this paper are as follows:

1) We extend gem5 [5] to support highly configurable CiM architectures based on STT-MRAM, ReRAM, and PCM, the prevailing NVM technologies. This extension allows accurate performance, power, and reliability evaluations, and broad design space exploration.
2) We develop and provide a comprehensive Application Programming Interface (API) for the user application code to call and use different CiM operations.
3) Sisyphus implements statistical fault injection (SFI) at the microarchitecture level for all hardware structures of the CPU and the CiM module operations. To the best of our knowledge, Sisyphus is the first SFI framework for CiM-enabled architectures.

Regular Paper

TABLE I
A COMPARISON OF WELL-KNOWN CIM SIMULATORS.

Simulator	Tech.	Full-System Support	Arch.	Detailed Circuit Model	Reliability Analysis
CIM-SIM [8]	NVM	✗	Boolean, NN	✗	✗
MNSIM [9]	ReRAM	✗	NN	✓	✗
NeuroSim [10]	ReRAM	✗	NN	✓	✗
MultiPIM [11]	CMOS	✓	Complex	✗	✗
PIMulator [12]	FPGA SoC	✓	Complex FPGA	✗	✗
SIM2PIM [13]	CMOS	✓	Complex	✗	✗
PIMSim [14]	CMOS NVM	✓	Complex	Rely on other	✗
Sisyphus	NVM	✓	Flexible	✓	✓

4) Sisyphus reports accurate power estimation for CiM architectures through workload-dependent on-chip as well as memory and CiM power consumption by extending the widely used McPAT framework [6] with accurate NVM models from NVSim [7] with CiM extensions.

The rest of this paper is organized as follows. In Section II, we review the essential background and related work. In Section III, we elaborate on the core CiM architecture, followed by Section IV that presents the experimental setup. Section V contains the quantitative co-exploration of the power, performance, and reliability, and finally, Section VI concludes the paper.

II. BACKGROUND AND RELATED WORK

A. Non-volatile Memory Technology

1) STT-MRAM: The main storage element of the STT-MRAM is the *magnetic tunnel junction (MTJ)* which consists of three layers. Two ferromagnetic layers sandwich an insulation layer. The magnetic orientation of the *free* layer can be rotated, while the magnetic orientation of the *pinned* layer is fixed. In the MTJ, the relative magnetic orientation of the free and pinned layers determines the resistive state. In the case of parallel magnetic orientation, the device is in the LRS, and otherwise, it is in the HRS.

2) ReRAM: The ReRAM cell is also a stack of three layers. Two metallic (Ohmic and Active) electrodes sandwich an oxide-based insulation layer. Within the oxide-based insulation layer, the oxygen vacancies (V_O^{2+}) can form a conducting filament connecting the metal electrodes. If the conducting filament exists in the insulation layer, the device is in LRS, and otherwise, it is in HRS.

3) PCM: PCM also consists of three layers. Two metallic electrodes and, in between, a phase change material. The phase change material can be either in *crystalline* or *amorphous* phases, which results in the LRS and HRS, respectively.

B. CiM-compatible operations and their NVM-based implementations

1) Boolean operation: Scouting logic is a general approach for performing NVM-CiM for vector-wise Boolean operations by activating multiple rows and modifying the sensing threshold (see Figure 1(a)). In this approach, the bit line (BL) current is compared with a reference current, and the output is determined based on this comparison [15]. If the BL current is less than the reference, the output is '0'; otherwise, it is '1'.

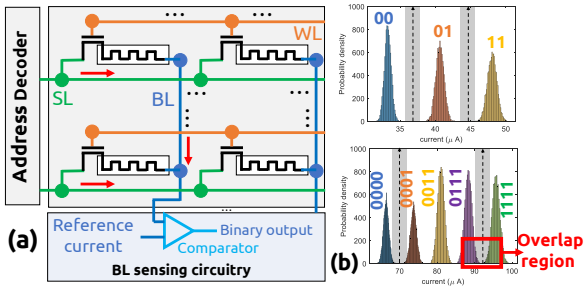

Fig. 1. (a) Performing the scouting Boolean NVM-CiM in the crossbar organization of NVM, (b) smaller margin between the distinct current states in the case of more operands (top: 2 operands, bottom: 4 operands).

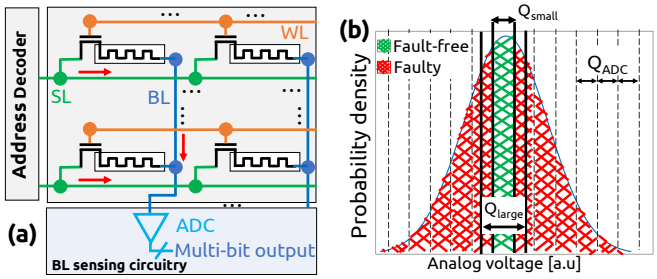

Fig. 2. (a) Performing the MAC operation in the crossbar organization of NVM, (b) Decision failure in the case of MAC operation utilizing ADC, the Q_{ADC} shows the quantization level of the ADC, the probability associated with the green region is for the fault-free scenarios, while the red regions show the probability of decision failure, also, enlarging the Q_{ADC} (Q_{large} instead of Q_{small}), i.e., fewer bits for ADC quantization, reduces the probability of decision failure.

2) Multiply-Accumulate (MAC): MAC operation is a kernel in various artificial intelligence (AI) applications and has a general form of $[V_1, \ldots, V_N] \times [G_1, \ldots, G_N]^T$. This operation can be accelerated using the analog NVM-CiM. By applying the first vector (V) as the encoded binary voltage to the source line (SL) and writing the second vector (G) into the NVM elements (see Figure 2(a)), the bit line (BL) current shows the result of the MAC operation. While the realization of MAC operation is done using analog NVM-CiM, the interface with the rest of the microarchitecture is still fully digital; thus, the result of the MAC operation, i.e., the BL current, needs to be digitized using the analog-to-digital converter (ADC).

C. Challenges of Reliable NVM-CiM

1) Boolean operation: The correct functionality of Boolean operations implemented using scouting logic can be impacted as NVM devices are prone to process variation which the sensed current values by the sense amplifiers could vary.

As shown in Figure 1(b), when LRS and HRS distributions are close to each other with possible overlap, the output of the scouting logic can be incorrect. This is known as *decision failure*. The probability of the decision failure (P_{DF}) is proportional to the area of the overlap region.

As the number of activated rows increase, i.e., the number of operands, P_{DF} increases as well. Figure 1(b) on the top, shows

the lower P_{DF} of a 2-operand Boolean operation, compared to the 4-operand operation on the bottom. Due to the increase in the number of operands, the current states to be distinguished are getting closer to each other, which leads to a higher P_{DF}.

2) MAC: Similar to the scouting logic, the MAC operation is also susceptible to decision failure. However, due to the ADC requirement and its implementation, the mechanism of decision failure is different from the scouting logic. As shown in Figure 2(b), due to the distribution of the BL current, there is a probability that the digital ADC output deviates from the expected value. Figure 2(b) also shows that the larger the quantization level of the ADC (Q_{large} instead of Q_{small}), the lower the P_{DF}.

D. Microarchitecture-Level Fault Injection

Typically, designers use either SFI [16] or analytical methods like the Architecturally Correct Execution (ACE) analysis [17] to gain insights into the resilience of programs against transient faults. For transient faults, both methods aim to measure the Architectural Vulnerability Factor (AVF) of hardware structures, but each has its advantages and disadvantages. SFI is very accurate, but slow since it requires multiple runs to reach high confidence, whereas ACE analysis is fast, but has a high development effort and may overestimate AVF. SFI is a reliability estimation technique that can provide full-system AVF estimation by directly accessing the program output produced with the injected faults. SFI is a widely adopted method for reliability assessment. It offers flexibility in terms of accuracy, depending on the size of the statistical sample, while providing failure samples generated through simulation.

E. Related Work

To gain a better understanding of what *Sisyphus* offers, in the following, we review the existing CiM simulators. CIM-SIM [8] is a functional behavior simulator for CiM architectures, written in SystemC. Although it is technology-agnostic, it is limited to its proposed 'Nano-Architecture'. MNSIM [9] focuses on simulating memristor-based neuromorphic acceleration. It operates at the memory-array level and aims to accelerate SPICE-level simulations. NeuroSim [10] targets neural network acceleration tasks using CiM based on 40-nm ReRAM technology. However, it lacks configurability, and evaluations are restricted to the provided design. PIMulator [12] uses an FPGA to emulate process in memory (PIM) architectures by integrating PIM-aware modules between the memory controller and the main memory. MultiPIM [11] is a configurable PIM simulator that can be integrated with full-system simulators, allowing for PIM timing simulations using trace files. Sim2PIM [13] is a high-level, PIM-agnostic simulator that enables fast PIM design profiling. PIMSim [14] is a highly reconfigurable full-system PIM simulator that relies on gem5's functionality, offering trade-offs between simulation accuracy and speed. None of the listed simulators offers reliability assessment, and *Sisyphus* tries to fill this gap.

III. CiM System Architecture

The majority of literature on NVM-based CiM architectures [18], [19], [20], [21] focuses on very specific tasks, such as accelerating neural networks or facilitating basic Boolean operations. However, this specialization limits the flexibility of diverse application implementations. Further, with these custom architectures, it is difficult to effectively model and analyze circuit-level factors (such as power consumption, latency, and error rates) in high-level simulations. To close this gap, we develop a general-purpose NVM-CiM architecture to address these concerns and assess the reliability impact across different applications. We discuss the architecture of *Sisyphus* and the proposed memory organization consisting of both storage and computational memories. Additionally, we discuss how the proposed architecture can support the execution of various CiM instructions.

A. Memory Organization

Figure 3 presents a high-level representation of a conventional full-system computer connected to the main memory module, highlighting our modifications. We assume that the main memory module is a well-established Load-Reduced Dual Inline Memory Module (LRDIMM), which enables intermediate buffering and separates the direct signal connection between the CPU and memory chips via its controller. To enable CiM, we extend this controller to filter out CiM commands or data, similar to regular data sent from the CPU to the main memory, but distinguishable by their addresses, and forward them to the CiM controller.

The CiM controller, consisting of a basic state machine, fetches and directs the hardwired signals embedded within the CiM command to dedicated components in the CiM-capable chip (highlighted in Figure 3), while preserving the timing order. The CiM-capable chip is similar to a conventional memory chip, but with modifications: the row decoders can activate multiple rows simultaneously, the sense amplifiers are enhanced to support subarray-level analog operations, and additional CMOS-based CiM logic circuitry is added to the output of the sense amplifier buffers. Figure 4 illustrates the internal structure of the CiM-capable chip along with these modifications. Additionally, we have integrated a direct memory access (DMA) module into the CiM controller to facilitate data movement between the memory chips and the CiM-capable chip, eliminating the need for CPU engagement. The motivation behind this decision is explained in Section III-C.

B. CiM Operations

Data-intensive workloads are the most suitable applications for taking advantage of CiM capabilities. These tasks can be parallelized using simple Single Instruction Multiple Data (SIMD) operations such as AND, OR, XOR, MAC, etc. Boolean CiM operations are implemented with the scouting concept using a comparator that is already adjusted based on the CiM operation. The MAC operation utilizes a specialized ADC as described in Section II-B2. However, relying solely on these operations is not enough to effectively harness CiM

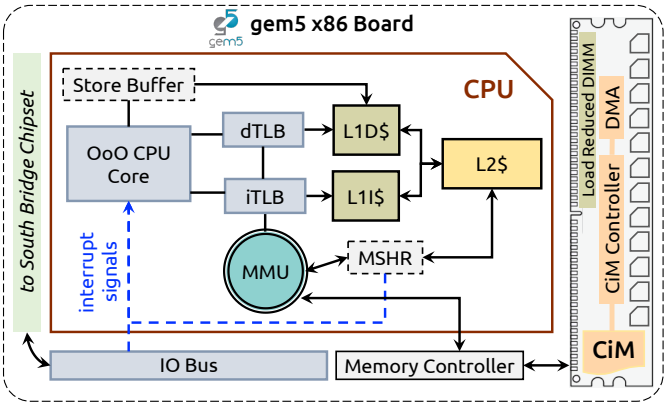

Fig. 3. High-level representation of a conventional full-system computer. The LRDIMM-based main memory controller is enhanced with a CiM controller and internal DMA, enabling efficient bulk data movement between CiM-capable logic and memory chips.

capabilities. The continuous data exchange between the CPU and CiM uses up more time and energy than calculating these operations internally within the CPU.

Besides, in *Sisyphus*, activation of two, four, or eight rows is supported for the aforementioned CiM-compatible operations. Although in some NVM technologies, it is possible to activate many rows, for most applications, this is satisfactory. In addition, to properly execute the primary portion of these workloads in the CiM, it is crucial to have a few specific operations beyond the basic operations described above. These operations include a COPY operation along with data rotation, a bitwise NOT operation, and a byte-wise comparison by `0x00`. Byte-wise comparison by `0x00` involves performing an OR operation on all the 8 bits, which will result in either `0xff` or `0x00` in the corresponding output byte. The last operation is necessary to enable the implementation of basic `if` statements. Furthermore, to demonstrate the extensibility of the CiM framework, the binary MAC operation has been added to enable Vector/Matrix to Matrix Multiplication, a dominant operation in neural network applications.

The format of CiM instructions contains information regarding the operation type and selected row indexes. It also includes various bit fields within the instruction structure, allowing the programmer to have fine-grained control over the operations. Specifically, the programmer can set these bits to specify on which banks, sub-arrays, or the corresponding bytes in the sub-arrays a particular operation needs to be performed.

```
//row2 <- ~(row1 & row3)
cimModule.AND({ 1, 3}); // row1 & row3
cimModule.NOT_COND(2, true, false); // bitwise NOT
```

Listing 1. Bitwise NAND operation on two vectors in our CiM API.

C. Interaction with CiM Module from the Application-layer

The CiM controller can be either active or inactive. When it is inactive, the LRDIMM controller can perform read and write operations in the CiM region in the same way as any other memory region. However, when the CiM controller is active, the LRDIMM controller must wait until the CiM controller completes its tasks before accessing the corresponding regions. To enable communication between the application layer and the CiM controller, we use a fixed physical shadow address which is an address with no specific physical location mapping to it. Instructions from the application layer are written to this address, and when the LRDIMM controller detects them, it forwards the content directly to the CiM controller. It is important to note that the memory mapping is non-cacheable and non-shareable.

Non-cacheability is essential because the CPU process writes data directly into the main memory. Non-shareability is also crucial to ensure data consistency and prevent a process from acquiring this hardware resource while another process has not released it. To enforce these requirements, *Sisyphus* provides an operating system driver and an API. Listing 1 shows a simple code snippet of this API for a bitwise NAND operation on two vectors, while Figure 4 illustrates the actual operations within the CiM units.

In the code snippet shown in Listing 1, we assume that some content is already stored in row #1 and row #3 within the CiM units. The AND operation is then executed simultaneously inside the subarrays by activating two lines (row #1 and #3). Subsequently, the NOT operation is performed on the intermediate results using the CiM logic, and finally, the outcomes are written back to row #2. To demonstrate how to use the `if` statement in the CiM API, assume the ternary operator in C++, as shown in Listing 2. To convert this sequential C++ code into CiM instructions, we assume that each vector has a maximum size that equals the CiM row size. For our simulations, we also consider the presence of 16 banks in CiM-capable chip, where each bank comprises 64 sub-arrays with a column size of 8 bytes. Therefore, in this example, we set the `MAX_ROW_SIZE` to $16 \times 64 \times 8 = 8$ KiB. In case the vectors are larger than this value, they can be divided into aligned sizes.

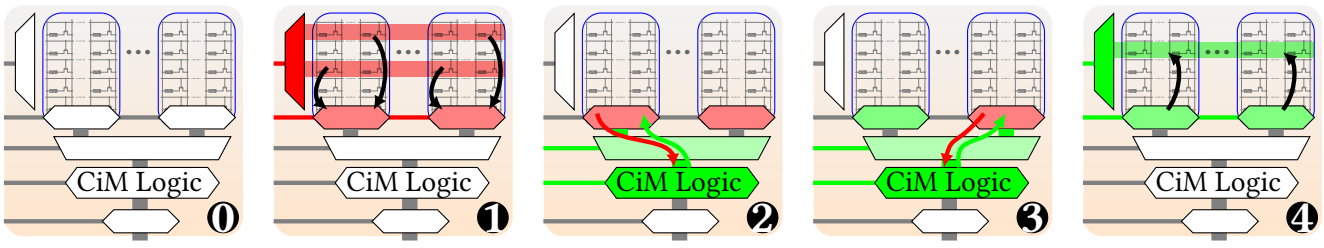

Fig. 4. Execution steps of a bitwise NAND operation in CiM units.

```
for(size_t i=0; i< MAX_ROW_SIZE; i++)
   a[i] = (b[i]==0x12) ? c[i] : d[i];
```
Listing 2. An example of a ternary operator in C++ language.

```
// row0 <- b[...]
cimModule.copy_to_cim(0, (void *)&b[0]);
cimModule.copy_to_cim(1, (void *)&CONSTx12[0]);
cimModule.copy_to_cim(2, (void *)&c[0]);
cimModule.copy_to_cim(3, (void *)&d[0]);
cimModule.copy_to_cim(4, (void *)&a[0]);

// for checking (b[i]==0x12)
cimModule.XOR({0,1})
// row5 <- 0xff if result is 0, otherwise 0x00
cimModule.NOT_COND(5, false, true);
// row6 <- ~row5
cimModule.NOT_COND(6, 5, true, false);
// row7 <- row2 & row5
cimModule.AND({2,5})
cimModule.COPY(7)
// row8 <- row3 & row6
cimModule.AND({3,6})
cimModule.COPY(8)
// a <- row7 | row8
cimModule.OR({7,8})
cimModule.COPY(4)
```
Listing 3. Converting sequential ternary operators to parallel CiM instructions.

The example code in Listing 3 demonstrates the use of the CiM API. In this approach, the programmer copies each vector to a dedicated row which allows operations on each byte to be executed in parallel, eliminating the need for a `for` loop. If our application has other types of loops, they will be unrolled during the CiM code generation, since our CiM Controller does not include loop control instructions. Similarly to using SIMD instructions in assembly language, it is the programmer's responsibility to convert sequential code into a parallel version in the CiM context as well.

CiM applications operate independently of CPU interaction during CiM operations and have a fixed number of instructions. Therefore, issuing CiM instructions individually from the application level to load files into the cache and offloading them to the CiM region is inefficient and not in line with the CiM concept. A more efficient approach is to store all the CiM instructions in the main memory, trigger the CiM controller to read and execute these instructions, and perform intra-main memory data loading if required. To achieve this, internal DMA must be integrated, and the internal memory bus must be shared between the DMA and the LRDIMM controller.

Integrating DMA into the CiM controller imposes significant challenges in the hardware, operating system, and application layers including the issue of data consistency in the case of cache involvement and accessing pages with a risk of segmentation fault. These issues, however, can be resolved at the application layer by assuming that the programmer has control of the system and is aware of these challenges.

IV. Experimental Setup

We present the modification of the cycle-level gem5 simulator with proper extensions for CiM, SFI, and power models.

A. CiM Extensions for gem5

To integrate CiM into the gem5 framework, we make minor modifications to the memory controller and memory interface libraries while preserving the integrity of the remaining gem5 source code. CiM controller functionalities have been added to the memory controller library, and technology-specific parameters are now part of the memory interfaces. Furthermore, to conduct a comprehensive simulation infrastructure, we employ a cross-layer technology-to-application flow in our framework. Starting at the technology level, we utilize realistic, measurement-validated Verilog-A models for the NVM devices. In both the scouting logic and MAC operations, the output is generated in an analog fashion, leveraging the resistive nature of the NVM devices. Therefore, the resistance distribution in LRS and HRS per each technology is relevant information which abstracted from the technology level and passed to the circuit level.

Circuit-level simulations are performed using SPICE tools and an industry-aligned process design kit (PDK). The circuit-level analysis outputs are (a) latency and energy of the CiM modules via single-run and (b) error probabilities via Monte Carlo simulation. These parameters will be abstracted and passed to the memory array simulation and microarchitecture-level fault injection, respectively. Besides, for specific requirements, such as the encoder for the flash-type ADC, latency and energy are derived by synthesizing the behavioral Verilog code using industrially aligned standard cell libraries.

At the memory array level, the NVSim tool is used to determine latency and energy, with updates according to the energy and latency of CiM-related modules, including the address decoder and sense amplifier, based on circuit-level analysis results. The behavior of memory sub-arrays and CiM operations is modeled at the system level within gem5. At the microarchitecture level, what changes between the different NVM technologies are the timing, the bit error rate (BER), and the power/energy consumption of each operation. Therefore, gem5 and McPAT interfaces are augmented with accurate results from lower-level simulations (updated CiM-aware version of NVSim), ensuring a detailed cross-layer analysis and abstraction for accuracy and technology awareness. Furthermore, a CiM API is provided at the application level to access CiM functionality. While 'address memory mapping' relates to the OS level, gem5 offers APIs to enable it through gem5 configuration files.

B. Microarchitecture-Level SFI

In this subsection, we describe our microarchitecture-level evaluation approach and explore the challenges of applying SFI to Compute-in-Memory systems in the next subsection. SFI should ideally be based on a real system with physical injection capabilities in all microarchitectural structures. However, such a system does not currently exist. Even if it did, making protection decisions after studying and implementing it would be too late. Alternatively, a very detailed low-level simulator (e.g., register transfer level (RTL), gate) that allows the same can be used. While low-level simulators may

provide accurate fault effects, their simulation throughput is extremely low and unaffordable. Additionally, they cannot model long-running workloads with OS activity. To address these challenges, *Sisyphus* relies on microarchitecture-level simulation using the gem5 simulator. This approach enables (i) deterministic, (ii) end-to-end, (iii) cycle-level execution of (iv) real workloads. This combination is impossible at lower levels. When it comes to CPU reliability evaluation, we follow the methodology as it is described in [16] and [22].

Briefly, all fault injection campaigns in *Sisyphus* (either for CPU or CiM) consist of injecting a set of faults (statistical samples) into simulations. For statistically significance, we inject 2,000 faults for every injection campaign. We follow the widely adopted formulation of [16] for the statistical fault sampling calculations; our 2,000 fault samples correspond to a 2.88% error margin with a 99% confidence level.

These simulations produce output results, which are parsed to estimate the vulnerability or other desired metrics. Depending on the component being studied, an appropriate statistical sample is chosen. A Python module serves the role of campaign controller and manages the campaigns. Each fault injection simulation requires a fault description file that is produced by the controller. This file defines the location and type of fault to be injected (for the calculation of AVF, single-bit-flip faults are chosen). The simulations are run, with the controller supplying necessary inputs and storing the results. *Sisyphus* can configure multiple workers to achieve 100% hardware resource utilization. Each instance corresponds to one injected fault and produces output files and logs. Finally, simulation outputs are stored for any further investigation.

C. SFI on the CiM Module

The challenge in implementing SFI on the CiM module stems from the different underlying fault model that causes the majority of CiM errors. While in CPUs a major fault type is transient Single-Bit-Upsets (SBUs) caused by alpha-particle and neutron radiation. The main source of error in CiM is internal and originates from the decision failure due to the impact of the process variation mainly affecting the different CiM operations (e.g., AND, OR, XOR, MAC). The BER for each operation not only depends on the number of operands of the operation, but is also data-dependent.

To find the data-dependent probability of the decision failure, i.e., the BER, we perform a Monte Carlo simulation on the NVM sub-array while considering the effect of process variation on the NVM devices. By obtaining the distribution of the output current, the probability of decision failure can be obtained from the overlap region. As shown in Figure 1 (b), the offset of the sense amplifier (S_{offset}) also affects the probability of decision failure and is considered in our setup. However, the traditional SFI approach used for SRAM arrays in CPUs considers that the errors are injected following a uniform distribution across all the bits of the array. As discussed, this scheme is inaccurate for CiM error modeling.

To solve this issue, we inject single-bit-flips weighed by the normalized average BER of each operation type (here operation type refers to the actual operation and the number of rows/operands being activated/processed). Assuming we inject N total faults, the number of faults injected to operation type OP_i is given by $Faults_i = \frac{BER_i}{\sum_j BER_j} \times N$, where BER_i is the average bit-error-rate of OP_i across the entire execution of the workload being evaluated. To compute these average BERs, a single fault-free execution is done using a custom mode of our gem5-based SFI infrastructure. Models that capture the count and data dependency of CiM operation's BERs have been built using data from the technology and circuit layers and are employed during this golden run. By weighing the number of faults injected into each operation by the normalized BER, we guarantee that our fault sample is representative of the workload-memory technology combination being evaluated.

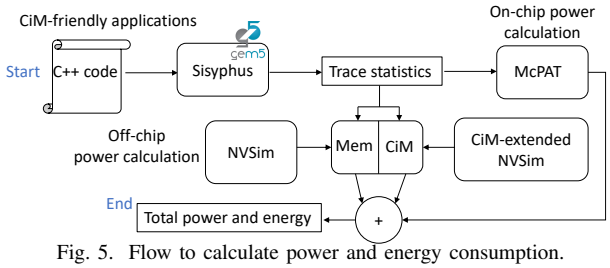

Fig. 5. Flow to calculate power and energy consumption.

D. Full-System Power/Energy Modeling

To model the entire system power, we split it into *on-chip* and *off-chip* parts, as shown in Figure 5. The on-chip power is consumed by the CPU and cache, while the off-chip power is consumed by the (storage) memory and the CiM module. McPAT [6], a widely used tool, computes the on-chip average power using statistics traces from gem5 for each executed workload. To compute the off-chip power of the memory, we use NVSim [7]. However, NVSim is designed only for storage memory (read and write accesses) and needs to be extended for the CiM module. There are two main differences between NVSim and our CiM extensions. First, in the CiM-extended version, the row decoder module activates more than one row at a time, leading to scaled power and energy consumption. Second, the power and latency of the sense amplifier module need updating based on the number of activated rows and the NVM technology. By accurately modeling these components at the circuit level (using SPICE simulation), the corresponding models of NVSim are updated for the entire CiM module. Similarly, for the off-chip power, the statistics trace is required to calculate the average power of the storage memory and CiM based on their respective number and type of accesses. These accesses are multiplied by the per-access power obtained through NVSim and its CiM extension. The total power of the application running on *Sisyphus* full system is the sum of the on-chip and off-chip power contributions.

E. Applications

We implement 9 prevalent CiM-friendly applications: (1) **BitIndexing (BIT)** [23]: bitmap-formatted database queries are being ORed with each other, and the results are being

ANDed with another query; (2) **AES-ECB encryption algorithm (AES)** [24]: there are 4 main computation phases: AddRoundKey, SubBytes, ShiftRows, and MixColumns; (3) **BLASTN algorithm (BLS)** [25]: searching short DNA sequences inside large DNA protein sequences; (4) **Morphological Image Processing (MIP)** [26]: we implemented Dilation and Erosion algorithms with binary-coded input images with various filter sizes; (5) **Marching Squares (MSQ)** [27]: an algorithm used for extracting contour lines from 2D images; (6) **Shifted Hamming Distance (SHD)** [28]: a famous algorithm used for calculating edit distances in short DNA sequences; (7) **BitWeaving (BWV)** [29]: a technique presented to scan database queries in memory; (8) **Matrix Multiplication (MAC)**: utilizes the MAC operation in our CiM framework. The size of the stationary matrix was kept fixed at 8×65536, and we use different matrix sizes for the non-stationary inputs; (9) **NN Inference (MNIST)**: utilizes a 4-layer neural network with binarized weights and inputs (values of $-1, 1$), employing the $\mathrm{sign}(x)$ activation function to perform inference on the MNIST test dataset.

For every implemented application, we verify the CiM-enabled results by comparing them with the results from the CPU-only execution. This comparison confirms that the CiM-enabled code functions correctly and ensures that any errors observed during our fault injection campaigns for reliability assessment are only due to fault injection.

F. Simulation Setup

Table II lists our main configuration for this simulation setup. All our experiments (for both CPU and CiM) are performed assuming unprotected designs for two reasons: (1) for fair comparisons between CPU and CiM, and (2) to evaluate the raw performance and vulnerabilities of each architecture under identical conditions. This approach ensures that our analysis remains unbiased and focuses on the core attributes of CPU and CiM without the impact of additional protective measures. However, any protection scheme can be added to Sisyphus since it is based on the well-established gem5 simulator and can be easily evaluated.

TABLE II
MAIN PARAMETERS OF THE SIMULATION SETUP.

NVM models	STT-MRAM [30] ReRAM (JART VCM Read variability) PCM [31], [32]
Subarray organization	256 Rows, 64 columns
CPU type	OoO
Compiler & Optimization Flag	g++ -O3
ISA / Clock Frequency	x86–64 / 1 GHz
# of Load/Store/Issue/ROB Entries	32 / 32 / 64 / 128
# of Physical Int/FP/Vec Registers	128 / 128 / 128
L1 Inst/Data Cache Size	32 KiB
L1 Tag/Data/Response Latency	2/2/2 cycles
L2 Cache Size	256 KiB
L2 Tag/Data/Response Latency	20/20/20 cycles
System Bus Latency	10 ns
Average Memory Controller Latency	15 ns
Read Latency (STT/ReRAM/PCM)	1 cycle / 1 cycle / 1 cycle
Write Latency (STT/ReRAM/PCM)	4 cycles / 45 cycles / 40 cycles

V. SYSTEM-LEVEL EVALUATION

A. Performance and Energy Efficiency

To measure the energy efficiency of CiM, we tried to simulate multiple scenarios for various NVM technologies and report total execution time, power, and energy consumption as the evaluation metrics. Our simulation scenarios include running the application on OoO CPU (i) without any assistance from CiM, (ii) with assistance from CiM, while the commands are issued from the application layer, and (iii) with CiM assistance while storing the commands in the main memory and utilizing the DMA of CiM for data movement.

Figure 6 shows the total execution time for each application across various scenarios and input sizes. In scenario (2), most applications, except for a few, prefer memory-bound operations over computation, resulting in less pronounced performance gains from CiM compared to scenario (1). In both scenarios, the large read/write latency between the CPU and main memory, significantly higher than the read/write latency of memory technologies, results in total execution times on the order of microseconds. Thus, we only reported results for STT technology. Additionally, the higher latency in scenario (2) arises from the CPU handling the copying of memory rows to and from the CiM region, which incurs higher copy latency compared to scenario (1), where the CPU leverages data locality and uses its caches more efficiently.

In scenario (3), by eliminating unnecessary data movement, we achieved performance improvements of several orders of magnitude for most applications compared to scenario (1). Additionally, the smaller write latency of STT technology, compared to ReRAM/PCM, contributes to superior performance. Furthermore, *STT-MRAM: 2-op(erand) represents STT technology in which the maximum number of activated rows is limited to 2.* On average, scenario (3) offers over a 7× latency improvement compared to scenario (1). It's noteworthy that in this scenario, we excluded the AES encryption algorithm due to ongoing interaction between CiM and the CPU.

The differing behavior between scenarios (2) and (3) in the MAC application (Figure 6) is because there are few write operations occurring. Typically, write operations are performed during the initialization stage for stationary matrices.

Figure 7 illustrates the average dynamic power of various components (on-chip power, including CPU and caches, memory accesses from the CPU side, and CiM + DMA power consumption), as well as energy consumption for different technologies. The power behavior depicted in this figure primarily depends on the characteristics of each application—some applications are memory-bound, some are computation-bound, and others are a combination of both.

For example, ReRAM and PCM have superior performance in memory-bound applications. Since this represents average dynamic power, during idle CPU modes, we observe reduced power consumption per application, applicable to all simulations in scenario (2). On the other hand, energy consumption is predominantly influenced by application latency, increasing with higher total execution times. On average, scenario (3)

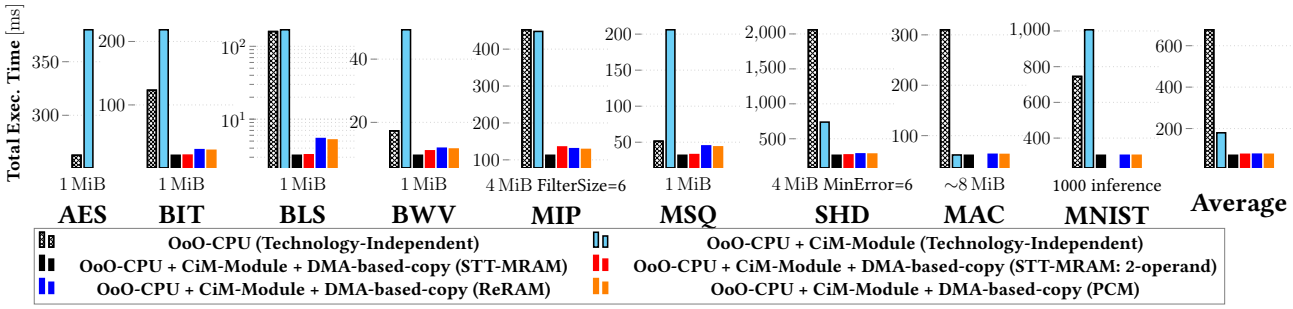

Fig. 6. Measuring the total execution time of various applications for a specified input size.

exhibits nearly a 19% and a 13× improvement in power and in energy consumption, respectively, compared to scenario (1).

B. Failures in Time (FIT)

FIT rate of a device is the number of failures that can be expected in one billion (10^9) device hours of operation. For each hardware structure in a heterogeneous system including CPU microarchitectural components and CiM accelerators, a different FIT is computed using the formula: $FIT_{struct} = AVF_{struct} \times rawFIT_{bit} \times \#Bits_{struct}$. The FIT of the structure depends on three parameters: (1) the FIT_{bit} (or $rawFIT_{bit}$) rate, which is determined by the fabrication technology, the operational conditions (and the workload being executed in the case of CiM) which expresses the fault probability of a single bit, (2) the number of bits of the structure, and (3) the AVF of the structure, which is affected by the microarchitecture, the architecture (ISA) and the executed workload. The raw FIT rate expresses the number of faults that will be introduced in the component, while the AVF is the derating factor that quantifies how many of these upsets will lead to failure. The product equals the FIT rate of a particular hardware structure. For the calculation of the FIT for the CPU, we use the raw FIT rate per bit, according to [33], which is 3.85×10^{-6} but, of course, any technology-related value can be used.

While the raw FIT rate (due to the underlying fault model being random alpha-particle and neutron hits) is workload-independent for CPU components, the raw FIT rate of the CiM module highly depends on the workload being executed. Different workloads activate different operations with a different number of operands that contain different data. It is thus imperative to calculate this raw FIT rate for each memory technology workload pair. The output of this experiment is the average BER of all the bits produced by the CiM module operations, i.e., AND, OR, XOR, and MAC. The raw FIT rate is finally calculated using the formula: $rawFIT = BER \cdot \frac{10^9\ [\text{h}]}{ExecTime_{workload}\ [\text{h}]}$.

As a first quantitative comparison, we show the FIT rates for all NVM technologies (ReRAM, PCM, STT) and the CPU configuration used in this study, as shown in Figure 8. The most important observation is that the CPU FIT rates, for all applications, exhibit the lowest values compared to all NVM technologies. Moreover, from Figure 8, four major insights can be extracted: (a) due to the sufficiently distinct resistive states measures as high HRS-LRS ratio, this technology is the most robust technology for CiM compared to PCM which, in turn, is more robust than STT, which is aligned with their relative HRS-LRS ratio: ReRAM > PCM > STT-MRAM. (b) Both PCM and STT provide, in some applications, virtually similar FIT (e.g., AES, BWV, MAC, and MNIST). However, in the majority of applications, their FIT difference is quite high (e.g., BIT, BLS, MIP, MSQ, SHD). This observation does not hold for the ReRAM when it is compared to the other two NVM technologies. The workload variability is extremely high. Specifically, there are significant differences among different NVM technologies regarding their sensitivity to faults. For example, comparing MIP and MSQ, although the latter is more robust than the former (i.e., the FIT rates of the latter are lower than of the former's) for ReRAM and STT technologies, the same applications in PCM show

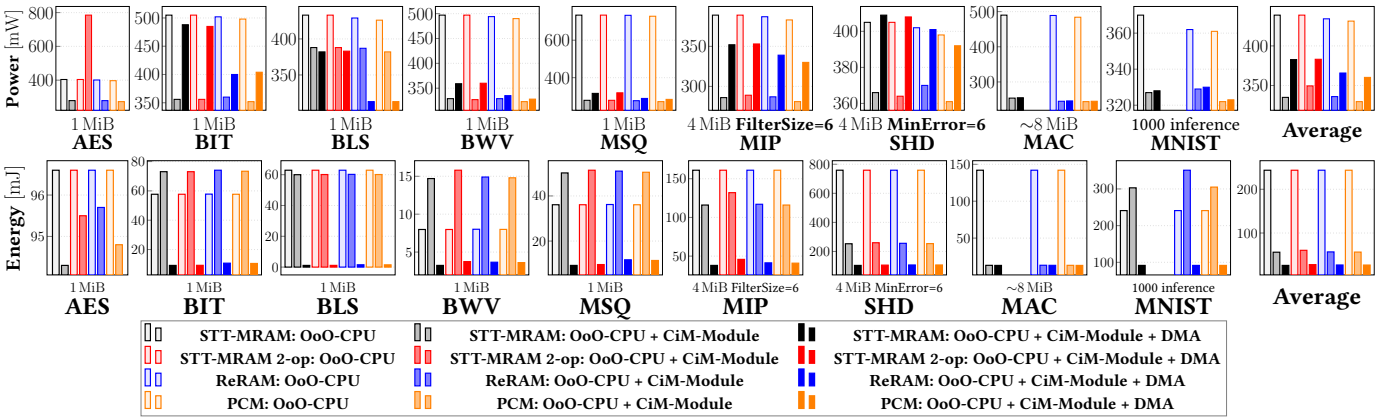

Fig. 7. Average dynamic power and energy consumption of various applications for a specified input size.

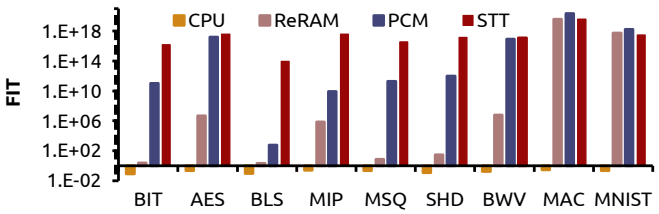

Fig. 8. Failures in Time (FIT) rates for all three NVM technologies (ReRAM, PCM, STT) and for the CPU configuration used in this study.

the opposite trend, i.e., the MSQ is less robust than MIP. Finally, (d) the matrix multiplication workload that uses the MAC operation has the worst FIT out of all the applications considered. The challenges of implementing a reliable MAC operation are discussed in Section II-C2. Interestingly, though, the MNIST workload, which also uses the MAC operation, has a comparatively lower FIT, possibly due to the inherent approximate and fault-tolerant nature of NN inference.

These observations suggest that, based exclusively on the FIT rates (i.e., the reliability-only aspect of devices, ignoring the performance and energy consumption), due to the relatively small HRS-LRS ratio, STT-MRAM technology is the most unreliable technology compared to all three NVM technologies considered in this study. However, each NVM technology and CPU provides different performance and energy characteristics for the same application. Therefore, it would be misleading if we just compare the FIT rates of each NVM technology.

C. Combined Energy and Performance Considerations

Energy consumption itself is a valuable metric that directly translates to cost, however, it occasionally implies very slow system configurations (very low frequencies). Such slow configurations could violate latency and throughput requirements, especially in heterogeneous environments accompanied by general-purpose CPU and CiM (or other domain-specific accelerators). To avoid this bias in the comparisons of different configurations, other metrics such as the energy-delay product ($EDP = E \times D$) and the energy-delay squared product ($ED2P = E \times D^2$) have been proposed for high-end systems. Given that this study focuses on high-end heterogeneous systems with CPUs and CiM configurations, we have chosen to present the energy-delay squared product ($ED2P$). This allows us to show a fair comparison for all of our experiments among benchmarks, CPU, and CiM system configurations. More importantly, the $ED2P$ metric provides a fair representation of the relation between energy and performance for different heterogeneous systems, such as general-purpose CPUs and CiM equipped with different NVM technologies.

Figure 9 illustrates the $ED2P$ for all seven workloads across three NVM technologies and the CPU-only execution model. To enhance data visualization, we present $1/ED2P$. A larger displayed value indicates that the computing device handles the energy-performance trade-off more effectively. It is evident that, in most cases, CiM prevails. On average, the execution of workloads with CiM enabled is 14.6× more efficient than CPU-only execution, according to the $ED2P$ metric. For three workloads (AES, MSQ, and BWV), the CPU takes the lead but with a relatively small margin. We attribute this to these applications being memory-bound, meaning that memory transfers between the CPU and CiM act as a bottleneck, preventing CiM from performing in its most efficient mode.

D. Putting It All Together: Performance, Energy, Reliability

To quantify the triple trade-off between performance, energy, and reliability, we employ the following metric: $ERC = \frac{1}{ED2P \cdot FIT}$. ERC stands for Efficiency-Resilience Coefficient and aggregates the three evaluation axes considered in this paper. A doubling in either energy consumption or FIT results in a halving of ERC whereas a doubling in delay results in a quartering of ERC. The greater emphasis on delay follows the rationale discussed in Section V-C. Figure 10 presents the ERC results for all considered workloads across three NVM technologies and the CPU-only case[1]. A higher result indicates better management of the three-parameter trade-off for the specific workload-technology pair. From Figure 10, it is evident that the CPU-only execution model holds a distinct advantage across the entire workload spectrum, with the sole exception being the BLS application, where ReRAM takes the lead. Among the three NVM technology models, ReRAM emerges as the front-runner, only surpassed by PCM in the MIP application and STT in the MNIST application. STT generally lags behind the other two, primarily due to its significantly inferior resilience.

An interesting observation pertains to the workload sensitivity of the ERC metric. For CPU evaluation, the metric remains within two orders of magnitude, whereas for CiM, it spans 20 orders of magnitude. This substantial variation is primarily due to the non-uniform resilience of CiM operations, which behave differently depending on the number of operands and the input data. Since different applications utilize CiM operations to varying extents, it is natural for ERC to fluctuate significantly.

In the reliability analysis of Sisyphus we take into account the alpha-particle and neutron radiation as the main sources of the error in CMOS-based components and the impact of the process variation for the NVM devices. However, it can be extended to consider other sources of errors in the CPU or CMOS-based memories as well as other failure mechanisms in different flavors of NVM-CiM.

VI. Conclusion

Sisyphus is a novel, cross-layer modeling framework that evaluates system architectures combining CPUs and CiM ac-

[1]The displayed numbers have been scaled by a uniform constant to ensure all results remain above the x-axis; the relative differences are preserved.

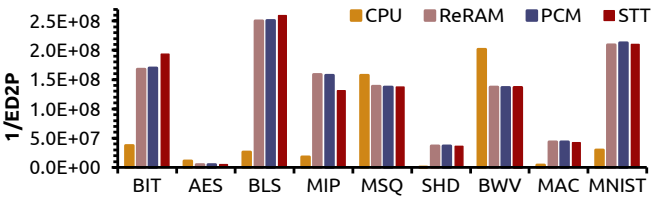

Fig. 9. Inverted Energy Delay Squared Product ($1/ED2P$) for NVM technologies (ReRAM, PCM, STT) and the CPU configuration used in this study.

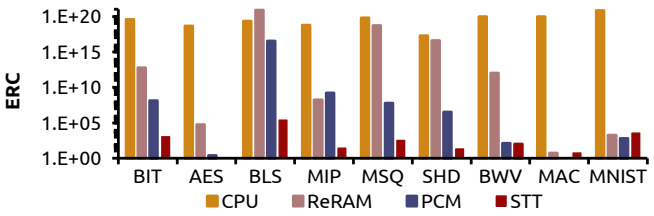

Fig. 10. Efficiency-Resilience Coefficient (ERC) for all three NVM technologies (ReRAM, PCM, STT) and for the CPU configuration used in this study.

celerators of NVM technologies. It assesses the performance, power, and resilience interactions by utilizing microarchitectural modeling on gem5, employing accurate models based on data from technology and circuit layers. We revealed trends and differences among PCM, ReRAM, and STT-MRAM in execution time, power consumption, and resilience.

ACKNOWLEDGMENTS

University of Athens authors are supported by the EU Horizon Europe research and innovation programme under grant No 101070238 (NEUROPULS), the Chips JU grant No 101097224 (REBECCA), and the EuroHPC JU grant No 101202459 (DARE SGA1). At the time of publication, the fourth author is with U Patras. Views and opinions expressed are however, those of the authors only and do not necessarily reflect those of the EU. Neither the EU nor the granting authority can be held responsible for them.

REFERENCES

[1] D. Pandiyan and C.-J. Wu, "Quantifying the energy cost of data movement for emerging smart phone workloads on mobile platforms," in *2014 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2014, pp. 171–180.

[2] W. Dally, "Challenges for future computing systems," *HiPEAC keynote*, pp. 1–66, 2015.

[3] Y.-D. Chih *et al.*, "13.3 a 22nm 32mb embedded stt-mram with 10ns read speed, 1m cycle write endurance, 10 years retention at 150°c and high immunity to magnetic field interference," in *2020 IEEE International Solid-State Circuits Conference - (ISSCC)*, 2020, pp. 222–224.

[4] W. C. Chien *et al.*, "Device study on ots-pcm for persistent memory application : Ibm/macronix phase change memory joint project," in *2022 6th IEEE Electron Devices Technology & Manufacturing Conference (EDTM)*, 2022, pp. 327–329.

[5] J. Lowe-Power *et al.*, "The gem5 simulator: Version 20.0+," 2020. [Online]. Available: https://arxiv.org/abs/2007.03152

[6] S. Li *et al.*, "Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2009, pp. 469–480.

[7] X. Dong *et al.*, "Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 7, pp. 994–1007, 2012.

[8] A. BanaGozar *et al.*, "Cim-sim: Computation in memory simuiator," in *Proceedings of the 22nd International Workshop on Software and Compilers for Embedded Systems*, ser. SCOPES '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 1–4. [Online]. Available: https://doi.org/10.1145/3323439.3323989

[9] L. Xia *et al.*, "Mnsim: Simulation platform for memristor-based neuromorphic computing system," in *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2016, pp. 469–474.

[10] P.-Y. Chen, X. Peng, and S. Yu, "Neurosim: A circuit-level macro model for benchmarking neuro-inspired architectures in online learning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 12, pp. 3067–3080, 2018.

[11] C. Yu, S. Liu, and S. Khan, "Multipim: A detailed and configurable multi-stack processing-in-memory simulator," *IEEE Computer Architecture Letters*, vol. 20, no. 1, pp. 54–57, 2021.

[12] S. Mosanu *et al.*, "Pimulator: a fast and flexible processing-in-memory emulation platform," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2022, pp. 1473–1478.

[13] P. C. Santos, B. E. Forlin, and L. Carro, "Sim2pim: A fast method for simulating host independent & pim agnostic designs," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2021, pp. 226–231.

[14] S. Xu *et al.*, "Pimsim: A flexible and detailed processing-in-memory simulator," *IEEE Computer Architecture Letters*, vol. 18, no. 1, pp. 6–9, 2019.

[15] L. Xie *et al.*, "Scouting logic: A novel memristor-based logic design for resistive computing," in *2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2017, pp. 176–181.

[16] R. Leveugle *et al.*, "Statistical fault injection: Quantified error and confidence," in *2009 Design, Automation & Test in Europe Conference & Exhibition*, 2009, pp. 502–506.

[17] S. Mukherjee *et al.*, "A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor," in *Proceedings. 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36.*, 2003, pp. 29–40.

[18] Z. Wan *et al.*, "Accuracy and resiliency of analog compute-in-memory inference engines," *J. Emerg. Technol. Comput. Syst.*, vol. 18, no. 2, mar 2022. [Online]. Available: https://doi.org/10.1145/3502721

[19] B. Yan *et al.*, "On designing efficient and reliable nonvolatile memory-based computing-in-memory accelerators," in *2019 IEEE International Electron Devices Meeting (IEDM)*, 2019, pp. 14.5.1–14.5.4.

[20] H. Liu *et al.*, "Afpr-cim: An analog-domain floating-point rram-based compute-in-memory architecture with dynamic range adaptive fp-adc," 2024.

[21] S. Li *et al.*, "Pinatubo: a processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories," in *Proceedings of the 53rd Annual Design Automation Conference*, ser. DAC '16. New York, NY, USA: Association for Computing Machinery, 2016. [Online]. Available: https://doi.org/10.1145/2897937.2898064

[22] G. Papadimitriou and D. Gizopoulos, "Demystifying the System Vulnerability Stack: Transient Fault Effects Across the Layers," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA '21)*, 2021, pp. 902–915. [Online]. Available: https://doi.org/10.1109/ISCA52012.2021.00075

[23] K. Stockinger *et al.*, "Using bitmap index for joint queries on structured and text data," in *New Trends in Data Warehousing and Data Analysis*. Springer, 2008, pp. 1–23.

[24] N. I. O. STANDARDS and T. G. MD, "Announcing the advanced encryption standard (aes)," *U.S. Department of Commerce*, 2001.

[25] S. F. Altschul *et al.*, "Basic local alignment search tool," *Journal of molecular biology*, vol. 215, no. 3, pp. 403–410, 1990.

[26] F. Y. Shih, *Image processing and mathematical morphology: fundamentals and applications*. CRC press, 2017.

[27] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," in *Seminal graphics: pioneering efforts that shaped the field*, 1998, pp. 347–353.

[28] H. Xin *et al.*, "Shifted Hamming distance: a fast and accurate SIMD-friendly filter to accelerate alignment verification in read mapping," *Bioinformatics*, vol. 31, no. 10, pp. 1553–1560, 01 2015.

[29] Y. Li and J. M. Patel, "Bitweaving: Fast scans for main memory data processing," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, 2013, pp. 289–300.

[30] F. Bernard-Granger *et al.*, "SPITT: A magnetic tunnel junction SPICE compact model for STT-MRAM," in *Proceedings of the MOS-AK Workshop of the Design, Automation & Test in Europe (DATE)*, 2015.

[31] M. Le Gallo *et al.*, "Evidence for thermally assisted threshold switching behavior in nanoscale phase-change memory cells," *Journal of Applied Physics*, vol. 119, no. 2, p. 025704, 2016.

[32] M. Le Gallo and A. Sebastian, "An overview of phase-change memory device physics," *Journal of Physics D: Applied Physics*, vol. 53, no. 21, p. 213002, 2020.

[33] G. Papadimitriou and D. Gizopoulos, "Characterizing Soft Error Vulnerability of CPUs Across Compiler Optimizations and Microarchitectures," in *2021 IEEE International Symposium on Workload Characterization (IISWC)*, 2021, pp. 113–124. [Online]. Available: https://doi.org/10.1109/IISWC53511.2021.00021