P-PIM: A Parallel Processing-in-DRAM Framework Enabling RowHammer Protection

Ranyang Zhou†, Sepehr Tabrizchi‡, Mehrdad Morsali†, Arman Roohi‡ and Shaahin Angizi†
†Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ 07102, USA
‡School of Computing, University of Nebraska–Lincoln, Lincoln NE, USA
rz26@njit.edu, aroohi@unl.edu, shaahin.angizi@njit.edu

***Abstract*—In this work, we propose a Parallel Processing-In-DRAM architecture named P-PIM leveraging the high density of DRAM to enable fast and flexible computation. P-PIM enables bulk bit-wise in-DRAM logic between operands in the same bit-line by elevating the analog operation of the memory sub-array based on a novel dual-row activation mechanism. With this, P-PIM can opportunistically perform a complete and inexpensive in-DRAM RowHammer (RH) self-tracking and mitigation technique to protect the memory unit against such a challenging security vulnerability. Our results show that P-PIM achieves ∼72% higher energy efficiency than the fastest charge-sharing-based designs. As for the RH protection, with a worst-case slowdown of ∼0.8%, P-PIM archives up to 71% energy-saving over the SRAM/CAM-based frameworks and about 90% saving over DRAM-based frameworks.**

***Index Terms*—Processing-in-DRAM, rowhammer, memory sub-array**

I. Introduction

The security of the main memory is crucial as it is associated with the regular operation of the computing systems. While the continuous shrinkage in the size of the DRAM bit-cell has increased the chip density to achieve a larger capacity [1], it is experimentally demonstrated that even the most recent DRAM chips suffer from a design flaw [2]. The vulnerability imposed by this flaw is called RowHammer (RH). As initially analyzed by Kim et al. [3] in 2014, such a vulnerability is exploited when an attacker repeatedly taps a specific row in the memory causing the adjacent rows to be inverted, thereby corrupting the data in the memory. This phenomenon is a result of electrical coupling between multiple rows. Thus, as the chip process node scales down, the adjacent cells' distance keeps reducing, resulting in a broader range of RH attacks [4]. A proof of concept shows that greater than two lines can be simply affected [1]. In the worst case, malicious code could escape the sandbox environment or even take over the whole system [5].

According to the experimental data [1], [3], the minimum time that one hammer can flip the row in modern DRAM should be larger than the refresh interval. If the row flips before the refreshing, the program will use incorrect data to perform the operations. In this case, the system will spend more time retrieving the data than protecting it. However, the cost of protecting rows from the RH attack is huge, both in terms of latency and power consumption [1], [5], [6]. The simplest solution to mitigate the RH attack is increasing the refresh rate [7]. However, this approach imposes a huge extra power consumption. Currently, the state-of-the-art solutions can be categorized into two groups: *i)* Probabilistic row activation [3] in which memory control randomly refreshes the high frequently activated rows or adjacent rows (victim rows) and *ii)* Counter-based detection [6]–[8] in which the target row refresh mechanism could be an effective and efficient solution for the RH mitigation. This mechanism scans all DRAM data rows and can determine the rows with abnormally higher activation than others. So, the system can retrieve the target rows instead of the whole DRAM.

We believe such a challenging vulnerability of the main memory can be plugged in by leveraging a new processing-in-DRAM architecture locally without engaging external memory units. Such in-DRAM computing platfroms have been widely used to overcome the von-Neumann architecture's memory wall bottleneck over the past few years [9]–[11] by incorporating logic units within memory to process data internally without high-frequency access and long-distance transmission. This stems from larger capacities and off-chip data transfer for DRAM compared to other memory technologies. The main contributions of this work are as follows. (1) We develop P-PIM as a parallel processing-in-DRAM architecture based on a set of novel microarchitectural and circuit-level schemes; (2) We develop ISA and the parallelism required to compute any user-defined in-DRAM operation; and (3) We show P-PIM's applicability in protecting against the RH vulnerability with an error-tolerant and inexpensive method that reduces area overhead and latency imposed in the state-of-the-art mechanisms.

II. Background

A. DRAM Background

DRAM organization. The DRAM chip is a hierarchical structure consisting of several memory banks. Each bank comprises 2D sub-arrays of memory bit-cells that are virtually ordered in memory matrices (mats), which have billions of DRAM cells on modern chips [1]. As shown in Fig. 1(a), each DRAM bit-cell consists of a capacitor and an access transistor. The principle of storing data in DRAM is to use the amount of stored charge in the capacitor to represent binary "1" or "0" [10].

DRAM refreshing. Due to the leakage current phenomenon in the transistor, the amount of charge stored on the capacitor may not be enough to correctly discriminate the data, resulting in data corruption. To secure the data in DRAM, the system recharges the DRAM periodically.

In-DRAM computing. RowClone [12] presents a very fast in-memory copy operation (<100ns) by issuing two back-to-back ACTIVATE commands (without PRECHARGE com-

This work is supported in part by the National Science Foundation under Grant No. 2228028, 2216772, and 2216773.

978-3-9819263-7-8/DATE23/© 2023 EDAA

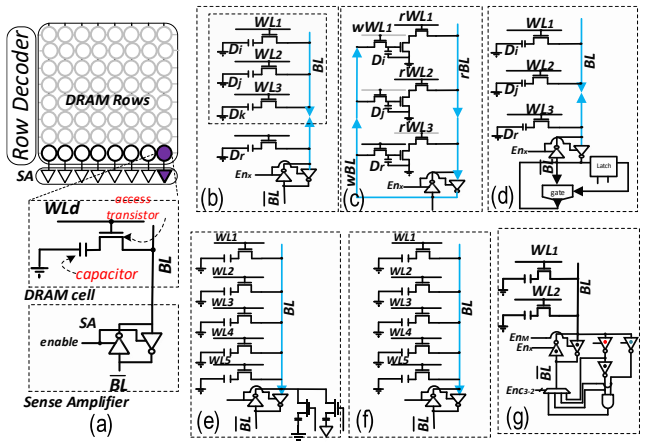

Fig. 1. (a) DRAM sub-array organization, (b) Ambit's TRA [10], (c) DRISA's 3T1C [9], (d) DRISA's 1T1C [9], (e) GraphiDe's QRA [11], (f) QRA [14], (g) ReDRAM's DRA [15].

mand in between) to the source and destination rows. Ambit [10] proposes a framework based on this technique, to realize the bulk bit-wise in-memory logic operations by (i) activating three DRAM rows, (ii) charge sharing among DRAM cells, and then (iii) leveraging the typical DRAM's sense amplifiers to generate output as shown in Fig. 1(b). Inspired by Ambit's Triple Row Activation (TRA) mechanism [10], prior works have proposed in-memory designs with multi-row activation that support more complex logic operations [9], [11], [13]–[17] through charge sharing between input operands stored in capacitors. Such designs support large operands' bit-width up to physical memory row size and high throughput for most logic operations (Fig. 1(c-g)). However, they show low reliability due to multi-row activation. Alternatively, pLUTo [18] and pPIM [19] support simultaneous querying for the LUT-based computation to perform complex arithmetic operations beyond the scope of prior proposals. However, they only support low-bit-width (up to 4-bit [18]) arithmetic operations with little area overhead.

B. RowHammer Mitigation

Both hardware and software level mitigation mechanisms have been actively explored and followed to eliminate RH vulnerability on next-generation DDRx systems. State-of-the-art works [1], [20], [21] proposed alternative measures to solve the problem. The memory controller and system manufacturers also include defenses such as increased refresh rates and hardware RHP [22]. However, the existing RH mitigation proposals have faced a huge overhead both from latency and energy consumption perspectives. In this work, we mainly focus on three objectives: *i) Reducing area overhead.* Most of the counter-based detection methods require add-on hardware to detect rows' activation. In the worst scenario, each row is equipped with a counter, and the best scenario is to use one counter for a group consisting of multiple memory rows [6]. Moreover, the system still needs a fast-read memory such as SRAM or CAM for storing the Activation Count Table (ACT). In the proposed DRAM-only mechanism, no counters and fast-read memory are required. *ii) Reducing latency.* When the system determines the target row under attack, it will retrieve the row. In the counter-based frameworks, the system will refresh the victim rows [20], but the P-PIM architecture intrinsically and locally supports bit-wise operations to refresh the row with a low energy cost. *iii) Error tolerance.* Refreshing the rows can only clear the number of the activation instead of retrieving the data. Therefore, if the target row has been flipped before refreshing, the following operations depending on the data will all write back fault results. In our design, we try to eliminate errors to make sure every data is usable.

III. P-PIM Architecture

Herein, the P-PIM is developed as a parallel charge-sharing-based in-DRAM accelerator on top of the existing DRAM hierarchy to execute bulk bit-wise in-memory operations. As shown in Fig. 2, each memory matrix consists of multiple n-row computational sub-arrays, each divided into two parts, i.e., n-17 Data rows controlled by a typical Row Decoder (RD), and 17 Compute rows dedicated to executing bulk bit-wise logic operations connected to a Modified Row Decoder (MRD). P-PIM's computational sub-arrays are designed to accelerate complex X(N)OR logic operation via a novel sense amplifier design. Supported by the P-PIM's ISA, this will enable the RH protection as explained in Section III.B.

A. Bit-wise Operation Mode with Dual-row Activation

Offering an area-efficient and high-throughput in-DRAM X(N)OR2 operation has been challenging due to the inherent complexity of X(N)OR-based logic implementations as discussed in many recent platforms ([9]–[11], [15]). Nevertheless, these platforms can utilize maximum memory-level parallelism and internal memory bandwidth to implement NOT, (N)OR, (N)AND, and MAJ/MIN functions. The P-PIM architecture not only supports memory and the TRA mechanism [10], [16], but also operates based on a novel Dual-row Activation (DRA) mechanism to implement bit-wise X(N)OR2 operation. A reconfigurable sense amplifier on top of the conventional DRAM sensing circuitry is developed. As depicted in Fig. 3(a), a regular inverter-based DRAM sense amplifier is equipped with add-on circuits, including two inverters, a pair of PMOS and NMOS transistors, and a 2:1 MUX. Four enable signals (C_m, C_{nand}, C_{nor} and C_{mux}) are responsible for controlling the sense amplifier. Leveraging the charge-sharing feature of the

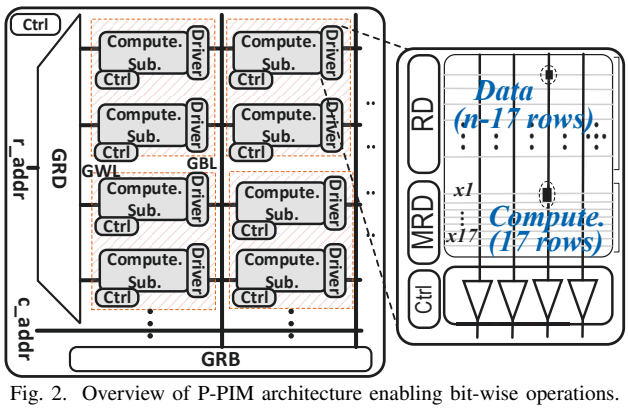

Fig. 2. Overview of P-PIM architecture enabling bit-wise operations.

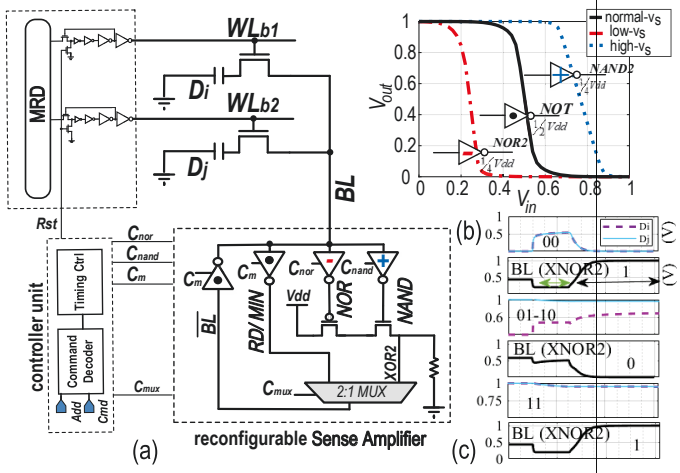

Fig. 3. (a) New sense amplifier design for P-PIM architecture, (b) VTC of the skewed inverters, (c) Transient simulation results.

DRAM cell, the X(N)OR2 logic can be implemented between two selected rows through static capacitive-NAND/NOR in only one cycle. Two different types of skewed inverters with shifted Voltage Transfer Characteristics (VTC) are utilized to perform capacitive-NAND/NOR functions. As shown in Fig. 3(b), a high switching voltage (V_s) inverter with standard high-V_{th} NMOS and low-V_{th} PMOS is responsible to carry out NAND logic. A low V_s inverter with standard low-V_{th} NMOS and high-V_{th} PMOS is responsible for carrying out the NOR logic. Thus, this way, NAND2, and NOR2 logic will be readily accessible. It is worth mentioning that, utilizing low/high-threshold voltage transistors along with normal-threshold transistors has been accomplished in low-power applications, and many circuits have enjoyed this technique in low-power design [15], [23].

To understand the operation of the DRA mechanism, assume that D_i and D_j operands in Fig. 3(a) are copied from data rows to $x1$ and $x2$ rows by RowClone method [12] and both BL and $\overline{BL}$ are precharged to $\frac{V_{dd}}{2}$. Utilizing an MRD which activates two rows at the same time (WL_{b1}, WL_{b2}) in Fig. 3(a), NAND and NOR logic functions can be achieved through charge sharing between BL and activated cells' data at the output of modified inverters. Then, applying these outputs to the additional pair of NMOS-PMOS transistors will readily result in XOR2 logic. A 2:1 MUX is responsible to select between normal DRAM read managed by the dotted inverters and XOR2 operations managed by skewed inverters. Configuration of the controlling bits in these two operations is tabulated in Table I. Fig. 3(c) depicts the transient voltage simulation results of the P-PIM considering all possible data combinations and how the single-cycle XNOR function is generated on the BL.

TABLE I
CONTROL BITS STATUS IN SENSE AMPLIFICATION STATE.

Ops.	Activation	C_m	C_{nor}	C_{nand}	C_{mux}
Read	Single	1	0	0	0
XNOR2	Dual	1	1	1	1
MAJ	Triple	1	0	0	0

B. RowHammer Attack Protection

The RH attack occurs if an aggressor row is frequently accessed, inducing a capacitive coupling between adjacent WLs

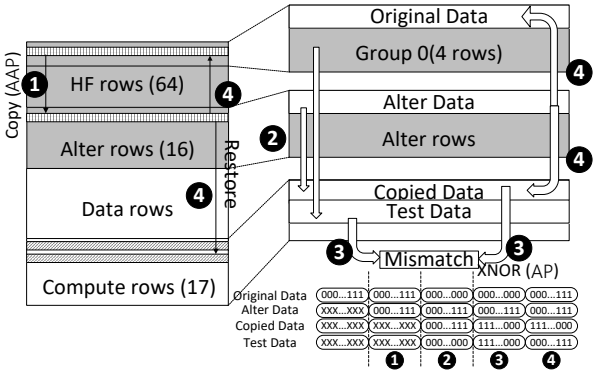

Fig. 4. The P-PIM-based RH attack protection with detection and retrieval.

[5]. Protecting the DRAM chip against the RH attack is a broad challenging task that can be translated into avoidance, detection, and retrieval mechanisms. Based on this concept, there are several mitigation approaches have been explored. In the proactive-refreshing approach, the victim rows are refreshed before the aggressor rows reach the RH threshold. To enable this, one counter is dedicated to each row to keep track of row activations [24]. However existing RH mitigation proposals face expensive hardware burdens. To improve this, a counter table can be inserted in the main memory [8], memory controller, or in SRAM-cache [25] to alleviate the performance penalty of counter data transfer. In the probabilistic approach, e.g., PARA [5], the avoidance is provided in a stateless manner by activating random rows with a probability, however, no retrieval mechanism is considered if a row is hammered. Herein, we show that P-PIM can protect (i.e., self-track and retrieve) data in any high-priority and frequently-accessed data rows defined by users against RH attacks in a cost-effective manner.

As shown in Fig. 4, we propose to partition the data rows in Fig. 2 and reserve 64 High-Frequent activated rows as *HF* and 16 empty rows as *Alter*. The HF and Alter rows are allocated to consecutive addresses. The HF rows are then virtually divided into groups of four. The proposed P-PIM-based RH protection mechanism is enabled in two phases, i.e., *i)* detection and *ii)* retrieval. According to the pertinent experimental data [1], the average time that one hammer can flip the row is ∼4ms in DDR3. Therefore, a Detection Interval (DI) is set to 1ms, and in each DI, the Alter rows are used for detection and retrieval. During the detection phase, at the beginning of the DI, P-PIM's controller issues an ACTIVATE-ACTIVATE-PRECHARGE (AAP) command to copy the Original Data from HF rows to Alter rows (Fig. 4 ❶), called Alter Data. Then at the end of DI, the Original Data and the Alter Data are copied to two compute rows, indicated by Test Data and Copied Data ❷. We opt for periodic detection to reduce energy consumption and support self-correction. P-PIM then issues an ACTIVATE-PRECHARGE (AP) command to perform XNOR operation between Copied Data and Test Data ❸. If the in-memory XNOR result is all "1"s, the Original Data is not flipped. Then P-PIM copies Alter Data to overwrite Original Data and repeat ❶ with another row in the group. During the retrieval phase, however, when the data is polluted (i.e., the in-memory XNOR result has at least one "0"), the Alter Data

Algorithm 1 P-PIM's RH Protection Algorithm

```
1:  Procedure: Protection
2:  if DI_Start then
3:      for group in HF_rows do
4:          select_row[group] ← BinaryTree(group)
5:          Alter_Data[group] ← select_row[group]
6:      end for
7:  else
8:      if DI_Detect then
9:          for row in select_row do
10:             Copied_Data ← select_row[row]
11:             Test_Data[group] ← Alter_Data[group]
12:             if !XNOR(Copied_Data,Test_Data) then
13:                 DI_Interrupt ← 1
14:                 select_row[group] ← Alter_Data
15:             else
16:                 DI_Interrupt ← 0
17:             end if
18:         end for
19:     end if
20:     else
21:     if DI_Interrup then break;
22:     end if
23:     end Procedure
```

is used to retrieve the original data ❹. The downside is that P-PIM is not able to locate the point at which the HF row was flipped (attacked), so we propose to back up the program at the beginning of the current DI and rerun it.

The P-PIM platform can protect the system against RH attacks. To detect the target rows quickly, we exploit the binary tree algorithm [6] (line-4 in Algorithm 1) to select the Copied Data since the row under the RH attack could bring up several victim rows. As shown in the P-PIM's RH protection algorithm, we consider three flags, i.e., DI_Start, DI_Detect, and DI_Interrupt to distinguish different steps in DI. DI_Start is dedicated to the beginning of DI, at which we create Alter_Data (line-5). When the detection phase in on (DI_Detect in line-8), P-PIM will perform a parallel single-cycle XNOR operation (line-12) to detect an RH attack and accordingly retrieve data if a row is attacked (line-14). DI_Interrupt is used only when the RH attack is detected and in this step, the program will roll back to the beginning of the DI. Fig. 5(a) shows a sample RH detection program used in P-PIM. We define the operations in DI0 and DI1 as OP0 and OP1, respectively, and the write-back operation of OP as WR. As can be seen, there is no error in DI0 and the RH attack occurs in DI1. As a result, the DI2 will repeat DI1 to assure the program is correct.

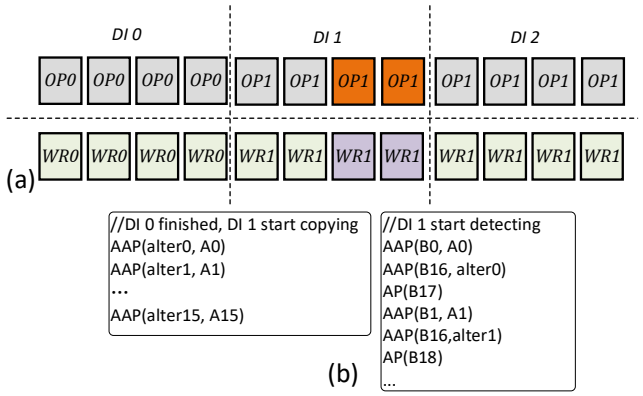

Fig. 5. (a) Timeline of a sample program with RH detection, (b) The μOp sequences for DI_Start and DI_Detect of DI.

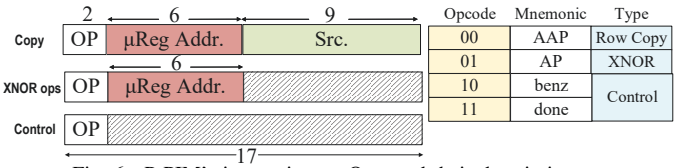

Fig. 6. P-PIM's instructions, μOps, and their description.

C. ISA Extension

From a programmer's standpoint, P-PIM is more of a third-party accelerator rather than a memory unit. Therefore, for general-purpose parallel execution, an ISA and virtual machine will be needed. With this, any user-level program can be translated at install time to the P-PIM's hardware instruction set. The P-PIM's μprogram consists of multiple instructions. As shown in Fig. 6. P-PIM is designed to process three 17-bit instruction types after compiling the upper-level code: (1) a copy instruction based on RowClone [12], (2) instruction for bit-wise XNOR operation, and (3) instruction for program control. Each P-PIM's μprogram then comprises a series of AAP and AP commands.

Row Clone. When OP is 00, P-PIM performs a row copy operation, which activates the source row and the destination row synchronously to share the charge.

Bit-wise XNOR operation. When OP is 01, the instruction performs a parallel XNOR operation. The 6-bit register address is the encoded address in the register addressed to the Compute rows, respectively. As shown in Table II, the first 17 rows of the register encode 16 Copied Data row T0-T15 and the Test Data row G0, separately, and the last 16 rows encode Test Data row with each row of 16 Copied Data row, separately. The query result will overwrite all the rows pointed by the instruction, as shown in Fig. 4.

Control. Opcodes 10 and 11 respectively represent simple control operations for loops and termination in the control flow. Also, it can jump to a specific position, which depends on the result given by the XNOR operation. To minimize the error ratio, we divide the program by the timescale equals 1ms. Fig. 5(b) gives an example of the μOp sequences for DI_Start and DI_Detect. Assuming DI0 has finished without errors, the DI1 starts right after that. The left box shows that the DRAM makes copies of all the original data. Then the program continues until DI1 enters the detection phase. The right box shows that to detect a row, the system will copy both original data and alter data to compute rows, then perform the XNOR operation. P-PIM will repeat this step until the error is detected.

TABLE II
COMPUTE ROWS' ADDRESSES MAPPING TO CORRESPONDING WL(s).

μReg.	WL(s)	μReg.	WL(s)	μReg.	WL(s)
B0∼B15	T0∼T15 addr	B16	G0 addr	B17∼B32	G0,T0∼T15 addr

IV. EVALUATION

A. Framework & Results

Framework. We demonstrate the advantages of P-PIM through a cross-layer evaluation framework as shown in Fig. 7. At the circuit-level, we first developed P-PIM's sub-arrays with new peripherals in Cadence Spectre with a 45nm NCSU PDK library [26] to verify the functionality and achieve the performance parameters. The memory controller and registers

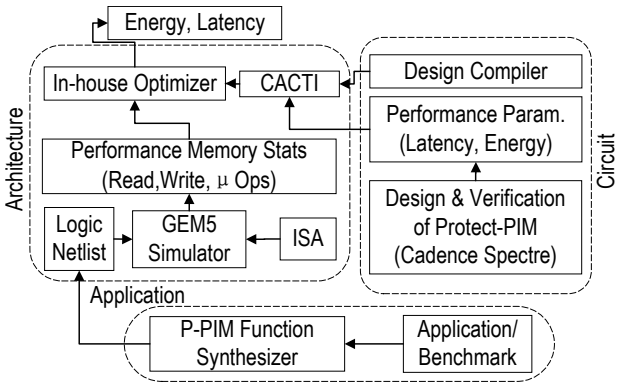

Fig. 7. Cross-layer evaluation framework for P-PIM architecture.

are designed and synthesized by Design Compiler with a 45nm industry library. At the architecture-level, we extensively modified CACTI [27] with the input from the circuit-level assessments. We then implemented P-PIM's ISA in gem5 [28] and exported the memory statistics and performance into an in-house optimizer, also taking the CACTI output as the input. At the application-level, we picked Advanced Encryption Standard (AES) workload and then gave it to the P-PIM's function synthesizer to convert it to a logic net-list and then to μOps by gem5.

Performance results. A comparison of the P-PIM's sub-array latency and state-of-the-art processing-in-DRAM designs (Ambit [10], SIMDRAM [16], DRISA-3T1C [9], GraphiDe [11], LAcc [13], and pPIM [19]) running basic logic operations (NOT, NAND, MAJ, X(N)OR) is reported in Fig. 8(a). Considering an ideal data layout [18], we constrained the designs by an 8GB memory capacity to get the best achievable performance in a narrow range of areas. As shown, thanks to the reconfigurable sense amplifier supporting both DRA and TRA mechanisms, P-PIM provides relatively fast and comparable performance for a variety of operations of interest. P-PIM offers the fastest XNOR operation ($\sim$270ns) as compared with other designs.

According to Fig. 8(b), P-PIM, Ambit [10], and GraphiDe [11] designs offer the least power consumption ($\sim$5W) compared to other designs due to their low power consumption sensing mechanism. We can also see the high power consumption of one of the LUT-based platforms called pPim [19] compared to our design. Fig. 9 analyzes and shows the energy-efficiency and performance per area parameters for various processing-in-DRAM architectures normalized to the P-PIM

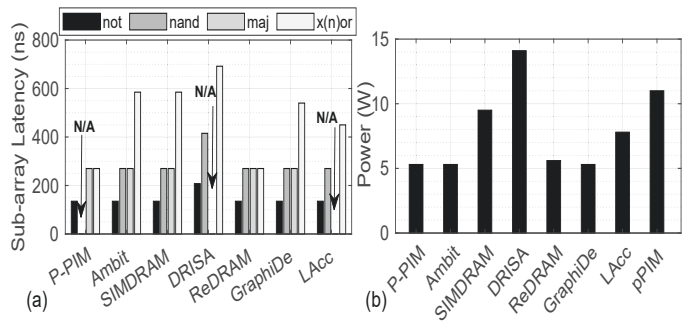

Fig. 8. (a) Sub-array latency in performing four basic bit-wise logic operations, and (b) Power consumption when the DRAM capacity is constrained by 8GB. N/A: Data is Not Achievable

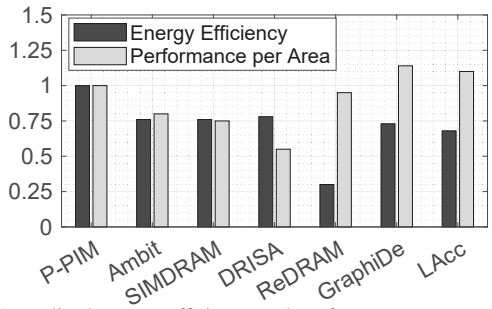

Fig. 9. Normalized energy efficiency and performance per area for various processing-in-DRAM architectures.

design. We observe that P-PIM achieves $\sim$72% higher energy-efficiency than that of the fastest charge-sharing-based design, i.e., ReDRAM [15]. However, GraphiDe [11] and LAcc [13] demonstrate higher performance per area compared to P-PIM.

B. Case study: in-DRAM Data Encryptor

The P-PIM architecture can provide encryption in high-assurance computing systems when the processor is typically trust-based. AES is an iterative symmetric-key cipher in which the sender and receiver use the same key to encrypt and decrypt. The AES algorithm is designed to work with 16-byte (128-bit) data organized in a 4×4 state matrix while using 3 different key lengths. SubBytes, ShiftRows, MixColumns, and AddRoundKey are enumerated as the transformations performed on input data for 128-bit key lengths. For ease of work with input data, each byte is divided into 8 bits resulting in 8 P-PIM sub-arrays filled with 4×4 bit-matrices. To accelerate each transformation inside the memory, P-PIM performs bulk bit-wise operations after mapping. The X(N)OR2 and (N)AND2 operations in SubBytes, MixColumns, and AddRoundKey stages contribute to more than 90% of the overall operations. In this study, we examine 128-bit AES implementations in terms of energy consumption and process cycles required on a General-Purpose Processor (GPP), ASIC, Ambit [10], DRISA-3T1C [9], ReDRAM [15], and P-PIM. For the GPP, AES C code is compiled, then cycle-accurate gem5 [28] is used to take AES binary and the McPAT [29] is used to estimate power dissipation. The Synopsys Design Compiler is used to implement the ASIC design (at 1.133GHz). Fig. 10(a) reports the number of cycles required for AES. ReDRAM and P-PIM require the least number of cycles (<560 cycles) compared with other processing-in-DRAM platforms and GPP. However, ASIC (with 336 cycles) shows better performance. Fig. 10(b) compares the energy-efficiency of the P-PIM with other platforms. As compared to GPP and ASIC designs, P-PIM shows 145× and $\sim$2.1× energy reduction, respectively. Moreover, P-PIM is found to reduce energy consumption by 58%, 45%, and 43%, respectively, relative to Ambit [10], DRISA-3T1C [9], and ReDRAM [15].

C. Row Hammer Mitigation

While there are several well-designed RH mitigation mechanisms in the literature, we only compare P-PIM with five designs as listed in Table III. The frameworks either directly leverage CMOS counters to detect the RH attack or employ fast

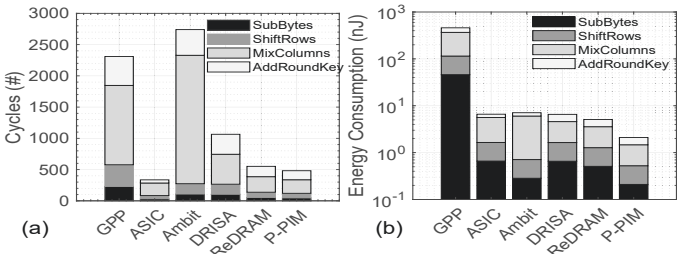

Fig. 10. Breakdown of (a) Number of cycles and (b) Energy consumption required for various AES implementations.

memories such as CAM and SRAM to store ACT. Our assessment shows that P-PIM is particularly suitable for DRAM self-detection of RH attacks. As discussed in Section III, scanning the whole DRAM in a single DI is unnecessary. Moreover, P-PIM can support fast row transfer and bit-wise operations. For this experiment, we followed the previous works' comparison strategy and took into account a 32GB DRAM capacity, and then normalized the slowdown and overhead for each bank. Here, we listed our key observations. *i)* Size per bank represents the capacity occupied for detecting RH. We can find that even the best frameworks such as Hydra [24] requires ∼1.76KB per bank from the SRAM. P-PIM achieves more than ∼92.8% space saving in memory capacity utilizing DRAM itself. *ii)* For a fair comparison of the latency, we expand our view on memories involved in the operation. Graphene [30], Hydra [24], and TWiCE [25] lead to latency in CAM and SRAM, while DRAM-based designs typically impose latency in DRAM. The baseline is a system that only supports the memory refresh with no further actions. P-PIM leverages row copy to restore the data, while other DRAM-based frameworks use refreshing. During the P-PIM detection phase, about 75% of operations are AAP, which cost much less than AP operations. As a result, in the best case, the slowdown of P-PIM increases up to 0.8% when all the rows are detected, which is about 10% overhead of other fast-memory-based frameworks [24], [25]. In the worst case, when the first selected row is flipped, the slowdown of P-PIM is 0.4%. The advantage is no SRAM or CAM resources are occupied for the detection. *iii)* Compared with the DRAM-only structure, the extra energy used for RH detection is represented as energy overhead. Our design achieves up to 71% energy-saving over the fast-memory-based frameworks and about 90% energy-saving over DRAM frameworks.

TABLE III
COMPARISON WITH PRIOR ROWHAMMAR MITIGATION FRAMEWORKS.

Framework	Memory involved	Size per bank (KB)	Slowdown	Energy overhead
graphene [30]	CAM-DRAM	2.45	0	0.34%
Hydra [24]	SRAM-DRAM	1.76	0.7%	0.2%
TWiCE [25]	SRAM/CAM-DRAM	21.68	0.7%	0.7%
Counter per Row	DRAM	4	0	21.92%
Counter Tree [6]	DRAM	0.25	0	1.37%-3.74%
P-PIM	DRAM	0.125	0.4%-0.8%	0.39%

V. CONCLUSION

In this work, a new processing-in-DRAM architecture named P-PIM was proposed with dual-row and triple-row activation support. P-PIM performs a complete in-DRAM RowHammer self-tracking and mitigation technique. Our evaluation showed that P-PIM achieves ∼72% higher energy efficiency than the fastest charge-sharing-based designs. As for the RowHammer protection, with a worst-case slowdown of ∼0.8%, it offers up to 71% energy-saving over the SRAM/CAM frameworks and about 90% saving over DRAM frameworks.

REFERENCES

[1] H. Hassan *et al.*, "Uncovering in-dram rowhammer protection mechanisms: A new methodology, custom rowhammer patterns, and implications," in *MICRO-54*, 2021, pp. 1198–1213.

[2] A. J. Walker *et al.*, "On dram rowhammer and the physics of insecurity," *TED*, vol. 68, 2021.

[3] Y. Kim *et al.*, "Flipping bits in memory without accessing them: An experimental study of dram disturbance errors," *ACM SIGARCH Computer Architecture News*, vol. 42, pp. 361–372, 2014.

[4] F. Yao *et al.*, "Deephammer: Depleting the intelligence of deep neural networks through targeted chain of bit flips," in *USENIX Security*, 2020, pp. 1463–1480.

[5] O. Mutlu and J. S. Kim, "Rowhammer: A retrospective," *IEEE TCAD*, vol. 39, pp. 1555–1571, 2019.

[6] S. M. Seyedzadeh *et al.*, "Counter-based tree structure for row hammering mitigation in dram," *CAL*, vol. 16, 2016.

[7] Yağlikçi *et al.*, "Blockhammer: Preventing rowhammer at low cost by blacklisting rapidly-accessed dram rows," in *HPCA*. IEEE, 2021, pp. 345–358.

[8] T. Bennett *et al.*, "Panopticon: A complete in-dram rowhammer mitigation," in *DRAMSec*, 2021.

[9] S. Li *et al.*, "Drisa: A dram-based reconfigurable in-situ accelerator," in *MICRO*. IEEE, 2017, pp. 288–301.

[10] V. Seshadri *et al.*, "Ambit: In-memory accelerator for bulk bitwise operations using commodity dram technology," in *Micro*. ACM, 2017, pp. 273–287.

[11] S. Angizi and D. Fan, "Graphide: A graph processing accelerator leveraging in-dram-computing," in *GLSVLSI*, 2019, pp. 45–50.

[12] V. Seshadri *et al.*, "Rowclone: Fast and energy-efficient in-dram bulk data copy and initialization," in *Micro*, 2013, pp. 185–197.

[13] Q. Deng *et al.*, "Lacc: Exploiting lookup table-based fast and accurate vector multiplication in dram-based cnn accelerator," in *DAC*, 2019.

[14] M. F. Ali *et al.*, "In-memory low-cost bit-serial addition using commodity dram technology," *TCAS I*, vol. 67, 2019.

[15] S. Angizi and D. Fan, "Redram: A reconfigurable processing-in-dram platform for accelerating bulk bit-wise operations," in *ICCAD*. IEEE, 2019, pp. 1–8.

[16] N. Hajinazar *et al.*, "Simdram: a framework for bit-serial simd processing using dram," in *asplos*, 2021, pp. 329–345.

[17] R. Zhou *et al.*, "Flexidram: A flexible in-dram framework to enable parallel general-purpose computation," in *ISLPED*, 2022, pp. 1–6.

[18] J. D. Ferreira *et al.*, "pluto: In-dram lookup tables to enable massively parallel general-purpose computation," *arXiv preprint arXiv:2104.07699*, 2021.

[19] P. R. Sutradhar *et al.*, "ppim: A programmable processor-in-memory architecture with precision-scaling for deep learning," *IEEE CAL*, vol. 19, 2020.

[20] P. Frigo *et al.*, "Trrespass: Exploiting the many sides of target row refresh," in *SP*. IEEE, 2020, pp. 747–762.

[21] P. Jattke *et al.*, "Blacksmith: Scalable rowhammering in the frequency domain," in *SP*, vol. 1, 2022.

[22] J. S. Kim *et al.*, "Revisiting rowhammer: An experimental analysis of modern dram devices and mitigation techniques," in *ISCA*. IEEE, 2020, pp. 638–651.

[23] M. W. Allam *et al.*, "High-speed dynamic logic styles for scaled-down cmos and mtcmos technologies," in *ISLPED*. ACM, 2000, pp. 155–160.

[24] M. Qureshi *et al.*, "Hydra: enabling low-overhead mitigation of row-hammer at ultra-low thresholds via hybrid tracking," in *ISCA*, 2022, pp. 699–710.

[25] E. Lee *et al.*, "Twice: Preventing row-hammering by exploiting time window counters," in *ISCA*, 2019, pp. 385–396.

[26] (2011) Ncsu eda freepdk45. [Online]. Available: http://www.eda.ncsu.edu/wiki/FreePDK45:Contents

[27] S. Thoziyoor *et al.*, "Cacti 5.1," Technical Report HPL-2008-20, HP Labs, Tech. Rep., 2008.

[28] N. Binkert *et al.*, "The gem5 simulator," *ACM SIGARCH computer architecture news*, vol. 39, pp. 1–7, 2011.

[29] S. Li *et al.*, "Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures," in *MICRO*. ACM, 2009, pp. 469–480.

[30] Y. Park *et al.*, "Graphene: Strong yet lightweight row hammer protection," in *MICRO*. IEEE, 2020, pp. 1–13.