Security Challenges of Processing-In-Memory Systems

Md Tanvir Arafin
mdtanvir.arafin@morgan.edu
Morgan State University
Baltimore, Maryland

Zhaojun Lu
lzj77521@umd.edu
University of Maryland
College Park, Maryland

ABSTRACT

Emerging memory systems such as resistive random access memory (RRAM), phase-change memory (PCM), and spin-transfer torque magneto-resistive random access memory (STT-MRAM) offer unique physical properties useful in designing next-generation processing in-memory (PIM) circuits and systems. Modified dynamic random access memory (DRAM) designs are also demonstrating on-chip data processing and bulk data operation capabilities. However, in-memory computation can fundamentally change the security models and assumptions of existing systems due to several key factors, such as modified system architecture, disparate programming models, side-channel effects, device reliability, hardware Trojans, and malicious perturbations in data processing. Therefore, in this paper, we survey and examine fundamental vulnerabilities arising from processing-in-memory systems. We aim to present the PIM system architects and designers an overview of security issues that can jeopardize the future of in-memory computation.

CCS CONCEPTS

• **Hardware → Memory and dense storage**; • **Security and privacy →** *Side-channel analysis and countermeasures*; *Embedded systems security*.

KEYWORDS

Processing-in-Memory (PIM); Resistive Random Access Memory (RRAM); Dynamic Random Access Memory (DRAM); Hardware Security.

ACM Reference Format:
Md Tanvir Arafin and Zhaojun Lu. 2020. Security Challenges of Processing-In-Memory Systems. In *Proceedings of the Great Lakes Symposium on VLSI 2020 (GLSVLSI '20), September 7–9, 2020, Virtual Event, China.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3386263.3411365

1 INTRODUCTION

John von Neumann's seminal work on the design of EDVAC [35] has dominated computer architecture and systems design for the last seventy-five years. It represents an efficient way of computing – moving the data and code from the memory system to a discrete central processing unit (CPU) using the system bus, and performing the computation described by the code on the fetched data in the CPU. However, with the expansion in data volume and increased processing power of CPUs, this *processor-centric* design starts suffering from data-movement congestion – commonly known as *von Neumann bottleneck*. Multi-level caches, modified Harvard architecture, branch predictors, scratchpad memories *etc.* have contributed to reducing the von Neumann bottleneck; yet, the processor/memory performance gap remains a steadily growing factor in modern computing [18].

The cost of data movement and the performance gap between the CPU and main memory set a substantial price on the total computation cost of emerging data-intensive applications [8, 20, 28, 29]. The main memory in modern computing systems consists of dynamic random access memories (DRAM). DRAMs enjoy low-cost, advanced fabrication technology, and high-data storage capacity. However, DRAM-based systems are still increasingly facing challenges in supporting computation involving large datasets [8, 20, 28, 29]. For example, algorithms in the realm of artificial intelligence (AI) and machine learning (ML) require to process large volumes of data to infer meaningful information. These computations are heavily throttled by the relatively lower memory channel width, more complex read and retention mechanisms in DRAMs, longer latency associated with random access patterns, and the main memory's energy consumption. Thus, contemporary *data-centric* computation is demanding a paradigm shift in the classical *processor-centric* system design [7, 8, 16, 20].

Processing-in-memory (PIM) is one of the critical ideas of data-centric system design. PIM designs optimize system performance by moving computation to (or near) the data in memory for a large-scale dataset. Near-memory designs utilize simple in-order processors at the memory die to support basic computation [17, 31]. In contrast, in-memory computation achieves arithmetic and/or logical operation using reconfigured memory cells [7, 16, 28, 29]. PIM minimizes the data-transfer for repeated straightforward tasks and allows the CPU to spend resources on processing complex computation. Unfortunately, existing DRAM systems are not capable of executing any form of data-processing in the main memory. Thus, fundamental changes in the DRAM system design have been proposed to support PIM architectures [10, 20, 28, 29]. On the other hand, emerging memory technologies such as resistive random access memory (RRAM), phase-change memory (PCM), and spin-transfer torque magneto-resistive random access memory (STT-MRAM) can disrupt DRAMs dominance in main memory design [7, 14, 16, 30, 32, 36]. Furthermore, these memory components are proven to be capable of supporting data processing in memory due to the distinctive physics of operation of these systems [3, 14, 16, 37].

GLSVLSI '20, September 7–9, 2020, Virtual Event, China
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-7944-1/20/09…$15.00
https://doi.org/10.1145/3386263.3411365

For example, recent progress in the design and implementation of resistive random access memory (RRAM) components has introduced opportunities for developing novel in-memory computing solutions using the non-volatile resistive state-preserving analog properties of these devices [7, 22, 30]. In-memory computations such as arithmetic addition, subtraction, matrix-multiplication, and fundamental logic operations have been demonstrated in RRAM systems [2–4, 7, 22, 30]. These computations can substantially accelerate data-intensive applications and services. Furthermore, unique device physics of RRAM is reported to be useful in approximate computing techniques, low-power system design, and development of hardware-oriented security primitives [2–4].

However, in-memory computing opportunities in emerging devices can open up new security and trust issues. In recent years, main memory systems are found to be vulnerable to attacks from hardware and software. Hardware oriented attacks such as the Rowhammer attack [9, 11] and its variants [24, 34] have exposed how physical proximity in data-rows in a DRAM can be exploited for malicious attacks. On the other hand, side-channel attacks on cache subsystems are effective against established cryptography protocols [23, 41]. Additionally, emerging attacks on system architecture such as Spectre [12], Meltdown [15], VoltJockey [25, 26] *etc.* are motivating computer architecture researchers to rethink and reevaluate the established architecture fundamentals for securing data and computation. These attacks exist primarily due to the lack of security consideration in the system development phase. Therefore, to avoid common pitfalls, emerging memory systems must consider the security implications of their design and implementation before wide-scale adaptation.

This paper investigates security and trust issues regarding PIM paradigms. Section 2 introduces common PIM structures, and Section 3 discusses the security vulnerabilities of these architectures. In Section 4, we move deeper into the circuit level to examine PIM's potential hardware security concerns. Finally, Section 5 concludes the paper.

2 PIM ARCHITECTURE

The integration of PIM-subsystems in modern computer architecture is an active area of research. Since PIM redefines the memory systems' capability, established design principles for programming models, cache coherence, virtual memory, and overall memory management mechanism must be redesigned to support computation in the main memory. PIM systems can be divided into three categories based on their architecture:

1. PIM integrated architecture;
2. PIM isolated architecture; and
3. Hybrid architecture.

PIM integrated design targets to reduce the overall changes from the existing system architecture. For example, Ahn *et al.* [1] has described PIM-enabled instructions (PEI) that introduce new instruction(s) for PIM related computation. PEIs can be executed either on the CPU cores or in the in-memory computation logic. Similar designs *i.e.,* [6, 29] takes the benefits of on-chip cache, virtual memory, and established memory control mechanism to perform computation using either the CPU or in-memory logic components based on the optimization constraints. Therefore, PIM integrated designs tend to expose the CPU core, on-chip caches, and shared memory areas to the PIM subsystems and *vice-versa.*

On the other hand, PIM isolated designs delegate a specific task altogether to a memory component that performs the logical and arithmetic computation required for the task. One such application is the deep neural network (DNN) based calculation using in-memory processing. Since new memory components such as RRAMs can perform logic operations and in-memory matrix multiplication by leveraging the crossbar structure, isolated DNN-engines and accelerators have been proposed in recent literature [7, 16, 22, 37]. For these isolated designs, arbitrary computations cannot be offloaded to PIM, and Therefore, they are more rigid and targeted towards data-intensive repeated computations.

The design dissimilarities discussed above lead to different security models for these PIM architectures. In this work, we assume a simple PIM architecture presented in Figure 1 that can represent both the integrated and isolated design. The PIM subsystem in Figure 1 is divided into two components: (1) a PIM controller, and (2) a PIM-compatible memory. The PIM controller manages PIM operations between the host CPU and the PIM compatible memory. The host CPU can execute or delegates the PIM instructions to the PIM controller depending on nature (*i.e.,* integrated or isolated) of the system architecture. Additionally, we assume that in PIM-integrated designs, a PIM controller has direct access to the host caches (as commonly found in [6, 10, 29]). The controller manages operations such as instruction atomicity management, cache coherence, locality monitoring for optimized data processing.

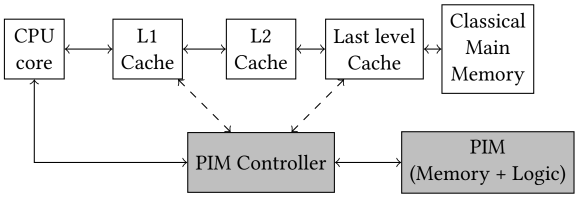


Figure 1: A simplified system architecture for integrating PIM subsystems. The dashed lines denote optional connectivity between different subsystems based on the PIM architecture. An attacker can choose to target vulnerabilities in any of the subsystems for compromising in-memory computations.

For the isolated designs, the PIM controller does not directly access the host CPU's internal caches or on-chip mechanisms; instead, the host CPU communicates with the PIM controller to delegate computation and fetch computation results. Moreover, the host CPU does not have access to the contents of the PIM memory. The PIM controller may have internal caches, logic, and memory management components for accelerating the fetch operation requested by the host CPU. By definition, there is minimal interaction between the host CPU and the PIM controller, and the PIM subsystem will behave as a separate device. It should be noted that there can also be hybrid systems that (1) leverage the isolation of PIM's computation logic for simpler memory management, and (2) integrate parts of the PIM controller with the host processes for optimum control.

From a security standpoint, a PIM integrated architecture has a larger attack surface than an isolated design. Hence, in the early stages of the PIM development, architecture selection for in-memory computing will determine its performance and resiliency against malicious adversaries.

3 GENERAL SECURITY ISSUES IN PIM ARCHITECTURE

PIM moves computation within the memory components, and thus offer attacker opportunities in corrupting data and computation. A malicious program with access to the PIM kernels can easily achieve the following goals:

- Leak data from memory (malicious read);
- Corrupt stored data (malicious write);
- Leak information about data (malicious PEI execution)

Current research on PIM integration illustrates the inevitable changes in the existing programming models and data-structures [1, 7, 10, 21, 29] . PIM systems have shown promises in executing a wide range of mathematical and logical operations, as listed in Table 1. Integrating these computations will require an extension of existing instruction set architecture (ISA). Without proper security-oriented foresight, these extended ISA would suffer from numerous hardware and software vulnerabilities. Some of these weaknesses are discussed below.

Type	Operation
Memory	Bulk Load, Store Row-clone [27]
Arithmetic	Integer addition, multiplication [39] Floating point addition [1] Integer increment [1] Integer min [1, 8]
Matrix Operation	Dot product, Euclidean Distance [1] Convolution [7, 30]
Logic	AND, OR, NOT [29, 36] IMPLY [38] Minority [32]
Algorithmic	Hash table manipulation [1] Pointer Chasing [10]

Table 1: A list of common PIM operations found in current literature.

3.1 Memory Corruption

Memory corruption due to insecure coding practices is one of the most prevailing security concerns for modern software and computing systems [33]. The key idea for memory corruption is to exploit vulnerable code for accessing out-of-the-bound memory addresses and corrupt the data in that location. Consequently, if PIM integrated architectures do not deploy strict memory access rules, we will observe memory corruption attacks in PIM.

For in-memory computation, data corruption at a PIM address has two-fold consequences: (a) presence of invalid data in the system, and (b) erroneous results for in-memory computation. Moreover, PIM enables stored data to be transformed via PIM operations. This makes attack detection, and error correction challenging since both an attack and a PIM operation changes the data. Additionally, PIM enables bulk load, store, and data cloning, which, in-turn, allows a malicious adversary to corrupt a block of data with a few PEIs.

Address space layout randomization (ASLR) is a common defense for memory corruption attacks. Therefore, modification to the system's ASLR logic will be required for PIM. Data space randomization [5] that obfuscates data representation is also a potential candidate for defending PIM systems against memory corruption.

3.2 Information Leak

In-memory computation is problematic from an information security perspective. Not only a successful breach will expose memory contents, but also PEIs such as logic operations and max/min find will be useful in deriving aggregate information from the data. Furthermore, non-volatile PIM memories will provide attack opportunities even when the device is powered off, which can expose the stored data, and the computation previously executed on the data. As a result, data, computation, and intellectual property (IP) protection techniques for PIM-based algorithms should be investigated to defend PIM systems against information leakage.

3.3 Cache Side-Channel Attacks

In the integrated PIM model, an in-memory computation can be handled by either the CPU or the PIM subsystems [1, 6]. Therefore, cache-coherence is an essential issue in maintaining data integrity. Ahn *et al.* [1] describes one of the most straightforward solutions for cache coherence, where the authors suggested to use back-invalidation and back-writeback mechanisms for coherent operation. The PIM-controllers have *cache invalidation* authority to invalidate and writeback the last level cache when the computation is offloaded to PIM. As side-channel attacks are commonplace for existing cache architecture, similar attacks can be expected with PIM integrated design. To understand this issue, we model a single cache line for PIM integrated design using a finite state machine in Figure 2, based on the discussions in [41].

Here we assume that a single cache-line can exist in the following states: I: invalid state, H: occupied by the host processor, P: occupied by the PIM controller, and A: occupied by an attacker. All the miss (or hit) events represent that the regarding entity has a cache miss (or hit) for a memory line that maps into the cache line. For a conventional cache, $A \rightarrow I$ and $H \rightarrow I$ transitions will not occur during load and store; however, if a PIM controller is introduced, back-invalidation [1] procedure will lead to a $P \rightarrow I$ transitions.

Given the FSM in Figure 2, it is evident that there is a substantial amount of attack surface for cache-based side-channel attacks in PIM integrated systems, that can leak information not only about the computation at the host CPU but also, in-memory computation. Let us consider O as an attacker's set of observations for the input set I on the cache line. Then, the mutual information L leaked due to cache side-channel can be written as [41]:

$$L = \sum_{i \in I} \sum_{o \in O} P_{I,O}(i,o) log \frac{P_{I,O}(i,o)}{P_I(i) P_O(o)} \tag{1}$$

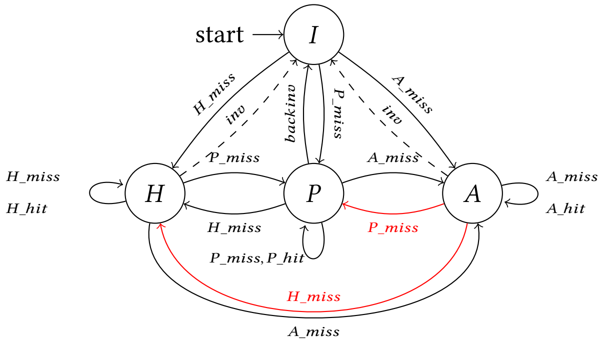


Figure 2: A finite state machine representation of a single cache line in PIM integrated architecture. Instances of external interference by an attacker are shown in red. Spy-Trojan (i.e., external timing-based attacks) such as the ones presented by Percival [23] can be executed using external interference paths.

where, $P_I(i)$ and $P_O(o)$ represents the probability of the input i and output o respectively. Now, if we assume a system with n cache lines, and I_h and I_p as the host's and PIM controllers input on the cache line, and the attacker's observation as O_a, then the interference probabilities for a PIM process ((i.e.,) $P_{I,O}(i, o)$ in Eqn. 1) can be written as, [41]:

$$P(I_p, O_a) = \frac{N(I_p, O_a)}{\sum_{0\le h'<n}\sum_{0\le a'<n} N(I_{h'}, O_{a'}) + \sum_{0\le a'<n} N(I_{inv}, O_{a'}) + \sum_{0\le p'<n}\sum_{0\le a'<n} N(I_{p'}, O_{a'})} \qquad (2)$$

where, $N(i, j)$ represents the count of interference $i \longrightarrow j$. Interestingly, from equation 2, it is evident that the host CPU's existence will make the cache side-channel attack on the PIM process more difficult for an adversary. This is not counter-intuitive; rather, it represents the increased noise on the cache line due to both the PIM controller and the CPU.

Defense techniques for cache side-channel attacks can be derived from the FSM in Figure 2. External interference attacks are eliminated by removing $A \longrightarrow P$ paths using statically partitioned caches. Moreover, random invalidation can be introduced to increase $N(I_{inv}, O'_a)$ term in Eqn 2. Also, a shift towards the isolated PIM architecture can reduce the cache-side channel attacks.

3.4 Denial of Service

The next class of attacks on the PIM subsystems can originate from denial-of-service (DoS) on the computation executed in memory. Memory performance attacks using DoS have already been reported for DRAM systems due to the unfairness in-memory sharing policies in multi-core architectures [19]. For PIM integrated designs, maintaining the PIM operation's atomicity can provide attackers opportunities to create a denial of service attacks.

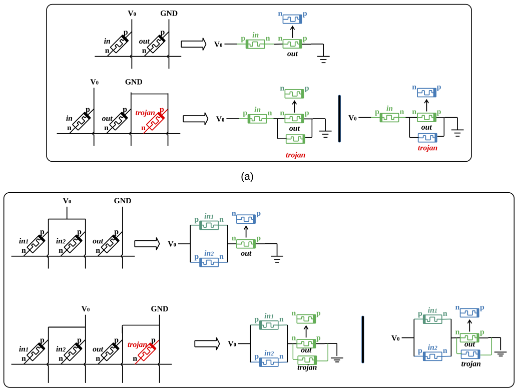
For example, Ahn *et.al.* [1] suggested the use of PIM directory (PIMD) for PEI atomicity management. The directory determines whether a memory block can be read/written during a given clock cycle. Therefore, targeted attacks on directory management can lead to delays in PIM access. An attacker may block the read (or write) operation by corrupting PIMD entries. As a result, both memory contention and scheduling-based attacks can be achieved by exploiting the PIMD.

4 HARDWARE-ORIENTED SECURITY AND TRUST FOR IN-MEMORY PROCESSING

In-memory computation would also be vulnerable to hardware attacks. It has been demonstrated that exploitation of hardware vulnerabilities such as an attack on the DRAM memory refresh controller can significantly alter the code and data for computation [40]. Similar attacks can jeopardize the in-memory computation on emerging memory systems (*i.e.,* RRAM, STT-MRAM, PCM). On the other hand, threshold alteration, voltage set-up manipulation, compromise in the sensing circuit, and muting comparators *etc.* can be used to attack resistive memories such as RRAMs and other PIM-systems.

4.1 Hardware Trojans

Trojan design and detection have been explored in current hardware security literature for CMOS-based integrated circuits and related components such as printed circuit board (PCB) *etc.* However, with the computations shift towards memory, hardware Trojans will also emerge for PIM systems. These Trojans can potentially be camouflaged in the memory crossbars. As an example, we present simple designs for memory Trojans in RRAM-based logic gates in Figure 3.



(a)

(b)

Figure 3: An example of Trojan insertion in RRAM-based logic gates. We have used MAGIC gates[13] for the logic operation. A blue RRAM represents a device in a low resistive state (logic 1), and a green RRAM represents a device in a high resistive state (logic 0). We insert a controllable RRAM (red) at the output of the logic gates. (a) Top: Correct operation of a NOT gate ; (b) Functionality of the NOT gate is corrupted by the Trojan. If the Trojan RRAM is in a low resistive state, the gate functions as before; however, if the Trojan is in a high resistive state, the inverter fails. (b) Top: Correct operation of a NOR gate ; (b) Corrupted functionality of the NOR gate with the Trojan inserted.

4.2 Fault Injection

Fault introduced at the memory array can be devastating for both computation and data storage. Both conventional DRAM-based systems and emerging memories (*i.e.,* RRAM, STT-MRAM) suffer from fault injection. Intentional fault injection in non-volatile components can persist over power-cycles, and therefore, can hide in the system to be triggered later.

Another security concern for the emerging memory system is stuck-at faults. These faults occur from fabrication defects, aging, and unbalanced memory operation. Malicious operation with a high bias voltage also leads to permanent damage to the memory cell. An example of a non-recoverable fault is presented in Figure 4. As the memory components participate in computation, stuck-at errors can have detrimental effects on in-memory computation. Therefore, the impact of persistent and non-persistent fault injection attacks in commonly used PIM computation (*e.g.*, Table 1) requires careful investigation before their deployment in security-critical systems.

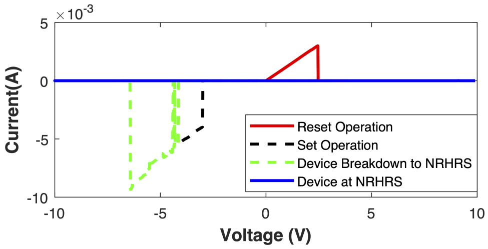


Figure 4: Sample I-V curve for the SET/RESET operation and hard breakdown of the fabricated HfO_x-based RRAM [4]. Hard dielectric breakdown occurs for the device when a larger RESET voltage with fast ramping is applied along the devices. After the breakdown, the device becomes incapable of switching to a low resistive state even with larger SET voltages.

4.3 Read/Write Unbalance and Malicious Wear-out

Rowhammer attack [9, 11] in the DRAM system is an excellent example of malicious memory corruption via repeated benign access. Similar attacks are achievable in emerging PIM systems. For example, assume two RRAM memory slices in the same 3D-stack, one is used for PIM computation, and the other is being used for memory operation. If the slices share the same voltage source, unintentional bit flips can occur due to power-demanding operation in the PIM section. Excessive voltage drops can be generated from legitimate computation in the PIM-slice, which will cause a voltage drop in the READ/WRITE operation in the memory slice leading to an unbalanced SET-RESET condition as shown in Figure 5. This can induce unintentional bit-flips during the write operation in the memory. Similar exploitation for STT-MRAM has been reported recently in [36]. Therefore, modified versions of the Rowhammer attack should be successful in compromising in-memory computation.

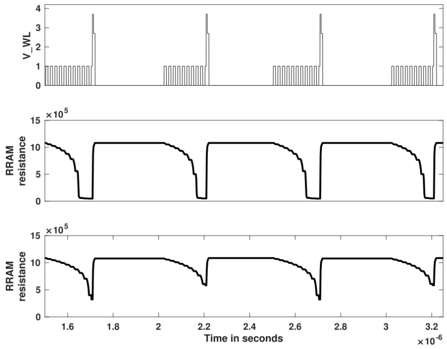


Figure 5: An example of read/write unbalance for RRAM circuits [3]. Top: typical-pulsed programming for an RRAM device. Middle: changes in an RRAM resistance for a stable SET/RESET condition. Bottom: effect of unbalanced biasing where the SET voltage is lowered by 12 mV than the previous balanced condition, and the RESET voltage is kept the same as before.

4.4 Electromagnetic Side-Channel Attacks

Although cryptographic computation in the PIM components is yet to be explored in detail, electromagnetic side-channels of PIM systems can compromise such calculation. Logic operations in-memory are substantially different processes than standard memory access, which leads to distinguishable power profiles between logic and memory operation. Therefore, without defensive mechanisms in the PIM controller, power-analysis based side-channel attacks can potentially reveal the computation steps and the keys for a cryptographic operation.

In sum, hardware attacks that are currently investigated by the hardware security community have the potential to affect the next generation of in-memory computation. Defense against these attacks will require a thorough understanding of the hardware and physical design of PIM systems. Therefore, hardware security informed design decisions must be incorporated early in developing PIM systems.

5 CONCLUSIONS

In-memory processing must acknowledge the security concerns related to the paradigm-shifting progress of this emerging technology. As discussed in this paper, PIM can make an existing system architecture more vulnerable to information leakage, cache attacks, and hardware-based attacks. Similarly, a carefully designed PIM architecture will offer significant opportunities in developing trusted and secure systems. In conclusion, this work serves to stimulate discussion and early research on the security challenges of system design for in-memory computation.

ACKNOWLEDGMENTS

We thank Dr. Gang Qu for his invaluable input in this work. We also thank Dr. Hassan Salmani for kindly reviewing the manuscript.

REFERENCES

[1] Junwhan Ahn, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. 2015. PIM-enabled instructions: a low-overhead, locality-aware processing-in-memory architecture. In *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 336–348.

[2] Md Tanvir Arafin, Carson Dunbar, Gang Qu, N McDonald, and L Yan. 2015. A survey on memristor modeling and security applications. In *Quality Electronic Design (ISQED), 2015 16th International Symposium on*. IEEE, 440–447.

[3] Md Tanvir Arafin and Gang Qu. 2018. Memristors for Secret Sharing-Based Lightweight Authentication. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 99 (2018), 1–13.

[4] Md Tanvir Arafin, Haoting Shen, Mark M. Tehranipoor, and Gang Qu. 2019. LPN-Based Device Authentication Using Resistive Memory. In *Proceedings of the 2019 on Great Lakes Symposium on VLSI* (Tysons Corner, VA, USA) *(GLSVLSI '19)*. Association for Computing Machinery, New York, NY, USA, 9–14.

[5] Sandeep Bhatkar and R Sekar. 2008. Data space randomization. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 1–22.

[6] A. Boroumand, S. Ghose, M. Patel, H. Hassan, B. Lucia, K. Hsieh, K. T. Malladi, H. Zheng, and O. Mutlu. 2017. LazyPIM: An Efficient Cache Coherence Mechanism for Processing-in-Memory. *IEEE Computer Architecture Letters* 16, 1 (2017), 46–50.

[7] Ping Chi, Shuangchen Li, Cong Xu, Tao Zhang, Jishen Zhao, Yongpan Liu, Yu Wang, and Yuan Xie. 2016. Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory. In *ACM SIGARCH Computer Architecture News*, Vol. 44. IEEE Press, 27–39.

[8] Saugata Ghose, Amirali Boroumand, Jeremie S Kim, Juan Gómez-Luna, and Onur Mutlu. 2019. Processing-in-memory: A workload-driven perspective. *IBM Journal of Research and Development* 63, 6 (2019), 3–1.

[9] Daniel Gruss, Clémentine Maurice, and Stefan Mangard. 2016. Rowhammer. js: A remote software-induced fault attack in javascript. In *International conference on detection of intrusions and malware, and vulnerability assessment*. Springer, 300–321.

[10] K. Hsieh, S. Khan, N. Vijaykumar, K. K. Chang, A. Boroumand, S. Ghose, and O. Mutlu. 2016. Accelerating pointer chasing in 3D-stacked memory: Challenges, mechanisms, evaluation. In *2016 IEEE 34th International Conference on Computer Design (ICCD)*. 25–32.

[11] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. 2014. Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. *ACM SIGARCH Computer Architecture News* 42, 3 (2014), 361–372.

[12] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom. 2019. Spectre Attacks: Exploiting Speculative Execution. In *2019 IEEE Symposium on Security and Privacy (SP)*. 1–19.

[13] Shahar Kvatinsky, Dmitry Belousov, Slavik Liman, Guy Satat, Nimrod Wald, Eby G Friedman, Avinoam Kolodny, and Uri C Weiser. 2014. MAGIC—Memristor-aided logic. *IEEE Transactions on Circuits and Systems II: Express Briefs* 61, 11 (2014), 895–899.

[14] Haitong Li, Tony F Wu, Subhasish Mitra, and H-S Philip Wong. 2017. Resistive RAM-centric computing: Design and modeling methodology. *IEEE Transactions on Circuits and Systems I: Regular Papers* 64, 9 (2017), 2263–2273.

[15] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. 2018. Meltdown: Reading Kernel Memory from User Space. In *Proceedings of the 27th USENIX Conference on Security Symposium* (Baltimore, MD, USA) *(SEC'18)*. USENIX Association, USA, 973–990.

[16] Qi Liu, Bin Gao, Peng Yao, Dong Wu, Junren Chen, Yachuan Pang, Wenqiang Zhang, Yan Liao, Cheng-Xin Xue, Wei-Hao Chen, et al. 2020. 33.2 A Fully Integrated Analog ReRAM Based 78.4 TOPS/W Compute-In-Memory Chip with Fully Parallel MAC Computing. In *2020 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 500–502.

[17] Zhiyu Liu, Irina Calciu, Maurice Herlihy, and Onur Mutlu. 2017. Concurrent data structures for near-memory computing. In *Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures*. 235–245.

[18] Sally A. McKee and Robert W. Wisniewski. 2011". *Memory Wall*. Springer US, Boston, MA, 1110–1116.

[19] Thomas Moscibroda and Onur Mutlu. 2007. Memory performance attacks: denial of memory service in multi-core systems. In *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*. 1–18.

[20] Onur Mutlu. 2019. Processing Data Where It Makes Sense in Modern Computing Systems: Enabling In-Memory Computation. In *Proceedings of the 2019 on Great Lakes Symposium on VLSI* (Tysons Corner, VA, USA) *(GLSVLSI '19)*. Association for Computing Machinery, New York, NY, USA, 5–6. https://doi.org/10.1145/3299874.3322805

[21] D. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Keeton, C. Kozyrakis, R. Thomas, and K. Yelick. 1997. A case for intelligent RAM. *IEEE Micro* 17, 2 (March 1997), 34–44. https://doi.org/10.1109/40.592312

[22] X. Peng, R. Liu, and S. Yu. 2019. Optimizing Weight Mapping and Data Flow for Convolutional Neural Networks on RRAM Based Processing-In-Memory Architecture. In *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*. 1–5.

[23] Colin Percival. [n.d.]. Cache missing for fun and profit.

[24] Peter Pessl, Daniel Gruss, Clémentine Maurice, Michael Schwarz, and Stefan Mangard. 2016. DRAMA: Exploiting DRAM addressing for cross-cpu attacks. In *25th USENIX Security Symposium (USENIX Security 16)*. 565–581.

[25] Pengfei Qiu, Dongsheng Wang, Yongqiang Lyu, and Gang Qu. 2019. VoltJockey: Breaching TrustZone by Software-Controlled Voltage Manipulation over Multi-Core Frequencies. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security* (London, United Kingdom) *(CCS '19)*. Association for Computing Machinery, New York, NY, USA, 195–209.

[26] P. Qiu, D. Wang, Y. Lyu, and G. Qu. 2019. VoltJockey: Breaking SGX by Software-Controlled Voltage-Induced Hardware Faults. In *2019 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*. 1–6.

[27] Vivek Seshadri, Yoongu Kim, Chris Fallin, Donghyuk Lee, Rachata Ausavarungnirun, Gennady Pekhimenko, Yixin Luo, Onur Mutlu, Phillip B Gibbons, Michael A Kozuch, et al. 2013. RowClone: fast and energy-efficient in-DRAM bulk data copy and initialization. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*. 185–197.

[28] Vivek Seshadri, Donghyuk Lee, Thomas Mullins, Hasan Hassan, Amirali Boroumand, Jeremie Kim, Michael A Kozuch, Onur Mutlu, Phillip B Gibbons, and Todd C Mowry. 2016. Buddy-ram: Improving the performance and efficiency of bulk bitwise operations using DRAM. *arXiv preprint arXiv:1611.09988* (2016).

[29] Vivek Seshadri, Donghyuk Lee, Thomas Mullins, Hasan Hassan, Amirali Boroumand, Jeremie Kim, Michael A Kozuch, Onur Mutlu, Phillip B Gibbons, and Todd C Mowry. 2017. Ambit: In-memory accelerator for bulk bitwise operations using commodity DRAM technology. In *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 273–287.

[30] Ali Shafiee, Anirban Nag, Naveen Muralimanohar, Rajeev Balasubramonian, John Paul Strachan, Miao Hu, R Stanley Williams, and Vivek Srikumar. 2016. ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. *ACM SIGARCH Computer Architecture News* 44, 3 (2016), 14–26.

[31] Gagandeep Singh, Lorenzo Chelini, Stefano Corda, Ahsan Javed Awan, Sander Stuijk, Roel Jordans, Henk Corporaal, and Albert-Jan Boonstra. 2018. A review of near-memory computing architectures: Opportunities and challenges. In *2018 21st Euromicro Conference on Digital System Design (DSD)*. IEEE, 608–617.

[32] Mathias Soeken, Saeideh Shirinzadeh, Pierre-Emmanuel Gaillardon, Luca Gaetano Amarú, Rolf Drechsler, and Giovanni De Micheli. 2016. An MIG-based compiler for programmable logic-in-memory architectures. In *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. Ieee, 1–6.

[33] Laszlo Szekeres, Mathias Payer, Tao Wei, and Dawn Song. 2013. Sok: Eternal war in memory. In *2013 IEEE Symposium on Security and Privacy*. IEEE, 48–62.

[34] Victor Van Der Veen, Yanick Fratantonio, Martina Lindorfer, Daniel Gruss, Clémentine Maurice, Giovanni Vigna, Herbert Bos, Kaveh Razavi, and Cristiano Giuffrida. 2016. Drammer: Deterministic rowhammer attacks on mobile platforms. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. 1675–1689.

[35] John von Neuman. 1945. First Draft of a Report on the EDVAC. (1945).

[36] Xueyan Wang, Jianlei Yang, Yinglin Zhao, Xiaotao Jia, Gang Qu, and Weisheng Zhao. [n.d.]. Hardware Security in Spin-Based Computing-In-Memory: Analysis, Exploits, and Mitigation Techniques. *ACM Journal on Emerging Technologies in Computing Systems (JETC)* ([n. d.]).

[37] Huaqiang Wu, Peng Yao, Bin Gao, Wei Wu, Qingtian Zhang, Wenqiang Zhang, Ning Deng, Dong Wu, H-S Philip Wong, Shimeng Yu, et al. 2017. Device and circuit optimization of RRAM for neuromorphic computing. In *2017 IEEE International Electron Devices Meeting (IEDM)*. IEEE, 11–5.

[38] Cong Xu, Xiangyu Dong, Norman P Jouppi, and Yuan Xie. 2011. Design implications of memristor-based RRAM cross-point structures. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011*. IEEE, 1–6.

[39] Sheng Xu, Xiaoming Chen, Ying Wang, Yinhe Han, Xuehai Qian, and Xiaowei Li. 2018. PIMSim: A flexible and detailed processing-in-memory simulator. *IEEE Computer Architecture Letters* 18, 1 (2018), 6–9.

[40] Pruthvy Yellu, Novak Boskov, Michel A. Kinsy, and Qiaoyan Yu. 2019. Security Threats in Approximate Computing Systems. In *Proceedings of the 2019 on Great Lakes Symposium on VLSI*. Association for Computing Machinery, New York, NY, USA, 387–392. https://doi.org/10.1145/3299874.3319453

[41] Tianwei Zhang and Ruby B Lee. 2014. New models of cache architectures characterizing information leakage from cache side channels. In *Proceedings of the 30th annual computer security applications conference*. 96–105.