Rethinking ECC in the Era of Row-Hammer

Moinuddin Qureshi

Abstract—DRAM is susceptible to Row-Hammer failures. Recent attacks show Row-Hammer can be leveraged to access unauthorized data and such attacks continue to break existing solutions for Row-Hammer mitigation. Recent research showed that Row-Hammer can be performed even on ECC memories, while these memories implement SECDED or Chipkill codes. We observe that existing ECC designs try to use the ECC bits to maximize correction, whereas the detection capability is simply a by-product of correction code. In this paper, we argue that the ECC designs of DRAM memories should be re-architected in the era of Row-Hammer – ECC space should provide strong detection (integrity protection), which can detect Row-Hammer failures. This would devolve the Row-Hammer problem from a security issue to a reliability issue and provide an "insurance policy" against future unknown attacks and newer failure modes that can corrupt memory contents. In this paper, we show that existing conventional designs for ECC-1 and Chipkill can be re-architected to provided strong detection while retaining similar level of correction capability as conventional designs. We accomplish this by foregoing the conventional constraint of implementing ECC at word-granularity (8 bytes) and instead implement ECC at a line-granularity (64 bytes) which is more storage-efficient and provides bits for implementing strong detection.

1 Background: Row-Hammer 101

DRAM scaling brings cells closer to each other and increases coupling between cells. Row-Hammer occurs when one row in the memory receives a large number of activations, and the rows neighboring this row do not have an activation (otherwise these rows would get a precharge and restore the data back to its original state). Figure 1 captures the problem of Row-Hammer, for a given row X, where the neighboring rows are labeled X-1 and X+1. If X is accessed frequently, and X-1 and X+1 are not accessed, then the contents of these neighboring rows can get corrupted due to Row-Hammer. The threshold for the number of activations within a refresh cycle required to cause data loss due to Row Hammering is called the *Row-Hammering Threshold*. With each technology generation, this threshold reduces, making the Row-Hammer problem much severe for both current and future DRAMs.

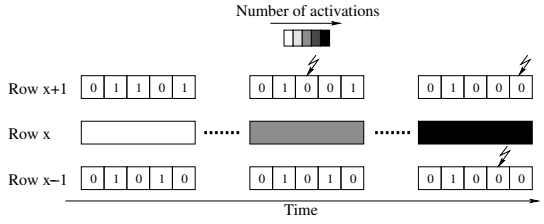

Fig. 1. Row Hammer Attack on Row "X"

While the problem of Row-Hammer was publicly disclosed in 2014 [3] [4], this problem continues to present significant vulnerability in current systems. For example, recent work shows that Row-Hammer can be done even in the presence of ECC-DIMMs [1], and the RamBleed attack [5] shows that Row Hammer can also be used to read unauthorized data from DRAM.

DRAM devices can be made robust against Row-Hammer by increasing the isolation between DRAM cells. Unfortunately, such device-level solutions has remained elusive and row hammer continues to be a problem for modern DRAM devices.

- *Moinuddin Qureshi is with the School of Computer Science, Georgia Institute of Technology, Atlanta*
  E-mail: moin@gatech.edu

2 Shortcoming of Existing Solutions

There have been several architectural solutions proposed for Row-Hammer. Unfortunately, these solutions continue to be broken by newer attacks or at newer technology generations. We discuss some of the prior solutions.

1. Increased Refresh Rate: Conventional systems use a refresh rate of 64ms. To mitigate Row-Hammer the system refresh rate can be increased by 2x-4x to do refresh at every 32ms or 16ms [4]. Unfortunately, higher refresh rate causes a significant overhead in terms of both power and performance. Furthermore, a refresh rate of 2x-4x may reduce the rate of Row-Hammer, but is not guaranteed to eliminate it.

2. Refresh Neighbor Rows Frequently: Row Hammer can be mitigated by refreshing (accessing and pre-charging) the victim rows. This can be done either by probabilistically refreshing the two neighbors (Probabilistic Row Activation or PRA [3] [4]) on an access to a given row, or by tracking the number of accesses for each row (Counter-Based Row Activation or CRA [3]) and refreshing the neighbor rows once the access count reaches a given threshold. While these schemes have low performance overhead, the key assumption of these schemes is that the Memory Controller (MC) knows the mapping or rows within the DRAM chip, and, more importantly, that the Row-Hammer threshold is known. Unfortunately, Row-Hammer threshold varies widely across bits and choosing the Row-Hammer threshold conservatively (say worst-case observed in the experimental data) may still not guarantee Row-Hammer protection, as some DIMMs may have a lower Row-Hammer threshold. Furthermore, the Row Hammer threshold can vary significantly across technology generations and over time.

3. Targeted Row-Refresh (TRR): DRAM vendors have equipped DDR4 memories with TRR, which "refreshes" the neighbors of few frequently accessed rows. The details of how the frequent rows are tracked within the DRAM is usually not publicly available. Nonetheless, recent attacks [2] have been able to cause Row-Hammer even with TRR.

3 Efficacy of ECC Against Row-Hammer

Memory systems are subjected to naturally occurring faults such as due to alpha-particle strike or chip-failures. To mitigate such errors, memory systems are often equipped with an additional chip to store the bits for the *Error Correction Code (ECC)*. Figure 2(a) shows the organization of an ECC

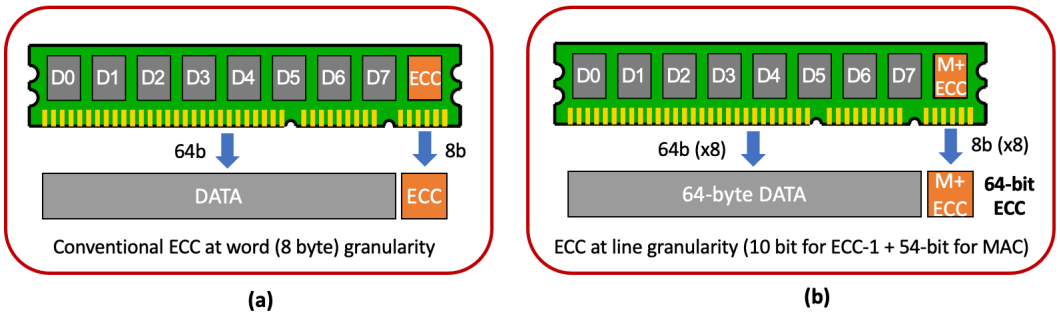

Fig. 2. ECC DIMM (a) Conventional SECDED (b) Proposed IPEM design provides ECC-1 for 64-byte line + 54-bit MAC within the same ECC space (IPEM provides correction strength similar to SECDED, as the chance of single-bit failures within multiple words of a line is negligible)

equipped DRAM. The 64-bit data bus is equipped with an additional an 8-bit ECC bus. When memory is accessed, the 8-bit ECC is used to perform a single-error-correction-double-error-detection (SECDED) on the 64-bit data. Note that, we need 7-bit for SEC and only 1 extra bit for DED, and this storage can be fit in the 8-bit of the ECC code.

While the conventional wisdom was that ECC memories would be resilient to Row-Hammer (as they can correct the bit flips caused by Row-Hammer), a recent work [1] showed that ECC-equipped memories can still be subjected to Row-Hammer. The key insight was to leverage the variation in latency due to ECC correction to figure out when a single error occurs, and then progressively cause more failures in the line. A similar strategy can be adopted if the line has the ability to correct multiple bits. Therefore, naively increasing the error-correction capability to ECC-2 or ECC-3 will not guarantee protection from Row Hammer. ECC memories can implement Chipkill, however, the recent work [1] showed that even Chipkill memories can be made to cause Row-Hammer. Therefore, extension of conventional ECC designs are insufficient at providing guaranteed mitigation of Row-Hammer for future memory systems.

4 Rethink ECC for Strong Detection

Conventional ECC codes are designed mainly for handling naturally occurring errors – such as alpha particle strike or chip failures. The main guiding principle in the development of ECC codes is to maximize the correction capability and the detection capability is simply a byproduct of the correction code. For example, for the 64-bit SECDEC code, doing single error correction (ECC-1) requires 7-bits and adding an extra bit provides double error detection (DED), hence the 8-bit SECDED code. Similarly, with Chipkill, the symbol-based code designed to correct one symbol (one chip failure) can easily provide the ability to detect the corruption in a second symbol, thus providing Single-Symbol-Correct-Double-Symbol-Detect (SSCDSD) capability.

Row-Hammer failures are not naturally occurring, but they can be orchestrated by an adversary. Thus, the rate of Row-Hammer failures is dependent on the attack patterns applied by the adversary. While there is research in trying to mitigate Row Hammer failures (and we encourage such research!), it is unlikely that we will have a solution that will guarantee elimination of Row-Hammer failures across device technologies and attack patterns.

As Row-Hammer is not just a reliability issue but a security threat, we note that it is important to at-least detect such failures. A reliable detection can avoid the system from consuming the corrupted data, which could have caused the attacker to break confidentiality and potentially take over the system. Therefore, we argue that the ECC codes in the era of Row-Hammer should be designed to provide strong detection capability (against arbitrary data corruption) while still providing a reasonable amount of correction capability. Thus, strong detection capability should be regarded as a first-class constraint in the design of future ECC codes. Such a design can provide protection against security threats that arise from adversarial memory corruption, including from Row-Hammer attacks.

Strong detection of arbitrary failures can be accomplished by *Message-Authentication Code (MAC)*, and such codes have been used in the implementation of secure memory designs. With an n bit MAC code, an arbitrary modification has only a 1-in-2^n chance of escaping detection. We propose that ECC code should be modified to put MAC as a component while still providing the correction capability for correcting naturally occurring errors. We show how such *Integrity Protected* memories can be built within the existing space of the ECC codes, for both SECDED designs and Chipkill designs, while incurring no additional storage overheads. Thus, we can get integrity protected memories, which can detect Row-Hammer failures and thus remove the security threat from such failures, at negligible overheads in terms of area, power, and performance.

5 Integrity-Protected ECC Memory (IPEM)

Conventional ECC designs form code-word at a 8-byte granularity, such that each burst of data arriving from the 64-bit bus gets protected by the 8-bit ECC bus, as shown in Figure 2(a). This is largely done for legacy reasons, as earlier designs would allow the DIMM to transmit as small an amount of memory as 1 burst (8 bytes). However, processors typically interact with the memory at the granularity of cachelines (e.g. 64 bytes), so one could do ECC computation at the line granularity instead of at the word granularity. Furthermore, modern standards for memory, such as the DDR4 standard, dictates that memory must have a minimum burst length of 8, which means when memory is accessed, it will provide at least 64 bytes. Thus, from both the processor and memory viewpoint, the minimum granularity of interaction is 64-bytes. Thus, instead of viewing the ECC space as 8-bit per 64-bit data, we can view it as 8-bytes per 64-byte of data. We can implement ECC-1 on 64-byte cache line with just 10 bits (note that the cost of the ECC increase logarithmically with the data size) and still have 54-bits left over. Our proposed design of *Integrity Protected ECC Memory (IPEM)* is shown in Figure 2(b), where the ECC code is formed at a line granularity, with ECC-1 protection, and the

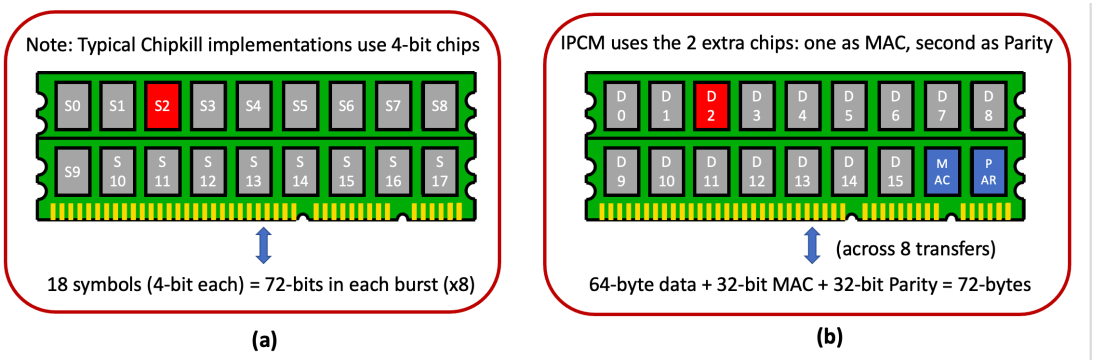

Fig. 3. Chipkill protected DRAM (a) Conventional Chipkill design using symbol-based code operating on 18-chips (S0-S17) (b) Proposed IPCM design provides single-chip-correction and strong detection by using MAC for error detection and chip-wise Parity for correction.

54-bits are used for MAC. Thus, IPEM can provide the same correction capability of conventional SECDED, but have much stronger detection. ECC-1 is computed on the 512-bit data + 54-bit MAC. When the line is accessed, the memory controller first performs ECC-1 correction (if any), and then computes the MAC of the 512-bit data, and compares it with the retrieved MAC. A mismatch signals integrity failure. Such a design will detect arbitrary corruption of data (due to Row-Hammer or any other reason) and would prevent the processor from consuming potentially malicious data. Thus, IPEM alleviates the security issues of Row-Hammer and instead makes Row-Hammer seem like a reliability problem (or at-most as a denial-of-service attack rather than attacks that can take over the system).

While the IPEM design has strong security properties, it does so without significantly affecting the correction capability of the conventional SECDED. Both IPEM and SECDED can correct 1-bit failures, IPEM does so at 64-byte granularity and SECDED does it at 8-byte granularity. The only case where SECDED can correct and IPEM cannot correct is when single bit failures happen in multiple words of a cache line, which, based on existing data on memory failures occurs with negligible probability. So, the correct capabilities are similar. However, it is possible to get mis-correction (or Silent Data Corruption (SDC)) with SECDED, when the number of failures exceed 2, however, with IPEM, given the MAC, episodes of mis-correction or SDC are virtually eliminated. Thus, IPEM offers better reliability.

Finally, one could tune IPM to provide varying level of ECC protection. For example, one could have ECC-2 for a 64-byte line with 44-bit MAC, or ECC-3 for the line with a 34-bit MAC. Thus, the increased granularity of IPEM can help with providing even higher level of correction than what is otherwise possible with conventional ECC designs.

6 Integrity-Protected Chipkill Memory

Memory systems can suffer large granularity failures such as a row failure, column failure or bank failure. Memories can be protected from such large-granularity failures using a stronger form of error-correction code called *Chipkill*, which can tolerate the failure of an entire chip. Figure 3(a) shows the operation of a conventional Chipkill design. Such memories require at least two extra chips, therefore they are formed at the granularity of 18 chips, with 16 chips for data and two for redundancy. Typically a Chipkill DIMM would be composed of 18 chips, each providing only 4-bit of data in each burst. For implementing Chipkill, the data is stored as symbol-based code, where each 4-bit represents a symbol, and the two extra symbols help with detecting and correcting symbol failure. Conventional Chipkill can provide single-chip correction and double-chip detection (SCCDCD). Similar to SECDED, the detection capability of Chipkill is simply a byproduct of the correction code and is not necessarily the key objective in designing the code.

While Chipkill can tolerate larger granularity failures, it cannot guarantee protection from Row-Hammer. For example, if there are failures in 3 chips, then Chipkill can cause mis-correction, leading to Silent-Data-Corruption and consumption of incorrect data values. Recent work [1] discusses how Chipkill designs could be overcome to cause Row-Hammer failures.

We argue that instead of limiting the detection of Chipkill to only 2 chip failures, it is important to provide strong detection capability for arbitrary failures. We show that it is possible to redesign Chipkill while ensuring strong detection, all within the same storage space of conventional Chipkill. Our proposed design of *Integrity Protected Chipkill Memory (IPCM)* is shown in Figure 3(b). Instead of storing data in symbol-based form, the data is kept in plain form in the 16 chips (D0-D15) and the two extra chips are used to store MAC and Parity. MAC is computed on the data chips and the Parity chip stores the chip-wise parity across all the data chips. On an access, the data from all the data chips are used to compute the MAC, and this MAC is compared with the stored MAC. A match indicates integrity verification. A mismatch requires correction. As we have parity, we can recover the data if we knew which data chip has failed. As we do not know the location of the failed chip, we can do an iterative search from D0 to D15, discarding one chip at a time, using parity to reconstruct that chip, recomputing MAC and matching with the stored MAC. On a match the reconstructed data is used. The location of the failed chip can be stored to reduce the iterative search for subsequent access. The IPCM design can correct a single chip failure (with very high probability) and can detect failures of arbitrary number of chips (again with high probability). IPCM virtually eliminates the rate of SDC, and thus makes Row-Hammer attacks seem more like reliability failures (Detected Unrecoverable Errors) rather than Security issues (consumption of corrupted data). IPCM does so while using the same form factor as Chipkill, while incurring negligible overheads in terms of power and performance.

Performance Consideration for IPCM: On a failure (mismatch on MAC), IPCM uses iterative correction to identify the faulty chip. Such iterative correction can incur latency of up-to 16 rounds of parity-based repair (about 2 cycles) and MAC computation (about 30 cycles), so the total latency of such correction can be in the range of few hundred cycles. While the repair overhead of IPCM may seem impractical, we point out that this overhead is incurred only in case of memory errors. This overhead can be negligible in practice for the following reasons:

1) Given the FIT rate of modern memories, most of the DIMMs are not expected to experience multi-bit failures in the lifetime of the server, so the iterative correction of IPCM would not impact the performance of such DIMMs. Furthermore, for transient failures (that happens at a low rate, say once in few months), the overhead of few hundred cycles is negligible, so the performance penalty of iterative correction is a performance issue mainly for permanent faults.
2) The full overhead of iterative correction must be incurred when the first time a chip failure is encountered. However, for subsequent corrections, we can remember the chip that failed the last time and start the iterative correction from that chip, so if the same chip failed, we incur only one extra correction and MAC computation (instead of 16). This would reduce the correction latency to about 30-40 cycles.
3) We can track the number of corrections that have occurred, and if this is above a certain threshold (within a period of time), then we can deem that a particular chip has failed, with the chip that caused the most number of faults identified as the faulty chip. We can modify IPCM to skip the first MAC check (where MAC was computed only on data chips), and directly reconstruct the data for the faulty chip using the Parity chip, and only then do a MAC check on the computed MAC. This way the reconstruction overhead of IPCM can be reduced to parity-based reconstruction (about 2 cycles) even in case of a chip failure.

7 Conclusion and Future Work

In this paper, we argue that ECC must be redesigned given the security threat of Row-Hammer (and potentially other types of memory integrity attacks in the future). ECC designs have thus far focused only on maximizing the correction capability, and the detection capability is a secondary attribute. We contend that the ECC code must be re-purposed to provide strong detection (while maintaining the correction capability), so that Row-Hammer induced memory failures can get caught and do not become security threats. We present two designs: IPEM and IPCM, that provide similar levels of correction as SECDED and Chipkill, while still providing integrity verification.

To build Row-Hammer aware ECC designs we envision that future work could consider several directions. We discuss two specific ones. First, Low-Latency MAC. Both IPEM and IPCM requires MAC calculation in the critical path which can impact performance compared to conventional ECC designs. Low-latency MAC designs (or alternative signature methods such as CRC, which are robust to reverse engineering) would be useful for deploying in future systems. Second, *ECC-Driven Adaptive Refresh.* Row Hammer can be reduced significantly by increasing the refresh rate. However, doing so all the time can cost significantly in terms of performance or power. With Adaptive Refresh we would trigger a higher refresh rate only when the ECC engine performs a correction and keep that higher refresh rate for a few minutes while other mitigation is triggered (remapping of memory or remapping of the process to another machine). For naturally occurring errors, we expect ECC correction to occur at a rate less than 1 time per month, so the overhead of higher refresh rate for a few minutes every month is a negligible cost for non-malicious ECC corrections.

References

[1] L. Cojocar, K. Razavi, C. Giuffrida, and H. Bos, "Exploiting correcting codes : On the effectiveness of ecc memory against rowhammer attacks," in *IEEE Security and Privacy*, 2019.

[2] P. Frigo, E. Vannacci, H. Hassan, V. van der Veen, O. Mutlu, C. Giuffrida, H. Bos, and K. Razavi, "TRRespass: Exploiting the Many Sides of Target Row Refresh," in *S&P*, May 2020.

[3] D. Kim, P. J. Nair, and M. K. Qureshi, "Architectural support for mitigating row hammering in dram memories," *IEEE Computer Architecture Letters*, vol. 14, no. 1, pp. 9–12, 2015.

[4] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping bits in memory without accessing them: An experimental study of dram disturbance errors," in *Proceeding of the 41st Annual International Symposium on Computer Architecuture*, ser. ISCA '14, 2014.

[5] A. Kwong, D. Genkin, D. Gruss, and Y. Yarom, "Rambleed: Reading bits in memory without accessing them," in *41st IEEE Symposium on Security and Privacy (S&P)*, 2020.