Forgetful Counters for Rowhammer Detection

Loïc France, Florent Bruguier, David Novo, Pascal Benoit

Forgetful Counters for Rowhammer Detection

Loic France, Florent Bruguier, David Novo, Pascal Benoit
LIRMM, CNRS, University of Montpellier, Montpellier, France

I. Introduction

To this day, counter-based Rowhammer mitigation proposals offer the best protection guarantee. These solutions use row activations counters to detect aggressor rows. Contrary to probabilistic solutions, counter-based proposals can offer a guaranteed protection against bit-flips, where an aggressor is always detected before the attack succeeds.

To account for periodic refreshes of the DRAM rows, the counters in a counter-based mitigation technique must be periodically reset. However, the periodic refresh of all DRAM rows does not happen at the same time in the cycle. Refreshes are spread out in the short chunks of time every t_{REFI}. As memory controller is not aware of which row is refreshed every t_{REFI}, the reset of a counter cannot be synchronised with the refresh of the row it is watching. As a result, most existing counter-based mitigation techniques reset all the counters at the same time, every t_{REFW}.

As most counters are not reset synchronously with the refresh of the row, aggressor could use this information to target rows whose counters will be reset in the middle of the attack while the row has not been refreshed, resulting in an attack not witnessed by the countermeasure. Figure 1 illustrates an attack on a row that is not refreshed synchronously with the counters reset. The refreshes on the considered row (2) are not synchronised with the reset of the counters (1). When the counters are reset, the actual activation count (ACT) of the row is higher than the value of the counter. If an attack uses this row as an aggressor, when the actual ACT count of the row exceeds ACT/2 (3), the value of the counter is below the actual count (4), as it was reset during the attack. In this case, the attack will not be detected.

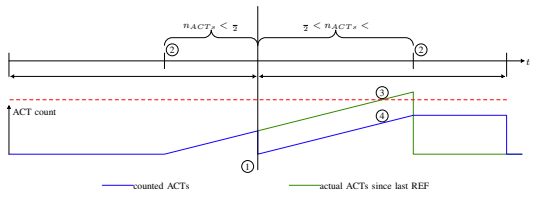

Fig. 1. Counters reset not synchronised with row refresh.

Existing mechanisms use multiple methods to take this fact into account. For example, BlockHammer [2] chooses to double the whole mechanism and alternately reset them, and Graphene [1] divides the detection threshold by 2, consequently using twice the initial number of counters.

Modern mitigation proposals are greatly affected by this unsynchronised refreshes issue, which forces them to double the number of counters.

II. Contribution

We propose a new counter-based detection mechanism that does not need a periodic reset of all its counters, and therefore is not affected by the unsynchronised refreshes.

It is based on the combination of ACT counters and ACT frequency evaluation. To simplify the demonstration, we assume that 2 ACTs will always take the same time to be executed, even if a periodic refresh command is issued in-between, *i.e.*, we will only consider the time during which the memory can be issued ACTs. The time will be considered *paused* when the memory is not accessible because of, *e.g.*, periodic refresh commands (that are issued every t_{REFI}).

In a bank, a total of W ACTs can be issued during t_{REFW}. Considering a corruption threshold T_{RH}, as the two neighbours of a victim rows can be used to corrupt it, the required number of ACTs per row to induce a bit-flip is HC_{first} = T_{RH}/2. We can deduce that a total of N_{agg} = W/HC_{first} aggressors can be used at the same time. Therefore, the mean period between two ACTs on an aggressor cannot exceed $P = N_{agg}\times$ for a successful attack. If the mean ACT period of a row is greater than P, it is not an aggressor as it cannot complete the attack within a refresh window (t_{REFW}).

To track only the potential aggressor, we can track only the rows that are activated frequently enough to be aggressors and store them in a table. We can do the following steps:

1) When a row is issued an ACT for the first time, allocate an entry in the table for it. The expiration time is set to $t + P$, where t is the current time.
2) If the row is activated again before the expiration time is reached, increase the expiration time by P. The expiration time will be set to $t_0 + n \times P$, where t_0 is the time of the first ACT, and n is the number of times it has been activated since the step 1.
3) When the expiration time is reached, the entry can be removed from the table.

Using this method, only the rows that are activated at least once every P on average will be kept in the table. Monitoring all expiration dates of the table to check for expired values would be too time-consuming. Instead of having a constant monitoring of expiration dates, new entries will first try to replace expired entries. Additionally, a periodic maintenance is put in place to periodically remove expired entries that were not replaced by new entries.

REFERENCES

[1] Yeonhong Park et al. Graphene: Strong yet lightweight row hammer protection. In *MICRO*, 2020.

[2] A Giray Yağlikçi et al. Blockhammer: Preventing rowhammer at low cost by blacklisting rapidly-accessed DRAM rows. In *HPCA*, 2021.