Improving DRAM with regards to performance, energy-efficiency and reliablity

Ton That, Hoai An

Karlsruher Institut für Technologie, Fakultät für Informatik, 76128 Karlsruhe, Germany

Abstract. Many fields of computer science are dependent upon data. Much of this data needs to be stored in devices capable of retaining the stored data for a prolonged duration. These devices need to be reliable, safe, fast, cheap and simple to produce. Such devices are Random Access Memory types like Dynamic Random Access Memory, Static Random Access Memory or Resistive Random Access Memory. In this seminar paper we will look at a technology called Copy-ROW Dynamic Random Access Memory to improve upon the aforementioned properties while being flexible and cheap to produce. The evaluation results show a significant improvement in performance, energy-efficiency as well as reliability.

Keywords: memory · performance · energy-efficiency · reliability

1 Introduction

Random Access Memory (RAM) is an elementary component in each computation system on the market. The consumer-based evaluation nowadays is mostly based on the memory capacity the RAM module provides. This metric does not represent the entire capabilities of a RAM module. Other factors that need to be accounted for are performance (e.g., refresh latency, access-time), security, reliability and durability. Many new explored RAM types try to mitigate the weaknesses of previous RAM type generations. Nonetheless the common types of RAM on the market are still Dynamic Random Access Memory (DRAM) and Static Random Access Memory (SRAM). Both RAM types are well explored till date. The important difference between those is that, DRAM can be easily replaced by the consumer while on the other side SRAM is integrated into the hardware making it a less important concern for the consumer. The goal of this seminar paper is to delve into different techniques and structures to improve DRAM and enable new possibilities regarding performance, energy-efficiency and reliability.

2 Background

This section describes the DRAM organization and its operations that are necessary to understand the inner workings of DRAM.

2.1 DRAM - Organization

A typical DRAM based system is shown in Fig. 1. In those systems the bits flow through separate Input/Output (I/O) buses from DRAM to Central Processing Unit (CPU). This flow is controlled by several memory controllers of the CPU that are connected through a DRAM channel. A DRAM channel can be connected to several modules. The DRAM module itself is ordered hierarchical, starting from DRAM rank down to DRAM subarray. The DRAM module consists of billions of DRAM cells ordered in the aforementioned hierarchy. The DRAM module itself contains several DRAM ranks which are ordered again into DRAM chips. Those DRAM chips contain several DRAM banks where the actual DRAM subarray containing the DRAM cells is present. The subarray is structured like a matrix (e.g., 128x128) [11,2,5,6]. A row of cells in the matrix is connected by a wordline which is selected by a row decoder. The columns are connected with bitlines. At the end of the bitlines, a row buffer sits to buffer the data from the activated line. The row buffer also contains the bitline sense amplifiers (BLSA) to amplify the signal [2,5]. This structure enables the DRAM to continuously access and update the stored data.

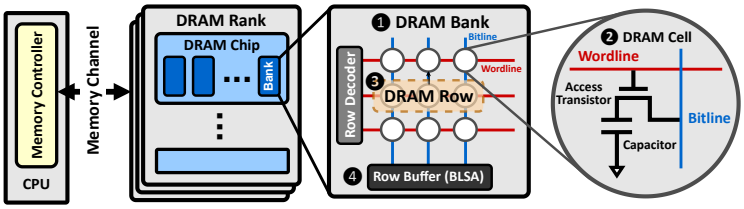

Fig. 1. DRAM module hierarchy [5]

2.2 DRAM - Internal cell functionality

A DRAM cell consists of a transistor and a capacitor as seen in Fig. 1. The DRAM cell contains a bit depending on the current charge level. Due to the nature of a capacitor and the size of the cell, the charge leaks with time and needs to be refreshed [11,2,5,4]. Periodically each row is activated, allowing the charges to travel through the bitlines towards the row buffer. For the periodic refresh, the bitlines need to be precharged to a certain level. Generally the required charge level is $V_{dd}/2$. The BLSA in the row buffer then amplifies the charge back into the cells. Each operation on a cell requires a specific amount of time until a stable state is reached, these times are called timing parameters [4].

2.3 DRAM - Operations

To communicate with the outside world, the memory controller provides the Precharge, Activate, Read, Write and Refresh operations. All these operations

are restricted by timing parameters to guarantee a correct execution. Within these operations the Refresh operation takes a special position as it is periodically called to prevent data loss due to charge leakage [2].

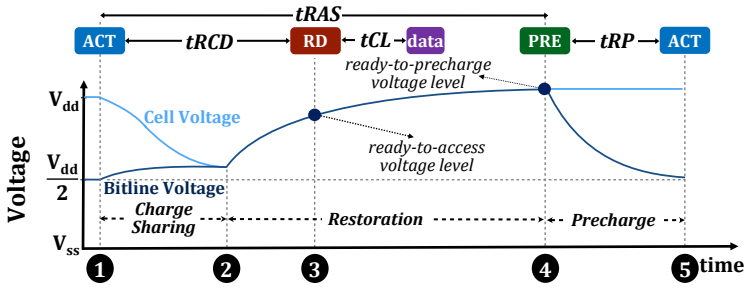

Fig. 2. DRAM operations, operation timings and current voltage during a Read [2]

2.3.1 Activate The Activate (ACT) operation applies an electrical current to the specified row address sent alongside the operation if the row was not already activated. The row decoder decodes the row address and determines the selected row. After the activation, the data contained in the cells are send to the row buffer. This operation is constrained by the timing parameter *Row Address to Column Address Delay* (tRCD) which is the required time until the data has stabilized in the row buffer. [2,5]. The operation requires the bitline to be precharged as opening the row allows the transistor charge to flow towards the bitlines. After stabilizing the charge, the BLSA detects the change in charge and restores the charge inside the transistor depending on the read change. The charge restoration is necessary as the capacitor does not contain the correct charge after the row opening [2].

2.3.2 Read The Read (RD) operation can be started after a row was activated. The column address is sent alongside this operation to specify the part inside the matrix to be retrieved. This data is then read and stored in the global row buffer. From the buffer the data then moves through a connected bus to the memory controller. The timing parameter for this operation is called *CAS latency* (tCL) and decides the required time until the data appears on the bus [2,4].

2.3.3 Write The Write (WR) operation operates similar to the ACT operation. The Write operation requires a charge restoration. Instead of just restoring the previous charge, the BLSA sends the to be written data into the capacitors. The timing parameter *Write Recovery Time* (tWR) defines the required passing time till the next operation [2,4].

2.3.4 Refresh As previously mentioned, a DRAM cell can not retain its state for a prolonged time due to the charge leakage. To ensure the correct retrieval of the data at any point in time, this refresh (REF) operation is periodically called in a specified interval. Each row is typically refreshed in a standardized time frame (e.g., 64ms) [2,5,4].

2.3.5 Precharge The Precharge (PRE) operation is necessary to close a row and prepare the bitlines for the next ACT. The timing parameters tRAS decides the time until the next PRE can be called. The time itself the PRE operations need is called *Row Precharge Time* (tRP). It allows the bitlines enough time to be charged to the required charge level [2,4].

3 Copy-Row DRAM

The Copy-Row DRAM (CROW) is a new technology proposed in the paper [2]. It tries to mitigate the bottleneck, which the regular DRAM causes, by using new mechanisms for faster, more energy-efficient and more reliable use of DRAM.

3.1 Copy-Row DRAM - Overview

CROW introduces two new components to achieve the desired properties.

The first component are *copy rows* that are, additionally to the regular rows, located in each subarray. Those can be used to copy or remap data stored in the regular rows [2].

The second component is the *CROW-table* located inside the memory controller. It tracks the occupied copy rows and its corresponding regular row [2].

As CROW uses two different types of rows, both need to be addressed separately thus resulting in another row decoder for the copy rows. This allows a new action called *multiple-row activation* (MRA) which activates copy rows and regular rows simultaneously. Two new DRAM primitives are possible due to this new action.

First, CROW can perform bulk data movement, it can copy content from a regular row to a copy row. This is achieved by activating the regular row first. After the ACT operation, the data is stored inside the buffers. The copy row is activated additionally to the regular row. The sense amplifers now redirect the charge towards the open regular row as well as the open copy row [2].

Second, CROW can perform reduced-latency DRAM access under certain conditions. If a regular row got copied beforehand into a copy row, the memory controller can activate both simultaneously [2]. Two charges are spreading inside the bitlines which results into faster charge time for the sense amplifiers. The sense amplifier can redirect the charge faster as a result thus decreasing the activation time.

3.1.1 Copy Row As mentioned previously, CROW contains two different types of rows as shown in Fig. 3. On one side the usual rows in every DRAM, denoted here as regular rows, on the other side the copy rows. These copy rows can be accessed independently as a second row decoder is used to address these rows. The occupied area of the second row decoder is much smaller than the one for the regular rows as the amount of copy rows will be kept small. Additionally to the regular row, now another address needs to be send from the memory controller. Due to the amount of copy rows, not many extra bits are necessary for the address.

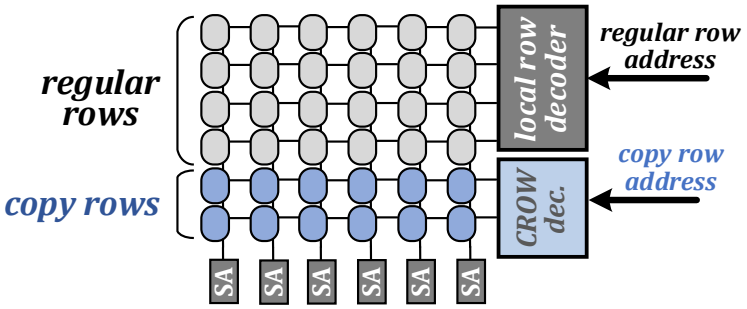

Fig. 3. CROW DRAM structure [2]

3.1.2 CROW-table The CROW-table is located in the memory controller. A rough organization of the data structure can be seen in Fig. 4. It stores data regarding the copied regular rows inside the copy rows. The CROW-table itself is n-way set associative, n is the amount of copy rows that are located in each subarray. The table is accessed by the memory controller by using the bank- and subarray-address to retrieve. The current design requires three fields for each entry. First, a valid field to check whether the copy row is valid. Second, a pointer towards the copied regular row. Third, additional information for CROW operations.

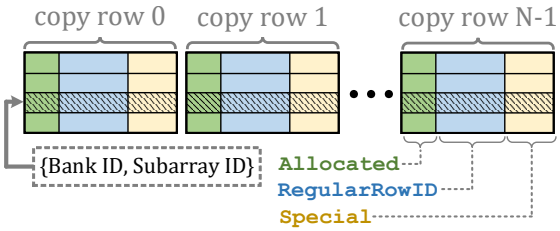

Fig. 4. CROW-table [2]

3.2 CROW - New operations

CROW introduces two new operations that exploit the new subarray structure. These operations aim to improve upon performance, energy-efficiency and reliability.

3.2.1 Activate-and-copy The Activate-and-copy operation (ACT-c) executes a copy operation from a regular row to a copy row. The operation first performs a regular row activation. After the sense amplifier is charged, the copy row is then activated as well. Due to there being two opens rows now, both will be charged with the corresponding data. For this operation the regular row address and copy row address are required. The addresses will then be supplied to the corresponding row decoder.

3.2.2 Activate-two The Activate-two operation (ACT-t) executes two activate primitives at once. Assuming the to be activated row is also existent in another copy row, the ACT-t operation activates both rows simultaneously. As this operation activates two rows at once, it requires the regular row and copy row address. A lookup into the CROW-table before ACT-c is required as the copy row might be invalid.

3.3 CROW - Improvement mechanisms

Three mechanisms are proposed in the paper [2] to improve upon the properties performance, energy-efficiency and reliability. The first strategy is called CROW-cache, it takes advantage of the MRA operation to achieve faster access time as well as less energy consumption. The second strategy is called CROW-ref and reduces refresh overhead by taking advantage of the copy rows for faster REF operations. The third strategy relies on remapping regular rows onto copy rows to counteract against rowhammer attacks.

3.3.1 CROW-cache The key idea behind the CROW-cache is to exploit the MRA operation to charge the bitlines from two locations. This results in an faster restoration of the sense amplifier, by using this fact the access towards specific rows can be increased.

Assuming N copy rows, N regular rows can be copied. The N copy rows will act as an cache and will store the most frequently accessed rows like a regular cache. The cache can be filled by using any operation systems cache-replacement-policy like First-In-First-Out. In case the cache-replacement-policy requires more bits, more bits need to be allocated for the CROW-table. The memory controller can now check on every access whether a row got copied. In case the CROW-table retrieves a miss, the regular row is getting copied to a copy row by using the newly introduced ACT-c operation. After the execution of the operation both rows contain the same data. Due to restoring the data to two different rows, we gain a slight overhead in restoration latency. The effect

on the entire is system is little as the cache-replacement-policy should provide a high hit-rate.

The second introduced operation ACT-t exploits the fact, that two rows containing the same data are existent in the DRAM. The key idea is to activate a copy row and regular row simultaneously to allow faster access. Assuming the to be access regular row already has an entry inside the CROW-table, we can execute and ACT-t. This results into faster charging of the bitlines and faster general activation time. By using this combination of operations the buffer retrieves the data faster and access-time can be speed up.

We gain another positive side-effect from activating two rows at once, an increased capacitance on the bitline. By exploiting this fact, we can relax the tRAS timing parameter by using a technique called partial restoration [10]. The key idea is to stop the row activation earlier than usual resulting into partially restored cells. The increased capacitance results into slower charge leak thus keeping the data until the next refresh. By combining the partial restoration technique with the previously mentioned techniques, we can achieve an even faster access time. Due to the partial charge state, those cells can only be accessed with ACT-t. The next section will touch upon this drawback. As previously mentioned, the access time can be decreased by using partially charged cells. There are two cases, where the the cells are not necessarily in that state.

First, the operations in between the PRE operation and the ACT operation surpass the time needed for full recharge.

Second, no further requests are made to the memory controller resulting in time to delay the PRE operation. In any other case, we need to store the information of the partial restoration state to prevent faulty accesses. This is accomplished by using the additional information bits inside the CROW-table. We use one bit to store the partial restoration state and call it isFullyRestored. This field is only false, if the time between ACT and PRE is below the the tRAS timing parameter. Due to the already existing timing information, the calculation of this field does not result into a significant overhead.

As already mentioned in previous sections, the copy rows act similar to a cache. The introduction of the partially charged states provides a new challenge regarding the eviction policy that needs to be addressed. Assume a copy row is evicted for a copy of another regular row, yet the copy row is not fully restored. The memory controller can access this information inside the CROW-table. To circumvent this issue, the memory controller issues an ACT-t operation and awaits the the regular tRAS timing parameter. After the passing of the regular timing parameter, the cell is fully restored. This trick results into a negligible overhead with the assumption of a high hit-rate.

3.3.2 CROW-ref CROW-ref is a mechanism to increase the refresh window by remapping weak regular rows to strong copy rows. This mechanism allows us to extend the worst-case timing parameter and reduce energy consumption. For the CROW-ref mechanism, a row retention profiler is necessary for identification of weak rows. The row retention profiler is run at start-up or during runtime

to analyze the weak and strong rows. The profiler tests all rows with specified procedures upon certain specifications to rate all rows. Works in the papers [1,7] find that most rows do not necessarily need worst-case refresh interval. This indicates, that using strong rows results into more efficient usage of refresh intervals. By using these results, we can calculate how many copy rows are neccessary to ensure a efficient usage of rows. The calculation for the probability of a weak row results from the counter probability for no weak cells.

$$P_{weak\_row} = 1 - (1 - P_{weak\_cell})^{N_{cells\_row}} \tag{1}$$

Using (1) and the binomial distribution, we determine the probability for more than n weak rows in a subarray.

$$P_{subarray\_n\_rows} = 1 - \sum_{k=0}^{n} \binom{N_{row}}{k} P_{weak\_row}^{k} * (1 - P_{weak\_row})^{N_{row}-k} \tag{2}$$

Under the assumption of $P_{weak\_cell} = 4 \cdot 10^{-9}$ and 512 rows per subarray with 8192 cells, we arrive at a probablity of $P_{subarray\_8\_rows} = 3.3 \cdot 10^{-11}$. This means, 8 copy rows will suffice for most objectives to cover most of the weak rows. By using the knowledge about weak and strong rows, we can remap all weak regular rows onto strong copy rows. The tracking of the copies happens with the CROW-table. After each remapping the CROW-table stores the regular row id in its corresponding field. At each memory access, the memory controller checks the CROW-table for a remapping. If the check is positive, the corresponding row is activated instead. By using the above-mentioned strategy, the worst-case timing parameter is extended due to strong rows having a longer lifetime. This results in less refresh periods as well as better energy usage. The profiling during runtime is necessary due to variable retention time (VRT) [9,8]. The VRT results into sporadic changes of retention time. To act against this behaviour, runtime profiling is necessary. The remapping strategy remains the same.

3.4 Row-hammer mitigation

The scaling nowadays result in many issues as bitlines and cells are positioned in close proximity. The electromagnetic fields surrounding each component may affect one another due to the distance leading to corrupted data. Such an attack is the rowhammer attack, the concept behind it is to consecutively activate a row to peturb nearby victim rows. Aside from reliability issues stemming from corrupted data, previous work also have shown that privileged access can be gained through this attack. CROW has the ability to mitigate this kind of attack by exploiting the copy mechanism. Several papers [3,11] already delved into the topic of row-hammer detection. By using CROW we can prevent a row from being corrupted. After detection, several ACT-c operations are executed to copy the adjacent victim rows to copy rows. By doing that, the content of the rows are saved, therefore negating the attack on the victim rows. For any following ACT operation, the corresponding copy row can be retrieved from the CROW-table inside the memory controller.

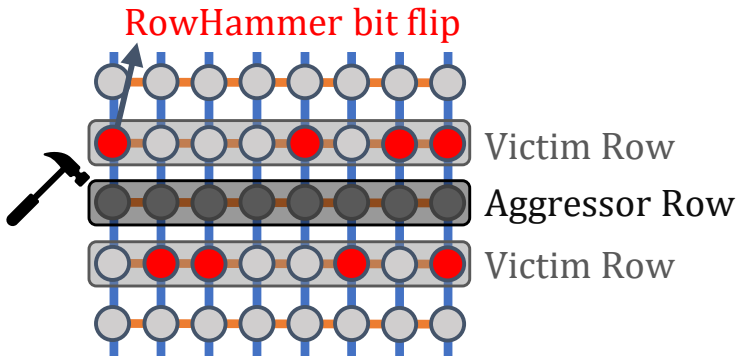

Fig. 5. Rowhammer attack [3]

4 Evaluation

Evaluation results on performance and energy-efficiency are presented in the paper [2] and will be discussed here.

4.1 CROW-cache

The evaluation for CROW-cache will be made on configurations with differing amount of copy rows. The CROW configuration will be denoted by CROW-N where N is the amount of copy rows. For the evaluation, CROW was run on several benchmark applications e.g., cactus. The quantity mentioned alongside the application refers to the misses per kilobyte instruction.

4.1.1 Single-core performance As discussed earlier, ACT-t provides an speedup due to faster access which can be seen in Fig. 6. CROW-1 already shows an average speedup compared to traditional DRAM of 5.5%, CROW-8 and CROW-256 respectively 7.1% and 7.8%. This improvement is also coupled with an more efficient energy-usage as more operations can be executed as result from the speedup. These numbers are based under the assumption of the Most Recently Used eviction policy. By just using this eviction policy, we can already see a high hit rate. This is proportional to the speedup as can be seen.

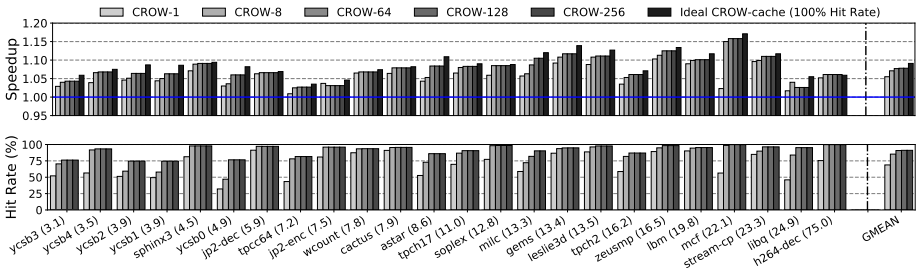

Fig. 6. Single-core performance [2]

4.1.2 Multi-core performance In Fig. 7 we can see the evaluations results for a four-core system. The red bars show the performance fluctuation between each core. The workload of each core is denoted by High (H), Medium (M) and Low (L). As can be seen, CROW-1 provides, on average, a 1.1% speedup, while CROW-8 and CROW-256 provide a 3.7% and 4.9% speedup, respectively. We can see, CROW-1 does not provide a great speedup boost in case of a multi-core performance system. Due to multiple cores accessing the same subarray, the copy row gets frequently replaced resulting into an inefficient usage of the CROW. In contrast to single-core performance, multi-core performance still achieves a significant speedup compared to the regular performance.

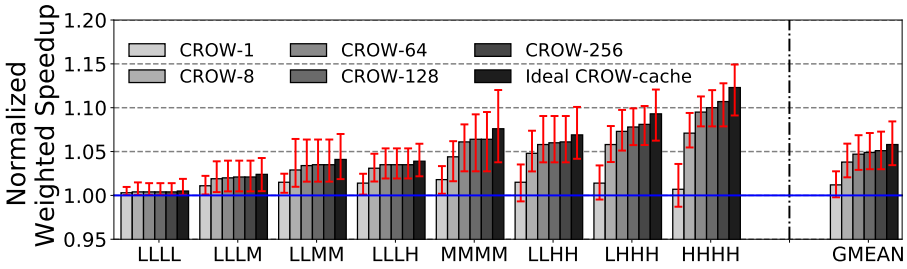

Fig. 7. Four-core performance [2]

4.2 Energy-efficiency

ACT-t and ACT-c are elementary operations for the CROW improvement strategies. Even though compared to ACT, both operations consume more energy on average, CROW-cache amortizes the energy-usage in general. This lies in more efficient usage of energy by executing operations faster. On average, the energy usage lies around 8.2% and 6.9% for single-core systems and multi-core systems respectively. It should be noted, the more operations are used, the more the energy-efficiency rises due to amortization over all operations

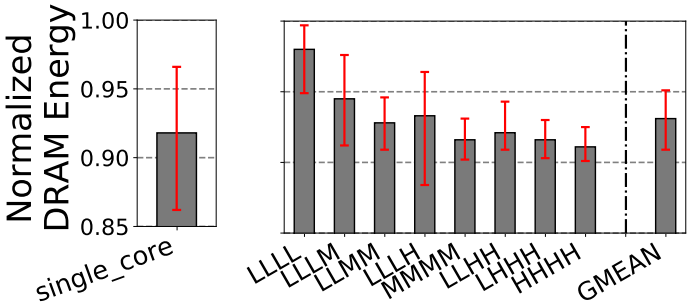

Fig. 8. CROW-cache energy consumption [2]

4.3 CROW-ref

For the CROW-ref evaluation, the refresh interval was extended to 128ms instead of the usual 64ms. CROW-ref provides benefits in the aspects performance and energy-efficiency. The interval extension results into less refreshes per cycle thus less energy used up for refreshes. As a result of less refreshes per cycle, other operations can be executed instead. Therefore a speedup in performance can be seen as well. It has to be noted, less rows are available for regular CROW-cache in case of using both strategies at once. The observation results for average energy consumption can be seen in Fig. 9.

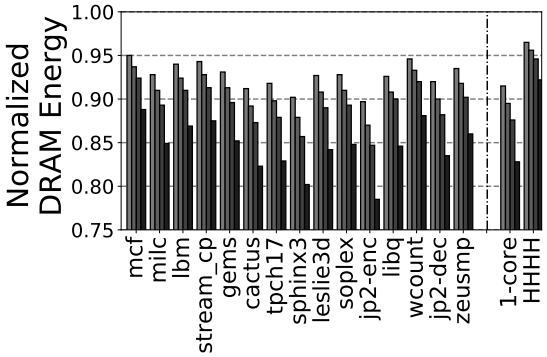

Fig. 9. CROW-ref energy consumption [2]

4.4 CROW-cache and CROW-ref

Furthermore, observations made in the paper [2] indicate a synergy in combining both mechanisms. CROW-ref does not necessarily use all copy rows and thus can be used for CROW-cache instead. This is due to the low probability of weak rows, resulting in free copy rows that can be used by CROW-cache instead. Only one extra bit is necessary to store which mechanisms is used for the copy row inside the CROW-table. Fig. 10 shows the comparisons of the two mechanisms separately as well as combined together with regard to the DRAM capacity for a single-core. We can identify, the benefits of the combination of both mechanisms is bigger than the sum of both mechanisms separately. Regarding the performance improvement, they are a result of the improved REF operations per cycle. Due to less REF operations, more operations can be executed in between, increasing the throughput of work-related operations. By decreasing the amount of REF operations, we can also see the increase that we would have gotten from CROW-ref independently. In total, we can mitigate the bottleneck by combining both mechanisms.

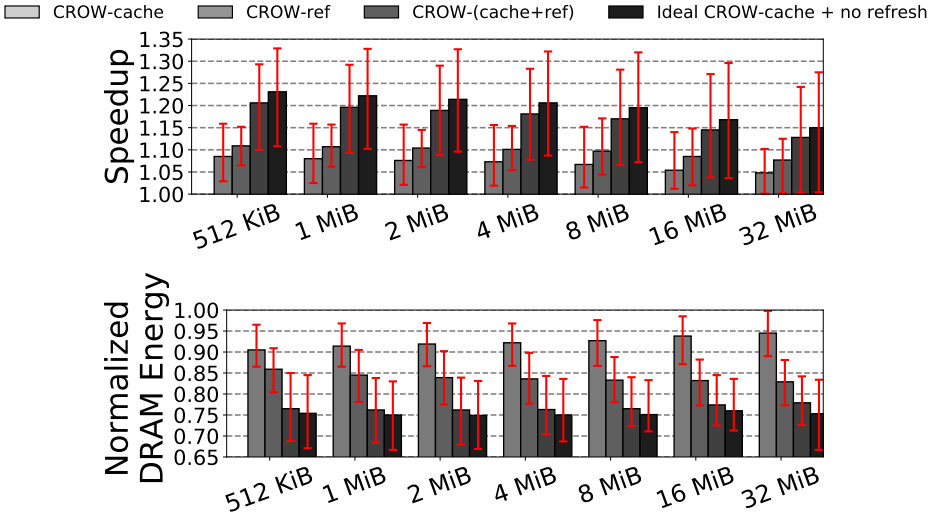

Fig. 10. CROW comparisons [2]

5 Conclusion

The paper [2] proposes a new structure called CROW that partitions the rows inside a DRAM into two types of rows which can be activated independently. By exploiting this fact, CROW introduced three strategies to improve upon performance, energy-efficiency as well as reliability. CROW-cache uses the copy rows as a kind of cache to achieve faster activation-time. CROW-ref profiles the rows of the DRAM to remap weak regular rows to strong copy rows. This achieved a reduction of refresh operation per cycle. CROW is also able to prevent rowhammer attacks by copying the victim rows into copy rows. By deploying all three mechanism improvement can be made on all respective fields.

References

1. Choi, H., Hong, D., Lee, J., Yoo, S.: Reducing dram refresh power consumption by runtime profiling of retention time and dual-row activation. Microprocessors and Microsystems **72**, 102942 (2020). https://doi.org/https://doi.org/10.1016/j.micpro.2019.102942
2. Hassan, H.: Improving dram performance, reliability, and security by rigorously understanding intrinsic dram operation (2023)
3. Hassan, H., Tugrul, Y.C., Kim, J.S., van der Veen, V., Razavi, K., Mutlu, O.: Uncovering in-dram rowhammer protection mechanisms: A new methodology, custom rowhammer patterns, and implications (2022)
4. Kim, J.S.: Improving dram performance, security, and reliability by understanding and exploiting dram timing parameter margins (2021)
5. Luo, H., Olgun, A., Yağlıkçı, A.G., Tuğrul, Y.C., Rhyner, S., Cavlak, M.B., Lindegger, J., Sadrosadati, M., Mutlu, O.: Rowpress: Amplifying read disturbance in modern dram chips (2023)
6. Olgun, A., Bostanci, F.N., Oliveira, G.F., Tugrul, Y.C., Bera, R., Yaglikci, A.G., Hassan, H., Ergin, O., Mutlu, O.: Sectored dram: An energy-efficient high-throughput and practical fine-grained dram architecture (2022)
7. Patel, M., Kim, J.S., Mutlu, O.: The reach profiler (reaper): Enabling the mitigation of dram retention failures via profiling at aggressive conditions. In: 2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA). pp. 255–268 (2017). https://doi.org/10.1145/3079856.3080242
8. Qureshi, M.K., Kim, D.H., Khan, S., Nair, P.J., Mutlu, O.: Avatar: A variable-retention-time (vrt) aware refresh for dram systems. In: 2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks. pp. 427–437 (2015). https://doi.org/10.1109/DSN.2015.58
9. Restle, Park, Lloyd: Dram variable retention time. In: 1992 International Technical Digest on Electron Devices Meeting. pp. 807–810 (1992). https://doi.org/10.1109/IEDM.1992.307481
10. Wang, Y., Tavakkol, A., Orosa, L., Ghose, S., Mansouri Ghiasi, N., Patel, M., Kim, J.S., Hassan, H., Sadrosadati, M., Mutlu, O.: Reducing dram latency via charge-level-aware look-ahead partial restoration. In: 2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). pp. 298–311 (2018). https://doi.org/10.1109/MICRO.2018.00032
11. Zhou, R., Liu, J., Ahmed, S., Kochar, N., Rakin, A.S., Angizi, S.: Threshold breaker: Can counter-based rowhammer prevention mechanisms truly safeguard dram? (2023)