Speeding Up Crossbar Resistive Memory by Exploiting In-memory Data Patterns

Wen Wen[†], Lei Zhao[§], Youtao Zhang[§], Jun Yang[†]
[†]Department of Electrical and Computer Engineering, [§]Department of Computer Science, University of Pittsburgh
[†]wew55@pitt.edu, [§]lez21@pitt.edu, [§]zhangyt@cs.pitt.edu, [†]juy9@pitt.edu

***Abstract*—Resistive Memory (ReRAM), when adopting crossbar architecture, has the smallest 4F^2 planar cell size, which is ideal for constructing dense memory with large capacity. However, crossbar cell structure suffers from large sneak leakage and IR drop on long wires. To ensure operation reliability, ReRAM writes, in particular, RESET operations, conservatively use the worst-case access latency of all cells in ReRAM arrays, which leads to significant performance degradation and dynamic energy waste.**

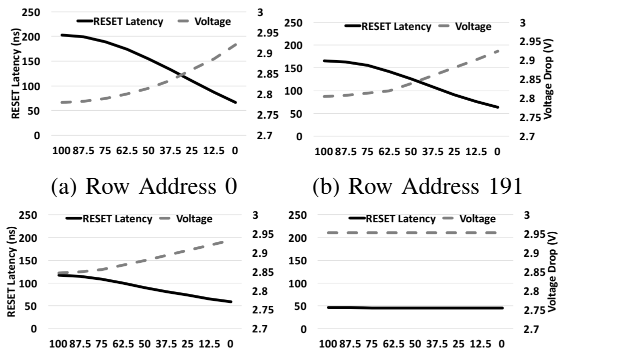
In this paper, we study the correlation between the RESET latency and the number of cells in low resistant state (LRS) along bitlines, and propose to dynamically speed up ReRAM RESET operations for the rows that have small numbers of LRS cells. We leverage the intrinsic in-memory processing capability of ReRAM crossbar and propose a low overhead runtime profiler that effectively tracks the data patterns in different bitlines. To achieve further RESET latency reduction, we employ data compression and row address dependent data layout to reduce LRS cells on bitlines. To the best of our knowledge, this paper is the first architectural innovation that exploits the bitline data patterns, e.g., the percentage of LRS cells, to speed up RESET operations in ReRAM crossbars. The experimental results show that, on average, our design improves system performance by 20.5% and 14.2%, and reduces memory dynamic energy by 15.7% and 7.6%, compared to the baseline and the state-of-the-art crossbar design.

I. Introduction And Motivation

IR Drop Issue: Studies have shown that ReRAM crossbar, even adopting diode selectors, has the currents flowing through all cells — while the sneaky currents flowing through not-selected cells are negligible, those flowing through half-selected cells are not. The sneak currents introduce large voltage drop along the wordline and bitlines, referred to as *IR drop* in the crossbar. Large IR drop not only hurts the energy efficiency, but also degrades the performances and write reliability. A recent study has shown that, due to IR drop, it takes longer time to RESET the ReRAM rows that are far away from the write driver [8]. With fast technology scaling, future ReRAM chips are expected to build upon large ReRAM crossbars, which have large wire resistance that worsens the IR drop issue.

The Correlation Between RESET Latency And Number of LRS Cells: The relationship between cell RESET switching time and IR drop on the target cell can be modeled using $t \times e^{kV_d} = C$, where t denotes cell RESET switching time; V_d denotes the voltage drop across the targeted cell; C and k are experimental fittings constants extracted from prior studies [7]. From the equation, the cell switching time is exponentially inverse correlation to the voltage drop. During RESET operation, half-selected cells do not change state and exhibit as resistive devices. Given the same voltage stress, a half-selected cell in LRS would have larger sneak current than the one in HRS.

Figure 1 summarizes the correlation among IR drop, the number of LRS cells and RESET latency for rows with different row addresses — `Row 0` and `Row 511` are the farthest and the closest rows to the write driver, respectively. In the experiments, we adopted the Verilog-A model from [6] to build and simulate a 512×512 Mat circuit model in HSPICE. From the figure, the more LRS cells there are in the bitline, the larger IR drop the sneak current brings, and the longer time the RESET operation takes. Another observation is, the impact diminishes as the row becomes closer to the write driver.



(a) Row Address 0 (b) Row Address 191

(c) Row Address 319 (d) Row Address 511

Fig. 1: Subfigures (a) to (d) show the variations of RESET latency and voltage drop at different bitline LRS cell percentages when accessing to different row address in ReRAM array.

II. Design Details

An Overview: Figure 2 presents an overview of our proposed scheme [1]. We assume that each cacheline has 512 bits, which are saved in 64 mats and each mat saves 8 bits from the cacheline. The 8 corresponding bitlines saving these 8 bits form a group. Two cachelines are mapped to use the same 8-bitline group, e..g. `a0` and `a1` use the first group, if their device addresses are separated by `K`, here `K` is a multiple of 64 depending on the number of mats, and line address interleaving. The cachelines that share the first 8-bitline group are `a0+i×K (0≤i<512)`, which are referred to as the *bitline-sharing-set* in the following discussion.

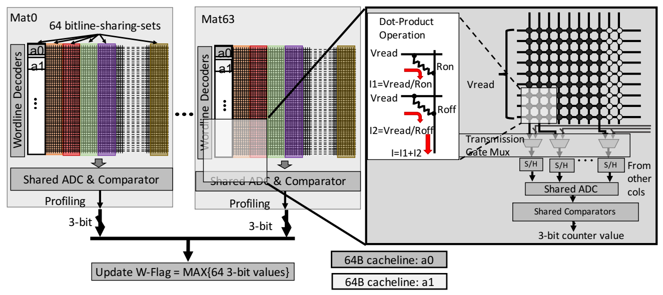
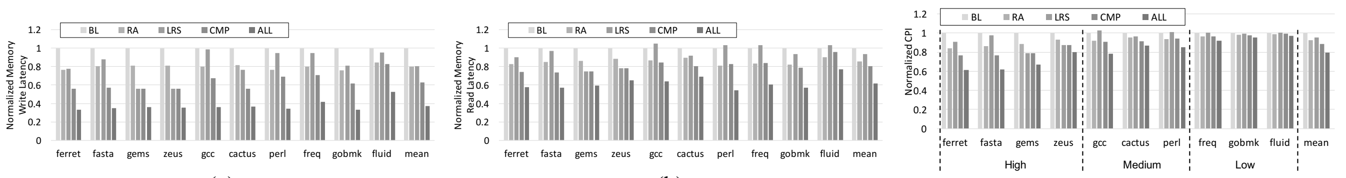


Fig. 2: An overview of the proposed architecture.

1) *Worst-case bitline flag* — `W-Flag`. We attach it to each bitline-sharing-set to record the worst case bitline. In practice, we first find the worst case bitline of each 8-bitline group in one mat, and then find the worst case from 64 mats. Instead of recording the accurate number, we divide the range [0..512] (the number of LRS cells on one bitline varies from 0 to 512) into 8 subranges such that each 3-bit flag denotes a subrange, e.g., '010' denotes subrange [128..191]. We exploit a runtime profiler

The full paper of this abstract appears in the 36th International Conference on Computer Aided Design (ICCAD), 2017, and can be viewed at http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8203787.



(a) (b) (c)

Fig. 3: The comparison of memory (a) write latency, (b) read latency, and (c) performance in CPI (cycles-per-instruction) with results normalized to `BL`. The benchmarks are categorized into High, Medium and Low memory intensity types based on RPKI and WPKI.

that periodically detects the worst case bitline in each mat and then determines the wrost case for the whole *bitline-sharing-set*.

2) *Tracking the worst-case* — `W-Cnt`. We attach it to each *bitline-sharing-set*. The counter is cleared each time when the worst-case flag is updated, that is, either after profiling update or due to `W-Cnt` overflow, which increments `W-Flag` if `W-Flag` does not saturates. At runtime, we increment the counter for each memory write that falls in the *bitline-sharing-set*, with conservatively assuming that the write always introduces one more LRS cell on the worst-case bitline within a *bitline-sharing-set*.
3) *RESET latency optimization.* We fetch its `W-Flag` and `W-Cnt` to determine the appropriate tWR time for the RESET operation.

Low Overhead Runtime Profiling: We leverage the current aggregation feature of ReRAM crossbar array [5], which has been widely exploited for accelerating in-memory computation. As illustrated in Figure 2, when there is a need to profile, all wordlines and the eight bitlines that belong to the *bitline-sharing-set* are applied with $\texttt{V}_{\texttt{read}}$; the selected eight bitlines are applied with 0V; and all other bitlines are applied with $\texttt{V}_{\texttt{read}}$ to depress sneaky currents, which is similar to dot-product operation in [5].

The currents flow through the eight bitlines are highly correlated to the number of LRS cells. The more LRS cells, the larger current will be applied to ADC and comparator circuits that are shared by all 64 8-bit read/write groups. After the analogy to digital conversion, the largest current (corresponds to the worst-case bitline in this mat) is represented as a 3-bit digital value. We set up the mapping from bitline currents to subranges before profiling. For an instance, subrange '011' corresponds to LRS cell percentage range [37.5%..50%), the bitline profiling current is $1.03mA$ if there are 255 LRS cells in one bitline. The `W-Cnt` tracks the write to the *bitline-sharing-set* after profiling. By default, the memory controller profiles the set again after 64 writes.

Reduce Bitline LRS Cells: A simple compression technique can reduce the number of LRS cells, however, we observed a direct application of it exhibits little help — the RESET latency is hardly changed since the RESET latency depends on the worst case of all 512 bitlines. Assume every cacheline in a *bitline-sharing-set* can be compressed to its half size and thus uses 256 cells. If every cacheline uses the first 256 bitlines, we would have zero LRS in the other 256 bitlines. Unfortunately, it is of little help because the worst case bitline may stay in the first 256 bitlines.

We therefore propose a row-address biased data layout to distribute extra 0s evenly to all bitlines. Given one *bitline-sharing-set* $\texttt{a0+i}\times\texttt{K (0}\leq\texttt{i<512)}$ where `a0` is the cacheline address that is mapped to the first row. When saving a compressed cacheline in, e.g., `row i`, we shift the row starting address to the right by `i` bits and then fill in the unused cells in the row with 0s.

Determine the RESET timing: At runtime, we use the physical address and the two flags `W-Flag` and `W-Cnt` to determine the appropriate tWR timing for the RESET operation as Table I shown, since row RESET latency also depends on its row index in one mat, i.e., the distance to the write drivers — given the same percentage of LRS cells along the bitlines, `row 0` and `511` have the largest and smallest RESET latencies, respectively. Therefore, we split the 512 rows in one mat to eight address subranges, and use the worst case RESET of this subgroup to write cachelines in each range.

TABLE I: The tWR (ns) for RESET Operation

LRS	Row Address Group							
Ratio	0	1	2	3	4	5	6	7
111	202.4	197.7	184.9	165.9	142.3	117.2	92.4	69.1
110	202.4	197.7	184.9	165.9	142.3	117.2	92.4	69.1
101	199	194	181.8	162.9	139.8	115	90.5	68
100	189	184.3	172.6	154.8	132.9	109	85.8	65.5
011	173.8	169.7	158.5	142	121.9	99.8	80.2	63.4
010	154.6	150.9	140.9	126	107.9	90.3	74.7	60.9
001	132.9	129.3	120.9	107.9	93.9	81.3	69.2	58.8
000	109.7	106.9	99.7	90.8	81.8	73.2	64.5	56.4

III. Evaluations

We simulate the proposed ReRAM access scheme and compare it to the conventional and state-of-the-art designs (including `BL`: conventional design with DSGB, `RA`: the state-of-the-art design [8] with row address awareness technique, `LRS`: the design only with data pattern profiling, `CMP`: `LRS` + data compression and row shifting, and `ALL`: proposed scheme with all enhancements) with benchmarks from SPEC2006 [4], PARSEC [3] and BioBench [2].

Memory Access Latency: As Figure 3a shown, on average,compared to `RA`, the proposed scheme `ALL` shows 53.5% more reduction. Similar to the write latency, the memory read latency is reduced by 38.2% for `ALL` as shown in Figure 3b. In summary, it is effective to reduce RESET latency by exploiting the number of LRS cells along bitlines.

System Performance: As shown in Figure 3c, `ALL` achieves large performance improvements on memory intensive benchmarks, e.g., `ferret` and `fasta_dna`. On average, the scheme `ALL` achieves 20.5% and 14.2% performance improvements over `BL` and `RA`, respectively.

Memory Energy Reduction: Though our proposed schemes introduce small profiling overheads, the scheme `ALL` achieves 15.7% and 7.6% dynamic energy reduction over `BL` and `RA`, respectively.

References

[1] W. Wen, *et al.* Speeding up crossbar resistive memory by exploiting in-memory data patterns. In *ICCAD*, 2017.
[2] K. Albayraktaroglu, *et al.* Biobench: A benchmark suite of bioinformatics applications. In *ISPASS*, 2005.
[3] C. Bienia and K. Li. Parsec 2.0: A new benchmark suite for chip-multiprocessors. In *Proceedings of the 5th Annual Workshop on Modeling, Benchmarking and Simulation*, 2009.
[4] J. L. Henning. Spec cpu2006 benchmark descriptions. *ACM SIGARCH Computer Architecture News*, 2006.
[5] M. Hu, *et al.* Dot-product engine for neuromorphic computing: programming 1t1m crossbar to accelerate matrix-vector multiplication. In *DAC*, 2016.
[6] Z. Jiang, *et al.* Verilog-a compact model for oxide-based resistive random access memory (rram). In *SISPAD*, 2014.
[7] C. Xu, *et al.* Overcoming the challenges of crossbar resistive memory architectures. In *HPCA*, 2015.
[8] H. Zhang, *et al.* Leader: Accelerating reram-based main memory by leveraging access latency discrepancy in crossbar arrays. In *DATE*, 2016.