PBA: Percentile-Based Level Allocation for Multiple-Bits-Per-Cell RRAM

Anjiang Wei, Akash Levy, Pu (Luke) Yi, Robert M. Radway, Priyanka Raina, Subhasish Mitra, Sara Achour
Stanford University, CA, USA
Email: {anjiang,akashl,lukeyi,radway,praina,subh,sachour}@stanford.edu

***Abstract*—Recently, researchers have demonstrated multiple-bits-per-cell (MBPC) data storage using resistive random access memory (RRAM) device technologies. In MBPC storage, a *level allocation algorithm* identifies a *level allocation* that maps resistance ranges to bit combinations. State-of-the-art level allocation algorithms, such as sigma-based allocation (SBA), fit cell characterization data to parameterized distributions and then use distribution parameters (i.e., programmed resistance standard deviation σ) to find level allocations. However, from the datasets we collected, the data points do not actually conform to the chosen distribution, and therefore the real-world analog behaviors are poorly approximated by the parameterized distribution-based approach. We present PBA, a percentile-based level allocation algorithm that computes level allocations directly from characterization data. We show that PBA level allocations have 30%-71% lower bit-error rates and 22%-41% lower ECC storage overheads than SBA on three fabricated RRAM storage arrays.**

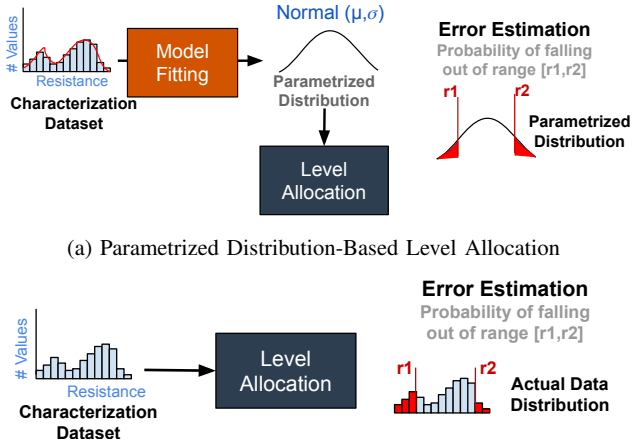
***Index Terms*—Resistive RAM (RRAM), Multiple-Bits-Per-Cell, Level Allocation, Hardware Modeling, Data Storage**

I. Introduction

There are several resistive memory technologies, including phase-change memories (PCM), resistive random access memories (RRAM) and conductive bridging random access memories (CBRAM), that offer a variety of performance, density, energy, and fabrication benefits [1]–[5] over existing non-volatile and volatile memories. These properties make these emerging memory technologies promising candidates for embedded applications [6], dense storage, and monolithic 3D integration [7], [8]. This work focuses on RRAM, one such promising memory technology.

RRAM supports both binary and multiple-bits-per-cell data storage schemes. In binary storage, bits are persistently stored by partitioning the resistance range of the RRAM cell into a high-resistance state (HRS - e.g., storing a "0") and low-resistance state (LRS - e.g., storing a "1"). Similarly, digital multiple-bits-per-cell (MBPC) data storage approaches promise to deliver higher storage densities for embedded digital systems. For example, in RRAM, 2-4 bits-per-RRAM-cell storage approaches have been demonstrated [3], [9]–[14]. To realize MBPC storage, a *level allocation*, which maps bit combinations to resistance ranges in a memory cell, is required. Typically, the level allocation is automatically derived with a *level allocation algorithm*, which partitions the memory array's resistance range into levels. A *data corruption* occurs when the resistance value read out of a memory cell corresponds to a level and binary combination that differs from the level and binary combination originally written into that cell.

Level allocation algorithms consider several analog effects when identifying a good level allocation for a given memory technology. RRAM devices suffer from various non-idealities, including process variation, which causes RRAM cell-to-cell, array-to-array, and wafer-to-wafer variations, and relaxation effects, which cause the stored resistance to change over time [3], [6], [15]–[20]. In addition, the read circuitry in the RRAM array can introduce errors due to insufficient precision (i.e., insufficient read margin) and thermal and voltage noise. These non-idealities can induce substantial data corruption if not managed. To consider these analog effects, level allocation algorithms typically work with characterization datasets that capture the effects of the read circuits, RRAM relaxation, and process variation to ensure the identified level allocations will result in low error across cells, arrays, and wafers. This process can also be performed during die test for die-optimized level allocation. The algorithm uses the collected characterization data to estimate the error of level allocations and to ultimately guide the search.



(a) Parametrized Distribution-Based Level Allocation

(b) Our Approach: Percentile-Based Level Allocation

Figure 1: Comparison between our approach and parameterized distribution-base level allocation.

A. Parametrized Distribution-Based Level Allocation

State-of-the-art level allocation algorithms, such as sigma-based allocation (SBA), construct a model of analog behavior by fitting a common, parameterized distribution (e.g., normal or normal/lognormal depending on resistance range) to the collected characterization data [3], [9], [12], [13]. Once fit to the data, the distributional parameters and probability density function are used to identify quality level allocations. These level allocation algorithms are often carefully architected to intelligently use particular distributional parameters (e.g., standard deviation σ) to find low-error level allocations specific to the memory technology.[1] As existing level allocation algorithms work with parameterized distributions, ***the combination of all empirically observed analog behaviors (i.e., read noise, relaxation, and process variation) are approximated with a single distribution. This approximation potentially eliminates or misrepresents analog***

[1] In SBA, the variance is a function of the written resistance and is computed from an empirically derived resistance-variance curve.

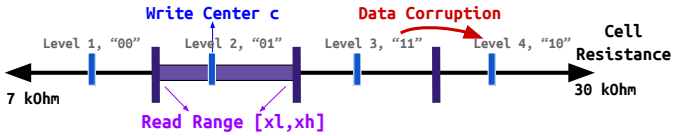


Figure 2: Level allocation for 2 bits-per-cell RRAM.

behaviors present in the characterization data that do not follow the assumed distribution.

For example, SBA (i.e., Sigma-Based Level Allocation) [3], [9], [12]–[14], a state-of-the-art level allocation algorithm for RRAM, models RRAM resistances with parameterized normal [3], or normal/lognormal [9] distributions, using the distribution's mean and standard deviation to derive level allocations. However, when a statistical test for normality is applied to the RRAM characterization data used for level allocation, more than 90% of the resistance data distributions are non-normal (Section VII). Therefore, RRAM level allocation algorithms that assume normality, such as SBA, work with an imprecise view of the memory array's analog behavior to construct level allocations. While, in practice, the parametrized normal distribution captures enough behavior to identify a reasonable level allocation, the identified level allocation may not fully exploit the capabilities of the memory technology.

B. PBA*: Percentile-Based Level Allocation*

We present **PBA**[2], a new level allocation algorithm that directly calculates percentiles over the underlying device characterization data to estimate error rather than fitting a parameterized model. Because PBA computes level allocations directly from data, it captures analog behaviors present in the data that cannot be easily modeled within a single parametrized distribution, such as a normal distribution. We also provide data collection guidelines for constructing PBA-compatible characterization datasets that produce good level allocations.

We evaluate PBA's level allocations on three different RRAM storage arrays fabricated using the same RRAM process technology [21]. We compare PBA with state-of-the-art parameterized-distribution-based algorithm SBA [3], [9], and find:

- **PBA *outperforms* SBA *on all tested RRAM storage arrays***: PBA delivers 30%-71% reduction in bit error rates (BER), and 22%-41% reduction in error correcting code (ECC) storage overheads.
- PBA's BER improvements become negligible when configured to assume the same parameterized distribution as SBA, which suggests that **PBA*'s direct use of characterization data to find good level allocations is critical to its success***.
- **PBA *outperforms* SBA *even when provided with reduced and multi-chip datasets***: PBA produces 1%-43% lower BERs than SBA on small datasets, 17%-29% lower BERs on datasets collected from multiple chips, and 15%-28% lower BERs on datasets collected from the non-target chip.

II. Background

Level Allocation. Figure 2 presents an example 2-bits-per-cell (2 BPC) *level allocation* that captures the mapping between a cell's resistance and stored bit combinations. Each bit combination must uniquely map to a level (e.g., "00", "01", "11", "10" are mapped to levels 1, 2, 3, 4 respectively). Each level is defined by a write center c (shown in blue in Figure 2) and a read range $[xl, xh]$ (shown in purple). Similar notions also apply to 3 BPC level allocation. Next, we will explain the write centers and read ranges in detail.

Write Centers. Given a level i to write to a cell j, the write algorithm will tune the resistance of cell j to match the target level i's write center c_i within some write tolerance. The write tolerance (z) determines how closely the write algorithm needs to tune a given cell's resistance to match the target write center ($c_i \pm z$). RRAM write algorithms typically tune each cell's resistance using a program-and-verify approach where the cell's resistance is iteratively adjusted and then read until it is within the tolerance [3]. With program-and-verify algorithms, larger write tolerances result in faster write operations but may accrue higher error rates. Typically, an expert hard-codes the write tolerance to the smallest value that still delivers reasonable write speeds. The effect of write failures is captured during data collection (see Section III).

All writes to RRAM target resistances must fall within the overall cell resistance window (e.g., $[7k\Omega, 30k\Omega]$ in Figure 2). Any writes that target resistances outside this window may permanently damage cells in the memory array [13] and further degrade the hardware. This requirement is enforced during chip characterization and regular operation of the hardware.

Read Ranges. In a current-mode read scheme, to read the value from a memory cell, the memory control logic applies a known small (i.e., below the level that programs the cell) voltage to the cell and then reads the outgoing current to identify the cell's resistance $\hat{r}$. The control logic determines the level i with a read range $[xl_i, xh_i]$ that contains $\hat{r}$. In the above example, any read resistance $\hat{r}$ contained within the read range of level 2 (between purple lines) encodes the bit value `01`. The read range is larger than the write range to account for potential resistance changes after a write. The read-out circuit noise may produce a resistance measurement $\hat{r} \neq r$ where r is the cell's true resistance. This read noise is accounted for during data collection (see Section III).

Data Corruptions. In multiple-bits-per-cell storage, a data corruption, or *error*, occurs when the data is written to some level i and later read as another level j. These errors occur because of unexpected conformational changes in the memory due to temperature and the applied write voltage, fabrication variations between cells, and non-idealities in the read circuitry [2], [22]–[24]. In Figure 2, the error (red arrow) illustrates a case where the value `11` is written to a cell, and the value `10` is later read from the same cell – here, the cell experiences a data corruption which induces a bit flip. Here, the bit flip error occurs because resistance is set to the write range of level 3 and ends up at the read range of level 4. To ensure an error to an adjacent level only triggers a single bit flip, practitioners have used binary data encodings such as gray codes to map bit sequences to levels [25].

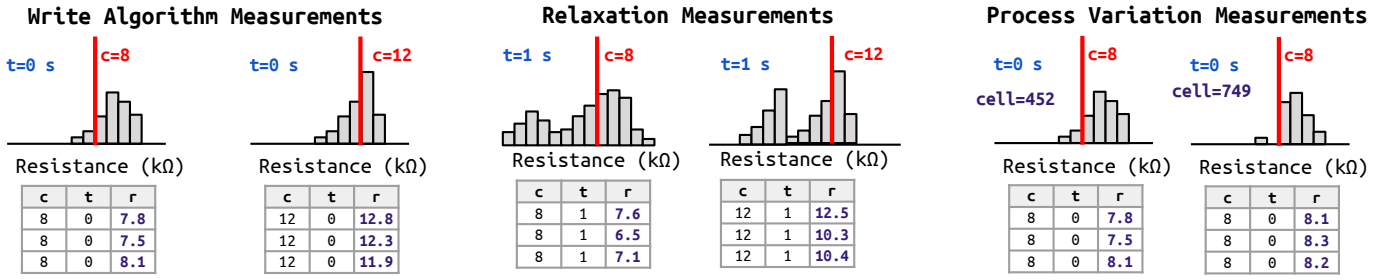
III. PBA Data Provider and Data Collection

We next present the PBA data provider, which provides an interface for querying resistance distributions backed by RRAM characterization data. The data provider returns datasets that capture the resistance distribution after a write operation to resistance c has been completed, and t seconds have elapsed:

$$\texttt{data-prov}(c, t) \rightarrow R = [r_0, r_1, ...]$$

The returned resistance data is used to estimate the error probability of the candidate level. Depending on how the characterization data

[2]https://github.com/Anjiang-Wei/PBA



c	t	r
8	0	7.8
8	0	7.5
8	0	8.1

c	t	r
12	0	12.8
12	0	12.3
12	0	11.9

c	t	r
8	1	7.6
8	1	6.5
8	1	7.1

c	t	r
12	1	12.5
12	1	10.3
12	1	10.4

c	t	r
8	0	7.8
8	0	7.5
8	0	8.1

c	t	r
8	0	8.1
8	0	8.3
8	0	8.2

Figure 3: Example experiments for characterizing write algorithm behavior, relaxation, and cell-to-cell process variations. In each experiment, multiple measurements are taken (tables) to characterize the resistance distribution (bar plot) after a write to resistance c (red line) is performed and t seconds (blue text) have elapsed. The r column in the table presents the measured resistances.

is collected, the returned resistance dataset can capture the effects of relaxation, process variation, and errors resulting from insufficient read margin.

A. Data Collection

The PBA data provider works with characterization datasets containing collections of measurements. Each measurement $\langle c, t, r \rangle$ is collected by writing a resistance c to a cell and then measuring the resistance r of the cell after t seconds have elapsed. Only write resistances within the resistance window of the RRAM storage array can be used for characterization. Characterization datasets that follow this basic structure can be analyzed by the data provider and used by the PBA level allocation algorithm. Figure 3 presents measurement strategies to capture RRAM non-idealities; all of these measurements should be performed with multiple write centers to capture dependences on the written resistance:

- **Write Algorithm and Read Noise.** The write algorithm behavior is characterized by measuring the resistance r immediately after a write to c. If the resistance measurement is performed using the storage array's read circuitry, these measurements also capture the effect of read noise on the read resistance.
- **Relaxation.** The effect of relaxation is captured by measuring a given cell's resistance over time after writing a resistance c.
- **Process variation.** The effect of process variation is captured by repeating a measurement with the same write center c and time t across different cells or storage arrays.

Best Practices. To maximize the chances of finding a good level allocation, PBA works best with datasets that meet the following criteria:

- To ensure a sufficiently large search space, the dataset should contain many distinct write resistances c.
- Enough relaxation measurements are collected with the time-to-read value that will be used for level allocation.
- To ensure error can be estimated with sufficient accuracy, the dataset should contain enough measurements for a given c and t to compute percentiles with a sufficiently fine granularity.
- The RRAM write algorithm is configured to use a fixed write tolerance for all write operations. This includes limitations on write retries (e.g., to keep write times to a reasonable value).

IV. Level Allocation Algorithm

Given a characterization dataset and the number of levels to fit in the resistive memory cell, n, PBA produces as output a n-level allocation scheme that minimizes the probability that an error occurs. An n-level allocation L is made up of a sequence of n levels

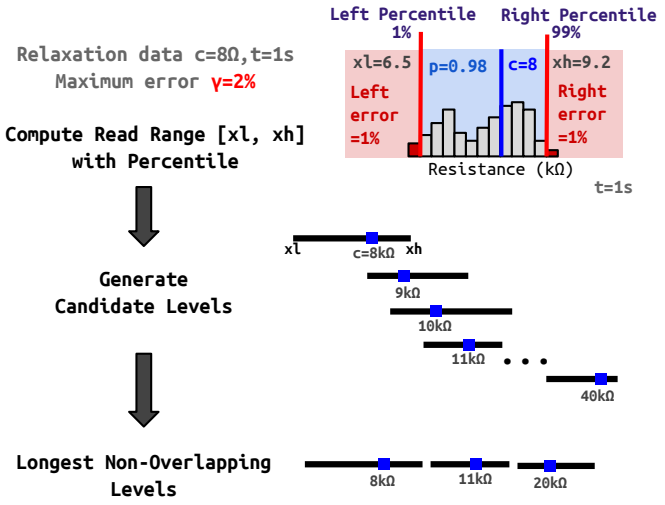


Figure 4: Example execution of level allocation algorithm with a maximum error $\gamma = 2\%$. For each write center, the algorithm constructs each candidate level by computing the read range $[xl, xh]$ with maximum error γ. The returned level allocation is the longest non-overlapping sequence of levels in the candidate level set.

$\{l_1, ..., l_n\}$. Each level $l_i = \langle c_i, xl_i, xh_i \rangle$ in the allocation is defined by a write center c_i, and a read interval lower bound xl_i and a read interval upper bound xh_i. The write algorithm's write tolerance is a fixed value that is set by the hardware designer.

A level allocation is valid if no two levels have overlapping read ranges. Specifically, for any distinct i, j in a valid n-level allocation, $[xl_i, xh_i]$ should not overlap with $[xl_j, xh_j]$. This property ensures that each resistance value maps to at most one level.

A. Intuition for Algorithm Design

Given a write center c_i and time-to-read t, we can construct a read range $[xl_i, xh_i]$ with an error probability γ by computing percentiles over the resistances returned by the data provider. The γ error probability is the maximum acceptable probability of error for the target level. For example, if we want to produce a read range with a maximum error probability of $\gamma = 2\%$, then we set xl_i and xh_i to the resistances at the 1% and 99% percentiles of the dataset (as shown in Figure 4). With this read range, 2% of the resistance dataset lies outside the read range, so the probability that a write to c_i results in a read outside the read range $[xl_i, xh_i]$ is 2%. The algorithm uses

this basic insight to compute read ranges from write centers and the maximum error probability γ. This procedure is performed for each candidate write center c_i to form a set of candidate levels for the level allocation algorithm to use.

Given the number of levels to allocate n, the algorithm still needs to (1) find the lowest error probability γ that can be used for the allocation, (2) find a set of n non-overlapping levels, where each level has an error probability γ. These goals ensure the level allocation is valid and finds the densest level allocation within its search space with the lowest error. To find the smallest γ, the algorithm searches over linearly spaced γ values and finds the densest level allocation for each γ value. The level allocations are computed by finding the longest non-overlapping subset of candidate levels with error probability γ; this ensures no two levels overlap.

Basic Algorithm. Given n levels to allocate, the algorithm finds the smallest γ value that produces an n-level allocation. The algorithm computes a level allocation for each γ value by first computing a set of candidate levels with error probability γ from the characterization dataset's write centers and then selecting the longest non-overlapping subset of candidate levels. The algorithm performs a binary search over γ values to expedite the search procedure. A binary search can be used since the number of levels in the level allocation is monotonic with respect to γ. Intuitively, larger γ values result in narrower read ranges and more non-overlapping levels, so the number of levels increases with increasing γ values.

B. Algorithm Description

Algorithm 1 presents the level allocation algorithm deployed by PBA. The algorithm tries to identify the level allocation with the lowest error probability that contains n levels. The algorithm accepts as input the target number of levels n and the error granularity ϵ. The algorithm also needs to know how we collect the dataset – a list of write centers C and the relaxation time t from the characterization dataset are both provided. The list of write centers C includes all the write centers c_i in the dataset so that the algorithm may query the data provider interface given c_i and t.

Algorithm 1: PBA Level Allocation Algorithm

```
Input:
  n                                  # Number of levels to be allocated
  ϵ                                  # Minimum granularity of error
Define:
  C                                  # List of write centers (from dataset)
  t                                  # Relaxation time (from dataset)
1  Function LevelAlloc(n, ϵ):
2      # The following loop can be binary search
3      for γ ∈ [0, ϵ, 2ϵ, ..., 1] do
4          Candidates = CandidateGen(γ)
5          Result = FindNonOverlap(Candidates)
6          if Result.length == n then
7              return Result
8  Function CandidateGen(γ):
9      Candidates = []                # Candidate levels for all write ranges
10     for c in C do
11         R = data-prov(c, t)        # List of resistance data points
12         xl = Percentile(R, 1/2 · γ)            # Lower bound
13         xh = Percentile(R, 1 − 1/2 · γ)        # Upper bound
14         assert Probability(R ∈ [xl, xh]) ≤ 1 − γ
15         Candidates.append(⟨c, xl, xh⟩)
16     return Candidates
17 Function Percentile(R, perc):
18     Rsorted = sort(R)              # Sort list of points in increasing order
19     index = perc · size(R)
20     return Rsorted[index]
21 Function FindNonOverlap(Cand):
22     Result = []                                # Non-overlapping levels
23     SortCand = sort(Cand, key = xh)            # Sort by xh
24     UpperBound = 0
25     for ⟨c, xl, xh⟩ ∈ SortCand do
26         if xl ≥ UpperBound then
27             Result.append(⟨c, xl, xh⟩)
28             UpperBound = xh                    # Update the bound
29     return Result
```

Main Algorithm (`LevelAlloc`)
The function `LevelAlloc` defines the entry point of the algorithm – it performs a binary search for the maximum error probability γ over the range of $[0, 1]$ at the granularity of ϵ. The algorithm terminates if it finds a satisfying level allocation of n levels. PBA generates its level allocation in two steps: it first constructs a set of all candidates with the maximum error γ by invoking `CandidateGen` (line 4), and then it selects the maximum number of non-overlapping levels from those candidates (line 5) by invoking another function `FindNonOverlap`.

Candidate Level Generation (`CandidateGen`)
The function returns a collection of candidate levels $\langle c_i, xl_i, xh_i \rangle$, all of which have errors less than or equal to the maximum error γ. To generate those candidates, the algorithm iterates over all the write centers c in the parameter space C, then selects xl and xh values at the $(\frac{1}{2}\gamma) \times 100\%$ percentile and the $(1 - \frac{1}{2}\gamma) \times 100\%$ percentile of the relaxation data points respectively (line 12-13). Therefore, all the generated candidates will conform to the maximum error (line 14). To compute the percentile (`Percentile` defined in line 17), we just need to sort the array of data points in the increasing order, and compute the index by multiplying the percentage parameter passed in ($perc$) with the size of the array, and finally return the element in the sorted array with the computed index.

Level Allocation Construction (`FindNonOverlap`)
PBA constructs the densest level allocation given a list of candidates ($Cand$). The candidates are a list of $\langle c, xl, xh \rangle$ tuples, and their $[xl, xh]$ may overlap with each other. This function is guaranteed to return a level allocation with the largest number of non-overlapping levels, which we will prove later. `FindNonOverlap` first sorts all the generated candidates by their upper read resistance xh (line 23) to produce a sorted list of levels $SortCand$. Then the algorithm constructs the optimal level allocation by iteratively adding the candidate level that does not overlap with the previously selected level's xh. In the iteration over all the candidates, the function appends the non-overlapping level into the $Result$ (consisting of a list of the tuples $\langle c, xl, xh \rangle$) and maintains the upper bound of current resistance range $UpperBound$ (lines 27-28). The function returns $Result$ after all the sorted levels have been processed, all of which have read ranges $[xl, xh]$ that do not overlap with one another.

C. Optimality Proof

We prove that the function `FindNonOverlap` always selects a level allocation that contains the maximum number of levels, given a list of candidate levels. We prove that sorting by xh and then iteratively selecting the first non-overlapping level finds the greatest number of non-overlapping levels and produces the densest level allocation.

Proof. Given a list of levels LS sorted by increasing upper read resistance, the algorithm produces a greedy level allocation $L = LS[x_1], ..., LS[x_k]$ that selects the levels at indices $\Sigma = \{x_1, ..., x_k\}$ from LS. Let $\Sigma' = \{y_1, ..., y_m\}$ be another set of indices that selects m levels from LS and orders these levels by their upper read resistances. Let's further state that Σ' is better than Σ ($m > k$).

We next show Σ' does not exist. This proof uses the notation $lr(x_i)$ and $ur(x_i)$ to denote the lower and upper read resistance for the level in LS at index x_i. We first prove by induction that for all $i \leq k$, $ur(x_i) \leq ur(y_i)$:

Base case: $m = 2, k = 1$, our algorithm first selects the level with the lowest upper read range, so $ur(x_1) \leq ur(y_1)$ must hold.

Induction case: For $i > 1$, if the statement holds for $i - 1$, it is true for i. The induction hypothesis states that $ur(x_{i-1}) \leq ur(y_{i-1})$, so any level that does not overlap with the optimal solution Σ' will also not overlap with our greedy solution Σ. Therefore, $ur(x_i) \leq ur(y_i)$ holds.

After proving that $ur(x_k) \leq ur(y_k)$, we show by contradiction that the better solution Σ' ($m > k$) cannot exist. If $m > k$, then the $(k+1)^{th}$ level must satisfy $lr(y_{k+1}) \geq ur(y_k)$ since Σ' encodes a well-formed allocation. Because $ur(x_k) \leq ur(y_k)$ (induction proof), the level at y_{k+1} is compatible with all levels in L, so our greedy algorithm would have added y_{k+1} to Σ. This is a contradiction, so L (produced by our algorithm) must be one of the optimal solutions. □

D. Implementation

The core algorithm is implemented within one hundred lines of Python code. One implementation detail is that the resulting non-overlapping levels may have resistance gaps between adjacent read ranges. We close the gaps between adjacent read ranges by extending them to meet in the middle so that no undefined resistance region exists across the whole resistance range.

V. Experimental Setup and Evaluation

Hardware Platforms. We evaluate PBA on three fabricated RRAM storage array instances with two different array designs using different write algorithms to configure resistance (see Table I). All three storage arrays are fabricated in a 40 nm silicon CMOS process using foundry-provided RRAM bit cells. The two Ember chips [27] incorporate an on-board write-verify algorithm for writing resistances within a given write range, and a 6 bit, 64 level ADC for reading the resistance of memory cells. Sapiens [26] uses off-chip measurement hardware to set and read resistances, thereby supporting a larger space of level allocations.

Data Collection. The last 4 columns of Table I summarize the characterization datasets that were collected by exercising each storage array. These datasets are used by the level allocation algorithms to generate level allocations. For Sapiens, we exercise 200 cells with 32 write centers across the entire resistance window of $[7.8k\Omega, 40k\Omega]$. We use a fixed value of 50Ω as the write tolerance. We measure at 0 seconds (for write measurements) and 1 second (for relaxation measurements). For the Ember chips, we exercise 16384 cells at 64 write centers and one write tolerance of 2 levels, and measure at 0 seconds and 1 second. For Sapiens, because the write center can be set to any resistance, we linearly interpolate over this dataset to fill in data for intermediate write center resistances and use this augmented dataset for level allocation.

Analysis. We use each level allocation algorithm to produce 4-level (2 BPC) and 8-level (3 BPC) allocations for the characterized RRAM storage array. The PBA implementation uses a minimum error granularity ϵ of 10^{-6}. For Sapiens, PBA is instantiated to search over 500 uniformly spaced write resistances between $[8, 40]$ $k\Omega$, with a write tolerance of 25 Ω. For the Ember chips, PBA is instantiated to search over 64 ADC resistance values, and a write tolerance of 2 ADC resistance values.

We empirically measure the observed error rates on the fabricated hardware. We then test 200-16384 random cells in the RRAM storage array, depending on the array type, with the produced level allocations to compute the empirically observed transition probability matrix. We then compute the bit error rate (BER) and the ECC overhead from the transition probability matrix. We use Gray coding [25] to assign bits to levels, and use the following formula for the BER:

$$BER = 100\% \times \frac{\#bit\ flips}{\#total\ bits}$$

Storage Overhead. We also evaluate the storage overhead associated with each level allocation in conjunction with an ECC scheme to attain a 10^{-14} unrecoverable bit error rate. The lowest-overhead Reed-Solomon, BCH, or Hamming code with a codeword size of at most 12 bits is used for this analysis. Together, these quantities capture the error rates associated with different level allocations and the additional storage needed to use the RRAM array as a reliable storage medium.

A. Comparison with Sigma-Based Allocation (SBA)

We compare PBA level allocations against the state-of-the-art SBA level allocation [9]. The SBA algorithm has been used to identify level allocations for a number of multi-level cell RRAM storage arrays [3], [9], [12]–[14], and models the resistance distribution with a parametrized normal or log-normal distribution. We implement SBA, and validate our implementation by reproducing the level allocation results shown in Figure 9 of the paper [9]. We then configure SBA to take as input the data that we collect, and adopt a normal distribution for data modeling (which was in prior work targeting the same fabrication process [3]) to produce level allocations for Sapiens and Ember chips.

B. Baseline: Sigma-Based Allocation

SBA is also a level allocation algorithm that tries to find an n-level allocation with the smallest error. For each maximum error probability, γ, SBA constructs a set of candidate levels with that error probability from the provided write centers, and then greedily finds a sequence of non-overlapping levels from the candidate level set. The overall algorithmic structure of SBA is largely the same as PBA's algorithm, with two key differences:

1) SBA fits data to a normal distribution, and then uses the normal distribution's cumulative distribution function (CDF) to compute the read range for each write center. The normal distribution's parameters are derived by computing the mean and standard deviation from the resistance dataset for each write center c. In contrast, PBA computes directly computes the percentile over the dataset.
2) SBA greedily allocates non-overlapping levels differently than PBA, and as a result may produce sub-optimal level allocations (Figure 5). SBA sorts levels by write center c, while PBA orders levels by the read range upper bound xh. In Figure 5, SBA finds a 2-level allocation, while PBA finds a 3-level allocation – SBA may therefore miss the densest level allocation.

C. Bit Error Rate and ECC Overhead Comparison with SBA

Table II presents an accuracy and storage overhead comparison between `sba` and `pba` level allocations. The first two columns present the storage array and the number of bits-per-cell (2 BPC for 4-level,

Chip	# Cells	Readout	Write Algorithm	# Cells	Resistance Window	# Resistances/Cell	# Cells Tested
Sapiens [26]	64k	off-chip	off-chip RADAR [3]	random 200	$7.8k\Omega - 40k\Omega$	32	random 200
Ember1,Ember2 [27]	3M	on-chip ADC (6 bits)	on-chip ISPP [3]	random 16k	1-64 ADC levels	64	same 16k

Table I: Summary of benchmark hardware platforms. For all chips, the relaxation time is 1 second. All chips use a 40 nm CMOS process.

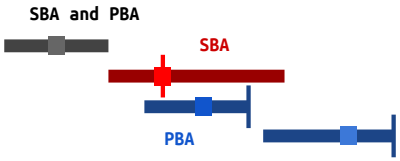


Figure 5: SBA sorts by write center c; PBA sorts by read upper bound xh.

		Bit Error Rate				ECC Overhead			
Chip	BPC	`sba`	`pba`	ΔBER	Rel. ΔBER	`sba`	`pba`	ΔECC	Rel. ΔECC
Sapiens	2	0.93%	0.27%	-0.66%	-71%	13%	8%	-5.5%	-41%
Sapiens	3	3.4%	2.4%	-1%	-30%	30%	23%	-6.4%	-22%
Ember1	2	0.046%	0%	-0.046%	-100%	4.6%	0%	-4.6%	-100%
Ember1	3	0.74%	0.38%	-0.36%	-49%	12%	9.1%	-3.2%	-26%
Ember2	2	0%	0%	0%	N/A	0%	0%	0%	N/A
Ember2	3	0.7%	0.37%	-0.34%	-48%	12%	9%	-2.8%	-23%

Table II: Comparison between SBA and PBA level allocations for 2 and 3 bits-per-cell.

3 BPC for 8-level). Columns 3-6 compare the raw bit error rates for SBA and PBA, and columns 7-10 present an iso-BER storage comparison of SBA and PBA where all MLC storage arrays are configured to use an error correcting-code (ECC) with a BER of 10^{-14}. For each comparison, both the absolute difference `pba` - `sba`, and the relative difference (`pba` - `sba`) / `sba` are reported, where negative absolute and relative differences are better. Note that the 2 BPC allocation for Ember2 produces no observable errors (0% for both `sba` and `pba`), and the 2 BPC allocation for Ember1 produces one observable error (0.046%) for the SBA allocation and zero observable errors for the PBA allocation.

Observation 1. PBA produces level allocations with lower bit error rates for all chip and bits-per-cell configurations compared to SBA. PBA delivers 0.046%-1.00% absolute reductions in BER over SBA, and achieves an overall relative reduction in error rate by 30%-71% (with 100% being an outlier).

Observation 2. PBA produces level allocations with lower ECC overheads for all chips and bits-per-cell configurations compared to SBA. PBA delivers 2.8%-6.4% absolute reductions in ECC storage overhead over SBA, and achieves an overall relative reduction in storage overhead of 22%-41% (with 100% being an outlier).

D. *Performance Comparison with* SBA

We empirically compare the running time of the PBA and SBA algorithms with the same dataset. We implement both algorithms in Python, and we measure the time for both algorithms to generate the level allocations for 4 to 8 levels (2 to 3 BPC). Note that level allocation is a one-time effort before storage array usage.

Observation 3. PBA's level allocation algorithm is more than two orders of magnitude faster than SBA. PBA takes 0.0065 seconds in total, while SBA takes 3.4 seconds. The main reason is that PBA's the binary search over the target error bound (line 2 in Algorithm 1) reduces the complexity of the algorithm while SBA does a linear search [9] over the target error bound.

VI. Ablation Study

We next perform an ablation study to evaluate the impact of PBA's algorithmic improvements and the impact of eliminating parametrized distribution-based modeling on the BERs of the produced level allocations. To perform this analysis, we introduce a second baseline, `pba-norm`, which uses SBA's normal distribution-based level construction approach (Section V-B, Point 1), and PBA's level allocation algorithm. Comparisons between SBA and `pba-norm` isolate the impact of the PBA algorithm, and comparisons between `pba-norm` and PBA isolate the impact of eliminating parametrized distributions from the algorithm.

A. *Comparison with* PBA *Parametrized Distribution Variant*

Table III presents the results of the ablation study. ΔBER1 computes the bit error rate difference `sba` - `pba-norm` representing the BER reduction of SBA over `pba-norm`. ΔBER2 computes `pba` - `pba-norm` indicating the BER reduction of PBA over `pba-norm`. Similarly, we also present the difference for ECC overhead.

Observation 4. The major improvement of PBA over SBA comes from getting rid of a parametrized distribution to model the data. The percentage reduction shown in ΔBER2 and ΔECC2 are bigger than the ΔBER1 and ΔECC1. We can see both positive numbers and negative numbers in ΔBER1 and ΔECC1, suggesting that `pba-norm` does not indeed provide much benefit over `sba`. This indicates that the major improvement of PBA over SBA comes from avoiding a parametrized distribution to model the data.

VII. Statistical Study of RRAM Datasets

In Section VI, we find that the PBA's variant `pba-norm` does not perform well because it adopts a parametrized normal distribution. This suggests that the raw data points are quite irregular and may not actually conform to a parametrized distribution. In this section, we will conduct a normality test for a series of RRAM characterization datasets to further confirm our hypothesis.

Table IV reports the percent of the data points for each write center that conform to a normal distribution (i.e., passing the D'Agostino's K-squared test with a p-value of 0.001). The table presents the normality results for the write and relaxation datasets collected in our experiment (Sapiens). We also run the normality test on the data collected in prior work, including RRAM write algorithm dataset (Radar) [3], [28] and the publicly available RRAM relaxation datasets [29] for three different device technologies (Tech A, Tech B, Tech C). We do not consider the data collected for Ember1 and Ember2 in our own experiments because their data are too coarse-grained (at the granularity of only 64 levels) versus the datasets used in Table IV. We conduct the normality test for both resistance values and their reciprocal, i.e., conductance values, as shown in the "Measure" column. For each write dataset, the normality of the data points for each write center c_i and the write tolerances are analyzed (the "Write" column); for each relaxation dataset, the normality of the data points for each write center c_i and relaxation time t are analyzed (the "Relax" column).

Observation 5. The write and relaxation datasets collected in our experiment are all highly non-normal. We find that for the Sapiens datasets, 97.9% of the write distributions, and 100% of the relaxation distributions are non-normal.

Observation 6. The write datasets and relaxation datasets reported in prior work are also highly non-normal. We find 0% of the RADAR

		Bit Error Rate					ECC Overhead				
Hardware	BPC	`pba-norm`	`sba`	ΔBER1	`pba`	ΔBER2	`pba-norm`	`sba`	ΔECC1	`pba`	ΔECC2
Sapiens	2	0.51%	0.93%	**0.42%**	0.27%	**-0.24%**	10%	13%	**3.3%**	8%	**-2.2%**
Sapiens	3	3.6%	3.4%	**-0.19%**	2.4%	**-1.2%**	31%	30%	**-1.4%**	23%	**-7.8%**
Ember1	2	0.05%	0.05%	**0%**	0%	**-0.05%**	4.6%	4.6%	**0%**	0%	**-4.6%**
Ember1	3	0.74%	0.74%	**0%**	0.38%	**-0.36%**	12%	12%	**0%**	9.1%	**-3.1%**
Ember2	2	0%	0%	**0%**	0%	**0%**	0%	0%	**0%**	0%	**0%**
Ember2	3	0.8%	0.7%	**-0.1%**	0.37%	**-0.43%**	13%	12%	**-0.1%**	9%	**-3.6%**

Table III: Ablation study. ΔBER1 computes `sba` - `pba-norm`, and ΔBER2 computes `pba` - `pba-norm`. The percentage reduction in columns ΔBER2 and ΔECC2 (improvement of `pba` over `pba-norm`) are bigger than ΔBER1, ΔECC1 (comparison between `pba-norm` and `sba`). This suggests the major improvement of `pba` over `sba` comes from avoiding a parametrized distribution to model the data.

			Normal Percentage	
Dataset	**# Cells**	**Measure**	**Write**	**Relax**
Sapiens	200	resistance	2.1%	0%
Sapiens	200	conductance	2.1%	0%
Radar	8192	resistance	0%	-
Radar	8192	conductance	0%	-
Tech A	16384	resistance	-	8.2%
Tech A	16384	conductance	-	8.7%
Tech B	32768	resistance	-	4.5%
Tech B	32768	conductance	-	6.6%
Tech C	16292	resistance	-	0.6%
Tech C	16292	conductance	-	3.6%

Table IV: D'Agostino's K-squared normality test results for RRAM characterization datasets. We conduct normality test on both resistance values and its inverse (conductance values). We report the fraction of write centers which conform to normal distribution.

algorithm write distributions are normally distributed. We also find that only 0.6%-8.7% of the relaxation distributions for the Tech A, Tech B, and Tech C device technologies are normally distributed. Almost all (more than 90%) of the studied distributions are non-normal, even though the data from prior work is collected over a much larger number of cells.

The above observations give us more insight into why avoiding a parametrized distribution to fit the data is a good idea. Due to the irregularities present in the dataset, a parametrized distribution cannot fully capture the high-order information in the original data points.

VIII. Datasets and Level Allocation

We next explore the interplay between the quality of the characterization dataset and the BERs of the SBA and PBA level allocations. This analysis focuses on 3 BPC level allocations for the Ember chips instances because the 3 BPC allocations exhibit higher error and the Ember chips datasets contain more entries.

A. Multi-chip Data and Level Allocation

In this subsection, we study how the level allocation algorithms will perform when taking into account the inter-chip variations.

We run the SBA and PBA level allocation algorithms for Ember1 and Ember2 using four different characterization datasets constructed from the original Ember1 and Ember2 data. The `100/0` dataset contains only the target chip's characterization data, and is the baseline dataset for this analysis. The balanced `50/50` dataset contains an equal amount of Ember1's and Ember2's characterization data. The unbalanced `10/90` dataset contains a small amount (10%) of the target chip's data, and a large amount (90%) of non-target chip's data, and the `0/100` dataset contains only the non-target chip's characterization data. Together, these datasets capture the effect of using multiple chip characterization datasets, and using chip characterization datasets collected from the non-target chip, on level allocation. Table V presents the BERs and ECC overheads associated with different characterization datasets.

Observation 7. PBA consistently outperforms SBA in BER and ECC overhead across diverse datasets, comprising a combination of data collected from the target chip as well as another similar chip, with varying proportions. Across all combined datasets, PBA produces level allocations with 15%-29% lower BERs and 8%-19% lower ECC overheads than SBA.

Observation 8. PBA can deliver even larger reduction in BER and ECC overhead over SBA when provided with sufficient target chip data. In general, increasing the amount of target chip data provided to the algorithm leads to improved performance. This trend is particularly pronounced for PBA in comparison to SBA. The disparity between the two algorithms, as indicated by the relative improvement (Rel.ΔBER and Rel.ΔECC), becomes more prominent with greater quantities of target chip data. For example, the ΔBER value with the `100/0` dataset exhibits the largest absolute value among all the ΔBER and Rel.ΔBER values across different dataset ratios, highlighting the substantial impact of increased target chip data on the performance differentiation between the algorithms.

B. Dataset Size and Level Allocation

We next investigate the impact of the dataset size on the fidelity of the level allocation. We construct datasets that contain 25%, 50%, 75%, 90%, and 100% of the original chip characterization data. Because these smaller datasets are sampled from the original dataset, we repeat the random sampling process 10 times and report the mean and standard deviation (σ) of the BERs across the 10 trials. Randomly sampling data represents characterizing a subset of cells' behavior, and this can show the algorithm's stability given the randomness in selecting cells for data collection. Table VI presents the bit error rates (BER) associated with different characterization dataset sizes. For the 100% dataset, $\sigma = 0\%$ as there is no sampling.

Observation 9. PBA finds lower-error level allocations than SBA across different sizes of the dataset, and PBA better exploits the behavior captured in large datasets than SBA. PBA attains 1%-43% lower BERs than SBA across all reduced dataset sizes, and the improvement over SBA is larger given more data.

Observation 10. PBA generally produces level allocations with BERs that are slightly more stable or at parity with SBA on smaller datasets. Intuitively, the BER of the level allocation is less stable for smaller dataset (i.e., larger σ is observed on smaller dataset). PBA's stability is generally better than or as good as SBA's stability (except for one data point of 75% dataset on Ember2).

IX. Discussion

Parametrized Distributions. Prior approaches to level allocation typically work with normal or normal/lognormal distributions that characterize the mean shift and variance of the read resistance as a

	Ember1 Bit Error Rate				Ember1 ECC Overhead				Ember2 Bit Error Rate				Ember2 ECC Overhead			
Dataset	**sba**	**pba**	ΔBER	Rel.Δ	**sba**	**pba**	ΔECC	Rel.Δ	**sba**	**pba**	ΔBER	Rel.Δ	**sba**	**pba**	ΔECC	Rel.Δ
100/0	0.74%	0.38%	-0.36%	**-49%**	12%	9.1%	-2.9%	**-26%**	0.7%	0.37%	-0.34%	**-48%**	12%	9%	-2.8%	**-23%**
50/50	0.62%	0.46%	-0.16%	**-26%**	11%	9.6%	-1.4%	**-13%**	0.8%	0.49%	-0.31%	**-29%**	13%	10%	-2.5%	**-19%**
10/90	0.78%	0.65%	-0.13%	**-17%**	12%	11%	-1%	**-8%**	0.86%	0.64%	-0.22%	**-26%**	13%	11%	-1.8%	**-14%**
0/100	0.75%	0.64%	-0.11%	**-15%**	12%	11%	-1%	**-8%**	1%	0.72%	-0.28%	**-28%**	14%	12%	-2.1%	**-15%**

Table V: Interchip level allocation comparison (3 bits-per-cell). 100/0 represents 100% target chip's dataset, and 50/50 represents 50% target chip's dataset combined with 50% from the other similar chip.

	Ember1 Bit Error Rate				Ember1 BER σ		Ember2 Bit Error Rate				Ember2 BER σ	
Dataset	**sba**	**pba**	ΔBER	Rel.Δ	**sba**	**pba**	**sba**	**pba**	ΔBER	Rel.Δ	**sba**	**pba**
25%	0.75%	0.74%	-0.01%	**-1%**	0.17%	0.17%	0.78%	0.57%	-0.21%	**-27%**	0.19%	0.17%
50%	0.76%	0.53%	-0.23%	**-30%**	0.10%	0.06%	0.70%	0.50%	-0.20%	**-29%**	0.16%	0.12%
75%	0.68%	0.43%	-0.26%	**-38%**	0.05%	0.05%	0.68%	0.46%	-0.22%	**-32%**	0.08%	0.10%
90%	0.68%	0.42%	-0.26%	**-38%**	0.052%	0.029%	0.67%	0.38%	-0.29%	**-43%**	0.1%	0.056%
100%	0.74%	0.38%	-0.36%	**-49%**	0%	0%	0.7%	0.37%	-0.34%	**-48%**	0%	0%

Table VI: Effect of dataset size on level allocation (3 bits-per-cell).

function of the target write resistance ($\mu(c)$, $\sigma(c)^2$) [3], [9], [12]–[14]. We find that both RRAM conductances and resistances are highly non-normal and cannot be precisely modeled with a normal distribution. Under certain conditions, though, the distributional parameters can reasonably approximate behavior. For example, the variance and standard deviation can be useful statistical measures to model the dispersion of a distribution. Practitioners can improve model fit by fitting data to more sophisticated distributions. Doing so requires re-engineering the level allocation algorithms and potentially complicates the algorithm's structure. In contrast, PBA's data-driven approach enables the level allocation algorithm to consider higher-order statistical characteristics present in the data without changing the algorithm's implementation.

Characterization Data. We observe that the composition and size of the characterization dataset have a substantial impact on the BERs of the produced level allocations. In Section VIII, we found PBA is able to identify lower error level allocations when provided with enough characterization data collected from the target chip and generally identifies better level allocations with more data.

We acknowledge that, in practice, it may not be feasible to extensively characterize each fabricated chip to compute a low error level allocation. However, it may be possible to augment data without requiring extensive chip characterization. Researchers have previously leveraged simulation-based data augmentation approaches to infer larger datasets from small amounts of empirical data [30]. These methods can be applied to generate data for level allocation.

Read Circuit Design. Our method assumes that read circuit non-idealities (e.g., sense amplifier margin, thermal and voltage noise, etc.), which may be reduced through enhanced analog circuit design, are included in the characterization datasets. However, one can separate out their impact and directly model their behavior if well known by adding a read margin to account for their effects as is done in SBA [9]. Such an approach can allow for read circuit refinements to increase effective storage density. We anticipate that this capability can be added to PBA with minor changes to how the non-overlapping levels are computed in our algorithm.

Temperature and Wear. Currently, PBA supports data queries over write centers and relaxation times and provides guidance for characterizing the write algorithms, read circuitry, process variation, and relaxation. We anticipate that PBA can be extended to support the characterization of device wear and temperature changes. In these usages, the data provider will accept additional parameters that capture different operating temperatures and wear conditions, and the allocation algorithm will retrieve data from the provider that reflects the desired operating conditions.

Other Resistive Memory Technologies. While we focus on applying PBA to RRAM, the PBA level allocation approach can be applied to other resistive memory technologies. Because PBA does not fit data to a parameterized distribution, it can potentially be applied to memories that exhibit a number of non-standard distributional characteristics.

X. Related Work

Researchers have used multiple-bits-per-cell (MPBC) storage for various applications [2], [9], [12]–[14], [17], [31]–[33]. Researchers have extensively studied the device modeling and statistical distributions present in RRAM and phase-change memory (PCM) storage arrays [1]–[3], [22], [34]–[36]. Level allocation algorithms in prior work typically work with a known distribution [3], [9], [12]–[14], [37] of the observed analog behavior and leverage the distribution's parameters for level allocation. Works applying MBPC storage for various applications often discuss the data distribution used without explaining the level allocation algorithm in detail, e.g., prior work on PCM-based image storage [38], RRAM-based approximate computing [32], and RRAM-based variation-aware level allocation [35]. In this work, we develop an algorithm that makes no assumptions about the distribution of the collected data or any device-specific statistical trends. Our work can reduce the storage overhead for applications using multiple-bits-per-cell storage.

XI. Conclusion

There has been a proliferation of emerging resistance-based non-volatile storage technologies that have the potential to deliver unprecedented storage and performance benefits – these technologies can deliver further density improvements if used as a multiple-bits-per-cell (MBPC) storage medium. In this paper, we presented PBA, a percentile-based level allocation algorithm that computes level allocations directly on characterization data. PBA is able to deliver higher accuracy MBPC storage schemes than state-of-the-art allocation algorithms that fit data to parametrized distributions. We believe that our approach can potentially generalize to a broader set of emerging memory technologies that support MBPC storage.

XII. Acknowledgements

This research was supported in part by DARPA 3DSoC, TSMC, Stanford SystemX Alliance, Stanford Precourt Institute for Energy, Samsung, and ACCESS – AI Chip Center for Emerging Smart Systems, sponsored by InnoHK funding, Hong Kong SAR.

REFERENCES

[1] A. Grossi et al. Fundamental variability limits of filament-based RRAM. In *IEDM*, 2016.

[2] Weier Wan et al. Edge AI without compromise: Efficient, versatile and accurate neurocomputing in resistive random-access memory. *arXiv*, 2021.

[3] Binh Q Le et al. RADAR: A fast and energy-efficient programming technique for multiple bits-per-cell RRAM arrays. *IEEE TED*, 2021.

[4] Mindy D Bishop et al. Monolithic 3-D integration. *Micro*, 2019.

[5] Max M Shulaker et al. Monolithic 3D integration of logic and memory: Carbon nanotube FETs, resistive RAM, and silicon FETs. In *IEDM*, 2014.

[6] Kartik Prabhu et al. CHIMERA: A 0.92-TOPS, 2.2-TOPS/W Edge AI Accelerator With 2-MByte On-Chip Foundry Resistive RAM for Efficient Training and Inference. *IEEE JSSC*, 2022.

[7] E. R. Hsieh et al. Four-bits-per-memory one-transistor-and-eight-resistive-random-access-memory (1t8r) array. *IEEE EDL*, 2021.

[8] Ping-Yi Hsieh et al. Monolithic 3D BEOL FinFET switch arrays using location-controlled-grain technique in voltage regulator with better FOM than 2d regulators. In *IEDM*, 2019.

[9] Binh Q Le et al. Resistive RAM with multiple bits per cell: Array-level demonstration of 3 bits per cell. *IEEE TED*, 2018.

[10] Biresh Kumar Joardar et al. Accured: High accuracy training of cnns on reram/gpu heterogeneous 3-d architecture. *IEEE TCAD*, 2020.

[11] Ping Chi et al. PRIME: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory. *ISCA*, 2016.

[12] Tony F Wu et al. 14.3 A 43pJ/Cycle non-volatile microcontroller with 4.7 μs shutdown/wake-up integrating 2.3-bit/cell resistive RAM and resilience techniques. In *ISSCC*, 2019.

[13] E. R. Hsieh et al. High-density multiple bits-per-cell 1t4r RRAM array with gradual SET/RESET and its effectiveness for deep learning. In *IEDM*, 2019.

[14] E. R. Hsieh et al. Four-Bits-Per-Memory One-Transistor-and-Eight-Resistive-Random-Access-Memory (1T8R) Array. *IEEE EDL*, 2021.

[15] Alessandro Grossi et al. Resistive RAM endurance: Array-level characterization and correction techniques targeting deep learning applications. *IEEE TED*, 2019.

[16] Cheng-Xin Xue et al. 24.1 A 1Mb multibit ReRAM computing-in-memory macro with 14.6 ns parallel MAC computing time for CNN based AI edge processors. In *ISSCC*, 2019.

[17] Wei-Hao Chen et al. A 65nm 1Mb nonvolatile computing-in-memory ReRAM macro with sub-16ns multiply-and-accumulate for binary DNN AI edge processors. In *ISSCC*, 2018.

[18] Cheng-Xin Xue et al. 15.4 A 22nm 2Mb ReRAM compute-in-memory macro with 121-28 TOPS/W for multibit MAC computing for tiny AI edge devices. In *ISSCC*, 2020.

[19] Cheng-Xin Xue et al. 16.1 A 22nm 4Mb 8b-precision ReRAM computing-in-memory macro with 11.91 to 195.7 TOPS/W for tiny AI edge devices. In *ISSCC*, 2021.

[20] Haitong Li et al. One-shot learning with memory-augmented neural networks using a 64-kbit, 118 GOPS/W RRAM-based non-volatile associative memory. In *Symposium on VLSI Technology*, 2021.

[21] Chung-Cheng Chou, Zheng-Jun Lin, Pei-Ling Tseng, Chih-Feng Li, Chih-Yang Chang, Wei-Chi Chen, Yu-Der Chih, and Tsung-Yung Jonathan Chang. An n40 256k×44 embedded rram macro with sl-precharge sa and low-voltage current limiter to improve read and write performance. In *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*, pages 478–480, 2018.

[22] D Ielmini et al. Physics-based modeling approaches of resistive switching devices for memory and in-memory computing applications. *Journal of Computational Electronics*, 2017.

[23] Meng-Fan Chang et al. A high-speed 7.2-ns read-write random access 4-Mb embedded resistive RAM (ReRAM) macro using process-variation-tolerant current-mode read schemes. *IEEE Journal of Solid-State Circuits*, 2012.

[24] Vasileios Ntinas et al. Power-efficient noise-induced reduction of ReRAM cell's temporal variability effects. *IEEE Transactions on Circuits and Systems*, 2020.

[25] Gray Codes, 2022. https://en.wikipedia.org/wiki/Gray_code.

[26] Haitong Li et al. SAPIENS: A 64-kb RRAM-based non-volatile associative memory for one-shot learning and inference at the edge. *IEEE TED*, 2021.

[27] Luke Upton et al. EMBER: A 100 MHz, 0.86 mm², Multiple-Bits-per-Cell RRAM Macro in 40 nm CMOS with Compact Peripherals and 1.0 pJ/bit Read Circuitry. In *ESSCIRC*, 2023.

[28] Write Dataset for RRAM, 2023. https://github.com/akashlevy/RRAM-RADAR-Tuning.

[29] Relaxation Dataset for RRAM, 2023. https://github.com/akashlevy/RRAM-Relaxation-Master-Model.

[30] Imtiaz Hossen, Mark A Anders, Lin Wang, and Gina C Adam. Data-driven rram device models using kriging interpolation. *Scientific Reports*, 12(1):5963, 2022.

[31] Xin Zheng et al. Error-resilient analog image storage and compression with analog-valued RRAM arrays: an adaptive joint source-channel coding approach. In *IEDM*, 2018.

[32] Boxun Li, Peng Gu, Yi Shan, Yu Wang, Yiran Chen, and Huazhong Yang. Rram-based analog approximate computing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(12):1905–1917, 2015.

[33] Beiye Liu, Hai Li, Yiran Chen, Xin Li, Qing Wu, and Tingwen Huang. Vortex: Variation-aware training for memristor x-bar. In *Proceedings of the 52nd Annual Design Automation Conference*, pages 1–6, 2015.

[34] Shyh-Shyuan Sheu et al. A 4Mb embedded SLC resistive-RAM macro with 7.2 ns read-write random-access time and 160ns MLC-access capability. In *ISSCC*, 2011.

[35] Artem Glukhov, Valerio Milo, Andrea Baroni, Nicola Lepri, Cristian Zambelli, Piero Olivo, Eduardo Pérez, Christian Wenger, and Daniele Ielmini. Statistical model of program/verify algorithms in resistive-switching memories for in-memory neural network accelerators. In *2022 IEEE International Reliability Physics Symposium (IRPS)*, pages 3C–3. IEEE, 2022.

[36] N Papandreou, A Sebastian, A Pantazi, M Breitwisch, C Lam, H Pozidis, and E Eleftheriou. Drift-resilient cell-state metric for multilevel phase-change memory. In *2011 International Electron Devices Meeting*, pages 3–5. IEEE, 2011.

[37] Cong Xu et al. Understanding the trade-offs in multi-level cell ReRAM memory design. In *DAC*, 2013.

[38] Qing Guo, Karin Strauss, Luis Ceze, and Henrique S Malvar. High-density image storage using approximate memory cells. *ACM SIGPLAN Notices*, 51(4):413–426, 2016.