DALA: Distribution-Agnostic Level Allocation for Multiple Bits-Per-Cell RRAM

Anjiang Wei, Pu Yi, Akash Levy, Robert M. Radway, Priyanka Raina, Subhasish Mitra, Sara Achour
Background

- Resistive random access memories (RRAM)
  - Analog storage
- Multiple-Bits-Per-Cell (MBPC)
  - 2 bits: 4 levels
  - 4 bits: 16 levels
- Mapping between levels and bits

An example 4-level (2 bit) allocation for resistive cell.
Background

- Write Operation
- Read Operation
- Analog Non-idealities
  - Level Drift Errors
  - Bit Error Rate

An example 4-level (2 bit) allocation for resistive cell.
Background

- Level Allocation Algorithm
  - Write Range: Write Center, Write Tolerance
  - Read Range: Read Low, Read High

An example 4-level (2 bit) allocation for resistive cell.
Prior Work

● Level Allocation
  ○ Collect and characterize hardware data
  ○ Design algorithm based on characterization data

● Limitations of prior work (SBA)
  ○ Parameter-based hardware models
    ■ Standard deviation
    ■ Over-approximation (assuming normal distribution)
  ○ Non-Generalizable algorithm for level allocation
    ■ Parameter-based models
Contribution

- **DALA**: a distribution- and technology-agnostic level allocation
  - RDR resistance hardware model
  - More general, technology-agnostic algorithm
- **RDR Resistance Model**
  - *Distribution-agnostic* analog resistance model
- **Level Allocation Algorithm**
  - *No assumptions* about statistical trends in the data
  - *Device-agnostic search*, can potentially be applied to other technologies
- **Comparison with prior work (SBA)**
  - 29.6%-71.0% lower bit-error rates
  - 21.6%-40.9% lower ECC storage overheads
RDR Analog Resistance Model

- Two kinds of errors:
  - Write Error (write center $c$, write tolerance $z$)
  - Drift Error (Write distribution, time $t$)

\[ \hat{r} \sim \text{RDR}(c, z, t) = P_{wr}(c, z) + P_{dr}(P_{wr}(c, z), t) \]

- Operations
  - PDF($\text{RDR}(c, z, t)$): construct a probability density function (PDF) from the model
  - Percentile($P$, $F$): compute the $F$ percentile from the probability density function
  - Probability($P$, $r_1$, $r_2$): compute the probability the resistance is within $[r_1, r_2]$
Data Collection for RDR Model

- Write Resistance Dataset
  - Try different write centers $c$ and write tolerances $z$
- Resistance Drift Dataset
  - Also try different timestamp $t$
- Construction of Distributions
  - Weighted linear interpolation over distributions
  - SBA: linear interpolation over standard deviation (sigma)
RDR Analog Resistance Model

- An abstraction layer, an interface between:
  - Real hardware data
  - Algorithm design
- Potentially transferable abstraction layer
  - For other hardware technologies
  - More data of the same hardware
Level Allocation Algorithm

● Basic Requirements
  ○ Read Ranges should NOT overlap
  ○ Write Ranges should be within Read Range

● Error evaluation:
  ○ Probability of writing to level $i$, ends up at level $j$ at time $t$

$$e_{i,j}(t) = \text{Probability}(\text{PDF}(\text{RDR}(c_i, z_i, t)), x_{l_j}, x_{h_j})$$

○ Average drift error probability

$$e_{avg}(t) = \frac{1}{n} \sum_{i=1}^{n} 1 - e_{i,i}(t)$$
Level Allocation Algorithm

- **Basic Algorithm**
- **Input:**
  - RDR: a hardware model
  - n: number of levels
  - t: timestamp
  - error margin
  - C, Z: defining the search space (write centers, write tolerances)
- **Idea:** progressively increases a maximum level drift error by the provided margin until it finds a satisfying level allocation of n levels

```python
Function LevelAlloc(RDR, n, t, ε, C, Z):
    for γ ∈ [0, ε, 2ε, ..., 1] do
        Levels = LevelGeneration(RDR, t, γ, C, Z)
        L = FindAllocation(Levels)
        if len(L) == n then
            return L
```
Level Allocation Algorithm

- Search Space Construction
- Idea: construct a candidate of levels w.r.t. the error bound specification
  - These candidates may overlap with each other
  - We need to pick longest non-overlapping levels for read ranges

**Function** LevelGeneration \((RDR, t, \gamma, C, Z)\):

```
for \langle c, z \rangle \text{ in } C \times Z \text{ do}
    P = PDF(RDR(c, z, t))
    xl = Percentile(P, [\frac{1}{2} \cdot \gamma] \times 100\%)
    xh = Percentile(P, [1 - \frac{1}{2} \cdot \gamma] \times 100\%)
    assert 1 - Probability(P, xl, xh) \leq \gamma
    assert [c - z, c + z] \subseteq [xl, xh]
    generate \langle xl, xh, c, z \rangle
```
Level Allocation Algorithm

- Level Allocation Construction
- Idea: find a level allocation with the largest number of non-overlapping levels

Function FindAllocation(L):

\[
\text{LevelAlloc} = [] \quad # \text{Selected Non-Overlapping Levels}
\]

\[
\text{SortLevels} = \text{sort}(\text{Levels}, \text{key} = xh)
\]

\[
\text{LargestR} = 0
\]

\[
\text{for } (xl, xh, c, z) \text{ in SortLevels do}
\]

\[
\text{if } |\text{LevelAlloc}| \leq 0 \lor xl \geq \text{LargestR} \text{ then}
\]

\[
\text{LargestR} = xh
\]

\[
\text{LevelAlloc} = \text{LevelAlloc} :: (xl, xh, c, z)
\]

\[
\text{return } \text{LevelAlloc}
\]
Level Allocation Algorithm

- We can guarantee that algorithm will always find an n-level allocation if one exists in the search space
- Algorithm:
  - First sorts all the generated candidate levels by their upper read resistance $x_h$
  - Iteratively adds the leftmost candidate level that does not overlap with the rightmost level in the level allocation

```python
Function FindAllocation(Layers):
    LevelAlloc = []  # Selected Non-Overlapping Levels
    SortLevels = sort(Layers, key = xh)
    LargestR = 0
    for ⟨xl, xh, c, z⟩ in SortLevels do
        if |LevelAlloc| == 0 ∨ xl ≥ LargestR then
            LargestR = xh
            LevelAlloc = LevelAlloc :: ⟨xl, xh, c, z⟩
        return LevelAlloc
```
Experimental Setup

- Baselines for comparison
  - SBA
  - Dala-sigma: same hardware model as SBA
  - Dala-norm: assuming normal distribution

- Data collection
  - All baselines share the same dataset
Result Analysis

- DALA consistently produces lower error level allocations than SBA
Result Analysis

- The **RDR model** enables DALA to produce much lower-error allocations
- Compare dala & dala-sigma
Result Analysis

- Assuming normal distribution leads to sub-optimal allocation
- Compare dala & dala-norm
Result Analysis

- Almost all write/relaxation data indicate that the distribution is NOT normal
  - Radar: write dataset
  - Tech A/B/C: relaxation dataset

<table>
<thead>
<tr>
<th>dataset</th>
<th>number of cells</th>
<th>measurement</th>
<th>Normal Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>DALA</td>
<td>200</td>
<td>resist.</td>
<td>2.1% 0% 0%</td>
</tr>
<tr>
<td>DALA</td>
<td>200</td>
<td>cond.</td>
<td>2.1% 0% 0%</td>
</tr>
<tr>
<td>radar</td>
<td>8192</td>
<td>resist.</td>
<td>0% - -</td>
</tr>
<tr>
<td>radar</td>
<td>8192</td>
<td>cond.</td>
<td>0% - -</td>
</tr>
<tr>
<td>tech A</td>
<td>16384</td>
<td>resist.</td>
<td>- 8.2% -</td>
</tr>
<tr>
<td>tech A</td>
<td>16384</td>
<td>cond.</td>
<td>- 8.7% -</td>
</tr>
<tr>
<td>tech B</td>
<td>32768</td>
<td>resist.</td>
<td>- 4.5% -</td>
</tr>
<tr>
<td>tech B</td>
<td>32768</td>
<td>cond.</td>
<td>- 6.6% -</td>
</tr>
<tr>
<td>tech C</td>
<td>16292</td>
<td>resist.</td>
<td>- 0.6% -</td>
</tr>
<tr>
<td>tech C</td>
<td>16292</td>
<td>cond.</td>
<td>- 3.6% -</td>
</tr>
</tbody>
</table>

Table I

D’Agostino’s K-squared normality test results for RRAM CHARACTERIZATION DATASETS.
Experiment Results

- Compared to SBA
- Bit Error Rates
  - 71.0% BER reduction for 2 bits-per-cell (BPC)
  - 29.6% BER reduction for 3 BPC
- Error Correcting Code Overheads
  - 40.9% reduction in ECC overhead for 2 BPC
  - 21.6% reduction in ECC overhead for 3 BPC
Conclusion

- **DALA: a distribution- and technology-agnostic level allocation**
  - RDR resistance hardware model
  - More general, technology-agnostic algorithm

- **Prior work (SBA)**
  - Fit a known distribution to the observed analog behavior
  - Exploit device-specific statistical trends

- **Comparison with SBA**
  - 29.6%-71.0% lower bit-error rates
  - 21.6%-40.9% lower ECC storage overheads