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

anjiang@stanford.edu
Emerging Non-Volatile Memories

- Emerging non-volatile memory technologies
  - Resistive random-access memory (RRAM or ReRAM)
  - Conductive bridge random access memory (CBRAM)
  - Phase-change memory (PCM)
- Multiple-bits-per-cell storage for higher memory density
Level Allocation

- Map bit combinations to resistance ranges
  - Example: 2-bits-per-cell needs 4 levels
- Write center $c$
  - Program-and-verify algorithm; “write tolerance $z$” set by expert
- Read range $[x_l, x_h]$
Data Corruption

• Hardware non-idealities
  • Read noise, process variation, relaxation (over time), etc.

• Example
  • Write to Level 3, “11”
  • Read out from Level 4, “10”
  • Cause 1 bit flip
Prior Work on Level Allocation

• Collect characterization datasets
• Fit a common, parameterized distribution
  • E.g., normal or lognormal
• Produce level allocation using distribution parameters
  • E.g., standard deviation $\sigma$
Prior Work on RRAM Level Allocation

• SBA\textsuperscript{1}: Sigma-Based Allocation
• Fit data with normal distribution
  • Standard deviation $\sigma$ is a function of resistance
• Such approximation misrepresents analog behaviors
  • The raw data points are highly non-normal

\textsuperscript{1}Le et al. “Resistive RAM with multiple bits per cell: Array-level demonstration of 3 bits per cell.” TED 2018.
PBA: Percentile-Based Allocation

- Directly work with the characterization data
- No fitting a parameterized model, no approximation
- More accurately capture analog behaviors present in the data
How does PBA work?
PBA Level Allocation Algorithm

• Input:
  • $n$ # number of levels to allocate
  • $\varepsilon$ # minimum granularity of error

Function `LevelAlloc(n, \varepsilon)`:

# The loop can be binary search
for $\gamma \in [0, \varepsilon, 2\varepsilon, \ldots, 1]$ do

Candidates = `CandidateGen(\gamma)`

Result = `FindNonOverlap(Candidates)`

if Result.length == n then
  return Result
Candidate Level Generation

• From dataset
  • $C$ # list of write centers
  • $t$ # relaxation time
• Example candidate:

<table>
<thead>
<tr>
<th>Left Percentile</th>
<th>Right Percentile</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_l = 6.5, \text{k}\Omega$</td>
<td>$x_h = 9.2, \text{k}\Omega$</td>
</tr>
</tbody>
</table>

Error Probability $\gamma = 2\%$

Function $\text{CandidateGen}(\gamma)$:

1. Candidates = []
2. for $c$ in $C$ do
   1. R = GetData($c$, $t$) # List of resistance data points
   2. $x_l = \text{Percentile}(R, \frac{1}{2} \gamma)$
   3. $x_h = \text{Percentile}(R, 1 - \frac{1}{2} \gamma)$
   4. Candidates.append($(c, x_l, x_h)$)
3. return Candidates

Function $\text{Percentile}(R, \text{perc})$:

1. Rsorted = sort(R) # Sort points in increasing order
2. index = perc $\times$ size(R)
3. return Rsorted[index]
Find Maximum Non-Overlapping Levels

• Example

\[
\begin{align*}
\text{Cand} & \quad \text{Result} \\
xl_1 & \quad c_1 \quad xh_1 \\
xl_2 & \quad c_2 \quad xh_2 \\
xh_3 & \quad c_3 \quad xh_3 \\
xl_4 & \quad c_4 \quad xh_4 \\
\text{Result} & \quad c_1 \quad c_4 \\
\end{align*}
\]

Function **FindNonOverlap** *(Cand)*:

\[
\text{Result} = [] \quad \# \text{Non-overlapping levels}
\]

\[
\text{SortCand} = \text{sort}(\text{Cand}, \text{key}=xh) \quad \# \text{Sort by } xh
\]

\[
\text{UpperBound} = 0
\]

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

\[
\text{if} \ \text{xl} \ \geq \ \text{UpperBound} \ \text{then}
\]

\[
\text{Result}.\text{append}(\langle c, xl, xh \rangle)
\]

\[
\text{UpperBound} = xh \quad \# \text{Update the bound}
\]

\[
\text{return} \ \text{Result}
\]
Evaluation
Experimental Setup

• Evaluate on 3 fabricated RRAM storage arrays

<table>
<thead>
<tr>
<th>Chip</th>
<th># Total Cells</th>
<th>Readout</th>
<th>Resistance</th>
<th># Tested write centers</th>
<th># Tested cells</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sapiens¹</td>
<td>64k</td>
<td>Off-chip</td>
<td>7.8 - 40 kΩ</td>
<td>32</td>
<td>Random 200</td>
</tr>
<tr>
<td>Ember² 1</td>
<td>3M</td>
<td>On-chip ADC</td>
<td>1 – 64 levels</td>
<td>64</td>
<td>16k</td>
</tr>
<tr>
<td>Ember 2</td>
<td>3M</td>
<td>On-chip ADC</td>
<td>1 – 64 levels</td>
<td>64</td>
<td>16k</td>
</tr>
</tbody>
</table>

• Baseline: Sigma-Based Allocation (SBA)
• 4-level (2 bits-per-cell) and 8-level (3 bits-per-cell) allocations

² Upton et al. “EMBER: a 100 MHz, 0.86mm², Multiple-Bits-per-Cell RRAM Macro in 40 nm CMOS with Compact Peripherals and 1.0 pJ/Bit Read Circuitry,” ESSCIRC 2023
Results for Bit Error Rate

• Gray coding to map bits to levels

• Bit error rate (BER) = \( \frac{\text{# bit flips}}{\text{# total bits}} \)

<table>
<thead>
<tr>
<th>Chip</th>
<th>BPC</th>
<th>SBA BER</th>
<th>PBA BER</th>
<th>Rel. ΔBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sapiens</td>
<td>2</td>
<td>0.93%</td>
<td>0.27%</td>
<td>-71%</td>
</tr>
<tr>
<td>Sapiens</td>
<td>3</td>
<td>3.4%</td>
<td>2.4%</td>
<td>-30%</td>
</tr>
<tr>
<td>Ember1</td>
<td>2</td>
<td>0.05%</td>
<td>0%</td>
<td>-100%</td>
</tr>
<tr>
<td>Ember1</td>
<td>3</td>
<td>0.74%</td>
<td>0.38%</td>
<td>-49%</td>
</tr>
<tr>
<td>Ember2</td>
<td>2</td>
<td>0%</td>
<td>0%</td>
<td>N/A</td>
</tr>
<tr>
<td>Ember2</td>
<td>3</td>
<td>0.7%</td>
<td>0.37%</td>
<td>-48%</td>
</tr>
</tbody>
</table>

• Relative BER reduction by 30% - 71% (with 100% an outlier)
Results for ECC Overhead

• Error Correcting Code (ECC) overhead
  • Reliable storage medium: $10^{-14}$ unrecoverable bit error rate
  • Codeword size of at most 12 bits

<table>
<thead>
<tr>
<th>Chip</th>
<th>BPC</th>
<th>SBA ECC</th>
<th>PBA ECC</th>
<th>Rel. ΔECC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sapiens</td>
<td>2</td>
<td>13%</td>
<td>8%</td>
<td>-41%</td>
</tr>
<tr>
<td>Sapiens</td>
<td>3</td>
<td>30%</td>
<td>23%</td>
<td>-22%</td>
</tr>
<tr>
<td>Ember1</td>
<td>2</td>
<td>4.6%</td>
<td>0%</td>
<td>-100%</td>
</tr>
<tr>
<td>Ember1</td>
<td>3</td>
<td>12%</td>
<td>9.1%</td>
<td>-26%</td>
</tr>
<tr>
<td>Ember2</td>
<td>2</td>
<td>0%</td>
<td>0%</td>
<td>N/A</td>
</tr>
<tr>
<td>Ember2</td>
<td>3</td>
<td>12%</td>
<td>9%</td>
<td>-23%</td>
</tr>
</tbody>
</table>

• Relative ECC overhead reduction by 22% - 41% (with 100% an outlier)
Ablation Study

• Which factor contributes more to PBA’s advantage over SBA?
  • Finding longest non-overlapping levels ($\Delta{ECC1} = \text{SBA} – \text{PBA Norm}$)
  • Avoiding parameterized distribution ($\Delta{ECC2} = \text{PBA} – \text{PBA Norm}$)

<table>
<thead>
<tr>
<th>Chip</th>
<th>BPC</th>
<th>PBA Norm ECC</th>
<th>SBA ECC</th>
<th>Abs. $\Delta{ECC1}$</th>
<th>PBA ECC</th>
<th>Abs. $\Delta{ECC2}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sapiens</td>
<td>2</td>
<td>10%</td>
<td>13%</td>
<td>+3%</td>
<td>8%</td>
<td>-2%</td>
</tr>
<tr>
<td>Sapiens</td>
<td>3</td>
<td>31%</td>
<td>30%</td>
<td>-1%</td>
<td>23%</td>
<td>-8%</td>
</tr>
<tr>
<td>Ember1</td>
<td>2</td>
<td>5%</td>
<td>5%</td>
<td>0%</td>
<td>0%</td>
<td>-5%</td>
</tr>
<tr>
<td>Ember1</td>
<td>3</td>
<td>12%</td>
<td>12%</td>
<td>0%</td>
<td>9%</td>
<td>-3%</td>
</tr>
<tr>
<td>Ember2</td>
<td>2</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>Ember2</td>
<td>3</td>
<td>13%</td>
<td>12%</td>
<td>-1%</td>
<td>9%</td>
<td>-4%</td>
</tr>
</tbody>
</table>

• $|\Delta{ECC2}| > |\Delta{ECC1}|$: Major improvement comes from avoiding a parametrized distribution.
Statistical Study of RRAM Datasets

- D'Agostino’s $K^2$ normality tests for write and relaxation dataset
  - Both resistance and its reciprocal (conductance)
  - Radar, Tech A, Tech B, Tech C are open-source RRAM datasets

<table>
<thead>
<tr>
<th>Dataset</th>
<th># Cells</th>
<th>Resistance: Write Normal %</th>
<th>Resistance: Relax Normal %</th>
<th>Conductance: Write Normal %</th>
<th>Conductance: Relax Normal %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sapiens</td>
<td>200</td>
<td>2.1%</td>
<td>0%</td>
<td>2.1%</td>
<td>0%</td>
</tr>
<tr>
<td>Radar$^1$</td>
<td>8192</td>
<td>0%</td>
<td>-</td>
<td>0%</td>
<td>-</td>
</tr>
<tr>
<td>Tech A$^2$</td>
<td>16384</td>
<td>-</td>
<td>8.2%</td>
<td>-</td>
<td>8.7%</td>
</tr>
<tr>
<td>Tech B$^2$</td>
<td>32768</td>
<td>-</td>
<td>4.5%</td>
<td>-</td>
<td>6.6%</td>
</tr>
<tr>
<td>Tech C$^2$</td>
<td>16292</td>
<td>-</td>
<td>0.6%</td>
<td>-</td>
<td>3.6%</td>
</tr>
</tbody>
</table>

- Result: < 10% of the studied distributions are normal

https://github.com/akashlevy/RRAM-RADAR-Tuning
Multi-Chip Datasets and Level Allocation

• Multi-chip datasets
  • 100/0 contains only the target chip’s data
  • 0/100 contains only the non-target chip’s data

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Ember1 SBA BER</th>
<th>Ember 1 PBA BER</th>
<th>Ember 1 Rel. Δ BER</th>
<th>Ember2 SBA BER</th>
<th>Ember 2 PBA BER</th>
<th>Ember 2 Rel. Δ BER</th>
</tr>
</thead>
<tbody>
<tr>
<td>100/0</td>
<td>0.74%</td>
<td>0.38%</td>
<td>-49%</td>
<td>0.70%</td>
<td>0.37%</td>
<td>-48%</td>
</tr>
<tr>
<td>50/50</td>
<td>0.62%</td>
<td>0.46%</td>
<td>-26%</td>
<td>0.80%</td>
<td>0.49%</td>
<td>-29%</td>
</tr>
<tr>
<td>10/90</td>
<td>0.78%</td>
<td>0.65%</td>
<td>-17%</td>
<td>0.86%</td>
<td>0.64%</td>
<td>-26%</td>
</tr>
<tr>
<td>0/100</td>
<td>0.75%</td>
<td>0.64%</td>
<td>-15%</td>
<td>1.00%</td>
<td>0.72%</td>
<td>-28%</td>
</tr>
</tbody>
</table>

• Observation:
  • PBA consistently outperforms on multi-chip datasets
  • PBA can deliver more benefits if provided with sufficient target chip data
Reduced Datasets and Level Allocation

• Use only 25%, 50%, 75%, 90% of the original datasets

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Ember1 SBA BER</th>
<th>Ember 1 PBA BER</th>
<th>Ember 1 Rel. Δ BER</th>
<th>Ember2 SBA BER</th>
<th>Ember 2 PBA BER</th>
<th>Ember 2 Rel. Δ BER</th>
</tr>
</thead>
<tbody>
<tr>
<td>25%</td>
<td>0.75%</td>
<td>0.74%</td>
<td>-1%</td>
<td>0.78%</td>
<td>0.57%</td>
<td>-27%</td>
</tr>
<tr>
<td>50%</td>
<td>0.76%</td>
<td>0.53%</td>
<td>-30%</td>
<td>0.70%</td>
<td>0.50%</td>
<td>-29%</td>
</tr>
<tr>
<td>75%</td>
<td>0.68%</td>
<td>0.43%</td>
<td>-38%</td>
<td>0.68%</td>
<td>0.46%</td>
<td>-32%</td>
</tr>
<tr>
<td>90%</td>
<td>0.68%</td>
<td>0.42%</td>
<td>-38%</td>
<td>0.67%</td>
<td>0.38%</td>
<td>-43%</td>
</tr>
<tr>
<td>100%</td>
<td>0.74%</td>
<td>0.38%</td>
<td>-49%</td>
<td>0.70%</td>
<td>0.37%</td>
<td>-48%</td>
</tr>
</tbody>
</table>

• Observation
  • PBA outperforms SBA across different sizes of datasets
  • Improvement over SBA is larger given more data
Conclusion

• Emerging non-volatile memories for MBPC storage
• We present PBA, a percentile-based level allocation algorithm
  • Outperforms state-of-the-art RRAM algorithm: SBA
  • Multi-chip and reduced dataset scenarios
  • Do not fit data to parameterized distributions
• Potentially generalize to a broader set of emerging memory technologies that support MBPC storage