Aegis: Partitioning Data Block for Efficient Recovery of Stuck-at-Faults in Phase Change Memory

Jie Fan, Jiwu Shu, Youhui Zhang, and Weimin Zheng

Song Jiang

Tsinghua University

Wayne State University
Stuck-at Faults in PCM

- PCM has limited endurance.

- Stuck-at fault occurs when memory cell fails to change its value.
  - It is a major type of errors in PCM.
  - Values in such faulty cells can still be read.
  - The faults are permanent and accumulate.

- Two general error correction approaches at the chip level.
  - Pointer-based correction: Record the address of each faulty bit and its replacement bit (e.g., ECP).
  - Inversion-based correction: Partition data block into a number of groups and exploit the fact that stuck-at values are still readable (e.g., SAFER).
Inversion-based Correction (Stuck-at-Right)

Data: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Write

PCM Data Block
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Inversion Vector

F
Inversion-based Correction (Stuck-at-Right)

Data

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Write

F

PCM Data Block

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Inversion Vector

0
Inversion-based Correction (Stuck-at-Right)

Data Block

Write

Inversion Vector

PCM Data Block

Read

Data: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Write:

F

Inversion Vector: 0

PCM Data Block:

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Read:

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Inversion-based Correction (Stuck-at-Wrong)
Inversion-based Correction (Stuck-at-Wrong)

Data Block

<table>
<thead>
<tr>
<th>PCM Data Block</th>
<th>1 1 0 1 1 1 1 1 1 1 1 1 1 1 1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Invert & Write

F

Inversion Vector

0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
**Inversion-based Correction (Stuck-at-Wrong)**

**Data**

```
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
```

**Invert & Write**

```
F
```

**PCM Data Block**

```
1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

**Inversion Vector**

```
1
```

**Read & Invert**

```
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
```
Inversion-based Correction: Two Faults

Data Block

Data

Inversion Vector

0 0 0 0 0 0 1 0 0 0 0 0 0 0 0

F F

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Inversion-based Correction: Two Faults

Data

0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0

Inversion Vector

Data Block

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>F</td>
<td>0</td>
<td>F</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Partition Calculation w/ Fault Addresses:

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

XOR
Inversion-based Correction: Two Faults

Data Block: 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0

Inversion Vector:

Partition Calculation w/ Fault Addresses:

XOR:

0 0 1 1 0 1 0 1 1 1 0 1 0 1 0 1

Partition Vector:
Inversion-based Correction: Two Faults

Data: 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0

Group 0: Write

Data Block: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Inversion Vector: 0

F F
Inversion-based Correction: Two Faults

Data Block: 0 1 0 1 0 1 0 0 0 1 0 1 0 1 0 1

Group 1: Invert & Write

Data: 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0

Inversion Vector: 0 1
Inversion-based Correction: Two Faults

Data Block

0 1 0 1 0 1 0 0 0 1 0 1 0 1 0 1

Inversion Vector

0 1

Group 0: Read

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Inversion-based Correction: Two Faults

Data Block: 0 1 0 1 0 1 0 0 0 1 0 1 0 1 0 1

Inversion Vector: 0 1

Group 1: Read & Invert

Result: 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
Inversion-based Correction: the Third Fault

Data Block

Inversion Vector
Inversion-based Correction: the Third Fault

Data Block

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Partition Calculation w/ Fault Addresses:

F

F

Inversion Vector
Inversion-based Correction: the Third Fault

Data Block

Partition Calculation w/ Fault Addresses:

0 1 1 1 1
1 1 1 1 1
1 0 0 0

XOR

Partition vector

Inversion Vector
Inversion-based Correction: the Third Fault

Data Block

Partition Calculation w/ Fault Addresses:

0 1 1 1
1 1 1 1
1 0 0 0

Inversion Vector

Partition vector

In the worst scenario, with only five faults the block cannot be furthered partitioned!
Issues with the State-of-the-art Partition Scheme

- For a given data block of $n$ bits, there are only $\log_2 n$ partition configurations available to resolve fault collisions.

- In the worst case, group count can increase exponentially with accumulating faults.

- Only $\log_2 n$ faults could exhaust the configurations and essentially demand an inversion vector as large as the data block itself.
Design Objectives of Aegis

- A larger set of partition configurations for resolving fault collision to tolerate more faults.
  - A new configuration is needed whenever two faults collide in a group.
  - More candidate configurations mean more tolerable faults.

- A smaller number of groups in each configuration to reduce space overhead.
  - Group count mainly determines space overhead.

- Actively shuffling bits among groups to even out cell wears.
  - Cells in a group with faults wear out faster.
Bits of a data block are placed on the Cartesian plane.
A set of parallel lines defines a partition configuration.
Aegis’s Group Partition Scheme

- Aegis arranges bits of an $n$-bit data block on an $A \times B$ rectangle on the Cartesian plane.
- Bits $(a, b)$, where $b = (a \times k + y) \% B$, for a given slope $k$ and a given line $y$, are in the same group.
- Each $k \in [0, B - 1]$ corresponds to a partition configuration, and each $y \in [0, B - 1]$ corresponds to a group in the configuration.
We have proved that under the Aegis partition scheme:

Any two bits in the same group of a data block in a partition configuration will not be in the same group in a different partition configuration as long as:

- \( B \) is a prime number.
- \( A \leq B \)
A Concern: How about Collisions after Re-partitions?

There is possibility that multiple re-partitions are needed to reach a configuration without any fault collisions.
Aegis Guarantees a Collision-free Configuration

- Collision of any pair of faults appear in only one partition configuration.
- A data block of \( f \) faults can generate at most \( \binom{f}{2} \), or \( \frac{f \times (f+1)}{2} \), different collisions of fault pairs.
- Each re-partition eliminates at least one such collision.
- As long as number of configurations in a partition scheme, \( B \), is larger than \( \frac{f \times (f+1)}{2} \), there exists at least one collision-free configuration.
- For a set of known faults, a pre-wired logic can be used to compute collision-free configuration(s).
Aegis’s Advantages

To guarantee a tolerance of $f$ faults:

- Aegis provides $B$ partition configurations to resolve collisions. ($B$ is the minimal prime number satisfying $\binom{f}{2} < B$)
  - SAFER provides only $f$ usable configurations.

- Aegis has only $B$ groups in a configuration.
  - SAFER has $2f$ groups in a configuration.

- Aegis can have a much smaller space overhead.
## Comparison of Space Cost

To guarantee a tolerance of $f$ faults in a 512-bit data block:

<table>
<thead>
<tr>
<th>$f$ (# of faults)</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECP</td>
<td>11</td>
<td>21</td>
<td>31</td>
<td>41</td>
<td>51</td>
<td>61</td>
<td>71</td>
<td>81</td>
<td>91</td>
<td>101</td>
</tr>
<tr>
<td>SAFER</td>
<td>1</td>
<td>7</td>
<td>14</td>
<td>22</td>
<td>35</td>
<td>55</td>
<td>91</td>
<td>159</td>
<td>292</td>
<td>552</td>
</tr>
<tr>
<td>Aegis</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>27</td>
<td>28</td>
<td>34</td>
<td>43</td>
<td>53</td>
</tr>
</tbody>
</table>
Aegis-rw: Tolerate More faults

Assume we know the distinction between stuck-at-wrong and stuck-at-right faults before the actual write.

Use a fail cache to record fault locations and stuck-at values.

Aegis-rw: allowing multiple W faults or R faults in a group.
- Only $f_w \times f_r + 1$ partition configurations are required.
Experimental Setup

- Cell lifetime follows the normal distribution with a mean lifetime of $10^8$ writes and a 25% coefficient of variance.

- A perfect wear leveling is assumed.

- A cell has a 50% probability to be updated in serving a write request.

- Compare with ECP, SAFER, and RDIS. SAFER may use a cache to avoid the second writes.

- We continuously issue page (4KB) writes to a 8MB PCM memory until all memory blocks are dead.
Average Number of Recoverable Faults in a 4KB Page

Faults: 293 vs. 711

<table>
<thead>
<tr>
<th></th>
<th>256 bits</th>
<th>512 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECP4</td>
<td>37</td>
<td>41</td>
</tr>
<tr>
<td>ECP5</td>
<td>46</td>
<td>51</td>
</tr>
<tr>
<td>ECP6</td>
<td>55</td>
<td>61</td>
</tr>
<tr>
<td>SAFER16</td>
<td>31</td>
<td>55</td>
</tr>
<tr>
<td>SAFER32</td>
<td>50</td>
<td>91</td>
</tr>
<tr>
<td>RDIS-3</td>
<td>64</td>
<td>159</td>
</tr>
<tr>
<td>Aegis 12x23</td>
<td>28</td>
<td>96</td>
</tr>
<tr>
<td>Aegis 9x31</td>
<td>36</td>
<td>28</td>
</tr>
<tr>
<td>Aegis 17x31</td>
<td>67</td>
<td>36</td>
</tr>
</tbody>
</table>

-31-
Average Number of Recoverable Faults in a 4KB Page

Faults: 465 vs. 711

The graph shows the average number of recoverable faults for different memory technologies and configurations, comparing 256 bits and 512 bits. The specific values are highlighted for comparison.
Average Number of Recoverable Faults in a 4KB Page

Faults: 264 vs. 474

- ECP4: 37
- ECP5: 46
- ECP6: 55
- SAFER16: 31
- SAFER32: 50
- RDIS-3: 64
- Aegis 12x23: 28
- Aegis 9x31: 36
- Aegis 23x23: 55
- Aegis 17x31: 61
- Aegis 9x61: 91

256 bits: 0, 100, 200, 300, 400, 500, 600, 700, 800
512 bits: 0, 100, 200, 300, 400, 500, 600, 700, 800
Improvement of 4KB-page's Lifetime

Space overhead < 12.5%

<table>
<thead>
<tr>
<th></th>
<th>256 bits</th>
<th>512 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECP4</td>
<td>37%</td>
<td>67%</td>
</tr>
<tr>
<td>ECP5</td>
<td>46%</td>
<td>96%</td>
</tr>
<tr>
<td>ECP6</td>
<td>55%</td>
<td>28%</td>
</tr>
<tr>
<td>SAFER16</td>
<td>31%</td>
<td>91%</td>
</tr>
<tr>
<td>SAFER32</td>
<td>50%</td>
<td>61%</td>
</tr>
<tr>
<td>RDIS-3</td>
<td>64%</td>
<td>55%</td>
</tr>
<tr>
<td>Aegis 12x23</td>
<td>28%</td>
<td>61%</td>
</tr>
<tr>
<td>Aegis 9x31</td>
<td>36%</td>
<td>91%</td>
</tr>
<tr>
<td>ECP4</td>
<td>41%</td>
<td>28%</td>
</tr>
<tr>
<td>ECP5</td>
<td>51%</td>
<td>36%</td>
</tr>
<tr>
<td>ECP6</td>
<td>64%</td>
<td>91%</td>
</tr>
<tr>
<td>SAFER32</td>
<td>96%</td>
<td>67%</td>
</tr>
<tr>
<td>SAFER128</td>
<td>159%</td>
<td>96%</td>
</tr>
<tr>
<td>RDIS-3</td>
<td>115%</td>
<td>28%</td>
</tr>
<tr>
<td>Aegis 23x23</td>
<td>67%</td>
<td>91%</td>
</tr>
<tr>
<td>Aegis 17x31</td>
<td>67%</td>
<td>67%</td>
</tr>
<tr>
<td>Aegis 9x61</td>
<td>67%</td>
<td>67%</td>
</tr>
</tbody>
</table>
# Improvement of 4KB-page's Lifetime

<table>
<thead>
<tr>
<th></th>
<th>256 bits</th>
<th>512 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECP4</td>
<td>37</td>
<td>67</td>
</tr>
<tr>
<td>ECP5</td>
<td>46</td>
<td>67</td>
</tr>
<tr>
<td>ECP6</td>
<td>55</td>
<td>67</td>
</tr>
<tr>
<td>SAFER16</td>
<td>31</td>
<td>159</td>
</tr>
<tr>
<td>SAFER32</td>
<td>50</td>
<td>96</td>
</tr>
<tr>
<td>RDIS-3</td>
<td>28</td>
<td>28</td>
</tr>
<tr>
<td>Aegis 12x23</td>
<td>36</td>
<td>36</td>
</tr>
<tr>
<td>Aegis 9x31</td>
<td>28</td>
<td>36</td>
</tr>
<tr>
<td>Aegis 17x31</td>
<td>28</td>
<td>36</td>
</tr>
<tr>
<td>Aegis 9x61</td>
<td>28</td>
<td>36</td>
</tr>
</tbody>
</table>

**Lifetime Improve.: 6.3X vs. 8.3X**
Each Bit’s Contribution to the Lifetime Improvement

Higher than any non-Aegis schemes
Survival Rate of a 4KB-page

Agies17x31 allows 16% more writes than SAFER32 of similar group count.
Survival Rate of a 4KB-page

Agies9x61 uses only 42% overhead bits and doesn’t use cache (compare to SAFER128-cache)
Compare Aegis with Aegis-rw

![Graph showing the average number of recoverable faults for different schemes: Aegis and Aegis-rw. The data points are as follows:
- 23 x 23: Aegis 28, Aegis-rw 28
- 17 x 31: Aegis 36, Aegis-rw 67
- 9 x 61: Aegis 67, Aegis-rw 78
- 8 x 71: Aegis 78, Aegis-rw 78]
Conclusions

- To meet the demand on PCM’s high fault tolerance, Aegis effectively separates many faults in different groups for inversion-based recovery.

- To minimize space overhead, Aegis provides a large number of partition configurations and a small number of groups in each configuration.

- Extensive experiments show Aegis provides substantially higher fault tolerance, longer lifetime, and lower cost.