An Efficient Wear-level Architecture using Self-adaptive Wear Leveling

Jianming Huang, Yu Hua, Pengfei Zuo, Wen Zhou, Fangting Huang

Huazhong University of Science and Technology

ICPP 2020
Non-volatile Memory

NVM features
- Non-volatility
- Large capacity
- Byte-addressability
- DRAM-scale latency

NVM drawbacks
- Limited endurance
- High write energy consumption

Intel Optane DC Persistent Memory
Multi-level Cell NVM

The MLC technique has been used in different kinds of NVM, including PCM, RRAM, and STT-RAM.

Compared with SLC NVM, MLC NVM:
- Higher storage density 😊
- Lower cost 😊
- Comparable read latency 😊
- Weaker endurance 😞

10^7 of SLC PCM vs 10^5 of MLC PCM

Wear-leveling schemes are necessary and important.
Wear-leveling Schemes

- Table based wear-leveling scheme (TBWL)
- Algebraic based wear-leveling scheme (AWL)
- Hybrid wear-leveling scheme (HWL)
# Wear-leveling Schemes

- **Table based wear-leveling scheme (TBWL)**

<table>
<thead>
<tr>
<th>LA</th>
<th>PA</th>
<th>WC</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>150</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>519</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>115</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>210</td>
</tr>
</tbody>
</table>

Trigger wear-leveling

- **Segment Swapping**

<table>
<thead>
<tr>
<th>LA</th>
<th>PA</th>
<th>WC</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>150</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>116</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>521</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>210</td>
</tr>
</tbody>
</table>
Wear-leveling Schemes

- Table based wear-leveling scheme (TBWL)
- Algebraic based wear-leveling scheme (AWL)

**Table based wear-leveling scheme (TBWL)**

- Algebraic based wear-leveling scheme (AWL)

**Algebraic based wear-leveling scheme (AWL)**

**Region-based Start-Gap (RBSG)**
Wear-leveling Schemes

- Table based wear-leveling scheme (TBWL)
- Algebraic wear-leveling scheme (AWL)
- Hybrid wear-leveling scheme (HWL)

### Step 1
Read Out

NVM

- prn0
  - A
  - B
  - C
  - D
- prn2
  - E
  - F
  - G
  - H

### Step 2
Line Shift

SRAM (Memory Controller)

- A
  - E
  - B
  - F
  - C
  - G
  - D
  - H

### Step 3
Write Back

SRAM (Memory Controller)

- D
  - H
  - C
  - G
  - B
  - F
  - A
  - E

Write Back to NVM

- prn0
  - H
  - G
  - C
  - F
  - D
  - E

- prn2
  - D
  - C
  - B
  - A
RAA Attack for TBWL

RAA attack
- Repeated Address Attack (RAA)
  - Repeatedly write data to the same address

The lifetime of TBWL under RAA attack
- One region contains the limited lines
  - All lines in one region are repeatedly written
  - NVM is worn out at the early stage
    - \((The \ number \ of \ lines \ within \ a \ region) \times (The \ endurance \ of \ a \ line)\)
BPA Attack for AWL

BPA attack

- Birthday Paradox Attack (BPA)

- Randomly select logical addresses and repeatedly write to each
BPA Attack for AWL

- BPA attack
- The lifetime of AWL under BPA attack
  - Lifetime is low
The lifetime of HWL under BPA attack

- Smaller wear-leveling granularity increases the NVM lifetime
Problems of Existing Wear-leveling Schemes

- TBWL and AWL fail to defend against attacks
  - RAA attack leads to low lifetime in TBWL
  - BPA attack leads to low lifetime in AWL
Problems of Existing Wear-leveling Schemes

- TBWL and AWL fail to defend against attacks

- HWL obtains high lifetime with small granularity
  - The large granularity leads to low lifetime
  - The small granularity leads to high lifetime
Problems of Existing Wear-leveling Schemes

- TBWL and AWL fail to defend against attacks
- HWL obtains high lifetime with small granularity
- The cache hit ratio of HWL is affected by the granularity
  - The wear-leveling entries stored on cache are limited
  - Entries with large granularity cover large NVM $\rightarrow$ high cache hit ratio
  - Entries with small granularity cover small NVM $\rightarrow$ low cache hit ratio
Problems of Existing Wear-leveling Schemes

- TBWL and AWL fail to defend against attacks
- HWL obtains high lifetime with small granularity
- The cache hit ratio of HWL is affected by the granularity
- High performance and lifetime are in conflict

How to address the conflict between the lifetime and performance is important
SAWL: self-adaptive wear-leveling scheme

High hit ratio & unbalanced write distribution  Split regions to decrease the granularity

achieve high lifetime and performance

Merge regions to increase the granularity  Low hit ratio & uniform write distribution
Architecture of SAWL

- **IMT (translation lines)**: record the locations in which the user data are actually stored

  - **wear-leveling the data lines**

<table>
<thead>
<tr>
<th>Region</th>
<th>Data lines</th>
<th>Translation lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>line 0</td>
<td>line 0</td>
</tr>
<tr>
<td></td>
<td>line 1</td>
<td>line 1</td>
</tr>
<tr>
<td></td>
<td>line 2</td>
<td>line 2</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>1</td>
<td>line 0</td>
<td>line 0</td>
</tr>
<tr>
<td></td>
<td>line 1</td>
<td>line 1</td>
</tr>
<tr>
<td></td>
<td>line 2</td>
<td>line 2</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>N</td>
<td>line 0</td>
<td>line 0</td>
</tr>
<tr>
<td></td>
<td>line 1</td>
<td>line 1</td>
</tr>
<tr>
<td></td>
<td>line 2</td>
<td>line 2</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>M</td>
<td>line 0</td>
<td>line 0</td>
</tr>
<tr>
<td></td>
<td>line 1</td>
<td>line 1</td>
</tr>
<tr>
<td></td>
<td>line 2</td>
<td>line 2</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- **Global Translation Directory (GTD)**

<table>
<thead>
<tr>
<th>tstma</th>
<th>tpsma</th>
<th>key</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- **Cached Mapping Table (CMT)**

<table>
<thead>
<tr>
<th>lrn</th>
<th>wlg</th>
<th>prn</th>
<th>key</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- **Integrated Mapping Table (IMT)**

<table>
<thead>
<tr>
<th>tpma</th>
<th>(prn, key)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(2,4),(4,5),..., (15,6)</td>
</tr>
<tr>
<td>1</td>
<td>(6,3),(5,5),..., (18,7)</td>
</tr>
<tr>
<td>2</td>
<td>(8,2),(10,7),..., (3,6)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- **Memory Controller**

- **Address Translation**

- **Region Split/Merge**

- **Data Exchange**

- **Cached Mapping Table (CMT)**

- **Integrated Mapping Table (IMT)**

- **Global Translation Directory (GTD)**

- **Data lines**

- **Translation lines**

- **Sync**

(NVM) (SRAM) (DRAM)
• CMT: buffer the recently used IMT entries
  reduce translation latency
Architecture of SAWL

- GTD: record the relationships between logical/physical addresses of translation lines
  → wear-leveling the translation lines
Operations in SAWL

**Merge the region**

1. Choose two unmerged neighborhood logical regions.

2. Physically exchange the data to satisfy the algebraic mapping between the logical and physical regions.

3. Update the relevant CMT entries on the SRAM and the IMT table on the NVM.
Operations in SAWL

- Merge the region
- Split the region

1. Logically split the region without data move. The data have already satisfied the algebraic mapping.

2. Update the CMT entries and IMT table.
When to Adjust the Region

- We use the hit ratio as the trigger to merge/split region
  - Hit ratio below 90% significantly decreases the performance
  - Hit ratio above 95% slightly impacts on the performance

hit ratio $\geq 95\% \quad \rightarrow \quad$ Split the regions

$\downarrow$

Merge the regions $\downarrow$

hit ratio $\leq 90\%$
Parameter in SAWL

- **SOW**: size of the observation window
  - Small SOW $\rightarrow$ cache hit rate frequently fluctuates

![Graph showing cache hit rate against runtime](image)
Parameter in SAWL

- **SOW**: size of the observation window
  - Small SOW → cache hit rate frequently fluctuates
  - Large SOW → systems miss important trigger point
Parameter in SAWL

- **SOW**: size of the observation window

  - Small SOW -> cache hit rate frequently fluctuates
  - Large SOW -> systems miss important trigger point
  - We use $2^{22}$ as the SOW value

![Graph showing cache hit rate over runtime](graph.png)

- SOW = $2^{22}$
Parameter in SAWL

- **SSW**: size of the settling window
  - Small SSW -> frequently adjust region size
Parameter in SAWL

SSW: size of the settling window

- Small SSW -> frequently adjust region size
- Large SSW -> fail to sufficiently adjust the region size
SSW: size of the settling window

- Small SSW --> frequently adjust region size
- Large SSW --> fail to sufficiently adjust the region size
- We use $2^{22}$ as the SSW value

SSW = $2^{22}$
Experimental Setup

- **Configuration of simulated system via Gem5**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>8 cores, X86-64 processor, 3.2 GHz</td>
</tr>
<tr>
<td>Private L1 cache</td>
<td>64KB</td>
</tr>
<tr>
<td>Shared L2 cache</td>
<td>512KB</td>
</tr>
<tr>
<td>CMT cache</td>
<td>256KB</td>
</tr>
<tr>
<td>DRAM/PCM Capacity</td>
<td>128MB/8GB</td>
</tr>
<tr>
<td>Read/Write latency model</td>
<td>DRAM 50/50ns, PCM 50/350ns</td>
</tr>
<tr>
<td>Address translation latency</td>
<td>Cache hit 5ns, Cache miss 55ns</td>
</tr>
</tbody>
</table>

- **Comparisons**
  - Baseline: an NVM system without wear-leveling scheme.
  - NWL-4: naive wear-leveling scheme with a region consisting of 4 memory lines.
  - NWL-64: naive wear-leveling scheme with a region consisting of 64 memory lines.
  - AWL schemes: RBSG and TLSR.
  - HWL schemes: PCM-S and MWSR.

- **Benchmark: SPEC2006**
SAWL has high cache hit rate, and adjusts the region size according to the hit rate.
Smaller swapping period increases the NVM lifetime at the cost of high write overhead. SAWL efficiently defends against the BPA attack.
SAWL achieves high lifetime in all applications.
Conclusion

- Existing wear-leveling schemes fail to efficiently work on MLC NVM

- SAWL dynamically tunes the wear-leveling granularity
  - Low cache hit ratio with uniform write distribution leads to merging regions
  - High cache hit ratio with unbalanced write distribution leads to splitting regions

- SAWL achieves high lifetime under attacks and general applications with low performance overhead
Thanks! Q&A

Email: jmhuang@hust.edu.cn