Designing Endurable PCMs: Balancing Bit Flips and Exploiting Multi-Level Storage Capability

Prof. Hamid Sarbazi-Azad
High-Performance Computing Architectures & Networks (HPCAN) Lab.
Sharif University of Technology, and
Institute for Research in Fundamental Sciences (IPM)
Tehran, Iran
Memory Challenges in CMPs

- Memory wall
  - Performance gap between processor and memory
  - ~14% per technology node
  - Exacerbated in multi-core era
  - Affected by program sizes

"Moore’s Law"

Processor-Memory Performance Gap: (grows 50% / year)
Memory Challenges in CMPs

- **Goal:** Having a memory system (cache+main memories) holding the working set of running programs.
  - It is far from reality
    - Multi-level caches with few MBs capacity
    - Few GBs main memories
  - SRAM or DRAM as baseline technology
- **Conventional SRAM/DRAM memories face challenges**
  - Energy consumption of memory system is becoming dominant
  - Leakage power as high as dynamic power
  - Scalability and reliability challenges
Why Non-Volatile Memories?

- Seeking for alternative technologies
  - Scalable and dense
  - Low-power (at least power friendly)
  - High performance
  - Highly reliable
  - CMOS compatible
  - RAM and byte-addressable
Non-Volatile Memories

- NV memory technologies have advantages of
  - Non-volatility (low standby power)
  - High density
  - Better scalability than conventional memories
  - Good resilience to soft-errors
  - Acceptable read performance and energy
  - CMOS compatible (partially)

- But suffer from disadvantages of
  - Short write endurance (10^5 to 10^{12} writes)
  - High write energy per cell
  - Long write latency
Non-Volatile Memory Hierarchy

Spin Torque Transistor RAM
- High-density: viable MLCs
- Fastest NV memory technology
- Random reads/writes
- Acceptable endurance (~ $10^{12}$)

Phase Change Memory
- High-density: viable MLCs
- Random reads/writes
- Challenging writes
- Low endurance (~ $10^8$)

Flash memory
- High-dense: viable MLCs
- Random reads
- Block-level writes (erase before writes)
- Low endurance (~ 105)
Example: Main Memory Level

DRAM Scaling Problem

• DRAM is not scalable [ITRS reports]
  – Capacitor must be large enough for reliable sensing
  – Access transistor should be large enough for low-leakage and high-retention time

• Energy/power is a key system design concern
  – ~40-50% of energy spent in off-chip memory hierarchy
  – DRAM consumes power even when not used (periodic refresh)

• Reliability concerns in DRAM technology
  – Single/multiple event(s) upset
  – Row-hammer problem

→ DRAM cannot further scale from capacity, cost, performance, energy, and power points of view.
Refresh Performance Overhead

Refresh Energy Overhead

Beside SEU/MEUs, there are other reliability issues including the Row-Hammer Problem.

Repeatedly opening and closing a row enough times within a refresh interval induces disturbance errors in adjacent rows in most real DRAM chips you can buy today.

Phase Change Memory (PCM) has none of these problems 😊

- **Data representation (physical state of PCM)**
  - Crystalline state (low resistance) → “1” logic
  - Amorphous state (high resistance) → “0” logic

- **Writing a cell (heating)**
  - Heat GST with 300°C~ 400°C over a moderately long period of time → Crystalline state
  - Above 600°C, melt the GST and then cool it down quickly → Amorphous state
  - Program-and-Verify → Partial amorphous

- **Reading a cell (determining its resistance level)**
  - Binary Search scheme or Sweeping scheme
PCM details

Amorphous State
High-Resistance

Crystalline State
Low-Resistance
PCM Read/Write Circuit

Read Circuit

- Read En.
- Read binary data

Write Circuit

- Set En.
- Reset En.

Top electrode
- GST
- Programmable region
- Resistor
- Insulator
- Bottom electrode

Reference current (from a dummy cell)

HPCAN Lab., SUT & IPM, Tehran, Iran.
MLC PCM Capability

- Multi-Level Cell \(\rightarrow\) Increases capacity in same area!
- Different pulses for read/write

**Read** (2-bit MLC PCM): passing the currents to the cell and compare with reference voltages

**Write**: repetitive P&V process:
PCM Reliability Issues

• **Soft Error (resistance drift)**
  - Meta-stable nature of the amorphous phase: changing the resistance value with temperature and time

• **Hard Error (cell wear-out)**
  - Repeated heating and cooling for SET/RESET/PROGRAM operations damage the junction applying the electrical current
  - Permanently stuck cell at SET/RESET value (0 or 1)
3D Integration and PCM

- 3D memory integration is promising
  - High bandwidth
  - Low latency
  - Low energy
  - Capability of integrating diverse technologies

- Increased power and temperature
  - Increases SRAM leakage power super-linearly
  - Increases DRAM refresh power by 2X
  - Makes writes to PCM easier due to its thermal-friendly nature

- Future memory system of CMPs:
  - LLC is made of STT-RAM and stacked above processor layer
  - Main memory is made of PCM and stacked on top of processor chip
Summary:

PCM as main memory

- Non-volatility
- High density (especially in MLC PCM)
- CMOS compatible process
- Highly scalable and suitable for 3D design

- Limited write endurance (~$10^8$ writes in SLC and ~$10^6$ in MLC PCMs)
- High write latency and energy
- Resistance drift (soft error) in MLC PCMs
How to extend PCM lifetime

Techniques to improve PCM lifetime can be grouped into 4 categories:

Category 1: Reducing number of writes
Category 2: Reducing bit-flips pressure
Category 3: Wear-leveling
Category 4: Tolerating hard errors
We use a shift-based mechanism and MLC capability of PCMs to tolerate more hard errors...

Outline

• Non-uniformity in Bit-Flip Distribution
• Our Byte-level Shifting Scheme (BLESS)
  • Improving Bit-Flips Uniformity
  • Tolerating Hard Errors
• Evaluation
• Conclusion
Bit-Flips Non-uniformity

• Most previous proposals assumed a uniform bit-flips distribution among all cells. But simulations show an extensive non-uniformity in bit-flips during write operations:

→ We need methods to reduce this non-uniformity, hence improve the lifetime
Improving Bit-Flips Uniformity

• Using a shift-based mechanism for each block, we rotationally shift the content of a block by one unit upon each write and using a counter to keep track of shifts.

**Example:** For a 64-unit block (using a 6-bit counter):

\[ \begin{array}{c}
\text{Counter} & \text{Data Block} \\
000000 & \begin{array}{cccccccc}
\text{Unit 63} & \text{Unit 62} & \text{Unit 61} & \cdots & \text{Unit 2} & \text{Unit 1} & \text{Unit 0} \\
\end{array} \\
\end{array} \]

\[ \text{first write} \]

\[ \begin{array}{c}
\text{Counter} & \text{Data Block} \\
000001 & \begin{array}{cccccccc}
\text{Unit 62} & \text{Unit 61} & \cdots & \text{Unit 2} & \text{Unit 1} & \text{Unit 0} & \text{Unit 63} \\
\end{array} \\
\end{array} \]

\[ \text{second write} \]

\[ \begin{array}{c}
\text{Counter} & \text{Data Block} \\
000010 & \begin{array}{cccccccc}
\text{Unit 61} & \cdots & \text{Unit 2} & \text{Unit 1} & \text{Unit 0} & \text{Unit 63} & \text{Unit 62} \\
\end{array} \\
\end{array} \]

→ More bit-flip uniformity at each memory block
Tolerating Hard Errors

• Using the same counter we used for balancing the bit-flips, we implement a pairing mechanism that can recover the data stored in a partially faulty block.

• We do so by combining the original data block with its shifted form and storing them together in MLC mode.

• When reading, we can recover the original data block by using the stored data and its shifted form.
Problem Formulation

The $n$-bit binary vector $V = v_{n-1}v_{n-2}... v_2v_1v_0$ defines the health status of an $n$-unit memory block, where $v_i = 0$ if the corresponding $i$-th unit of the block is healthy and $v_i = 1$ when it is faulty.

**Definition 1.** Two vectors $U$ and $V$ are compatible if

$$U \otimes V = u_{n-1}v_{n-1} + u_{n-1}v_{n-1} + u_{n-2}v_{n-2}+...+ u_1v_1 + u_0v_0 = 0.$$ 

Otherwise, they are called incompatible.

It is clear that:

- Any vector $U$ is compatible with null vector $Z = 00...0$;
- Any vector $V$ is compatible with its health-complement vector $\overline{V} = \overline{v_{n-1}}\overline{v_{n-2}}\cdots\overline{v_1}\overline{v_0}$
- Any non-null vector $U \neq Z$ is incompatible with the completely-faulty vector $I = 11...1$. 

Problem Formulation (cont.)

Definition 2. The $i$-position rotated vector $V^{\rightarrow_i}$ is defined as

$$V^{\rightarrow_i} = v_{i-1}v_{i-2}...v_0v_{n-1}v_{n-2}...v_i.$$ 

Definition 3. Vector $V$ is called $i$-self-compatible if $V$ is compatible with $V^{\rightarrow_i}$.

Definition 4. Vector $V$ is called self-compatible if there exist an $i$, $0 \leq i \leq n-1$, for which $V$ is $i$-self-compatible.

It is clear that: If Haming_Weigth($V$) > $n/2$, then $V$ cannot be self-compatible.

Let $F_j$ be the set of all vectors with Hamming weight $j$, and $C_j \subset F_j$ be the set of such vectors that are self-compatible. The probability that a vector with Hamming weight $j$ is self-compatible is given by:

$$P_j = \frac{|C_j|}{|F_j|} = \frac{|C_j|}{\binom{n}{j}}$$
Let $i_{\min}(V)$ be the minimum value of $i$, $0 \leq i \leq n-1$, for which $V$ is $i$-self-compatible.

The average number of shifts to recover a block with $j$ faulty units is given by:

$$i_{avg} = \frac{\sum_{V \in F_j} i_{\min}(v)}{|F_j|}$$

Figure: $P_j$ as a function of the number of faulty units in a 16-unit block (left) and the average number of shifts to recover in such a block (right).
Example: Writing a block

Suppose a memory of 4-unit 32-bit blocks.

After many writes to the block, some cells of the block become faulty. Using read-after-write mechanism, it is detected that these cells are in Unit 1 and Unit 2.

Fault-vector: \(<0\ 1\ 1\ 0>\)

Fault-Vector \(<0110>\) is 2-self-compatible as: \(0110 \oplus 1001 = 0000\)

Assuming a data block: \(11010100\ 10101011\ 00010101\ 00001110\)

and 2-unit shifted form: \(00010101\ 00001110\ 11010100\ 10101011\)

we write the original data block and its shifted form together into the memory block in MLC mode as:
Example: Recovering the block

The original data block 11010100 10101011 00010101 00001110 and its 2-unit shifted form are written in MLC mode as:

After reading the block in MLC mode, we can easily recover the original data block using the shift counter and the read data from the memory as (using data stored in Unit 0 and Unit 3):

The original data block 😊
Write Operation of Faulty Blocks (summary)

- Determining the byte-level fault-mask of the block
- Shifting the fault-mask (here a maximum of 8 shifts) and checking if it is self-compatible.

If so,
  - Converting the storage type of block to MLC and
  - Sending the original data and its shifted compatible to memory write circuit to realize an MLC write of the combined data blocks onto the block cells

If not:
  - Block is marked as "permanently faulty"
  - Exclude the faulty block from the memory space
Read Operation (summary)

Checking if the block is stored in MLC/SLC mode:

If in SLC mode:
- Block is healthy: realizing one normal SLC read, apply shifting according to the shift-counter value to form the original data block and delivering it to the memory controller

If in MLC mode:
- Read the content of block in MLC mode
- Obtaining the correct content based on its fault-mask
- Ignoring the faulty bit positions
- Retrieving the bits from healthy cells and forming the original data block and passing it to the memory controller
Simulation Environment

- **System-level simulation**
  - GEM-5 full-system simulator
  - Benchmarks: ROI of the PARSEC-2 programs

- **Evaluated architectures:** ECP-6, SAFER-32 and Aegis-23x23

- **Specification of the simulated system**

<table>
<thead>
<tr>
<th>Component</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td>4-core ALPHA21264, 2.0GHz.</td>
</tr>
<tr>
<td>L1 Cache</td>
<td>Split I and D cache; 32KB private; 4-way; 64B line size; LRU; write-back;</td>
</tr>
<tr>
<td></td>
<td>1 port; 2ns latency</td>
</tr>
<tr>
<td>L1 Coherency</td>
<td>MOESI directory; 4×2 grid packet switched NoC; XY routing; 3 cycle</td>
</tr>
<tr>
<td></td>
<td>router; 1 cycle link</td>
</tr>
<tr>
<td>L2 Cache</td>
<td>4MB, UCA shared; 16-way; 64B line size; LRU; write back; 8 ports; 4ns</td>
</tr>
<tr>
<td></td>
<td>latency</td>
</tr>
<tr>
<td>DRAM Cache</td>
<td>16MB; 4-way; 64B line size; LRU; write-back; 8 ports; 26ns latency</td>
</tr>
<tr>
<td>Off-Chip Main Memory</td>
<td>8GB; 16 banks; 64 B; open page, SLC: Read Latency 80 ns (6ns tPRE +</td>
</tr>
<tr>
<td></td>
<td>69ns tSENSE + 5ns tBUS); Write Latency 250ns.</td>
</tr>
<tr>
<td>Flash SSD</td>
<td>Unlimited size; 25μs latency</td>
</tr>
</tbody>
</table>
Lifetime Evaluation

Lifetime indicator is the number of writes which reduces the memory capacity to 50% of its initial capacity.

- Improving lifetime by 15-25% over state-of-the-art line-level schemes
Number of Required Shifts/Recovered Errors

More than 60% of recovered faulty lines are recovered by only one shift and about 2% of them are recovered by 8 shifts.

Average number of recovered errors per page for different schemes (BLESS works better than others, on average).
The Impact on System Performance

Considering three working intervals:

- When no error has occurred
- When the first hard error occurs in Aegis
- When the first hard error occurs in BLESS
Bit-flips Uniformity

Amount of reduction in bit-flips for different workloads can be evaluated using *IntraV* equation:

\[
IntraV = \frac{1}{BF_{\text{aver}} \cdot N} \times \sum_{i=1}^{N} \sqrt{\frac{\sum_{j=1}^{512} (BF_{ij} - \sum_{j=1}^{512} w_{ij}/512)^2}{511}}
\]

→ BLESS can reduce the variation of bit-flips in each block by 27% on average.

<table>
<thead>
<tr>
<th>Workload</th>
<th>Baseline</th>
<th>BLESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blackscholes</td>
<td>209</td>
<td>166</td>
</tr>
<tr>
<td>Caneal</td>
<td>83</td>
<td>30</td>
</tr>
<tr>
<td>Facesim</td>
<td>6.5</td>
<td>4.3</td>
</tr>
<tr>
<td>Fluidanimate</td>
<td>14.1</td>
<td>13.2</td>
</tr>
<tr>
<td>Raytrace</td>
<td>73</td>
<td>67</td>
</tr>
<tr>
<td>Swaptions</td>
<td>236</td>
<td>166</td>
</tr>
<tr>
<td>X264</td>
<td>73</td>
<td>63</td>
</tr>
<tr>
<td>Bodytrack</td>
<td>132</td>
<td>128</td>
</tr>
<tr>
<td>Dedup</td>
<td>56</td>
<td>30</td>
</tr>
<tr>
<td>Ferret</td>
<td>39</td>
<td>28.8</td>
</tr>
<tr>
<td>Freqmine</td>
<td>173</td>
<td>104</td>
</tr>
<tr>
<td>Streamcluster</td>
<td>39</td>
<td>31</td>
</tr>
<tr>
<td>Vips</td>
<td>24</td>
<td>11.2</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>89.04</strong></td>
<td><strong>64.80</strong></td>
</tr>
</tbody>
</table>
Impact of Unit Size

• Meta-data storage size for each line of 64B (8-bit unit):
  – We need:
    1) one SLC/MLC mode bit,
    2) a 6-bit counter, and
    3) one bit per byte to indicate its parity bit

• A total of 1+6+64=71 bits per block, that is 71/512 = 13.8% storage overhead.
Conclusion

- We proposed the Byte-LEvel Shifting Scheme (*BLESS*) to improve lifetime of PCM
- Compared to state-of-the-art line-level schemes it improved the lifetime by 15%-25%
- Compared to page-level schemes, DRM and PAYG, the average lifetime is improved by 17% and 14%, respectively
- Note that *BLESS* is orthogonal to most previous schemes which work at line or page level, so it can be used with them to further improve their efficiency
Thank you.

Questions?