Increasing Endurance and Security of Phase-Change Memory with Multi-Way Wear-Leveling

Hongliang Yu and Yuyang Du

Abstract—Phase-change memory (PCM) is a promising alternative of DRAM. Nonetheless, it has a well-known problem that is the limited number of writes to storage cells. Thus, wear-leveling, which makes the writes uniform, is crucial to boost PCM's lifetime. This paper proposes *multi-way wear leveling* (MWWL) to increase both endurance and security of PCM. MWWL can efficiently distribute writes to physical addresses uniformly from a multiple of ways while incurring little write overhead and almost no extra hardware overhead. More important, MWWL is a fundamental scheme that can be applied to existing leveling algorithms. As a case study, we extended a state-of-the-art technique, Security Refresh, to its multi-way version, Multi-Way Security Refresh (MWSR). The experimental results show that MWSR can achieve the same or better lifetime than that of the original two-level Security Refresh but with much less write overhead (from 11.7% down to 1.5%).

Index Terms—Phase-change memory, wear-leveling, endurance, security, lifetime, write overhead

1 Introduction

Dynamic Random Access Memory (DRAM) has been the building block for computer systems during the past forty years. As DRAM faces increasingly severe scalability and power consumption issues, Phase-Change Memory (PCM) that employs electrical resistivity to store data has emerged as a promising alternative to replace DRAM. PCM has a number of good properties such as non-volatility, energy saving, high density, and sustainable scalability.

However, as PCM's storage cell is capable of enduring a limited number of writes, the wear-leveling mechanism must be applied to prevent some cells from being worn out sooner than the others so that the overall lifetime is increased. Traditionally, an address mapping table like that used in flash memory can be employed for wear-leveling [5]. The table based wear-leveling techniques, however, are not suitable for PCM because of the intrinsic differences between PCM and flash memory. Algebraic mapping based wear-leveling [14] was consequently proposed to leverage an algebraic algorithm to calculate the mapping between the logical address and the physical address, instead of looking for the mappings in a table.

Unfortunately, as the literature [18], [20] pointed out, the algebraic mapping based wear-leveling is susceptible to malicious attacks. An adversary can continuously write to some addresses in order to wear out the cells and eventually fail the whole PCM system. To deal with this security threat, many wear-leveling schemes have been researched, such as randomized region based Start-Gap [14], multi-level Security Refresh [18], and Online Attack Detection [16], etc. However, these schemes suffer from high write overhead or extra hardware overhead, because they need to frequently swap data in order to accelerate remapping the logical address under attack to different physical addresses. The extra write overhead induced by swapping data can not only substantially increase the access latency but also wear out the storage cells. Moreover, these schemes may be subjected to "partial" leveling, since they limit the mapping of logical addresses to a sub-spaces instead of the whole memory space. Thus, the wear among different sub-spaces may be uneven.

In order to increase both endurance and security while inducing minimal write overhead or hardware cost, this paper proposes *multi-way wear-leveling* (MWWL), a novel wear-leveling scheme that can quickly distribute writes to the entire physical address space uniformly. In MWWL, we first divide the logical address space into equally sized sub-spaces; each sub-space individually undertakes the wear-leveling for the addresses in the sub-space, which is called one way of wear-leveling. Each way (sub-spaces) maps and remaps their local logical addresses to the entire physical space. The logical space is small so that the logical address can be remapped along with a small number of logical addresses being remapped. When any address is under heavy writes (such as an attack), the actually written physical address will be changed frequently. This is why MWWL can operate at very slow remapping rate. In addition, the physical space that can be remapped to by any logical address in one way is as large as the entire address space. Therefore, MWWL can achieve almost the ideal lifetime.

More important, MWWL is a fundamental scheme, which can be applied to existing leveling algorithms, such as Security Refresh and Start-Gap. In this paper, we present Multi-Way Security Refresh (MWSR), which applied the MWWL approach on top of the Security Refresh technique as a case study. We compared MWSR to the original two-level Security

- *H. Yu is with the Computer Science Department, Tsinghua University, Beijing 100084, China, and the Research Institute of Tsinghua University Shenzhen, Shenzhen 518057, China. E-mail: hlyu@tsinghua.edu.cn.*
- *Y. Du is with Intel China Ltd., Beijing 100086, China.*

Manuscript received 27 June 2012; revised 22 Oct. 2012; accepted 12 Nov. 2012; published online 9 Dec. 2012; date of current version 29 Apr. 2014.
Recommended for acceptance by J. Plusquellic.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TC.2012.292

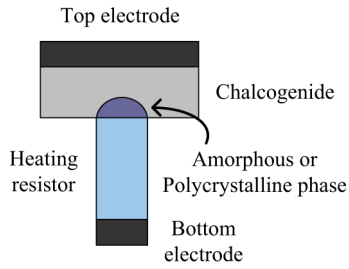


Fig. 1. Basic PCM cell structure [10].

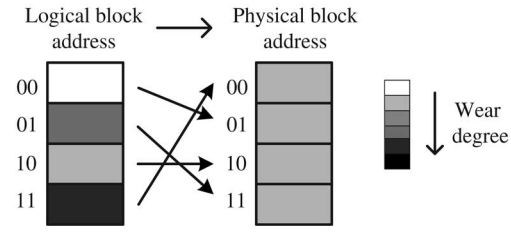


Fig. 2. Block address mapping for wear-leveling.

Refresh and showed that the MWSR offers significant advantages. MWSR can achieve the same or better lifetime (more than 80% of the ideal lifetime) than that of the two-level security refresh but with much less write overheads (from 11.7% down to 1.5%) and almost no extra hardware required. Our new mechanism is also shown better scalability than the two-level Security Refresh scheme as the sub-space scales down to smaller size.

The remainder of the paper is organized as follows. In Section 2, we briefly introduce the PCM technology, the wear-leveling mechanism, and algebraic mapping algorithms. In Section 3, we motivate our paper by analyzing existing schemes and then propose multi-way wear-leveling. Then we extend the Security Refresh to the multi-way version in Section 4 as a case study. We evaluate our scheme in Section 5. We discuss related work in Section 6 and draw the conclusion and future work in Section 7.

2 Background

2.1 Phase-Change Memory

Phase-Change Memory (PCM) [17], [23] is an emerging non-volatile memory technology. PCM employs reversible change of electrical resistivity in different phases to store data. Typically used storage material is chalcogenide, an alloy of Germanium, Antimony and Tellurium (e.g., $Ge_2Sb_2Te_5$). A PCM storage cell (Fig. 1) is usually comprised of a layer of chalcogenide sandwiched between two electrodes and a heating resistor extended from one of the electrodes to contact the chalcogenide layer [10]. Phase change of chalcogenide is induced by intense localized Joule heating. In the melted amorphous phase, the material exhibits high resistivity and reflectivity because of the disordered crystalline lattice, which can represent a binary "0". In the frozen polycrystalline phase, the chalcogenide exists in a regular crystalline structure and exhibits low reflectivity and resistivity, which can represent a binary "1" [10].

PCM offers many advantages over the massively used memories [11]. One is scalability. The ITRS projects unknown manufacturable solution for scaling DRAM under 40 ns, since a capacitor must be large enough to store charge for reliable sensing, so must be a transistor to effectively control the channel. The SIA Roadmap predicts difficulties in scaling flash beyond 20 nm, since a small number of electrons tunneling from the floating gate will cause the cell state lost. In contrast, PCM is amenable to process scaling. There is already 20 nm device prototype, and the ITRS aggressively projects to scale it to 9 nm [17]. Another advantage is low power consumption. Since DRAM must be periodically refreshed to retain data stored there due to the capacitor's leakage, it has been reported that traditional DRAM memory subsystem can consume as much as 40% of the total system power budget. In comparison, PCM consumes little energy in idle state and more importantly it is non-volatile, so potentially can be turned off when not in use to reduce power consumption.

However the PCM device is expected to be able to endure only about 10^5 to 10^9 writes per cell before a stuck-at failure occurs to the cell. The non-uniformity of memory write pattern (either on normal or extreme condition) makes the situation worse because the PCM lifetime is determined by a few frequently written cells, which can dramatically reduce the endurance of a large array of memory cells (about 30s is enough to wear out a cell [14]). One way to mitigate this threat is reducing the write number (i.e., write coalesce) [2], [8], [24]. Another way is wear-leveling, a low-level mechanism to even writes uniformly, which is the focus of our paper.

2.2 Wear-Leveling

After a limited number of writes, permanent device fault (wear out) could appear in PCM cell. The fault may occur in a new write when the heating resistor detaches from the cell due to thermal expansion and contraction [8], [17]. As a result, some cells will necessarily be worn out sooner than the others because they are written more frequently. Qureshi et al. [15] analyzed the distribution of the write traffic in pages, which exhibits significant write non-uniformity.

To distribute writes evenly for extending PCM lifetime, wear-leveling must be leveraged. Generally, wear-leveling makes use of a layer of indirection to *map* logical address to physical address where data are actually stored. Fig. 2 illustrates with an example. Although writes to logical addresses are uneven (different gray levels), the wear-leveling mechanism can *remap* the logical addresses to make the writes uniform throughout the entire physical addresses.

To enable address mapping and remapping, intuitively a table can be employed, such as that used in translation layer for the flash [5]. In the table, each entry records the physical address of a logical address. And the write number to every physical address is counted and then sorted to find the most and least worn blocks. Thus, upon write or space reclamation, the most and least worn addresses could be exchanged. As tables incur great space and time overhead, Qureshi et al. [14] proposed to use algebraic mapping to calculate the address mappings instead of looking them up in the table. Their wear-leveling scheme constantly relocates the mapped physical blocks to their neighbors for all the logical blocks every certain

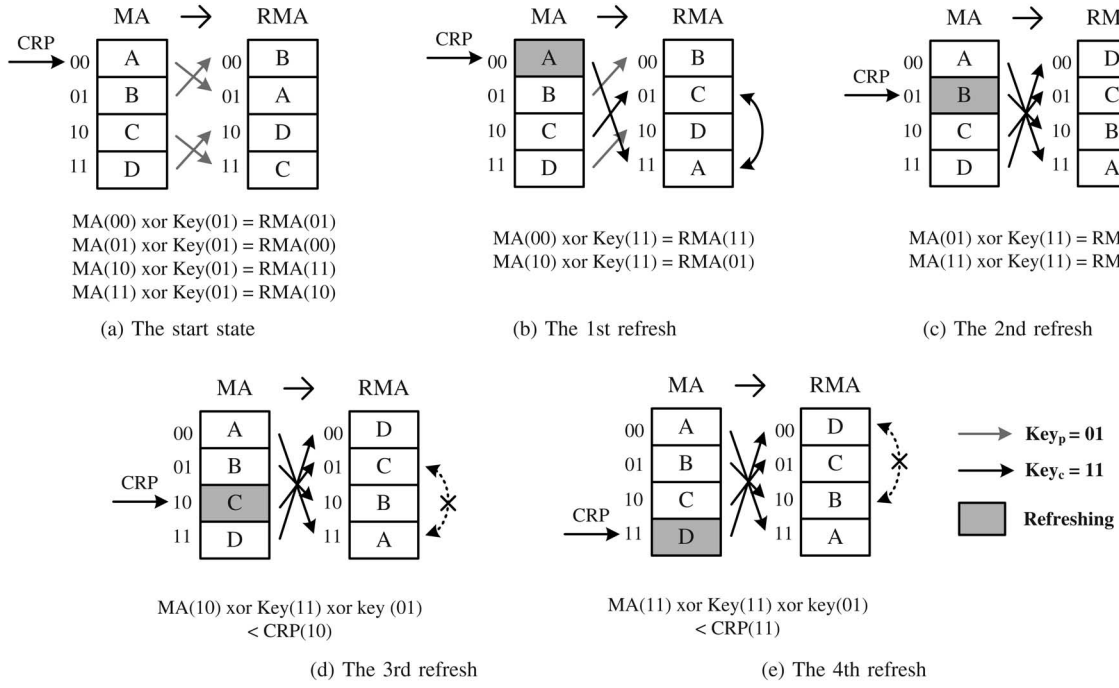


Fig. 3. An example of memory address remappings in a complete Security Refresh round.

number of normal writes (e.g., 100), irrelevant of the number of writes to the logical blocks (hence no need for tracking writes).

Algebraic mapping based wear-leveling can significantly reduce time and space complexity. Unfortunately, it is susceptible to malicious attacks [14], [18], [20]. Contriving such attacks can be as simple as approximately 10 lines of C code [14]. Essentially, the vulnerability exists because the numbers of writes to cells are not consulted to generate the remappings. Therefore, the attacker can intentionally focus on some memory addresses and repeatedly write to those addresses. [18] presented a good tutorial on how to carry out a malicious attack.

In summary, the attacks fit into three flavors (assume that the PCM cannot distinguish memory attacks from normal accesses): 1) The attacker can compromise the operating system, and thereafter infer the logical blocks that will be subsequently mapped to the same physical block based on the knowledge of the wear-leveling algorithm or side-channel information gathered from the system. We call this attack the *address inference attack* (AIA). To conquer it, the wear-leveling scheme needs to randomly remap the logical address to the physical address for confusing the attackers; 2) The adversary simply writes to the same address repeatedly, forming the *repeated address attack* (RAA) [16]. To survive RAA, the physical address mapped to the attacked logical address must be changed fast enough to level the wear. Moreover, the physical address space that the attacked address can be remapped to should be as large as possible in order that the writes can be shared sufficiently; 3) The attacker randomly selects logical addresses and repeatedly writes to each one precisely until it is remapped to another physical address. After a surprisingly small number of attempts, a memory cell is likely to be selected enough times and finally be worn out. This attack is dubbed *birthday paradox attack*[1] (BPA) [20]. To guard against BPA, the wear-leveling scheme must make the remappings so fast that the attacker must pick the same physical address for a good many times, and there should be a significant number of possible mapping candidates for the attacker to select from in order to reduce the possibility of reselecting the same address.

For algebraic mapping based schemes, however, the objectives of fast address remapping, reducing write overhead, and fully leveling all the physical space may lead to conflicts. We will elaborate on these issues in the following section. Before proposing multi-way wear-leveling, we introduce a state-of-the-art algebraic mapping algorithm, Security Refresh, and the wear-leveling schemes based on it.

2.3 Security Refresh

Security Refresh [18] (SR) is a recently proposed algebraic mapping algorithm. SR is capable of avoiding information leakage to the outside of the PCM. Like other algebraic mapping algorithms, SR refreshes[2] the physical address for one logical address every certain number of normal writes, defined as *refresh rate*. In each refresh round, the physical addresses for all the logical addresses are recalculated with a randomly generated key (since it is linear mapping). With the new address, the data at the logical address is swapped between the old physical address and the new one.

We briefly describe SR through an example. Fig. 3 illustrates a complete refresh round, which is composed of four

1. The birthday paradox captures the fact that the number of random trials required to find an item twice in a collection of items is very low. For example, to find a pair of people having the same birthday on average requires only 23 randomly picked individuals [16].

2. In Security Refresh related algorithms, we use refresh and remap interchangeably.

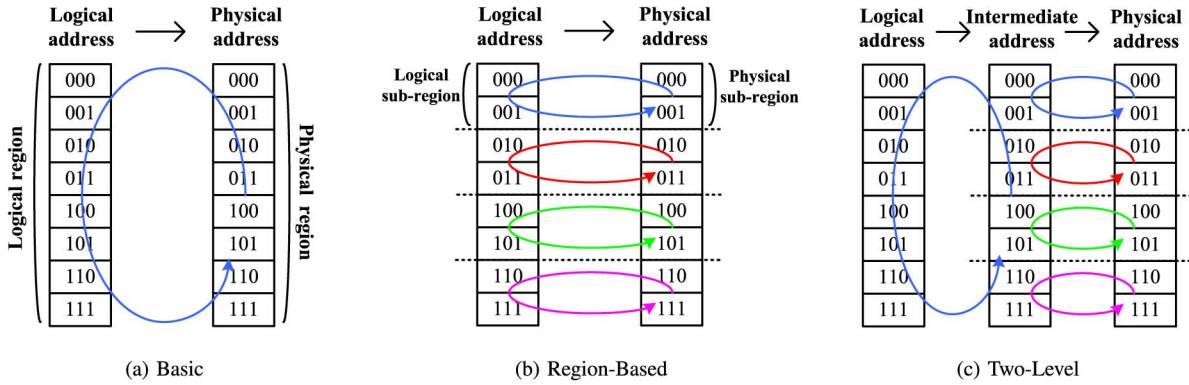


Fig. 4. Comparison of different wear-leveling schemes. The circle arrow represents the address mapping.

refreshes since there are four logical/physical blocks. Conforming to their terminology, the logical address is called *memory address* (MA) and the physical address is called *remapped memory address* (RMA). When to refresh, the RMA of the block pointed to by the current refresh pointer (CRP) will be recalculated through bitwise operation, exclusive-or (xor), expressed as $RMA = MA\ xor\ key$, where the key has the same number of bits as the address. After the data swap between the old and new physical addresses, the CRP will be incremented until the last block (then overflow and be moved to the first block). Fig. 3(a) shows the end mappings of the previous round where the new round is started.

Fig. 3(b) displays the first refresh in the new round. The refreshed physical address for MA(00) will be RMA (11) ($MA(00)\ xor\ key(11) = RMA(11)$). The data in MA(00)'s old RMA(01), `A`, and the data in MA(00)'s new RMA(11), `C`, are swapped. Next, MA(01) will be refreshed similarly [Fig. 3(c)].

For each refresh, two MAs (and two RMAs) are associated. The finish of the refresh for one MA fortunately helps the finish of the refresh for the other MA. This is called the *pairwise remapping property* in SR. In the example, MA(00) is remapped from RMA(01) to RMA (11), and correspondingly MA(10) is remapped from RMA(11) to RMA(01), which is just about to happen later in this round. Therefore, when the CRP points to MA(10) [Fig. 3(d)], the duplicate swap must be discovered and avoided. For SR to test whether an address has been associated with a previous refresh, $MA\ xor\ key_p\ xor\ key_c$ (key_p and key_c stand for the previous and current key respectively) is computed. If the result is smaller than the CRP, it means the address has been refreshed already in the round, and vice versa [see Fig. 3(d) and (e)]. After all four logical addresses have been refreshed, the next refresh round will be started.

Upon each memory access (read and write), the MA must be translated to its current RMA. To do so, if the MA or the result of $MA\ xor\ key_p\ xor\ key_c$ is less than the CRP, current key should be used for calculating its RMA, otherwise the key in the previous round should be used. The SR wear-leveling is implemented in hardware by a Security Refresh Controller (SRC) in each memory region (bank).

Wear-leveling based on SR can also be organized in a two-level scheme as depicted in Fig. 4(c). In the two-level scheme, another layer of indirection for address mapping, *intermediate remapped memory address* (IRMA), is inserted between MA and RMA. In the inner level, the region is divided into equally sized sub-regions. Overall, the outer-level SR iteratively remaps each MA to different IRMAs in a large region, and then the inner-level SR iteratively remaps each IRMA to different RMAs in a isolated small sub-region. Although both levels apply the SR algorithm, they are independent from one another.

3 Multi-Way Wear-Leveling

3.1 Problem

As discussed in Section 2.2, the wear-leveling schemes based on algebraic mapping can achieve good performance and low complexity, but are susceptible to malicious attacks. To defend the attacks, the ideal wear-leveling scheme is required to concurrently satisfy four properties. 1) *Secure mapping*: The address mapping should be invisible to the users. Randomly changing the mapping is more robust; 2) *Efficiency*. The mapped physical address of the logical address should be changed quickly enough when the logical address is attacked; 3) *Sufficiency*. The physical address space to which the logical address can be mapped should be as large as possible so that the wear can be distributed sufficiently; 4) *Low overhead*. The overhead generated by wear-leveling mechanism (e.g., the swaps in SR and the two translations in two-level SR) should be low because the overhead would result in longer access latency and extra wear.

However, existing approaches cannot satisfy the above requirements, especially for the basic algebraic mapping that is described in Section 2.2. The rationale for these basic schemes is shown in Fig. 4(a). The circle arrow represents the address mapping and remapping between the logical space and the physical space. It can be observed that along with one logical address being remapped, all other logical addresses must be remapped. If the attacker simply writes to one address repeatedly, it only takes about 32 seconds to fail a memory cell [14]. This result applies to both basic Start-Gap and Security Refresh. Therefore, more sophisticated schemes based on the basic algebraic mapping are proposed to enhance the security of PCM under attacks. Those include among others: Randomized Region Based Start-Gap (RBSG) and two-level Security Refresh.

With regard to the RBSG, its rationale is illustrated with Fig. 4(b). The address space is divided into isolated sub-spaces, and one logical sub-space can only be mapped to one dedicated physical sub-space. Thus the overall lifetime is decreased to the shortest lifetime of the sub-spaces [see Fig. 4(b) for illustration]. Besides, RBSG cannot dynamically change

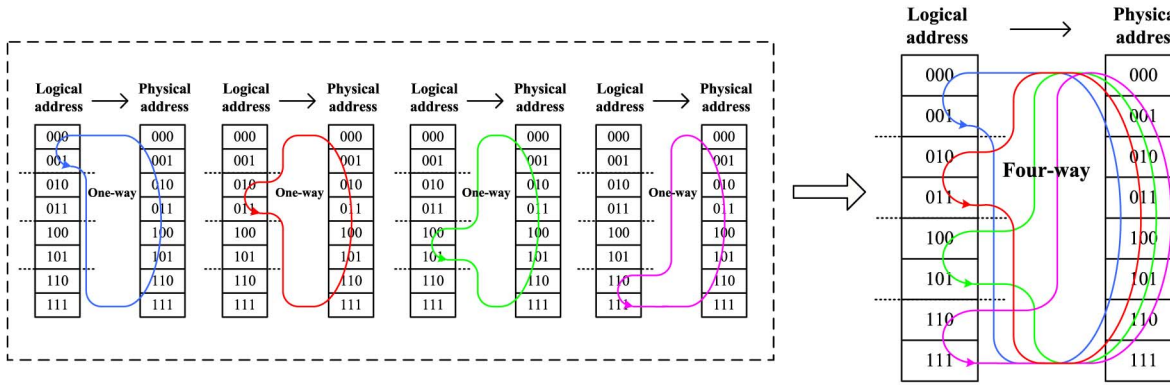


Fig. 5. Four-Way Wear-Leveling.

the initial random mappings due to the reason that it uses a static randomizer that keeps the randomized mapping constant throughout system execution, and the randomizer logic can only be programmed either at design time or at boot time.

Seong et al. proposed a two-level wear-leveling scheme based on SR as depicted in Fig. 4(c). In the two-level SR, another layer of indirection for address mapping, *intermediate remapped memory address* (IRMA), is inserted between MA and RMA (MA and RMA refer to logical address and physical address respectively). In the inner level, the region is divided into equally sized sub-regions as that in RBSG. The outer-level SR iteratively remaps each MA to different IRMAs in the region as that in basic schemes [Fig. 4(a)], and then the inner-level SR iteratively remaps each IRMA to different RMAs in an small sub-region. Both levels apply the SR algorithm, but are independent from one another. In the inner level, the sub-regions are independent too. The two-level SR scheme incurs significant write overhead under repeated address attacks (about 11.7% of the total writes). Decreasing the write overhead can substantially reduce the lifetime, e.g., if we constrain the write overhead to 2%, the lifetime will be only half the longest. In addition, for normal memory accesses (both reads and writes), the logical address must be translated twice to get the physical address. We argue that such high overhead makes the scheme less attractive. We will discuss the reason for the high overhead in the next section.

3.2 Solution

After examining the high overhead in the two-level SR, our observation is that the addresses are only effectively remapped in one isolated sub-region at the inner level, while the outer level is far less effective. Because the logical addresses are remapped from the first address to the last, e.g., 1 GB memory is comprised of four million 256B blocks. For each address remapped, all of the four million addresses must be remapped altogether, which results in the exceedingly slow address changes. If attacked, the scheme can do nothing but increase the refresh rate to accelerate the address remappings to distribute the writes to the attacked physical blocks.

Generally, the logical addresses sequentially change their mapped physical addresses in one algebraic mapping round. The logical address space is always equal to the physical address space [see Fig. 4(a)], such that to remap any logical address, all the addresses in the logical space must be remapped. The leveling efficiency is determined by the size of the logical space. Decreasing the logical space increases the leveling efficiency. While the leveling sufficiency is dictated by the physical addresses in the physical space, whose size is equal to that of the logical space traditionally. Increasing the physical space increases the leveling sufficiency. Consequently, increasing both the leveling efficiency and the sufficiency is impossible.

Based on the above insight, we propose *multi-way wear-leveling* (MWWL), a novel but simple scheme that can quickly distribute writes to the entire physical address space uniformly. MWWL is a multiple of *wear-leveling ways*, each of which individually maps (and remaps) one small logical address space to the entire physical address space. In one wear-leveling way, the logical space is small so that the logical address can be remapped along with a small number of logical addresses being remapped. When any way is heavily written (such as an attack), the wear-leveling can be provided so efficiently that the written physical address will be changed very fast. This is why MWWL can operate at very slow remapping rate. While the physical space that can be remapped to by any logical address in one way is as large as the entire address space. So the wear can be distributed sufficiently. MWWL can achieve almost the ideal lifetime. MWWL can be easily applied to existing algebraic mapping algorithms, such as Security Refresh. We present Multi-Way Security Refresh (MWSR), which applied the MWWL approach on top of the Security Refresh technique, in Section 4.

Fig. 5 illustrates the rationale of the MWWL scheme with an instance, Four-Way Wear-Leveling (4WWL). In 4WWL, we first divide the logical address space into equally sized four sub-spaces, then each sub-space individually undertakes the wear-leveling for the addresses, which is called one way of wear-leveling. Each way of the four ways (sub-spaces) maps and remaps their local logical addresses to the entire physical space (see left in the figure). When one logical sub-space starts a new remapping round, it will engage another logical sub-space (called its pair) in that round and interact with the pair for remapping (see right in the figure).

Fig. 6 aids the discussion. Logical sub-space $\mathbf{L1}$ is mapped to physical sub-space $\mathbf{P1}$ in round $\mathbf{M}$, while logical sub-space $\mathbf{L2}$ is mapped to physical sub-space $\mathbf{P2}$ in round $\mathbf{N}$. Then both ways start the next remapping round. In round $\mathbf{M+1}$, $\mathbf{L1}$ will be remapped to $\mathbf{P2}$, and in round $\mathbf{N+1}$, $\mathbf{L2}$ will be remapped to $\mathbf{P1}$ accordingly. We call $\mathbf{L1}$ and $\mathbf{L2}$ pair sub-spaces, in which their logical blocks switches with each other (called pair

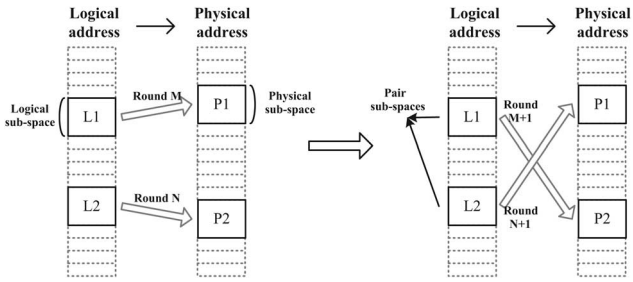


Fig. 6. Pair sub-spaces.

blocks likewise). Pair sub-spaces can be in different rounds of mapping (M, N), which are decided by the write intensity to different sub-spaces since each way individually performs the wear-leveling (address remapping). Note that in each remapping time (dictated by remapping rate), only one block is remapped. The remapping patterns are determined by the specific algebraic mapping algorithms. Take MWSR (next section) for example, the pairs are determined randomly by the remapping key generated at the beginning of each round, which is based on the intrinsic property of exclusive-or operations. The two pairs both start and end a remapping round at the same time. If one way has finished the remapping round, it will start the next remapping round immediately. If one way finishes and is going to remap with another way that in middle of a round, it will wait and accelerate the lagging way to remap once. We will elaborate the remapping pattern for MWSR in the next section.

Interestingly, according to our taxonomy, the Start-Gap [14] and the Security Refresh [18] [see Fig. 4(a)] fall into the category of *one-way wear-leveling*. The Region Based Start-Gap [Figure 4(b)] and the inner-level SR [Fig. 4(c)] are a multiple of independent instances of one-way wear-leveling. MWWL can be applied to the existing algebraic mapping algorithms, such as the Start-Gap [14] and the Security Refresh [18]. To demonstrate the applicability of MWWL, we present *Multi-Way Security Refresh* that is constructed on top of basic Security Refresh, and formally describe the multi-way version in the next section. For other algebraic mapping algorithms, the conversion can follow the same way.

4 Multi-Way Security Refresh

Multi-Way Security Refresh (MWSR) improves SR by means of converting it from one-way wear-leveling to multi-way wear-leveling. Constructed on top of the SR[3], the MWSR randomly generates a new key for each refresh round. For MWSR, the key is comprised of two sub-keys: one sub-key for calculating the remapped sub-region (the region key, key_{region}), the other for the block offset in that sub-region (the block key, key_{block}). For brevity, we assume there are 2^m sub-regions and 2^n blocks in each sub-region, hence the key is expressed as:

$$\underbrace{s_1 s_2 \cdots s_m}_{key_{region}}, \underbrace{b_1 b_2 \cdots b_n}_{key_{block}}.$$

.

3. We used the same terms as those of Seong et al.[18], such as region and sub-region, MA and RMA in the following discussion.

It will be N-Way SR if the memory region is divided into N sub-regions. In the next subsections, we will first introduce the simplest case, Two-Way SR, and then formally describe the N-Way SR. A small number of registers are induced for storing the state for the mapping. Generally, each logical sub-region (LSR) has a write counter (WC) that counts the number of writes for the LSR, two mapping keys for the previous and current round, and CRP pointing to the next address to refresh. Each physical sub-region (PSR) has a CLSR register to indicate the current LSR that is being remapped to the PSR.

4.1 Two-Way Security Refresh

In the two-way SR (2WSR), the region is thus divided into two LSRs, $lsr0$ and $lsr1$, and two PSRs, $psr0$ and $psr1$. If each sub-region has four blocks, the key is then in the form of $s_1, b_1 b_2$, where s_1 is region key, $b_1 b_2$ is block key. We describe the 2WSR through different remapping cases in Fig. 7, each of which is detailed in this section.

- Fig. 7(a), the start of one-way's remapping round. New region keys and block keys are generated for two sub-regions (ways).
- Fig. 7(b), if the current region key of one way is the same as that in the previous round, data swapping are restricted inside the sub-region according to the block key. There is no data swapping between two sub-regions.
- Fig. 7(c), if the current region key of one way is different from that in the previous round, a pair region is found through algebraic calculation using region key. In this example, these two sub-regions are in pair, and the data is swapped from one sub-region to the other for wear-leveling.
- Fig. 7(d), if the remapping of a block is already done in the pair sub-region, the redundant remapping can be detected and avoided through algebraic calculation using keys.
- Fig. 7(e), the two pair sub-regions finish remapping round at the same time (if one finishes, so does the pair).
- Fig. 7(f), if one sub-region enters a new mapping round and finds that its new pair sub-region has not finished the current mapping round, the sub-region needs to hold its refreshing till its pair sub-region finishes the current mapping round. In this case, the writes to the leading sub-region (holding sub-region) will trigger the refreshing in the lagging sub-region to accelerate the current mapping round of the lagging sub-region.

Fig. 7(a) shows the new round for $lsr0$ with $key(1, 00)$ and $lsr1$ with $key(1, 01)$. When an LSR first starts to refresh, a new key will be randomly generated for that LSR. If the new region key is the same as the old one for that LSR, the logical blocks will be remapped to the same physical sub-region as that in the previous round. So the refreshes in one LSR are independent of those in the other LSR. In such case, the 2WSR has the same behavior as SR. In Fig. 7(b), both LSRs have the same new region key, $key_{region}(1)$. When $lsr1$ refreshes the MA(101), the physical blocks RMA(000) and RMA(010) will be swapped.

Otherwise, if the new region key differs from the old one, the logical blocks in the LSR will be remapped to a different physical sub-region. As a result, the block to be refreshed in one LSR will have a pairwise block to be refreshed in another

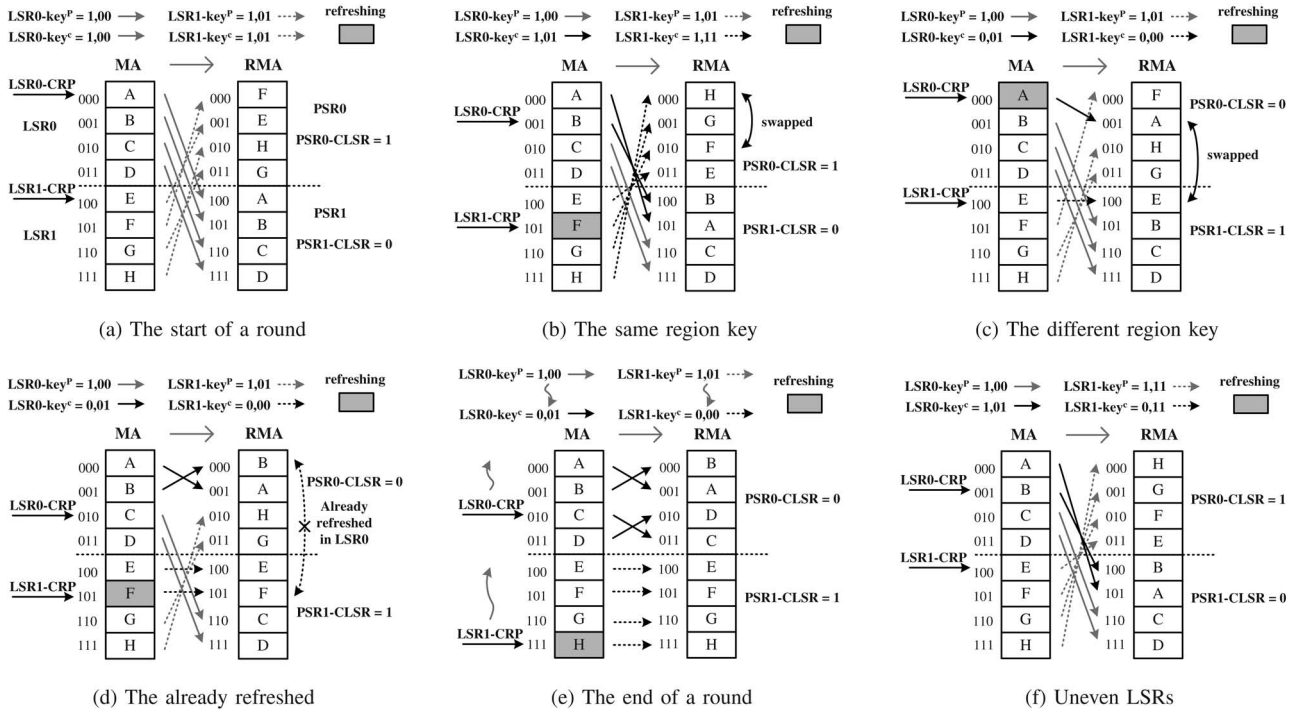


Fig. 7. An example of memory address remappings in Two-Way Security Refresh. The different arrows indicate the current mappings between logical and physical blocks. The gray box represents the block is being remapped. All the registers are updated.

LSR (the pairwise remapping property). We call either of the two blocks the *pair block* to the other, and their LSRs the *pair sub-region* to the other[4]. In MWSR, determining the pair block is different from SR because LSRs have different keys. Let $lb.pair$ denote the pair block of block, lb, in LSR, lsr. Let $lsr.pair$ be the pair sub-region of lsr. Let $lsr.keys.cr$ and $lsr.keys.pr$ represent the current and previous keys for lsr respectively (the notations are the same for $lsr.pair$). Thus $lb.pair$ is expressed as follows. $lsr.pair$ can be calculated similarly.

$$lb.pair = lb\ xor\ lsr.keys.cr\ xor\ lsr.pair.keys.pr.$$

In Fig. 7(c), $key_{region}(0)$ is newly generated for $lsr0$, which is different from the previous region key. Therefore, $lsr0$'s pair sub-region will be $lsr1$ and MA(000) in $lsr0$ will have the pair block MA(100) in $lsr1$. In this case, in order to maintain the pairwise remapping property for the pair LSRs and blocks, the new current round key $lsr1.keys.cr$ for $lsr1$ will be determined by the formula:

$$lsr1.keys.pr\ xor\ lsr0.keys.cr\ xor\ lsr0.keys.pr$$

.

The CLSRs for both $psr0$ and $psr1$ will be changed to their new LSRs accordingly. When $lsr0$ refreshes MA(000), RMA (001) in $psr0$ and RMA(100) in $psr1$ will be swapped.

To find out duplicate swap when refreshing block that has already been refreshed along with its pair block in the pair sub-region, the original test used in SR needs modification: if the address of the pair block is smaller than the pair sub-region's CRP, then the block has already been refreshed, and vice versa. In Fig. 7(d), MA(101) is to be refreshed. Its pair block will be:

$$MA(101)\ xor\ key(000)\ xor\ key(100) = MA(001).$$

As MA(001) is smaller than the $lsr0.crp$ that points to MA (010), the duplicate swap can be identified and hence avoided.

LSRs can make different progress in their refresh rounds due to the uneven writes to the LSRs. When a round is finished in an LSR, the next round will be started for both of the LSR and its pair sub-region (the pair sub-region may be the LSR itself). As such, both of their CRPs should be moved to the first logical block in the sub-regions and the current mapping key should be saved in the previous mapping key register for the two LSRs. For instance [Fig. 7(e)], after MA(111) in $lsr1$ is refreshed, the rounds for both $lsr0$ and $lsr1$ are finished.

If a sub-region finishes a round and enters a new one, but its next pair sub-region for the new round is still in an ongoing round, the leading sub-region will make the next pair sub-region refresh to accelerate the lagging refresh round. Fig. 7(f) illustrates this case. In the new round, a new region key, $key_{region}(0)$, is generated for $lsr1$. But $lsr1$'s pair sub-region, $lsr0$, is still in the round with $key(1, 01)$. As a result, refreshing MA(100) in $lsr1$ will be changed to refreshing MA(001) in $lsr0$ first.

4.2 N-Way Security Refresh

We describe the N-Way Security Refresh (NWSR) algorithms formally through pseudo-codes. The codes extend the 2WSR to arbitrary number of ways while consistently maintaining the integrity of the mappings for all the logical and physical addresses. The notations used in the algorithms are defined in Table 1. For each write to the LSR, lsr, the write counter for lsr, $lsr.wc$, is incremented. If the $lsr.wc$ reaches the refresh rate,

4. If the region key is the same as the previous, both pair blocks will be in the same LSR and the LSR's pair sub-region will be itself.

TABLE 1
Notations Used in the Algorithms

lsr	Logical sub-region
$lsr.wc$	Write counter for lsr
$lsr.rr$	Refresh rate for lsr
$lsr.cr$	Current refresh round for lsr
$lsr.pr$	Previous refresh round for lsr
$lsr.crp$	Current refresh pointer for lsr
$lsr.keys$	Mapping keys for lsr
$lsr.keys.cr$	Mapping key for $lsr.cr$
$lsr.keys.pr$	Mapping key for $lsr.pr$
$lsr.keys_{region}$	Region key for round n
$lsr.pair$	Pair logical sub-region of lsr
psr	Physical sub-region
$psr.clsr$	Current LSR mapped to psr
lb	Logical block
$lb.pb^c$	Phys. block of lb in curr. round
$lb.pb^p$	Phys. block of lb in prev. round
$lb.pair$	Pair block of lb in $lsr.pair$
$lb.lsr$	Logical sub-region of lb
pb^n	Phys. block mapped in round n
$pb^n.psr$	Phys. sub-region of pb^n

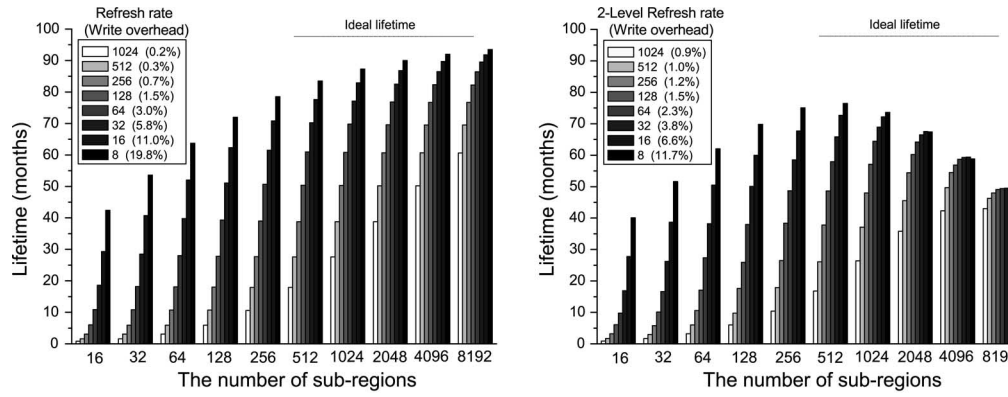
$lsr.rr$, the address refresh procedure (in Algorithm 1) is invoked (the $lsr.wc$ is then cleared). Once accessed, the logical address must be translated into physical address. Note that there is no need to maintain the round count because only the keys of the previous and current rounds are needed for remapping in the current round. The algorithm for address translation is shown in Algorithm 2. We assume a region is in a bank, which allows one request to access the cell array at a time.

Algorithm 1 Refresh a block in lsr

if $lsr.cr = lsr.pr$ **then**
  $lsr.cr \leftarrow (lsr.pr + 1)\%2$
  $lsr.keys.cr \leftarrow$ randomly generated number
end if
$lb \leftarrow lsr.crp$
$lb.pb^c \leftarrow lb\ xor\ lsr.keys.cr$
$psr \leftarrow lsr\ xor\ lsr.keys_{region}.cr$
$lb.pair.pb^c \leftarrow lb\ xor\ lsr.keys.pr$
$psr.pair \leftarrow lsr\ xor\ lsr.keys_{region}.pr$
if $psr.clsr = lsr$ **then**
  $lsr.pair \leftarrow psr.pair.clsr$
  $lb.pair \leftarrow lb.pb^c\ xor\ lsr.pair.keys.pr$
  if $lb.pair < lsr.pair.crp$ **then**
    lb has already been refreshed
    $lsr.crp \leftarrow lsr.crp + 1$
    if lsr completes current round **then**
      update both lsr and $lsr.pair$
    end if
  else
    swap $lb.pb^c$ and $lb.pair.pb^c$
    $lsr.crp \leftarrow lsr.crp + 1$
    if lsr completes current round **then**
      update both lsr and $lsr.pair$
    end if
  end if
else if $psr.clsr$ is not in any ongoing round **then**
  $psr.clsr \leftarrow lsr$
  $lsr.pair \leftarrow psr.clsr$
  $lsr.pair.cr \leftarrow (lsr.pair.pr + 1)\%2$
  $lsr.pair.keys.cr \leftarrow lsr.keys.cr\ xor$
    $lsr.keys.pr\ xor\ lsr.pair.keys.pr$
  $lb.pair \leftarrow lb.pb^c\ xor\ lsr.pair.keys.pr$
  $psr.pair.clsr \leftarrow lsr.pair$
  swap $lb.pb^c$ and $lb.pair.pb^c$
  $lsr.crp \leftarrow lsr.crp + 1$
  if lsr completes current round **then**
    update both lsr and $lsr.pair$
  end if
else
  refresh $psr.clsr$
end if

Algorithm 2 Translate lb in lsr into physical address

$lb.pb^c \leftarrow lb\ xor\ lsr.keys.cr$
$lb.pb^p \leftarrow lb\ xor\ lsr.keys.pr$
if $lsr.cr = lsr.pr$ **then**
  return $lb.pb^p$
else
  if $lb.pb^c.psr.clsr = lsr$ **then**
    if $lb < lsr.crp$ **then**
      return $lb.pb^c$
    else
      $lsr.pair \leftarrow lb.pb^p.psr.clsr$
      $lb.pair \leftarrow lb.pb^c\ xor\ lsr.pair.keys.pr$
      if $lb.pair < lsr.pair.crp$ **then**
        return $lb.pb^c$



(a) Multi-way SR (b) Two-level SR

Fig. 8. The lifetime vs. different sub-regions under repeated address attacks (the higher the better). The top line is the expected ideal lifetime (the duration to wear out all of the addresses).

else

return $lb.pb^p$

end if

end if

else

return $lb.pb^p$

end if

end if

Note that the recursion in Algorithm 1 happens when a sub-region finishes but its next round pair is currently in an unfinished round. But the depth is limited to 1 because it is impossible to involve another pair (in the round after the next). Tracking the logical sub-region mapped to the physical sub-region is needed for checking the status of its current pair, as well as its next pair.

5 Evaluation

In addition to the baseline implementation [18] for MWSR, the inner-level implementation of the two-level SR scheme should be extended. The circuit for the remapping checker (RC) and the address translation logic (ATL) needs to be boosted as in Algorithm 1 and Algorithm 2 respectively to allow the interaction between any two pair sub-regions. The circuit design is out of the scope in this paper. The hardware overhead will be discussed in Section 5.3. In terms of the access delay, MWSR involves single-level address mapping, thus at most one swap and only one address translation is needed for each access, which can significantly reduce the access latency. Additionally, MWSR incurs very low write overhead compared with two-level SR, this result can further accelerate the PCM I/O operations. The performance overhead will be discussed in Section 5.2. In our experiments, we evaluate MWSR by simulation using architecture parameters similar to that used in [14], [18] to make a fair comparison. We assume that the PCM used as main memory is backed up with an off-chip DRAM as a last-level cache. The region to be leveled is a memory bank of 1G bytes. The PCM read time is 150 ns and write time is 450 ns. The size of the block is the same as that of the last-level cache line (256B). We also assume that the write endurance for storage cell is 10^8 and there are 64 K spare lines.

5.1 Endurance

To evaluate the leveling performance of MWSR, we conduct as many malicious attacks as the system can endure and measured the lifetime, including birthday paradox attack and repeated address attack. In fact, for MWSR the birthday paradox attack (BPA) similar to the repeated address attack (RAA) that writes to one single logical address. This is because 1) the SR mapping algorithm randomly remaps every logical block to a new physical block in the entire physical space after writing to a random address (the number of blocks in a sub-region $\times$ Refresh Rate) times, and 2) if picking a different random address just once (as that in BPA), it is more harmful than keeping writing to the same address, but after that, the "picking" and the "keeping" become the same.

We iteratively simulate different memory configurations without caching and calculate the average lifetime. We compare MWSR with existing two-level SR. Fig. 8 shows the lifetime of MWSR (left) and two-level SR (right)[5] for comparison. The top line is the expected ideal lifetime (the duration to wear out all of the blocks). It can be seen that the lifetime of MWSR scales well with the increase of refresh rate and also with the increase of the number of sub-regions. This is because smaller sub-regions can result in attacked blocks being remapped to other sub-regions more quickly, so can higher the refresh rate. In comparison, MWSR can extend the lifetime by up to 94% as two-level SR. But more importantly, MWSR is able to achieve more than 76 months lifetime (80% of the ideal lifetime) by only incurring less than 2% write overhead (the refresh rate more than 128), such as when there are more than 2048 sub-regions. For 2LSR to reach this lifetime, however, its write overhead will be over 11% (only at the refresh rate 8). Note that the spare lines in PCM (which can be substitutes for worn-out lines) make marginal contribution to the lifetime, especially for good wear-leveling schemes, such as MWSR. Because the interval from the occurrence of the first failed block to the failure of all spare lines is very short.

5. The lifetime data that we calculated and reproduced here for the two-level SR differs from the original data in their paper within $\pm 3\%$. The outer-level refresh rate for two-level SR is 128. The following mentioned refresh rate for two-level SR is the inner-level refresh rate.

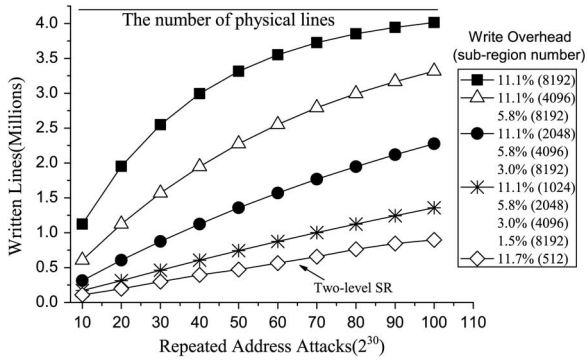


Fig. 9. Written lines with increased repeated writes. Different lines indicate varied write overhead and sub-regions number combinations. Some Combinations overlap with each other and are grouped.

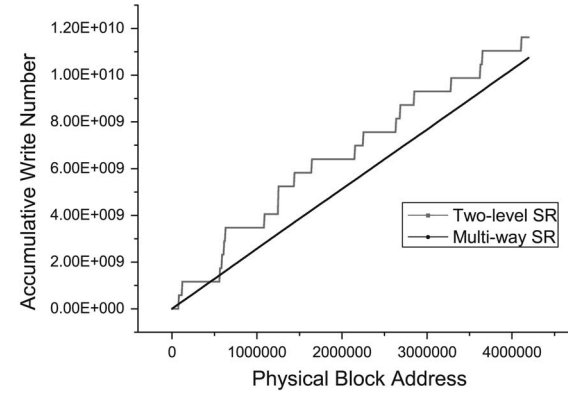


Fig. 10. The distribution of write under 10×2^{30} repeated address attacks (the smoother the better). The y-axis is the accumulated write number; the x-axis is the physical blocks in address order. The write number w for address x would be $w(x) = y(x) - y(x-1)$.

In Fig. 8, the write overhead difference between those two schemes needs more explanation. The write overhead comes from data swaps in remapping blocks. Given refresh rate (RR) in SR, the swap occurs every $1/2$ RR writes (half of the swaps are canceled because when a block is remapped, its pair block is remapped too). Provided two writes per swap, the write overhead for one level is $1/(\mathrm{RR}+2)$. For 2LSR, the calculation will be more complicated. But roughly calculating the case where the outer-level and inner-level RR are 128 and 8 respectively, the overall write overhead is $(1/130 + 1/10)$, which approximates 11.7%. Our actual method counts every write in the simulation. For our MWSR scheme, the logical sub-region usually swaps with another (pair) sub-region. Under the attack, since only one sub-region is heavily written, its pair does not have the chance to trigger remapping. Therefore, the swap occurs every RR writes (no canceled swaps), which entails the write overhead, $2/(\mathrm{RR}+2)$. For instance, if the RR is 8, the write overhead $2/(8+2)$ approximates 19.8%. However, if RR is 128 for both of the two levels in 2-level SR, its overall overhead is $2 * 1/(128+2) = 1.5\%$, which equals the case where RR is 128 in our scheme (only one level). But at such slow RR, our scheme still promises very long lifetime because it works well at very slow RR, e.g. 128, hence low write overhead.

To have a better look at the leveling performance, we exercise increasingly large number of writes to a dedicated block (from 10 billion to 100 billion, the x-axis in Fig. 9) and count the number of written physical blocks (the y-axis)[6]. Written physical block number reflects how many blocks the writes are shared in order to prevent blocks under intensive writes from being worn out sooner than the others. More written blocks mean better leveling performance. Under various configurations (write overhead and sub-regions listed in the right box), MWSR can not only distribute writes to more written lines, but also distribute writes at *faster* speed, which indicates that MWSR leveling is more effective and efficient.

The number of written lines partially represents leveling performance, because the number of writes across the written lines may be unbalanced. To observe the distribution of writes, we plotted the accumulative graph that shows the accumulated write number across all the physical addresses. In Fig. 10, the y-axis is the accumulated write number; the x-axis is the physical blocks in address order. Thus, the write number w for address x would be $w(x) = y(x) - y(x-1)$. Smoother graph reflects that the real write numbers across all blocks have less variance. We only compare one configuration for MWSR against one for two-level SR. The configuration for two-level SR is that there are 512 sub-regions and the refresh rate is 8 with the write overhead 11.73%, in such case the two-level SR's lifetime is 76.5 months. The configuration for MWSR is 8192 sub-regions and refresh rate 128 with write overhead (1.54%), whose lifetime is even longer, i.e., 82.2 months. The result in Fig. 10 shows that MWSR is able to achieve more uniform leveling than the best of two-level SR while still incurring little write overhead, reduced from 11.73% down to 1.54%.

5.2 Performance Overhead

We used Simics simulator [21] to evaluate the performance degradation for MWSR. In our UltraSparc III platform, the system employs 8-core in-order CMP (1 GHz), private 128KB L1 cache, shared 4 MB L2 cache, and a 16 MB L3 DRAM cache for PCM. The memory read latency is set to 150 cycles, write latency is 450 cycles, and single micro-controller with FIFO scheduling. The peripheral address remapping logic needs 20 ns and the address translation logic requires 5 ns. We run 7 multi-threaded PARSEC benchmarks and 27 SPEC2006 benchmarks. MWSR's refresh rate is set to be 64, 128, and 256. Accordingly the write overhead is 3.0%, 1.5%, and 0.7%. Due to the space limitation, we only report the overall performance numbers at the 3 refresh rates. For PARSEC, the completion time increases about 1.24%, 1.07%, and 1.01%. For SPEC2006, the average performance degradation is very low, all less than 0.5%. The results conclude that the performance impact on MWSR is negligible. Note that we use the same read/write latencies to PCM as those in the Security Refresh paper (ISCA'10) to make a fair comparison. With longer write latency, the slower refresh rate (lower write overhead) will be preferred because the extra writes will result in more performance degradation. In that case, MWSR shows more benefits because it has lower write overhead, compared to 2-level SR.

6. Note that the written lines in the following of the paper are the physical lines that have been written by the attacks, without the lines that have only been written by swapping lines generated from the leveling mechanism.

5.3 Hardware Overhead

In this section, we discuss MWSR's hardware overhead. Specifically, for each sub-region, the storage overhead includes two keys (one for current round and one for previous round), write counter that is determined by the refresh rate, current remapping pointer, and current logical sub-region pointer. The two swap buffers can be shared by all the sub-regions. Suppose each block is of the size b, there are 2^n sub-regions, each sub-region contains 2^x blocks, and refresh rate is 2^y. The total hardware overhead will be approximately: $2^n \times (3 \times x + y + n) + 2 \times b$, where each sub-region has three mapping keys, one write counter, and one pair sub-region register, plus two shared swap buffers in total.

Fig. 8(a) shows that the endurance increases with the sub-region number. As the number of sub-regions increases, a slower refresh rate, which means lower write overhead, is required to achieve the same lifetime. The more storage overhead, however, is induced to keep the keys of sub-regions. So it can be considered as a tradeoff between the lifetime and the "effective capacity". The "sweet spot" depends on the capacity and the lifetime requirement of the real system. For example, under the architecture assumed throughout this section, if there are 2 K sub-regions and the refresh rate 64, the approximate hardware overhead will be 12.5KByte out of 1GByte per PCM bank. Such configuration can achieve 76.8 months lifetime, but only incurs 3.0% write overhead with almost the same hardware cost as the SR approach.

6 Related Work

Though not being used commercially yet, PCM technology has been developed for years. We summarized related works as three categories below.

Build PCM systems: It has been proved viable to build main memory as a hybrid of DRAM and PCM with the operating system support to manage such hybrid memories [9]. In addition, Caulfield et al. [1] proposed storage array based on PCM. By using SRAM or DRAM, some efforts [15], [22] have been done to exploit using hybrid memory architecture designs to reduce access latency and overall power consumption to harvest both the latency benefits of RAM and the capacity benefits of PCM.

Improve PCM based memory performance: To reduce the read access time to fill the gap between PCM and DRAM, Qureshi et al. [12] presented adaptive write cancellation and write pausing policies. They altered the number of bits per cell in terms of the varying workloads to improve the slower access latency for multi-level cell PCM devices [13]. Architecture support was proposed for PCM to address long latencies, high energy writes, and finite endurance by reorganizing memory buffer and using partial writes [8]. To reduce the write times and energy and increase write endurance, read-modify-write operation [2] was used to replace PCM write. Condit et al. [3] presented file system to provide atomic, fine-grained updates to offer strong reliability and better performance. All memory transfers between the last level cache and persistent memory are encrypted to provide confidentiality during system suspend and across reboots [4]. To address failure caused by write endurance, Ipek et al. [6] introduced hardware and operating system design to dynamically replicate memory pages when hard faults occur in PCM cells. To increase the number of recoverable failures for longer lifetime, Seong et al. [19] proposed a recovery method (stuck-at errors in writes).

PCM durability study: Redundant write elimination was investigated [24] to reduce PCM writes and row-shifting and segment rotation for wear-leveling to achieve the best tradeoff between energy and performance. Qureshi et al. [14] proposed randomized region based Start-Gap algorithm that rotates blocks and for the first time introduced algebraic mapping in PCM wear-leveling. To further avoid information leak, Security Refresh that remaps blocks randomly was proposed and then this scheme was extended it to two-level scheme [18]. Hierarchical table based secure wear-leveling was designed by Seznec [20] to survive overwrite attacks. Our method is fundamentally different from the table based mapping in Seznec's work, in that after the new region key is randomly generated, the blocks in the sub-region are remapped to the new sub-region, but only one block is swapped at any time of refreshing. On the contrary, Seznec's scheme swaps two regions at a time (see Section 2.2.5 in his paper). More important, our scheme is not a simple combination of Seznec's table approach and algebraic approach. From a high level view, the Multi-Way method does not map sub-regions by table. When modifying table, we do not swap the two entire sub-regions or apply SR inside each sub-region. Multi-Way actually uses random keys to remap just one block in the logical sub-region that is being written intensively to a random physical sub-region at remapping time according to the refreshing rate. Obviously Seznec's mechanism is different from ours and is not adopted in our work. Kong and Zhou [7] adjusted the strength of error correction code to extend the lifetime of PCM. Qureshi et al. [16] investigated online attack detection to adapt the rate of block rotation depending on the properties of the memory reference stream, differently under normal applications or malicious attacks. Our MWWL is an algebraic mapping based method. But it differs in that the wear-leveling is performed in a multiple of ways, each of which remaps blocks to the entire physical region, and that two levels of remapping are unnecessary. We also extended SR to MWSR and demonstrated that it can offer significant advantage over the two-level SR.

7 Conclusion and Future Work

PCM is an emerging non-volatile memory technology, which is considered a promising DRAM alternative. Existing schemes for wear-leveling suffer low efficiency and sufficiency, which results in significant write overhead, performance degradation, or high hardware cost. This paper proposes MWWL, a novel wear-leveling scheme that simultaneously distributes writes uniformly from a multiple of ways and guarantees each logical address can potentially be remapped to the entire physical address space. The experiments show the MWWL is able to increase endurance and security while incurring little overhead. In terms of the future work, the policy for generating mapping keys for each LSR may be defined to further mitigate the write overhead by means of taking into account the write pattern.

Acknowledgment

This paper was supported by the National Basic Research Program of China(2012AA012600) and the National Science Foundation of China(60973143).

References

[1] A. Caulfield, A. De, J. Coburn, T. Mollov, R. Gupta, and S. Swanson, "Moneta: A high-performance storage array architecture for next-generation, non-volatile memories," in *Proc. Int. Symp. Microarchitecture (MICRO-43)*, 2010, pp. 385–395.

[2] S. Cho, and H. Lee, "Flip-n-write: A simple deterministic technique to improve pram write performance, energy and endurance," in *Proc. Int. Symp. Microarchitecture (MICRO-42)*, 2009, pp. 347–357.

[3] J. Condit, E. Nightingale, C. Frost, E. Ipek, B. Lee, D. Burger, and D. Coetzee, "Better I/O through byte-addressable, persistent memory," in *Proc. 22nd ACM Symp. Oper. Syst. Princ. (SOSP'09)*, 2009, pp. 133–146.

[4] W. Enck, K. Butler, T. Richardson, P. McDaniel, and A. Smith, "Defending against attacks on main memory persistence," in *Proc Annu. Comput. Security Appl. Conf. (ACSAC'08)*, 2008, pp. 65–74.

[5] E. Gal, and S. Toledo, "Algorithms and data structures for flash memories," *ACM Comput. Surveys*, vol. 37, no. 2, pp. 138–163, 2005.

[6] E. Ipek, J. Condit, E. Nightingale, D. Burger, and T. Moscibroda, "Dynamically replicated memory: Building reliable systems from nanoscale resistive memories," vol. 38, no. 1, pp. 3–14, 2010.

[7] J. Kong and H. Zhou, "Improving privacy and lifetime of PCM-based main memory," in *IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN'10)*, 2010, pp. 333–342.

[8] B. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting phase change memory as a scalable DRAM alternative," in *Proc. 36th Int. Symp. Comput. Architecture (ISCA)*, 2009, pp. 2–13.

[9] J. Mogul, E. Argollo, M. Shah, and P. Faraboschi, "Operating system support for NVM+ DRAM hybrid main memory," in *Proc. 12th HotOS*, 2009, p. 14.

[10] Numonyx. (2008). The Basics of Phase Change Memory (PCM) Technology [Online]. Available: http://www.numonyx.com/Documents/WhitePapers/PCM_Basics_WP.pdf

[11] Numonyx. (2009). Phase Change Memory (PCM): A New Memory Technology to Enable New Memory Usage Models [Online]. Available: http://www.numonyx.com/Documents/WhitePapers/Numonyx_PhaseChangeMemory_WhitePaper.pdf

[12] M. Qureshi, M. Franceschini, and L. Lastras-Montano, "Improving read performance of phase change memories via write cancellation and write pausing," in *Proc. 16th IEEE Int. Symp. High-Performance Comput. Architecture (HPCA'10)*, 2010, pp. 1–11.

[13] M. Qureshi, M. Franceschini, L. Lastras-Montaño, and J. Karidis, "Morphable memory system: A robust architecture for exploiting multi-level phase change memories," in *Proc. 37th Int. Symp. Comput. Architecture (ISCA)*, 2010, pp. 153–162.

[14] M. Qureshi, J. Karidis, M. Franceschini, V. Srinivasan, L. Lastras, and B. Abali, "Enhancing lifetime and security of PCM-based main memory with start-gap wear leveling," in *Proc. Int. Symp. Microarchitecture (MICRO-42)*, 2009, pp. 14–23.

[15] M. Qureshi, V. Srinivasan, and J. Rivers, "Scalable high performance main memory system using phase-change memory technology," in *Proc. 36th Int. Symp. Comput. Architecture (ISCA)*, 2009, pp. 24–33.

[16] M. K. Qureshi, A. Seznec, L. Lastras, and M. Franceschini, "Practical and secure PCM systems by online detection of malicious write streams," in *Proc. 17th IEEE Int. Symp. High-Performance Computer Architecture (HPCA'11)*, 2011, pp. 478–489.

[17] S. Raoux, G. Burr, M. Breitwisch, C. Rettner, Y. Chen, R. Shelby, M. Salinga, D. Krebs, S. Chen, H. Lung, et al., "Phase-change random access memory: A scalable technology," *IBM J. Res. Develop.*, vol. 52, no. 4.5, pp. 465–479, 2010.

[18] N. Seong, D. Woo, and H. Lee, "Security refresh: Prevent malicious wear-out and increase durability for phase-change memory with dynamically randomized address mapping," in *Proc. 37th Int. Symp. Comput. Architecture (ISCA)*, 2010, pp. 383–394.

[19] N. Seong, D. Woo, V. Srinivasan, J. Rivers, and H. Lee, "SAFER: Stuck-at-fault error recovery for memories," in *Proc. Int. Symp. Microarchitecture (MICRO-43)*, 2010, pp. 115–124.

[20] A. Seznec, "A Phase Change Memory as a Secure Main Memory," *IEEE Computer Architecture Lett.*, vol. 9, no. 1, pp. 5–8, Jan.–Jun. 2010.

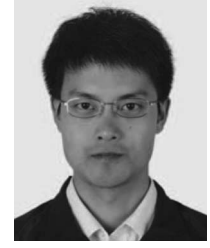
[21] Wind River. (2009). Wind River Simics [Online]. Available: http://www.windriver.com/products/simics/.

[22] X. Wu, J. Li, L. Zhang, E. Speight, R. Rajamony, and Y. Xie, "Hybrid cache architecture with disparate memory technologies," in *Proc. 36th Int. Symp. Comput. Architecture (ISCA)*, 2009, pp. 34–45.

[23] Y. Xie, "Modeling, architecture, and applications for emerging memory technologies," *IEEE Des. Test Comput.*, vol. 28, no. 1, pp. 44–51, Jan./Feb. 2011.

[24] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "A durable and energy efficient main memory using phase change memory technology," in *Proc. 36th Int. Symp. Comput. Architecture (ISCA)*, 2009, pp. 14–23.

Hongliang Yu received the doctor's degree in computer science from Tsinghua University, Beijing, China, in 2006. He is an associate professor with the Department of Computer Science and Technology, Tsinghua University. His research interests include storage systems, distributed systems, and mobile systems.



Yuyang Du received the doctor's degree in computer science from Tsinghua University, Beijing, China, in 2011. He is a software engineer in Intel, Beijing, China. He is now working on browser performance and power optimization.