Partial-PreSET: Enhancing Lifetime of PCM-Based Main Memory with Fine-Grained SET Operations

Yang Shi, Yanmin Zhu, and Linpeng Huang

Shanghai Jiao Tong University
{shiyang6017, yzhu, lphuang}@sjtu.edu.cn

Abstract. Phase change memory (PCM) is one of promising candidates to replace DRAM with its attractive features such as zero leakage power and high scalability. In PCM, SET operation needs much more time than RESET operation. A typical write request concurrently writes 64 bytes to the PCM memory line. Therefore, write latency is determined by SET operation. PreSET is proposed to improve PCM performance by exploiting asymmetry in write time. A PreSET operation pro-actively SETs all the bits in the memory line before a dirty cache line is written to PCM memory. Later, when a write request is processed, only RESET operation is performed. Consequently, PreSET shortens write latency which improves system performance. However, such PreSET operation is conducted at a coarse-grained level, which reduces the endurance of PCM. Through empirical study we find that in most applications the number of dirty words in a dirty line is actually quite limited. If we only SET only those dirty words, instead of the whole cache line, we would significantly extend the lifetime of PCM while still achieving desirable system performance. Inspired by this observation, we propose Partial-PreSET which balances performance and endurance of PCM system. The core idea of this scheme is to SET the dirty part of a cache line in a fine-grained fashion. Our experiments show that the proposed Partial-PreSET scheme significantly improves the average lifetime of PCM system, up to 2.79X, while incurring only 2% system performance loss, compared with the state-of-the-art scheme, i.e., PreSET.

Keywords: Memory architecture, Phase Change Memory, System performance, Write endurance

1 Introduction

Having been serving for the main memory for decades, DRAM fails in scaling down to smaller features. And the energy cost increases tremendously, which accounts for about 40% of total power consumption in computer system [1]. Therefore, new technologies which better handle these two problems have been proposed. One of the most promising technologies is Phase Change Memory. PCM has several prominent characteristics, like low standby power consumption

and high density, which make PCM a perfect substitute for DRAM. However, write latency and endurance of PCM are not desirable. A PCM cell can be written about only 10^{10} times [2], while a DRAM cell almost has infinite lifetime. And PCM's write latency is intolerable, which is almost 8 times larger than its read latency.

Qureshi et al.[3] proposed PreSET which improves performance of PCM by exploiting asymmetry in write time. Write latency in PCM is highly data dependant. To RESET a PCM cell(logically writing '0') , a short power duration pulse is needed[4], but a long power duration pulse is required when to SET a PCM cell(logically writing '1'). The latency of SET operation is almost 8X longer than that of RESET operation. A write request concurrently writes hundreds of bits which always contains both "1" and "0". Therefore, write latency is always determined by SET operation. However, a write operation to a memory line in which all cells have already been SET, incurs much lower latency. Based on the above observation, before a dirty cache line is evicted from cache, PreSET will in advance SET all the bits in the memory line. Later, when such cache line is being written to the memory, only RESET operation is required. PreSET operation is performed only if the memory bank is idle, so it hardly block read requests. Therefore, PreSET significantly improves write performance. However, when taking endurance problem of PCM into consideration, PreSET operation is highly costly. A PCM memory line will be written twice by PreSET. Firstly, a PreSET operation SET the whole memory line. Then, a write request will write the real data into the line. At worst, although only one bit is modified in a dirty cache line, PreSET has to SET the whole memory line that always contains hundreds of bits. Therefore, PreSET may tremendously shorten lifetime of PCM system.

Our work focuses on an improved scheme based on PreSET which takes good care of both endurance and performance problem. A key insight that motivates our work is that only a few of words in a dirty cache line are really modified. In Mohammad Arjomand' s work [5], an experiment is conducted with SPEC2006 benchmark to inspect dirty words distribution in a dirty cache line. The conclusion in their work shows that in 76.6% of cache lines (on average) the dirty words are less than 4 in majority of applications. We conduct the same experiment with our own system configuration, in which a cache line size is 64 bytes and holds 8 words. As depicted in Figure 1, the percentage of dirty cache lines which only change one word is high, at least 33% (gcc) and up to 63% (mcf). And the average number of dirty words in a dirty cache line is from 1.88 (mcf) to 3.84 (cactusADM). To sum up, the actual amount of data to be updated on each cache line is limited. If we only SET the dirty words in the memory line , we just need RESET some bits in these dirty words when we perform a write request. Therefore, we can significantly reduce the average write times for each word while still acquire desirable system performance.

To balance performance and endurance problem of PCM system, we propose Partial-PreSET, a fine-grained scheme based on PreSET. Once a cache line be-

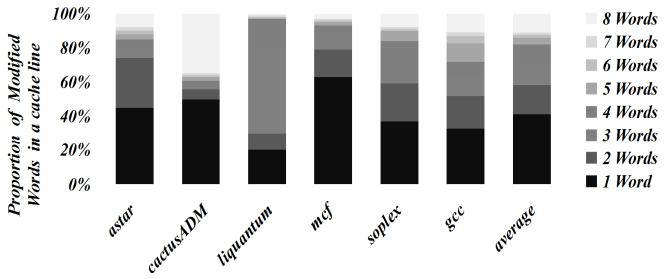

Fig. 1: Proportion of modified words in a dirty cache line

ing written, the positions of dirty words are marked and Partial-PreSET will delicately SET the dirty words in the memory line.

We discuss the extensions to cache architecture and memory system in order to facilitate Partial-PreSET. We extend dirty bit in all caches from 1 bit to 8 bits so as to indicate whether each word in a cache line is dirty. And in the PCM memory controller we add a new queue to hold Partial-PreSET requests.

Our experiment has displayed that dirty word distribution in the dirty cache line is highly skewed. For instance, The possibility that the dirty word is located in the first position is higher than other positions, so the first word may suffer SET operations(caused by Partial-PreSET) much more times. We take advantage of an effective level-wearing method proposed by Zhou et al.[6] to make the SET operations spread over the memory line to further extend the memory lifetime.

Our evaluations show that Partial-PreSET will significantly improve lifetime of PCM system by 2.79X while only lost 2 % system performance, compared with PreSET.

2 Architecting Memory System for Partial-PreSET

The key insight that motivates our work is that only a few of words in a dirty cache line is modified. We take advantage of the insight and propose an improved scheme called Partial-PreSET based on PreSET. In Partial-PreSET we only in advance SET dirty words. When a real write request arrives at PCM memory, we only do RESET operation in the dirty words in the memory line. The following section describes the policies and architecture design to enable efficient Partial-PreSET in PCM system.

2.1 Initializing Partial-PreSET Request

There are two questions in designing a system with Partial-PreSET, when a Partial-PreSET request for a given line is performed and what kind of information a Partial-PreSET request holds.

As soon as the cache line in the Last Level Cache (LLC) is marked as dirty, a PreSET request will be issued. And for one dirty cache line, PreSET operation will be performed only once. Partial-PreSET is different. A Partial-PreSET request only SET the dirty words in a memory line and it may be issued several times for one dirty cache line.

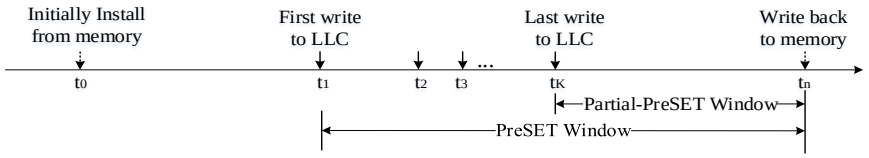

Fig. 2: The window for Partial-PreSET and PreSET

Figure 2 depicts these concepts using lifetime of a cache line from install to eviction. At t_0 the cache line is installed. PreSET will be conducted in t_1, while Partial-PreSET may be performed k times from t_1 to t_k. If the last Partial-PreSET can be finished before t_n, we will achieve the same system performance with PreSET. Although a cache line can be written for thousands of times, Partial-PreSET operation for a dirty line will be performed at most 8 times, as there is only 8 words in a memory line and we definitely do not SET the words which have been already SET in previous Partial-PreSET operation. To mark the dirty words in a cache line, the dirty bit for each line is not 1 bit, it is extended to be 8 bits, the i-th bit will index whether the i-th word is dirty. For instance, the third word of a cache line is written in t_1. We modify dirty bit to "00100000" and immediately issue a Partial-PreSET request to SET the third word of the memory line. In t_2, the third and sixth words are written. The dirty bit is changed to "00100100". And a Partial-PreSET request is performed to SET the sixth word in the memory line, as the third word has been already SET.

Except for address information, Partial-PreSET operation holds 8 bits to specify the location of dirty words so as to SET them. The specific rule is depicted in the following section.

2.2 Architecture Support

Architecture extension in cache and memory system is required to facilitate Partial-PreSET, as displayed in Figure 3.

Except for read queue (RDQ) and write queue (WRQ), memory controller has an added queue called Partial-PreSET queue (PPSQ), which holds Partial-PreSET requests. Except for address information, a PPSQ entry does not carry real data. In our simulated baseline system , we assume that there are 8 words

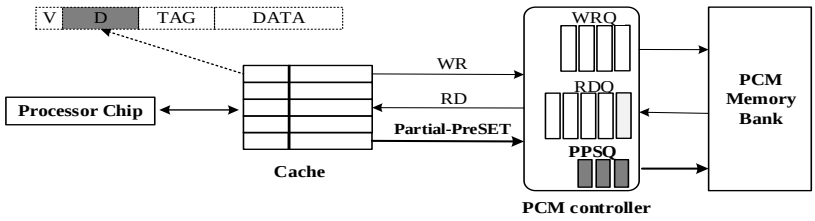

Fig. 3: Architecture Extensions to Support Partial-PreSET

in a cache line. Hence, a PPSQ entry needs another 8 bit to indicate the location of dirty words in a memory line.

In addition, dirty bit (D) in all the caches is extended to have 8 bits. As soon as the dirty L2 line is evicted to LLC, we perform a bitwise OR on the old dirty bit of the LLC line and the dirty bit of the evicted L2 line to get the updated dirty bit of the LLC line. Then, we conduct a bitwise XOR operation on the old dirty bit of the LLC line and the updated dirty bit of the LLC line to generate the dirty word positions (DWP) which need Partial-PreSET in the memory line, just as depicted in the following format.

$$DWP = OldDirtyBit \oplus UpdatedDirtyBit \tag{1}$$

when DWP equals to zero, no word in the memory line needs Partial-PreSET this time since previous Partial-PreSET operation might have already SET the words.

2.3 Scheduling Partial-PreSET at Memory Bank

Basically, we utilize the same scheduling policy with PreSET. Partial-PreSET will be serviced during idle cycles at the memory bank. Hence, it is off the critical path. The evaluation conducted in [3] shows the average memory utilization is only about 30 %, which means there is enough time to do Partial-PreSET. Partial-PreSET operation has a lower priority over write and read operations. When RDQ and WRQ of the bank are both empty, a request from PPSQ is scheduled for service.

Since read operation is in the critical path, we will cancel an ongoing Partial-PreSET operation to improve system performance when a read request arrives [7]. However, we do not cancel Partial-PreSET operation when a write request comes since write request is actually off the critical path and it can wait in the WRQ for its turn.

2.4 Combining Partial-PreSET with Row-Shifting

The dirty word distribution in the dirty cache line is skewed, which is illustrated in Figure 4. The words that are written in a row tend to be localized. Some

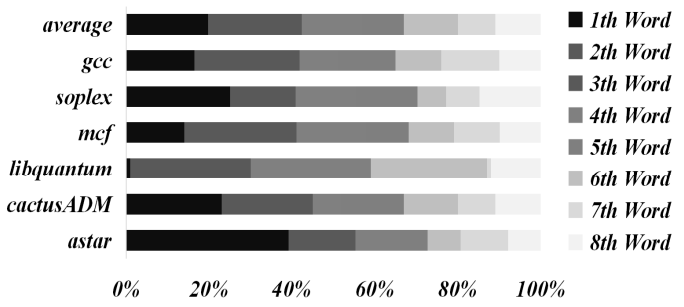

Fig. 4: Distribution of dirty words in a cache line

words in a memory line suffer write operations more times than other words. For instance, write operations in the first word account for 20 % in average of the total write operations. And Partial-PreSET as well as the following write request will both target on the first word position in the memory line, which makes this position more vulnerable.

We need a wear-leveling strategy to even out Partial-PreSET operations to the whole memory line. We take advantage of Row-Shifting scheme proposed by Zhou et al. [6]. Row is one of the basic physical structure inside the PCM device and it contains several memory lines. We shift a row (also called a page) at word granularity when it suffers a certain number of writes (including Partial-PreSET). When doing Row-shifting, we move words one by one to the next position and the last word will be shifted to the first position of the row. As such shifting causes extra writes, we can not conduct it frequently. For each row, we set an counter to record the number of writes. When the counter value exceeds a certain threshold, we perform Row-Shifting and then reset the counter. Hence, extra write caused by Row-Shifting can be extremely amortized. Based on our system configuration, dirty word distribution is effectively evened out by utilizing Row-Shifting. In addition, for the sake of fairness, PreSET will also adopt Row-Shifting.

3 Experimental Methodology

3.1 Configuration

We conduct experiments with Micro-Architectural and System Simulator (MARSS)[8] , which is a wide-used cycle-accurate full system simulator. The baseline system in our studies is depicted in Figure 5 and it follows parameters in Table 1. We use a simple processor model with eight out-of-order cores. Each core has its own L1 and L2 cache supporting MESI protocol. Eight cores share a L3 cache. In all the caches, dirty bit (D) for a line is extended to be 8 bits. The PCM-based main memory has 4 ranks of 8 banks. Each write and read request are served at the granularity of 64 bytes in one bank. Each bank has a 32-entry read

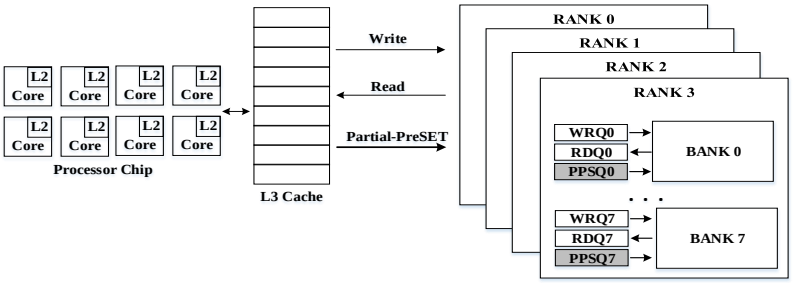

Fig. 5: Simulated Baseline System with support for Partial-PreSET

queue (RDQ) , a 128-entry write queue (WRQ) and 128-entry Partial-PreSET queue (PPSQ). The latency for each operation is following the value in PreSET. Read latency is 125ns. SET latency is 1us and RESET latency is 125ns. A write request is assumed to take 125ns, as it only contain RESET operation. A Partial-PreSET request is supposed to take 1us, since such operation SET dirty words. And Partial-PreSET requests target on dirty words, ignoring of clean words in a memory line. Read request has the highest priority among all the requests. However, when the WRQ is 80% full, write request is serviced in advance. Such scheme allows read request to have a higher priority over write request, but write requests will be finally serviced.

Table 1: Baseline Configuration

System	8-core single-issue out-of-order CMP, 3.4GHZ
L2 cache	2MB, 8-way, LRU, 64B linesize, write-back, 3ns
L3 cache	32MB, 8-way, LRU, 64B linesize, write-back, 5ns
Main memory	1 channel, 4 ranks/channel, 8 banks/rank, 32 entries read-queue/bank, 128 entries write queue/bank
PCM latency	reads : 125ns, write (SET only): 1us, write (RESET only): 125ns

3.2 Workloads

We use a representative slice of six benchmarks from SPEC2006 suite: astar, cactusADM, liquantum, mcf, soplex and gcc. These benchmarks are chosen since they have relatively heavy memory access in our simulated system and most of them are also chosen in the work of PreSET. We also use three multiprogrammed workloads and each contains 2 copies of six benchmarks. The basic information is displayed in the table 2.

Table 2: Benchmark Description

Name	Description	Name	Description
astar_ r	8 × astar	cactus_ r	8 × cactusADM
libqtum_ r	8 × libquantum	mcf_ r	8 × mcf
soplex_ r	8 × soplex	gcc_ r	8 × gcc
mix1	4 × astar, 4 × cactusADM	mix2	4× libquantum, 4 × mcf
mix3	4× soplex, 4 × gcc		

3.3 Figure of Merit

For a PCM system, supposing the size is S GB, write speed is B GBps, endurance of a word is W_{max} and the whole system will last for at most Y years, the relation [9] of each parameters are given below:

$$SystemLifetime = Y \approx \frac{W_{max} * S}{B} * 2^{-25} \tag{2}$$

Such formula works under the assumption that writes request can be issued uniform for the entire PCM capacity. Same with PreSET, we adopt the assumption. Recent works such as Security Refresh [10] and Randomized Start-Gap [11] offer a lifetime very close(97%) to ideal wear leveling at the cost of only 1% write overhead. Hence, our assumption is to some extent practical. We assume that every memory line can receive a write at the same possibility. Moreover, in a memory line, we take advantage of Row-Shifting to even out write operation to every word.

The system performance is depicted as the relative speedup [3] :

$$Speedup = \frac{ExecutionTimeOfBaseline}{ExecutionTimeWithProposedTechnique} \tag{3}$$

4 Results and Analysis

4.1 Impact on System Lifetime

We compare the relative lifetime obtained by PreSET and Partial-PreSET. Figure 6 shows that, compared with PreSET, Partial-PreSET extends lifetime of PCM system from 2.08X(cactus_r) to 4.26X(mcf_r). In average, endurance is extended by 2.79X. Partial-PreSET has better endurance performance, as it only targets on the limited dirty words of a cache line.

4.2 Average Number of Partial-PreSET Operations

For each dirty cache line, only one PreSET operation is required while two or more Partial-PreSET operations may be needed. If too many Partial-PreSET

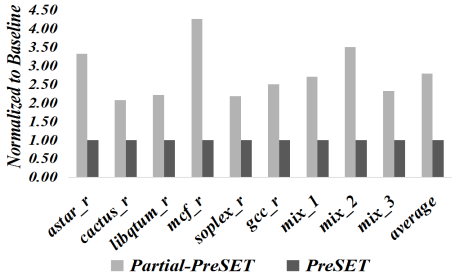

Fig. 6: Comparison of lifetime of different PCM systems

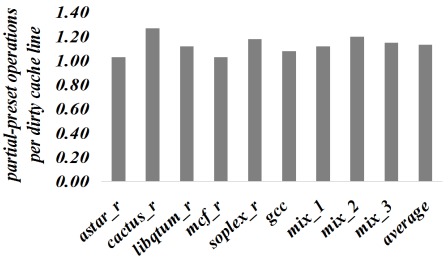

Fig. 7: The average number of Partial-PreSET per dirty line

operations are issued, they definitely delay the incoming write requests and read requests.

Fortunately, as showed in Figure 7, the number of Partial-PreSET per dirty cache line ranges from 1.03(astar_r) to 1.27(cactus_r). The average number is only about 1.13. There are two reasons for explaining the statistics. Firstly, as mentioned in the previous section, the average number of dirty words per dirty line is quite low. If there is only one dirty word in a cache line, we need only Partial-PreSET once. Secondly, Partial-PreSET operation is issued only if the dirty bit in last level cache (LLC) is changed. We adopt write-back policy in caches. If CPU writes a line in L1 or L2 successfully, CPU will not immediately write the line in LLC. When the L2 line is evicted to LLC, the LLC line will be modified. CPU may write a cache line in L1 or L2 cache for many times. However, LLC has no aware of such operation happening and it does not trigger a Partial-PreSET operation. when such dirty line which contains several dirty words is evicted to LLC from L2, LLC just need to perform one Partial-PreSET operation to SET all these dirty words in the memory line.

4.3 Coverage Rate of Partial-PreSET

For a certain dirty cache line, a serial of Partial-PreSET operations will be performed. The key point to performance improvement is that the last one of such operations has to be completed before write request arrives at that memory. As described in Figure 2, Partial-PreSET window is between last write to cache and write-back to PCM memory. If the window size is too small, we may have not enough time to complete Partial-PreSET operation. Then the incoming write request still has to do SET operation, which leads to a failure of speedup. Meanwhile, PreSET window is from the first write to cache line and write-back to memory, which means a PreSET operation have higher possibility to be finished than Partial-PreSET.

Supposing that T_{total} is the number of the total write requests and T_{hit} is the number of the write requests whose Partial-PreSET or PreSET operations

have all been done. We define coverage rate as the follow formation:

$$CoveragerRate = \frac{T_{hit}}{T_{total}} \tag{4}$$

As indicated in Figure 8, in average, the coverage rate of Partial-PreSET is only 3% less than that of PreSET.

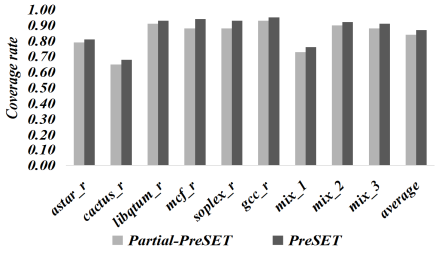

Fig. 8: Coverage rate

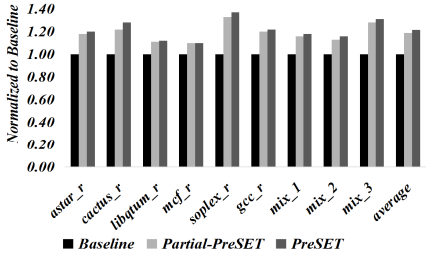

Fig. 9: Speedup for various systems

Two key facts are responsible for the statistics. Firstly, as the average number of Partial-PreSET operations for a dirty cache line is only around 1.13, which means that most of times the first Partial-PreSET operation is also the last Partial-PreSET operation for a line. Under the circumstance, the window size of Partial-PreSET and PreSET window is the same. Secondly, we use Least Recently Use (LRU) strategy to evict a cache line in LLC. As a result, when a cache line suffer the last write, it still sustains in the LLC for quite a period. Taking advantage of this time interval, the last Partial-PreSET can be completed successfully.

4.4 Impact on System Performance

Figure 9 shows the speedup of baseline, PreSET and Partial-PreSET. Compared with Baseline, Partial-PreSET gains performance improvement from 10%(mcf_r) to 33%(soplex_r). And in contrast to PreSET, Partial-PreSET only lost 2% system performance in average. The most important reason to explain the statistics is that the coverage rate of Partial-PreSET and PreSET is almost the same. Most of Partial-PreSET operation can be finished before the write request arrives, which significantly shortens the latency of incoming write.

5 Related Work

Qureshi et al.[3] propose PreSET which improves system performance of PCM system by exploiting asymmetry in write times. Write latency in PCM is highly data dependant. SET operation is almost 8X longer than RESET operation.

Hence, based on the observation, before a dirty cache line is evicted from cache, PreSET will pro-actively SET all the bits in the memory line. When such cache line being written to PCM memory, only RESET operation is required. PreSET significantly improves write performance, which leads to better system performance. However, PreSET is conducted at a coarse-granularity, which will extremely shorten lifetime of PCM system. Partial-PreSET is an improved policy to alleviate the endurance problem caused by PreSET.

Zhou et al. [6] proposed redundant bit removal in order to reduce unnecessary bit writes. Before writing the cells in a memory line, we will read the old data and compare it with the updated data from write request. Then, the write request will be performed only on the bits that have changed. Such policy is also adopted in Partial-PreSET and PreSET. When We do Partial-PreSET, we will only SET the bits that are in RESET state. Futhermore, when a write request happens, we only RESET the bits that in SET state.

After redundant bit-writes removal, the bits that are written most in a row tend to be localized, rather than spread out. Hence, Zhou et al. applied a simple shift mechanism to even out the writes in a row to all cells rather than a few cells. We adopt the Row-shifting Policy in the Partial-PreSET to further extend the lifetime of PCM. We shift a row (also called a page) at word granularity when it suffers a certain number of writes (including Partial-PreSET). when doing Row-shifting, we move the words one by one to the next position and the last word will be shifted to the first position of the row.

6 Conclusions

In Phase Change Memory, write operation is much slower than read operation. When such write request is serviced, it will lead to severe contention for read access. Qureshi et al. proposed PreSET. Before a dirty cache line is being written to PCM memory, PreSET will pro-actively SET all the bits in the memory line. PreSET significantly shortens write latency and as a result improves system performance. However, such policy may tremendously shorten lifetime of PCM system.

Our work focuses on an improved scheme based on PreSET which addresses both endurance and performance problem. This paper exploits the fundamental observation that the amount of data to be modified on a dirty line is limited. Hence, to in advance SET the whole memory line is quite expensive and unnecessary. if we only SET the dirty words, we can significantly reduce write times while still acquire desirable system performance.

We propose Partial-PreSET, a fine-grained scheme based on PreSET to balance performance and endurance problem of PCM system. Once a cache line being written, in stead of SET the bits in the whole line, we only SET the dirty words. We notice that the dirty word distribution in the dirty cache line is highly skewed. And we combine Row-Shifting with Partial-PreSET to further extend lifetime of PCM system.

To sum up, Partial-PreSET will significantly improve lifetime of PCM system by 2.79X while only lost 2% system performance, compared with PreSET. We hope that Partial-PreSET will be a promising technology adopted in the PCM system to speed up system performance as well as properly handle endurance problem.

References

1. Charles Lefurgy, Karthick Rajamani, Freeman Rawson, Wes Felter, Michael Kistler, and Tom W Keller. Energy management for commercial servers. *Computer*, 36(12):39–48, 2003.
2. M Qureshi, S Gurumurthi, and B Rajendran. Phase change memory: From devices to systems. *Synthesis Lectures on Computer Architecture*, 6(4):134, 2011.
3. Moinuddin K Qureshi, Michele M Franceschini, Ashish Jagmohan, and Luis A Lastras. Preset: improving performance of phase change memories by exploiting asymmetry in write times. *ACM SIGARCH Computer Architecture News*, 40(3):380–391, 2012.
4. Kwang-Jin Lee, Beak-Hyung Cho, Woo-Yeong Cho, Sangbeom Kang, Byung-Gil Choi, Hyung-Rok Oh, Chang-Soo Lee, Hye-Jin Kim, Joon-Min Park, Qi Wang, et al. A 90 nm 1.8 v 512 mb diode-switch pram with 266 mb/s read throughput. *IEEE Journal of Solid-State Circuits*, 43(1):150–162, 2008.
5. Mohammad Arjomand, Mahmut T Kandemir, Anand Sivasubramaniam, and Chita R Das. Boosting access parallelism to pcm-based main memory. In *International Symposium on Computer Architecture*, pages 695–706, 2016.
6. Ping Zhou, Bo Zhao, Jun Yang, and Youtao Zhang. A durable and energy efficient main memory using phase change memory technology. In *International Symposium on Computer Architecture*, pages 14–23, 2009.
7. M. K Qureshi, M. M Franceschini, and L. A Lastras-Montano. Improving read performance of phase change memories via write cancellation and write pausing. In *IEEE International Symposium on High PERFORMANCE Computer Architecture*, pages 1–11, 2010.
8. Avadh Patel, Furat Afram, Shunfei Chen, and Kanad Ghose. Marss: a full system simulator for multicore x86 cpus. In *Design Automation Conference, DAC 2011, San Diego, California, Usa, June*, pages 1050–1055, 2011.
9. Moinuddin K Qureshi, Vijayalakshmi Srinivasan, and Jude A Rivers. Scalable high performance main memory system using phase-change memory technology. In *International Symposium on Computer Architecture*, pages 24–33, 2009.
10. Nak Hee Seong, Dong Hyuk Woo, and Hsien-Hsin S Lee. Security refresh: prevent malicious wear-out and increase durability for phase-change memory with dynamically randomized address mapping. *ACM SIGARCH computer architecture news*, 38(3):383–394, 2010.
11. M. K Qureshi, J Karidis, M Franceschini, and V Srinivasan. Enhancing lifetime and security of pcm-based main memory with start-gap wear leveling. In *Ieee/acm International Symposium on Microarchitecture*, pages 14–23, 2009.