Word- and Partition-Level Write Variation Reduction for Improving Non-Volatile Cache Lifetime

SHUAI WANG, GUANGSHAN DUAN, YUPENG LI, and QIANHAO DONG,
Nanjing University

Non-volatile memory technologies are among the most promising technologies for implementing the main memories and caches in future microprocessors and replacing the traditional DRAM and SRAM technologies. However, one of the most challenging design issues of the non-volatile memory technologies is the limited write. In this article, we first propose to exploit the narrow-width values to improve the lifetime of non-volatile last-level caches with word-level write variation reduction. Leading zeros masking scheme is proposed to reduce the write stress to the upper half of the narrow-width data. To balance the write variations between the upper half and the lower half of the narrow-width data, two swapping schemes, the swap on write (SW) and swap on replacement (SRepl), are proposed. Two existing optimization schemes, the multiple dirty bit (MDB) and read before write (RBW), are adopted with our word-level swapping design. To further reduce the write variation on the partition level, we propose to exploit the cache partitioning design to improve the lifetime. Based on the observation that different applications demonstrate different cache access (write) behaviors, we propose to partition the last-level cache for different applications and balance the write variations by partition swapping. Both software-based and hardware-based partitioning and swapping schemes are proposed and evaluated for different situations. Our experimental results show that both our word- and partition-level designs can improve the lifetime of the non-volatile caches effectively with low performance and energy overheads.

CCS Concepts: • **Computer systems organization → Processors and memory architectures**; • **Hardware → Non-volatile memory**;

Additional Key Words and Phrases: Non-volatile memory, last-level cache, wear-leveling, narrow-width value, cache partitioning

ACM Reference format:
Shuai Wang, Guangshan Duan, Yupeng Li, and Qianhao Dong. 2017. Word- and Partition-Level Write Variation Reduction for Improving Non-Volatile Cache Lifetime. *ACM Trans. Des. Autom. Electron. Syst.* 23, 1, Article 4 (July 2017), 18 pages.
https://doi.org/10.1145/3084690

1 INTRODUCTION

For modern microprocessors today, the DRAM and SRAM are still the dominant technologies to implement the main memory and on-chip caches. However, due to the drawbacks of the DRAM

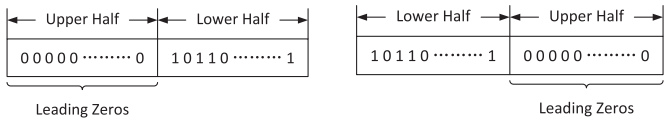
Authors' addresses: S. Wang, G. Duan, Y. Li, and Q. Dong, State Key Laboratory of Novel Software Technology, Department of Computer Science and Technology, Nanjing University, Nanjing, Jiang Su, China 201146; emails: swang@nju.edu.cn, {guangshan_duan, liyupeng, dongqianhao}@smail.nju.edu.cn.

https://doi.org/10.1145/3084690

and SRAM, such as the high leakage power and reliability issues against soft errors, recent research has been working on developing new technologies to replace the traditional implementation. The non-volatile memory technology, such as phase change memory (PCM), spin-torque transfer RAM (STT-RAM), and resistive RAM (ReRAM), is one of the most promising technologies to build the memories and caches in future microprocessors (Qureshi et al. 2009; Schechter et al. 2010; Zhao et al. 2014; Wen et al. 2013, 2014; Ahn et al. 2014; Kang et al. 2015).

Besides their non-volatility, the non-volatile memory technologies can avoid the refreshing of the memory cell in the DRAM design and also reduce the leakage power. They can also provide higher cell density compared to the traditional DRAM and SRAM technologies. For the reliability, the non-volatile memory is more robust against particle-induced soft errors. The non-volatile memories are among the most promising techniques to realize low-power computing. A non-volatile processor (NVP) implemented with non-volatile memories will benefit from the zero-standby power, nanosecond sleep/wakeup speed, high resilience to power failures, and high resilience to soft errors (Wang et al. 2014a). However, compared to traditional DRAM and SRAM, the main disadvantages for the non-volatile memory are higher write latency, higher write energy, and less write endurance (Joo et al. 2010). Especially if we want to adopt the non-volatile memory technology at the on-chip cache level, the write endurance will become worse compared to the lower-level memory hierarchy, such as main memory. If the cache is closer to the CPU (e.g., level one cache), then it will have more write frequency. The high frequency of write operations on caches will reduce the lifetime of the non-volatile caches dramatically. However, even for the last-level cache, the reliability is also very important, because it will have an impact on the correctness of the execution or make the processor fail to work if there are failures in the cache cells. For the exiting cache management schemes for traditional caches, none of them is write variation-aware. Although recent work has proposed some wear leveling schemes to increase the lifetime of the non-volatile main memory (Zhou et al. 2009; Qureshi et al. 2009; Seong et al. 2010; Luo et al. 2014), we cannot directly apply their schemes to the on-chip caches due to the different operational behaviors between main memory and on-chip caches. Therefore, techniques targeted at improving the lifetime of the non-volatile on-chip caches are needed.

In this work, we first propose the microarchitectural level solution to improve the lifetime of the non-volatile last-level cache (LLC) on the word level by exploiting the narrow-width values. The reason we currently target the last-level cache is that it has less write frequency and longer access latency compared to its upper-level caches and is more feasible to implement. The presence of narrow-width values in last-level cache is first studied and a leading zeros masking (LZM) scheme is proposed. To mitigate the write variation on our LZM design, two swapping schemes, the swap on write (SW) and swap on replacement (SRepl), are proposed to balance the write operations to the upper and lower halves in our narrow-width data. We also adopt two existing optimization schemes, the multiple dirty bit (MDB) and read before write (RBW), to reduce the unnecessary writes to the cache. To further improve the lifetime of the non-volatile cache, based on the observation that different applications demonstrate different write behaviors during the execution, we propose a cache partitioning and swapping scheme to improve the lifetime on the partition level. We first partition the shared last-level cache for different applications and categorize them into two groups: the cache write-intensive and non cache write-intensive applications. Both software-based and hardware-based partitioning schemes are proposed for different situations. To improve the lifetime, the swapping schemes are proposed to balance the write variations on the partition level. Static and dynamic swapping schemes are proposed for software-based and hardware-based designs. The experimental results indicate that both our word- and partition-level designs can improve the lifetime of the non-volatile cache effectively with negligible performance and energy overheads.



(a). Narrow Width (b). After Swapping

Fig. 1. Narrow-width values (leading zeros) in the cache.

The major contributions of our work are summarized as follows:

- We proposed a word-level scheme to improve the lifetime of the non-volatile LLC by exploiting the narrow-width values, and it can be adopted with other existing technique, such MDB and RBW.
- We proposed two partition-level schemes, the software-based partitioning and hardware-based partitioning, to further improve the lifetime of the non-volatile LLC for different situations.

The rest of the article is organized as follows. In the next section, we discuss our word-level write variation reduction design by exploiting narrow-width values. In Section 3, we provide details on our partition-level write variation reduction design by cache partitioning. We present our experimental setup and results in Section 4. Section 5 discusses the related work, and Section 6 presents the conclusion.

2 WORD-LEVEL WRITE VARIATION REDUCTION BY EXPLOITING NARROW-WIDTH VALUES

2.1 Narrow-Width Value

The narrow-width values in high-performance microprocessors have been well studied and exploited for performance, power, and reliability optimizations in register files and caches (Brooks and Martonosi 1999; Ergin et al. 2006; Kong and Chung 2012). In general, values with leading "0"s are referred to as narrow-width values. Therefore, we can only store portion of the data without the leading "0"s and restore it by the zero-extension logics when reading it out. Figure 1(a) shows an example of the narrow-width value we defined in this work. First, we divide the data into two halves, the lower half and the upper half. If the upper half of the data are all zeros, then we define this type of data as narrow-width data. Therefore, when we write this type of data into the cacheline, the write operation of the upper-half bits can be avoided, and thus the write stress to the cacheline can be reduced. We call this scheme the leading zeros masking (LZM) scheme, which has been used in energy reduction in traditional cache design (Villa et al. 2000). Note that the data unit we mentioned here can be a byte, 16 bits, 32 bits, or 64 bits in the cacheline. If we choose the byte-level study for the narrow-width data, then the percentage of the narrow-width data in the cache may be high. However, the area overhead will be also high due to the extra bit N (1/8 = 12.5%) needed for indicating whether the data is narrow-width or not. Therefore, we choose the 64-bit as the data unit in our following study for its low area cost.

2.2 Swap for Write Variation Balancing

The leading zeros masking scheme can reduce the number of writes to the upper half of the narrow-width data in the cacheline and thus improve its lifetime. However, the number of writes to the lower half of the narrow-width data will remain the same. Therefore, the lifetime of the

non-volatile cells in the lower half will not be improved. To further improve the lifetime of the lower half, we propose to swap the data between the lower half and the upper half periodically based on certain strategies. Figure 1(b) shows the narrow-width data after the swapping. If proper timing for the swapping can be chosen, then the write stress to both halves will be reduced and the write variations between two halves will be well balanced. However, if a very frequent interval is chosen, the overhead for the swapping will be high. If we choose a very large swapping interval, then the write variations between two halves will not be well balanced. Therefore, we propose two strategies to do the swapping: the SW and the SRepl.

2.2.1 Swap on Write. For this strategy, we do our swapping on every write operation to the data unit. For example, when we need to write a narrow-width value into the cache, the swap status of the current data unit will be checked. If the current data in the cache is in a non-swapped status, then we will swap the lower half and the upper half of the incoming narrow-width value and only store the lower half of the narrow width value into the upper half of the data unit in the cacheline. If the current data in the cache is in a swapped status, then normal leading zeros masking write will be performed and only the lower half of the incoming value will be written into the lower half of the data unit in the cacheline.

2.2.2 Swap on Replacement. Since the swap operation for the SW scheme is based on the write operation to the cache, it will diminish the space for integrating our design with the existing optimizing scheme, such as the RBW that we will discuss later. Therefore, we propose the SRepl scheme, where the swapping is done on every replacement of the cacheline. During the cacheline replacement, if a narrow-width value needs to be written into the cacheline, we will check the swap status of the current data unit and do the similar swapping to that in the SW scheme. Due to the reduced swapping frequency of the SRepl scheme, the effectiveness of the write variations balancing in SRepl scheme will be reduced compared to the SW scheme.

2.3 Microarchitecture of the Word-level Write Variation Reduction Design

To implement our leading zeros masking and swapping design, one additional narrow bit (N) for each data unit (64-bit) and one additional swap bit (S) for each cacheline are needed. The narrow bit (N) indicates whether the value in the current data unit is narrow width or not. The swap bit (S) indicates whether the data in the current cacheline is in the swapped or non-swapped status. Note that since the access to last-level cache is cacheline based, we choose the cacheline-level swap bit stead of one S bit per 64-bit unit to further reduce the space overhead of our design. For instance, for a last-level cache with a 128-byte cacheline, the space overhead of our non-volatile cache compared to the data array is only $(1/64 + 1/(128\times8) = 1.7\%)$.

Figure 2 shows the block diagram of our proposed word-level write variation reduction design with leading zeros masking and swapping. In the tag array, one narrow bit (N) is added for each 64-bit data unit in the data array, and one swap bit (S) is added for each cacheline. During a cache write, the narrowness of the incoming data will be checked first. If the data are not narrow width, then a normal write will be performed and the N bit will be set to zero. If the data is a narrow-width data, then the S bit will be checked, and the swapping will be done for the SW scheme. Otherwise, the swapping will be only done during cache write caused by the cache replacement for the SRepl scheme. Based on the status of the S bit, the lower half of the narrow-width value will be written into the lower or the upper half of the data unit as discussed above. During a cache read, the N bit will be checked first. If the N bit is zero indicating that the data is not narrow width, then the data will be readout in its current format. If the N bit is 1, then the S bit will be also checked. If the S bit is zero, then it means that the narrow-width data is currently in the

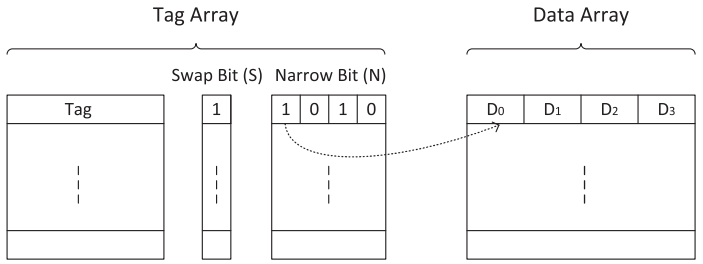


Fig. 2. Block diagram of the proposed word-level write variation reduction design.

non-swapped status. We can read out the lower half of the data and do the zero-extension to restore the original 64-bit value. If the S bit is 1, then we need to readout the upper half of the data and do the zero-extension.

2.4 Combining Existing Techniques

To further improve the lifetime of our proposed non-volatile cache with word-level optimizations, we adopt two existing optimization schemes to reduce the write stress on the cacheline.

2.4.1 Multiple Dirty Bit. The MDB scheme has been proposed for improving the reliability of the data cache against soft errors (Wang et al. 2009). The idea is that not all the words in a dirty cacheline are modified (dirty) during the write-back. Therefore, if we can have one dirty bit for each word, we can only write back the dirty words into the lower cache to avoid unnecessary writes for the clean words and also save the energy. We adopt this MDB scheme for reducing the write stress to the last-level cache and thus improving its lifetime.

2.4.2 Read Before Write. The RBW scheme was previously proposed for reducing the amount of writes in the PCM memory design (Zhou et al. 2009; Joo et al. 2010). A pre-read operation is performed before the write to eliminate the write of the new value same to its previous value. By combining our proposed MDB scheme, we can apply a selective RBW by only pre-reading the data unit that the dirty words of upper-level cache will be written into to further reduce the overhead caused by the pre-read operation.

2.5 Power Consumption

Our word-level write variation reduction design not only can improve the lifetime of the non-volatile caches but also might reduce the power consumption of the entire cache architecture. There are mainly three factors that can incur power reduction in our cache design: (a) the leading zeros masking scheme that can avoid the write power consumption for the upper half of the narrow-width data in the last-level non-volatile cache, (b) the multiple dirty bit scheme that can reduce the write power consumption caused by the writebacks from the upper-level cache, and (c) the read before write scheme that might reduce the power consumption by avoiding the unnecessary writes to the unchanged data. However, there are also some factors that can increase the power consumption in our design: (a) the swapping scheme that will cause additional power consumption and (b) the read before write scheme that will increase the power consumption due to the extra reads. Therefore, we will carefully study the power consumption of our design in the experimental evaluation.

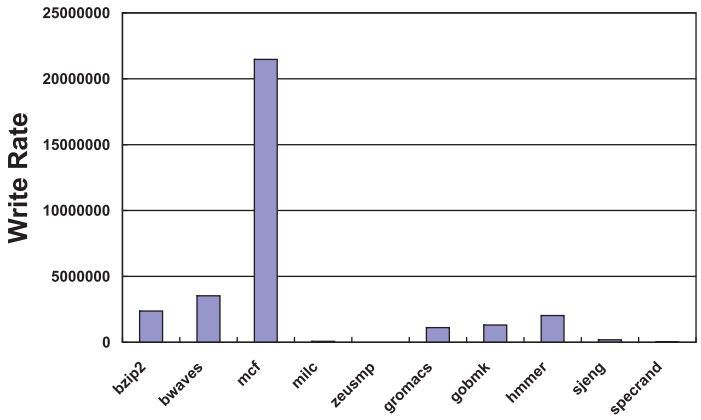


Fig. 3. Write rates of the last-level cache in our simulated processors.

3 PARTITION-LEVEL WRITE VARIATION REDUCTION BY CACHE PARTITIONING

3.1 Write-Intensive and Non Write-Intensive Applications

Different applications might demonstrate different memory access behaviors. Some of the applications are memory-intensive applications, which means that there are heavy memory accesses in this applications. Some of the applications are non memory intensive, or CPU intensive, which means that there are fewer memory accesses in these applications, and the performance of these applications is normally determined by the CPU. Based on this observation, we conduct our study on the SPEC2006 benchmarks and Figure 3 shows the comparisons of the write rates (write frequencies) of the last-level cache in our simulated processor for different applications. The results confirm that there might be huge differences in the write behaviors to the last-level caches for different applications. For example the *mcf* application has a very high write rate in the LLC during its execution, which is more than 1K times compared to the applications such as *specrand* and *zeusmp*.

The different cache write behaviors of different applications provide us an opportunity to manage and balance the write variation in the cache, if we can know or control the exact place in the cache where an application will write. The cache partitioning techniques could be one of the solutions.

Cache partitioning schemes, especially for shared low-level caches in the multi-thread/multi-core processors, have been studied to optimize cache utilization and reduce the contention of different applications, and thus to improve the system performance (Qureshi and Patt 2006; Cho and Jin 2006; Wang and Chen 2014). We propose to utilize the cache partitioning schemes to improve the lifetime of the non-volatile cache in the processor.

Based on the observation in the previous section, the cache write behaviors of different applications might vary significantly. If we can first partition the cache for different applications with different cache write behaviors, and then conduct the partition swapping between the write-intensive applications and non write-intensive applications, then the write variations of different partitions will be balanced, and thus the lifetime of the entire cache will be improved. Therefore, we propose two strategies to do the cache partitioning: the software-based partitioning and the hardware-based partitioning.

3.2 Software-based Partitioning

For the software-based cache partitioning, the page coloring is a widely used scheme to partition the shared cache for concurrent running applications to reduce the cache conflicts through the

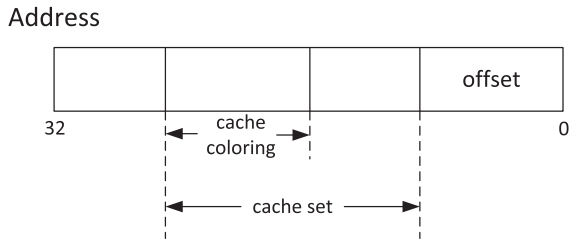


Fig. 4. The address mapping for software-based partitioning by page coloring.

OS level (Cho and Jin 2006; Lin et al. 2008). Therefore, we propose to adopt the page coloring scheme to do the cache partitioning on the software level. In our proposed page coloring-based cache partitioning design, the cache is divided into different partitions, which is called page colors, as shown in Figure 4. The physical pages are divided into several regions based on the cache size, the page size, and the associativity of the cache. Cache coloring maps every memory region to a unique color in the cache. A small mapping table is used to record this mapping, and, by changing the mapping between the physical pages and the cache colors, we can map a particular region to desired cache color. This cache coloring scheme does not introduce any changes in the virtual address to physical address mapping. Therefore, the implementation overhead is small. The size of the mapping table is also very small, which indicates negligible overhead. Our proposed cache partitioning will map the partitions with the same color to a single application/thread. Therefore, the cache partitions with the same color can be accessed only by the specific application. The cache conflicts caused by different applications will be reduced in this design, which might improve the overall performance for the processor. However, the cache size that can be used by a single application will be reduced due to the cache partitioning, which might hurt the performance. Therefore, the performance impacts of our proposed cache partitioning scheme need to be studied and evaluated in details. Note that the advantage of the software-based cache partitioning scheme is its low hardware modification and overhead.

3.3 Hardware-based Partitioning

The hardware-based partitioning schemes have also been studied to improve the performance of the shared cache (Suh et al. 2004; Qureshi and Patt 2006), among which the cache way partitioning scheme is one of the most simple and effective methods to do the partitioning. Therefore, we propose to adopt the way partitioning scheme in our design for non-volatile cache lifetime improvement. In our design, we partition the cache by assigning different cache ways to different applications/threads. Each cache way is added with an application/thread id that will control the specific application that was assigned into the way, as shown in Figure 5. The advantages of the hardware-based partitioning schemes are that they are transparent to the operating system or software and can be done dynamically.

3.4 Swap for Write Variation Balancing

After the partitioning, the cache partitions assigned to the write-intensive applications will suffer from a high write stress, which means shorter lifetime. The cache partitions with non write-intensive applications will have less write accesses and longer lifetime. However, the lifetime of the entire cache depends on the first cell that fails. Therefore, we need to further balance the write accesses for different partitions.

To further improve the lifetime of the entire cache, we propose the partition swapping scheme to balance the write access of different partitions. For both software-based and hardware-based

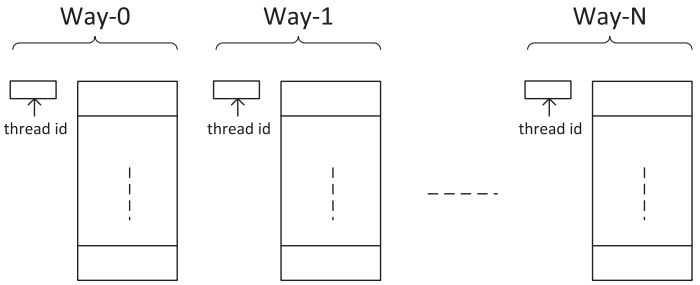


Fig. 5. The proposed hardware-based cache way partitioning scheme.

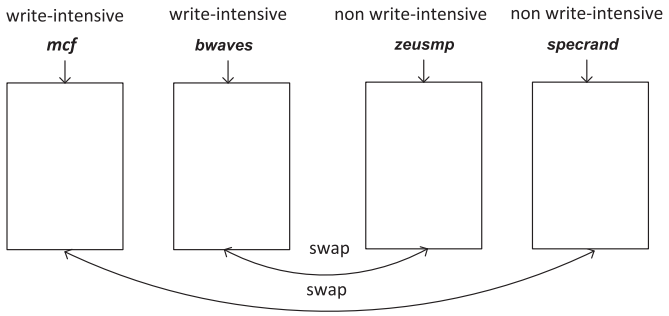


Fig. 6. Swapping for write variation balancing.

partitioning schemes, we propose to swap the partitions between write-intensive applications and non write-intensive applications periodically. For example, as shown in Figure 6, we can swap the partition assigned with *mcf* applications (write-intensive) with the partition with *specrand* applications (non-write intensive) periodically. Note that the partition swapping will incur cache flushing, which might hurt the performance if the flushing occurs too frequently.

3.4.1 How to Determine Write-Intensive and Non Write-Intensive Applications. To do the cache partition swapping in our design, we need to identify the cache write-intensive and non-intensive applications first. The easiest way is to do the profiling for each applications before running. In the system where the applications that will run are fixed or can be known before running, we can profile the write rates of the last-level cache for each applications first, similarly to what we did in Section 3.1. The software-based partitioning scheme can be used if we can know the cache write behaviors of the application ahead.

For the system where the profiling cannot be conducted before application running, the hardware-based partitioning scheme can be adopted. We can add a write counter to each cache partition. After certain period of time, the partition with high write number in the counter will be swapped with the partition with the low write number in the counter.

3.4.2 When to Swap. Another major issue needed to be solved in our design is when to do partition swapping. If a proper time interval for the swapping can be chosen, then the write stress to different partitions will be reduced, and the write variations will be well balanced. However, if a very frequent interval is chosen, then the overhead for the swapping will be high, since the swapping of the partitions will involve cache flushing, which will impact the performance and energy consumption. If we choose a very large swapping interval, then the write variations for

Table 1. Configurations of the Simulated Processor

Processor Core	
Core Architecture	ALPHA
Core Frequency	2GHz
Number of Cores	4
Memory Hierarchy	
L1 ICache	64KB, 2 ways, 64B blocks
L1 DCache	64KB, 2 ways, 64B blocks
L2 UCache	4MB, 8 ways, 128B blocks
Replacement Policy	LRU
Cache Coherence Protocol	MOESI
Interconnection	CrossBar

Table 2. Configurations of the Simulated Processor

Workloads	Benchmarks
1	mcf, gromacs, milc, zeusmp
2	bzip2, bwaves, zeusmp, sjeng
3	bwaves, mcf, zeusmp, specrand
4	bwaves, gromacs, sjeng, specrand
5	bzip2, gromacs, milc, sjeng
6	bzip2, mcf, milc, specrand
7	mcf, gobmk, milc, zeusmp
8	bzip2, gobmk, hmmer, specrand
9	gromacs, gobmk, hmmer, sjeng
10	bwaves, zeusmp, gobmk, hmmer
11	bzip2, gobmk, sjeng, specrand
12	zeusmp, gromacs, hmmer, sjeng

different partitions will not be well balanced. Therefore, we need to choose the swapping intervals carefully.

4 EXPERIMENTAL EVALUATION

4.1 Experimental Setup

We derive our simulator from SimpleScalar (Burger and Austin 1997) and Gem5 (Binkert et al. 2011) to model the high-performance microprocessor in our experiments. To model the cache/memory hierarchy, cache coherence protocols, and interconnection network, the module Ruby for Gem5 is adopted. For the processor architecture and the ISA, we use ALPHA for our full-system simulation. Table 1 gives the detailed configuration of the simulated microprocessor. For experimental evaluation, we use the SPEC CPU2006 benchmark suite compiled for the Alpha Instruction Set Architecture (ISA). Ten benchmarks are randomly selected for our experimental evaluation. Eight workloads mixed with cache write-intensive and non cache write-intensive applications are also chosen for evaluating our partition-level write variation reduction design as shown in Table 2. For evaluating the energy consumption in our design, we choose the STT-RAM implementation for our non-volatile caches. Our power model is built on top of McPat (Sheng et al. 2009) (at 22nm technology) and the STT-RAM caches are modeled by the NVSim (Dong et al. 2012).

4.2 Lifetime Metrics

Since the lifetime of the no-volatile memory cell (bit) is depending on the amount of writes to the cell, we define our Lifetime Improvement (LI_W) for a data unit (word) as follows:

$$LI\_W = MIN\left(\frac{W_{upper\_org} - W_{upper\_impr}}{W_{upper\_impr}}, \frac{W_{lower\_org} - W_{lower\_impr}}{W_{lower\_impr}}\right), \tag{1}$$

where W_{upper_org} is the number of bits needed to be written into the upper half in the original cache design and W_{lower_org} is the number of bits needed to be written into the lower half in the original cache design. W_{upper_impr} is the number of bits needed to be written into the upper half in our lifetime improvement scheme and W_{lower_impr} is the number of bits needed to be written into the lower half in our lifetime improvement scheme. Since the lifetime of the cache is depending on the first cell that fails. Therefore, we define our lifetime improvement for the data unit (word-level) as the minimum of lifetime improvement for the upper half and the lower half.

To evaluate our partition-level write variation reduction, we use the similar lifetime improvement metric in Wang et al. (2013). First, we define the coefficient of inter-partition variations as follows:

$$InterPV = \frac{1}{W_{avg}}\sqrt{\frac{\sum_{i=1}^{M}\left(\frac{\sum_{j=1}^{N} w_{ij}}{N} - W_{avg}\right)^2}{M-1}}, \tag{2}$$

where M is the number of partitions in the cache and N is the total number of cachelines in one partition. w_{ij} is the write count of the jth cacheline in the ith partition. W_{avg} is the average write count of the cacheline defined as

$$W_{avg} = \frac{\sum_{i=1}^{M}\sum_{j=1}^{N} w_{ij}}{MN}. \tag{3}$$

We further define our Lifetime Improvement metric (LI_P) for partition-level write variation reduction as follows:

$$LI\_P = \frac{W_{avg\_org}(1 + InterPV_{org})}{W_{avg\_imp}(1 + InterPV_{imp})} - 1, \tag{4}$$

where *org* means the original cache design (baseline) and *imp* means the improved cache design with the partition-level variation scheme.

4.3 Experimental Results and Analysis for Word-Level Write Variation Reduction

4.3.1 Narrow-Width Values. To study our leading zeros masking scheme, we first conduct the experimental analysis on the percentage of the narrow-width values in our simulated last-level cache. As mentioned above, the data unit we choose is 64 bit. Figure 7 shows that on average 54.6% of the values in the last-level cache is narrow width, which means that 54.6% of the 64-bit data in the cache have all zeros in their leading 32 bits. Therefore, our leading zeros masking scheme has very high possibility in reducing the write stress in the upper half of the data unit.

4.3.2 Lifetime Improvement. Only applying the leading zeros masking scheme will not reduce the amount of write operations to the lower half of the data unit. Therefore, we proposed two swapping schemes, the SW and the SRepl, to further balance the write variations between the lower and upper halves. We first apply the SW scheme to balance the write variation and Figure 8 shows that the lifetime of our non-volatile cache will be improved by 43.0% on average. Note that the improvement rate of the *hmmer* is relatively low, because the percentage of the narrow-width values in the application is very low as shown in Figure 7. Therefore, the effectiveness of our lifetime improvement scheme will be hurt.

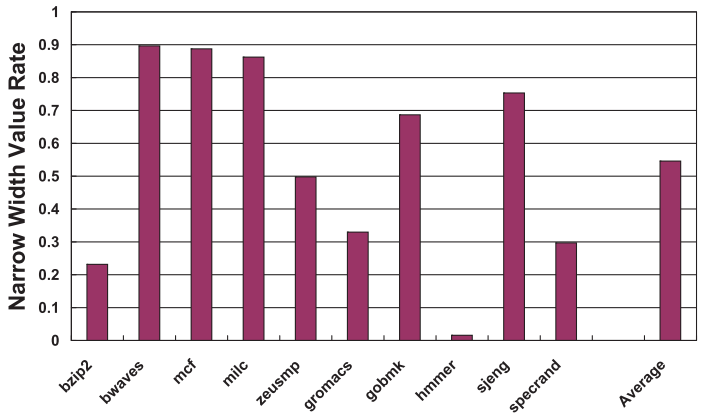


Fig. 7. The percentage of the narrow-width values (leading zeros) in the last-level cache.

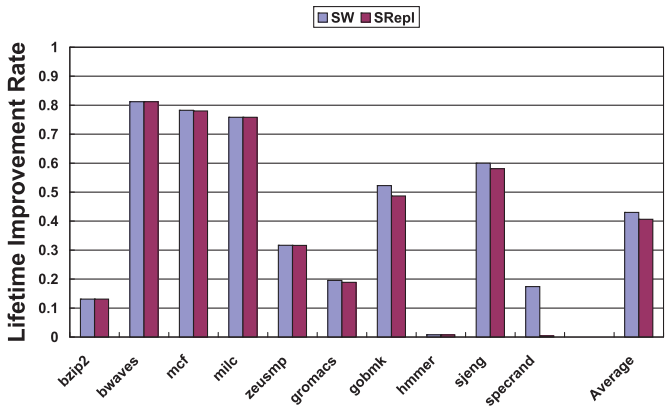


Fig. 8. The lifetime improvement rate by applying the SW and SRepl schemes.

Figure 8 also shows that if we apply the SRepl scheme, the lifetime improvement rate will be reduced to 40.7%, which is slightly less than the SW scheme. However, as we mentioned in Section 2, if we conduct the swapping on every write, the possibility for avoiding writing the same value will be much lower for the RBW technique. The energy saving and lifetime improvement will be hurt. Therefore, we choose the SRepl scheme as the default scheme for our following study.

The further study the design space of our proposed scheme, we conducted the sensitivity analysis for the size of the swapped data and the results in Figure 9 show that the lifetime improvement rates increase to 53.2% and 55.0% for the 32-bit and 16-bit level swapping with SRepl. However, the area overhead of the 32-bit and 16-bit level swapping will increase significantly, which is the similar idea for choosing the granularity for the parity/ECC coding in the reliable system design. Therefore, we choose the 64-bit level swapping in our study.

To further improve the lifetime of our non-volatile cache, we adopted two existing optimization schemes, the MDB and RBW schemes. For the space overhead consideration, we choose a 64-bit level dirty bit for the upper-level cache, that is, the data cache in our study. For the RBW scheme, we also consider the 64-bit level control for selective pre-read combined with the MDB scheme, which means that for a write-back dirty word (64-bit), if the word is the same as the old word in the

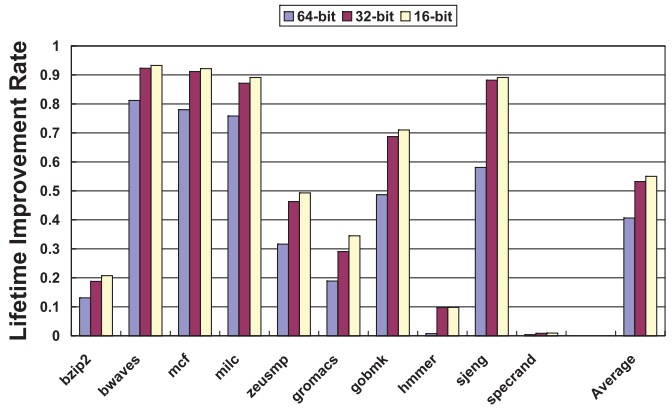


Fig. 9. The lifetime improvement rates for 64-bit, 32-bit, and 16-bit level swapping schemes with SRepl.

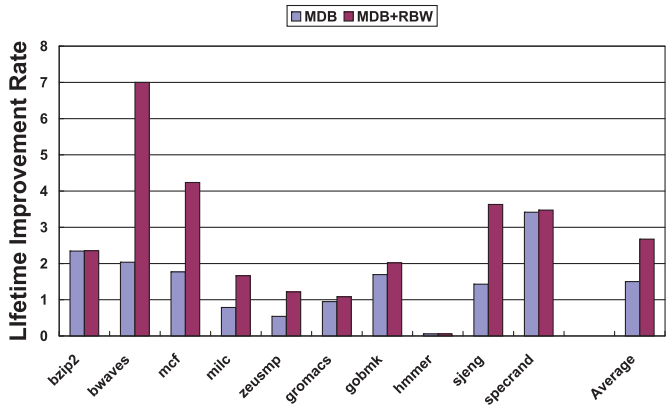


Fig. 10. The lifetime improvement rate by combining the MDB and RBW schemes.

cache, the write operation will be avoided. If two values differ, then the dirty word will be written into the cache according to the proposed leading zeros masking and swapping schemes. Figure 10 shows that if we only adopt the MDB scheme, the lifetime improvement rate will be improved to 150% on average. If we combine the MDB and RBW schemes, then the lifetime improvement rate will further increase to 267% on average.

4.3.3 Area, Performance and Power Evaluation. For the area overhead, as we discussed in Section 2.3, the major overhead is introduced by the narrow bit (N) and swap bit (s), which is 1.7% compared to the data array for the simulated 128-byte cacheline. The area overhead from the NVSim (Dong et al. 2012) simulation is 1.5% for the entire cache. For the performance overhead, according to Wang et al. (2009) and Joo et al. (2010), the MDB and RBW schemes will have negligible performance impact on the performance of the processor. Therefore, the major performance overhead in our design could be caused by the swapping. However, our design does not adopt additional cycles to do the swapping, which may have the impact on the performance when the cache needs to be accessed during the swapping. Instead, we only do the swapping during the write or replacement, which can be fit into the write/replacement cycles and does not have much impact on the performance. For the power consumption, as discussed in Section 2.5, there are some factors,

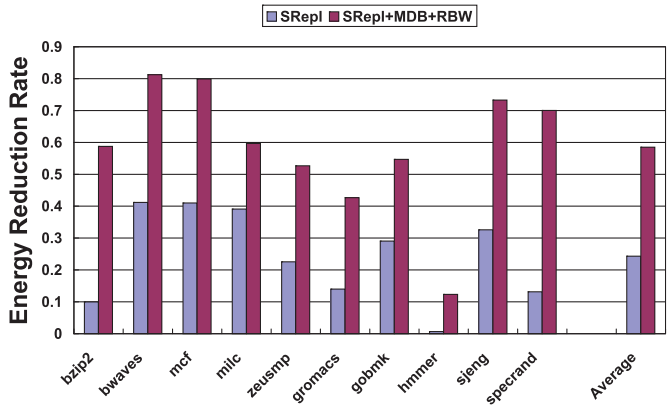


Fig. 11. The energy consumption reduction rates for the SRepl only and the combined SRepl, MDB, and RBW scheme.

such as the LZM scheme, that can lead to low power design. There are also some factors, such as the swapping, that can cause additional power consumption. Our experimental results show that our SRepl scheme can achieve a 24.3% energy reduction and we can further reduce the energy consumption by 58.5% with the SRepl, MDB, and RBW schemes, as shown in Figure 11.

4.4 Experimental Results and Analysis for Partition-Level Write Variation Reduction

4.4.1 Static Swapping Scheme for Software-based Cache Partitioning. For the software-based cache partitioning scheme, we divide the last-level cache into four colors based on the number of applications concurrent running in our simulated processor. Different applications will be mapped to different partitions with different colors. We assume that the profiling can be conducted before the execution. Therefore, the OS knows which partitions need to be swapped after a certain period of time, which is named as static swapping.

As we discussed in Section 3, a small swapping interval will hurt the performance due to cache flushing. Therefore, a relatively large swapping interval would work well for our software-based static swapping scheme. Based on our simulation study, the swapping interval can be 1s, 1 minute, 1 hour, or even 1 day, if the applications running on the system are fixed and pre-known. If a small interval is used, then the performance overhead will increase. If a large interval is adopted, then the effectiveness of write balancing of this scheme might be reduced. In our experiments, we choose a 4M-cycle swapping interval to evaluate the effectiveness of our scheme in write variation balancing. The results in Figure 12 show that our software-based cache partitioning and static swapping scheme will improve the lifetime of the non-volatile cache by 92.8%.

We also study the performance impact of our proposed scheme and the results in Figure 13 show that our software-based partitioning and static swapping scheme only incurs 1.2% performance loss.

4.4.2 Dynamic Swapping Scheme for Hardware-based Cache Partitioning. For the hardware-based cache partitioning, we propose a dynamic swapping scheme to handle situation that the applications running on the system cannot be profiled or pre-known. In our dynamic scheme, we propose to add a counter to each partition to record the number of writes to that partition. In our simulated system, where there is an eight-way set-associative last-level cache for four concurrent running applications, each partition will contain two cache ways. After a certain predefined time interval, we check the write counters of each partition and do the swapping based the contents of

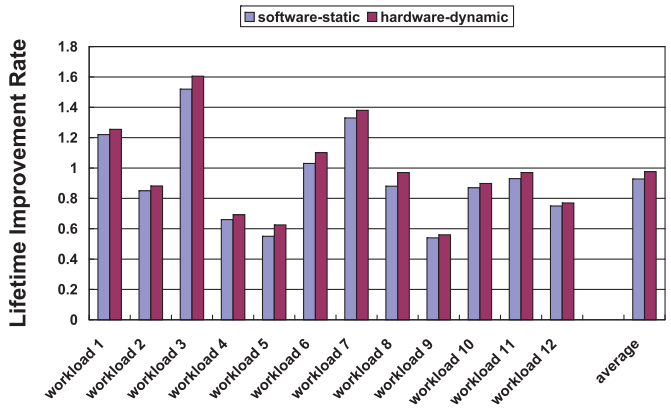


Fig. 12. The lifetime improvement rates for our software-based and hardware-based partitioning and swapping schemes.

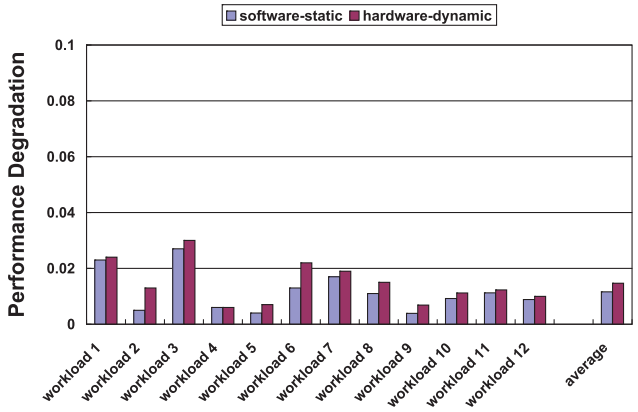
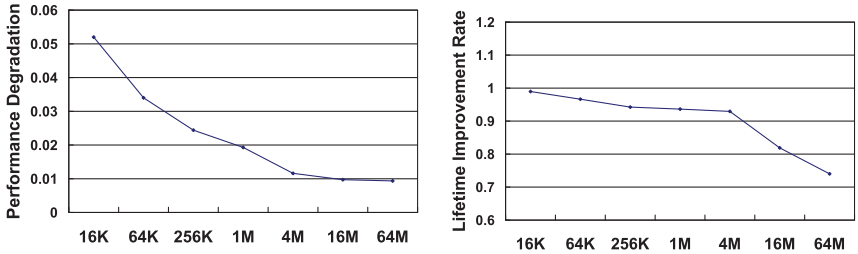


Fig. 13. The performance degradation of our software-based and hardware-based partitioning and swapping schemes.

the counter. The partition with the highest write number in the counter will be swapped with the one having the lowest write number. The partition with the second-highest write number will be swapped with the one with the second-lowest write number, and so on. After the swapping, write counters of all the partitions will be reset.

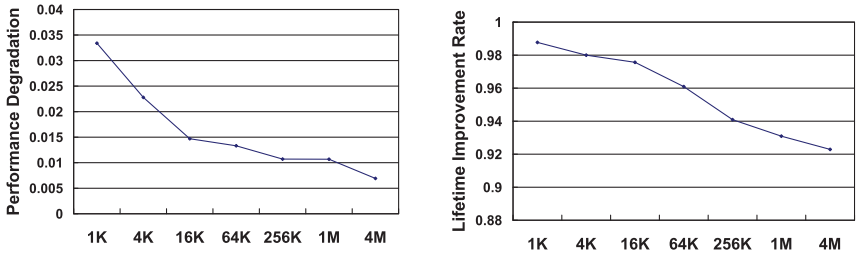
The major design issue in the dynamic swapping scheme is also how to choose the swapping interval, similar to the static scheme. However, in the hardware implementation, we can use a much smaller time interval compared to software-based design. In addition, a large time interval will incur the large write counters to avoid the saturation of the counter before the end of the time interval, which will increase the hardware overheads. Based on our study, we adopt a 16K-cycle interval for the swapping to minimize the hardware and performance overhead, while maximizing the write variation balancing in our design. The experimental results in Figure 12 show that our hardware-based partitioning and dynamic swapping scheme will increase the lifetime of the nonvolatile cache by 97.6%, with a small 1.5% performance degradation as shown in Figure 13.

4.4.3 The Swapping Interval Sensitivity Study. As discussed above, the selection of the swapping interval is one of the critical issues in our cache partitioning and swapping design. A small



(a)Performance degradation (b)Lifetime improvement

Fig. 14. The average performance degradation and lifetime improvement of our software-based schemes with different swapping time intervals.



(a)Performance degradation (b)Lifetime improvement

Fig. 15. The average performance degradation and lifetime improvement of our hardware-based schemes with different swapping time intervals.

swapping interval will increase the wear-leveling frequency that will improve the cache lifetime, but it will also increase the performance and energy overheads caused by the cache swapping.

For the software-based scheme, we assume that all the applications are known and can be profiled before running. We use a relatively large interval to reduce the overhead caused by the software partitioning and swapping. Figure 14 shows the average performance and lifetime improvement of the software partitioning with different time intervals for all workloads. From the results, we can see that the 4M-cycle interval can achieve a high lifetime improvement with low performance overhead.

For the hardware-based scheme, we prefer to choose a relatively small time interval, because of the write counter limitation and less swapping overheads in the hardware implementation. The results in Figure 15 show that the 16K-cycle interval is a good choice for our hardware-based design.

4.4.4 Area and Energy Overheads of Cache Partitioning and Swapping. For the hardware overheads, the software-based scheme nearly introduces no hardware overhead. The major hardware overheads introduced by the hardware-based scheme are thread id tags and write counters for each cache way. However, compared to the size of the data and tag array of each cache way, their overheads are negligible. For example, in our simulated processor, for a 2-bit thread id tag and 15-bit write counter for each way, the overhead is $(2 + 15)/(128 \times 8 \times 4K) \approx 0.0002\%$ compared to the data array. To avoid the fast wear-out due to the high write stress of the write counter, we can implement the write counter with SRAM cells, which normally will increase the area overhead by

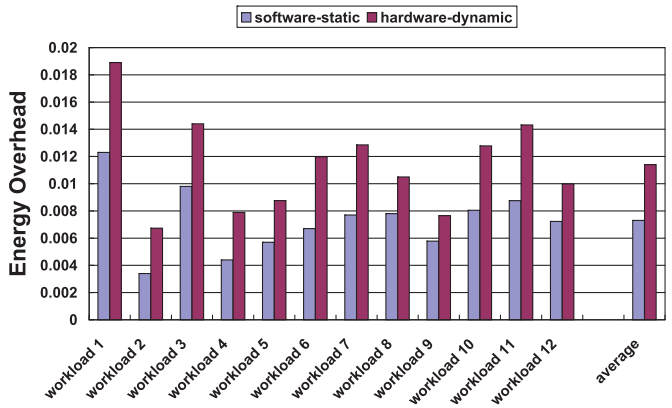


Fig. 16. The energy overheads of our software-based and hardware-based partitioning and swapping schemes.

four times if the non-volatile STT-RAM is chosen (Ahn et al. 2014; Tsai et al. 2014; Wang et al. 2014b). However, even with the four times overhead of the SRAM implementation, the total area overhead is just 0.0008%, which is also negligible. Our evaluation from NVSim simulation also shows that the hardware overhead of our hardware-based scheme is under 1%.

For the energy overheads, we evaluate our schemes by using the NVSim with STT-RAM implementation. The major energy overheads in our design are incurred by the cache flushing and swapping and the tag and counter access. However, if a proper swapping interval is chosen, the energy overhead caused by the swapping can be minimized. Figure 16 shows the energy overheads of both our software-based and hardware-based scheme, which are only 0.7% and 1.1%, separately.

5 RELATED WORK

To improve the lifetime of the non-volatile memories, there are mainly three types of solutions: (a) Write Reduction (Zhou et al. 2009; Rodriguez-Rodriguez et al. 2013; Cho and Lee 2009), which targets at reducing the amount of writes or bit flips in the memory cells; (b) Write Variation Balancing (Wear Leveling) (Qureshi et al. 2009; Joo et al. 2010; Wang et al. 2013), which tries to even out the unbalanced write frequencies to all memory lines or cachelines; and (c) Error Correction (Schechter et al. 2010; Ipek et al. 2010; Yoon et al. 2011), which uses redundancy or error correction coding schemes to extend the lifetime of the memory by fault tolerance.

Large amount of the previous work was targeting at improving the lifetime of the non-volatile main memory, not the on-chip caches (Zhou et al. 2009; Cho and Lee 2009; Khouzani et al. 2014). For the non-volatile cache lifetime improvement, Wang et al. (2013) proposed a cache set-level mitigation scheme by balancing the inter- and intra-set write variations. Mittal et al. (2014) also proposed an intra-set wear-leveling scheme for non-volatile cache. However, there are few schemes targeted at balancing the write variation for non-volatile cache on the word level and partition level. Our proposed designs first focus on the word-level write reduction and variation balancing within the cacheline by exploiting the narrow-width values, and then target the partition-level write variation balancing by adopting both hardware and software cache partitioning and swapping schemes. It can be combined with any cache set-level designs and error correction coding schemes to further improve the lifetime of the non-volatile on-chip caches.

As we discussed in Section 2, the narrow-width values have been well studied for performance, power, and reliability optimizations in high-performance microprocessors (Brooks and Martonosi

1999; Ergin et al. 2006; Kong and Chung 2012). In our work, we proposed to improve the lifetime of the non-volatile caches by exploiting the narrow-width values. The ideas of adopting the partitioning scheme to improve the reliability of the non-volatile caches were introduced and studied in Guo et al. (2012) and Lin and Chiou (2015). A scheme that adaptively switches between the SRAM and STT-RAM based on the write frequency of the applications were proposed in Jadidi et al. (2011). However, all of their studies are based on a hybrid cache architecture, which means that their cache has both traditional (SRAM) partitions and non-volatile partitions, and they tried to improve the lifetime of non-volatile partitions by moving the hot (write) data to the SRAM partitions. Our scheme is based on the pure non-volatile cache implementation and is targeted at balancing the write intensity among different non-volatile partitions.

6 CONCLUSION

In this article, we first propose to exploit the narrow-width values to improve the lifetime of the non-volatile last-level caches on the word level. We propose an LZM scheme to reduce the amount of write operations to the upper half of the narrow-width data. Then, two swapping schemes, the SW and SRepl, are proposed to mitigate the write variations between the upper half and the lower half of the narrow-width data. To further reduce the write stress to the non-volatile cache, we adopted two existing optimization schemes, the MDB and RBW. The experimental results show that our proposed word-level swapping schemes can improve the lifetime of the non-volatile caches by 40.7%, with 24.3% energy reduction, and, combined with the MDB and RBW schemes, the lifetime improvement will increase to 267%, with 58.5% energy reduction. We also propose the cache partitioning and swapping schemes to improve the lifetime on the partition level. We propose both software-based and hardware-based partitioning schemes to separate the cache partitions for write-intensive and non write-intensive applications. Then, both static and dynamic partition swapping schemes are proposed to mitigate the write variations among different cache partitions under different situations. Our experimental results show that our proposed partition-level schemes can improve the lifetime of the non-volatile caches by nearly 100% on average with small impact on the overall performance and energy consumption.

REFERENCES

J. Ahn, S. Yoo, and K. Choi. 2014. DASCA: Dead write prediction assisted STT-RAM cache architecture. In *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA'14)*. 25–36.

Nathan Binkert et al. 2011. The gem5 simulator. *ACM SIGARCH Comput. Arch. News* 39, 2 (May2011), 1–7.

D. Brooks and M. Martonosi. 1999. Dynamically exploiting narrow width operands to improve processor power and performance. In *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA'99)*.

D. Burger and T. M. Austin. 1997. *The Simplescalar Tool Set, Version 2.0.* Technical Report 1342. Computer Sciences Department, University of Wisconsin.

Sangyeun Cho and Lei Jin. 2006. Managing distributed, shared l2 caches through OS-level page allocation. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*.

Sangyeun Cho and Hyunjin Lee. 2009. Flip-n-write: A simple deterministic technique to improve pram write performance, energy and endurance. In *Proceedings of IEEE/ACM International Symposium on Microarchitecture*.

X. Dong et al. 2012. NVSim: A circuit-level performance, energy, and area model for emerging nonvolatile memory. *IEEE Trans. Comput.-Aid. Des. Integr. Circ. Syst.* 31, 7 (2012), 994–1007.

O. Ergin et al. 2006. Exploiting narrow values for soft error tolerance. *IEEE Comput. Arch. Lett.* 5, 2 (July-Dec. 2006).

Sanchuan Guo and others. 2012. Wear-resistant hybrid cache architecture with phase change memory. In *Proceedings of the IEEE Seventh International Conference on Networking, Architecture, and Storage*. 268–272.

Engin Ipek et al. 2010. Dynamically replicated memory: Building reliable systems from nanoscale resistive memories. In *Proceedings of the International Symposium on Architectural Support for Programming Languages and Operating Systems*.

Amin Jadidi, Mohammad Arjomand, and Hamid Sarbazi-Azad. 2011. High-endurance and performance-efficient design of hybrid cache architectures through adaptive line replacement. In *Proceedigns of the IEEE/ACM International Symposium on Low Power Electronics and Design*. 79–84.

Yongsoo Joo et al. 2010. Energy- and endurance-aware design of phase change memory caches. In *Proceedings of the Design, Automation and Test in Europe Conference (DATE'10)*. 136–141.

W. Kang et al. 2015. Yield and reliability improvement techniques for emerging nonvolatile STT-MRAM. *IEEE J. Emerg. Select. Top. Circ. Syst. (JETCAS'15)* 5, 1 (2015), 28–39.

H. A. Khouzani et al. 2014. Prolonging PCM lifetime through energy-efficient, segment-aware, and wear-resistant page allocation. In *Proceedings of the IEEE/ACM International Symposium on Low Power Electronics and Design*. 327–330.

Joonho Kong and Sung Woo Chung. 2012. Exploiting narrow-width values for process variationtolerant 3-D microprocessors. In *Proceedings of the 49th Design Automation Conference (DAC'12)*. 1193–1202.

Ing-Chao Lin and Jeng-Nian Chiou. 2015. High-endurance hybrid cache design in CMP architecture with cache partitioning and access-aware policies. *IEEE Trans. VLSI Syst.* 23, 10 (2015), 2149–2161.

Jiang Lin et al. 2008. Gaining insights into multicore cache partitioning: Bridging the gap between simulation and real systems. In *Proceedings of the IEEE 14th International Symposium on High Performance Computer Architecture*.

Xianlu Luo and others. 2014. Enhancing lifetime of NVM-based main memory with bit shifting and flipping. In *Proceedings of the IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'14)*. 1–7.

Sparsh Mittal, Jeffrey S. Vetter, and Dong Li. 2014. WriteSmoothing: Improving lifetime of non-volatile caches using intra-set wear-leveling. In *Proceedings of the Great Lakes Symposium on VLSI (GLSVLSI'14)*. 139–144.

M. K. Qureshi et al. 2009. Enhancing lifetime and security of PCM-based main memory with start-gap wear leveling. In *Proceedings of IEEE/ACM International Symposium on Microarchitecture*.

Moinuddin K. Qureshi and Yale N. Patt. 2006. Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*.

Roberto Rodriguez-Rodriguez et al. 2013. Reducing writes in phase-change memory environments by using efficient cache replacement policies. In *Proceedings of the Conference on Design, Automation and Test in Europe*.

Stuart Schechter et al. 2010. Use ECP, not ECC, for hard failures in resistive memories. In *Proceedings of the International Symposium on High Performance Computer Architecture*.

N. H. Seong et al. 2010. Security refresh: Prevent malicious wearout and increase durability for phase-change memory with dynamically randomized address mapping. In *Proceedings of the International Symposium on Computer Architecture*.

Li Sheng et al. 2009. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*. 469–480.

G. E. Suh et al. 2004. Dynamic partitioning of shared cache memory. *J. Supercomput.* 28, 1 (2004).

H. J. Tsai et al. 2014. Leveraging data lifetime for energy-aware last level non-volatile SRAM caches using redundant store elimination. In *Proceedings of the Design Automation Conference (DAC'14)*. 1–6.

L. Villa, M. Zhang, and K. Asanovic. 2000. Dynamic zero compression for cache energy reduction. In *Proceedings of the 33rd International Symposium on Microarchitecture (Micro'00)*. 214–220.

Jue Wang et al. 2013. i2WAP: Improving non-volatile cache lifetime by reducing inter- and intra-set write variations. In *Proceedings of the International Symposium on High Performance Computer Architecture*.

Ruisheng Wang and Lizhong Chen. 2014. Futility scaling: High-associativity cache partitioning. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*.

Shuai Wang, Jie Hu, and Sotirios G. Ziavras. 2009. On the characterization and optimization of on-chip cache reliability against soft errors. *IEEE Trans. Comput.* 58, 9 (September 2009), 1171–1184.

Yiqun Wang and others. 2014a. Register allocation for hybrid register architecture in nonvolatile processors. In *Proceedings of the IEEE International Symposium on Circuits and Systems*. 1050–1053.

Z. Wang et al. 2014b. Adaptive placement and migration policy for an STT-RAM-based hybrid cache. In *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA'14)*. 13–24.

W. Wen et al. 2013. CDECC: Content-dependent error correction codes for combating asymmetric nonvolatile memory operation errors. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD'13)*. 1–8.

Wujie Wen et al. 2014. PS3-RAM: A fast portable and scalable statistical STT-RAM reliability/energy analysis method. *IEEE Trans. Comput.-Aid. Des. Integr. Circ. Syst.* 33, 11 (2014), 1644–1656.

D. H. Yoon et al. 2011. FREE-p: Protecting non-volatile memory against both hard and soft errors. In *Proceedings of the International Symposium on High Performance Computer Architecture*.

J. Zhao, O. Mutlu, and Y. Xie. 2014. FIRM: Fair and high-performance memory control for persistent memory systems. In *Proceedings of the International Symposium on Microarchitecture (MICRO'14)*. 153–165.

P. Zhou and others. 2009. A durable and energy efficient main memory using phase change memory technology. In *Proceedings of the International Symposium on Computer Architecture*. 14–23.

Received September 2016; revised January 2017; accepted April 2017