JOURNAL OF SEMICONDUCTOR TECHNOLOGY AND SCIENCE, VOL.17, NO.4, AUGUST, 2017
https://doi.org/10.5573/JSTS.2017.17.4.514
ISSN(Print) 1598-1657
ISSN(Online) 2233-4866

Adaptive Writeback-aware Cache Management Policy for Lifetime Extension of Non-volatile Memory

Sang-Ho Hwang[1], Ju Hee Choi[2], and Jong Wook Kwak[1,*]

***Abstract*—In this paper, we propose Adaptive Writeback-aware Cache management (AWC) to prolong the lifetime of non-volatile main memory systems by reducing the number of writebacks. The last-level cache in AWC is partitioned into Least Recently Used (LRU) segment and LRU using Dirty block Precedence (DP-LRU) segment. The DP-LRU segment evicts clean blocks first for giving reuse opportunity to dirty blocks. AWC can also determine the efficient size of DP-LRU segment for reducing the number of writebacks according to memory access patterns of programs. In the performance evaluation, we showed that AWC reduced the number of writebacks up to 29% and 46%, and saved the energy of a main memory system up to 23% and 49% in a single-core and multi-core, respectively. AWC also reduced the runtime by 1.5% and 3.2% on average compared to previous cache managements for non-volatile main memory systems, in a single-core and a multi-core, respectively.**

***Index Terms*—Cache replacement policy, last-level cache, DP-LRU, adaptive writeback, non-volatile memory**

I. INTRODUCTION

Non-volatile memory (NVM) has emerged as an alternative main memory due to advantages such as low energy consumption and high density. However, two major drawbacks must be considered for NVM-based main memory systems. First, NVM is generally asymmetry in latency and an energy cost in terms of reads and writes. Writes have much longer latency and higher energy consumption than those of reads. Second, NVM has the limited lifetime. For example, the write endurance of phase-change memory (PCM), as one of representative NVMs, is about $10^8 \sim 10^9$, unlike the DRAM whose the write endurance is about 10^{15} [1]. To solve the problem of limited lifetime of PCM, wear leveling techniques such as Start-Gap [2], Security Refresh [3] and Multi-Way Wear Leveling [4] were proposed, which distributed write operations across whole PCM memory. Data-Comparison Write [5] and Flip-N-Write [6] were also proposed in order to reduce the number of write operations. However, these studies cannot prevent the frequent write operations according to the patterns of write operations.

In this paper, to solve this problem, we propose Adaptive Writeback-aware Cache management (AWC) to extend the lifetime of NVM. The Last-Level Cache (LLC) in AWC is partitioned into Least Recently Used (LRU) segment and LRU using Dirty block Precedence (DP-LRU) segment, which considers a dirty bit as well as LRU bits for selecting a victim block. The DP-LRU segment evicts clean blocks (i.e., non-dirty) first for reducing the number of writebacks by giving reuse opportunity to dirty blocks. AWC can also determine the efficient size of DP-LRU segment for reducing the number of writebacks according to memory access patterns of programs. The rest of this paper is organized as follows. Section 2 presents background knowledge

Manuscript received Jan. 25, 2017; accepted Jul. 5, 2017
[1] Department of Computer Engineering, Yeungnam University, Gyeongsan, Korea
[2] Department of Computer Science and Engineering, Seoul National University, Seoul, Korea
E-mail : kwak@yu.ac.kr

Table 1. Memory technologies of various NVMs

	Cell Size(F^2)	Endurance(Cycle)	Read Time(ns)	Write Time(ns)	Volatility
SRAM	150	-	2	2	Volatile
DRAM	10	10^{15}	20	20	Volatile
STT-MRAM	20	10^{15}	5	5-30	Non-Volatile
NAND-Flash	4	10^4	10^5	10^6	Non-Volatile
NOR-Flash	10	10^5	15	10^3	Non-Volatile
FeRAM	22	10^{12}	40	65	Non-Volatile
RRAM	30	10^5	100	100	Non-Volatile
PCM	4	10^9	12	100	Non-Volatile

and related works. Section 3 proposes cache replacement policy of AWC and cache partition mechanism. Section 4 evaluates the performance of AWC compared with others. Section 5 concludes the paper.

II. Background and Related Works

1. Background

Recently, there has been a lot of discussion about replacing main memory from DRAM to NVM [2-10]. The NVMs have higher density and less energy consumption compare to DRAMs. Table 1 compares characteristics of memory technologies in terms of density, endurance, speed of read and write operations [1, 11-13].

As representative NVMs, there are STT-RAM, FeRAM, RRAM and PCM as well as conventional NAND/NOR Flash NVMs. Among them, PCM is the most suitable for replacing DRAM because of its integration density for large capacity and its read/write operation time which is similar to that of DRAM. However, as in all NVMs, PCM also suffered from the limited number of write operations and the asymmetric speed of read and write operations. Therefore, for DRAM alternatives, the lifetime management and the handling asymmetric operations of PCM have to be considered.

Meanwhile, PCM is an NVM that stores data by using a characteristic of resistance differences between crystalline with row-resistance and amorphous with high-resistance. Fig. 1 shows a cell structure of PCM. In the PCM, storing data is done by RESET and SET operations. RESET operation is to make physical state amorphous by heating the phase change material to high temperature over 600℃. SET operation is to make physical state crystalline by heating the phase change material to lower temperature than that of RESET. Fig. 2 shows the current pulses of the SET and RESET in PCM with respect to time line.

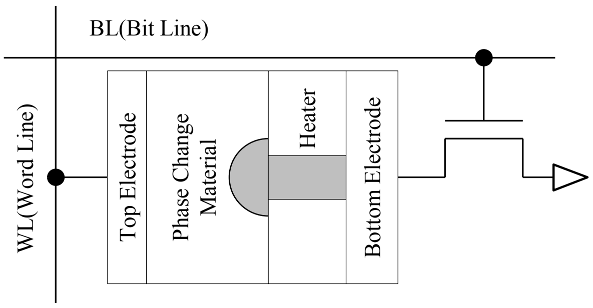


Fig. 1. Cell structure of PCM.

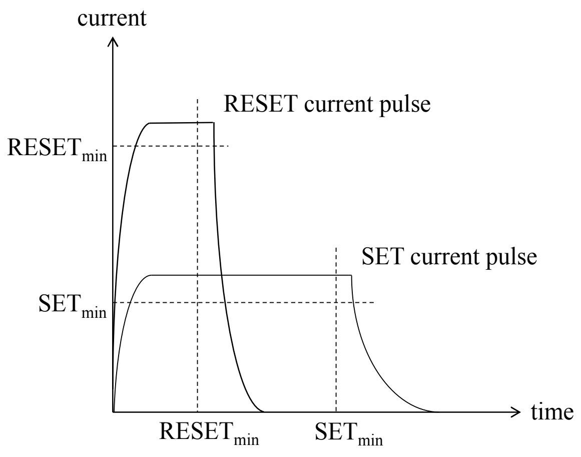


Fig. 2. Current pulses of SET and RESET in PCM.

2. Related Works

Until now, to increase the performance efficiency of cache and main memory systems, cache replacement policies have been mostly focused on conventional main memory systems using DRAM only [14]. These studies

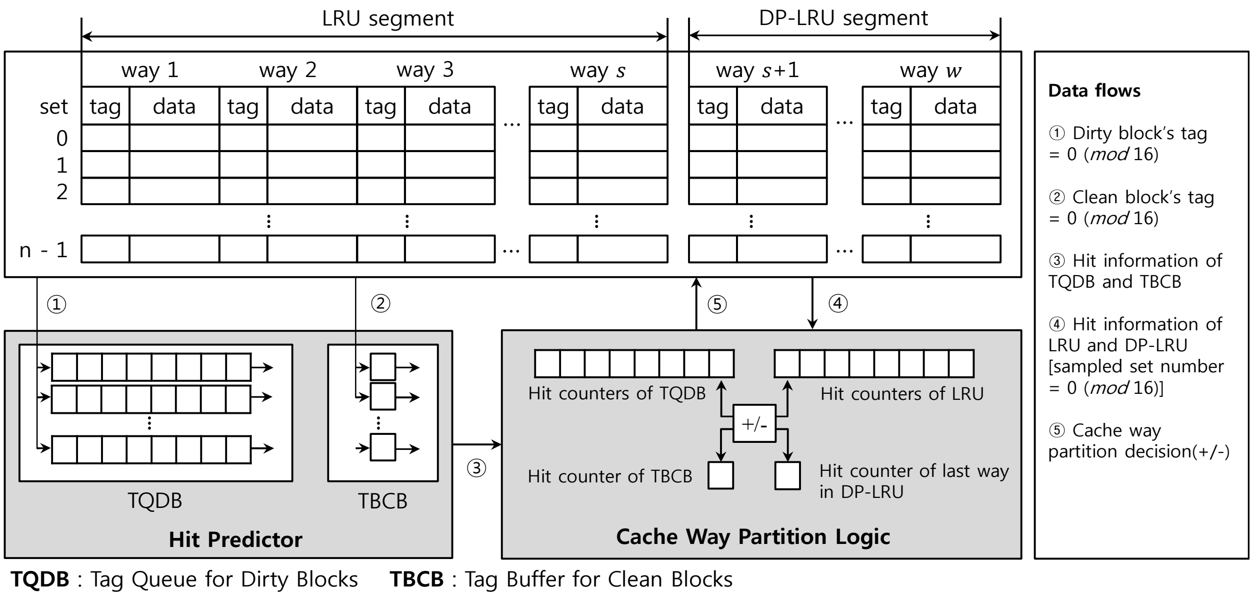

LRU segment
DP-LRU segment
way 1
way 2
way 3
way s
way s+1
way w
set
tag
data
0
1
2
n - 1
①
②
③
④
⑤
Hit counters of TQDB
Hit counters of LRU
+/-
Hit counter of TBCB
Hit counter of last way in DP-LRU
TQDB
TBCB
Hit Predictor
Cache Way Partition Logic
TQDB : Tag Queue for Dirty Blocks
TBCB : Tag Buffer for Clean Blocks
Data flows
① Dirty block's tag = 0 (mod 16)
② Clean block's tag = 0 (mod 16)
③ Hit information of TQDB and TBCB
④ Hit information of LRU and DP-LRU [sampled set number = 0 (mod 16)]
⑤ Cache way partition decision(+/-)

Fig. 3. The structure of adaptive writeback-aware cache management.

mainly exploited the patterns of write operations to increase cache hit ratio. However, cache replacement policies for non-volatile main memory systems have been increasingly studied recently. NVM-based main memory systems must manage write operations for increasing the performance and prolonging the lifetime of NVM systems. Meanwhile, writebacks of LLC result in a number of write operations to NVM-based main memory systems. Therefore, cache management schemes have been proposed for reducing the number of writebacks to NVM and the following shows representative techniques of managing cache blocks for NVM-based main memory systems.

N-Chance can extend the lifetime of NVM by early eviction of non-dirty blocks. N-Chance selects the oldest clean block as a victim block among N blocks from LRU-side [7]. If there is no clean block among N blocks from LRU-side, a dirty block placed in LRU is selected as a victim block for replacement. While N-Chance reduced the number of writebacks, it is vulnerable to pattern changes in programs by using a static parameter.

Writeback-Aware Dynamic CachE (WADE) keeps dirty blocks in LLC by using two lists which are frequent and non-frequent writeback lists, respectively, for reusing dirty blocks [9]. WADE predicts write-intensive blocks which accordingly result in frequent writebacks into memory. These blocks are managed by the two lists, and WADE tries to keep the best size of each list to obtain high hit ratio of dirty blocks.

Adaptive and Combined Wear-out-Aware Replacement (AC-WAR) finds dirty blocks to stay in the cache by tracking sub-block modification [10]. AC-WAR can reduce the number of bit-flips of NVM by selecting the least modified block as a victim block in Least Modified block First (LMF) policy. However, AC-WAR had the capacity overhead for tracking sub-block modification in whole blocks.

III. Adaptive Writeback-aware Cache Management

1. Cache Replacement Policy for AWC

The proposed cache management policy, called AWC, helps to improve the lifetime extension of NVM-based main memory systems by reducing the number of writebacks. Fig. 3 shows the structure of the proposed AWC in w-way set-associative LLC. The LLC of proposed scheme is partitioned into two segments, which use LRU and DP-LRU policy and are called as LRU segment and DP-LRU segment, respectively. AWC can reduce the number of writebacks, which results in the extension of the lifetime of NVM, by the early eviction of clean blocks placed in way positions where have low hit ratio of clean blocks. On the other hand, dirty blocks are moved to DP-LRU segment. The DP-LRU segment

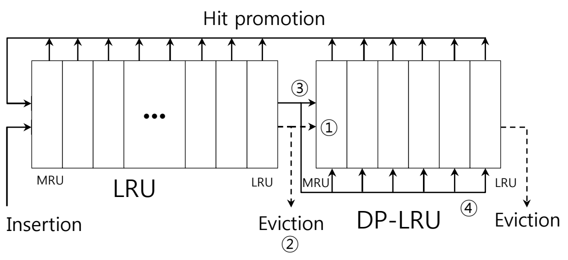
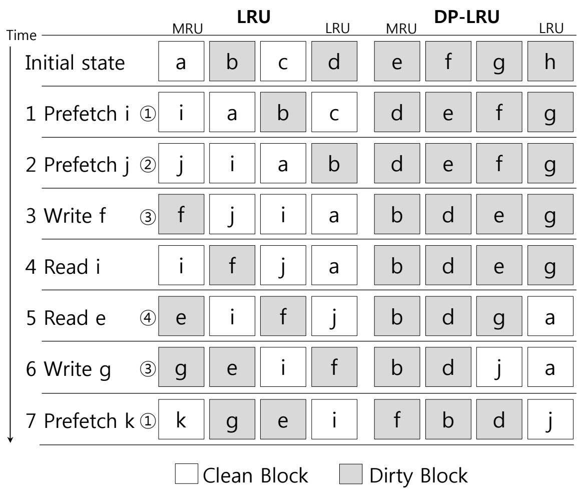


Fig. 4. Segmentation of last-level cache.



Time	MRU	LRU		LRU	MRU	DP-LRU		LRU
Initial state	a	▨ b	c	▨ d	▨ e	▨ f	▨ g	▨ h
1 Prefetch i ①	i	a	▨ b	c	▨ d	▨ e	▨ f	▨ g
2 Prefetch j ②	j	i	a	▨ b	▨ d	▨ e	▨ f	▨ g
3 Write f ③	▨ f	j	i	a	▨ b	▨ d	▨ e	▨ g
4 Read i	i	▨ f	j	a	▨ b	▨ d	▨ e	▨ g
5 Read e ④	▨ e	i	▨ f	j	▨ b	▨ d	▨ g	a
6 Write g ③	▨ g	▨ e	i	▨ f	▨ b	▨ d	j	a
7 Prefetch k ①	k	▨ g	▨ e	i	▨ f	▨ b	▨ d	j

☐ Clean Block ▨ Dirty Block

Fig. 5. The example scenario of AWC.

gives dirty blocks the precedence for reusing these blocks.

Fig. 4 shows the segmentation of the LLC for a given set. When a new block is inserted, the block placed in LRU position in the LRU segment is evicted. If the evicted block is dirty, the block is moved to DP-LRU segment (Case ①). Otherwise, a clean block is evicted directly with no migration (Case ②). When a block is hit in the LLC, the block is moved to MRU position in the LRU segment, and then other blocks are shifted to LRU position. Finally, the block of LRU position is moved to DP-LRU segment, and the positions of whole blocks in DP-LRU segment are adjusted according to whether dirty (Case ③) or not (Case ④)

Fig. 5 shows the example of AWC assuming a mixed access pattern. When a new block *i* is inserted, a dirty block *d* is moved to DP-LRU segment. In second step, a clean block *c* placed in LRU position is evicted. When a block is hit in LLC, the block placed in LRU is moved to DP-LRU segment without considering whether dirty or not. In step 3, 5 and 6, the block placed in LRU position is moved to DP-LRU segment. However, the position of whole blocks in DP-LRU segment can be adjusted according to DP-LRU policy. For example, the positions of the block *a* and the block *j*, which are clean in step 5 and step 6 respectively, are adjusted by DP-LRU policy. This position adjustment enables that dirty blocks can stay longer than clean blocks in LLC.

2. Cache Way Partition for AWC

AWC determines the efficient size of DP-LRU segment. To find optimal cache partitioning size between LRU and DP-LRU, AWC exploits the hit predictor and the cache way partition logic, which are presented as a gray color in Fig. 3, and AWC observes the hit information for each way during a given period of time. The hit predictor stores the tags of both dirty and clean blocks evicted from the LLC and observes whether the tag is hit or not. AWC increases or decreases the size of DP-LRU segment based on this observation results. To observe the hit of tags obtained in the evicted blocks, the hit predictor has Tag Queue for Dirty Blocks (TQDB) and Tag Buffer for Clean Blocks (TBCB). TQDB inserts tags of dirty blocks, which are evicted from LLC, to observe the reuse of dirty blocks by assuming additional ways for these blocks. TBCB inserts tags of clean blocks, which are evicted from LLC, to observe the reuse of clean blocks by assuming additional ways for these blocks. That means TBCB monitors the efficiency of DP-LRU segment. Note that this observation uses sampled sets, e.g., our evaluation uses the sampled 128 sets among 2048 sets.

Increasing the size of DP-LRU segment is calculated as follows:

$$w = \{x \mid \sum_{n=1}^{x}\sum_{g=1}^{G} TQDB_g^n - \sum_{n=1}^{x}\sum_{g=1}^{G} LRU_g^{s-n-1} \geq 0\} \qquad (1)$$

$$DPLRU_{size}^{p} = DPLRU_{size}^{p-1} + \max(w) \qquad (2)$$

where $TQDB^n_g$ is the hit counts for nth way of gth sampled set in TQDB. LRU^g_n is hit counts of clean blocks for nth way of gth sampled set in LRU segment. *G* is the total number of the sampled sets and *s* is the size of LRU segment. $DPLRU^p_{size}$ is size of DP-LRU at pth. Eq. (1) calculates the gain of the hit when additional ways are allowed for dirty blocks. The max(*W*), which satisfies the Eq. (1), is selected as the number of additional ways

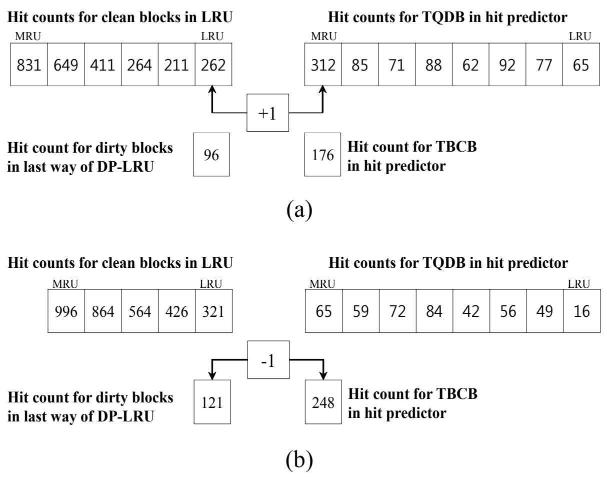

Hit counts for clean blocks in LRU
MRU
LRU
831 649 411 264 211 262
Hit counts for TQDB in hit predictor
MRU
LRU
312 85 71 88 62 92 77 65
+1
Hit count for dirty blocks in last way of DP-LRU
96
176
Hit count for TBCB in hit predictor
(a)
Hit counts for clean blocks in LRU
MRU
LRU
996 864 564 426 321
Hit counts for TQDB in hit predictor
MRU
LRU
65 59 72 84 42 56 49 16
-1
Hit count for dirty blocks in last way of DP-LRU
121
248
Hit count for TBCB in hit predictor
(b)

Fig. 6. The example of cache way partition logic (a) Increasing the number of DP-LRU segment, (b) Decreasing the number of DP-LRU segment.

for the updated size of DP-LRU segment (Eq. (2)). Fig. 6(a) shows the example of increasing the size of DP-LRU segment. After reaching a given period of time for monitoring the hit prediction, AWC first proceeds an increasing process of the number of DP-LRU segments. When the increased number of hits of dirty blocks due to the addition of the DP-LRU segment is greater than the reduced number of hits of clean blocks because of the reduction of LRU segment, AWC increases the size of DP-LRU. As shown in Fig. 6(a), the number of dirty block hits in the first way of TQDB which monitors the hit counts under the assumption of additional ways for DP-LRU is greater than that of clean block hits in the last way of LRU segment. However, the sum up to the second way in TQDB (312+85) is less than the sum from the last way to next of LRU segment (211+262). Therefore, AWC increase the size of DP-LRU segment by one way.

Decreasing the size of DP-LRU segment is determined as follows:

$$\sum_{g=1}^{G} TBCB_g - \sum_{g=1}^{G} DPLRU_g \geq 0 \tag{3}$$

$$DPLRU_{size}^{p} = DPLRU_{size}^{p-1} - 1 \tag{4}$$

where $TBCB_g$ is the hit counts for gth sampled set in TBCB and $DPLRU_g$ is hit count of dirty blocks for gth sampled set in the last way of DP-LRU segment. Note that we monitor the last way in each set to prevent decreasing the ways for dirty blocks too quickly. When the hit ratio of the last way in DP-LRU segment is smaller than that of TBCB (Eq. (3)), the size of DP-LRU segment is decreased (Eq. (4)). Fig. 6(b) shows the example of decreasing the size of DP-LRU segment. When the increase condition of DP-LRU is not qualified, AWC proceeds with the reduction procedure. This procedure checks the hit efficiency of the last way in DP-LRU. In Fig. 6(b), the hit count for dirty blocks in the last way of DP-LRU (121) is less than the hit count for TBCB (248). This means that the overall hit rate drops due to DP-LRU, so AWC reduces the size of DP-LRU by one way.

Table 2. Details of experiment environment.

CPU	3 GHz, 1-core/4-core
L1 instruction	32 KB, 4-way, 64B line, 2 cycle latency, private
L1 data	32 KB, 4-way, 64B line, 2 cycle latency, private
L2	2 MB/1-core, 8MB/2-core, 16-way, 64B line, 20 cycle latency, shared
Memory (PCM)	4 GB Array read energy = 2.47 pJ/bit Array write energy = 16.82 pJ/bit Row buffer read energy = 0.93 pJ/bit Row buffer write energy = 1.02 pJ/bit

Table 3. Benchmark programs.

1-core	light	astar, h264ref, omnetpp, provray, tonto
	middle	calculix, gobmk, namd, sjeng, sphinx3
	high	bwaves, bzip2, lbm, mcf, soplex
4-core	Mix1	bzip2,soplex,bwaves,calculix
	Mix2	astar,povray,hmmer,soplex
	Mix3	bzip2,soplex,hmmer,h264ref
	Mix4	povray,libquantum,calculix,sphinx3
	Mix5	soplex,astar,bzip2,bzip2
	Mix6	libquantum,mcf,hmmer,povray
	Mix7	mcf, sjeng, soplex, povray
	Mix8	omnetpp,bwaves,mcf,sphinx3
	Mix9	omnetpp,bzip2,mcf,soplex
	Mix10	mcf,mcf,astar,bzip2

IV. Performance Evaluation

We used GEM5 simulator [15] to evaluate the performance of AWC. Table 2 shows the experiment parameters used in GEM5 simulator. We use energy parameters of PCM as from [16]. We used SPEC CPU 2006 benchmarks [17] to evaluate our proposal. In case of single-core performance evaluation, we divided benchmarks into three groups according to the ratio of

Algorithm 1: Cache way partitioning

```
   Input : $partitionPos_{present}$(Size of $LRU$), $TQDB$, $TBCB$, $DPLRU$
   Output : $PartitionPos_{present}$
1  $Pos_{LRU} \leftarrow PartitionPos_{present}$
2  $Pos_{TQDB} \leftarrow 0$
3  $PartitionPos_{prvios} \leftarrow PartitionPos_{present}$
4  while $Pos_{LRU} > 0$ do
5      $Gain$ += $TQDB.HitCount[Pos_{TQDB}]$
6      $Loss$ += $LRU.CleanBlkHitCount[Pos_{LRU} - 1]$
7      if $Gain \geq Loss$ then
8          $PartitionPos_{prvios} \leftarrow Pos_{LRU}$
9      end if
10     $Pos_{LRU} \leftarrow Pos_{LRU} - 1$
11     $Pos_{TQDB} \leftarrow Pos_{TQDB} + 1$
12 end while
13 if $PartitionPos_{prvios} = PartitionPos_{present}$ then
14     if DPLRU.LastWayHitCount
           $\leq$ TBCB.HitCount then
15         $PartitionPos_{present} \leftarrow PartitionPos_{present} + 1$
16     end if
17 end if
```

writebacks. To evaluate the performance of multi-core environment, we used 10 application-mixes. Table 3 shows the used benchmarks in this paper. We evaluated the performance of AWC compared with baseline (LRU), N-Chance (N=8) and AC-WAR (d=1, 3, 5) in terms of the number of writebacks, energy consumption and runtime. For accuracy in the evaluation, we ran one billion instructions after fast forwarding of one billion instructions. The monitoring period of AWC is one hundred thousand cycles.

Algorithm 1 shows the overall of the cache way partition algorithm which is about increasing or decreasing the size of DP-LRU segment mentioned above. First, AWC compares the hit counts of the tags of TQDB with the hit counts of clean blocks in LRU segment (line 4-12). If there is a section having higher hit counts of TQDB than those of clean blocks in LRU segment, AWC increases the size of DP-LRU segment by those amounts (line 7-8). When there is no section to increase the size of DP-LRU segment, AWC checks the efficiency of the current DP-LRU segment (line 13-17). If the hit count of the last way in DP-LRU segment is less than that of TBCB, AWC reduces the size of the DP-LRU segment by one to resolve the problem in which the overall hit rate drops (line 14-15).

1. Evaluation for Single-core System

Fig. 7 shows the normalized number of writebacks. The number of writebacks is closely related to the lifetime of NVM-based main memory systems. AC-WAR reduces the number of writebacks by using LMF, which conducts an eviction of the least modified block first. However, in the benchmarks which have the light ratio of writebacks, AC-WAR has a low efficiency of LMF segment because of the lack of correlation between the writebacks and sub-blocks changed. However, AWC can reduce the number of writebacks due to the high reuse ratio of the dirty blocks. AWC reduced the number of writebacks from 4.2% to 29% compared to others.

Fig. 8 shows the normalized energy consumption. The energy consumption is directly influenced by the number of main memory accesses. In particular, write operations need more energy than read operations. Therefore reducing the number of writebacks also positively affects the energy consumption as well as the lifetime of NVM-based main memory systems. By reducing the number of writebacks effectively, AWC reduced the energy consumption from 6% to 23%, due to the high reuse ratio of the dirty blocks compared to others.

Fig. 9 shows the normalized runtime. By providing higher hit ratio of the dirty blocks, the number of writebacks was reduced in AWC. However, there is a possibility that the hit ratio of clean blocks is decreased on the contrary. In this case, the overall runtime of the entire system may differ depending on applications. Typically, the runtime may increase in read-intensive applications (light), while it may decrease in write-intensive applications (high) which can reduce the number of writebacks. In AC-WAR, it had difference runtime depending on the value of d, because the small value of d resulted in a deteriorated runtime by the degradation of hit ratio. However, our proposal can reduce the runtime or, at least, keep it similar to that of LRU, due to determine the efficient cache partition size depending on the writeback pattern of benchmarks. AWC reduced the runtime by 1.5% on average compared to others.

2. Evaluation for Multi-core System

In this section, we simulate AWC in a multi-core

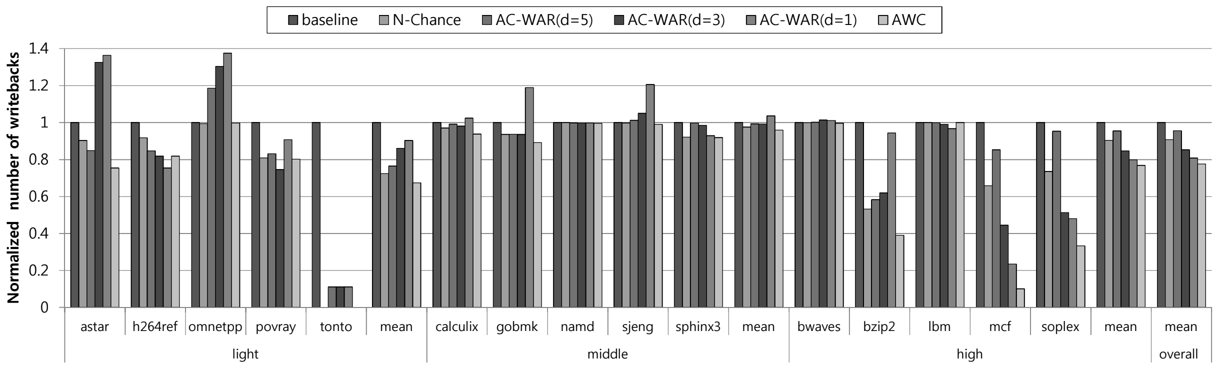


Fig. 7. Normalized number of writebacks in single-core application.

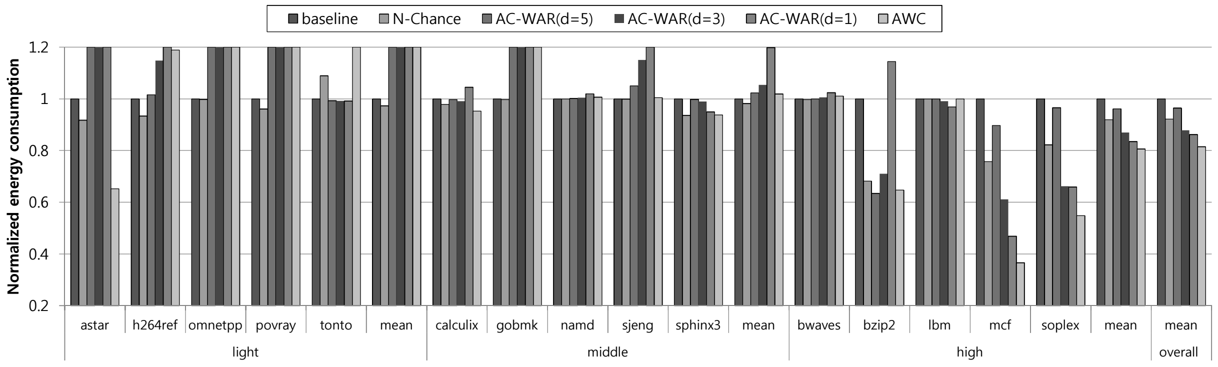


Fig. 8. Normalized energy consumption in single-core application.

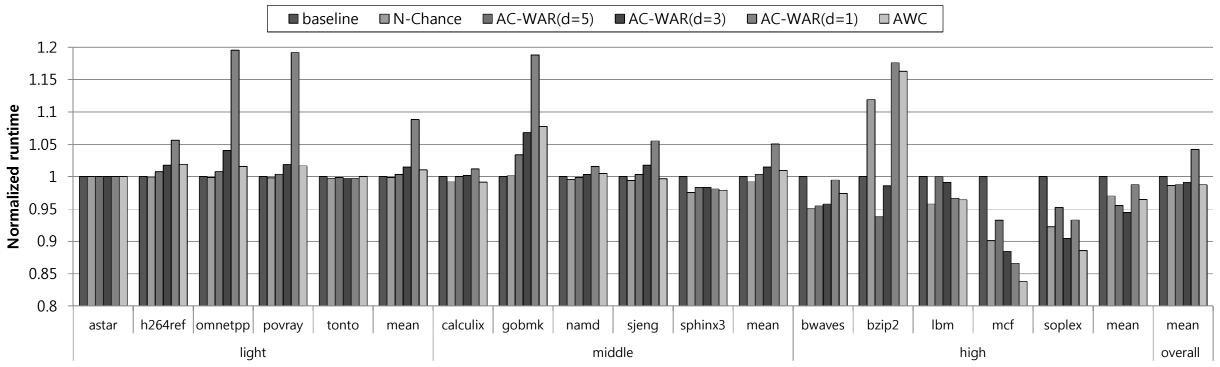


Fig. 9. Normalized runtime in single-core application.

configuration which has a 4-core with an 8MB 16-way associative LLC. Fig. 10 shows the normalized number of writebacks in multi-core applications. In multi-core configuration, the hit ratio of dirty blocks in LLC differs for each application. AC-WAR provided a low performance in terms of writeback reduction in some applications, such as bwaves and sjeng, which show a low correlation between the writebacks and sub-blocks changed. However, AWC can reduce the number of writebacks due to the adaptive determination of the efficient size of DP-LRU segment, in most applications. AWC reduced the number of writebacks from 13.5% to 46% compared to others.

Fig. 11 shows the normalized energy consumption in multicore applications. As the results in single-core applications, AWC can save energy of main memory by reducing the number of write operations which need more energy than that of read operations. As shown in

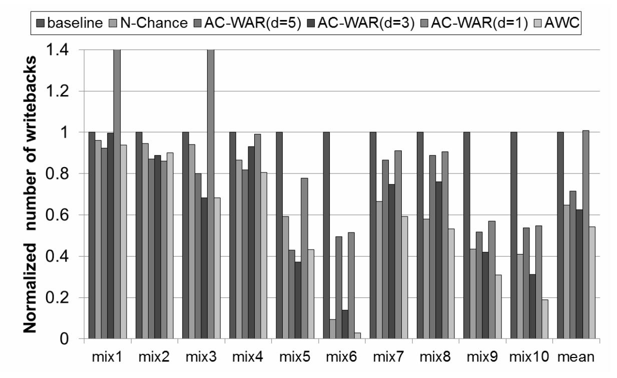


Fig. 10. Normalized number of writebacks in multi-core application.

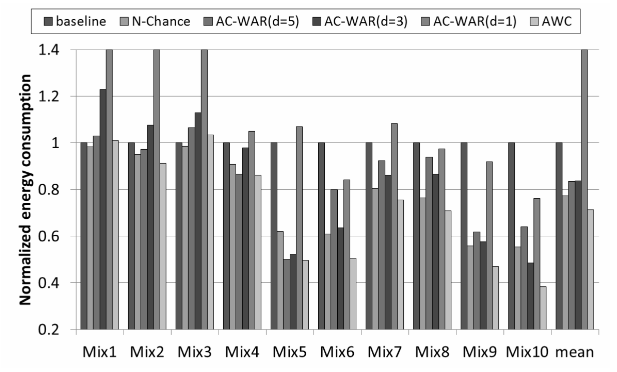


Fig. 11. Normalized energy consumption in multi-core application.

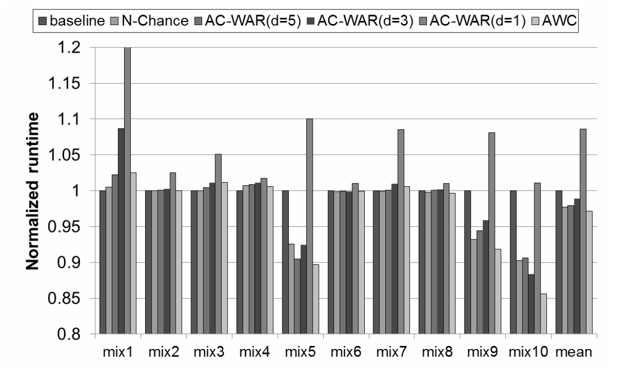


Fig. 12. Normalized runtime in multi-core application.

Fig. 11. AWC can reduce the energy consumption of PCM-based main memory systems from 8% to 49% compared to others.

Fig. 12 shows the normalized runtime in multi-core applications. As mentioned earlier, the runtime is related to the hit ratio of both dirty blocks and clean blocks. As the result in the single-core applications, AWC can reduce the runtime or keep it similar to that of LRU in the multi-core application environment as well. AWC achieves the reduction of the runtime by 3.2% on average compared with other policies.

V. CONCLUSIONS

In this paper, we proposed AWC for NVM-based main memory systems. The LLC in AWC is partitioned into LRU segment and DP-LRU segment. The DP-LRU segment evicts clean blocks first for giving reuse opportunity to dirty blocks. AWC can reduce the number of writebacks by the adaptive cache partition according to the memory access patterns of programs. In experimental part, we evaluated the performance of AWC and showed that AWC reduced the number of writebacks up to 29% and 46%, and saved the energy consumption of the main memory system up to 23% and 49% in a single-core and multi-core, respectively. AWC also reduced the runtime by 1.5% and 3.2% in the single-core and the multi-core, respectively.

ACKNOWLEDGMENTS

This work was supported by the 2016 Yeungnam University Research Grant.

REFERENCES

[1] Mittal, Sparsh, and Jeffrey S. Vetter. "A survey of software techniques for using non-volatile memories for storage and main memory systems." IEEE Transactions on Parallel and Distributed Systems, Vol.27, No.5, pp. 1537-1550, 2016.

[2] Qureshi, Moinuddin K., et al. "Enhancing lifetime and security of PCM-based main memory with start-gap wear leveling." Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture, pp. 14-23, 2009.

[3] Seong, Nak Hee, 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, Vol. 38, No. 3, pp. 383-394, June 2010.

[4] Yu, Hongliang, and Yuyang Du. "Increasing Endurance and Security of Phase-Change Memory with Multi-Way Wear-Leveling." IEEE Transactions on Computers, Vol. 63, No. 5, pp. 1157-1168, May 2014.

[5] Yang, Byung-Do, et al. "A low power phase-change random access memory using a data-comparison write scheme." 2007 IEEE International Symposium on Circuits and Systems, pp. 3014-3017, May 2007.

[6] Cho, Sangyeun, and Hyunjin Lee. "Flip-N-Write: a simple deterministic technique to improve PRAM write performance, energy and endurance." 2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), pp. 347-357, Dec 2009.

[7] Ferreira, Alexandre P., et al. "Increasing PCM main memory lifetime." Proceedings of the conference on design, automation and test in Europe. European Design and Automation Association, pp. 914-919, 2010.

[8] Zhou, Miao, et al. "Writeback-aware partitioning and replacement for last-level caches in phase change main memory systems." ACM Transactions on Architecture and Code Optimization (TACO), Vol. 8, No. 4, Article No. 53, January 2012.

[9] Wang, Zhe, et al. "WADE: Writeback-aware dynamic cache management for NVM-based main memory system." ACM Transactions on Architecture and Code Optimization (TACO), Vol. 10, No. 4, Article No. 51, December 2013.

[10] Abad, Pablo, et al. "AC-WAR: Architecting the Cache Hierarchy to Improve the Lifetime of a Non-Volatile Endurance-Limited Main Memory." IEEE Transactions on Parallel and Distributed Systems, Vol. 27, No. 1, pp.66-77, January 2016.

[11] Xia, Fei, et al. "A survey of phase change memory systems." Journal of Computer Science and Technology, Vol. 30, No. 1, pp. 121-144, January 2015.

[12] Torres, Lionel, et al. "Trends on the application of emerging nonvolatile memory to processors and programmable devices." 2013 IEEE International Symposium on Circuits and Systems (ISCAS2013), pp. 101-104, May 2013.

[13] Mittal, Sparsh, Jeffrey S. Vetter, and Dong Li. "A survey of architectural approaches for managing embedded DRAM and non-volatile on-chip caches." IEEE Transactions on Parallel and Distributed Systems, Vol. 26, No. 6, pp. 1524-1537, June 2015.

[14] Jaleel, Aamer, et al. "High performance cache replacement using re-reference interval prediction (RRIP)." ACM SIGARCH Computer Architecture News, Vol. 38. No. 3, pp. 60-71, June 2010.

[15] Binkert, Nathan, et al. "The gem5 simulator." ACM SIGARCH Computer Architecture News, Vol.39, No.2, pp.1-7, 2011.

[16] Lee, Benjamin C., et al. "Architecting phase change memory as a scalable dram alternative." ACM SIGARCH Computer Architecture News, Vol. 37. No. 3,pp.2-13, 2009.

[17] J. L. Henning, "Spec cpu2006 benchmark descriptions," ACM SIGARCH Computer Architecture News, Vol.34, No.4, pp.1–17, 2006.

Sang-Ho Hwang received the B.S., M.S. and Ph.D. degrees in Computer Engineering from Yeungnam University, Gyeongsan, South Korea, in 2009, 2013 and 2017 respectively. His current research interests include embedded systems and non-volatile memory systems.

Ju Hee Choi received the B.S. degree in computer science from Yonsei University, Seoul, South Korea, in 2004, and the M.S. and Ph.D. degrees in computer science and engineering from Seoul National University, Seoul, in 2006 and 2016, respectively. Since 2006, he has been with the S.LSI, Samsung Electronics Company, Ltd., Suwon, South Korea. His current research interests include system-on-chip design methodology, embedded system design, and low-power design.

Jong Wook Kwak received the B.S. degree in computer engineering from Kyungpook National University, Daegu, South Korea, in 1998, and the M.S. degree in computer engineering and the Ph.D. degree in electrical engineering and computer science from Seoul National University, Seoul, South Korea, in 2001 and 2006, respectively. From 2006 to 2007, he was a Senior Engineer with the system-on-chip (SoC) Research and Development Center, Samsung Electronics Company, Ltd., Suwon, South Korea. During 2012–2013, he was a Visiting Scholar at the Georgia Institute of Technology, Atlanta, GA, USA. He is currently an Associate Professor with the Department of Computer Engineering, Yeungnam University, Gyeongsan, South Korea. His current research interests include SoC design, advanced processor architecture, low-power mobile embedded system design, and high-performance parallel and distributed computing.