ELectronics EXpress

IEICE Electronics Express, Vol.20, No.6, 1–6

LETTER

A method of defense against cache timing attack in non-volatile memory

Juhee Choi[1, a)]

Abstract Attackers of modern computer architecture found that cache access latency difference between cache hit and cache miss is a point where secure data are overlooked. To prevent such data leakage, cache partitioning technique is utilized for defenders via cache hit isolation. Although this approach is effective in increasing resistance against cache timing attack, it is not suitable for emerging memory system, which is based on non-volatile memories, because it overlooks the weaknesses of the write operations. This paper proposes a secure-aware partitioning guide architecture to improve performance and write endurance by removing the necessity of cache flushing. During changing cache partitioning status, the write counts are considered for the new status and no cache lines are evicted in the proposal. As a result, the lifetime is extended by 1.77 times and the penalty of cache flushing is saved by 7.8%.

Keywords: non-volatile memory, STT-RAM, security, side channel attack
Classification: Integrated circuits

1. Introduction

Cache side-channel attack is a well-known course through which secure data leak out in modern processor techniques, such as cache and speculative access [1, 2]. Typically, a cache can mitigate the performance gap between the main memory and processors, so-called memory wall problem. Furthermore, out-of-order execution and speculative access change the order of instruction execution and data access to speed up the performance unless the final results of the program are corrupted. These schemes show dramatic improvement in performance, resulting in essential parts of the modern computing systems.

However, they also bring a critical side effect, namely providing vital hints to attackers. The latency difference between cache hit and cache miss is inevitable, and it is one of the sources which makes the security of the system weak. Out-of-order execution and speculative access imply that the secure data are touched without permission check if the attacker generates a wrong prediction. The latency gap between cache hit and miss becomes a tool to check whether the target data are stored in the cache. Meltdown [3] and Spectre [4], including its variants [5], are some of the most famous attack methods by exploiting these weaknesses.

One of the promising solutions to the problem is cache partitioning [6, 7]. Traditionally, this field is considered a performance related study topic. Some research groups concentrated on the quality of service (QoS) of the specific program [8], while others aimed to improve utilization of the cache [9]. On the contrary, Kiriansky et al. reported that cache partitioning helps data isolation among ways and developed this approach as dynamically allocated way guard (DAWG) [6].

Although Kiriansky's work achieved defense against attackers, it is not perfectly suitable for the emerging memory system based on non-volatile memory (NVM). One of the important considerations in designing an NVM-based cache architecture is reducing the write counts. This is because updating the information of NVM cells introduces several problems, such as limited write endurance, high energy consumption, and long latency for write operations. In this context, the present paper proposes a secure-aware partitioning guide architecture (SaPGA). Since there is no strict cache partitioning restriction for tag matching process, no cache flush is needed during partitioning change in the SapGA. Instead, each cache line has its own secure flag to check whether it is a secure-aware cache line. In addition, to manage the secure-aware partitioning guide considering the lifetime of the cache, write counter array and manager are also inserted.

2. Background and motivation

2.1 Background

With advancement of fabrication technology, leakage energy consumption has become one of the major concerns to the manufactures due to low operating voltage. As a complementary product, researchers have focused on NVMs [10, 11, 12, 13] because of their significantly low amount of static power dissipation. Since a bit information is indicated as a state of the material of NVM instead of electric current, no or a few electric sources are required to sustain the information. The names of each type of NVM usually come after its type of memory cell, such as phase change memory (PCM) and spin-torque transfer magnetic RAM (STT-MRAM).

For example, the PCM [10, 14] is composed of alloy material such as Ge2Sb2Te5 and AgInSbTe. Its phase is determined from electrical resistivity and optical reflectivity. There are two phases such as an amorphous phase and a crystalline phase to store "0" value or "1" value in a bit.

Upon employing NVMs, the disadvantages of write operations are found to accompany. Updating the states of cells consumes a significant amount of energy and takes long latency compared with the traditional volatile memories. In addition, the capacity of the number of write operations is

[1] Dept. of Smart Information Communication Engineering, Sangmyung University, 31 Sangmyungdae-gil, Dongnam-gu, Cheonan 31066, Korea
[a)] jhplus@smu.ac.kr

DOI: 10.1587/elex.20.20220477
Received October 25, 2022
Accepted January 30, 2023
Publicized February 6, 2023
Copyedited March 25, 2023

Copyright © 2023 The Institute of Electronics, Information and Communication Engineers

small due to poor write endurance, resulting in limited lifetime problem. About 10^7–10^8 times of writing to the PCM cell is allowed, while SRAM endures more than 10^{15} writing times. One of approaches to overcome the write problems of NVM is to avoid redundant bit-writing by observing the new value and the old value in NVM cells [15, 16, 17]. Plus, hybrid cache architecture (HCA) was introduced [18] to reduce the write-intensity of NVM by containing SRAM cells as well as NVM cells in the same structure. It can be applied from L1 cache level [19, 20] to last-level cache [21] or main memory [22, 23, 24]. In addition, FPGA or IoT devices are also objectives of HCA studies [25, 26, 27].

2.2 Motivation

The cache partitioning schemes have commonly assumed that the domain policy is dynamically adjusted to maximize the utilization of all ways [8, 9, 28, 29]. Since the access pattern to the cache varies across the working set of the program, fixing the usage of each way during execution cannot be done efficiently. The methods to estimate the optimal partitioning policy differ according to their prime interests. Some research groups studied for fairness [28] or QoS [8], while others concentrated on energy saving [9, 29].

However, there is a fundamental difference between the secure-aware partitioning and previous partitioning studies in terms of cache hit isolation. A generic cache system ignores the current partitioning status during tag matching.

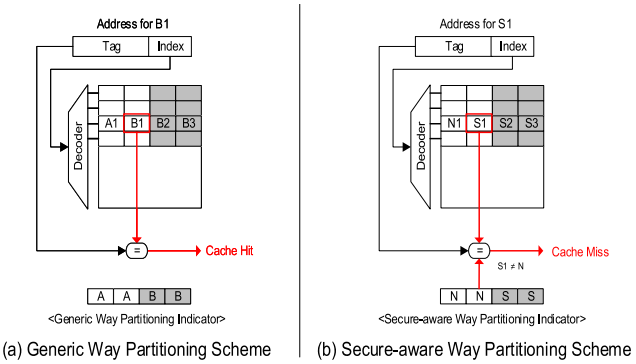

Fig. 1 Motivation. A1 is a cache line of Group A, and B1, B2, and B3 are cache lines of Group B. N1 is a cache line of Group N, while S1, S2, and S3 are cache lines of Group S.

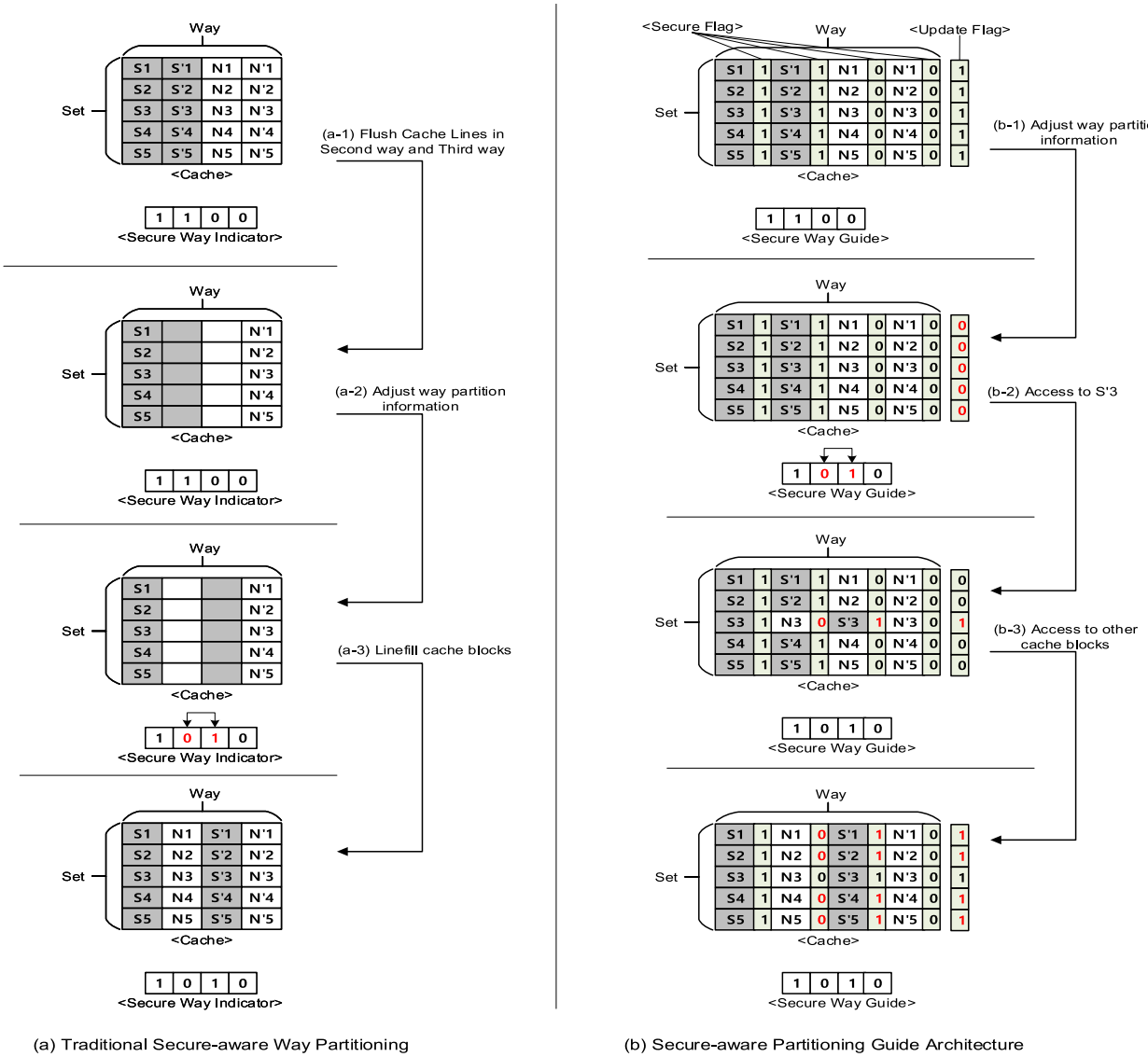

Fig. 2 Operations.

For example, let us assume that A and B are two partition groups in Fig. 1(a). Although B1 is dedicated to the group B and it is located in the second way, which is assigned to the group A, access to the cache line ends up with success.

On the other hand, a cache hit is prohibited for S1 in Fig. 1(b) for the mismatched group in the secure-aware partitioning schemes to hide any latency difference. In this regard, the concept of secure way indicator (SWI) was introduced to store the characteristics of each way, i.e. secure way or non-secure way. The traditional tag matching process was modified for the SWI to determine the cache hit. The detailed implementation can vary with schemes; "set_metadata" and "policymask" are the variant of the SWI in DAWG.

The previous approaches are cost-efficient for a fixed partitioning scheme; however, they are not suitable when the partitions are adjusted dynamically. If a way in group A is changed to group B, all the cache lines in the way are flushed in advance before switching. It can be easily expected that frequent cache flushing generates performance hurt. For an NVM-based memory system, another point to consider is the penalty of extra write counts caused by cache invalidation. Therefore, to realize performance improvement and lifetime extension, only applying the traditional schemes to secure-aware partitioning does not solve the problems, and a novel scheme is required to this end considering the characteristics of NVM.

3. Secure-aware partitioning guide architecture

3.1 Operations

Fig. 2 demonstrates the process of changing the cache partitioning status of the traditional scheme and SaPGA in detail. The key idea of the proposed scheme is that cache flushing is not necessary to adjust the cache partitioning status, while the previous scheme required cleaning up the cache lines of the ways which are supposed to be switched.

The initial status of cache partitioning in Fig. 2 is that the first way and the second way are dedicated to secure ways, while the third way and the fourth way are marked as the non-secure ways. The cache partitioning policy manager tries to change the second way and the third way by switching their secure flag bits. S and S' cache lines are for the secure group, while N and N' cache lines are for the non-secure group.

In the conventional scheme, all cache lines in the second way and the third way are evicted before modifying the information of the SWI (a-1). Then, the target bits are changed (a-2). After consecutive cache misses for all cache lines, the cache lines are refilled (a-3).

However, in SaPGA, the cache partitioning status is simply changed by modifying a secure way guide (SWG) register (b-1). The SWG does not participate in cache hit process unlike the SWI, but it only provides a blueprint for efficient cache partitioning status; only the SWG is activated during Victim selection process. Instead, each cache line has its own secure flag (SF) to confirm the guaranty of its security.

All SFs in a cache set are simultaneously updated when any cache line in the set is accessed. If the values of the SFs differ from the current SWG, the cache lines are exchanged or invalidated according to the SWG (b-2). As the application executes, all the cache sets are accessed and cache partitioning is completed without any extra cycle of unnecessary cache flushing (b-3). Fig. 3 shows the detailed processes of the SaPGA in terms of read operation and cache partitioning updating.

Algorithm : Dynamic Secure-cache partitioning

```
* Parameters
N: Number of ways
S: Number of sets
Address: Address for cache access
TagAddr: Tag for cache line from Address
Way[i]: ith way
SF: Secure flag
UF: Update flag
```

```
* Read Operation
1 : TagAddr ← extract tag from address(Address)
2 : for i ← 1 to N:
3 :   if TagAddr != Tag of Way[i]:
4 :     if SecureAccess and SF of Way[i] == 1 or
5 :           NonSecureAccess and SF of Way[i] == 0:
6 :         Forward cache line to requestor (Cache hit occurs)
7 :     end if
8 :   end if
9 : end for
10: if UF == 0:
11:   for i ← 1 to N:
12:     if UF of Way[i] != SWG[i]:
13:       for j ← i+1 to N:
14:         if UF of Way[j] == SWG[j]:
15:           exchange cache lines Way[i] ↔ Way[j]
16:           UF of Way[i] ← 1 - UF of Way[i]
17:           UF of Way[j] ← 1 - UF of Way[j]
18:           exit to
19:         end if
20:       end for
21:     else
22:       UF of Way[i] = 1 - UF of Way[i]
23:       Invalidate cache line of Way[i]
24:     end if
25:   end for
26:   UF ← 1
27: end if
```

```
* Update cache partitioning Status
1 : Update SWG from the cache partitioning manager
2 : for i ← 1 to S:
3 :   UF ← 0
4 : end if
```

Fig. 3 Algorithm of dynamic cache partitioning.

3.2 Overall architecture

Fig. 4 depicts the overall architecture of SaPGA. The basic components of cache are given in gray boxes and the newly inserted components of SaPGA are illustrated in orange. For tag array, two kinds of flag bits were introduced and cache hit detection logic was modified. A secure bit (S bit) was appended to the conventional tag information of each cache line. It indicates that the corresponding cache line has permission for secure access. The cache hit detection logic utilizes the S bit to decide the cache hit as well as tag matching.

A table was also added to contain an array of update flag (U bit) for each cache set. A secure way guide (SWG) register, a cache partitioning manger, and a write counter array were inserted in the data array. When a cache line

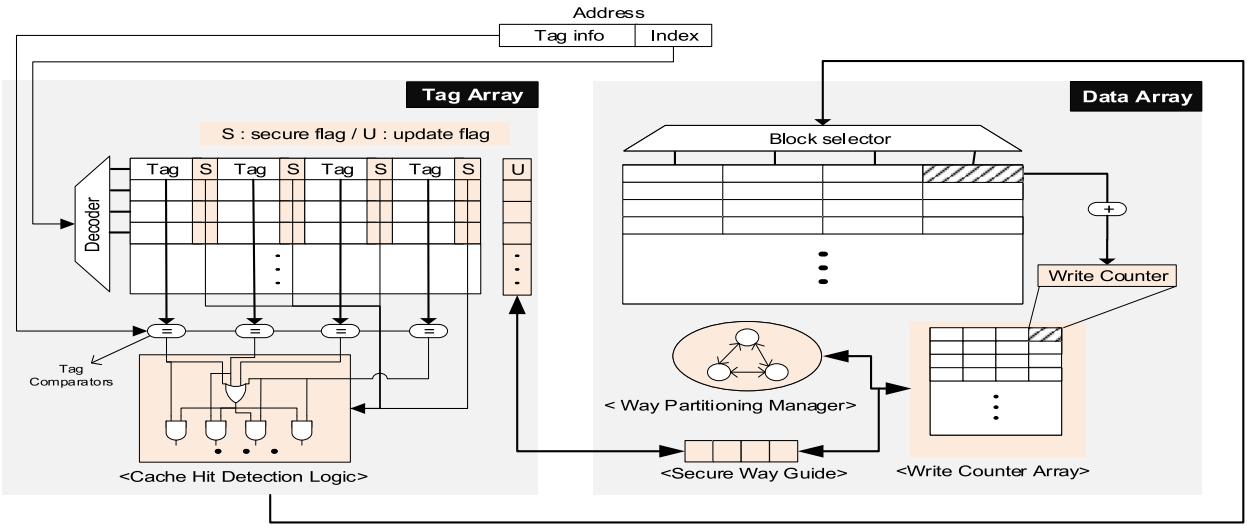

Fig. 4 Overall architecture.

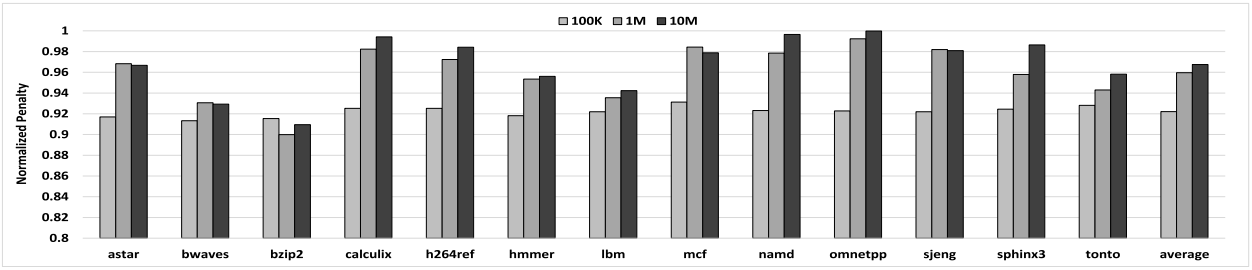

Fig. 5 Normalized penalty for dynamic cache partitioning adjustment.

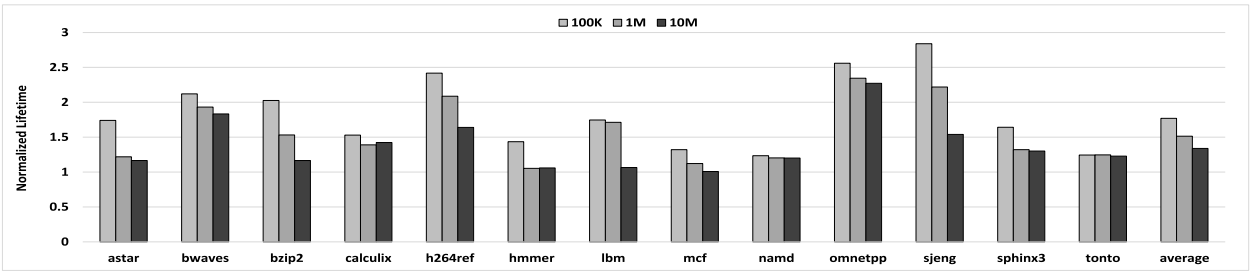

Fig. 6 Normalized lifetime.

is written, the corresponding write counter increased by 1. Periodically, the cache partitioning manager checks the variation of write counts in each cache set. If a particular way is more frequently updated than the other ways, it is exchanged by the least frequently written way in the same set. Note that this paper focuses on handling the weakness of NVM, thus, discussing cache partitioning mechanism itself is beyond the scope of this paper. Currently, the manager simply considers the write counts, however, the detailed combination of the other previous schemes and the manager in the SaPGA will be studied in the future work.

3.3 Overheads

To implement SaPGA, storage overhead needs to be discussed. Eq. (1) shows the relationship between extra area and cache configurations

$$Overhead\ Rate = \frac{4 * CL + S + M * CL + Total}{Total} \quad (1)$$

Table I Storage overheads.

Cache Configuration	Overhead Rate
8-way 256 KB	3.32%
8-way 512 KB	3.32%
8-way 1 MB	3.32%
16-way 2 MB	3.23%
16-way 4 MB	3.23%

where CL refers to the total number of cache lines, S means the number of sets, and Total indicates the original capacity of the cache.

The traditional secure-aware cache partitioning requires only N bits to indicate a cache partitioning status, where N is the number of ways. However, the proposed scheme inserts 1 bit (S bit) for each cache line. Furthermore, each cache set has a U flag for tag array while, the write counter array contains an M bits counter for every cache line. Table I lists the overhead rates for various cache configurations. Regard-

less of capacity and set-associativity of cache, around 3.3% extra storage is required on average.

4. Experimental results

Experiments were performed using gem5 [30], which is a well-known cycle-accurate simulator for cache studies. Some programs were selected from the SPEC2006 benchmark suite [31] to evaluate the effectiveness of the proposal. A two-level cache hierarchy was utilized for experiments; an L1 4-way 32 KB instruction cache, an L1 4-way 32 KB data cache, and unified L2 cache with various configurations, such as 8-way 256 KB, 8-way 512 KB, 8-way 1 MB, 16-way 2 MB, and 16-way 4 MB were used. The baseline for normalization is the value of the DAWG.

The main points of the simulation results are summarized in Fig. 5 and Fig. 6 with various sizes of adjustment periods from 100K to 10M cycles. The penalty in Fig. 5 refers to the extra cycles to flush cache lines when the cache partitioning status changes. If the cache partitioning adjustment is frequently performed, the penalty also increases, resulting in more performance hurt. A smaller normalized value implies a lesser amount of penalty is generated.

On average, about 7.8% access latency is reduced in SaPGA compared with that of DAWG for 100K. The penalty gap starts to narrow as the period extends to 1M or 10M. It was observed that 4.0% and 3.3% cycle overheads are saved for 1M-cycle period and 10M-cycle period, respectively. The improvement ratio of write endurance with various periods is shown in Fig. 6. On average, the lifetime was prolonged by 1.77 time, 1.51 times, and 1.33 times for 100K, 1M, and 10M period, respectively. This trend is opposite to that of the normalized penalty. In general, the cache access patterns significantly vary across program executions. Therefore, if inspection of the write endurance is frequently performed, it is helpful to modify the cache partitioning adaptive to the write behavior of the current working set.

5. Conclusion

The current study proposed SaPGA to compensate the problems when secure-aware cache partitioning schemes are applied to NVM-based cache structure. In the proposal, each cache line has its own secure flag bit instead of separate flag registers to indicate which ways are dedicated to the secure and the non-secure groups. By monitoring the write counts of data array, the frequently written cache lines are dynamically moved to the non-write-intensive ways to improve the write endurance of the system. The simulation results showed that the SaPGA achieved 77% lifetime improvement and 7.8% reduction in the penalty of dynamic adjustment of cache partitioning with small storage overhead.

Acknowledgments

This research was funded by a 2021 research Grant from Sangmyung University.

References

[1] F. Liu, *et al.*: "Last-level cache side-channel attacks are practical," IEEE Symposium on Security and Privacy (2015) 605 (DOI: 10.1109/SP.2015.43).
[2] Z. He and R.B. Lee: "How secure is your cache against side-channel attacks?," Annual IEEE/ACM International Symposium on Microarchitecture (2017) 341 (DOI: 10.1145/3123939.3124546).
[3] M. Lipp, *et al.*: "Meltdown: reading kernel memory from user space," Communications of the ACM **63** (2020) 46 (DOI: 10.1145/3357033).
[4] P. Kocher, *et al.*: "Spectre attacks: exploiting speculative execution," Communications of the ACM **63** (2020) 93 (DOI: 10.1145/3399742).
[5] A. Prout, *et al.*: "Measuring the impact of spectre and meltdown," IEEE High Performance Extreme Computing Conference (2018) 1 (DOI: 10.1109/HPEC.2018.8547554s).
[6] V. Kiriansky, *et al.*: "DAWG: A defense against cache timing attacks in speculative execution processors," Annual IEEE/ACM International Symposium on Microarchitecture (2018) 974 (DOI: 10.1109/MICRO.2018.00083).
[7] H. Omar, *et al.*: "OPTIMUS: a security-centric dynamic hardware partitioning scheme for processors that prevent microarchitecture state attacks," IEEE Trans. Comput. **69** (2020) 1558 (DOI: 10.1109/TC.2020.2996021).
[8] X. Zhang, *et al.*: "HiCa: hierarchical cache partitioning for low-tail-latency QoS over emergent-security enabled multicore data centers networks," 2020 IEEE International Conference on Communications (2020) 1 (DOI: 10.1109/ICC40277.2020.9148825).
[9] K.T. Sundararajan, *et al.*: "Cooperative partitioning: energy-efficient cache partitioning for high-performance CMPs," IEEE International Symposium on High-Performance Comp. Architecture (2012) 1 (DOI: 10.1109/HPCA.2012.6169036).
[10] N. Yamada, *et al.*: "Phase change memory," Proc. IEEE **98** (2010) 2201 (DOI: 10.1109/JPROC.2010.2070050).
[11] J. Zhu and C. Park: "Magnetic tunnel junctions," Materials Today **9** (2006) 36 (DOI: 10.1016/S1369-7021(06)71693-5).
[12] I. Inoue, *et al.*: "Nonpolar resistance switching of metal/binary-transition-metal oxides/metal sandwiches: homogeneous/inhomogeneous transition of current distribution," Physical Review B **77** (2008) 035105 (DOI: 10.1103/PhysRevB.77.035105).
[13] M. Dawber, *et al.*: "Physics of thin-film ferroelectric oxides," Reviews of Modern Physics **77** (2005) 1083 (DOI: 10.1103/RevModPhys.77.1083).
[14] B.S. Lee, *et al.*: "Anoscale nuclei in phase change materials: origin of different crystallization mechanisms of Ge2Sb2Te5 and AgInSbTe," Journal of Applied Physics **115** (2014) 063506 (DOI: 10.1063/1.4865295).
[15] Y. Joo, *et al.*: "Energy-and endurance-aware design of phase change memory caches," Design, Automation & Test in Europe Conference & Exhibition (2010) 136 (DOI: 10.1109/DATE.2010.5457221).
[16] S. Cho and H. Lee: "Flip-n-write: a simple deterministic technique to improve pram write performance, energy and endurance," International Symposium on Microarchitecture (2009) 347 (DOI: 10.1145/1669112.1669157).
[17] A.N. Jacobvitz, *et al.*: "Coset coding to extend the lifetime of memory," International Symposium on High Performance Computer Architecture (2013) 222 (DOI: 10.1109/HPCA.2013.6522321).
[18] X. Wu, *et al.*: "Hybrid cache architecture with disparate memory technologies," ACM SIGARCH Computer Architecture News **37** (2009) 34 (DOI: 10.1145/1555815.1555761).
[19] K. Kuan and T. Adegbija: "Energy-efficient runtime adaptable L1 STT-RAM cache design," IEEE Trans. Comput.-Aided Design Integr. Circuits Syst. **39** (2019) 1328 (DOI: 10.1109/TCAD.2019.2912920).
[20] J. Kong: "A novel technique for technology-scalable STT-RAM based L1 instruction cache," IEICE Electron. Express **13** (2016) 20160220 (DOI: 10.1587/elex.13.20160220).
[21] J.Y. Luo, *et al.*: "TAP: reducing the energy of asymmetric hybrid last-level cache via thrashing aware placement and migration," IEEE Trans. Comput. **68** (2019) 1704 (DOI: 10.1109/TC.2019.2917208).
[22] C. Liu, *et al.*: "Fast cache line-based data replacement for hybrid DRAM and STT-MRAM main memory," IEICE Electron. Express

17 (2020) 20200090 (DOI: 10.1587/elex.17.20200090).
[23] Y. Ro, *et al.*: "Selective DRAM cache bypassing for improving bandwidth on DRAM/NVM hybrid main memory systems," IEICE Electron. Express **14** (2017) 20170437 (DOI: 10.1587/elex.14.20170437).
[24] J. Zhan, *et al.*: "Energy-aware page replacement and consistency guarantee for hybrid NVM–DRAM memory systems," Journal of Systems Architecture **89** (2018) 60 (DOI: 10.1016/j.sysarc.2018.07.004).
[25] H. Park, *et al.*: "Application specific cache design using STT-RAM based block-RAM for FPGA-based soft processors," IEICE Electron. Express **15** (2018) 20180330 (DOI: 10.1587/elex.15.20180330).
[26] J. Lu, *et al.*: "A novel NVM memory file system for edge intelligence," IEICE Electron. Express **19** (2022) 20220079 (DOI: 10.1587/elex.19.20220079).
[27] M. Ni, *et al.*: "Elastic adaptive prefetching for non-volatile cache in IoT terminals," IEICE Electron. Express **19** (2022) 20220225 (DOI: 10.1587/elex.19.20220225).
[28] K.K. Dutta, *et al.*: "A fairness conscious cache replacement policy for last level cache," 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE) (2021) 695 (DOI: 10.23919/DATE51398.2021.9474096).
[29] M. Nejat, *et al.*: "Coordinated management of DVFS and cache partitioning under QoS constraints to save energy in multi-core systems," Journal of Parallel and Distributed Computing **144** (2020) 246 (DOI: 10.1016/j.jpdc.2020.05.006).
[30] J. Power, *et al.*: "Gem5-GPU: a heterogeneous CPU-GPU simulator," ACM SIGARCH Computer Architecture News **14** (2015) 34 (DOI: 10.1109/LCA.2014.2299539).
[31] J.L. Henning: "Spec cpu2006 benchmark descriptions," ACM SIGARCH Computer Architecture News **34** (2006) 1 (DOI: 10.1145/1186736.1186737).