Exploiting Bit-Level Write Patterns to Reduce Energy Consumption in Hybrid Cache Architecture

Juhee Choi and Heemin Park

Abstract A hybrid cache architecture (HCA) is introduced to alleviate the drawbacks of non-volatile memory (NVM) technologies. Although researchers have offered meaningful ways to conserve energy, little attention has been paid to focus on write counts that are non-uniformly spread over a cache line. We propose a novel HCA to reduce the NVM write counts by exploiting bit-level write patterns. The data array is refined to separately store bits in the cache line to the NVM region and the SRAM region. As a result, 20.1% of energy is saved over prior works.

key words: Non-Volatile Memory, STT-RAM, Energy saving techniques, Hybrid cache architecture

Classification: Integrated circuits

1. Introduction

Interests for reducing the area and energy consumption of cores are increasing to meet the surging demand for IoT devices. As a viable candidate for SRAM, non-volatile memory (NVM) has emerged for on-chip caches due to its significantly low amount of leakage and high density. However, NVM also has comparable penalties for write operations in terms of its large dynamic energy consumption and long write latency.

The purpose of a hybrid cache architecture (HCA) is to mitigate these drawbacks by reducing the write counts of NVM cells. Instead of implementing all of the storage in the cache as NVM, some cells are composed of SRAM to accommodate write-intensive cache lines. To increase the benefits of the SRAM region, several schemes have been suggested to detect frequently written cache lines and move them into the SRAM region. These approaches alleviated the write pressure of NVM; however, they overlooked the disparate write characteristics among programs. This paper starts by disclosing the non-uniform write intensity of each bit per cache line. This analysis becomes a step toward proposing a bit-level write pattern-based HCA (BLWP). We divide each cache line in the data array into chunks of the same size. Some chunks reside in the NVM region, while the other chunks are inserted in the SRAM region. Unlike a conventional HCA, both regions are used for the same cache line. To support this idea, we develop components to check the write counts of chunks and control the chunks during read or write access.

We perform experiments to evaluate the effectiveness of the proposed architecture. Among the several types of NVM, spin-torque transfer RAM (STT-RAM) is chosen for the simulation since it is mainly explored for the cache system. The results show that a 28.8% reduction in the NVM write counts is achieved and that the dynamic energy consumption is reduced by 20.1% on average.

2. Background and Motivation

2.1 Background

A variety of HCAs have been researched to compensate for the problems related to the write operations of NVM. These approaches tried to merge two different types of memories into a single memory system. One of the notable subjects for HCA is to predict the write pattern of the last-level cache. The read-write aware hybrid cache architecture (RWHCA) was proposed to embody a new replacement policy. When a load miss occurs, the incoming block is forwarded to the NVM region. In other cases, a cache line in the SRAM region is selected as a victim. If a block is frequently targeted by write access, the block in the NVM region is switched to the SRAM region. Ahn et al. elaborated the concept of the write intensity by referring to cache miss behavior. Adaptive placement and migration techniques categorize write requests in detail and utilize the types of writes during linefill operations. One of the state-of-the-art mechanisms for HCAs is access-based victim block placement (AVBP). The main contribution of AVBP is to extend the range of HCA studies to the whole memory subsystem, including the victim cache. The fields of HCA study have been diversely broadened. For a multicore environment, applying cache partitioning with an HCA has been studied. In addition to the last-level cache, the L1 cache is also becoming a target of HCAs.
rather than SRAM and NVM[19][20][21][22]. In addition, Park et al. attempted to apply HCA with FPGA[23].

2.2 Motivation
For inspiration, we revisit the existing analysis of the write accesses of NVM. It is known that the majority of data require fewer bits out of all of the bits in the cache line[24]. The authors of previous papers have enhanced the utility of SRAM in HCAs by exploiting narrow-width values[25][26]. Even though they are different as an aspect of ways to deal with the NVM region, they laid the common foundation of cache line granularity.

To investigate the write pattern of each bit, we depict the distribution of the write counts of benchmark programs as shown in Fig. 1. The bit width of a cache line is 512 (=64 B). First, write operations are concentrated on a small portion of the cache line, as mentioned above. However, we learn that the write pattern drastically differs from application to application. For games, the number of transitions is evenly distributed over the cache line, while the write operation positions are extremely unbalanced for libquan. Additionally, each application has its own positions of write-intensive bits, such as bwaves and hmmer. Consequently, it is sub-optimal that the fixed positions of the NVM cells are assigned to all programs based on an average write frequency. From this observation, a new scheme is required to monitor and exploit the bit-level write pattern for each application.

3. Bit-Level Write Pattern-based Hybrid Cache Architecture

3.1 Overall Architecture
To satisfy the requirement discussed above, we propose a bit-level write pattern-based HCA (BLWP). Fig. 2 presents the overall architecture of the conventional HCA and the BLWP. Note that our proposed architecture also employs the RBW[13] because it is suitable for taking advantage of the bit-level write pattern. The most distinguishing feature of the BLWP is that the cache line is separately stored in both the NVM region and the SRAM region. For the existing HCAs, the SRAM region and the NVM region are mutually exclusive (Fig. 2a). If the cache consists of N ways, the NVM region uses T ways out of N ways, and the SRAM region uses the remaining S ways (N = T + S). When the request to the cache arrives, the provided cache line comes from
either of them. In contrast, the SRAM region in the BLWP (Fig. 2b) contains some bits in the cache line, and the NVM region has the other bits in the same cache line.

Fig. 3 shows the concept of sub-blocks and chunks we invented to handle this structure. A cache line is composed of several sub-blocks, and a sub-block consists of some chunks. If the number of sub-blocks is $m$ and the number of chunks in a sub-block is $n$, a cache line consists of $m \times n$ chunks. In other words, the NVM region has $(m - 1)$ chunks, and the SRAM region has the reminder of the chunks for each sub-block.

In addition, special components are devised to manage the chunks: chunk flag bits (CFs), chunk combiner, write-intensive chunk detector, and chunk distributor. First, the tag array is extended to hold CFs to mark the position of the chunk located in the SRAM region. The width of the CFs is the same as the number of chunks. The role of the chunk combiner is to establish the original cache line. For the BLWP, a method is needed to restore the cache line because each cache line is split into the NVM region and the SRAM region. When the requester, such as a higher-level cache controller or core, asks the data of the target cache line, the chunk combiner merges chunks from both regions. In write-intensive chunk detector, we adopt saturated counters to observe the bit-level write pattern called chunk write counters (CWCs) and a chunk write pattern register (CWPR). The CWPR indicates which chunk should be stored in the SRAM region by gathering the most significant bit in the CWCs. Finally, we employ a chunk distributor to divide the data of the cache line into the NVM region and the SRAM region. Thus, it has the opposite function of the chunk combiner. There are muxes in the data array to select the chunks to be written.

3.2 Operations

Fig. 4 and Fig. 5 show the newly designed gadgets in detail and a description of their operations. For simplicity, we illustrate only the extra features for the BLWP and omit traditional components, including the RBW, such as tag matching or bit-comparison logics.

Cache operations are usually categorized into read operations, write operations, and linefill operations. Read operations simply generate read access to the cache, while write the operations and linefill operations require both read access and write access to the cache. A write operation means that the new value of the same cache line is written due to write-back from higher-level cache hierarchy or write instruction; the address of the cache line is not changed. A linefill operation includes the change in the tag information because the current cache line is replaced by another cache line.

For a read operation, in the beginning, the CFs in the tag array and the tag information are read. Next, all chunks in the SRAM region and the NVM region are activated. Simultaneously, the selection signals for muxes are generated. If the $k^{th}$ bit of the CFs is 1, the chunk combiner shifts the chunks after the $k^{th}$ chunk in the NVM array to the right and inserts the chunk from the SRAM region. Finally, the derived cache line is sent to the requester. For example, assume that $n$ is 2, $m$ is 4, the 3rd bit is 1, and the final output cache line is composed of $T_{11}, T_{12}, T_{13}, T_{21}, T_{22}, T_{23},$ and $T_{23}$.

The data flow for the write operation is presented in the bottom part of Fig. 4 and Fig. 5. In a write request to the cache, every chunk of the original cache line is compared to the chunks of the cache line to be written. Note that the logic is easily implemented by modifying the bit-level comparators of the RBW. If the $k^{th}$ chunk is modified, the $k^{th}$ CWC in
the write intensive chunk detector is increased. The CWPR is automatically updated according to the CWCSs. To store the new value, the chunk distributor generates the selection signals for the muxes by using the CFs. Subsequently, the selected chunks are written. The chunk distributor uses the value of the CWPR instead of the CFs for the linefill operation, as shown in Fig. 5. Then, the CFs are updated to the value of the CWPR for the new cache line. The reason for this updating is that the previous CFs denote the outdated bit-level write pattern.

**Table I:** Processor configurations

| Core Type | x86, out-of-order, 2 GHz |
|-----------|--------------------------|
| Inst. Cache Config. | Private, 32 KB SRAM, 4-way, 64B block size, 1-cycle latency |
| Data Cache Config. | Private, 32 KB SRAM, 4-way, 64B block size, 1-cycle latency |
| L2 Cache Config. | Shared, 1 MB 16-way (4-way SRAM and 12-way STT-RAM), 64B block size, 4 x 4 chunks in a cache line |
| L2 Cache Latencies (Read/Write) | SRAM: 6 / 6 cycles, STT-RAM: 6 / 23 cycles |
| L2 Cache Energy (Read/Write) | SRAM: 0.116 / 0.116 nJ, STT-RAM: 0.188 / 2.117 nJ |
| Victim Cache | 32 entries, SRAM, fully-associative, 64B block size, 2-cycle latency Read: 0.007 nJ, Write: 0.007 nJ |
| Main Memory | 2GB, 160-cycle latency |

4. Experimental Results

4.1 Experimental Setup
To evaluate the benefits of our proposed architecture, we built a simulation environment by modifying the gem5[28] cycle-accurate simulator. Some programs are chosen from the SPEC2006 benchmark suite[29] as workloads. The memory system comprises an L1 4-way 32 KB instruction cache and an L1 4-way 32 KB data cache along with a 16-way 1 MB unified L2 cache; this memory hierarchy is adopted from one of the Intel cores for the embedded system[27]. The NVSim model is applied with 32 nm technology[30], with STT-RAM for the energy consumption and latency of caches. More details of the experimental setup are given in Table I. To fairly compare the results of the BLWP with those of other studies, the parameters are derived from the AVBP. We also conducted experiments with the baseline hybrid cache (Base), RWHC, and AVBP. The standard of normalization in our results is the base, which has no special policy and is operated as a conventional cache, except that it consists of both SRAM and STT-RAM cells. The RWCHC is one of the most cited papers in hybrid cache architecture studies, and the AVBP is the state-of-the-art scheme for the combination of SRAM and NVM.

4.2 Reduction in NVM Write Accesses
Since the BLWP mainly aims to reduce the NVM write counts, we first examined the normalized write accesses to the NVM region for each policy (Fig. 6).

The RWCHC, AVBP, and BLWP decreased the NVM write counts by 42.0%, 54.4%, and 67.5% compared with the base, respectively. Over the AVBP, the BLWP obtained a 28.8% reduction in the NVM write counts. Upon closer examination, we additionally found significant savings in the write accesses for some programs, such as bwaves, cactus, libquan, and sjeng. In contrast, there are no advantages for other benchmarks, such as gromacs, games, namd, and sphinx3. This comes from the variation in the bit-level write pattern across the programs, as we discussed (refer to Section 2.2). The modification counts of the NVM cells of libquan for the BLWP are nearly one-fifth of the write counts for the AVBP because the write operations are considerable in a small region across the cache line. On the other hand, the number of state changes of sphinx3 are evenly distributed among the bytes, leading to the inefficient results of the BLWP.

In addition, we investigated the impact of the swap operations on the total write accesses. To reveal the portion of the write counts that occurred by the swap, each bar in the figure is partitioned into two parts by different colors. The first part (WR) gives the write accesses that occurred by traditional write operations, such as write instructions or cache linefills. The second part (SWAP) shows the write accesses caused by switching the cache lines between the NVM and SRAM regions. The portion of write accesses over the total write accesses for the swap operations of the RWCHC, AVBP, and BLWP are approximately 5.3%, 4.8%, and 4.2%, respectively. Thus, the overhead for the swap operation of the BLWP is not significant, nor is that of the other polices.
Fig. 6: Normalized NVM write accesses for various policies. Each bar is divided into the write access for demand, write (WR) and write access for swap (SWAP). There is no swap operation in the base.

Fig. 7: Normalized read and write energy consumption levels. Each bar is divided into the read energy (RD) and write energy (WR).

4.3 Reduction in Dynamic Energy Consumption

Fig. 7 provides a comparison between the normalized energy consumption of our proposed architecture and that of the various other policies. The normalized energy consumption is on average less than 37.6%, 47.4%, and 56.2% compared with the base. The BLWP achieved approximately 20.1% energy savings with respect to the AVBP. Overall, the energy consumption results are proportional to the trend of NVM write accesses. However, the reduction gap in the energy savings is lower than that of NVM write accesses. To investigate these differences further, we distinguish the read energy consumption (RD) from the write energy consumption (WR). The proportion of the read energy consumption over the total energy consumption for the BLWP is 26.9%, while those of the RWHCA and AVBP are 9.8% and 11.6%, respectively. In particular, cactus consumes more energy for read accesses than for write accesses. These increases mainly come from the read-before-write operations in the BLWP.

4.4 Storage Overhead

In this paper, we chose $n$ as 4, $m$ as 4, and $p$ as 32 bits. Based on these values, the width of the CWPR is 4, and 5-bit saturated counters are assigned to the CWCs. The L2 cache consists of 1024 sets with 16-way associativity. Since the width of the CFs is also 8, the total storage overhead is 4 bits x 1024 x 16 = 8 KB for CFs in the tag array and 4 + 5 x 4 = 24 flip-flops for the CWPR and CFs. Considering the capacity of the L2 cache, this storage overhead is negligible.

5. Conclusion

We proposed the BLWP to address the penalties of the NVM-based cache. The key idea is to separate the whole cache line into the SRAM region and the NVM region with bit-level granularity. By monitoring the bit-level write pattern, the
write-intensive portion in a cache line is dynamically loaded into the SRAM region. In our evaluation with STT-RAM, our proposed architecture improved the power efficiency by 20.1% over the AVBP.

Acknowledgments

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (NRF-2021R1G1A1004340).

References

[1] J. Zhu and C. Park: “Magnetic tunnel junctions,” Materials Today 9 (2006) 36 (DOI:10.1016/S1369-7021(06)71693-5)
[2] N. Yamada, et al.: “Rapid-phase transitions of getesbte3 pseudobinary amorphous thin films for an optical disk memory,” Journal of Applied Physics 69 (1991) 2849 (DOI:10.1063/1.348620)
[3] L. 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)
[4] J. F. Scott and C. A. P. De Araujo: “Ferroelectric memories,” Science 246 (1989) 1400 (DOI: 10.1126/science.246.4936.1400)
[5] WU, Xiaoxia, et al.: “Hybrid cache architecture with disparate memory technologies,” ACM SIGARCH Computer Architecture News 37 (2009) 34 (DOI: 10.1145/1555815.1555761)
[6] X. Wu, et al.: “Power and performance of read-write aware hybrid caches with non-volatile memories,” Design, Automation & Test in Europe Conference & Exhibition (DATE) (2009) 737 (DOI: 10.1109/DATA.2009.5909762)
[7] J. Ahn, et al.: “Write intensity prediction for energy-efficient non-volatile caches,” Low Power Electronics and Design (2013) 223 (DOI: 10.1109/ISLPED.2013.6629298)
[8] J. Ahn, et al.: “Prediction hybrid cache: An energy efficient STT-RAM cache architecture,” IEEE Transactions on Computers 65 (2016) 940 (DOI: 10.1109/TC.2015.2435772).
[9] S. Agarwal and H. Kapoor: “Improving the performance of hybrid caches using partitioned victim caching,” ACM Transactions on Embedded Computing Systems (TECS) 5 (2020) 1 (DOI: 10.1145/3411368).
[10] Z. Wang et al.: “Adaptive placement and migration policy for an STT-RAM-based hybrid cache,” 20th International Symposium on High Performance Computer Architecture (HPCA) (2014) 13 (DOI: 10.1109/HPCA.2014.6835933)
[11] Z. Diao, et al.: “Spin-transfer torque switching in magnetic tunnel junctions and spin-transfer torque random access memory,” Journal of Physics: Condensed Matter 19 (2006) 165209 (DOI: 10.1088/0953-8984/19/16/165209)
[12] S. Motaman, et al.: “Impact of process-variations in STT-RAM and adaptive boosting for robustness,” Design, Automation & Test in Europe Conference & Exhibition (DATE) (2015) 1431 (DOI:10.7873/DATE.2015.1018)
[13] Y. Joo, et al.: “Energy-and endurance-aware design of phase change memory caches,” Design, Automation & Test in Europe Conference & Exhibition (DATE) (2010) 136 (DOI: 10.1109/DATA.2010.5457221)
[14] N. Kim, et al.: “Benzene: an energy-efficient distributed hybrid cache architecture for manycore systems,” ACM Transactions on Architecture and Code Optimization (TACO) 15 (2018) 1 (DOI:10.1145/3177963)
[15] B. Donvanavard, et al.: “Exploring hybrid memory caches in chip multiprocessors,” 13th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (2018) 1 (DOI:10.1109/ReCoSoC.2018.8449386)
[16] S. Bhosal and S. Pasricha: “SLAM: High performance and energy efficient hybrid last level cache architecture for multicore embedded systems,” IEEE International Conference on Embedded Software and Systems (2019) 7 (DOI:10.1109/ICESS.2019.8782441)
[17] K. Kuan and T. Adegbija: “Energy-efficient runtime adaptable L1 STT-RAM cache design,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 39 (2019) 1328 (DOI:10.1109/TCAD.2019.2912920)
[18] 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)
[19] C. Liu, et al.: “Fast cache line-based data replacement for hybrid DRAM and STT-RAM main memory,” IEICE Electron. Express 17 (2020) 20200090 (DOI:10.1587/exle.17.20200090)
[20] 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/exle.14.20170437)
[21] 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)
[22] J. He and J. Callenes-Sloan: “A novel architecture of large hybrid cache with reduced energy,” IEEE Transactions on Circuits and Systems I 64 (2017) 3092 (DOI:10.1109/TCSI.2017.2719000)
[23] H. Park, et al.: “Application specific cache design using STT-RAM based block-RAM for FPGA-based soft processors,” IEICE Electron. Express 15 (2018) 201808330 (DOI:10.1587/exle.15.201808330)
[24] M. Imani, et al.: “DCM: Double capacity cache architecture for narrow-width values,” International Great Lakes Symposium on VLSI (2016) 113 (DOI:10.1145/2902961.2902990)
[25] M. Imani, et al.: “A low-power hybrid magnetic cache architecture exploiting narrow-width values,” 5th Non-Volatile Memory Systems and Applications Symposium (2016) 1 (DOI:10.1109/NVMSA.2016.7547174)
[26] G. Duan and S. Wang: “Exploiting narrow-width values for improving non-volatile cache lifetime,” Design, Automation & Test in Europe Conference & Exhibition (DATE) (2014) 1 (DOI:10.7873/DATE.2014.065)
[27] Intel Atom® Processor C2316 (2017) https://ark.intel.com
[28] J. Power, et al.: “Gem5-GPU: A Heterogeneous CPU-GPU Simulator,” ACM SIGARCH Computer Architecture News 44 (2015) 34 (DOI:10.1109/LCA.2014.2299539)
[29] J. L. Henning: “Spec cpu2006 benchmark descriptions,” ACM SIGARCH Computer Architecture News 34 (2006) 1 (DOI:10.1145/1186736.1186737)
[30] X. Dong, et al.: “Nvsm: A circuit-level performance, energy, and area model for emerging non-volatile memory,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 31 (2012) 994 (DOI:10.1109/TCAD.2012.2185930)