ADAM: Adaptive Block Placement with Metadata Embedding for Hybrid Caches

Beomjun Kim
Kyungpook National University
beomjun0816@knu.ac.kr

Prashant J. Nair
The University of British Columbia
prashantnair@ece.ubc.ca

Seokin Hong
Kyungpook National University
seokin@knu.ac.kr

***Abstract*—Spin-Transfer Torque Random Access Memory (STT-RAM) is a potential alternative for SRAM-based on-chip caches. STT-RAM offers high density and low leakage power, thereby can be used to build a large capacity last-level caches (LLC). Unfortunately, the write latency of the STT-RAM is significantly longer, and its write energy is considerably higher compared to SRAM. To mitigate these concerns, researchers have proposed hybrid caches that are comprised of SRAM and STT-RAM regions. In such hybrid caches, an intelligent block placement policy is necessary to store as many write-intensive blocks in the SRAM region. This paper proposes an adaptive block placement framework with metadata embedding (ADAM) for hybrid caches. ADAM embeds metadata (i.e., write-intensity) into a cache block when it is evicted from LLC. When a cache block is brought from the main memory, metadata embedded in the block is extracted and used to determine the write-intensity of the block. Our evaluation shows that ADAM can improve performance by 26% (on average) over a baseline block placement scheme.**

***Index Terms*—Last-level Cache, Hybrid Cache, Non-Volatile Memory, STT-RAM**

I. Introduction

On-chip caches occupy a large portion of the on-chip area in the modern processors. To make matters worse, the ever-growing working set of modern applications require larger on-chip last-level caches (LLC) [1]. Unfortunately, Static Random Access Memory (SRAM), the conventional memory technology for LLCs, does not scale well due to its high power consumption and low density. Therefore, many researchers are investigating emerging non-volatile memory technologies, such as Spin-Transfer Torque RAM (STT-RAM), as an alternative to SRAM. STT-RAM is attractive as it offers higher density and lower leakage power consumption over SRAM. However, STT-RAM suffers from long write latency and high power consumption on write operations, which can offset the attractive characteristics of STT-RAM.

As both SRAM and STT-RAM have strengths and weaknesses, to get the best of both worlds, researchers have proposed hybrid caches that integrate both SRAM and STT-RAM [2]–[4]. In the hybrid caches, the data array is partitioned into two regions: SRAM and STT-RAM regions. To mitigate long write latency and high write power consumption of the STT-RAM, the hybrid caches employ an adaptive block placement policy to allocate write-intensive blocks to the SRAM region. As the read-latency and power of the STT-RAM region are low, hybrid caches try to place the read-intensive blocks in the STT-RAM region proactively. Thus, an optimal block placement policy is crucial for efficient hybrid caches.

Prior proposals on the block placement policy predict the write-intensity of blocks every time they are installed in the LLC. After the block is installed, the actual write-intensity is learned during the execution of the program. If the write-intensity prediction was incorrect, the block is migrated from the STT-RAM region to the SRAM region and vice versa. Write-intensity prediction is a key challenge for the hybrid cache. When a cache block is brought from the main memory, the reference history for the block is not available because all information about a block is removed when it is evicted from the LLC. Therefore, it is likely that the prediction is incorrect and can result in significant performance degradation.

To tackle this problem, we propose ADAM, a new adaptive block placement framework with metadata embedding. ADAM is based on the key observation that the write-intensity of the cache blocks tends to be almost constant during the execution of a program. Based on this finding, ADAM embeds the write-intensity metadata within the cache block by using the data compression techniques. When a block is brought from the main memory, the embedded metadata is extracted from the block and used to determine the appropriate region (STT-RAM or SRAM) for placing the block. By using the metadata embedding technique, ADAM can track the write-intensity of an individual block without additional storage elements.

II. Motivation

A. Limitation of Prior Work: Loss of the Metadata on Eviction

Prior approaches predict the write-intensity of the blocks when loading them into the caches, and then keep tracking the number of writes on the individual block to determine its actual write-intensity. If it turns out that the prediction was incorrect, the corresponding block is migrated into an appropriate region. These approaches can work well only if the target applications have high data locality or the high accuracy of the write-intensity predictor. In these approaches, when a block is evicted from the cache, the metadata (e.g., write-counter value) that was used to determine the write-intensity of the block is also removed from the cache. Therefore, whenever a cache block is loaded from the main memory, it is required to *re-learn* the actual write-intensity of the block.

2576-6996/20/$31.00 ©2020 IEEE
DOI 10.1109/ICCD50377.2020.00077

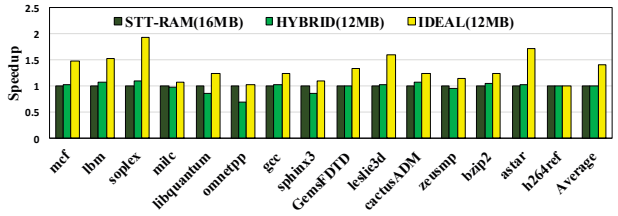


Fig. 1: Performance of hybrid cache using a baseline block placement technique.

Figure 1 shows the performance of three LLC caches: 16MB STT-RAM-based cache (denoted by STT-RAM), 12MB hybrid cache with a baseline block placement (denoted by HYBRID), 12MB hybrid cache with an ideal block placement (denoted by ideal case). *These designs are chosen because they are expected to consume similar on-chip area in our simulation with NVSim [5].* The baseline block placement policy only uses the type of instructions triggering the cache misses as done in [6]. The experimental environment in this section is the same as we used in section V. For this experiment, we used memory-intensive benchmarks showing high MPKI (Miss per kilo instruction). As shown in the Figure 1, the hybrid cache with an ideal block placement, where all write-intensive blocks are located to the SRAM region, achieves a speedup of 41% (on average). However, the hybrid cache using the naive block placement achieves limited performance improvement. Even for some benchmarks such as omnetpp and sphinx3, using a hybrid cache instead of the STT-RAM-based cache incurs performance degradation.

The main cause of the hybrid cache's limited performance gain for some benchmarks is frequent block eviction on the LLC due to the limited data locality in the benchmarks. Even if the write-intensity of cache blocks is determined while the blocks reside in the LLC, it is removed when the corresponding block is evicted from the LLC. To maintain each block's write-intensity, we can store metadata regarding the write-intensity of the block in the main memory and use a metadata cache to store the metadata of the frequently or recently referenced blocks. However, this approach cannot mitigate this problem for memory-intensive workloads with irregular memory access patterns, as studied in [7].

B. Opportunity: Write-intensity is Almost Constant

To design a novel data placement scheme, we start with a workload characterization on our simulation infrastructure. We observe that the write-intensity (WI) of a cache block is almost constant during the execution across several workloads. Figure 2 shows the distribution of the cache blocks of which write-intensity does not change. On average, the write-intensity is not changed for 98% of the cache blocks fetched from the main memory. This motivational result indicates that the write-intensity of a block can be used to predict the future write-intensity of the block multiple times once it is learned.

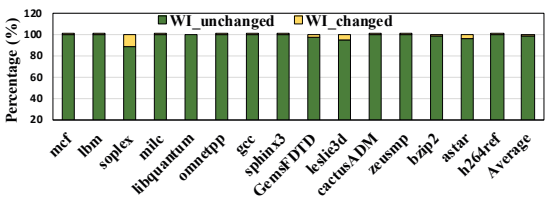


Fig. 2: Ratio of blocks with constant write-intensity (WI).

III. ADAM: Adaptive Block Placement with Metadata Embedding

A. Overview

To fully exploit the benefits of the hybrid caches, we propose ADAM, an adaptive block placement framework with Metadata Embedding. Figure 3 shows an overall architecture of the hybrid cache with the ADAM framework. The ADAM framework consists of four components. First, the per-block write-counter in the tag array that counts the number of writes. Second, a write-intensity detection unit for determining the number of writes. Third, a metadata embedding unit which reads or writes the metadata while reading or writing a cache block from the main memory system. Fourth, the block placement component that places the block appropriately into the SRAM or STT-RAM based on write-intensity during reads.

When a cache block is modified within the LLC, the block is marked dirty, and the write-counter for the block is incremented. Thereafter, when a dirty cache block is evicted from the LLC, the write-intensity detection unit uses the write-counter to generate the metadata for the cache block. The cache block, along with the metadata, is then transferred into the metadata embedding unit. The metadata embedding unit compresses the cache block and places the metadata alongside the block before writing it into the main memory.

During a read, the metadata embedding unit tries to extract the metadata out of the cache block. If the cache block contains the metadata that deems the block write-intensive, the block placement unit stores the block in the SRAM region. If the metadata indicates that the block is not write-intensive, the block is placed in the STT-RAM region.

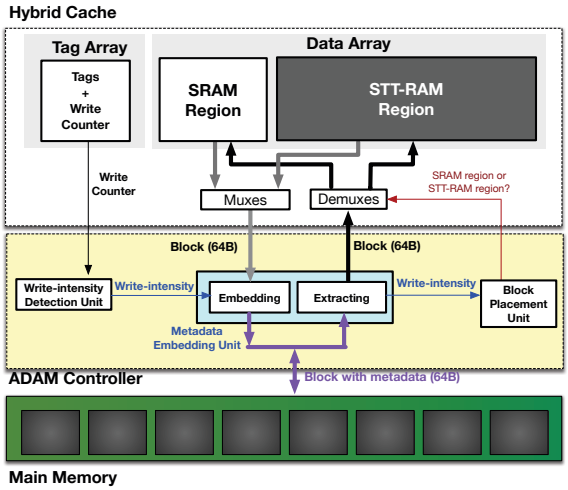


Fig. 3: The ADAM framework

B. Per-Block Write Counter

The write-intensity of a cache block is tracked using a 3-bit saturated counter called a write-counter. Tag array contains the write-counters, as shown in Figure 3. The write-counter for a block is incremented by one on a write hit. On the other hand, on a read hit, the counter is decremented by one.

C. The Write-Intensity Detection Unit

The 3-bit write-counters from the tags are probed by the write-intensity detection unit. The write-intensity detection unit compares the write-counter value of the victim block with a write-intensity threshold. If the write-counter value is greater than the threshold, the write-intensity detection unit generates 1-bit metadata indicating a low or high write-intensity.

D. The Metadata Embedding Unit

The metadata embedding unit tries to place the 1-bit metadata within the cache block. Unfortunately, cache blocks are typically 64 Bytes in size, and when placed in memory, they do not have any additional space to store any metadata.

Prior work Attaché [7], overcomes the problem of embedding metadata by using blended metadata. The metadata embedding unit evolves the blended metadata framework to store the write-intensity metadata. Unlike Attaché that needs to compress a 64-Byte block to 30 Bytes, ADAM only needs to compress a 64-Byte block up to 61 Bytes. In this work, the metadata embedding unit uses the Base-Delta-Immediate (BDI) and Frequent-Pattern-Compression (FPC) techniques to compress a cache block to at least 61 Bytes and chooses the best of the two [8], [9]. In our experiments, 82% of the blocks are compressible to less than 61 bytes on average.

The metadata embedding unit stores a 2-byte signature alongside the 61-Byte compressed cache block, as shown in Figure 4. Similar to the Attaché framework [7], the 2-byte signature consists of a 15-bit Compression ID (CID) and a 1-bit Exclusive ID (XID). The CID helps identify compressed cache blocks, and XID helps detect CID collisions and eliminate false positives. The metadata embedding unit then stores 1-byte of metadata alongside the signature, as shown in Figure 4. The 1-byte metadata contains 1 bit to indicate the type of compression technique and 1 bit to indicate the write-intensity. The remaining 6 bits can be used to store any information regarding the corresponding cache block.

If the cache block is compressible, the metadata embedding unit stores the signature (2 Bytes), metadata (1 Byte), and compressed data (61 Bytes) tuple into the memory system. If the data is not compressible, then the data is stored as it is. However, like Attaché, if the first 15 bits of the uncompressed data collide with the CID, then the 16th bit (XID) is set to 0, and the original 16th bit of the data is placed in a separate region within the main memory.

E. The Block Placement Unit

On a read, the metadata embedding unit decompresses the cache block and extracts the write-intensity metadata. The metadata embedding unit then forwards the write-intensity information to the block placement unit. If the block is deemed to be write-intensive, then the block placement unit places this block into the SRAM region. If not, the block placement unit places this block into the STT-RAM region.

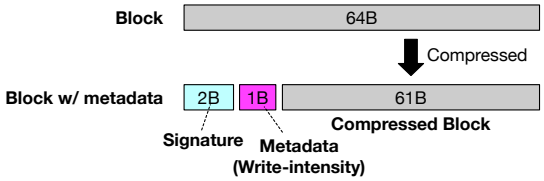


Fig. 4: Embedding metadata into the cache block.

F. Selectively Writing Back Clean Blocks

An evicted block is deemed clean if it is not updated during its lifetime in the cache. Clean evicted blocks are traditionally not written back into the main memory to save memory bandwidth. However, a clean cache block can change its write-intensity (WI). For instance, suppose a block with a high write-intensity is read into the cache. Thereafter, the block may encounter only reads during its lifetime in the cache. This would decrement the write-intensity counter. As the block remains clean, on eviction, this block would not be written back to the memory. However, as there is a change in write-intensity, the ADAM framework will write back these clean evicted blocks to the main memory. We call the write requests for the clean evicted blocks as *Clean Writes (CW)* in this paper.

IV. Evaluation Methodology

To evaluate the performance benefits of the ADAM, we developed a hybrid-cache simulator based on USIMM [10]. Table I lists the simulated system configuration. The LLC is configured to have multiple banks to service multiple requests in parallel.

TABLE I: Baseline System Configuration

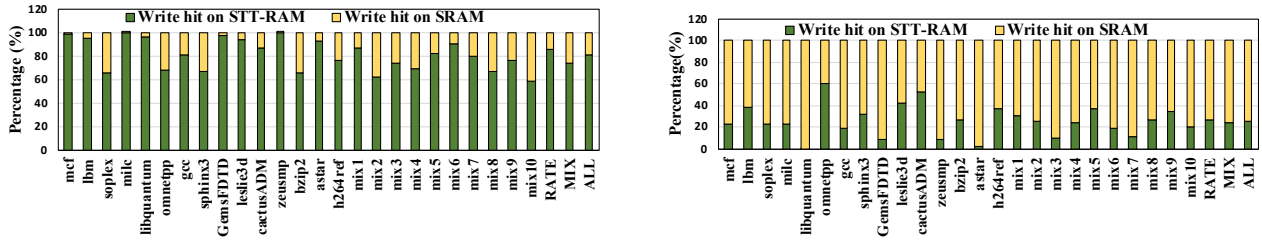
Processor	3.2GHz, 4 cores
L1 Cache L2 Cache	32KB, 8-Way, 4 cycles 256KB, 8-Way, 12 cycles
LLC (Hybrid Cache)	12MB (SRAM: 4MB, STT-RAM: 8MB), 16-Way SRAM Read/Write: 30 cycles STT-RAM Read : 30 cycles STT-RAM Write : 90 cycles

The efficiency of ADAM is compared to a baseline and an ideal block placement. The baseline block placement scheme predicts the write-intensity of a cache block only with the type of operation (i.e., load or store) triggering a cache miss. In the ideal scheme, we assume that all write-intensive blocks are allocated to the SRAM region.

V. Simulation Results

A. Write Hits on SRAM and STT-RAM Regions

Figure 5 shows the distribution of write hits on LLC. The primary goal of the block placement scheme for the hybrid cache is to reduce write-hits on the STT-RAM region to



(a) Baseline hybrid cache

(b) Hybrid cache with ADAM

Fig. 5: Distribution of write hits on hybrid caches.

mitigate a long write latency and high energy consumption of the STT-RAM. As shown in the Figure, ADAM yields low write hits on the STT-RAM region compared to the baseline scheme for all of the benchmarks. On average, the percentage of write hits on the STT-RAM region is reduced from 81% to 25%.

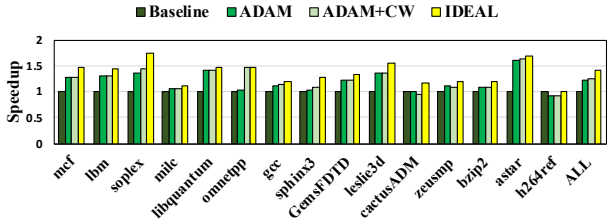


Fig. 6: The performance improvement of ADAM over baseline block placement.

B. Performance

Figure 6 shows the speedup of ADAM when compared to a baseline block placement. ADAM improves performance by 24% on average. Ideally, if we allocate all write-intensive blocks to the SRAM region, we get a speedup of 40% on average. The performance results show that libquantum and astar benefit the most from ADAM due to the dramatic reductions in the write hits on the STT-RAM region. Our analysis shows that the Clean Write (CW) scheme can further improve the performance for some benchmarks such as omnetpp by writing back the clean blocks to the main memory to maintain the write-intensity information. For omnetpp benchmark, ADAM delivers a speedup of 4% without the Clean Write scheme. With the Clean Write scheme, ADAM achieves a speedup of 46% for the omnetpp benchmark, which is comparable to the speedup with ideal block placement. On average, ADAM achieves a speedup of 26% when the Clean write is applied. Most benchmarks can benefit from accurate block placement with ADAM. However, for some benchmarks such as cactusADM and h264ref, ADAM shows lower performance compared to the baseline scheme. The performance degradation for this benchmark is caused by the increased misses on LLC. In hybrid caches, the SRAM region is smaller than the STT-RAM region, and therefore when many blocks are allocated to the SRAM region, the LLC miss rate will increase. To address this problem, we can extend the ADAM to consider the pressure on the SRAM region as well as the block's write-intensity.

VI. Conclusion

In this paper, we proposed ADAM, a new adaptive block placement framework with metadata embedding for hybrid caches. ADAM maintains the write-intensity of an individual block by embedding metadata in the cache block. When installing a block in the hybrid cache, ADAM extracts the embedded metadata and uses it to determine the write-intensity of the block. ADAM provides an efficient framework for hybrid cache management by enabling the storage of metadata without additional storage elements.

Acknowledgments

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) [NRF-2019R1G1A1011403]. Seokin Hong is the corresponding author.

References

[1] S. Hong, B. Abali, A. Buyuktosunoglu, M. B. Healy, and P. J. Nair, "Touché: Towards ideal and efficient cache compression by mitigating tag area overheads," in *52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019.

[2] G. Sun, X. Dong, Y. Xie, J. Li, and Y. Chen, "A novel architecture of the 3d stacked mram l2 cache for cmps," in *15th IEEE International Symposium on High Performance Computer Architecture*, 2009.

[3] Z. Wang, D. A. Jiménez, C. Xu, G. Sun, and Y. Xie, "Adaptive placement and migration policy for an stt-ram-based hybrid cache," in *20th IEEE International Symposium on High Performance Computer Architecture*.

[4] Y. Chen, J. Cong, H. Huang, B. Liu, C. Liu, M. Potkonjak, and G. Reinman, "Dynamically reconfigurable hybrid cache: An energy-efficient last-level cache design," in *2012 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2012.

[5] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, "Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 7, pp. 994–1007, 2012.

[6] Xiaoxia Wu, Jian Li, Lixin Zhang, E. Speight, and Yuan Xie, "Power and performance of read-write aware hybrid caches with non-volatile memories," in *2009 Design, Automation Test in Europe Conference Exhibition*, 2009.

[7] S. Hong, P. J. Nair, B. Abali, A. Buyuktosunoglu, K. Kim, and M. Healy, "Attaché: Towards ideal memory compression by mitigating metadata bandwidth overheads," in *51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2018.

[8] G. Pekhimenko *et al.*, "Base-delta-immediate compression: Practical data compression for on-chip caches," in *21st International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2012.

[9] A. R. Alameldeen and D. A. Wood, "Frequent pattern compression: A significance-based compression scheme for l2 caches," *Dept. Comp. Scie., Univ. Wisconsin-Madison, Tech. Rep*, vol. 1500, 2004.

[10] N. Chatterjee *et al.*, "Usimm: the utah simulated memory module a simulation infrastructure for the jwac memory scheduling championship," 2012.