electronics

Article

Exploiting Data Compression for Adaptive Block Placement in Hybrid Caches

Beomjun Kim [1], Yongtae Kim [1], Prashant Nair [2] and Seokin Hong [3,*]

[1] School of Computer Science and Engineering, Kyungpook National University, Daegu 41566, Korea; beomjun0816@knu.ac.kr (B.K.); yongtae@knu.ac.kr (Y.K.)
[2] Department of Electrical and Computer Engineering, University of British Columbia, Vancouver, BC V6T 1Z4, Canada; prashantnair@ece.ubc.ca
[3] Department of Semiconductor Systems Engineering, Sungkyunkwan University, Suwon 16419, Korea
* Correspondence: seokin@skku.edu

Abstract: STT-RAM (Spin-Transfer Torque Random Access Memory) appears to be a viable alternative to SRAM-based on-chip caches. Due to its high density and low leakage power, STT-RAM can be used to build massive capacity last-level caches (LLC). Unfortunately, STT-RAM has a much longer write latency and a much greater write energy than SRAM. Researchers developed hybrid caches made up of SRAM and STT-RAM regions to cope with these challenges. In order to store as many write-intensive blocks in the SRAM region as possible in hybrid caches, an intelligent block placement policy is essential. This paper proposes an adaptive block placement framework for hybrid caches that incorporates metadata embedding (ADAM). When a cache block is evicted from the LLC, ADAM embeds metadata (i.e., write intensity) into the block. Metadata embedded in the cache block are then extracted and used to determine the block's write intensity when it is fetched from main memory. Our research demonstrates that ADAM can enhance performance by 26% (on average) when compared to a baseline block placement scheme.

Keywords: last-level cache; hybrid cache; non-volatile memory; STT-RAM

Citation: Kim, B.; Kim, Y.; Nair, P.; Hong, S. Exploiting Data Compression for Adaptive Block Placement in Hybrid Caches. *Electronics* **2022**, *11*, 240. https://doi.org/10.3390/electronics11020240

Academic Editor: José L. Abellán

Received: 6 December 2021
Accepted: 7 January 2022
Published: 12 January 2022

1. Introduction

Modern processors use on-chip multi-level caches to compensate for main memory systems' restricted latency and bandwidth. Unfortunately, on-chip caches take up a significant amount of space. To make matters worse, the ever-expanding working set of modern applications, as well as their bandwidth demands, necessitates industry manufacturers in providing larger on-chip last-level caches (LLC). However, during the last few decades, LLC capacity per core has remained constant. Due to its high power consumption and low density, Static Random Access Memory (SRAM), the traditional memory technology for LLCs, does not scale well. As a result, numerous researchers are looking into non-volatile memory technologies such as Spin-Transfer Torque RAM (STT-RAM) as a potential replacement for SRAM. STT-RAM is appealing because it has a higher density and lower leakage power consumption than SRAM, allowing it to scale more efficiently. However, STT-RAM has a significant write latency and consumes a lot of power during write operations, which can negate the benefits of the STT-RAM. By using a combination of STT-RAM and SRAM technologies, this research intends to provide a system that enables performance-efficient LLCs.

As both SRAM and STT-RAM have advantages and disadvantages, researchers have developed hybrid caches that combine the best of both worlds [1–5]. The data array in hybrid caches is divided into two regions: SRAM and STT-RAM. Hybrid caches utilize an adaptive block placement policy to allocate write-intensive blocks to the SRAM area, which helps to offset the STT-RAM's long write latency and high write power consumption. Hybrid caches strive to insert read-intensive blocks in the STT-RAM area ahead of time due

to its low read latency and low read power consumption. As a result, for efficient hybrid caches, an optimal block placement policy is critical.

Prior proposals on the block placement policy predict the write-intensity of the cache blocks each time they are installed in the LLC. The exact write intensity is learned during program execution once the cache block is implemented. The block is migrated from the STT-RAM region to the SRAM region if the predicted write intensity is inaccurate and vice versa. The hybrid cache faces a significant issue in predicting write intensity. Due to the fact that all information about a block is discarded when it is evicted from the LLC, the reference history for a cache block is not available when it is brought from the main memory. As a result, the prediction is highly likely to be incorrect, resulting in significant performance degradation. Figure 1 shows the percentage of writes to the STT-RAM region in the hybrid-caches for high memory-intensive SPEC CPU2006 workloads. Even after employing an intelligent baseline policy that predicts write intensity based on whether LLC misses are reads or writes, nearly 81% of writes are directed to STT-RAM. This paper aims to reduce the number of writes into STT-RAM (by nearly 20%) while achieving near-ideal performance benefits.

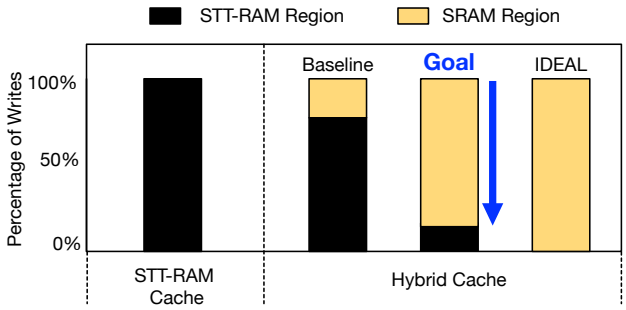


Figure 1. Ratio of writes on STT-RAM and SRAM regions. A baseline hybrid cache with intelligent block placement can have up to 81% of the writes into an STT-RAM bank (according to our experimental results that will be discussed in detail in Section 5.1). The goal of this paper is to reduce the number of STT-RAM writes to nearly 20% and obtain near-ideal performance.

In order to address this problem, we introduce ADAM, a new adaptive block placement framework with metadata embedding. ADAM is based upon two significant observations. First, the write-intensity of cache blocks is nearly constant during program execution. Second, after being evicted from LLCs, the majority of cache blocks are re-fetched. ADAM utilizes data compression techniques to embed write-intensity metadata within the cache block based on these two observations. When a block is read from the main memory, embedded metadata is retrieved and used to indicate the region (STT-RAM or SRAM) the block should be placed in. By using the metadata embedding technique, ADAM can track the write intensity of a single block without the use of additional storage components. The following includes the highlights of this paper's contributions:

- We make two key observations about write intensity and re-fetch rate of cache blocks;
- We propose a new adaptive block placement framework for hybrid caches based on a metadata embedding technique. This allows the write intensity of cache blocks to be determined precisely without the need for additional storage.
- We evaluate the performance of the proposed block placement framework for memory-intensive SPEC CPU2006 benchmark running on a simulated multicore processor. In comparison to a baseline block placement scheme, the proposed framework provides a speedup of 26% on average.

2. Background and Motivation

2.1. STT-RAM/SRAM-Based Hybrid Caches

2.1.1. STT-RAM

Spin-Transfer Torque Random Access Memory (STT-RAM) is a dense non-volatile memory technology [1,2]. STT-RAM has a read access time that is similar to SRAM, and its static power consumption is much lower than the SRAM. Therefore, it is expected that STT-RAM can be used to build large-capacity on-chip caches, such as the last-level cache for the multicore processors. STT-RAM uses a magnetic tunnel junction (MTJ), which is composed of two ferromagnetic layers and an oxide barrier (MgO). One of the two ferromagnetic layers is called a reference layer, and the other ferromagnetic layer is called a free layer. The magnetic orientations of the layers within the MTJ determine the resistance of the STT-RAM cell. When the magnetic directions of the free layer and the reference layer are in the same direction (i.e., parallel state), the resistance of MTJ is low, and when the magnetic directions of the two layers are in different directions (i.e., anti-parallel state), the resistance of MTJ is high. We can use one of two states of the MTJ to represent logic '0' or '1'.

The advantages of STT-RAM are obtained at the cost of a high-latency write operation. This is because updating the state of the STT-RAM cell involves updating the states of its physical material. An STT-RAM write requires the injection of a high write current into MTJ for a long time. These long-latency operations update the magnetic orientation of a layer within the MTJ, essentially changing the contents of the STT-RAM cell. Therefore, STT-RAM suffers from higher write latency and higher write energy consumption compared to the SRAM. Table 1 compares area, latency, dynamic energy, and leakage power of the SRAM-based and STT-RAM-based caches. These parameters are obtained using NVsim [1]. As shown, the STT-RAM-based cache has significantly higher write latency and write dynamic energy consumption than the SRAM-based cache. In contrast, SRAM-based cache consumes much more leakage power than STT-RAM-based cache.

Table 1. Area, latency, and energy consumption of SRAM-based and STT-RAM-based caches.

	SRAM-Based Cache (16 MB)	STT-RAM-Based Cache (16 MB)
Area (mm^2)	11.448	5.024
Read Latency (ns)	6.589	6.904
Write Latency (ns)	3.274	11.898
Read Dynamic Energy (nJ)	0.191	0.361
Write Dynamic Energy (nJ)	0.182	1.127
Leakage Power (mW)	462.891	64.824

2.1.2. Hybrid Caches

As a solution to these problems, researchers have proposed hybrid caches that use the SRAM as well as STT-RAM as their memory cells [3–7]. In hybrid caches, a data array is partitioned into SRAM and STT-RAM regions, as shown in Figure 2, and frequently written cache blocks, which we call write-intensive blocks, are allocated to the SRAM region to reduce write activity in the STT-RAM region. Since write latency and write energy of the SRAM are much smaller than those of typical STT-RAM, it is necessary to reduce write activities in the STT-RAM, thereby improving overall system performance and reducing dynamic energy consumption.

In order to minimize the required modification in the cache design, a tag array of the hybrid caches has a structure similar to that of conventional SRAM-based caches. The cache ways in both the SRAM and STT-RAM regions are treated similarly when cache access occurs, even though the number of ways in the SRAM region is physically smaller than that in the STT-RAM region. In addition, a tag array is implemented by only

using the SRAM. This is because the write latency of the tag array is critical in terms of the performance since it contains several metadata such as dirty bits and replacement information as well as tags, which are frequently updated when a cache is accessed. In particular, the last-level caches usually have a higher cache miss rate than the level-1 or level-2 caches, which results in frequent updates on the tag. Since the size of the tag array is much smaller than the data array, its contribution to the chip area and the energy consumption is small, even if it is implemented only with SRAMs.

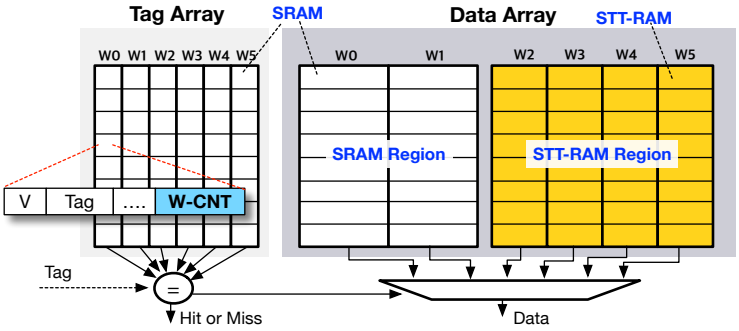


Figure 2. Baseline hybrid cache architecture. The data array is partitioned into SRAM and STT-MRAM regions. The tag arrays have a write counter (W-CNT). The method for SRAM and STT-RAM regions is treated the same manner when cache access occurs.

2.2. Hurdle: Adaptive Block Placement in Hybrid Cache

In hybrid caches, it is essential to store write-intensive blocks in the SRAM region as much as possible in order to minimize write operations on the STT-RAM region. To this end, several block placement policies have been proposed to intelligently place write-intensive blocks in SRAM regions [3–7]. On a cache miss, the write-intensity of the incoming block is predicted to determine an appropriate region (i.e., SRAM or STT-RAM) for the block.

After installing the block in the hybrid cache, the number of write operations on each block needs to be tracked continuously to determine its actual write-intensity. To this end, a write-counter (W-CNT) can be used for each tag entry, as shown in Figure 2. The counter is increased for every write operation on the corresponding block, and if the counter value is greater than a write intensity threshold value, the block is considered as write intensive. On a misprediction, a write-intensive block can be installed in the STT-RAM region. In such a situation, the write-intensive block is migrated from the STT-RAM to the SRAM region. While block migration can reduce the impact of misprediction, frequent migrations can increase dynamic energy consumption and degrade the overall performance of hybrid caches.

Due to the importance of the write-intensity prediction, many researchers have proposed write-intensity prediction schemes [3,4,8]. In [3,4], a simple heuristic is used for the prediction. On a cache miss, if the miss is triggered by a store instruction, the incoming block is predicted as a write-intensive block; therefore, the block is installed in the SRAM region. On top of this, the memory address of load instructions is used to determine the write-intensity of cache blocks when they are loaded due to read misses [8].

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

Prior techniques predicted the write intensity of the blocks when placing them into caches, then the amount of writes on each block is tracked to ascertain its actual write intensity. If the prediction turns out to be inaccurate, the associated block is relocated to the correct region. These methods can only be effective if the target applications have increased data locality or if the write-intensity prediction is very accurate. When a block is evicted from the cache, metadata (e.g., the write-counter value) used to determine the block's write-intensity are also discarded from the cache. As a result, whenever a cache

block is loaded from main memory again, it must *relearn* the block's real write-intensity. This can result in inaccurate write-intensity predictions and lower LLC performance.

The performance of three LLC caches can be observed in Figure 3: a 16 MB STT-RAM-based cache (denoted by STT-RAM), a 12 MB hybrid cache with a baseline block placement (denoted by HYBRID), and a 12 MB hybrid cache with an ideal block placement (denoted by IDEAL). These designs are chosen because they are estimated to consume a equivalent on-chip area in our simulation with NVSim [1]. As shown in [4], the baseline block placement policy only uses a certain kind of instruction that induce cache misses. In this section, we adopt the same experimental environment as in Section 5. We employed memory-intensive benchmarks with high MPKI (Miss per kilo instruction) in this experiment. The hybrid cache with an optimum block location, where all write-intensive blocks are placed in the SRAM region, provides a speedup of 41% (on average), as shown in Figure 3. The hybrid cache with naive block placement, on the other hand, produces very minor performance enhancements. Using a hybrid cache instead of the STT-RAM-based cache degrades performance for some benchmarks, such as omnetpp and sphinx3.

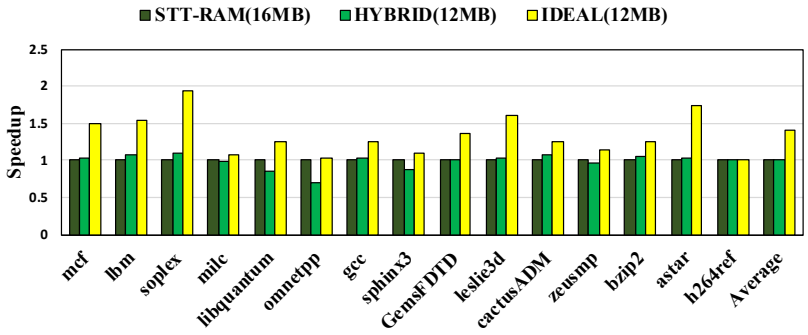


Figure 3. Performance of hybrid cache using a baseline block placement technique. Overall, using a large 16 MB STT-RAM cache (with the same on-chip area) provides the same performance as a 12 MB hybrid cache. This is because write-intensity mispredictions of the baseline block placement technique offset the write-latency savings on the SRAM region. Ideally, we can obtain 41% speedup using a 12MB hybrid cache with oracle predictions.

The main cause of the limited performance improvement of the hybrid cache for some benchmarks is frequent block eviction on the LLC due to the limited data locality in those benchmarks. The write-intensity of a cache block is determined while the block resides in the LLC. However, when the eviction of the block occurs in the LLC, write-intensity information associated with the victim block is also eliminated from the LLC. We can store information about the write intensity of each block in main memory and utilize a metadata cache to store metadata of frequently or recently referenced blocks to maintain the write intensity of each block. However, as studied in [9], this strategy cannot overcome this problem for memory-intensive applications with irregular memory access patterns. It also requires supplementary storage, which can be costly in terms of both space and overhead.

2.4. Key Observations

In order to design a novel data placement scheme for the hybrid caches, we make two key observations about the write intensity and re-fetch rate of the cache blocks. This subsection summarizes the key observations we made from our experiment.

2.4.1. Observation 1: Write Intensity Is Almost Constant

We observe that a cache block's write intensity (WI) is almost constant during the execution across several workloads. Figure 4 shows the distribution of cache blocks with constant write intensity. On average, write intensity remains constant for 98% of the cache blocks fetched from the main memory. Even though some benchmarks, such as soplex and leslie3d, have cache blocks with varying write intensity, the percentage of those blocks

is less than 12%. 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 during workload execution.

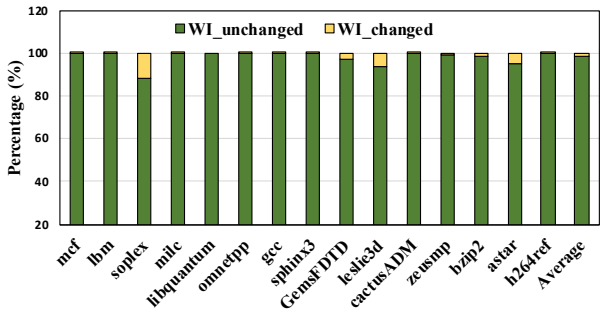


Figure 4. Ratio of cache blocks with constant write intensity (WI). On average, 98% of the cache blocks have the same write intensity during the execution of the workload.

2.4.2. Observation 2: Most Blocks Are Re-fetched after Eviction

Aside from the fact that the write intensity (WI) of a cache block remains constant during workload execution, we observe another characteristic of workloads on the reuse rate of the cache blocks. According to our experimental results, most cache blocks are re-fetched to LLC after being evicted from LLC. Figure 5 shows the distribution of the cache blocks that are re-fetched after an eviction. On average, 96% of the cache blocks are re-fetched from the main memory. This result implies that if we keep information about the write intensity of cache blocks in a specific storage element, we can use that information in the future to determine the write intensity of re-fetched blocks.

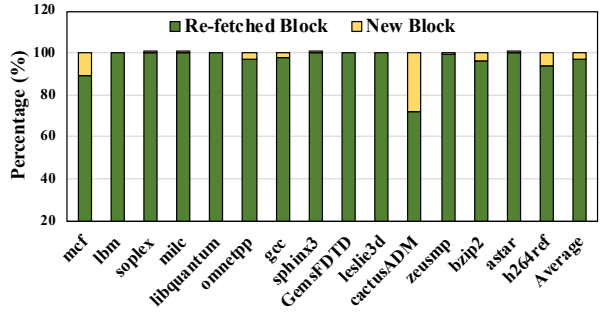


Figure 5. Ratio of cache blocks re-fetched from the main memory. On average, 96% of the cache blocks are fetched again from the main memory after they are evicted from the LLC cache.

3. ADAM: Adaptive Block Placement with Metadata Embedding

3.1. Overview

We propose ADAM, an adaptive block placement framework with metadata embedding in order to fully exploit the benefits of hybrid caches. Figure 6 shows an overall architecture of the hybrid cache with the ADAM framework. The ADAM framework consists of four components: per-block write counter, write-intensity detection unit, metadata embedding unit, and block placement unit. First, the tag array's write counter is used to keep track of the number of write operations performed on each cache block. Second, when cache blocks are evicted from the cache, the write-intensity detection unit determines their write intensity. Third, the metadata embedding unit stores or extracts the metadata of the cache blocks when writing or reading the blocks to/from the main memory. Finally, the block placement unit places the block appropriately in either the SRAM or STT-RAM based on the write intensity acquired by the metadata embedding unit when fetching the block from the main memory and storing it in the cache.

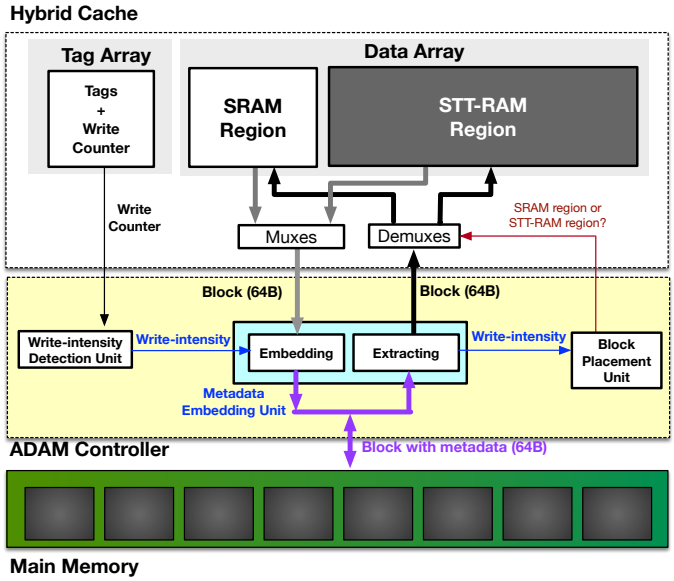


Figure 6. The ADAM framework consists of four components: write counters, write-intensity detection unit, metadata embedding unit, and block placement unit.

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

The metadata embedding unit attempts to extract metadata from the cache block during a read. If the cache block contains the metadata, the extracted metadata is passed to a block placement unit, which determines the best region for the block. If the extracted metadata indicates that the block is write-intensive, the block placement unit moves it to the SRAM region. If not, the block is placed in the SRAM region.

3.2. Per-Block Write Counter

A write counter, which comprises a 3-bit saturated counter, is used to track the write intensity of a cache block. The write counters are stored in the tag array, as shown in Figure 6. On a write hit, the write counter for a block increases by one. On the other hand, a read hit decrements the counter by one. By using a 3-bit saturated write counter per block, we can track down the write intensity (i.e., frequency of write operations) in its seven most recent accesses. Since the write intensity of a block is almost constant, as discussed in Section 2.4.1, tracking the recent access history is sufficient to determine the write intensity of the block.

3.3. Write-Intensity Detection Unit

The write-intensity detection unit probes the tags' 3-bit write counters. The write-intensity detection unit compares the victim block's write-counter value to a write-intensity threshold. If the value of the write-counter exceeds the threshold, the write-intensity detection unit generates 1-bit metadata indicating a high write-intensity. If the write-counter value is less than the threshold, it generates 1-bit metadata indicating a low write intensity.

3.4. Metadata Embedding Unit

The metadata embedding unit attempts to insert the 1-bit metadata into the cache block. Unfortunately, cache blocks are typically 64 bytes in size, and when placed in memory, there is no extra space to store metadata. As a result, in order to fit the metadata into a 64-byte block, the metadata embedding unit compresses 64-byte data to 61 bytes. Metadata is then stored in the 64-byte block's unused 3-byte space. Due to the fact that metadata is stored

within the data, this approach saves memory space and bandwidth. It does not necessitate additional memory space or memory bandwidth to transfer metadata between the main memory and on-chip caches.

The efficiency of metadata embedding is highly related to the ratio of blocks that can be compressed to a specific size. Fortunately, most blocks can be compressed to the target size because on-chip cache data has a high degree of redundancy as also demonstrated in numerous previous works [9–17], and the compression ratio required for metadata embedding is low. Figure 7 shows the percentage of the blocks (64 bytes) that can be compressed to less than 61 bytes for memory-intensive SPEC CPU2006 benchmarks. On average, 82% of the blocks can be compressed to less than 61 bytes. In this study, the metadata embedding unit compresses a cache block to at least 61 bytes using the Base-Delta-Immediate (BDI) [10] and Frequent-Pattern-Compression (FPC) [11] techniques and selects the best of the two techniques. BDI and FPC both compress a block to at least 61 bytes; BDI is the default choice.

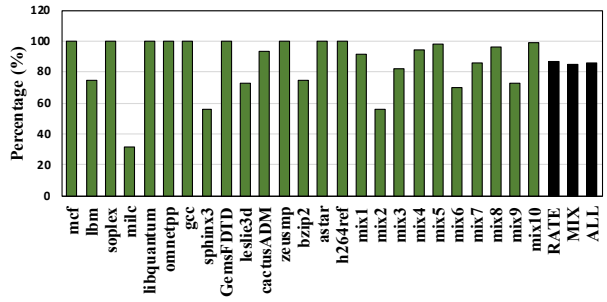


Figure 7. The Percentage of cache blocks (64 B) compressible to 61 B. On average, 82% of the blocks are compressible to 61 bytes.

As shown in Figure 8, the metadata embedding unit stores a 2-byte signature alongside the 61-byte compressed cache block. Similar to the Attach framework [9], the 2-byte signature consists of a 15-bit Compression ID (CID) and a 1-bit Exclusive ID (XID). CID helps to identify compressed cache blocks in the main memory. If CID matches a predefined 15-bit value, the corresponding block is identified as a compressed block. CID collision can occur if the high-order 15 bits of the uncompressed block are equal to the CID. XID helps to detect CID collisions and eliminates false positives. The high-order 16th bit of the uncompressed blocks for which their high-order 15 bits are identical to CID is replaced by XID (i.e., '0'). The original 16th bit is then stored in a separate memory region (around 0.2% of main memory space). On a CID match, the XID (i.e., 16th bit) is checked to detect a CID collision. If XID is 0, it is determined that a CID collision has occurred on an uncompressed block. Thus, each CID collision requires additional memory access to recover the original 16th data-bit that XID replaced. Fortunately, however, the probability of collisions is only $\frac{1}{2^{15}}$ because we use a 15-bit CID.

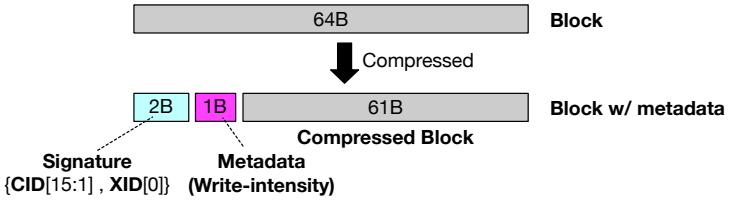


Figure 8. Embedding metadata into the cache block. The compressed block is 61 B in size, with a 2 B signature to indicate it is compressed and 1 B of metadata to indicate the compression technique and write intensity.

The metadata embedding unit then stores 1-byte of metadata alongside the signature, as shown in Figure 8. In our implementation, one bit of the 1-byte metadata is used to

specify the write intensity of the block, and another one bit is used to identify a compression technique (BDI or FPC) used to compress the block. The remaining six bits can be used to specify some other characteristics, such as the reuse distance or the latest hit count, of 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 data are not compressible, then data are stored as it is. However, as we described, if the first 15-bit of uncompressed data collide with CID, then the 16th bit (XID) is set to 0, and then the original 16th bit replaced by the XID is placed in a separate region within the main memory. As the CID collision rarely happens (only 0.003%), the additional accesses involved in obtaining the original 16th bit from the main memory have a negligible impact on performance and energy.

3.5. Block Placement Unit

On a read operation, the metadata embedding unit decompresses the cache block and extracts write-intensity metadata. The metadata embedding unit then forwards the write-intensity information to the block placement unit. If the cache block is deemed write intensive, the block placement unit places the block into the SRAM region. If the block is deemed non-write intensive, the block placement unit places it into the STT-RAM region.

3.6. ADAM Operations: Tying It Together

The flowchart in Figure 9 shows the set of operations that occur on a cache miss while using the ADAM framework. We discuss these operations in detail.

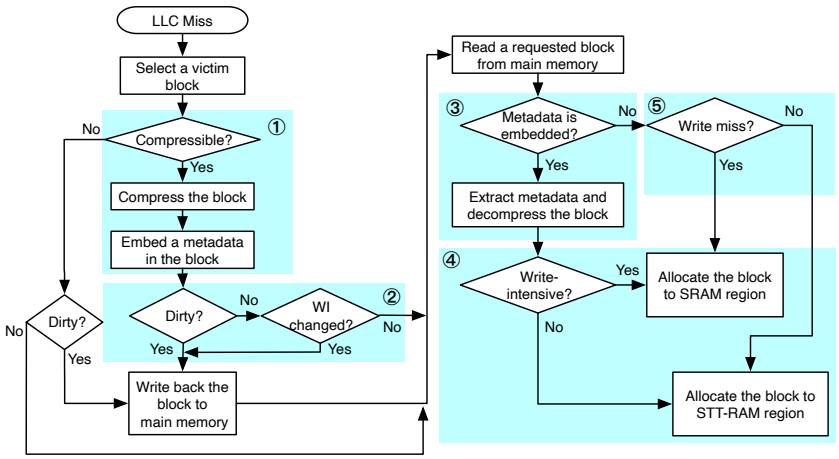


Figure 9. The flowchart detailing the high-level operations of the ADAM framework. These operations occur on an LLC miss. Overall, there are five key operations.

3.6.1. Embedding Metadata in Evicted Blocks

On a cache miss, the metadata embedding unit attempts to embed write-intensity metadata within the evicted block by compressing it to at least 61 bytes. This operation is denoted by ① in Figure 9. If the block cannot be compressed to less than 61 bytes, we cannot embed the metadata in the victim block. In that case, the block is written back to the main memory as it is without compression.

3.6.2. Selectively Writing Back Clean Blocks

An evicted block is deemed clean if it is not updated during its lifetime in the cache. The clean evicted blocks are not written back into the main memory in the conventional caches in order to save memory bandwidth. However, in the ADAM framework, the clean blocks will be written back to the main memory if their write intensity (WI) is updated. For instance, suppose a block with a high write intensity is read into the cache. When the block is installed in the cache, its 3-bit write counter is set to the maximum value (e.g.,

0×7). After that, the block may only be subjected to read operations during its time in the cache. Thus, the write-intensity counter is decremented to zero. Since the block remains clean throughout its lifetime in the cache, it is unnecessary to write back the block to memory during an eviction. However, as there is a change in the write intensity, from high write intensity to low write intensity, the ADAM framework will write back these clean evicted blocks with the updated metadata to the main memory.

We call the write requests for the clean evicted blocks as *Clean Writes (CW)* in this paper. If all clean evicted blocks are written back to the main memory, it will significantly increase the number of write requests to the main memory and can reduce performance. Therefore, in order to minimize the performance impact of the clean writes, ADAM writes back the clean block only if the write intensity of the block changes after it is installed in the LLC. This enables ADAM to keep track of the changes in write-intensity for cache blocks (as denoted by ②).

Fortunately, the write intensity of the cache blocks changes infrequently, as we discussed in Section 2.4.1. Thus, the impact of clean writes on the memory bandwidth is low. Figure 10 shows the breakdown of memory accesses on a memory system with ADAM framework. As shown in the figure, read requests account for 71% of the total memory accesses. Dirty writes, which are the write requests for dirty blocks, account for 25%, and the clean writes consume only 4% of the total memory accesses. Clean writes account for around 20% of total memory accesses in some benchmarks, such as omnetpp and cactusADM. For such workloads, ADAM is configured to disable the clean write. We will discuss the impact of clean writes on performance in Section 5.2.

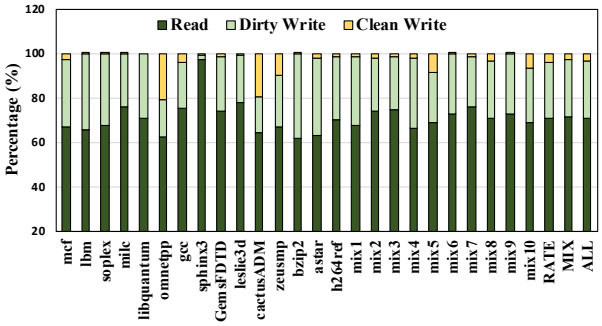


Figure 10. Breakdown of memory accesses with ADAM framework. Reads consume 71% of the accesses. Dirty writes consume 25% of the accesses. Clean writes consume only 4% of the accesses.

3.6.3. Extracting Metadata from Re-Fetched Blocks

As discussed in Section 2.4.2, most blocks that are fetched from the main memory are re-fetched blocks that are reloaded after they are evicted from the LLC. A re-fetched block might contain metadata if it was compressed at the time it was previously evicted from the LLC. As described in Section 3.4, a compressed block contains a 2-byte signature and 1-byte metadata as well as the actual data block compressed to 61 bytes. The metadata embedding unit compares the leftmost 2 bytes of the fetched block with a predefined signature to determine whether the block is compressed or not. If the block is compressed, the metadata embedding unit extracts 1-byte metadata, as shown in Figure 11. This scenario is called metadata hit. It then sends write intensity and compression algorithm information included in the metadata to the block placement unit and the decompressor, respectively. If the block is not compressed, the metadata embedding unit does not obtain any write-intensity information from the block. This scenario is called a metadata miss. The entire operation is denoted by ③ in Figure 9.

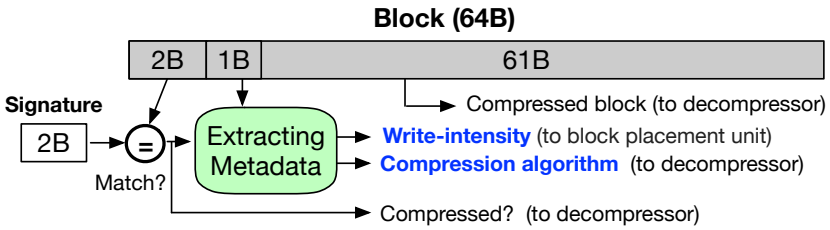


Figure 11. Extracting write-intensity metadata from a block. The metadata embedding unit decompresses the block and uses 1-byte (1B) metadata to identify the write intensity of the block.

3.6.4. Block Placement on Metadata Hit

When write-intensity metadata is found in a cache block (i.e., a metadata hit), the block placement unit allocates the block to either the SRAM region or the STT-RAM region. This allocation depends on the write-intensity information of the block (denoted by ④ in Figure 9). The write-intensity information for a block is learned when the corresponding block previously resided within the LLC. We observed that, even though the block placement decision is made based on the previous reference pattern of the block, the decision is mostly correct. This is because, as shown in Section 2.4.1, the write intensity of a block tends to remain almost constant. In the case where the write intensity of a block frequently changes, we can force the block to be allocated to the SRAM region or the STT-RAM region by storing preferable region information in the metadata when embedding metadata in the block.

3.6.5. Block Placement on Metadata Miss

The metadata embedding unit cannot extract any metadata from a cache block if it is not compressible. Moreover, if a cache block is loaded from the main memory for the first time, the block will not have any metadata. In the case where the fetched block does not have metadata (i.e., metadata miss), the block placement unit checks if the memory request was a read miss or a write miss. If the block is fetched from the main memory due to a read miss, the block is allocated to the STT-RAM region by assuming the block as a non-write-intensive block. Otherwise, the block is allocated to the SRAM region. We note that this simple block placement policy is also used in prior work [4]. The block placement on metadata miss is denoted by ⑤ in Figure 9.

4. Evaluation Methodology

In order to evaluate the performance benefits of ADAM, we developed a hybrid-cache simulator based on USIMM [18]. The simulator models the out-of-order processor core, a detailed cache hierarchy including the hybrid LLC and main memory. Table 2 lists the simulated system configuration. The LLC is configured to have multiple banks to service multiple requests in parallel. STT-RAM and SRAM parameters are obtained using NVSim [1]. Since ADAM employs low-latency compression techniques (i.e., BDI and FPC) specifically designed for the on-chip caches, we assume that decompression takes a single clock cycle, as performed in many prior studies [9–11,15–17].

Table 2. Baseline System Configuration.

Processor	3.2 GHz, 4 cores, out of order 8-width issue/decode, 160 entry reorder buffer
L1 Cache L2 Cache	32 KB, 8-Way, 64 B lines, shared, 4 cycles 256 KB, 8-Way, 64 B lines, shared, 12 cycles
LLC (Hybrid Cache)	12 MB (SRAM: 4 MB, STT-RAM: 8 MB), 16-Way 64 B lines, shared, 16 banks, SRRIP Tag access latency: 5 cycles Data access latency (SRAM): 30 cycles Data access latency (STT-RAM Read): 30 cycles Data access latency (STT-RAM Write): 90 cycles
Main Memory	1600 MHz (DDR4 3200 MHz) Channels: 2, Ranks per a channel: 1 Bank groups per a rank: 4 Banks per a bank group: 4

For evaluations, we chose memory-intensive benchmarks, which have greater than 1 MPKI (LLC Misses Per Kilo instructions), from SPEC CPU2006. We warm up LLC for 2 billion instructions and execute 1 billion instructions. We execute all benchmarks in rate mode where all cores run the same benchmark. As shown in Table 3, we also made ten 4-threaded mixed workloads by randomly selecting one benchmark from three categories (low MPKI, medium MPKI, and high MPKI).

Table 3. Workload Mixes.

mix1	bzip2, libquantum, astar, cactusADM
mix2	lbm, omnetpp, mcf, GemsFDTD
mix3	h264ref, milc, bwaves, gcc
mix4	sphinx3, astar, soplex, zeusmp
mix5	bzip2, mcf, dealII, gcc
mix6	omnetpp, bwaves, GemsFDTD, cactusADM
mix7	libquantum, milc, sphinx3, lbm
mix8	leslie3d, xalancbmk, h264ref, astar
mix9	soplex, GemsFDTD, cactusADM, dealII
mix10	lbm, sphinx3, leslie3d, h264ref

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.

5. Simulation Results

5.1. Write Hits on SRAM and STT-RAM Banks

Figure 12 shows the distribution of write hits on the LLC. The primary goal of the block placement scheme for the hybrid cache is to reduce write hits on the STT-RAM region in order to mitigate long write latency and high write energy of the STT-RAM. As shown in Figure 12, ADAM yields low write hits on the STT-RAM region across all benchmarks compared to the baseline scheme. The simulation results show that ADAM reduces almost all of write hits on the STT-RAM region for libquantum. On average, the percentage of write hits on the STT-RAM region is reduced from 81% to 25%. This is close to our initial goal to reduce write hits on the STT-RAM region to nearly 20%.

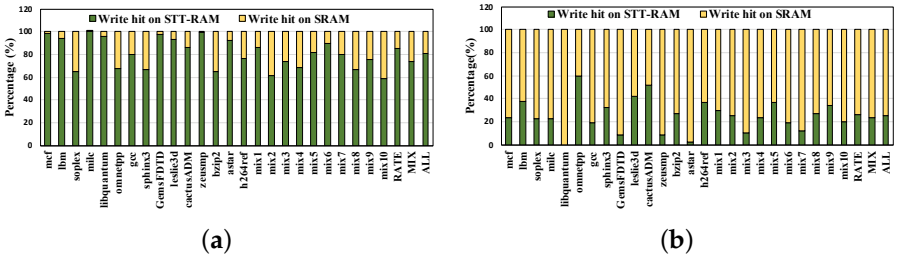


Figure 12. Distribution of write hits on hybrid caches. The ADAM framework reduces the percentage of write hits on the STT-RAM region from 81% (baseline hybrid cache) to 25% (hybrid cache with ADAM). (**a**) Baseline hybrid cache; (**b**) Hybrid cache with ADAM.

5.2. Performance

Figure 13 shows the speedup of ADAM when compared to a baseline block placement and ideal block placement. ADAM improves performance by 24% on average. Ideally, if we allocate all write-intensive blocks to the SRAM region, we obtain a speedup of 40% on average. Performance results show that libquantum and astar benefit the most from ADAM due to dramatic reductions in the write hits on STT-RAM. They achieved a 44% and 65% of performance improvement, respectively.

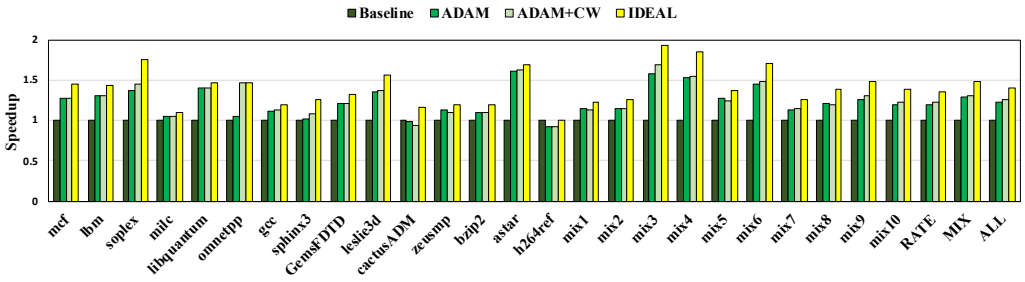


Figure 13. The performance improvement of ADAM over baseline block placement. On average, ADAM and ADAM-CW provide 24% and 26% speedup as compared to the baseline. The ideal block placement provides a speedup of 40%.

Our analysis shows that the Clean Write (CW) scheme can improve performance further 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 CW scheme. With the CW 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 CW scheme is applied.

Most benchmarks can benefit from accurate block placement with ADAM. However, ADAM performs worse than the baseline scheme for some benchmarks, such as cactusADM and h264ref. The increased misses on LLC cause performance degradation for these benchmarks. In hybrid caches, the SRAM region is smaller than the STT-RAM region; therefore, when many cache blocks are allocated to the SRAM region, the LLC miss rate will increase. In order to address this problem, we can extend ADAM to take into account the pressure on the SRAM region as well as the write intensity of the block.

5.3. Energy Consumption

Figure 14 compares the energy consumption of the hybrid cache with and without ADAM. When compared to the baseline block placement, ADAM reduces the energy consumption of the hybrid cache by 35% on average. This significant saving in energy consumption is mainly due to reduced write energy. As shown in Table 1, the write operation consumes significantly more power than the read operation. Thus, frequent

write hits on the STT-RAM region increase the total energy consumption of the hybrid cache. As we discussed in Section 5.1, ADAM yields significantly fewer write hits on the STT-RAM region than the baseline placement policy, resulting in much lower dynamic energy consumption. The energy consumption results show that lbm and libquantum benefit the most from ADAM because the baseline placement policy results in frequent write hits on the STT-RAM region for these benchmarks. They achieve 56% and 59% of reductions in the total energy consumption of the hybrid cache, respectively.

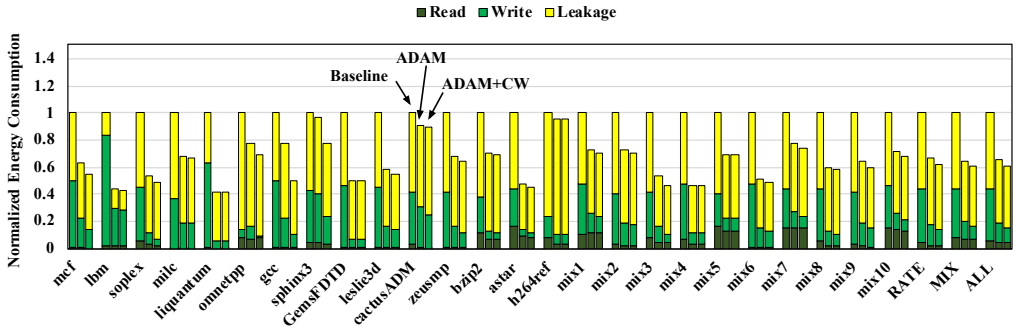


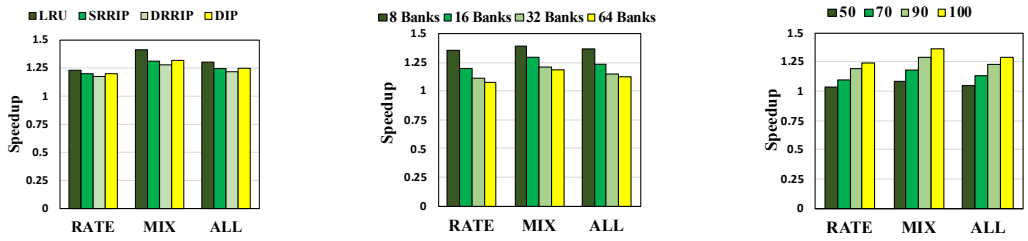
Figure 14. Energy consumption of ADAM over baseline block placement. On average, ADAM and ADAM-CW reduce energy consumption of the hybrid cache by 35% and 39%, as compared to the baseline.

The energy results also show that the Clean Write (CW) scheme further reduces energy consumption, especially for the benchmarks such as gcc, sphinx3, and omnetpp, where preserving write-intensity information for clean victim blocks is necessary. On average, ADAM saves 4% more energy with the CW scheme.

5.4. Sensitivity Analysis

5.4.1. Sensitivity to LLC Replacement Policy

Figure 15a shows the speedup of ADAM over the baseline for four different cache replacement policies. Overall, ADAM achieves higher performance compared to the baseline regardless of the replacement policy. The performance gain with ADAM is high, especially for replacement policies that yield a higher LLC miss rate. Such policies (such as LRU) enable frequent block installations and thereby enable these blocks to be placed efficiently. On average, ADAM achieves the speedup of 31%, 25%, 24%, and 25% over the baseline while using LRU, SRRIP, DRRIP, and DIP policies, respectively.



(**a**) LLC Replacement Policy (**b**) The Number of LLC Banks (**c**) Write Latency of STT-RAM

Figure 15. Sensitivity to various parameters of a hybrid cache. (**a**) For ADAM, the LRU policy provides the most speedup as it evicts most of the blocks and enables them to learn about write intensity. (**b**) As the number of banks increases, speedup is reduced due to bank-level parallelism. (**c**) Speedup increases as the write latency of STT-RAM increases.

5.4.2. Sensitivity to LLC Bank Count

Figure 15b shows the speedup of ADAM for different numbers of banks. ADAM delivers higher speedup for the LLC with smaller bank counts; it achieves a speedup of

12% and 37% for 64-bank and 8-bank configurations, respectively. Even if increasing bank counts helps mitigate the long write latency of the STT-RAM, it will increase the chip area and dynamic power consumption of LLC. ADAM shows better performance than the baseline for all bank counts.

5.4.3. Sensitivity to Write Latency of STT-RAM

Figure 15c shows the impact of write latency of the STT-RAM on the effectiveness of ADAM. As the write latency of STT-RAM increases, ADAM achieves a higher speedup over the baseline. Regardless of the write latency of STT-RAM, ADAM provides better performance over the baseline scheme.

5.5. Hardware Overhead Analysis

The majority of hardware overhead of ADAM comes from the metadata embedding unit and the per-block write counter. The metadata embedding unit comprises compressor and decompressor, which take roughly 290K NAND2 gates (according to [19]) and only consumes 0.016 mm^2 die area with 22 nm technology (0.2% of the hybrid cache). Employing a write counter per a block in the tag array increases the die area of the hybrid cache by 7.6%. The die area of the hybrid cache with the per-block write counter is calculated by using the NVSim. The total area overhead of the ADAM is less than 8% of the hybrid LLC size. This area overhead of ADAM would be trivial when considering its significant gains in performance and energy consumption.

6. Related Work

Many prior works have proposed adaptive block placement schemes for hybrid caches [3–7,20]. In [6], memory access patterns are exploited for block placement and migration in a hybrid LLC. Chen et al. [7] proposed combining static and dynamic schemes in order to optimize block placement in the hybrid cache. In [3], a counter-based approach was proposed for predicting write-intensive blocks. Jadidi et al. [5] proposed a technique to reduce write variance between STT-RAM lines by migrating frequently written cache blocks to other STT-RAM or SRAM lines. None of the prior studies have considered storing the write-intensity of individual blocks as performed in ADAM.

Dynamic LLC Bypassing can be a good solution for mitigating the long write latency of STT-RAM [21–28]. Wang et al. [21] defined an interesting characteristic called LLC-obstruction, which can occur by a write-intensive process, and used it for dynamic LLC bypassing. In [22], an LLC congestion-aware bypassing technique is proposed to eliminate a large fraction of writes. Cheng et al. [23] introduced the concept of loop-block and proposed a loop-block-aware replacement policy to keep the loop-block in the LLC. Ahn et al. [24] defined dead write, which is the data written on LLC and not re-referenced during the lifetime of the corresponding cache block. By detecting dead writes and bypassing them from LLC, system performance and energy efficiency can be improved. Moreover, there are bypassing methods for different inclusion techniques, exclusive [26,27] and inclusive [25,29]. Gupta et al. [25] proposed a bypass buffer, which helps maintain the inclusive property when bypassing LLC in an inclusive cache system. When a decision is made to bypass the cache block, it is allocated to the bypass buffer instead of LLC. If the memory request misses LLC and hits the bypass buffer, the bypass buffer provides the requested block to the LLC, and the block is de-allocated from the bypass buffer and migrated to LLC. They also suggested a dataless bypass buffer, which only installs the tag of bypassed cache block into the bypass buffer to reduce hardware overhead. ADAM is orthogonal to these LLC bypassing techniques; therefore, they can be synergistically combined to unlock the performance of hybrid caches.

Several prior studies tried to enhance the performance of STT-RAM in order to use it for building a large cache [2,30–37]. In [30–32], and the retention time of STT-RAM is reduced to mitigate the long write latency of STT-RAM. Clinton et al. [30] and Adwait et al. [31] proposed hybrid architecture, which includes SRAM-based L1 cache with volatile

STT-RAM-based L2 cache or L3 cache. Zhenyu et al. [32] suggested STT-RAM-based L1 cache by exploiting STT-RAM cells with various data retention time. Hameed et al. [33] proposed a selective read policy for STT-RAM to reduce energy consumption. Chi et al. [2] introduced state-of-the-art architectural approaches to adopt STT-RAM in the cache. Kuan et al. [34,35] proposed an STT-RAM-based cache that allows LLC configurations and retention time to be adapted to applications' runtime execution requirements.

Several prior works have proposed low-cost compression techniques [10,11,38]. As these compression techniques have low decompression latency and low implementation cost, they have been used to improve the effective capacity, energy efficiency, and bandwidth of the memory systems [9,12–17]. ADAM employs BDI [10] and FPC [11] as compression techniques and to obtain the idea of metadata embedding from [9]. To our best knowledge, this study is first to exploit the compression technique for adaptive block placement in the hybrid caches.

7. Conclusions

Static Random Access Memory (SRAM), the conventional memory technology for the last-level caches, has difficulty in scaling due to its high power consumption and low density. Spin-Transfer Torque RAM (STT-RAM) has emerged as a substitute for SRAM. It offers higher density and lower leakage power consumption over SRAM. However, STT-RAM has long latency and high power consumption on write operations. Therefore, a hybrid cache, which integrates both SRAM and STT-RAM, has been proposed to employ the strengths of two different memory technologies. Since the long write latency of STT-RAM can significantly reduce system performance, we need an accurate block placement scheme to allocate write-intensive cache blocks on the STT-RAM region.

This paper proposed ADAM, a new adaptive block placement framework with metadata embedding for hybrid caches. ADAM maintains write-intensity information of an individual block by embedding it within the cache block. When evicting a cache block from the LLC, ADAM embeds the block's metadata (i.e., write-intensity information) within a block by compressing it to make room for metadata. When a cache block is brought from the main memory into the hybrid cache, ADAM extracts the embedded metadata and utilizes it to determine the write intensity of the block. With extracted write-intensity information, ADAM allocates a cache block into the appropriate data region. ADAM provides an efficient framework for hybrid cache management by maintaining metadata without additional storage elements.

Author Contributions: Conceptualization, B.K. and S.H.; investigation, B.K., P.N., and S.H.; data curation, B.K., Y.K., and S.H; methodology, P.N. and S.H.; writing—original draft preparation, B.K. and S.H.; writing–review and editing, Y.K., P.N., and S.H.; project administration, S.H. All authors have read and agreed to published version of the manuscript.

Funding: This study was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea Government (MSIT) (NRF-2019R1G1A1011403).

Acknowledgments: The authors would like to acknowledge the participants in the study.

Conflicts of Interest: The authors declare no conflicts of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

References

1. Dong, X.; Xu, C.; Xie, Y.; Jouppi, N.P. NVSim: A Circuit-Level Performance, Energy, and Area Model for Emerging Nonvolatile Memory. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2012**, *31*, 994–1007. [CrossRef]
2. Chi, P.; Li, S.; Cheng, Y.; Lu, Y.; Kang, S.H.; Xie, Y. Architecture design with STT-RAM: Opportunities and challenges. In Proceedings of the 2016 21st Asia and South Pacific Design Automation Conference, Macao, China, 25–28 January 2016; pp. 109–114.
3. Sun, G.; Dong, X.; Xie, Y.; Li, J.; Chen, Y. A novel architecture of the 3D stacked MRAM L2 cache for CMPs. In Proceedings of the 15th International Symposium on High Performance Computer Architecture, Raleigh, NC, USA, 14–18 February 2009; pp. 239–249.
4. Wu, X.; Li, J.; Zhang, L.; Speight, E.; Xie, Y. Power and performance of read-write aware hybrid caches with non-volatile memories. In Proceedings of the 2009 Design, Automation Test in Europe Conference Exhibition, Nice, France, 20–24 April 2009; pp. 737–742.

5. Jadidi, A.; Arjomand, M.; Sarbazi-Azad, H. High-endurance and performance-efficient design of hybrid cache architectures through adaptive line replacement. In Proceedings of the IEEE/ACM International Symposium on Low Power Electronics and Design, Fukuoka, Japan, 1–3 August 2011; pp. 79–84.
6. Wang, Z.; Jiménez, D.A.; Xu, C.; Sun, G.; Xie, Y. Adaptive placement and migration policy for an STT-RAM-based hybrid cache. In Proceedings of the 2014 IEEE 20th International Symposium on High Performance Computer Architecture, Orlando, FL, USA, 15–19 February 2014; pp. 13–24.
7. Chen, Y.; Cong, J.; Huang, H.; Liu, B.; Liu, C.; Potkonjak, M.; Reinman, G. Dynamically reconfigurable hybrid cache: An energy-efficient last-level cache design. In Proceedings of the 2012 Design, Automation Test in Europe Conference Exhibition, Dresden, Germany, 12–16 March 2012; pp. 45–50.
8. Ahn, J.; Yoo, S.; Choi, K. Write intensity prediction for energy-efficient non-volatile caches. In Proceedings of the International Symposium on Low Power Electronics and Design, Beijing, China, 4–6 September 2013; pp. 223–228.
9. Hong, S.; Nair, P.J.; Abali, B.; Buyuktosunoglu, A.; Kim, K.; Healy, M. Attaché: Towards Ideal Memory Compression by Mitigating Metadata Bandwidth Overheads. In Proceedings of the 2018 51st Annual IEEE/ACM International Symposium on Microarchitecture, Fukuoka, Japan, 20–24 October 2018; pp. 326–338.
10. Pekhimenko, G.; Seshadri, V.; Mutlu, O.; Kozuch, M.A.; Gibbons, P.B.; Mowry, T.C. Base-delta-immediate compression: Practical data compression for on-chip caches. In Proceedings of the 2012 21st International Conference on Parallel Architectures and Compilation Techniques, Minneapolis, MN, USA, 19–23 September 2012; pp. 377–388.
11. Alameldeen, A.R.; Wood, D.A. Frequent pattern compression: A significance-based compression scheme for L2 caches. *Comput. Sci.* **2004**. Available online: https://www.semanticscholar.org/paper/Frequent-Pattern-Compression%3A-A-Significance-Based-Alameldeen-Wood/e7c6f67a70b5cf0842a7a2fc497131a79b6ee2c5 (accessed on 6 January 2022).
12. Alameldeen, A.R.; Wood, D.A. Adaptive cache compression for high-performance processors. In Proceedings of the 31st Annual International Symposium on Computer Architecture, Munich, Germany, 19–23 June 2004; pp. 212–223.
13. Kim, S.; Kim, J.; Lee, J.; Hong, S. Residue cache: A low-energy low-area L2 cache architecture via compression and partial hits. In Proceedings of the 2011 44th Annual IEEE/ACM International Symposium on Microarchitecture, Porto Alegre, Brazil, 3–7 December 2011; pp. 420–429.
14. Sardashti, S.; Seznec, A.; Wood, D.A. Skewed Compressed Caches. In Proceedings of the 2014 47th Annual IEEE/ACM International Symposium on Microarchitecture, Cambridge, UK, 13–17 December 2014; pp. 331–342.
15. Hong, S.; Abali, B.; Buyuktosunoglu, A.; Healy, M.B.; Nair, P.J. Touché: Towards Ideal and Efficient Cache Compression By Mitigating Tag Area Overheads. In Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, Columbus, OH, USA, 12–16 October 2019; pp. 453–465.
16. Pekhimenko, G.; Seshadri, V.; Kim, Y.; Xin, H.; Mutlu, O.; Gibbons, P.B.; Kozuch, M.A., Mowry, T.C. Linearly Compressed Pages: A Low-complexity, Low-latency Main Memory Compression Framework. In Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture, New York, NY, USA, 7–11 December 2013; pp. 172–184.
17. Choukse, E.; Erez, M.; Alameldeen, A.R. Compresso: Pragmatic Main Memory Compression. In Proceedings of the 2018 51st Annual IEEE/ACM International Symposium on Microarchitecture, Fukuoka, Japan, 20–24 October 2018; pp. 546–558.
18. Chatterjee, N.; Balasubramonian, R.; Shevgoor, M.; Pugsley, S.H.; Udipi, A.N.; Shafiee, A.; Sudan, K.; Awasthi, M.; Chishti, Z. USIMM: The Utah SImulated Memory Module A Simulation Infrastructure for the JWAC Memory Scheduling Championship. 2012. Available online: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.387.8483 (accessed on 6 January 2022).
19. Chen, X.; Yang, L.; Dick, R.P.; Shang, L.; Lekatsas, H. C-Pack: A High-Performance Microprocessor Cache Compression Algorithm. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2010**, *18*, 1196–1208. [CrossRef]
20. Kim, B.; Nair, P.J.; Hong, S. ADAM: Adaptive Block Placement with Metadata Embedding for Hybrid Caches. In Proceedings of the 2020 IEEE 38th International Conference on Computer Design, Hartford, CT, USA, 18–21 October 2020; pp. 421–424.
21. Wang, J.; Dong, X.; Xie, Y. OAP: An obstruction-aware cache management policy for STT-RAM last-level caches. In Proceedings of the Design, Automation Test in Europe Conference Exhibition, Grenoble, France, 18–22 March 2013; pp. 847–852.
22. Korgaonkar, K.; Bhati, I.; Liu, H.; Gaur, J.; Manipatruni, S.; Subramoney, S.; Karnik, T.; Swanson, S.; Young, I.; Wang, H. Density Tradeoffs of Non-Volatile Memory as a Replacement for SRAM Based Last Level Cache. In Proceedings of the 45th Annual International Symposium on Computer Architecture, Los Angeles, CA, USA, 1–6 June 2018; pp. 315–327.
23. Cheng, H.; Zhao, J.; Sampson, J.; Irwin, M.J.; Jaleel, A.; Lu, Y.; Xie, Y. LAP: Loop-Block Aware Inclusion Properties for Energy-Efficient Asymmetric Last Level Caches. In Proceedings of the 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture, Seoul, Korea, 18–22 June 2016; pp. 103–114.
24. Ahn, J.; Yoo, S.; Choi, K. DASCA: Dead Write Prediction Assisted STT-RAM Cache Architecture. In Proceedings of the 2014 IEEE 20th International Symposium on High Performance Computer Architecture, Orlando, FL, USA, 15–19 February 2014; pp. 25–36.
25. Gupta, S.; Gao, H.; Zhou, H. Adaptive Cache Bypassing for Inclusive Last Level Caches. In Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing, Boston, MA, USA, 20–24 May 2013; pp. 1243–1253.
26. Gaur, J.; Chaudhuri, M.; Subramoney, S. Bypass and insertion algorithms for exclusive last-level caches. In Proceedings of the 2011 38th Annual International Symposium on Computer Architecture, San Jose, CA, USA, 4–8 June 2011; pp. 81–92.
27. Chaudhuri, M.; Gaur, J.; Bashyam, N.; Subramoney, S.; Nuzman, J. Introducing Hierarchy-awareness in replacement and bypass algorithms for last-level caches. In Proceedings of the 2012 21st International Conference on Parallel Architectures and Compilation Techniques, Minneapolis, MN, USA, 19–23 September 2012; pp. 293–304.

28. Li, L.; Tong, D.; Xie, Z.; Lu, J.; Cheng, X. Optimal bypass monitor for high performance last-level caches. In Proceedings of the 2012 21st International Conference on Parallel Architectures and Compilation Techniques, Minneapolis, MN, USA, 19–23 September 2012; pp. 315–324.
29. Kim , M.; Choi, J.; Kwak, J.; Jhang, S.; Jhon, C. Bypassing method for STT-RAM based inclusive last-level cache. In Proceedings of the 2015 Conference on Research in Adaptive and Convergent Systems, Prague, Cezh Republic, 9–12 October 2015; pp. 424–429.
30. Smullen, C.W.; Mohan, V.; Nigam, A.; Gurumurthi, S.; Stan, M.R. Relaxing non-volatility for fast and energy-efficient STT-RAM caches. In Proceedings of the 2011 IEEE 17th International Symposium on High Performance Computer Architecture, San Antonio, TX, USA, 12–16 February 2011; pp. 50–61.
31. Jog, A.; Mishra, A.; Xu, C.; Xie, Y.; Narayanan, V.; Iyer, R.; Das, C. Cache Revive: Architecting Volatile STT-RAM Caches for Enhanced Performance in CMPs. In Proceedings of the 2012 49th ACM/EDAC/IEEE Design Automation Conference, San Francisco, CA, USA, 3–7 June 2012.
32. Sun, Z.; Bi, X.; Li, H.; Wong, W.F.; Ong, Z.L.; Zhu, X.; Wu, W. Multi retention level STT-RAM cache designs with a dynamic refresh scheme. In Proceedings of the 2011 44th Annual IEEE/ACM International Symposium on Microarchitecture, Porto Alegre, Brazil, 4–7 December 2011; pp. 329–338.
33. Hameed, F.; Khan, A.A.; Castrillón, J. Performance and Energy-Efficient Design of STT-RAM Last-Level Cache. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2018**, *26*, 1–14. [CrossRef]
34. Kuan, K.; Adegbija, T. HALLS: An Energy-Efficient Highly Adaptable Last Level STT-RAM Cache for Multicore Systems. *IEEE Trans. Comput.* **2019**, *68*, 1623–1634. [CrossRef]
35. Kuan, K.; Adegbija, T. Energy-Efficient Runtime Adaptable L1 STT-RAM Cache Design. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2020**, *39*, 1328–1339. [CrossRef]
36. Hong, S.; Lee, J.; Kim, S. Ternary cache: Three-valued MLC STT-RAM caches. In Proceedings of the 2014 IEEE 32nd International Conference on Computer Design, Seoul, Korea, 19–22 October 2014; pp. 83–89.
37. Hong, S. Dead Block-Aware Adaptive Write Scheme for MLC STT-MRAM Caches. *J. Korea Soc. Comput. Inf.* **2020**, *25*, 1–9.
38. Kim, J.; Sullivan, M.; Choukse, E.; Erez, M. Bit-Plane Compression: Transforming Data for Better Compression in Many-Core Architectures. In Proceedings of the 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture, Seoul, Korea, 18–22 June 2016; pp. 329–340.