Prediction Hybrid Cache: An Energy-Efficient STT-RAM Cache Architecture

Junwhan Ahn, *Student Member, IEEE*, Sungjoo Yoo, *Member, IEEE*, and Kiyoung Choi, *Senior Member, IEEE*

Abstract—Spin-transfer torque RAM (STT-RAM) has emerged as an energy-efficient and high-density alternative to SRAM for large on-chip caches. However, its high write energy has been considered as a serious drawback. Hybrid caches mitigate this problem by incorporating a small SRAM cache for write-intensive data along with an STT-RAM cache. In such architectures, choosing cache blocks to be placed into the SRAM cache is the key to their energy efficiency. This paper proposes a new hybrid cache architecture called prediction hybrid cache. The key idea is to predict write intensity of cache blocks at the time of cache misses and determine block placement based on the prediction. We design a write intensity predictor that realize the idea by exploiting a correlation between write intensity of blocks and memory access instructions that incur cache misses of those blocks. It includes a mechanism to dynamically adapt the predictor to application characteristics. We also design a hybrid cache architecture in which write-intensive blocks identified by the predictor are placed into the SRAM region. Evaluations show that our scheme reduces energy consumption of hybrid caches by 28 percent (31 percent) on average compared to the existing hybrid cache architecture in a single-core (quad-core) system.

Index Terms—Cache memories, hybrid caches, non-volatile memories, prediction, STT-RAM

1 Introduction

Recently, non-volatile memory technologies have emerged as alternatives to SRAM and DRAM due to their inherent low-power characteristics and better scalability. In general, they have much lower static power consumption while having comparable performance to conventional memory technologies. To exploit these benefits, researchers have explored the possibility of using them as on-chip caches and/or main memory and have shown their potential for energy efficiency and scalability.

However, one major downside of non-volatile memory technologies is their inefficient write operations in terms of both latency and energy consumption. On top of that, some of them wear out during write operations thereby limiting the maximum number of write operations that can be performed for each cell. Due to these reasons, many researchers have tried to mitigate the adverse effect of write operations.

Utilizing SRAM/STT-RAM hybrid caches [1], [2], [3], [4], [5], [6], [7] is one of the most common approaches to compensate the effect of inefficient write operations of non-volatile memory. Since write energy of SRAM is much smaller than that of typical non-volatile memory, it helps to reduce write activities in non-volatile memory thereby reducing dynamic energy consumption. In that approach, cache ways are partitioned into SRAM and STT-RAM regions and frequently written blocks, which we call *write-intensive* blocks, are allocated to the SRAM region (instead of the STT-RAM region) to reduce write activity of the STT-RAM region.

In such architectures, their efficiency highly depends on how accurately we can identify write-intensive blocks. If too many blocks are chosen to be allocated to the SRAM region, frequent evictions from the SRAM region will degrade hit rate and thus will hurt overall performance. On the other hand, if some write-intensive blocks are allocated to the STT-RAM region, large energy penalty would be incurred due to high write energy of STT-RAM.

Despite such importance of this problem, solutions that have been proposed by previous work are either too simple or too static. More specifically, many researchers proposed to determine the region where to place a block based only on the type of operations (read/write) [1], [2], [6], [7], which is not good enough to identify write-intensive blocks accurately. Recent studies proposed compilation techniques to identify write-intensive data at compilation time and to provide the information as a hint to the runtime system [8], [9], [10]. One drawback of these approaches is that applications need to be recompiled for each architecture, which is not possible in many cases. Moreover, it is inherently impossible for compiler-based approaches to take dynamic characteristics of applications into account. Although they showed the efficiency of their techniques on single-level cache hierarchy [9] or write-through caches [10], their static nature makes them very hard to be adopted into writeback caches since the number of writebacks on each block is difficult to be estimated at compilation time. This could be a serious

- *J. Ahn and K. Choi are with the Department of Electrical and Computer Engineering, Seoul National University, Seoul, Republic of Korea. E-mail: {junwhan, kchoi}@snu.ac.kr.*
- *S. Yoo is with the Department of Computer Science and Engineering, Seoul National University, Seoul, Republic of Korea. E-mail: sungjoo.yoo@gmail.com.*

Manuscript received 12 Feb. 2014; revised 1 Mar. 2015; accepted 30 Apr. 2015. Date of publication 19 May 2015; date of current version 10 Feb. 2016. Recommended for acceptance by R. F. DeMara.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TC.2015.2435772

limitation for caches based on non-volatile memory that prefer the writeback policy.[1]

This paper proposes a novel mechanism called *write intensity predictor* to predict write intensity of each cache block dynamically. It is based on the key observation that there is a high correlation between write intensity of blocks and addresses of load/store instructions that incur misses of the blocks. The predictor exploits this observation by keeping track of instructions that tend to load write-intensive blocks and utilizing that information to predict write intensity of blocks that will be accessed in the future. This enables dynamic prediction of write intensity even for data blocks accessed for the first time, which is not possible with simple per-block counters.

Based on this concept, this paper proposes a new hybrid cache architecture called *prediction hybrid cache*. In this architecture, block placement is determined by prediction information from the write intensity predictor. Moreover, it periodically monitors application characteristics and adjusts the threshold of the write intensity predictor for better adaptivity.

This paper is an extension of our previous work [11]. The extension includes an improved cost model of write intensity (Section 2.1), area overhead reduction through dynamic set sampling (Section 3.3), and a new hardware structure for dynamic threshold adjustment (Section 4.4). This paper also provides comprehensive evaluations including sensitivity analysis and multicore evaluations.

2 Problem and Motivation

2.1 Problem Definition

The objective of this work is to predict write intensity of cache blocks on their misses. To solve this problem, we first introduce a new cost model for each block of hybrid caches. The idea is to formulate energy cost of placing a block into STT-RAM instead of SRAM. Formally, the *cost* of a block is defined as

$$\begin{aligned} c &= N_{\mathrm{r}}\Delta E_{\mathrm{r}} + N_{\mathrm{w}}\Delta E_{\mathrm{w}} \\ &= N_{\mathrm{r}} \times \left(E_{\mathrm{r}}^{\mathrm{STT}} - E_{\mathrm{r}}^{\mathrm{S}}\right) + N_{\mathrm{w}} \times \left(E_{\mathrm{w}}^{\mathrm{STT}} - E_{\mathrm{w}}^{\mathrm{S}}\right), \end{aligned} \tag{1}$$

where N_{r} and N_{w} denote the number of reads and writes to the block until its eviction, $E_{\mathrm{r}}^{\mathrm{STT}}$ and $E_{\mathrm{w}}^{\mathrm{STT}}$ denote normalized read and write energy of the STT-RAM region, and $E_{\mathrm{r}}^{\mathrm{S}}$ and $E_{\mathrm{w}}^{\mathrm{S}}$ denote those of the SRAM region.[2] Our cost model includes only dynamic energy consumption since static energy consumption is dependent on system performance, which will be optimized by a separate mechanism (see Section 4.4).

Based on this model, we define *write-intensive* blocks as cache blocks whose costs are higher than or equal to κ where κ is the write intensity threshold. Note that, since the cost of a block is defined with the number of reads and writes *until the time of its eviction*, it is impossible to perfectly determine the write intensity of a block on its miss (i.e., when the block is first introduced into the cache). In addition, κ needs to be chosen carefully so as to allocate neither too many nor too few blocks into the SRAM region. This paper tackles these two problems through architectural solutions.

1. Note that write-through caches request too many writes to deeper-level caches thereby increasing write energy significantly.

2. For SRAM/STT-RAM hybrid caches, ΔE_{r} is usually negative since higher density of STT-RAM makes interconnects shorter and thus reduces energy dissipated from wires [12], [13], while ΔE_{w} is positive. However, our formulation is not limited to such a specific case and can model other memory technologies with different ΔE_{r} and ΔE_{w}, which could be evaluated in our future work.

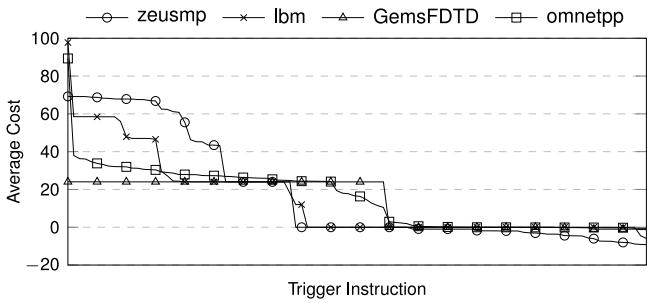


Fig. 1. Distribution of the average cost per block according to trigger instructions in four applications. Markers are drawn every five points for better visibility.

2.2 Motivation

Our key idea is that write intensity of blocks can be inferred from the load/store instructions that load the blocks into the cache. In other words, there is a strong correlation between costs of blocks and the instructions that incur cache misses of the blocks. We call such instructions *trigger instructions* and focus on their potential to be a good proxy for *predicting* write intensity of blocks.

Fig. 1 shows an experimental evidence of this intuition obtained from four SPEC CPU2006 benchmark applications under a two-level cache hierarchy (details of the simulation configuration are given in Section 5.1). In this figure, the x-axis shows the trigger instructions sorted by their y-values, while the y-axis shows the average cost per L2 cache block loaded by each of the trigger instructions. We set ΔE_{r} and ΔE_{w} of our cost model to -1 and 24, respectively. For better visibility, the figure shows only the top 100 trigger instructions in terms of the number of cache blocks loaded. Although not shown in the figure, we confirmed that other applications in the benchmark share similar trends with the ones shown here.

The most important observation from the figure is that blocks loaded by some trigger instructions tend to have much higher average costs than others. This indicates that blocks that will be loaded by such trigger instructions in the future are also expected to have high costs, which in turn implies good predictability of write intensity. Therefore, one can predict write intensity of a block *on its miss* by observing whether the corresponding trigger instruction has already loaded write-intensive (or high-cost) blocks or not. Our architecture is designed on the basis of this idea to predict write intensity of blocks accurately. Although a few researches observed a correlation between cache hit/miss information and the instruction addresses that cause the cache accesses [14], [15], [16], [17], our work is the first to discover and exploit the correlation between write intensity of blocks and instructions that induce their cache misses.

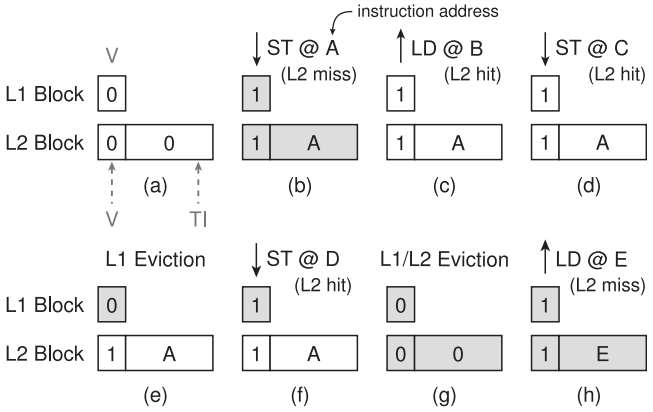


Fig. 2. An example of updating TI fields (shaded boxes indicate that the values are updated).

3 Write Intensity Predictor

Based on the motivation, we formulate the problem of identifying write-intensive blocks as finding trigger instructions that tend to load write-intensive (or high-cost) blocks, which we call *hot* trigger instructions.[3] For this purpose, we propose a new hardware structure called *write intensity predictor*, which is comprised of two steps: keeping track of trigger instructions and identifying hot trigger instructions. In the following sections, we describe the details of the two steps. For this, we assume without loss of generality that write intensity of L2 cache blocks needs to be predicted under a two-level writeback cache hierarchy. This mechanism could also be used for other variants of cache hierarchy, such as multi-level cache hierarchies or write-through caches.

3.1 Keeping Track of Trigger Instructions

The first step is to keep track of trigger instruction information for each block. To achieve this goal, we add an extra field TI (trigger instruction) to each L2 cache block to store an address of a trigger instruction. When an L1 cache miss occurs, the address of the instruction that incurs the miss is sent to the L2 cache along with the block fill request. This information is recorded into the L2 cache block only if the request also incurs an L2 cache miss. Through this, we can keep track of a trigger instruction address per block until the L2 cache block is evicted.

Fig. 2 illustrates an example sequence of updating TI fields. The figure shows changes in V (valid) and TI of an L2 cache block and the corresponding L1 cache block according to a sequence of load/store instructions. Let us assume that both blocks are initially invalid (a). After executing the store instruction having address A, the TI field of the L2 cache block is updated to A as it loads the block into the L2 cache (b). Since the block is now valid, subsequent load/store instructions do not update its TI field (c and d). Its value remains the same even on L1 cache misses as long as the block is still valid in the L2 cache (e and f). The TI field of the L2 cache block is cleaned on its eviction (g) and the next access to it updates the TI field with the new address E (h).

3. We use the term *cold* trigger instructions as the opposite meaning.

Note that trigger instruction addresses do not need to be exact since inaccurate write intensity prediction does not alter the semantics of program execution at all. In this paper, the lower-order 12 bits[4] of a trigger instruction address are stored into a TI field instead of the entire address to reduce the area overhead.

3.2 Identifying Hot Trigger Instructions

The second step is to identify hot trigger instructions, and eventually, to predict write intensity of blocks. The basic idea is to keep track of per-block costs and associate that information with the corresponding trigger instructions to predict write intensity of blocks to be loaded in the future.

First, we maintain the cost per L2 cache block so that we can determine write intensity of a block on its eviction. An 8-bit cost field is added into each L2 cache block for this purpose. The cost field of a block is incremented by ΔE_{r} and ΔE_{w} on each read and write operation to the block, respectively, and is reset on its eviction.

Based on this information, we update the states associated with trigger instructions to detect hot trigger instructions. For this, we add a lookup table called *predictor table*, which maintains a state for each trigger instruction. Each state is represented by a two-bit saturating counter where values 0 and 1 indicate cold trigger instructions and values 2 and 3 indicate hot trigger instructions. When a block is evicted, the value of the cost field is compared with the write intensity threshold κ. If the cost reaches or exceeds the threshold, the state of the corresponding trigger instruction is updated toward 'hot' state (i.e., incremented); otherwise, it is updated toward 'cold' state (i.e., decremented).

We found that this mechanism is very similar to a branch predictor, a well-known component of computer systems. The idea of transforming our problem into branch prediction is simple and intuitive. If the cost of a cache block is higher (lower) than the write intensity threshold, we consider it as a taken (untaken) branch at the corresponding trigger instruction. Through this formulation, taken and untaken branches in branch prediction are translated respectively into hot and cold trigger instructions in our problem.

To help better understanding of the proposed scheme, an example of the operating sequence is provided in Fig. 3. It shows an L2 cache block and a predictor table entry (state) corresponding to the trigger instruction of the block. For simplicity, we assume ΔE_{r}, ΔE_{w}, and κ to be -1, 24, and 20, respectively, in this example. Initially, the cost is set to zero and the state is set to weakly cold (i.e., value 1) (a). The cost is incremented by 24 for each write operation (b and d), whereas it is decremented by one for each read operation (c). Since the cost of the evicted block is higher than the threshold, the state of the corresponding trigger instruction is updated toward 'hot' state (e). On the

4. In fact, the length of a trigger instruction address is determined by the size of the predictor table (see Section 3.2) since trigger instruction addresses are used only for accessing the predictor table, which is tagless and direct-mapped. Since our architecture uses a predictor table with 4,096 ($= 2^{12}$) entries (see Section 5.1), lower-order 12 bits are enough to index predictor table entries.

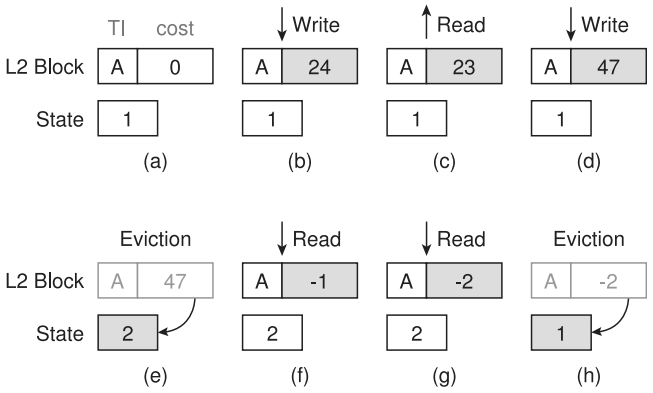


Fig. 3. An example of updating per-block costs and states ($\Delta E_r = -1$, $\Delta E_w = 24$, $\kappa = 20$).

contrary, assuming that the block is loaded again and is read twice before eviction (f and g), the state of the corresponding trigger instruction is updated toward 'cold' state on its eviction since its cost after the two read operations is lower than the threshold (h).

3.3 Dynamic Set Sampling

Our prediction scheme requires two additional fields to be added into each L2 cache block: the TI and cost fields. Although they introduce relatively small storage overhead (i.e., 24 bits per block), it might not be negligible for large caches.

Therefore, we introduce the concept of dynamic set sampling [18] into write intensity predictors to reduce the area overhead. The key idea behind this is that characteristics of the entire cache (in this case, whether a trigger instruction is hot or cold) can be generalized by observing a few cache sets. This is accomplished by adding a small partial tag array called *sampler* that contains only a few cache sets and simulates cache replacement behavior of those sets by sampling the cache accesses to them.

In this paper, the sampler is constructed by selecting $\frac{1}{32}$ of the entire sets[5] based on the simple-static policy [18]. Each sampler entry contains a valid bit, a 16-bit partial tag,[6] replacement policy bits (i.e., LRU bits), a 12-bit TI field, and an 8-bit cost field. On cache accesses to the sampled sets, the TI field and the cost field of the corresponding sampler entry are updated according to the aforementioned scheme. The sampler follows the plain LRU replacement policy (whereas the L2 cache uses a modified version of the LRU replacement policy for hybrid caches) in order to prevent write intensity prediction from being affected by previous block placement decisions.

Adopting dynamic set sampling in our architecture reduces the storage overhead incurred by the TI field and the cost field down to 1.28 bits per block without any noticeable impact on prediction accuracy, which will be discussed more in Section 6.5.

5. This sampling ratio has been used by many previous researches based on dynamic set sampling [15], [18], [19].

6. In the sampler, only the lower-order 16 bits of tags are stored because infrequent mismatches on sampler tags have negligible impact on prediction.

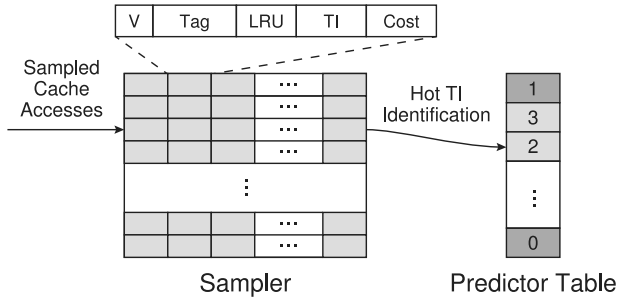


Fig. 4. The write intensity predictor (not to scale).

3.4 Summary

Fig. 4 summarizes the proposed write intensity predictor design. It consists of a sampler (structurally similar to tag arrays) and a predictor table (similar to bimodal branch predictors). The sampler simulates cache replacement behavior of sampled sets by monitoring cache accesses to them. When a sampler entry is evicted, the predictor table entry addressed by the value of the TI field is updated accordingly. This allows us to keep track of hot trigger instructions in the predictor table.

4 Prediction Hybrid Caches

In this section, we present a new hybrid cache architecture called *prediction hybrid cache* based on the proposed write intensity predictor.

4.1 Need for Write Intensity Prediction

As introduced before, energy efficiency of hybrid caches heavily depends on the policy of selecting blocks to be allocated to the SRAM region instead of the STT-RAM region. One common approach to this is to allocate blocks that are loaded due to write (or store) misses[7] into the SRAM region and other blocks to the STT-RAM region [1], [2], [6], [7]. The major weakness of it is that it has a high possibility to miss many write-intensive blocks due to its simplicity. For example, it cannot detect the case where a block is loaded by a read (or load) miss, but is written many times after the miss. To compensate the weakness, the approach is usually used along with block migrations, which swap a block in the STT-RAM region that has been written frequently with another block in the SRAM region. However, migrations could incur significant energy overhead since each migration requires at least one read and write operation for each region as studied in previous work [20].

Our inspiration is that, if we could know in advance whether a block will be write-intensive or not, efficiency of hybrid caches can be improved by simply allocating predicted write-intensive blocks into the SRAM region. This information is exactly what the write intensity predictor provides. Therefore, we utilize the write intensity predictor to improve block placement in hybrid caches for better energy efficiency.

7. We use the term 'load/store miss' as a cache miss incurred by a load/store instruction, respectively, as in the previous work [2].

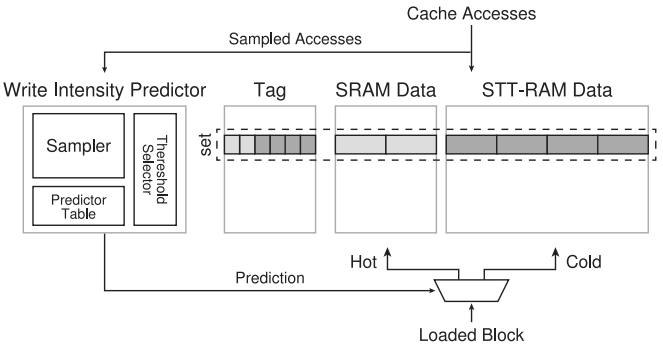


Fig. 5. Overview of the prediction hybrid cache.

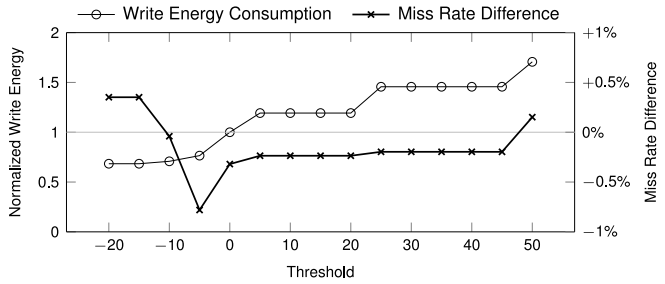


Fig. 6. Impact of the write intensity threshold on write energy consumption and miss rates in zeusmp.

4.2 Organization

Fig. 5 shows the overview of the prediction hybrid cache. The system is composed of an SRAM/STT-RAM hybrid cache and a write intensity predictor. Similarly to other hybrid cache architectures, each cache set is partitioned into SRAM ways and STT-RAM ways. For simplicity, we assume a six-way set-associative cache with two SRAM ways and four STT-RAM ways.

Unlike the data arrays, the tag array is constructed only with SRAM since each tag contains status bits and replacement information, which can be updated frequently [12], [21]. The sampler and the predictor table are also constructed with SRAM due to their small size. The write intensity predictor can optionally include a *threshold selector*, a hardware structure that adaptively adjusts the write intensity threshold according to application characteristics (see Section 4.4 for details).

4.3 Operations

Cache accesses in prediction hybrid caches are performed as in conventional caches. Only two small modifications are introduced to support write intensity prediction. First, accesses to the sampled sets are also sent to the sampler of the write intensity predictor in order to update necessary information for identifying hot trigger instructions. Second, the lower-level caches (e.g., L1 cache) are modified to transfer the lower-order 12 bits of the trigger instruction address along with the read request [14], [15], [16], [17]. If there are multiple levels of caches between processors and prediction hybrid caches, such information is simply propagated through the intermediate caches.

The uniqueness of our architecture is in its block placement policy. On a cache miss, the predictor table entry indexed by the corresponding trigger instruction address is checked to see if it is in 'hot' state. If so, the loaded block is allocated into the SRAM region since blocks loaded by hot trigger instructions are expected to be write-intensive; otherwise, it is allocated into the STT-RAM region.[8] After that, a victim block is selected among blocks in the target region according to the LRU replacement policy.

Unlike previous approaches, our architecture does not migrate blocks between the SRAM region and the STT-RAM region. The rationale behind this is that the migration scheme requires per-block counters to track the number of writes for each block and extra control logic and buffers to swap blocks between two regions [1], [2], [3], [4], [5], [6], [7], which incur nonnegligible overhead in terms of both energy and area. In particular, such overhead is not worthwhile considering that migration plays a role of correcting imperfect block placement, which takes only a small portion in our scheme due to the high prediction accuracy (see Section 6.3 for accuracy analysis). According to our experimental results, incorporating migration into our scheme further reduces energy consumption of the cache by 6.3 percent on average.[9] We can achieve such an improvement since the migration technique is orthogonal to our work. However, to show clearly the sole impact of our contribution, we do not consider migration throughout this paper.

4.4 Dynamic Threshold Adjustment

One of the most important parameters of our architecture is the write intensity threshold. If the threshold is set either too high or too low, the miss rate increases thereby increasing energy consumption of the main memory because most of the blocks are allocated into one region, which leave the other underutilized. Between the two cases, too low thresholds are more critical because of the small capacity of the SRAM region. However, too high thresholds also incur a large amount of write energy due to the increased number of blocks allocated to the STT-RAM region. Such behavior is well illustrated in Fig. 6, which shows the trend of write energy consumption and miss rates under various thresholds. The former is write energy consumed by the L2 cache during the execution of zeusmp, which is normalized to that of $\kappa = 0$. The latter is shown as a difference from the miss rates of the iso-capacity STT-RAM-only cache with the LRU replacement policy (e.g., $+1$ percent of miss rate difference indicates that the L2 cache miss rate increases by 1 percent compared to a conventional cache that uses the LRU replacement policy).

The difficulty here is that the optimal threshold depends on the distribution of per-block costs, which varies across different applications and program phases even within a single application. The problem of determining the best threshold becomes even more challenging in multicore systems since it is nearly impossible to

8. For non-blocking caches, miss status holding registers (MSHRs) need to keep track of trigger instruction addresses associated with in-flight misses. Otherwise, that information will not be available when the block data arrive from the next level of memory.

9. This result was obtained by exploring the best threshold for each single-core application under our architecture with migration and comparing it with Static PHC (see Section 6 for its meaning).

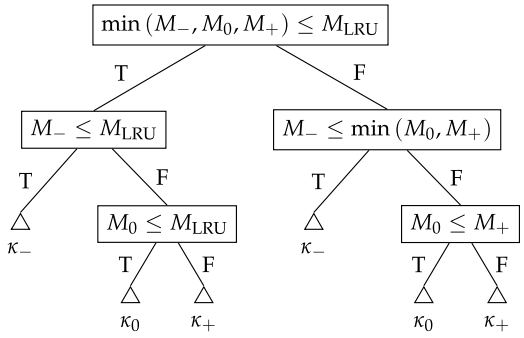


Fig. 7. Decision tree for dynamic threshold adjustment.

profile every possible combination of workloads that could be run simultaneously.

Therefore, we propose a hardware-based approach to adapt the write intensity threshold in runtime by monitoring application characteristics. It is designed to find the smallest threshold among the ones that do not increase the miss rate compared to the baseline replacement policy (e.g., LRU replacement policy in this paper). More specifically, our mechanism incrementally finds the best threshold by either incrementing or decrementing the current threshold according to the application characteristics. This raises two follow-up questions: (1) When should the threshold be incremented/decremented? (2) How much should it be changed?

For the first question, we estimate miss rates under different replacement policies as follows.

- M_{LRU}: the number of misses under the LRU replacement policy.
- M_0: the number of misses under the hybrid cache replacement policy with the current threshold κ_0.
- M_+: M_0 with a higher threshold $\kappa_+ > \kappa_0$.
- M_-: M_0 with a lower threshold $\kappa_- < \kappa_0$.

In order to reduce overhead of estimation, we use the number of misses in sampled sets (see Section 3.3) during a fixed interval as a proxy for miss rates of the entire cache.

M_0 can be obtained by monitoring misses in sampled sets of the hybrid cache itself because it uses the current threshold κ_0 for block placement. For this purpose, we add a 16-bit miss counter, which is incremented on each cache miss in sampled sets.

On the other hand, M_{LRU} assumes a conventional cache with the LRU replacement policy and thus may not be obtained from the hybrid cache. Fortunately, however, it can be estimated from the sampler without any additional hardware because the sampler follows the LRU replacement policy when replacing its entries (explained in Section 3.3). Thus, we add a 16-bit miss counter, which is incremented every time a sampler entry is installed (i.e., sampler miss).

In the case of M_+ and M_-, we add a predictor table and a partial tag array for each of them to simulate cache replacement under κ_+ and κ_-. The predictor tables are exactly the same as that in the write intensity predictor, except that they are updated with threshold κ_+ and κ_- (instead of κ_0) using the same mechanism described in Section 3.2. The partial tag arrays simulate cache replacement of sampled sets under the higher and the lower thresholds by using the newly added predictor tables. For this purpose, each entry of the partial tag arrays is composed of a valid bit, a 16-bit partial tag, and replacement policy bits.

Based on these values, the write intensity threshold is adjusted periodically with the policy depicted in Fig. 7. If any of κ_-, κ_0, or κ_+ shows the same or lower miss rate than that of the LRU replacement policy, the algorithm chooses the smallest threshold among the ones that satisfy the condition to reduce write energy consumption as much as possible (left subtree). Otherwise, it selects the one with the smallest miss count to reduce the miss rate (right subtree). After that, all miss counts are reset for the next interval.

The remaining issue is to determine the amount of threshold adjustment, i.e., the values of κ_+ and κ_-. A naive way is to set κ_+ and κ_- to $\kappa_0 + 1$ and $\kappa_0 - 1$, respectively. However, according to our experiments, it takes a long time to find the best threshold because of the wide range of cost values as shown in Fig. 1. Simply increasing the step size does not work either as the miss rate becomes sensitive to threshold change in a certain range of thresholds (e.g., from -15 to 5 in Fig. 6).

Therefore, we propose to monitor costs of evicted sampler entries during each interval and track the highest below-threshold cost and the lowest above-threshold cost among them for threshold candidates of κ_- and κ_+. The former and the latter can be defined as follows:

$$c_-(\kappa_n) = \begin{cases} \max C_- & \text{if } C_- = \{c \in C \mid c < \kappa_n\} \neq \varnothing \\ \kappa_n & \text{otherwise} \end{cases}$$

$$c_+(\kappa_n) = \begin{cases} \min C_+ & \text{if } C_+ = \{c \in C \mid c > \kappa_n\} \neq \varnothing \\ \kappa_n & \text{otherwise,} \end{cases}$$

where C is a set of costs of evicted sampler entries during the current interval and κ_n is the write intensity threshold for the next interval determined by the decision tree in Fig. 7. However, since it is unknown which of κ_-, κ_0, and κ_+ will become κ_n until the end of the current interval, we track c_- and c_+ for each of κ_-, κ_0, and κ_+. Note that, since $c_-(\kappa_n)$ (or $c_+(\kappa_n)$) is the maximum (or minimum) cost among the ones below (or above) the threshold, it can be tracked by adding only a single 8-bit register (which has the same bit width as the cost field in the sampler) without storing entire C. Thus, tracking c_- and c_+ for each of κ_-, κ_0, and κ_+ requires six 8-bit registers. At the end of each interval, κ_- and κ_+ are updated to $c_-(\kappa_n)$ and $c_+(\kappa_n)$, respectively. This has an effect of enlarging the step size of threshold adjustment when the cost distribution is sparse thereby allowing faster adaptation. For example, in Fig. 6, the threshold 20 can directly be decremented to 0 without sweeping through any intermediate values, while it is adjusted more carefully between 0 and -15. In total, our threshold selector introduces low storage overhead of 1.7 bits per block under our single-core configuration (see Section 5.1 for details).

5 Evaluation Methodology

5.1 Simulator Configuration

We evaluate our architecture under single-/quad-core systems by using a cycle-accurate x86-64 simulator based on Pin [22]. The simulator models 3 GHz, out-of-order, four-issue superscalar cores with 128-entry instruction window.

TABLE 1
Characteristics of the L2 Cache

	Single-Core (1 MB)			Quad-Core (4 MB)		
	SRAM	STT	Hybrid	SRAM	STT	Hybrid
Read Latency†	10	10	10	13	13	13
Write Latency†	10	37	10/37*	13	40	13/40*
Read Energy (pJ)	0.09	0.07	0.09/0.07*	0.18	0.08	0.18/0.08*
Write Energy (pJ)	0.09	0.64	0.09/0.64*	0.18	0.69	0.18/0.69*
Static Power (mW)	14.7	2.32	5.41	69.5	7.80	20.8
Area (mm^2)	3.77	0.95	1.66	16.9	2.74	6.28

† Latencies shown in cycles under 3 GHz.
* Parameters for accesses to the SRAM/STT-RAM region, respectively.

TABLE 2
Workloads from SPEC CPU2006

Workload	WBPKI
lbm	43.8
mcf	26.9
soplex	20.8
libquantum	18.9
leslie3d	10.8
omnetpp	10.8
GemsFDTD	10.6
gcc	8.9
h264ref	8.0
milc	7.6
zeusmp	7.3
xalancbmk	5.6
bzip2	5.2
tonto	4.6
wrf	4.4
hmmer	2.9

Mix	Workloads
high1	lbm, mcf, soplex, libquantum
high2	leslie3d, omnetpp, GemsFDTD, gcc
high3	lbm, soplex, leslie3d, GemsFDTD
high4	mcf, libquantum, omnetpp, gcc
mid1	lbm, mcf, hmmer, wrf
mid2	soplex, libquantum, tonto, bzip2
mid3	leslie3d, omnetpp, xalancbmk, zeusmp
mid4	GemsFDTD, gcc, milc, h264ref
low1	hmmer, wrf, tonto, bzip2
low2	xalancbmk, zeusmp, milc, h264ref
low3	hmmer, tonto, xalancbmk, milc
low4	wrf, bzip2, zeusmp, h264ref

The cache hierarchy is composed of 32 KB, 4/8-way set-associative private L1 instruction/data caches and a 1 MB per core, 16-way set-associative shared L2 cache, whose block size is 64 bytes. Both the L1 data cache and the L2 cache are non-blocking caches with eight MSHRs. Cache coherence is managed by the MESI protocol without enforcing the inclusion property. The main memory consists of two dual in-line memory modules (DIMMs), each of which is comprised of one rank containing eight 2 Gb DDR3-1600 11-11-11-28 devices [23]. The memory controller uses the FR-FCFS scheduling under the open-row policy [24] and has a 64-entry request queue.

The proposed hybrid cache architecture is applied to the L2 cache and is compared against read-write aware hybrid caches (RWHCA) [2]. The hybrid caches are configured to have 4 SRAM ways and 12 STT-RAM ways. In RWHCA, migrations are triggered on four consecutive hits in the wrong region as in the original paper. In our scheme, dynamic threshold adjustment is triggered every five million cycles (chosen based on simulation results).

Table 1 shows energy and delay characteristics of the L2 cache at a 45 nm technology modeled by CACTI 6.5 [25] and NVSim [26]. LOP devices are used for peripheral circuits and SRAM cells as they are known to be best matched to the characteristics of last-level caches in commercial high-performance processors [27]. STT-RAM cell parameters are obtained from the previous researches [28], [29]. As shown in the table, read and write operations in hybrid caches are modeled to have different latencies and energy characteristics depending on the target region. Based on these values, we set $\Delta E_{\mathrm{r}} = -1$ and $\Delta E_{\mathrm{w}} = 24$ for the single-core system and $\Delta E_{\mathrm{r}} = -1$ and $\Delta E_{\mathrm{w}} = 6$ for the quad-core system.

We also model energy consumption of additional circuits for hybrid cache management by using CACTI 6.5. In RWHCA, a two-bit counter is added into each block for migration, which incurs 4 KB overhead in the single-core system. In our architecture, a write intensity predictor is composed of a sampler (2.6 KB), a 4,096-entry bimodal predictor table (1 KB), and a threshold selector (3.3 KB) thereby adding 6.9 KB in the single-core system. Although we use a small predictor table, our simulation results confirm that larger predictor tables do not have any noticeable impact on prediction accuracy (e.g., less than 0.1 percent difference in accuracy with a four times larger predictor table). The overhead of the two-bit counters, the sampler, and tag arrays of the threshold selector is quadrupled in the quad-core system due to the four times larger cache capacity, while the size of the predictor table remains the same, i.e., 19.4 KB in total. In summary, our scheme introduces 0.67 percent/0.45 percent storage overhead in the single-/quad-core system.

According the latency modeling from CACTI 6.5, the write intensity predictor is not on the critical path of cache accesses, and thus, does not increase cache access latency. This is because (1) the sampler is updated in parallel with cache accesses and (2) the predictor table is accessed only on cache misses.

Lastly, energy consumption of the main memory is modeled by Micron System Power Calculator [30].

5.2 Workloads

We use 16 write-intensive[10] benchmarks from SPEC CPU2006 [31] as single-core workloads. For benchmarks having more than one input set, we choose the representative ones by referring to the previous work [32]. Each benchmark is run for one billion instructions of its representative phase [33].

From the single-core workloads, we construct 12 multi-programmed workloads as shown in Table 2. To evaluate with a wide range of write intensity, we first classify single-core workloads into top eight (denoted as HIGH) and bottom eight (denoted as LOW) in terms of write intensity and assemble four HIGHs into high1 to high4, two HIGHs and two LOWs into mid1 to mid4, and four LOWs into low1 to low4. Each workload is run for four billion instructions.

We also use four multithreaded workloads from PARSEC 2.1 [34] for multicore evaluations. Each benchmark is run for two billion instructions from the beginning of regions-of-interest (ROI).

6 SINGLE-CORE EVALUATIONS

In this section, we compare our architecture (denoted as Dynamic PHC) against SRAM baseline, STT-RAM baseline, and RWHCA [2]. We also evaluate a static version of

10. Write intensity of a benchmark is defined as WBPKI (writeback per kilo instruction) of the L1 data cache under the single-core system.

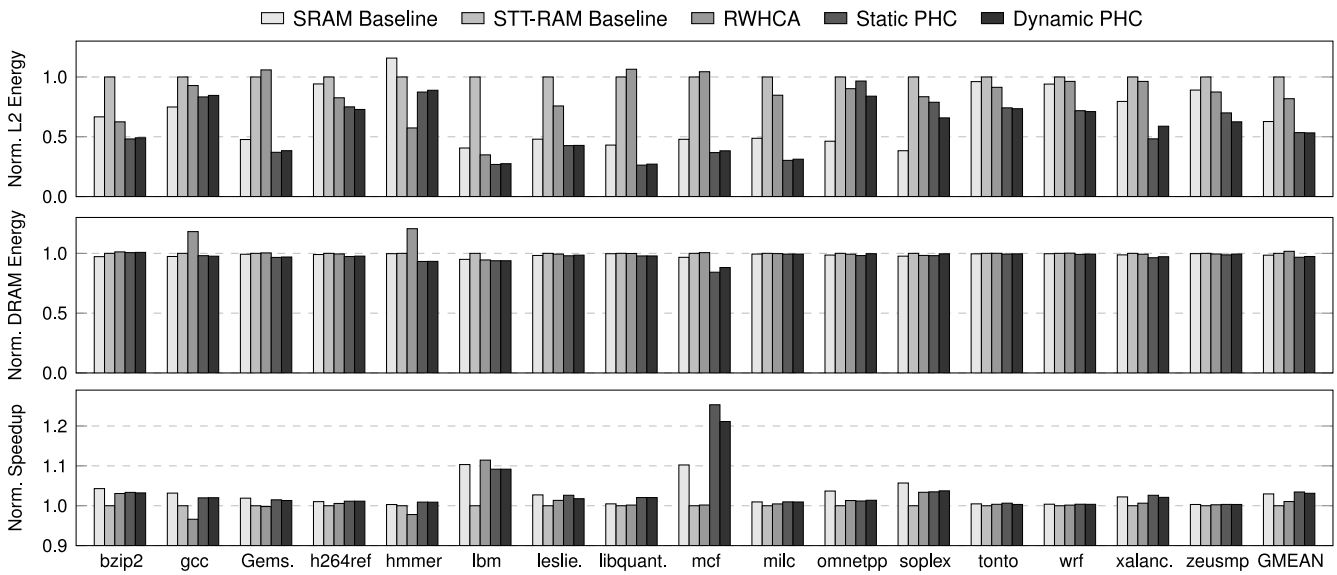


Fig. 8. Energy consumption and speedup under the single-core system (normalized to the STT-RAM baseline).

our architecture (denoted as Static PHC) as a limitation study, in which the write intensity threshold is statically selected on a per-application basis through profiling from $\kappa = -20$ to 50 at intervals of 5. For each application, we choose the best threshold that minimizes energy consumption of the L2 cache while not increasing energy consumption of the main memory compared to the STT-RAM baseline.

6.1 Energy Consumption and Speedup

Fig. 8 shows energy consumption and speedup of different configurations under the single-core system. Energy consumption of the L2 cache includes that of additional circuits for hybrid cache management. The last set of bars labeled by 'GMEAN' represents geometric mean of energy consumption or speedup.

As shown in the figure, Dynamic PHC reduces energy consumption of the L2 cache by 47 percent compared to the STT-RAM baseline. Moreover, its energy consumption is even slightly less than that of the SRAM baseline. This is a promising result considering that the 16 benchmarks evaluated here are write-intensive, in which STT-RAM caches are undesirable due to its seven times higher write energy (see Section 6.6 for results for other benchmarks).

Compared to RWHCA, our architecture achieves a 35 percent reduction in energy consumption of the L2 cache. This is because our scheme identifies more write-intensive blocks than RWHCA does in general, and thus, more write operations are handled in the SRAM region. Note that allocating more blocks into the SRAM region is beneficial as long as it does not increase energy consumption of the main memory. For example, RWHCA consumes 35 percent less L2 cache energy in hmmer compared to our scheme, but increases energy consumption of the main memory by 29 percent, which leads to a 29 percent increase in total energy consumption. On average, our scheme even reduces energy consumption of the main memory by 4.3 percent compared to RWHCA, indicating that our scheme achieves a better balance between write energy reduction and miss rates.

Also, Dynamic PHC reaches approximately the same level of energy efficiency as that of Static PHC without the need for per-application profiling. In some cases (e.g., omnetpp, soplex, and zeusmp), Dynamic PHC even outperforms Static PHC because it can adapt to different phases within an application at runtime. On average, Dynamic PHC consumes 0.4 percent less energy in the L2 cache with a 0.7 percent increase in energy consumption of the main memory.

Lastly, our scheme improves system performance by 3 and 2 percent compared to the STT-RAM baseline and RWHCA, respectively, thereby reaching the performance of the SRAM baseline. The major contribution of this speedup comes from the reduction in write operations to the STT-RAM region, each of which takes three times longer than that in the SRAM region. Also, the reduction in main memory activities also improves the performance in some benchmarks (e.g., mcf).

6.2 Energy Breakdown

Fig. 9 shows energy breakdown of the L2 cache under the baselines and our architecture. Each bar is decomposed into static energy consumption of the L2 cache, dynamic energy consumption of the L2 cache (further decomposed into tag access, cache read, and cache write), and energy

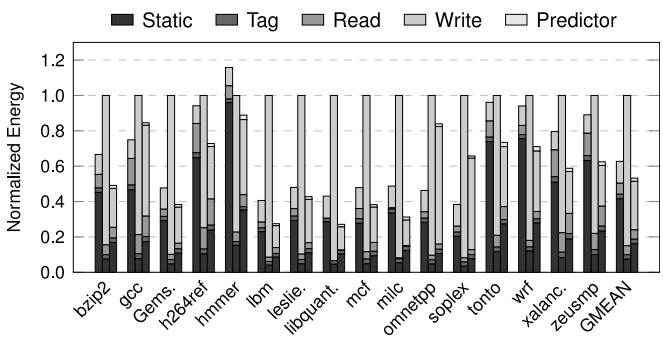


Fig. 9. Energy breakdown of the SRAM/STT-RAM baseline (left/middle) and Dynamic PHC (right).

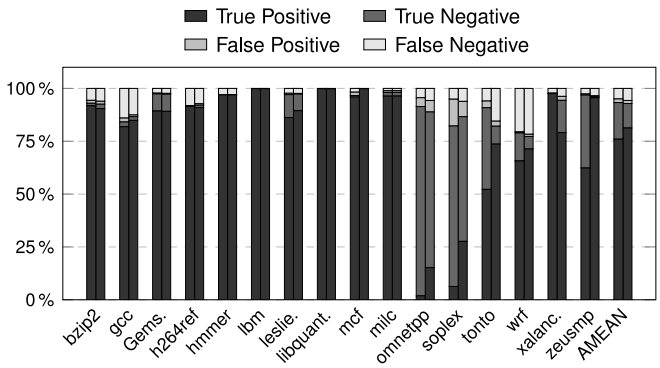


Fig. 10. Coverage and accuracy of the write intensity predictor with static (left) and dynamic (right) thresholds.

consumption of the write intensity predictor (only for Dynamic PHC).

In the SRAM baseline, high static power of SRAM contributes the most to L2 cache energy. Although replacing SRAM with STT-RAM mitigates this issue, it also significantly increases write energy consumption. Dynamic PHC balances these two aspects by (1) constructing caches with large STT-RAM and small SRAM to minimize static power and (2) judiciously placing cache blocks to SRAM in a way to maximize the benefit of low write energy of SRAM. On average, Dynamic PHC consumes 61 percent less static energy compared to the SRAM baseline and 61 percent less dynamic energy compared to the STT-RAM baseline.

6.3 Coverage and Accuracy

Fig. 10 shows coverage and accuracy of the write intensity predictor with and without dynamic threshold adjustment. Legends that start with 'True/False' indicate correct/incorrect prediction. Also, legends that end with 'Positive/Negative' represent cases where blocks are predicted to be write-intensive/non-write-intensive. For example, 'True Positive' represents the case where a block is predicted to be write-intensive at block fill, and its cost at eviction is actually greater than or equal to the threshold. On average, the write intensity predictor achieves 93 percent of accuracy under the static threshold policy, and using dynamic threshold adjustment maintains about the same level of accuracy.

One interesting observation is that the optimal ratio of blocks allocated to the SRAM region (83 percent) is much higher than the ratio of the SRAM region capacity (25 percent). One might expect the two ratios to be about the same in order to balance the capacity pressure of two regions, i.e., maintaining ratio of the number of blocks inserted into the SRAM region to that of the STT-RAM region to be the capacity ratio of the two. However, we found that allowing more blocks to be allocated into the SRAM region does not increase miss rates until a certain point because most of the blocks are reused within a short time interval, in which case early eviction of them does not increase miss rates. This is the reason why Dynamic PHC is based on monitoring miss rates, rather than simply controlling the insertion ratio of two regions.

6.4 Sensitivity to Write Intensity Threshold

Fig. 11 shows total energy consumption (the L2 cache and the main memory) in Static PHC with different thresholds

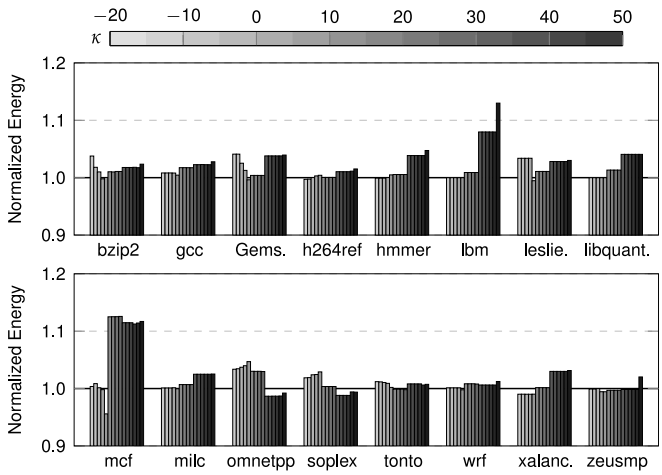


Fig. 11. Total energy consumption of Static PHC with different thresholds (Normalized to Dynamic PHC).

from -20 to 50. The results are normalized to those of Dynamic PHC, which are shown as a thick line.

As can be seen in the figure, the best value for the write intensity threshold widely varies across applications. For example, the best threshold for bzip2 is -5, while that for soplex is 40. This signifies the need for dynamic threshold adjustment, especially considering that static profiling is not always possible and static thresholds are less robust against runtime variation.

6.5 Impact of Dynamic Set Sampling

Our architecture uses the concept of dynamic set sampling for both sampler and threshold selector to reduce storage overhead. To evaluate its impact on prediction accuracy, we compare total energy consumption (excluding the write intensity predictor) without dynamic set sampling against that with it. According to the evaluation, the two differs by only 1.3 percent on average while dynamic set sampling reduces storage overhead by $\frac{1}{32}$.

6.6 Results for Non-Write-Intensive Workloads

Fig. 12 shows energy consumption of the L2 cache and the main memory for 13 non-write-intensive benchmarks in SPEC CPU2006. The results are normalized to the STT-RAM baseline.

As opposed to Fig. 8, the SRAM baseline consumes 73 percent higher energy in the L2 cache compared to the STT-RAM baseline. This is because, due to the low write intensity of the benchmarks, static energy becomes the dominant source of the energy consumption of the L2 cache.

Under this circumstance, our architecture achieves almost the same energy consumption compared to the STT-RAM baseline. This, together with the results in Fig. 8, can be summarized as follows:

Our architecture reaches energy consumption of an SRAM cache under high write intensity and that of an STT-RAM cache under low write intensity.

This is an extremely desirable property since it takes advantage of both technologies with runtime adaptation. On average across all SPEC CPU2006 workloads, our scheme achieves 29 and 30 percent reductions in energy

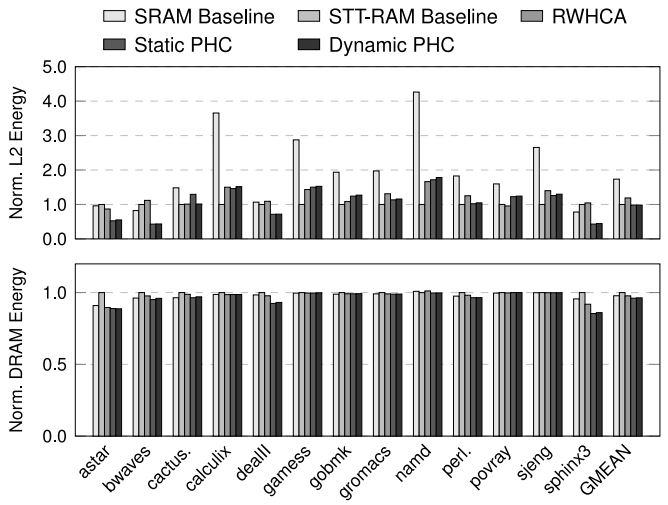


Fig. 12. Energy consumption of the L2 cache and the main memory for benchmarks with low write intensity.

consumption of the L2 cache compared to the SRAM baseline and the STT-RAM baseline, respectively, while those of RWHCA are only 2.3 and 3.2 percent.

7 Multicore Evaluations

In this section, we evaluate our architecture under the quad-core system described in Section 5.1. Simulation configurations are similar to those in the single-core evaluations, except the four times larger capacity of the L2 cache. Also, ΔE_{r} and ΔE_{w} are different from the ones in the single-core system. Due to this, we choose the best write intensity threshold for Static PHC from $\kappa = -12$ to 32 at intervals of 4, instead of using the same range of possible thresholds as in the single-core evaluations.

Fig. 13 compares energy consumption and speedup of our architecture against previous approaches. The results are normalized to the STT-RAM baseline. We use weighted speedup [35] as a performance metric for multiprogrammed workloads.

As opposed to the single-core evaluations shown in Fig. 8, the SRAM baseline now consumes more energy than the STT-RAM baseline does. This is because the quadrupled capacity of the L2 cache increases the portion of static energy in its energy consumption. Similarly, effectiveness of RWHCA in energy reduction degrades as well due to the increased static power from the SRAM region.

Nevertheless, Dynamic PHC still achieves a 30 percent reduction in energy consumption of the L2 cache compared to the STT-RAM baseline. This, in turn, leads to a 38 percent smaller energy consumption in the L2 cache compared to the SRAM baseline. These results are achieved through fully utilizing a small SRAM cache (which minimizes increase in static energy compared against the SRAM baseline) to reduce write energy consumption in the STT-RAM region. In addition, energy consumption of the main memory in Dynamic PHC is 6.1 and 3.6 percent smaller than that of the STT-RAM baseline and the SRAM baseline, respectively.

In addition, Dynamic PHC improves system performance by 8.2 percent compared to the STT-RAM baseline. This is contributed mainly by the reduction in write operations to the STT-RAM region as in the single-core evaluations. However, the amount of speedup is higher in the quad-core system since the L2 cache suffers from higher contention in the quad-core system than in the single-core system, which can be alleviated in our architecture by reducing write operations to the STT-RAM region.

As in the single-core evaluations, Static PHC and Dynamic PHC show negligible difference in terms of energy and performance under the quad-core system. This is an important result in the context of multicore systems where static profiling of workloads is impractical due to runtime variation.[11]

Prediction accuracy in the quad-core system remains almost the same as that in the single-core system. On average, Static PHC and Dynamic PHC achieve 96 and 95 percent of accuracy, respectively.

8 Related Work

As introduced previously, high write energy has been considered as a major drawback of STT-RAM caches [13], [36]. Consequently, there have been many researches that try to mitigate such weakness for better energy efficiency. In this section, we provide a brief introduction of such techniques except for hybrid caches (already covered in Section 1).

Recently, researchers have discovered that reducing retention time of STT-RAM cells leads to a reduction in write energy at the cost of losing non-volatility. Several researchers have proposed to use such device-level techniques for STT-RAM caches by exploiting the observation that lifetime of cache blocks is short in most cases. Smullen et al. [37] evaluated volatile STT-RAM cells for on-chip caches and found that reducing retention time of STT-RAM can improve energy efficiency of STT-RAM caches. Sun et al. [38] proposed a cache hierarchy design composed of volatile STT-RAM L1 caches and mixed retention time STT-RAM L2/L3 caches. Jog et al. [39] designed revived STT-RAM caches, in which expired cache blocks are held in a small buffer before eviction to reduce writebacks. Our technique is applicable for such proposals since a combination of volatile and non-volatile STT-RAM cells can also be thought as a form of hybrid caches [38].

Also, there have been several attempts to reduce write energy through device-/circuit-level techniques. Zhou et al. [40] proposed differential writes for phase-change memory to cancel bit writes that do not change the cell data. Cho and Lee [41] proposed Flip-N-Write, a simple invert coding technique to enhance the efficiency of differential writes. Guo et al. [42] presented subbank buffers, which helps to implement differential writes in STT-RAM arrays. Aside from such circuit-level approaches, there also have been a number of researches to develop perpendicular STT-RAM devices [43], which have been known to consume less write energy compared to in-plane STT-RAM devices, although they are currently encountering many challenges to be matured. Such techniques can be applied on top of our

11. In the case of multiprogrammed workloads, combinations of applications to be run simultaneously are hard to be known in advance. Similarly, actual scheduling of threads in multithreaded workloads highly depends on runtime information.

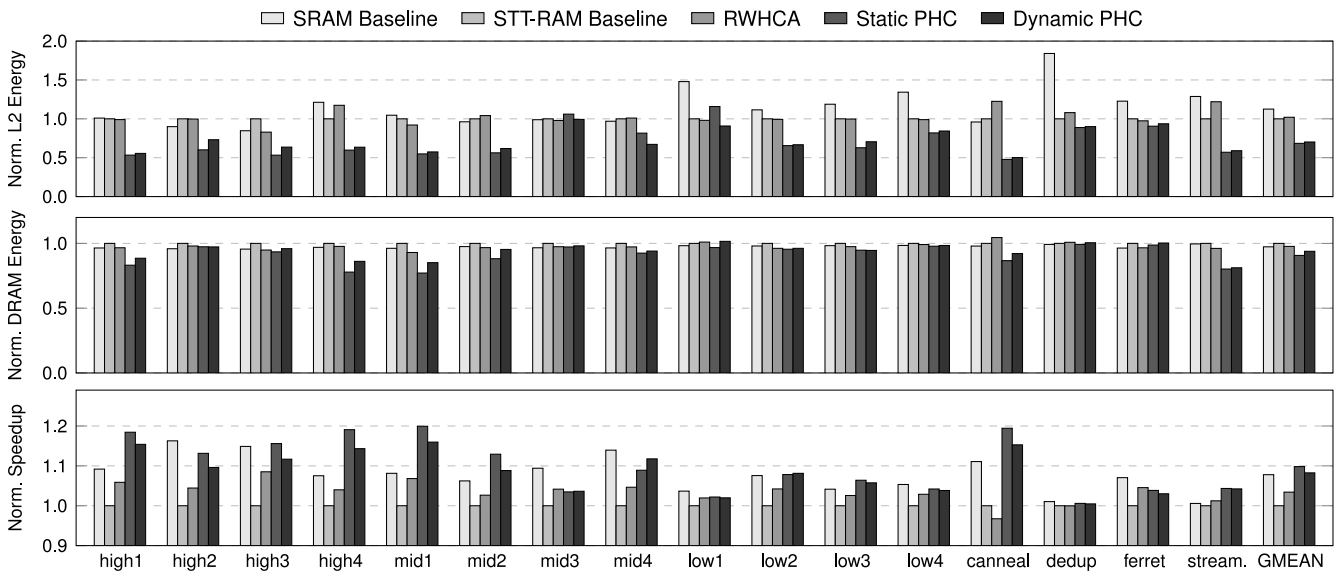


Fig. 13. Energy consumption and speedup under the quad-core system (normalized to the STT-RAM baseline).

architecture as hybrid caches reduce write operations to the STT-RAM caches at a cache block granularity whereas they do at a much finer granularity.

9 Conclusion

This paper proposed prediction hybrid caches, an STT-RAM based hybrid cache architecture with a novel block placement policy. The key concept of it is write intensity prediction, which predicts write intensity of cache blocks on their misses and determines block placement based on it. For this purpose, our mechanism utilizes a newly discovered correlation between write intensity of blocks and instruction addresses that incur their misses. Moreover, it includes a mechanism to dynamically adapt the threshold of write intensity according to application characteristics. Experimental results show that our architecture achieves 28 percent (31 percent) energy reduction in hybrid caches, 3 percent (4 percent) energy reduction in main memories, and 1.4 percent (4.7 percent) speedup in the single-core (quad-core) system compared to the existing hybrid cache architecture. Although this paper utilizes the concept of write intensity prediction for hybrid caches, we believe that it can also be adopted for many other non-volatile memory systems where prior knowledge of write-intensive data could be helpful for improving energy efficiency.

Acknowledgments

This work was supported by the National Research Foundation of Korea (NRF) grants funded by the Korean government (MEST) (No. 2012R1A2A2A06047297), the IT R&D program of MKE/KEIT (No. 10041608, Embedded System Software for New Memory-based Smart Devices), and Samsung Electronics.

References

[1] G. Sun, X. Dong, Y. Xie, J. Li, and Y. Chen, "A novel architecture of the 3D stacked MRAM L2 cache for CMPs," in *Proc. Int. Symp. High Perform. Comput. Archit.*, 2009, pp. 239–249.

[2] X. Wu, J. Li, L. Zhang, E. Speight, and Y. Xie, "Power and performance of read-write aware hybrid caches with non-volatile memories," in *Proc. Design Automat. Test Eur. Conf.*, 2009, pp. 737–742.

[3] X. Wu, J. Li, L. Zhang, E. Speight, R. Rajamony, and Y. Xie, "Hybrid cache architecture with disparate memory technologies," in *Proc. Int. Symp. Comput. Archit.*, 2009, pp. 34–45.

[4] X. Wu, J. Li, L. Zhang, E. Speight, R. Rajamony, and Y. Xie, "Design exploration of hybrid caches with disparate memory technologies," *ACM Trans. Archit. Code Optim.*, vol. 7, no. 3, pp. 15:1–15:34, Dec. 2010.

[5] A. Jadidi, M. Arjomand, and H. Sarbazi-Azad, "High-endurance and performance-efficient design of hybrid cache architectures through adaptive line replacement," in *Proc. Int. Symp. Low Power Electron. Design*, 2011, pp. 79–84.

[6] J. Li, C. J. Xue, and Y. Xu, "STT-RAM based energy-efficiency hybrid cache for CMPs," in *Proc. Int. Conf. VLSI Syst.-on-Chip*, 2011, pp. 31–36.

[7] J. Li, L. Shi, C. J. Xue, C. Yang, and Y. Xu, "Exploiting set-level write non-uniformity for energy-efficient NVM-based hybrid cache," in *Proc. Symp. Embedded Syst. Real-Time Multimedia*, 2011, pp. 19–28.

[8] Y.-T. Chen, J. Cong, H. Huang, C. Liu, R. Prabhakar, and G. Reinman, "Static and dynamic co-optimizations for blocks mapping in hybrid caches," in *Proc. Int. Symp. Low Power Electron. Design*, 2012, pp. 237–242.

[9] Q. Li, M. Zhao, C. J. Xue, and Y. He, "Compiler-assisted preferred caching for embedded systems with STT-RAM based hybrid cache," in *Proc. Int. Conf. Lang. Compil. Tools Theory Embedded Syst.*, 2012, pp. 109–118.

[10] Y. Li, Y. Chen, and A. K. Jones, "A software approach for combating asymmetries of non-volatile memories," in *Proc. Int. Symp. Low Power Electron. Design*, 2012, pp. 191–196.

[11] J. Ahn, S. Yoo, and K. Choi, "Write intensity prediction for energy-efficient non-volatile caches," in *Proc. Int. Symp. Low Power Electron. Design*, 2013, pp. 223–228.

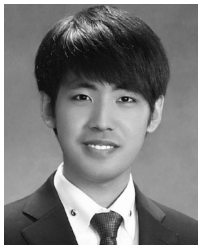
[12] S. P. Park, S. Gupta, N. Mojumder, A. Raghunathan, and K. Roy, "Future cache design using STT MRAMs for improved energy efficiency: Devices, circuits and architecture," in *Proc. Design Automat. Conf.*, 2012, pp. 492–497.

[13] M.-T. Chang, P. Rosenfeld, S.-L. Lu, and B. Jacob, "Technology comparison for large last-level caches (L^3Cs): Low-leakage SRAM, low write-energy STT-RAM, and refresh-optimized eDRAM," in *Proc. Int. Symp. High Perform. Comput. Archit.*, 2013, pp. 143–154.

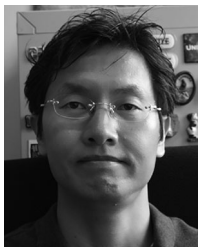
[14] M. Kharbutli and Y. Solihin, "Counter-based cache replacement and bypassing algorithms," *IEEE Trans. Comput.*, vol. 57, no. 4, pp. 433–447, Apr. 2008.

[15] S. Khan, Y. Tian, and D. A. Jiménez, "Sampling dead block prediction for last-level caches," in *Proc. Int. Symp. Microarchit.*, 2010, pp. 175–186.

[16] C.-J. Wu, A. Jaleel, W. Hasenplaugh, M. Martonosi, S. C. Steely Jr, and J. Emer, "SHiP: Signature-based hit predictor for high performance caching," in *Proc. Int. Symp. Microarchit.*, 2011, pp. 430–441.
[17] M. K. Qureshi and G. H. Loh, "Fundamental latency trade-off in architecting DRAM caches: Outperforming impractical SRAM-tags with a simple and practical design," in *Proc. Int. Symp. Microarchit.*, 2012, pp. 235–246.
[18] M. K. Qureshi, D. N. Lynch, O. Mutlu, and Y. N. Patt, "A case for MLP-aware cache replacement," in *Proc. Int. Symp. Comput. Archit.*, 2006, pp. 167–178.
[19] M. Qureshi and Y. Patt, "Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches," in *Proc. Int. Symp. Microarchit.*, 2006, pp. 423–432.
[20] Q. Li, J. Li, L. Shi, C. J. Xue, and Y. He, "MAC: Migration-aware compilation for STT-RAM based hybrid cache in embedded systems," in *Proc. Int. Symp. Low Power Electron. Design*, 2012, pp. 351–356.
[21] M. Rasquinha, D. Choudhary, S. Chatterjee, S. Mukhopadhyay, and S. Yalamanchili, "An energy efficient cache design using spin torque transfer (STT) RAM," in *Proc. Int. Symp. Low Power Electron. Design*, 2010, pp. 389–394.
[22] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood, "Pin: Building customized program analysis tools with dynamic instrumentation," in *Proc. Conf. Program. Lang. Design Implement.*, 2005, pp. 190–200.
[23] *2Gb: x4, x8, x16 DDR3 SDRAM*, Micron Technology, 2006.
[24] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens, "Memory access scheduling," in *Proc. Int. Symp. Comput. Archit.*, 2000, pp. 128–138.
[25] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, "CACTI 6.0: A tool to model large caches," HP Lab., Palo Alto, CA, USA, Tech. Rep. HPL-2009-85, 2009.
[26] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, "NVSim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 31, no. 7, pp. 994–1007, Jul. 2012.
[27] D. S. Gracia, G. Dimitrakopoulos, T. M. Arnal, M. G. H. Katevenis, and V. V. Yúfera, "LP-NUCA: Networks-in-cache for high-performance low-power embedded processors," *IEEE Trans. VLSI Syst.*, vol. 20, no. 8, pp. 1510–1523, Aug. 2012.
[28] Y. Chen, X. Wang, H. Li, H. Xi, Y. Yan, and W. Zhu, "Design margin exploration of spin-transfer torque RAM (STT-RAM) in scaled technologies," *IEEE Trans. VLSI Syst.*, vol. 18, no. 12, pp. 1724–1734, Dec. 2010.
[29] Y. Zhang, W. Wen, and Y. Chen, "The prospect of STT-RAM scaling from readability perspective," *IEEE Trans. Magn.*, vol. 48, no. 11, pp. 3035–3038, Nov. 2012.
[30] *Calculating Memory System Power for DDR3*, TN-41-01, Micron Technology, 2007.
[31] J. L. Henning, "SPEC CPU2006 benchmark descriptions," *ACM SIGARCH Comput. Archit. News*, vol. 34, no. 4, pp. 1–17, Sep. 2006.
[32] A. Phansalkar, A. Joshi, and L. K. John, "Subsetting the SPEC CPU2006 benchmark suite," *ACM SIGARCH Comput. Archit. News*, vol. 35, no. 1, pp. 69–76, Mar. 2007.
[33] H. Patil, R. Cohn, M. Charney, R. Kapoor, A. Sun, and A. Karunanidhi, "Pinpointing representative portions of large Intel ® Itanium ® programs with dynamic instrumentation," in *Proc. Int. Symp. Microarchit.*, 2004, pp. 81–92.
[34] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite: Characterization and architectural implications," in *Proc. Int. Conf. Parallel Archit. Compilat. Techn.*, 2008, pp. 72–81.
[35] A. Snavely and D. M. Tullsen, "Symbiotic jobscheduling for a simultaneous multithreaded processor," in *Proc. Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, 2000.
[36] X. Dong, X. Wu, G. Sun, Y. Xie, H. Li, and Y. Chen, "Circuit and microarchitecture evaluation of 3D stacking magnetic RAM (MRAM) as a universal memory replacement," in *Proc. Design Automat. Conf.*, 2008, pp. 554–559.
[37] C. W. Smullen, V. Mohan, A. Nigam, S. Gurumurthi, and M. R. Stan, "Relaxing non-volatility for fast and energy-efficient STT-RAM caches," in *Proc. Int. Symp. High Perform. Comput. Archit.*, 2011, pp. 50–61.
[38] Z. Sun, X. Bi, H. Li, W.-F. Wong, Z.-L. Ong, X. Zhu, and W. Wu, "Multi retention level STT-RAM cache designs with a dynamic refresh scheme," in *Proc. Int. Symp. Microarchit.*, 2011, pp. 329–338.
[39] A. Jog, A. K. Mishra, C. Xu, Y. Xie, V. Narayanan, R. Iyer, and C. R. Das, "Cache revive: Architecting volatile STT-RAM caches for enhanced performance in CMPs," in *Proc. Design Automat. Conf.*, 2012, pp. 243–252.
[40] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "A durable and energy efficient main memory using phase change memory technology," in *Proc. Int. Symp. Comput. Archit.*, 2009, pp. 14–23.
[41] S. Cho and H. Lee, "Flip-N-Write: A simple deterministic technique to improve PRAM write performance, energy and endurance," in *Proc. Int. Symp. Microarchit.*, 2009, pp. 347–357.
[42] X. Guo, E. Ipek, and T. Soyata, "Resistive computation: Avoiding the power wall with low-leakage, STT-MRAM based computing," in *Proc. Int. Symp. Comput. Archit.*, 2010, pp. 371–382.
[43] D. Apalkov, A. Khvalkovskiy, S. Watts, V. Nikitin, X. Tang, D. Lottis, K. Moon, X. Luo, E. Chen, A. Ong, A. Driskill-Smith, and M. Krounbi, "Spin-transfer torque magnetic random access memory (STT-MRAM)," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 9, no. 2, pp. 13:1–13:35, May 2013.



Junwhan Ahn (S'12) received the BS degree in electrical engineering from Seoul National University, Seoul, Republic of Korea, in 2011, where he is currently working toward the PhD degree in electrical engineering and computer science with the Department of Electrical and Computer Engineering. His current research interests include memory system design and emerging memory technologies. He is a student member of the IEEE.



Sungjoo Yoo (M'00) received the PhD degree from Seoul National University in 2000. He worked as a researcher at TIMA laboratory, Grenoble France, from 2000 to 2004. He was a principal engineer at Samsung System LSI from 2004 to 2008. He was at POSTECH from 2008 to 2015. He joined Seoul National University in 2015 and is currently an associate professor. His main research interests include memory-related issues in mobile devices and servers. He received the Best Paper Award at International SoC Conference (ISOCC) in 2006 and Best Paper Award nominations at Design Automation Conference (DAC) in 2011 and Design Automation and Test in Europe (DATE) in 2002 and 2009. He is a member of the IEEE.

Kiyoung Choi (M'88-SM'08) received the BS degree in electronics engineering from Seoul National University, Seoul, Korea, in 1978, the MS degree in electrical and electronics engineering from Korea Advanced Institute of Science and Technology, Seoul, Korea, in 1980, and the PhD degree in electrical engineering from Stanford University, Stanford, CA, in 1989. From 1989 to 1991, he was with Cadence Design Systems, Inc. In 1991, he joined the faculty of the School of Electrical Engineering and Computer Science, Seoul National University. His primary research interests include computer-aided electronic systems design, low-power systems design, and computer architecture. He is a senior member of the IEEE.