Per-Operation Reusability Based Allocation and Migration Policy for Hybrid Cache

Minsik Oh, Kwangsu Kim, Duheon Choi, Hyuk-Jun Lee, and Eui-Young Chung, *Member, IEEE*

Abstract—Recently, a hybrid cache consisting of SRAM and STT-RAM has attracted much attention as a future memory by complementing each other with different memory characteristics. Prior works focused on developing data allocation and migration techniques considering write-intensity to reduce write energy at STT-RAM. However, these works often neglect the impact of operation-specific reusability of a cache line. In this paper, we propose an energy-efficient per-operation reusability-based allocation and migration policy (ORAM) with a unified LRU replacement policy. First, to select an adequate memory type for allocation, we propose a cost function based on per-operation reusability – gain from an allocated cache line and loss from an evicted cache line for different memory types – which exploits the temporal locality. Besides, we present a migration policy, victim and target cache line selection scheme, to resolve memory type inconsistency between replacement policy and the allocation policy, with further energy reduction. Experiment results show an average energy reduction in the LLC and the main memory by 12.3 and 21.2 percent, and the improvement of latency and execution time by 21.2 and 8.8 percent, respectively, compared with a baseline hybrid cache management. In addition, the Energy-Delay Product (EDP) is improved by 36.9 percent over the baseline.

Index Terms—Hybrid cache, energy efficient allocation and migration policy, unified LRU replacement policy, per-operation reusability

1 Introduction

As the number of cores increases in processors, the size of shared last-level caches (LLCs) is increased to capture the working set of data-sharing applications. However, the large size of LLCs increases energy consumption. Various cache management schemes have been proposed to reduce their energy consumption while improving the utilization. Despite these efforts, SRAM-based LLCs suffer from substantial leakage power as memory capacity rapidly is increased. To solve this problem, a STT-RAM (Spin Transfer Torque RAM) based LLCs are proposed.

The recent emerging technology, STT-RAM, stores data by using a Magnetic Tunnel Junction (MTJ) consisting of a reference layer and a free layer with one transistor. A bit is stored by changing the magnetic orientation of the free layer. It does not consume any additional energy for preservation. Due to this structure of STT-RAM, the technology is approximately 4 times higher in density and 10 times lower in leakage power, compared with SRAM. However, higher write latency and write energy consumption should be properly addressed to efficiently use STT-RAM for cache memory [3], [30]. In order to use STT-RAM as a cache memory, many researches have explored methods to alleviate the write penalty. However, these efforts still suffer from write penalty by allocating unavoidable data as in the fill operations of reusable data. For this, a SRAM-STT-RAM hybrid caches have been proposed to minimize the penalty due to the write operations on the STT-RAM. The hybrid structure has shown effectiveness in reducing energy consumption by allocating data with write-oriented accesses to SRAM in order to reduce write penalties.

To efficiently utilize the hybrid cache memory and optimize the energy consumption, a cache line should be carefully allocated either on SRAM or STT-RAM. Most prior research has focused on allocation policies-based on the access pattern analysis [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18]. In these schemes, the memory type is determined based on the current request operation (read or write) and the expected write count to the cache line after allocation. These techniques may be effective in reducing LLC energy consumption caused by writes to the STT-RAM. However, they do not fully take into account the benefits and losses caused by allocating a cache line on a particular type of cache memory. That is, a total cost of access is dependent on the future access to the evicted cache line by being data allocated, as well as the operation type and the reusability of a cache line in the selected memory. We call this *per-operation reusability* in this paper. To the best of our knowledge, there has been no hybrid cache management scheme which considers per-operation reusability for energy efficiency under the unified LRU replacement policy. In addition, the fundamental limitation of an allocation-only policy is that it only performs static allocation of a cache line.

To dynamically reallocate the data between SRAM and STT-RAM cache memory, data migration techniques are also

- *M. Oh, K. Kim, D. Choi, and E.-Y. Chung are with the School of Electrical and Electronic Engineering, Yonsei University, Seoul 03722, Korea. E-mail: stom4241@gmail, {nep, cdh0527, eychung}@yonsei.ac.kr.*
- *H.-J. Lee is with the School of Computer Science and Engineering, Sogang University, Seoul, Korea.. E-mail: hyukjunl@sogang.ac.kr.*

Manuscript received 1 Apr. 2019; revised 20 Aug. 2019; accepted 15 Sept. 2019. Date of publication 27 Sept. 2019; date of current version 13 Jan. 2020. (Corresponding author: E.-Y. Chung.)
Recommended for acceptance by T. Li.
Digital Object Identifier no. 10.1109/TC.2019.2944163

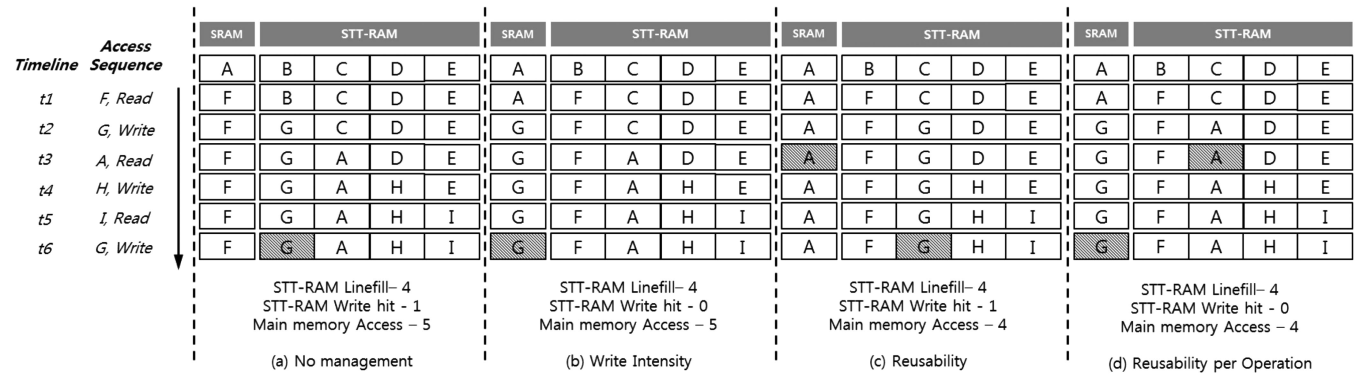

Fig. 1. Motivation example. (a) The data allocation based on a unified LRU replacement policy without additional management schemes. (b) The allocation method solely based on the write intensity. (c) The allocation method that preserves data with high reusability. (d) The allocation method which considers both reusability and operation type.

studied [7], [8], [9], [19], [20]. For these techniques, two important factors of interest are when and what to migrate. In general, when write intensive data is allocated in the STT-RAM, it is migrated to SRAM to reduce energy consumption from frequent writes to the STT-RAM cache line. However, since data migration consumes additional energy in the process of rewriting data, frequent migration can adversely affect energy con-sumption. Prior works considered the benefit from migration without thoroughly analyzing the side-effect of unnecessary migration. When selecting a target cache line to be migrated, they focus on identifying write intensive data to migrate it from the STT- RAM to the SRAM. However, there are limitations in predicting write intensity in the non-inclusive writeback cache as write request behavior is often dynamically changed by the state of upper-level caches. To improve the prediction accuracy, more sophisticated metrics should be considered other than write intensity.

Finally, prior research using both allocation and migration policies in the hybrid cache structures do not adopt a unified replacement policy across SRAM and STT-RAM partitions, which may provide optimization of replacement decisions in global-scope using reuse-distance whereas existing schemes perform memory-specific replacement policies in local-scope. However, when applying an allocation policy to a hybrid cache using a unified LRU replacement policy, the memory type selected by an allocation policy may differ from the memory type of victim cache line, which forced prior works to use memory specific replacement policies.

In this paper, we propose novel allocation and migration policies to improve energy efficiency in a hybrid SRAM-STT-RAM caches using a unified replacement policy. Main contributions are as follows:

- First, we propose a per-operation reusability-based cost function for the cache line allocation policy to estimate expected future energy consumption, which considers both hits from the allocated cache lines and misses from the evicted cache lines.
- Second, we identify and address key challenges in cache management in the hybrid cache under a unified LRU replacement policy for SRAM and STT-RAM partitions.
- Finally, we propose an energy-efficient migration policy which consists of a victim and target cache line selection scheme. The victim cache line selection scheme chooses a line to minimize migrations while avoiding eviction of useful data. Target cache line selection scheme chooses a cache line to migrate by considering per-operation reusability after migrations.

The rest of the paper is organized as follows. Section 2 discusses motivation for our techniques. Section 3 presents the proposed allocation and migration policy in detail. Then, we analyze experiment results in Section 4. Section 5 reviews prior hybrid SRAM-STT-RAM cache management techniques. Finally, we draw conclusions in Section 6.

2 Motivation

To manage the hybrid last-level cache efficiently, prior works focused on accurately estimating write intensity as write operations on the STT-RAM significantly impact the energy consumption of the overall system. Therefore, prior studies have researched allocation policies for selecting memory by predicting expected write accesses in workloads. However, they only took into account the energy differences of each memory, without considering per-operation reusability.

Fig. 1 depicts a motivational example to show the benefit of the proposed technique. In the figure, a row represents data in the cache blocks of a single cache set at a given time. Each row indicates data allocation as the result of a request (data and operation type specified on the left with timeline) for the same cache line. We assume that cache blocks in a set are sorted in the beginning as the leftmost cell represents an LRU position. In addition, cache allocates 1-way to SRAM and 4-way to STT-RAM for simplicity. A square box indicates a cache line whose data is shown in an uppercase alphabet and a hatched box represents a hit upon an access.

As the policy in Fig. 1a ignores both reusability and write intensity, the cache line A is evicted and then reallocated at $t1, t3$. The cache line G is allocated in the STT-RAM and consumes unnecessary write energy at $t2, t6$. Allocation based on the write intensity, which maps data with high write intensity to SRAM, in Fig. 1b allocates the write intensive cache line G on SRAM at $t2$ and benefits from a hit for a subsequent write of G. However, the cache line A is evicted and reallocated at $t1, t3$ (incurring an external memory access) because it does not consider reusability of the cache line A. An allocation based on reusability in Fig. 1c assumes that the reusability of a cache line can be ideally determined in advance. It further

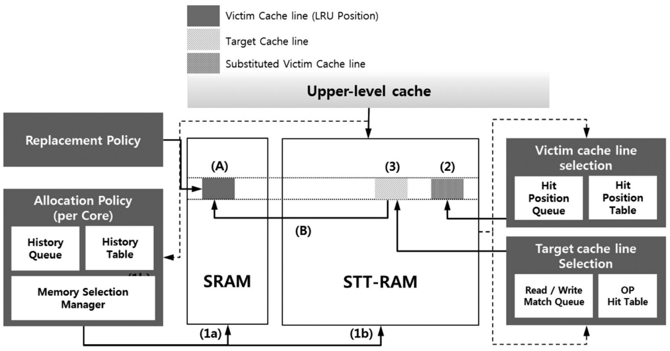

Fig. 2. Overall proposed scheme including architecture and management techniques.

reduces main memory accesses by preserving the cache lines A and G, which are expected to be re-used at $t3, t6$, respectively. While the hit count is increased, however, writes to the STT-RAM are increased as the write intensive cache line G is allocated to STT-RAM at $t2$ without considering its future operation type. From these examples, it is clear that both the reusability and the operation types need to be considered to efficiently utilize the hybrid cache. The proposed method in Fig. 1d reduces not only main memory accesses but also writes to the STT-RAM by forecasting following operations. Cache line A is preserved in SRAM until a write intensive request for the cache line G is issued. Because of its high write intensity, the cache line G is allocated in SRAM, which triggers migration of the cache line A to the victim cache line occupied by data C in STT-RAM at $t2$. By considering the reusability and operation type of cache line A and G, the proposed method (d) minimizes writes to STT-RAM and external memory accesses.

3 Hybrid Cache Management Scheme

3.1 Overview

A baseline hybrid SRAM-STT-RAM cache partitions cache way conforming to the density ratio of SRAM and STT-RAM (e.g., 1:4). Fig. 2 presents the overview of the proposed hybrid cache architecture and management scheme which consists of (1) an allocation policy, (2) a victim cache line selection scheme, and (3) a target cache line selection scheme for data block migration. The cache hierarchy consists of private L1, L2 caches and a shared L3 cache with a unified LRU replacement policy.

In our hybrid LLC, the proposed allocation policy determines the memory type (e.g., SRAM or STT-RAM) by estimating current and future energy consumption upon allocating data on each memory type. If the allocation policy and the victim cache line under the replacement policy point to the same memory type (A and 1a in Fig. 2), the victim line chosen by the replacement policy is replaced. If the memory types pointed by the two are different (A and 1b in Fig. 2), data should be allocated to a target memory chosen by the allocation policy. After a target cache line in one memory is selected by the allocation policy, its data is migrated to a victim cache line in another memory. This process incurs additional energy and latency for migration. To avoid unnecessary frequent migration, we propose a victim cache line selection scheme to search an alternative cache line in the target memory which is not to be likely reused (2). Despite this effort, there may be situations where migration should be performed. In this case, we need to determine a target cache line where we allocate incoming data. For this, we propose a target cache line selection scheme that determines a cache line by considering both the expected operation after migration and the memory characteristics (3). Then, its data is migrated to the victim cache line (B) which was selected by replacement policy (A) so that a target cache line can be used for new incoming data.

In the allocation policy, we maintain a history queue and history table, which will be discussed in the following section, per core to record access patterns per each workload, and a memory selection manager to handle cache line allocation. In victim and target cache line selection, a global queue and global table for hit position or operation matching are maintained as we use the unified LRU replacement policy across applications.

3.2 Allocation Policy

The allocation policy determines the type of memory to allocate a cache line on a cache miss. The proposed allocation policy takes into account the energy consumption from the operation of a current request and expected operations in the future as a result of allocating a cache line. It collects cache access statistics for energy computation and decides to allocate on SRAM or STT-RAM, or bypass to main memory.

First, to collect and organize statistics of cache accesses, we maintain a history queue and a history table per core.

- *History Queue*: A history queue collects cache access patterns for each core. The history queue is a first-in-first-out (FIFO), consisting of 4-bit entries where each bits represents bypass (B), memory type (S/ST), hit/miss (H/M) and operation type (R/W) respectively.
- *History Table: A h*istory table maintains cache access statistics to provide a basis for selecting the memory to allocate. The history table stores the number of occurrences for each pre-defined event by counting them from the history queue. The events indicate all possible cases that could occur.

History queues and history tables are managed together in a moving window to reflect only recent access patterns. When a request arrives, a new entry is enqueued into the history queue and a count of its corresponding event for the request is incremented in the history table. Since the history queue is a FIFO, the oldest entry is dequeued and its corresponding event count decremented in the history table. Fig. 3 illustrates a processing example of history queue and table where a writeback to LLC is allocated to STT-RAM. In this case, B, S/ST, H/M and R/W bits would be set to 0 (no bypass), 1 (STT-RAM), 1 (miss), and 1 (write) in the history queue with the counter for STT-RAM/Miss/Write entry incremented in the history table. At the same time, the oldest event denoted as 1000 in the history queue is evicted with the counter for Bypass/Read entry decremented in the history table.

After collecting the statistics, the allocation policy calculates expected energy consumption using information from the history table for different allocation choices and selects the best memory type or decides to bypass. The detailed implementation of the policy is shown in Algorithm 1.

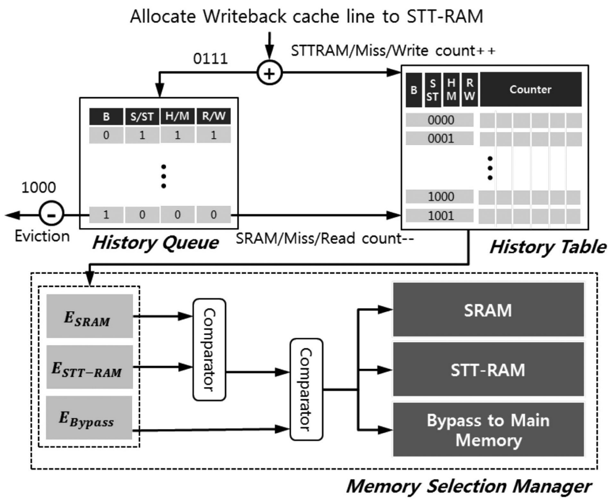

Fig. 3. Allocation policy with History Queue and History Table.

Algorithm 1. The Algorithm of Allocation Policy

```
Input: $\mathrm{Req_{curr}}$
Input: $\mathrm{Rate_{MOH}, E_{MOH}}$
        where $\mathrm{M = \{T, S.ST, MEM, -\}, O = \{R, W, -\}, H = \{H, M, -\}}$
Parameter: $\mathrm{E_{SRAM}, E_{STT-RAM}, E_{Bypass}}$, $\mathrm{weight_a}$, $\mathrm{weight_b}$
Output: Allocation State
 1. Pre-setting the input Ex
 2.     $\mathrm{E_{XRH} = E_{TR} + E_{XR}}$,
 3.     $\mathrm{E_{XRM} = E_{TR} + E_{TW} + E_{XW} + E_{XR} + E_{MEM}}$,
 4.     $\mathrm{E_{XWH} = E_{TR} + E_{XW}}$,
 5.     $\mathrm{E_{XWM} = E_{TR} + E_{TW} + E_{XW}}$, where x is memory type
 6. Calculate the expected energy consumption at each memory
 7.     $\mathrm{E_{Allocation\,State} = E_C + weight_a * E_H + weight_b * E_M}$,
 8.     $\mathrm{E_C}$: energy consumed by the current request
 9.     $\mathrm{E_H}$: expected energy consumed by subsequent hit on the
        allocated cache line
10.  $\mathrm{E_M}$: expected energy consumed from subsequent miss due
     to the evicted cache line.
11. Decide memory type
12. if $\mathrm{E_{STT-RAM} > E_{SRAM}}$ then
13.     if $\mathrm{E_{SRAM} > E_{Bypass}}$ then Allocation State is Bypass to
        Main Memory
14.     else then Allocation State is SRAM
15. else then
16.     if $\mathrm{E_{SRAM} > E_{Bypass}}$ then Allocation State is Bypass to
        Main Memory
17.     else then Allocation State is STT-RAM
18. end if
```

Inputs to the algorithms are $\mathrm{Req_{curr}, Rate_{MOH}}$, and $\mathrm{E_{MOH}}$ where subscript MOH represents memory type (M), operation (O) and hit/miss (H). M may be Tag directory/SRAM/STT-RAM/Main Memory/none (T/S/ST/MEM/-), and O and H may be read/write/none (R/W/-) and hit/miss/none (H/M/-), respectively. The subscript of Rate and E are constructed by combining each component for M, O and H. $\mathrm{Req_{curr}, Rate_{MOH}}$, and $\mathrm{E_{MOH}}$ denote a current request, a rate or the amount of energy consumption for the state of MOH, respectively.

$\mathrm{Rate_{MOH}}$ is obtained from counter values stored in the history table and are computed as the sum of counter numbers in the entries of the table which constitute the cases corresponding to MOH. $\mathrm{E_{MOH}}$ indicates the amount of energy consumed for the cases corresponding to MOH where it represents a read or write operation in cache and memory. Again, $\mathrm{E_{MOH}}$ is calculated as the sum of energy consumed by operations occurring in each case, which is described in lines 1-5. For example, $\mathrm{E_{SRM}}$ (SRM stands for a read miss on SRAM) is derived by summing the energy for searching and updating a tag memory, accessing main memory, updating the data memory of cache. Energy consumption for each operation in cache is pre-determined using Cacti /NVSim and main memory access energy is determined using the configuration in Table 1. To minimize the overhead of calculating the cost function, the energy values are rounded to 2 decimal places after normalized by the largest energy value and then treated as integers. All input values are provided to the memory selection manager in the beginning.

Next, it calculates expected energy consumption for the current request in each memory. It reflects all future energy consumption from subsequent hits and misses due to the current request and is expressed in line 6. $\mathrm{E_{Allocation\,State}}$ denotes the expected total energy consumption for three cases: SRAM, STT-RAM and Bypass to Main Memory. The first $(\mathrm{E_c})$ is the energy consumed by allocating the data of the current request. The second $(\mathrm{E_H})$ is the expected energy consumed due to subsequent hits on the allocated cache line in the future. The last item $(\mathrm{E_M})$ represents the energy consumption from subsequent misses due to the evicted cache line from the current request. Weights represent the influence of each item over $\mathrm{E_{Allocation\,State}}$ as shown in lines 7-9.

$$\begin{aligned}E_{SRAM} &= E_{SRM}[E_{SWM}]\\ &\quad + weightax(Rate_{RH}xE_{SRH} + Rate_{WH}xE_{SWH})\\ &\quad + \ weight_bx(Rate_{SRH}xE_{SRM} + \ Rate_{SWH}xE_{SWM})\end{aligned} \tag{1}$$

$$\begin{aligned}E_{STT-RAM} &= E_{STRM}[E_{STWM}]\\ &\quad + weight_ax(Rate_{RH}xE_{STRH} + Rate_{WH}xE_{STWH})\\ &\quad + \ weight_bx(Rate_{STRH}xE_{STRM} + Rate_{STWH}xE_{STWM})\end{aligned} \tag{2}$$

$$\begin{aligned}E_{Bypass} &= E_{TR} + \ E_{MEM}\\ &\quad + \ weight_ax(Rate_{RH}xE_{SRM}[E_{STRM}] + Rate_{WH}xE_{SWM}[E_{STWM}])\\ &\quad + \ weight_bx(Rate_{SRH}xE_{SRH}[Rate_{STRH}xE_{STRH}]\\ &\quad + Rate_{SWH}xE_{SWH}[Rate_{STWH}xE_{STWH}])\end{aligned} \tag{3}$$

Based on this, the cost function for three states (SRAM, STT-RAM, or Bypass to Main Memory) are calculated as shown in Eq. (1), (2), (3). In the equations for allocating to SRAM and STT-RAM (Eqs. (1), (2)), the first item, $\mathrm{E_{MOH}}$, is set according to the current request operation (O) for memory (M). This item can be for a read or a write (in a square bracket) according to the operation of a current request. The following two items are the expected energy consumption from read or write hits on the cache line assuming it is allocated. They are calculated by multiplying the probability of read/write hit, $\mathrm{Rate_{RH}/Rate_{WH}}$, with corresponding $\mathrm{E_{MH}}$. The last two indicate the expected energy consumption from misses due to cache line eviction. Note that there is a difference between $\mathrm{Rate_{OH}}$ and $\mathrm{Rate_{MOH}}$ factors for expected energy consumption for hits and misses. For the hit case, once a cache line is allocated on either SRAM or STT-RAM, it

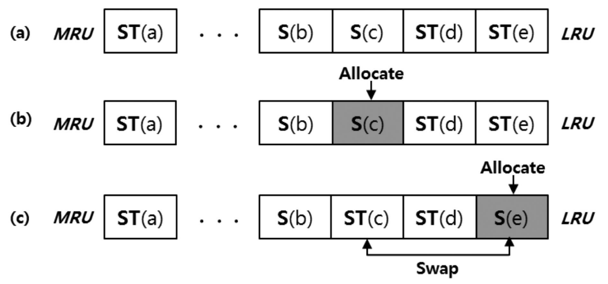
(a) MRU ST(a) . . . S(b) S(c) ST(d) ST(e) LRU
Allocate
(b) MRU ST(a) . . . S(b) S(c) ST(d) ST(e) LRU
Allocate
(c) MRU ST(a) . . . S(b) ST(c) ST(d) S(e) LRU
Swap

Fig. 4. An example of victim cache line selection to allocate a cache line in SRAM under the unified LRU replacement policy.

will stay in the cache until the unified LRU algorithm replaces it. This guarantees the lifetime to be proportional to the size of the whole set including both SRAM and STT-RAM. Thus, the probability of a hit will be proportional to the global hit rate, $\mathrm{Rate_{RH}}$ or $\mathrm{Rate_{WH}}$. For subsequent misses from cache line eviction, the probability is proportional to the hit rate of specific memory type (e.g., $\mathrm{Rate_{SRH}}, \mathrm{Rate_{SWH}}, \mathrm{Rate_{STRH}}, \mathrm{Rate_{STWH}}$). The algorithm also computes the expected total energy consumed in bypassing to prevent allocating a cache line with low reusability in Eq. (3). The energy consumption for bypassing is similar to ones for cache memory except for all hit and miss energy penalties being switched to miss and hit energy penalties respectively. If a cache line is determined to be un-allocated (bypassed), its request reads tag memory and directly accesses external memory $(\mathrm{E_{TR}} + \mathrm{E_{MEM}})$. In this case, if request to the same data is received in the future, it results in a miss. The energy consumption due to future misses is proportional to the probability of hits and miss penalty as the cache line is not allocated. Meanwhile, as the useful data is not evicted due to bypass, future accesses to the cache memory may observe additional hits due to the un-evicted cache line. This translates into last two terms in $\mathrm{E_{Bypass}}$. Like the previous equations, it can be substituted by the expression in the square bracket depending on the memory type to allocate.

Using the calculated expected total energy, the memory type is determined in lines 10-17. In this process, the memory with the smallest cost is compared with the energy consumption for the bypass case. As a result, the memory type or bypass is selected by comparing the expected total energy for bypass with that of the best memory type.

To calculate the cost function, additional circuitry is needed: 4×18-bit adders and 4×18-bit multipliers for each allocation state. First, for each allocation state, the multiplication operations of $Rate_X$ and E_X are computed in parallel. Then the results of the multiplication operations are added in three steps. The cost functions are calculated at the same time as the tag access. And the time of calculating cost function is hidden since the it is the same at clock cycles compared to the tag access time in 1 GHz memory system.[1] Similarly, the energy consumption for calculating the cost function for the cache access is also marginal.

1. A 16-bit multiplier consumes 424μW while its dealy is 528ps and a 16-bit adder consumes 74μW while its delay is 467ps at 45nm technology with a supply voltage($\mathrm{V_{dd}}$) of 1.1V [31]. The power delay product of the additional circuitry is under 3.1 e^{-12}J even we used 18-bit adders and multipliers, which is marginal.

3.3 Victim Cache Line Selection

In the previous section, we describe the proposed allocation policy, which decides on bypassing or the memory type. As the allocation policy makes a decision without any knowledge of the candidate victim cache line, there may be inconsistencies between the memory type selected by the allocation policy and the memory type that a selected victim cache line at the LRU position belongs to.

Fig. 4 illustrates the problem. In the figure, a rectangle represents a cache line, and uppercase alphabets and alphabets in parentheses denote the memory type and data, respectively. Fig. 4a shows a cache set sorted from an MRU position to an LRU position. As we use a unified LRU algorithm for the replacement, a line marked with ST(e) is chosen for a victim cache line. If a new cache line needs to be allocated to SRAM, a sub-optimal solution may be selecting a cache line S (c) in the SRAM near the LRU position as in Fig. 4b. However, in this case, a victim cache line is evicted before reaching the LRU position. Alternatively, we can swap data *c* in the cache line of SRAM with data *e* in the cache line of STT-RAM and reset positions in the LRU stack as in Fig. 4c. Then we can replace S(e) using a unified LRU replacement algorithm. This process is called migration. However, if data migration occurs frequently, it would consume substantial energy.

The proposed victim cache line selection scheme aims to prevent unnecessary data migration and selects a victim cache line from the memory determined by the allocation policy whereas it conforms to a unified LRU replacement as much as possible. To measure the reuse distance of cache lines for victim cache selection, we maintain a global Hit Position Queue (HPQ) which records hit positions in the LRU stack and a Hit Position Table (HPT) which counts the number of hits per position. A victim cache line is selected by utilizing the HPT.

Fig. 5 illustrates the victim cache line selection process.

When a cache hit occurs, it checks its hit position within the LRU stack, enqueues the position into HPQ, and increases the counter of the associated position in HPT. Since HPQ is a FIFO (a moving window), the oldest entry is removed, and its associated counter is decremented in the HPT. In Fig. 5, a cache hit occurs at 7th position of the LRU stack and both HPQ and HPT are updated accordingly. In addition, the oldest hit position, number 10, is removed from HPQ and the associated counter value in HPT is decremented.

HPT provides a distribution for reuse distance, and it helps to choose candidate victim cache lines within LRU stack. In other words, the distribution helps to distinguish highly reusable cache lines from less reusable ones so that we can choose a victim cache line from the latter. We describe the victim cache line selection scheme in Algorithms 2 and 3.

Algorithm 2 describes a method to determine victim line candidates. It takes HIT as an input. HIT represents an array of hit counts in the HPT. HIT_x represents a particular counter where *x* may be either *assoc* or index, where assoc represents the maximum associativity value. $\mathrm{HIT_{Threshold}}$ is a constant parameter that is determined experimentally. SUM represents the sum of hit counts from the LRU position to the current index. The output $\mathrm{Line_{victim}}$ is the position in the sorted LRU stack, which marks the boundary of victim line candidates. From $\mathrm{Line_{victim}}$ to LRU position, cache lines can be chosen as a victim cache line as they exhibit low reusability.

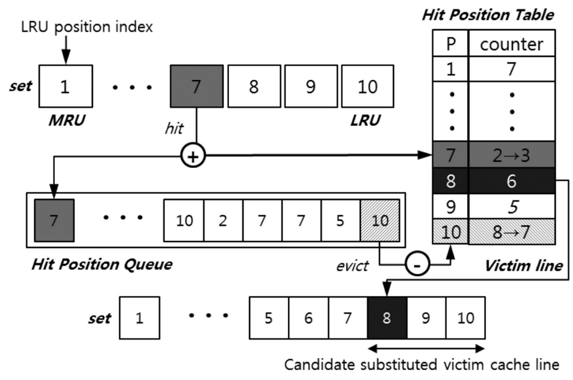

Fig. 5. Hit Position Queue and Table management.

Algorithm 2. Finding a range for the victim cache line

```
Input: HIT
Parameter: $HIT_{Threshold}$, SUM
Output: $Line_{victim}$
1. Find a lower index for the victim line in LRU stack
2. for index = assoc→1 do
3.     if $(SUM < HIT_{Threshold} \& HIT_{index} < HIT_{assoc})$ then
       $SUM+ = HIT_{index}$
4.     else $Line_{victim}$ = index, and then break
5. end for
```

Algorithm 3. Selecting a victim cache line

```
Input: $memType_{alloc}$, $memType_{LRU}$, $Line_{victim}$
Parameter: $memType_{index}$
Output: victim cache line
1.  Select a victim cache line when a miss occurs
2.  if $memType_{alloc}$ ! = $memType_{LRU}$, then
3.    for index = assoc→1 do
4.      if ($memType_{index}$ == $memType_{alloc}$ & the cache line is
        invalidated),
5.      then select index^th cache line for a victim
6.    end for
7.    if it cannot find invalidated cache line, then
8.      for index = assoc→Linevictim do
9.        if mem $Type_{index}$ == $memType_{alloc}$, then select
          index^th cache line for a victim
10.     end for
11.   end if
12.   if it cannot find a victim cache line, then do migration.
13. else then select a cache line in the LRU position for a
    victim.
```

To find $Line_{victim}$, algorithm evaluates SUM and HIT_{index} of each position from LRU (lines 2-5). HIT_{index} is the hit count of a current position pointed by index and it should not exceed the hit count of the LRU position to be chosen as a candidate victim cache line. In addition, SUM should be less than the pre-defined $HIT_{Threshold}$. These two conditions prevent the selection of highly reusable cache lines as victim. Both conditions guarantee cache lines between $Line_{victim}$ and LRU position to be a good candidate for replacement.

Algorithm 3 describes the victim cache line selection process. $memType_{alloc}$ and $memType_{LRU}$ denote the cache memory type selected from allocation policy and LRU position, respectively. $memType_{index}$ represents the cache memory type of a cache line at the position *index* of the LRU stack.

Algorithm 3 compares the memory type determined from allocation policy and the memory type of the cache line at the LRU position (line 2). If two memory types match, the cache line at the LRU position is selected as a victim (line 13). Otherwise, we search for a victim cache line that matches the memory specified by the allocation policy. For this, it searches for an invalidated cache line (lines 3-6). If it is not found, we scan the cache line from the LRU position to $Line_{victim}$ that matches the memory type determined by the allocation policy (line 7-11). If no victim candidate is found, we perform migration (line 12).

The victim cache line selection scheme is similar to protecting distance-based policy (PDP) [23] as victim cache line selection considers reuse-distance. In this paper, we use conventional LRU as a replacement policy. However, other replacement policies can be integrated with the victim cache line selection scheme with only minor modification.

3.4 Target Cache Line Selection for Migration

When the memory type of a victim cache line does not match the memory type selected by the allocation policy, we perform migration. In migration, we move the data of the target cache line selected by the target cache line selection scheme to the victim cache line. After the move, the target cache line is replaced with a new incoming data.

The objective of target cache line selection is to minimize the energy consumption of LLC after migration. Upon selecting a cache line to migrate from the target memory, if the victim cache line is SRAM, it is beneficial to select a cache line which is most likely to have more write accesses. Conversely, if the victim cache line is STT-RAM, a cache line with the lowest possibility of write accesses should be selected.

To predict which operation will be occur at a specific cache line in the future, we implement a read/write match queue and operation hit table (OHT). They record the frequency of two subsequent operations (e.g., read or write) when a cache hit occurs. The target cache line selection scheme chooses a cache line to minimize energy using this information.

Fig. 6 shows how OHT is managed. To record the last operation for each cache line, one bit is used per each cache line. When a cache is accessed for either read or write, it reads the last operation bit and records it in its corresponding read/write match queue. At the same time, OHT containing operation match counters is updated. Since read/write match queues are FIFOs, they are maintained as a moving window. Upon accessing a cache line, the type of a match queue is determined by the last operation to that cache line. Then, the value inserted into a match queue is determined by the type of the last and current operation. If they are identical, 1 is inserted and 0 otherwise. In Fig. 6, the first example shows a case where the last access was a read and a write access occurs to the same line. Then the value of '0' is inserted to the read match queue, indicating that an opposite operation (write) occurs. In addition to updating the match queue, associated counters of OHT are also updated. A counter named as 'x after y' records how many times operation x occurs after operation y during a

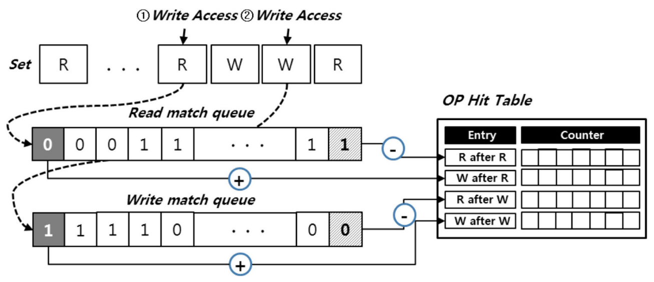

Fig. 6. OP Hit Table management.

Algorithm 4. The algorithm of target cache line selection

```
Input: $C_{RR}, C_{RW}, C_{WR}, C_{WW}, OP_{index}, memType_{victim}$
Output: target cache line
1.  Select target cache line when miss is occurred
2.  if $memType_{victim}$ is SRAM then
3.      for index = assoc $\rightarrow$ 1 belong to the STT-RAM, do
4.          if ($C_{WR} > C_{WW}$&$OP_{index}$ is READ), select index^th target cache line
5.          if ($C_{WR} < C_{WW}$&$OP_{index}$ is WRITE), select index^th target cache line
6.      end for
7.      if target cache line was not found then
8.          for index = assoc$\rightarrow$1, do
9.              if index^th cache line is belong to the STT-RAM, then select index^th target cache line
10.         end for
11.     end if
12. else then
13.     for index = 1$\rightarrow$assoc belong to the SRAM, do
14.         if ($C_{RR} > C_{RW}$&$OP_{index}$ is READ), select index^th target cache line
15.         if ($C_{RR} < C_{RW}$&$OP_{index}$ is WRITE), select index^th target cache line
16.     end for
17.     if target cache line was not found then
18.         for index = 1$\rightarrow$assoc, do
19.             if index^th cache line is belong to the SRAM, then select index^th target cache line
20.         end for
21.     end if
22. end if
```

window. Thus, in the first example, a counter with 'W after R' is incremented. In the second example of Fig. 6, a write access occurs to the cache line whose last access was a write. This increases the counter with 'W after W'. Only four global counters are managed. Each time when a new operation is inserted into the read/write match queue, the oldest operation is removed from the FIFO, causing its associated counter to be decremented. Using the information of OHT, a target cache line is selected. The process of target cache line selection is described in Algorithm 4.

C_{xy} denotes a counter for x after y where x and y represent an operation (R/W). OP_x represents the operation at position x of the LRU stack, and $memType_{victim}$ represents the memory type of the cache line selected by the proposed victim cache line selection scheme.

When a victim cache line with a matching memory type is not found by the victim cache line selection scheme, a target cache line should be selected. In case of migrating to SRAM in which a victim cache line resides (line 2), the algorithm search es a cache line from the LRU position to the MRU position until it finds one belonging to STT-RAM (line 3). And then after, STT-RAM cache line that is likely to be written is selected as a target cache line as long as it satisfies the condition in lines 4-5. For instance, if the last operation of a chosen cache line was a read and C_{WR} is greater than C_{WW}, it is selected as a target cache line because a cache line with a recent history of a read is more likely to be written again once it is moved to SRAM than one with a history of a write. However, if a target cache line is not found through the steps described in line 4-5, it chooses the first cache line belonging to STT-RAM as a target cache line while searching from the LRU position to the MRU position (lines 7-11).

On the other hand, when migrating to STT-RAM, we find a cache line in SRAM that is likely to be read. Since a cache line has either been read or written, one with higher chance of being read in the future will be selected by comparing C_{RR} and C_{RW} (lines 13-16). Note that the searching direction of the LRU stack is opposite when finding a target cache line. Since the cache line at the MRU position will be replaced later than the cache line at the LRU position, it has more opportunity of a read hit for the migrated cache line. It also reduces write energy consumption in the STT-RAM caused by allocating a new cache line from early eviction of the migrated cache line. For this reason, if a target cache line is not found, it searches a target cache line belonging to SRAM from the MRU position to the LRU position (lines 17-21). When the target cache line is determined, data migration is performed as follows. First, data of the victim cache line and the target cache line are read. Then, data of the victim cache line is written back to the main memory or invalidated when it is clean. Simultaneously, data of the target cache line is buffered until the victim cache line is available for a write. Lastly, the data from the target cache line is written to the victim cache line. New data is allocated to the target cache line after data migration is completed.

4 Experiment Evaluation

4.1 Evaluation Environment

In this paper, we present the SRAM-STT-RAM hybrid LLC management scheme including the allocation and migration policy based on per-operation reusability. We use GEM5 simulator [24] for evaluation. The detailed system environment is described in Table 1.

The cache architecture consists of L1/L2/L3 caches with writeback and non-inclusive policy. Most works allocate 1 MB per core in SRAM LLC. For a fair comparison, we use the same configuration. And since the density of STT-RAM is 4 times higher than the density of SRAM, we allocate 4 MB and 16 MB for the SRAM-only and STT-RAM-only LLC configuration, respectively. In case of the hybrid LLC, we use 2 MB SRAM and 8 MB STT-RAM to maintain the same area cost. It is managed by the unified LRU replacement policy and treated like a single cache memory. To calculate an accurate timing and energy consumption, each level of caches and its tag directory are modeled using Cacti [27] and NVSim [28] for a 45nm process technology, as summarized in Table 2.

TABLE 1
Simulation Configuration

CPU model	Out-of-Order, X86, 2GHz
	Cache Configuration
L1	2-way, 32 kB, 2 MSHRs 64B line
L2	8-way per core, 128 kB, 8 MSHRs, 8 writeback entries, 64B line
L3	16-way (SRAM-only 4 MB, STT-RAM-only 16 MB), 10-way (Hybrid Cache 10 MB: 2-way SRAM, 8-way STT-RAM), 10 MB, 16 MSHRs, 16 writeback entries, 64B line
Main Memory	DDR3_1600_x64, 2 GB, 800 MHz, 64-bit data width, 2 ranks, 8 banks, Open page policy, FIFO scheduling

TABLE 2
Characteristics of Cache Memory

	Tag	SRAM	STT-RAM
Read Latency (ns)	1.791	2.189	3.774
Write Latency (ns)	1.791	2.189	13.247
Read Energy (nJ)	0.014	0.076	0.084
Write Energy (nJ)	0.014	0.076	0.649
Leakage Power (mW)	9.618	30.053	9.011

TABLE 3
Benchmark Classification Based on Write Intensity

Classification	Benchmarks
High WPKI (H)	bzip2, mcf, sjeng, lbm, libquantum, gobmk, soplex, milc, zeusmp, leslie3d
Low WPKI (L)	hmmer, povray, h264ref, namd, calculix, GemsFDTD, omnetpp, sphinx3, swaptions, streamcluster, freqmine, canneal

TABLE 4
Workload Mix

Name	Workloads
H4-1	bzip2, lib, gobmk, lbm
H4-2	mcf, sjeng, soplex, zeusmp
H3L1-1	bzip2, mcf, sjeng hmmer
H3L1-2	lbm, leslie3d, povray, gobmk
H3L1-3	soplex, lbm, sjeng, calculix
H2L2-1	milc, sphinx3, libquantum, streamcluster
H2L2-2	bzip2, zeusmp, omnetpp, namd
H2L2-3	freqmine, swaptions, mcf, leslie3d
H1L3-1	povray, canneal, mcf, leslie3d
H1L3-2	milc, freqmine, GemsFDTD, canneal
L4-1	calculix, GemsFDTD, h264ref, sphinx3
L4-2	Hmmer, omnetpp, streamcluster, swaptions

For evaluation, we use SPEC 2006 benchmark suites [25] and PARSEC benchmark suites [26]. Considering different memory characteristics for SRAM and STT-RAM, we classify workloads based on write intensity as shown in Table 3. The write intensity is measured as write counts per kilo-instructions (WPKI). Write operations represent all possible write operations that occur in LLC. Applications are classified as high or low according to their write intensity. By mixing benchmark applications, we generate L3 workloads as listed in Table 4. Each benchmark application runs until it executes 500M instructions, and the experiment is conducted as a cold start.

We propose allocation and migration policy with consideration of per-operation reusability. In our experiment, the overall overhead of calculating cost function from a performance nor a power perspective is considered by referencing the paper [31].

We use a hybrid cache management scheme using a unified LRU replacement policy without any allocation policy as a baseline. In addition, we evaluate both SRAM-only and STT-RAM-only LLC along with recently proposed schemes: PTHCM [15] and APM [7], which use both allocation and migration techniques to improve hybrid LLC energy efficiency.

4.2 Impact of Allocation Policy

In this section, we evaluate the proposed allocation policy. It aims to allocate data efficiently to reduce LLC energy consumption and improve data reusability. In Fig. 7, we show experimental results from applying per-operation reusability (OR) in terms of LLC dynamic power and hit counts.

Experiments are performed without a victim cache line selection scheme as it affects allocation policy. The baseline scheme is data allocation by the unified LRU policy. The $\mathrm{weight_a}$ and $\mathrm{weight_b}$ are all set based on the experiment results. We sweep each weight from 0 to 2 by 0.25 units. The overall LLC energy consumption decreases as the values increase. And the energy consumption is almost saturated when both $\mathrm{weight_a}$ and $\mathrm{weight_b}$ are all 1. For this reason, we set the $\mathrm{weight_a}$ and $\mathrm{weight_b}$ to 1. All results are normalized to the baseline.

In Fig. 7a, *with OR* includes two last terms (hits and misses in the future) in Eq. (1) whereas *without OR* excludes them. The allocation policies without and with OR improve LLC dynamic power by 7.1 and 21.5 percent on average compared to the baseline, respectively. With OR, LLC dynamic power is decreased as much as 30.7 percent compared to the result without OR. Looking into the breakdown, OR decreases dynamic power for the write operation performed in STT-RAM by 8.4 percent on aver age compared to no OR case. The allocation policy with and without OR increase LLC hit counts by 21.9 and 16.3 percent respectively compared to the baseline.

The proposed allocation policy with OR decreases LLC dynamic power dramatically in most workloads. These result from not only reduced write accesses to STT-RAM but efficient bypass. Per-operation reusability based allocation policy

reduces unnecessary writes to STT-RAM by allocating data based on not only the current request's operation but also future expected operations. However, in L4-1, write power in STT-RAM for the OR is worse than the no OR case. In L4-1, LLC accesses are less than 20 percent of other workloads and compulsory misses to STT-RAM in the beginning of execution dominate total writes, causing relatively large dynamic power consumption.

As shown in Fig. 7b, per-operation reusability substantially increases LLC hit counts compared to the baseline.

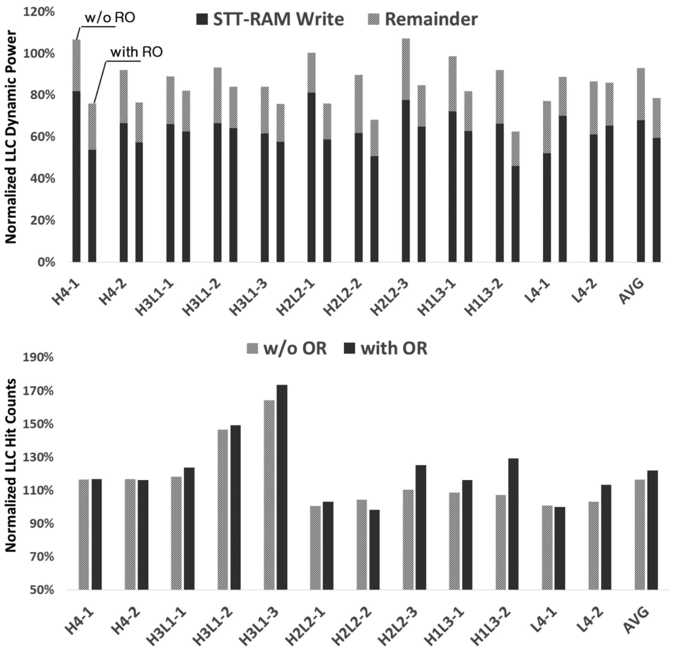

Fig. 7. Normalized (a) LLC dynamic power and (b) hit counts without a victim cache line selection scheme.

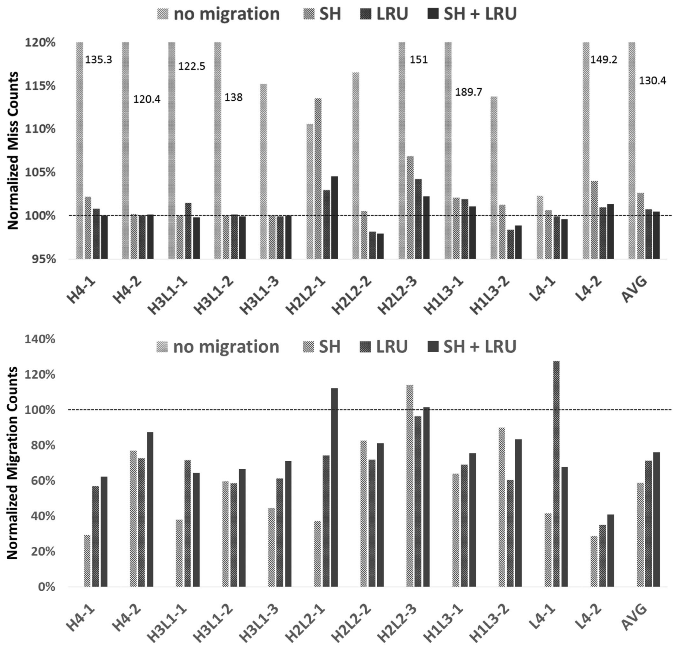

Fig. 8. An Impact of victim cache line selection scheme on (a) miss count and (b) migration count improvement.

Considering a current request as well as future per-operation reusability provides more accurate decision on data allocation or bypass, avoiding space waste and ensuring allocation of useful data. Only exception is H2L2-2. In this case, the allocation policy favors bypass in prediction while bypass and allocation show similar benefits. However, this decision causes slightly more misses. Nevertheless, the hit count is decreased by merely 2 percent compared to the baseline.

In summary, the proposed allocation policy significantly increases LLC hit counts while decreasing LLC dynamic power effectively for most workloads.

4.3 Impact of Victim Cache Line Selection Scheme

Per-operation Reusability based Allocation and Migration (ORAM) adopts a unified LRU replacement policy to increase cache utility. In this scheme, migration should be performed when the memory type chosen by allocation policy is different from the one chosen by the unified LRU algorithm. The proposed victim cache line selection scheme chooses an alternative victim cache line than one in LRU position to prevent energy waste caused by unnecessary migration. The victim cache line is selected by two conditions. First, least recently used cache lines whose sum of the hit count is less than a predetermined threshold can be selected. Second, the cache lines whose hit count is less than that of the LRU position can be selected. The former is named as a sum of hit count (SH) and the latter is named as LRU count (LRU), respectively, and their effects on migration and performance are separately measured in experiments.

For comparison, we also add two more cases where we choose a cache line at the LRU position as a victim line (all migration) or any eligible cache line between LRU and MRU position (no migration).

We measure cache miss and migration counts to evaluate victim cache line selection schemes. Experiments are conducted under the proposed allocation policy, and $\mathrm{HIT}_{\mathrm{Threshold}}$ is set to 256 for 4096 HPT entries.

Results in Fig. 8 represent the miss counts and migration counts normalized to the all migration case (set to 100 percent). First, the Fig. 8a show that the miss counts increase by 30.4, 2.6, and 0.8 percent on average for no migration, SH, and LRU, respectively. And the proposed scheme combined with SH and LRU (the rightmost bar), only increases by 0.4 percent. The overall increase in miss count is caused by the fact that we select a victim in non-LRU position. In H2L2-1 and H2L2-3, data allocated to SRAM is replaced too early because the victim cache line close to the MRU position is chosen. As a result, the miss counts increases as much as 4.5 percent. Nevertheless, the proposed scheme shows the least increase in miss count for most workloads, which is comparable to the miss count of all migration case.

On the other hand, in the Fig. 8b, migration counts are reduced to 39.6, 26.2 and 22.1 percent for SH, LRU and proposed scheme, respectively on average whereas the baseline (all migration) is 100 percent and no migration is 0 percent. As shown in figure, SH and LRU significantly reduce the migration counts in most workloads whereas the proposed scheme (SH+LRU) has slightly smaller reduction in the migration counts as it chooses better miss rate over the increased migration cost.

In our ORAM using a unified replacement policy, migration is inevitable. Our victim cache line selection scheme effectively reduces migrations while minimizing the miss count.

4.4 Impact of Target Cache Line Selection Scheme

To analyze the impact of the target cache line selection scheme, we measure the hit matching accuracy, which indicates the percentage of write and read hits in SRAM and STT-RAM respectively at the target cache line after migration.

In this experiment, the baseline always selects a target cache line at the LRU or MRU position according to the destination of data migration. That is, a cache line at the LRU

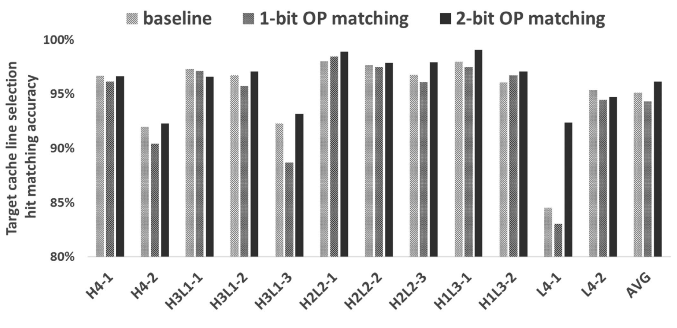

Fig. 9. Hit Matching Accuracy for Target Cache Line Selection Scheme.

position is selected when it migrates to SRAM, and a cache line at the MRU position is selected in the opposite case. For 1-bit operation (OP) matching, we select a target cache line by only considering the previous operation of a cache line. That is, when a target cache line needs to migrate to SRAM, we select a previously written cache line by searching from the LRU position to the MRU position. The opposite case searches a previously read cache line by searching from the MRU position to the LRU position. When the proposed scheme (2-bit operation (OP) matching) is used, a target cache line is selected by the proposed operation matching scheme.

In Fig. 9, results show hit matching accuracy of 95, 94.3 and 96.2 percent for the baseline, 1-bit OP matching, and proposed scheme, on average respectively. Although three methods show reasonably high accuracy for many cases, they are reduced significantly in L4-1 except for the proposed 2-bit OP matching scheme. This workload exhibits the characteristics of dominant read operations and a small working set size fitting in L3 but not in L2. Therefore, a replaced cache line from L2 cache is likely to be re-invoked, which is captured by the proposed 2-bit OP matching scheme. For this reason, the proposed scheme achieves accuracy improvement by 7.9 percent compared to the baseline in L4-1.

4.5 LLC Energy Efficiency

Our proposed allocation and migration policy aim to reduce the energy consumption at both main memory and LLC by reducing the miss rate while efficiently distributing write operations within the hybrid cache memory. In this section, we show the results for energy consumption of the LLC. All experimental results are normalized to the baseline scheme which uses the unified LRU replacement policy. We also compare results with the SRAM-only and STT-RAM-only LLC, APM, and PTHCM.

Fig. 10 presents the distribution of write operations per memory type in LLC. Total number of write operations processed by SRAM-only, STT-RAM-only, APM, PTHCM, and ORAM are different: 101.4, 96.7, 101.4, 92.7 percent and 81.8 percent respectively compared to the baseline. In case of the SRAM-only and STT-RAM-only LLC and the baseline, the difference in write counts is caused by varying cache misses from different capacity. On the other hand, in case of hybrid cache, the difference in the allocation and migration scheme causes the variation in writes. Write operations to STT-RAM for APM and PTHCM account for 72.2 and 69.4 percent respectively whereas they are only 56 percent in ORAM. In further analysis, we break down the LLC dynamic energy and total LLC energy in Figs. 11 and 12, respectively. The dynamic LLC energy is broken into tag, hit and miss, and migration energy in Fig. 11. The total LLC energy is broken into dynamic energy and leakage energy in Fig. 12. The largest portion of LLC energy is due to the leakage energy, solely accounting for 70 percent. The next largest portion is the miss energy. As shown in Fig. 11, most workloads consume large energy at the time of data allocation, which depends on the miss count and write energy consumption. For this reason, the miss energy of the SRAM-only accounts for 23.9 percent while the STT-RAM-only accounts for 91.3 percent despite its large capacity, compared to the baseline. In case of APM and PTHCM, they represent 60 and 61.9 percent of the dynamic energy consumed by the baseline. On the other hand, ORAM effectively bypasses unnecessary LLC accesses, reducing the energy required for cache misses to 24.2 percent on average. The reduction in the miss count and miss energy comes from not allocating useless data while preserving the useful data. For this reason, hit energy accounts for 30 percent at ORAM, while it only represents 25.6 and 22.2 percent for APM and PTHCM, respectively. On the other hand, migration overhead in ORAM is a significant part of dynamic energy, representing 15 percent, while they represent less than 3.7 percent in other schemes. Although the proposed per-operation reusability increases migration, decreased miss energy at the cost of migration proves much larger benefit. In terms of hit energy, SRAM-only shows the lowest hit energy for 15.6 percent due to the low-write energy while in case of STT-RAM-only, it accounts for up to 50.5 percent due to high-write energy, on average.

In summary, LLC dynamic energy is smaller by 54 percent in SRAM-only and is larger by 51 percent in STT-RAM-only. It is improved by 24.2 percent for proposed ORAM while only 5.7 and 7.9 percent for APM and PTHCM respectively, compared to the baseline. Finally, total LLC energy including both dynamic and leakage is reduced by 6.2, 20.8, 5.7, 3.3 and 12.3 percent on average for SRAM-only, STT-RAM-only, APM, PTHCM and ORAM, as shown in the Fig. 12.

4.6 Main Memory Energy Efficiency

In this section, we present the energy consumption of main memory, shown in Fig. 13. The main memory access is closely related to the LLC hit count. Under the same cache management scheme, SRAM-only and STT-RAM-only show the opposite tendency. The SRAM-only consumes 15.6 percent larger energy while the STT-RAM-only consumes 11.1 percent less energy on average, due to the difference in capacity. In SRAM-only, some workloads like H4-1 and H3L1-3 consume less energy than the baseline, which is the result of reduced leakage energy consumption of the main memory due to shorter LLC write latency. In the hybrid cache, ORAM reduces the energy by 21.2 percent and APM and PTHCM reduce by 16.4 and 14.3 percent respectively on average. Energy consumption is reduced for all workloads in ORAM. This is because it improves the LLC hit count by efficiently bypassing unnecessary data at the data allocation stage. In addition, the proposed scheme further reduces main memory accesses as it uses the unified LRU replacement policy to more accurately consider reuse-distance.

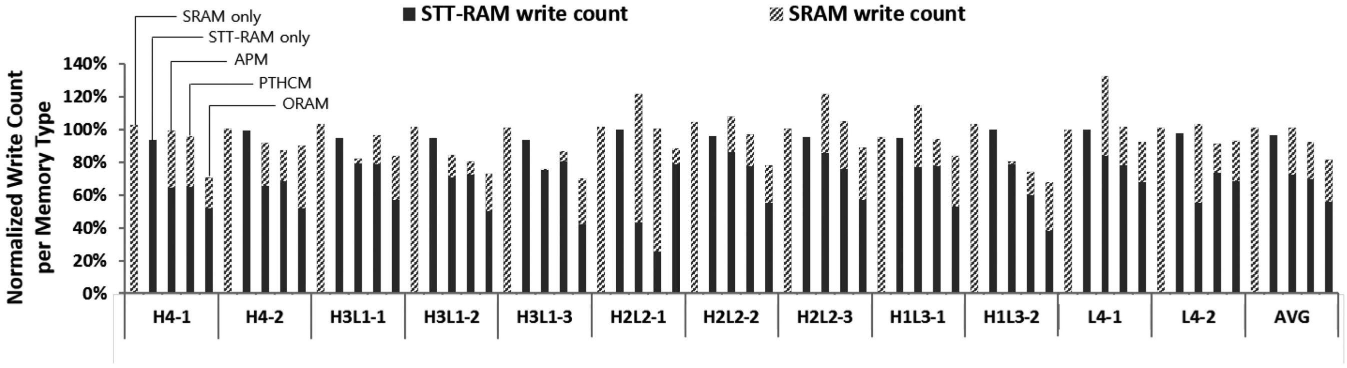

Fig. 10. Normalized Write Count Distribution per Memory Type.

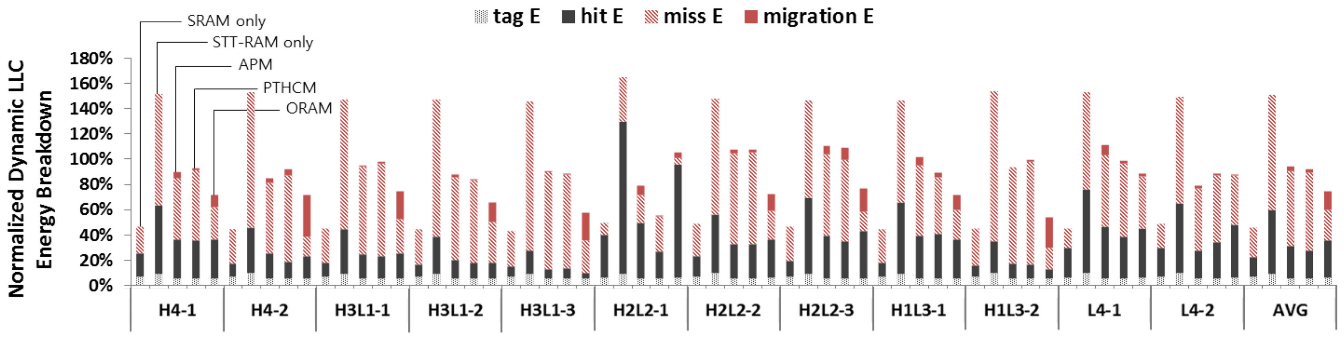

Fig. 11. Normalized LLC Dynamic Energy Breakdown.

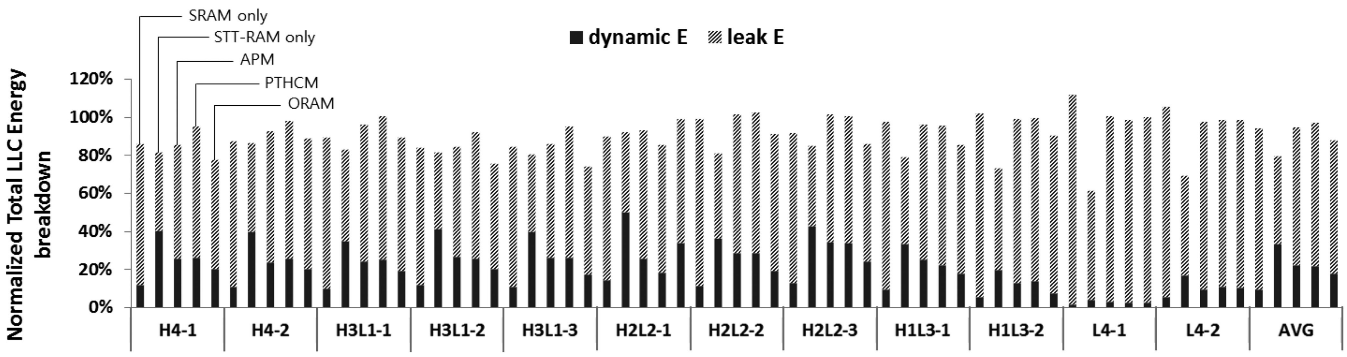

Fig. 12. Normalized Total LLC Energy Breakdown.

4.7 Performance

We evaluate the performance in terms of latency, execution time and IPC. First, latency is the direct performance indicator to evaluate cache management. In this experiment, latency includes the LLC hit and miss time, which is directly affected by the structural and management techniques in the LLC. As shown in Fig. 14, ORAM shows the largest latency improvement by 21.2 percent due to reducing both number of writes in the STT-RAM and accesses to the main memory, while APM and PTHCM achieve only 11.1 and 8.7 percent improvement respectively, on average. Latency is increased by 2.9 and 1.9 percent for SRAM-only and STT-RAM-only, respectively.

In the Fig. 15, execution time shows a similar tendency to the latency shown in Fig. 14. However, the improvement in execution time is reduced compared to the latency as it represents the performance of the entire system. The major distinction between the results of latency and execution time is that the execution time of STT-RAM is larger than SRAM, in H3L1-1, H3L1-2, and H2L2-3. This is because benchmarks such as mcf, lbm, sjeng, and leslie3d, which have very large miss counts in the LLC and poor write performance in STT-RAM-only surpasses the gain by its larger capacity. In summary, SRAM-only reduces execution time by 3.1 percent and STT-RAM-only increases execution

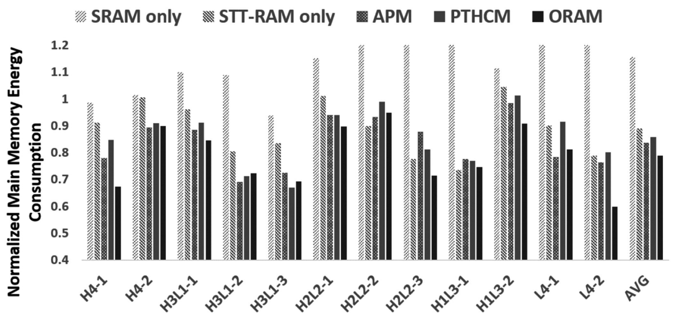

Fig. 13. Normalized Main Memory Energy Consumption.

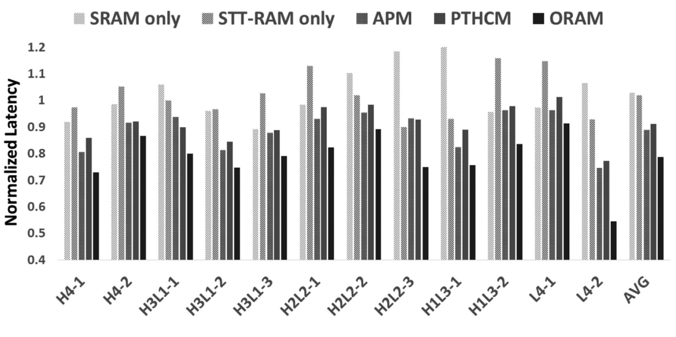

Fig. 14. Performance (Latency).

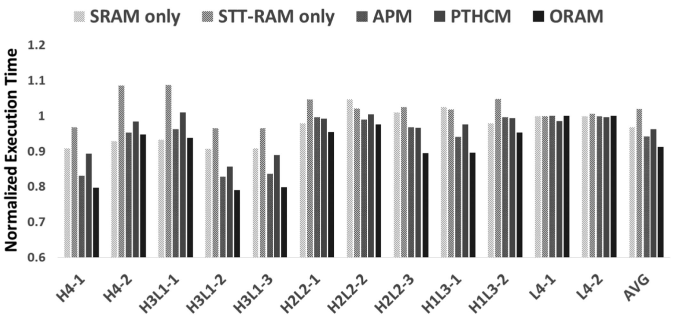

Fig. 15. Performance (Execution Time).

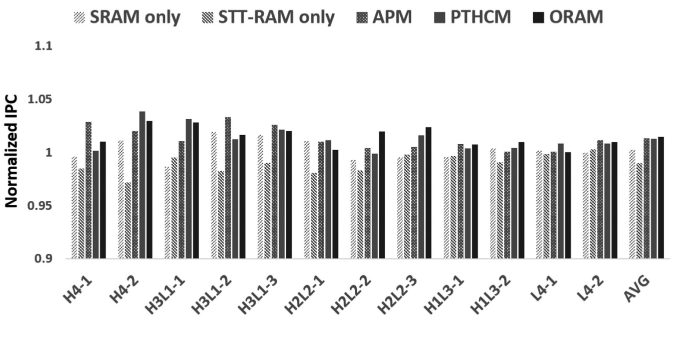

Fig. 16. Performance (IPC).

time by 2 percent compared to the baseline. In the hybrid cache group, ORAM reduces execution time by 8.8 percent while APM and PTHCM achieve 5.8 and 3.8 percent reduction, respectively.

IPC, normalized to the baseline, are shown in Fig. 16. On average, ORAM improves IPC by 1.5 percent while APM and PTHCM achieve 1.3 and 1.3 percent improvement respectively. SRAM-only improves the IPC by 0.2 percent and STT-RAM-only degrade IPC by 1 percent. Our experiments use a three-level cache hierarchy. As the majority of requests are processed in the upper-level caches, the impact of the LLC on the overall system performance would be limited. Overall, our allocation and migration policy perform best for all performance metrics.

4.8 Energy-Delay Product (EDP)

The energy-delay product normalized to the baseline is shown in Fig. 17. In our evaluation, EDP calculation uses energy and latency for the LLC and main memory. We evaluate LLC management scheme. The results show SRAM-only and STT-RAM-only have 120 and 90.8 percent and APM, PTHCM, and ORAM have 75.1, 80.5 and 63.1 percent for EDP, compared to the baseline.

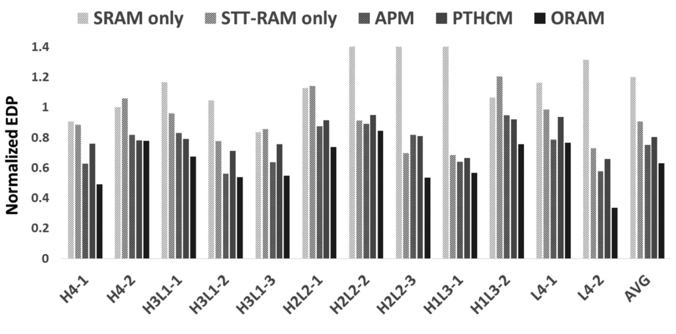

Fig. 17. Energy Delay Product (EDP).

4.9 Storage Overhead

ORAM consists of three modules for an allocation policy, a victim cache line selection scheme, and a target cache line selection scheme. The optimal hardware size for each module is determined from experiments.

First, the allocation policy module consists of per-core history queues and tables. Each entry of the history queue requires 4 bits to store the state information including bypass, memory type, hit/miss, and operation type. We use 4096 entries for the history queue considering trade-off between energy saving and hardware overhead. A history table requires only 10 counters. The size of each counter is 12 bits enough to cover the number of entries of the history queue. In total, 8 KB is needed for the memory component of allocation module. And the computation circuitry of the allocation module needs 4 $\times$ 18-bit adders and 4 $\times$ 18-bit multipliers. Therefore, the area cost for these circuits is about 1 percent compared to the LLC size[2].

The module for victim cache line selection consists of a global hit position queue and table. Since the hit position queue stores the hit position of the LRU stack, each entry requires 4 bits for a 10-way associative cache in ORAM and we need 4096 entries. The hit position table is composed of counters similar to the history table in allocation module. The size of each counter is 12 bits to represent the number of entries in the hit position queue and 10 counters are needed for 10-way associativity. In total, the storage overhead of the victim cache line selection module is about 8 KB.

The target cache line selection module requires 1-bit flag for each cache line to record the latest operation. To store recent read/write match history, each operation match queue contains 4096 1-bit entries. The OP hit table consists of 4 counters, and each counter has 12 bits to cover the size of the operation match queue. Therefore, the target cache line selection module requires about 21 KB.

The storage overhead for ORAM is about 37 KB, and it occupies 0.36 percent of the 10 MB hybrid cache. In conclusion, total hardware overhead including control circuits is 1.36 percent.

2. The area of a 16-bit adder and a 16-bit multiplier are 883μm^2 and 12,420μm^2, respectively [31]. The area of the whole computation logic is 0.160mm^2 even we used 18-bit adders and multipliers, which is marginal compared to the area of the hybrid cache, 15.652mm^2, derived from CACTI and NVSim ($\fallingdotseq$1%).

5 Related Work

Since its introduction, many researchers explored the cases of utilizing the STT-RAM memory as a cache [1], [2], [3], [4]. Most prior researches have mainly studied techniques to manage write operations to utilize STT-RAM as a cache memory. Several works proposed bypass schemes to prevent unnecessary write accesses to STT-RAM caches [5], [6]. The objective of a bypass scheme is to avoid allocation of dead blocks which are not likely to be reused and forward them directly to the upper-level cache to reduce unnecessary write operations. To identify dead blocks, some researchers suggested bypass techniques based on write access pattern analysis. Wang et al. [2] proposed an obstruction-aware prediction method to decide cache line allocation or bypass by calculating the cost based on access pattern analysis. Ahn et al. [3] provided a dead block prediction scheme to bypass the dead blocks identified using PC based addresses. However, these STT-RAM management schemes only reduce unnecessary write accesses due to dead blocks and cannot resolve the inherent write penalty of write-intensive cache lines. This is a fundamental problem of an STT-RAM only LLC.

The SRAM-STT-RAM hybrid cache architecture is proposed to alleviate write penalty by allocating write-intensive cache lines to SRAM. Prediction table-based hybrid cache management scheme (PTHCM) [15] allocates cache lines used by hot trigger instructions to SRAM. Hot trigger instructions are identified by maintaining the history of write counts based on PC. In addition, they prevent data from being allocated only in a certain cache memory to avoid overuse of one type of memory. However, they do not fully consider per-operation reusability - gain from hits of an allocated cache line and loss from misses of an evicted cache line for different operation and memory type.

Other studies proposed data allocation schemes with dynamic data migration. In general, migration policy redirects or swaps data based on allocated data and the operation type of a request. Wang et al. [7] proposed adaptive placement and migration (APM) based on the access pattern indexed by PC addresses. It decides whether to allocate to a cache memory or to bypass by identifying write burst and dead blocks through a pattern simulator. However, this scheme also neglects per-operation reusability after migration, leading to the loss of useful data. In addition, excessive migration may lead to an increase in energy consumption.

Along with allocation and migration schemes, a hybrid memory specific replacement policy was proposed. Syu et al. [21] proposed Least Recently Written (LRW) replacement policy to prevent least recently written data from occupying SRAM for a long time. However, as it has separate LRU stack per memory, it performs worse than a globally optimized unified LRU replacement policy.

In addition, several techniques deploy compiler based write intensity prediction [17], [18]. However, as the cache level becomes deeper, these techniques can not reflect the characteristics of actual workload observed in the LLC. Finally, other hybrid cache management schemes extended the techniques for a single processor for multi-processor applications [22], [30]. However, these techniques do not address the problems discussed above.

6 Conclusion

In this paper, we propose energy efficient per-operation reusability-based allocation and migration policy (ORAM). Prior works mostly focused on predicting write intensity for cache line allocation and migration.

We propose an energy efficient allocation policy by calculating expected total energy based on per-operation reusability. The expected total energy consists of potential energy consumption of an evicted and allocated cache line for the current request. We use the unified LRU replacement policy to increase reusability, which introduces discrepancy between the memory type determined from the allocation policy and the memory type which a victim cache line belong to. To minimize unnecessary migration, we propose a victim cache line selection scheme to select an alternative victim cache line. To reduce energy consumption due to data migration, we propose a target cache line scheme which considers per-operation reusability.

Experiment results show that ORAM improves LLC energy consumption by 12.3 percent and main memory energy consumption by 21.2 percent compared to the hybrid cache management scheme using unified LRU replacement policy. The latency and execution time of our management scheme show improvement by 21.2 and 8.8 percent over the baseline. In addition, the proposed scheme surpasses the best existing scheme by 7 and 4.8 percent for the energy consumption in the LLC and the main memory and by 6.6 and 4.8 percent for the latency and execution time, respectively. As a result, our proposed scheme improves EDP by 36.9 percent over the baseline hybrid cache management scheme using unified LRU replacement policy.

Acknowledgments

This work was supported in part by the IT R&D program of MOTIE/KEIT. [10052716, Design technology development of ultra-low voltage operating circuit and IP for smart sensor SoC]; in part by the MOTIE(Ministry of Trade, Industry & Energy) (10080590) and KSRC (Korea Semiconductor Research Consortium) support program for the development of the future semiconductor device.

References

[1] M. Rasquinha, D. Choudhary, S. Chatterjee, S. Mukhopadhyay, and S. Yalamanchili, "An energy efficient cache design using spin torque transfer (STT) RAM," in *Proc. ACM/IEEE Int. Symp. Low-Power Electron. Des.*, Aug. 2010, pp. 389–394.

[2] J. Want, X. Dong, and Y. Xie, "OAP: An obstruction-aware cache management policy for STT-RAM last-level caches" in *Proc. Des. Autom. Test Europe Conf. Exhib.*, Mar. 2013, pp. 847–852.

[3] J. Ahn, S. Yoo, and K. Choi, "DASCA: Dead write prediction assisted STT-RAM cache architecture" in *Proc. IEEE 20th Int. Symp. High Performance Comput. Archit.*, Feb. 2014, pp. 25–36.

[4] H. Kim, S. Kim, and J. Lee, "Write-amount-aware management policies for STT-RAM caches," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 25, no. 4, pp. 1588–1592, Apr. 2017.

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

[6] S. Lhan, Y. Y. Tian, and D. A. Jemenes, "Sampling dead block prediction for last-level caches," in *Proc. 43rd Annu. IEEE/ACM Int. Symp. Microarchitecture*, Dec. 2010, pp. 175–186.

[7] Z. Wang, D. A. Jimenez, C. Xu, G. Sun, and Y. Xie, "Adaptive placement and migration policy for an STT-RAM-based hybrid cache," in *Proc. IEEE 20th Int. Symp. High Perform. Comput. Archit.*, Feb. 2014, pp. 13–24.

[8] B. Quan, T. Zhang, T. Chen, and J. Wu, "Prediction table based management policy for STT-RAM and SRAM hybrid cache," in *Proc. 7th Int. Conf. Comput. Convergence Technol.*, Dec. 2012, pp. 1092–1097.

[9] A. Jadidi, M. Arhomand, and H. Sarbazi-Azad, "High-endurance and performance-efficient design of hybrid cache architectures through adaptive line replacement," in *Proc. IEEE/ACM Int. Symp. Low Power Electron. Des.*, Aug. 2011, pp. 79–84.

[10] J. Li, C. J. Xue, and Y. Xu, "STT-RAM based energy-efficiency hybrid cache for CMPs," in *Proc. IEEE/IFIP 19th Int. Conf. VLSI Syst-on-Chip*, Oct. 2011, pp. 31–36.

[11] 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. ACM/IEEE Int. Symp. Low Power Electron. Des.*, Aug. 2012, pp. 237–242.

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

[13] N. Kim, J. Ahn, W. Seo, and K. Choi, "Energy-efficient exlusive last-level hybrid caches consisting of SRAM and STT-RAM," in *Proc. IEEE/IFIP Int. Conf. Very Large Scale Integr.*, Oct. 2015, pp. 183–188.

[14] M. Imani, S. Patil, and T. Rosing, "Low power data-aware STT-RAM based hybrid cache architecture," in *Proc. 17th Int. Symp. Quality Electron. Des.*, Mar. 2016, pp. 88–94.

[15] J. Ahn, S. Yoo, and K. Choi, "Prediction hybrid cache: An energy-efficient STT-RAM cache architecture," *IEEE Trans. Comput.*, vol. 65, no. 3, pp. 940–951, Mar. 2016.

[16] 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. Des.*, Aug. 2012, pp. 191–196.

[17] D. W. Lee and K. Choi, "Energy-efficient partitioning of hybrid caches in multi-core architecture," in *Proc. 22nd Int. Conf. Very Large Scale Integr.*, Oct. 2014, pp. 1–6.

[18] J.-H. Choi and G.-H. Park, "NVM way allocation scheme to reduce NVM writes for hybrid cache architecture in chip-multiprocessors," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 10, pp. 2896–2910, Oct. 2015.

[19] X. 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. Des. Autom. Test Europe Conf. Exhib.*, pp. 737–742, Apr. 2009.

[20] Q. Li, J. Li, L. Shi, M. Zhao, C. J. Xue, and Y. He, "Compiler-assisted STT-RAM-based hybrid cache for energy efficient embedded systems," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 22, no. 8, pp. 1829–1840, Aug. 2014.

[21] S.-M. Syu, Y.-H. Shao, and I.-C. Lin, "High-endurance hybrid cache design in CMP architecture with cache partitioning and access-aware policy," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 23, no. 10, pp. 2149–2161, Oct. 2015.

[22] W. Wei, D. Jiang, J. Xiong, and M. Chen, "HAP: Hybrid-memory-Aware Partition in Shared Last-Level Cache," in *Proc. IEEE 32nd Int. Conf. Comput. Des.*, Oct. 2014, pp. 28–35.

[23] N. Duong, et al., "Improving cache management policies using dynamic reuse distances," in *Proc. 45th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Dec. 2012, pp. 389–400.

[24] N. Binkert, et al., "The gem5 Simulator," *ACM SIGARCH Comput. Archit. News*, vol. 39, pp. 1–7, May. 2011.

[25] J. L. Henning, el al., "SPEC CPU2006 benchmark descriptions," *ACM SIGARCH Comput. Architecture News*, vol. 34, pp. 1–17, Sept. 2006.

[26] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite: Characterization and architectural implications," in *Proc. Int. Conf. Parallel Archit. Compilation Tech.*, Oct. 2008, pp. 72–81.

[27] N. Muralimanohar, R. Balasubramoniam, and N. P. Jouppi, "CACTI 6.0: A tool to understand large caches," in *HP Laboratories*, Apr. 2008. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.147.3834&rep=rep1&type=pdf

[28] 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 Des. Integr. Circuits Syst.*, vol. 31, no. 7, pp. 994–1007, Jul. 2012.

[29] D. Apalkov, et al., "Spin-transfer torque magnetic random access memory (STT-RAM)," in *ACM J. Emerging Technol. Comput. Syst.*, vol. 9, May 2013, Art. no. 13.

[30] B.-M. Lee and G.-H. Park, "Performance and energy-efficiency analysis of hybrid cache memory based on SRAM-MRAM," in *Proc. Int. SoC Des. Conf.*, Nov. 2012, pp. 247–250.

[31] M. Shoba and R. Nakkeeran, "Energy and area efficient hierarchy multiplier architecture based on vedic mathematics and GDI logic," *Eng. Sci. Technol. Int. J.*, vol. 20, pp. 321–331, Jun. 2016.

Minsik Oh received the BS degree from Yonsei University, Seoul, South Korea, in 2012, where he is currently working toward the PhD degree in electrical and electronic engineering. His current research interests include high performance memory design and system architecture.

Kwangsu Kim received the BS degree from Yonsei University, Seoul, South Korea, in 2014, where he is currently working toward the PhD degree in electrical and electronic engineering. His current research interests include memory hierarchy, near-data processing, and ultralow power computing. He did an internship in Qualcomm Korea, Seoul, in 2012.

Dooheon Choi received the BS degree in electrical and electronic engineering from Yonsei University, Seoul, Korea, in 2015, where he is currently working toward the PhD degree in electrical and electronic engineering. His research interests include memory architecture and system-level design.

Hyuk-Jun Lee received the BS degree in computer engineering from the University of Southern California, Los Angeles, CA, in 1993, and the MS and PhD degrees in electrical engineering from from Stanford University, Stanford, CA, in 1995 and 2001, respectively. From 2001 to 2011, he served as a senior engineer in routing technology group at Cisco System, San Jose, CA, where he participated in developing CRS-1 and CRS-3. Currently, he is an assistant professor with the Department of Computer Science and Engineering, Sogang University, Seoul, Korea. His research interests include embedded systems, bio-computing, low-power design, and memory architectures.

Eui-Young Chung received the BS and MS degrees in electronics and computer engineering from Korea University, Seoul, Korea, in 1988 and 1990, respectively, and the PhD degree in electrical engineering from Stanford University, Stanford, California, in 2002. From 1990 to 2005, he was a principal engineer with SoC R&D Center, Samsung Electronics, Yongin, Korea. Currently, he is a professor in the School of Electrical and Electronics Engineering, Yonsei University, Seoul, Korea. His research interests include system architecture, bio-computing, and VLSI design, including all aspects of computer-aided design with the special emphasis on lowpower applications, and flash memory applications. He is a member of the IEEE.