# Energy-Efficient Partitioning of Hybrid Caches in Multi-core Architecture

Dongwoo Lee, Kiyoung Choi



# Energy-Efficient Partitioning of Hybrid Caches in Multi-Core Architecture

Dongwoo Lee and Kiyoung Choi

Department of Electrical and Computer Engineering, Seoul National University
Seoul, Republic of Korea
dongwoolee@dal.snu.ac.kr, kchoi@snu.ac.kr

**Abstract.** This chapter presents a technique for reducing energy consumed by hybrid caches that have both SRAM and STT-RAM (Spin-Transfer Torque RAM) in multi-core architecture. It is based on dynamic way partitioning of the SRAM cache as well as the STT-RAM cache. Each core is allocated with a specific number of ways consisting of SRAM ways and STT-RAM ways. Then a cache miss fills the corresponding block in the SRAM or STT-RAM region based on an existing technique called read-write aware region-based hybrid cache architecture. Thus, when a store operation from a core causes an L2 cache miss (store miss), the block is assigned to the SRAM cache. When a load operation from a core causes an L2 cache miss (load miss) and thus causes a block fill, the block is assigned to the STT-RAM cache. However, if all the allocated ways are already placed in the SRAM, then the block fill is done into the SRAM regardless of store or load miss. The partitioning decision is updated periodically. We further improve our technique by adopting the so called *allocation switching technique* to avoid too much unbalanced use of SRAM or STT-RAM. Simulation results show that the proposed technique improves the performance of the multi-core architecture and significantly reduces energy consumption in the hybrid caches compared to the state-of-the-art migration-based hybrid cache management.

**Keywords:** Spin-Transfer Torque RAM (STT-RAM), hybrid caches, cache partitioning

## 1 Introduction

Non-volatile memories such as Spin-Transfer Torque RAM (STT-RAM) have been researched as alternatives to SRAMs due to their low static power consumption and high density. Among such memories, STT-RAM has a relatively high endurance compared to other memories and thus it is regarded as the best candidate for substituting SRAM used in last-level shared caches in modern chip multi-processors. [2]

However, STT-RAM has asymmetric characteristics of read and write. Writing into STT-RAM consumes significantly larger energy and takes more time than reading. Such characteristics can significantly increase energy consumption and degrade performance of the system if the program running on the processor

cores need frequent writes into the shared cache. To mitigate the adverse effect of the characteristics of STT-RAM, many methods are proposed [1, 3, 7, 9, 18, 17] on hybrid caches, where a small SRAM cache is combined with a large STT-RAM cache. In the hybrid caches, a block that is expected to be written frequently is placed in the SRAM cache, which has much lower write overhead compared to STT-RAM. So, in the hybrid cache approaches, where to place a block between SRAM and STT-RAM is one of the most important issues in reducing energy consumption and enhancing performance.

Fig.1. shows that block fill on a read miss is one major source of write operations into a cache. So reducing the number of block fills is another method of lowering dynamic energy consumption of hybrid caches. A cache partitioning technique is developed to increase the performance of chip multi-processors by dynamically partitioning ways of the cache such that the number of misses is minimized and consequently the number of block fills is reduced. So the application of cache partitioning has great potential of reducing the dynamic energy consumption of the hybrid caches. However, conventional cache way partitioning techniques cannot be applied to hybrid caches directly, because ways in the hybrid caches are separated into SRAM and STT-SRAM and thus ways assigned to a core as a result of partitioning also should be divided into SRAM and STT-RAM. Properly dividing the ways of a partition into the two caches and placing a block into a more efficient cache are new challenging issues when applying a partitioning technique to hybrid caches.

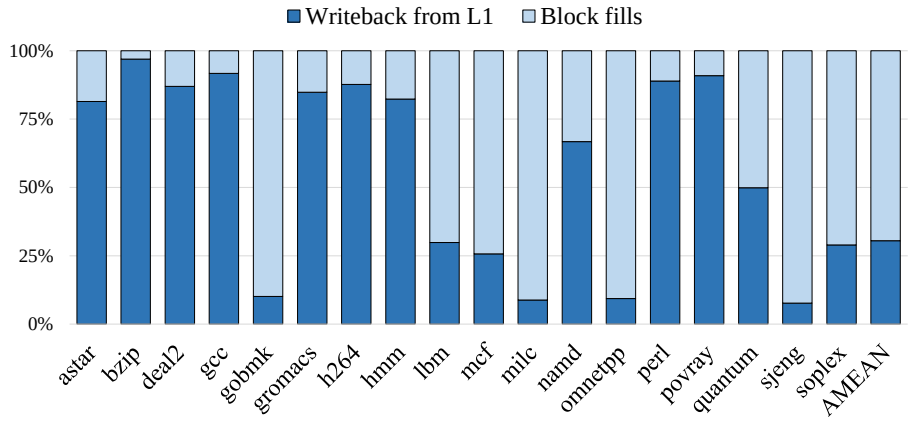


**Fig. 1.** Breakdown of write-inducing events in the L2 cache.

This chapter proposes a technique that adopts the cache partitioning scheme in a hybrid cache for reducing the energy consumption. We assume that the hybrid cache is a last-level shared cache in multi-core architecture. A conventional partitioning technique called utility-based partitioning is used to determine the

sizes of the partitions, one for each core, such that the number of misses is minimized. To incorporate the technique into hybrid caches, the replacement policy should be redesigned. When a store operation of a core causes a miss in the shared cache, the corresponding new block is placed in SRAM. If all the allocated ways in the SRAM are already in use, a victim is selected among them. In the case of a load miss, the block is placed in STT-RAM if there is an unused way among those allocated to the core. If all the allocated ways in the STT-RAM are full, a victim block is selected. However, if all the ways allocated to the core have already been assigned to SRAM, a victim is selected among these blocks in SRAM. So, within a partition, the ratio between SRAM ways and STT-RAM ways can be adjusted and the ratio can be adjusted differently for different sets. Simulation results show that our techniques improve the performance of a quad-core system by 4.9%, reduce the energy consumption of hybrid caches by 10.0%, and decrease the DRAM energy by 5.9% compared to the state-of-the-art migration based hybrid cache management technique.

In the above scheme, it is possible that cache accesses are concentrated to either SRAM or STT-RAM so that the partitioning decision does not fully apply to hybrid caches. We resolve this situation by switching an allocation policy of load and store miss with small constant probability. Thus, within the small constant probability, the block allocation on a load miss is done as if it were a store miss and vice versa. This switching technique, which we call allocation switching technique, further improves the performance of a multi-core system by 1.8% (6.7% in total), and reduces the energy consumption by 6.6% (16.6% in total), and DRAM energy by 1.6% (7.5% in total).

This chapter is organized as follows. Section 2 explains the background of STT-RAM technology, hybrid caches, and partitioning techniques. Section 3 describes the details of our proposed technique that exploits partitioning scheme for hybrid caches. Section 4 shows the evaluation methodology and Section 5 discusses the results of evaluation. Section 6 summarizes the survey of the related work and Section 7 concludes this chapter.

## 2 Background

### 2.1 STT-RAM Technology

Spin-Transfer Torque RAM (STT-RAM) is an emerging memory technology, which uses a Magnetic Tunnel Junction (MTJ) as an information carrier. The MTJ consists of two ferromagnetic layers and one tunnel barrier between them as shown in Fig. 2. The reference layer has a fixed direction of magnetic flow and the free layer can change its magnetic direction when a spin-polarized current flows through the MTJ with intensity above a threshold. If the directions of the two layers are in parallel, the MTJ has resistance lower than that of anti-parallel case. Thus, by measuring the voltage difference across the MTJ with a small current flow, the state of MTJ can be detected. Because of its non-volatility, STT-RAM has a very low leakage current. For read operations, it requires energy and latency comparable to SRAM, but in case of write operations, it consumes much higher

energy and takes much longer than SRAM. Other noticeable properties of STT-RAM are its high endurance compared to other non-volatile memories such as Phase Change RAM (PRAM) or Resistive RAM (ReRAM) and its high density compared to SRAM.

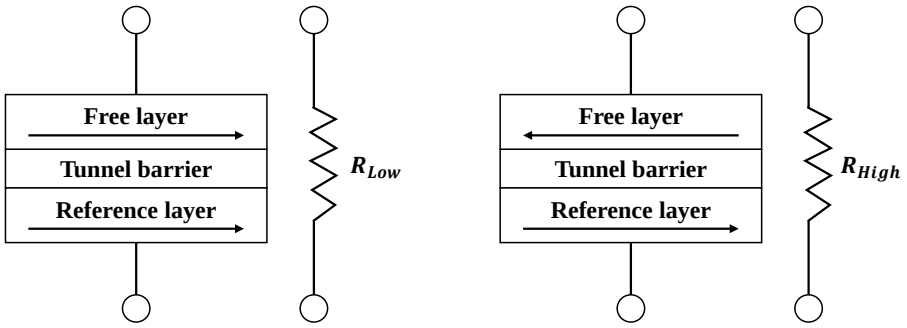


**Fig. 2.** Magnetic tunnel junction of STT-RAM.

### 2.2 Hybrid Approach for Last-Level Caches

Unlike other emerging non-volatile memories targeting off-chip storage, STT-RAM is expected to replace SRAM used for current last-level caches (LLCs) in chip multi-processors mainly due to its fast read latency close to that of SRAM and high endurance. But its high overhead of write operation obstructs the use of pure STT-RAM LLCs. To mitigate the shortcoming, there have been researches on hybrid caches that combine a small sized SRAM cache with a large sized STT-RAM cache, which lowers the write overhead of STT-RAM significantly. These two different caches are usually combined by a region-based manner. The two memories are placed at a same level of cache hierarchy and share tags of the caches. Ways are divided into the SRAM and the STT-RAM region and consequently, block placement becomes an important issue. it will be more efficient to place blocks that will be frequently written in SRAM and place others in STT-RAM.

One of the state-of-the-art techniques is the read-write aware hybrid cache architecture [17], in which a block fill caused by a store miss is made to a write-efficient SRAM region and a block fill caused by a load miss is made to a read-efficient STT-RAM region based on the assumption that a block filled by a store miss is prone to frequent write and a block filled by a load miss is prone to frequent read. If there are consecutive hits on a block in a way opposite to the initial assumption, the block is migrated to the other region.

### 2.3 Cache Partitioning Technique

The utility-based cache partitioning technique is proposed to improve the performance of chip multiprocessors by minimizing the number of misses on a shared cache [12]. It has the same number of Utility-MONitors (UMONs) and CPU cores. A UMON consists of an Auxiliary Tag Directory (ATD) and counters for measuring the number of hits for each position of LRU stack by observing the access of sampled sets in a cache. The algorithm periodically calculates the partition size of each core to maximize the number of hits in the cache. Bits are added to cache tags to identify the core that owns the block. The replacement policy gradually adjusts the size (number of ways) of each partition to the calculated size. The partitioning technique is especially efficient when cache insensitive (e.g., low hit rate) applications and cache sensitive ones are running at the same time on a multi-core processor. It tends to restrict the cache usage of the low hit-rate applications, while increasing the performance of other cache sensitive ones.

## 3 Partitioning technique for hybrid caches

This section explains the details of our technique proposed in [6], which exploits the partitioning technique to reduce the energy consumption of hybrid caches in a multi-core processor. It also describes the extended work on the allocation switching technique to mitigate too much concentration on SRAM or STT-RAM.

### 3.1 Motivation

Read access of the last-level shared cache comes from a load or store miss of the upper level cache. In the case of store miss, a block written into the upper level cache becomes dirty, and so it eventually causes a write-back to the shared cache. Write access of the last-level cache can be classified into a write-back from the upper-level cache and a block-fill on a read miss. A read-write aware hybrid cache [17] utilizes the property of read access. If a read miss on the shared cache is caused by a store miss of the upper level cache (store miss), the new block is placed in SRAM so that a write-back to that block occurs in the write-efficient region. In case of a read miss caused by a load miss of the upper level cache (load miss), the block is placed in STT-RAM where read operations are efficient. If the decision is wrong, migration from one cache to the other occurs to remedy the situation. However, this technique does not consider the block-fill caused by a read miss in the shared cache and a write-back to a block (in the shared cache) that was originally loaded by a load miss of the upper level cache.

We propose a technique for hybrid caches that handles these cases by exploiting the advantage of a dynamic cache partitioning technique. The dynamic partitioning scheme adjusts the sizes of partitions such that the total number of misses in the shared cache is minimized. Thus it helps reduce the number of block-fills caused by read misses. In case that the number of SRAM ways used

by a core is already greater than[1] or equal to that allocated to the core, a new block loaded into the LLC due to a miss is placed in the SRAM even if the miss is a load miss. The rationale is that utilizing the already allocated SRAM ways helps reducing energy and latency since subsequent write-backs to the block can be done in SRAM.

### 3.2 Architecture

Fig. 3 shows a structure to implement our technique. Basically, it combines a cache partitioning architecture with hybrid caches. For a dynamic cache partitioning technique, it has a set of UMONs that sample cache accesses to designated sets of the cache and count the number of hits on each position of an LRU stack. It periodically calculates the partition size of the cache for each of the multiple cores such that the total number of misses in the cache is minimized. It is done by using the values of hit counters collected by the UMONs during the previous period. In the cache tags, bits are added per block to identify the core that owns the block.

The last-level shared cache consists of SRAM and STT-RAM. Thus the data array is a hybrid of the two memories, but the tags are made of SRAM only. Ways of a set are divided unevenly; SRAM contains a smaller number of ways and STT-RAM covers the rest. When a new block needs to be placed in hybrid caches, the result of the partitioning technique is used to decide the type of caches for allocation and select a victim block in the resulting type of cache as explained in the following section.

For simplicity, we assume that an independent application runs on each core in a multi-core architecture, so a UMON is attached to each core. However, the technique can also be applied in a per-application manner. In that case, the UMONs are required as many times as the number of applications running on the architecture.

### 3.3 Replacement Policy

To maximize the effect of applying a partitioning technique to hybrid caches, where to insert a new block should be carefully decided. If a core performs a store operation and eventually causes a miss in the shared cache, SRAM in the shared cache is selected as a location to place a new block because the block allocated in the upper level cache will become dirty, and thus there will be a write-back into the LLC. The victim selection policy within the SRAM is changed from the traditional LRU policy to a more complicated one that involves partitioning decision.

Let us define several notations as follows.

[1] Since we perform dynamic partitioning, the number of cache ways allocated to a core can be reduced after repartitioning. Thus $N_S$(i, j) can temporarily exceed $N_{UT}$(i) set by the new partitioning. The excess ways can be claimed later by other cores and thus $N_S$(i, j) will be reduced to the new value of $N_{UT}$(i).

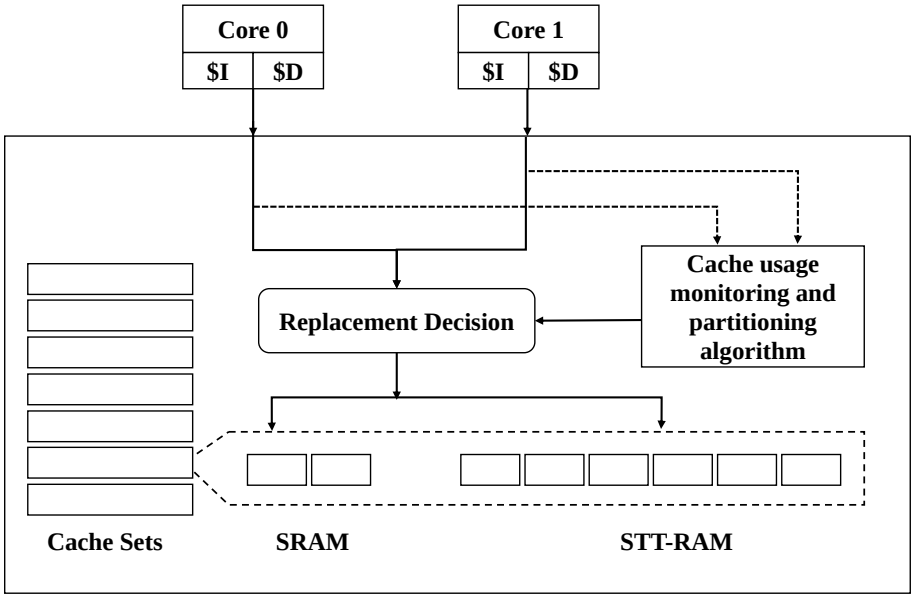


**Fig. 3.** Structure of hybrid cache partitioning.

- $N_S$(i, j): the number of SRAM blocks currently in use by core i in set j.
- $N_M$(i, j): the number of STT-RAM blocks currently in use by core i in set j.
- $N_{UT}$(i): the total number of ways allocated to core i by the partitioning.
- $N_{UM}$(i, j) = $N_{UT}$(i) - $N_S$(i, j): the maximum number of ways that can be allocated to core i in set j of STT-RAM.

If $N_S$(i, j) $\geq N_{UT}$(i), then the LRU among these blocks is chosen as a victim. If $N_S$(i, j) $< N_{UT}$(i), the LRU among all blocks in SRAM is selected as a victim. This modification is to better utilize SRAM ways which is relatively smaller than STT-RAM ways.

If a miss in the shared cache is caused by a load miss in the upper level cache, the new block can be placed either in SRAM or in STT-RAM. If $N_S$(i, j) $\geq N_{UT}$(i), the LRU among them is chosen as a victim. This decision procedure is exactly the same as that of the store miss case. It allows using already allocated SRAM ways and avoids unnecessary overhead of allocating a new way in STT-RAM. And by placing the block into the SRAM, the block-fill due to the read miss is done in the write-efficient region, reducing the dynamic energy of caches without performance degradation of multi-core processors.

If $N_S$(i, j) $< N_{UT}$(i), the new block is placed in STT-RAM. The total number of blocks in a set of both caches is maintained not to exceed $N_{UT}$(i), and thus $N_M$(i, j) is maintained not to exceed $N_{UM}$(i, j). Therefore, $N_M$(i, j) is adjusted dynamically according to the change of $N_S$(i, j) and/or $N_T$(i). For example, if $N_M$(i, j) becomes larger than $N_{UM}$(i, j) due to a new partitioning, the LRU

among the STT-RAM blocks owned by core i is selected to be replaced by a new block of another core. If $N_M(\mathrm{i}, \mathrm{j}) < N_{UM}(\mathrm{i}, \mathrm{j})$, the LRU of blocks owned by other cores is chosen as a victim. Contrary to the utility-based partitioning, our approach allows a zero value for $N_{UM}(\mathrm{i}, \mathrm{j})$, and thus we allow to select the LRU of blocks allocated by other cores in the STT-RAM is chosen as a victim when there is no existing block in the STT-RAM owned by the core that requests the new block.

In this replacement policy, new blocks introduced by store misses can be placed only in SRAM, but new blocks introduced by load misses can be placed either in SRAM or in STT-RAM so that the utilization of the SRAM region can be maintained relatively high. The ratio between $N_S(\mathrm{i}, \mathrm{j})$ and $N_M(\mathrm{i}, \mathrm{j})$ can be adjusted freely under the constraint given by $N_S(\mathrm{i}, \mathrm{j}) + N_M(\mathrm{i}, \mathrm{j}) \leq N_{UT}(\mathrm{i})$, which results in a decrease of the total misses in hybrid caches.

### 3.4 Allocation switching technique

In the above replacement policy, it is possible that too many block-fills are guided into either SRAM or STT-RAM, in which case, the partitioning decision cannot be properly applied to hybrid caches. For example, if there is an application that has a large number of load misses and no store miss, then a cache block of this application cannot be allocated to the SRAM. Similarly, if there is an application that has plenty of store misses and rare load misses, then a cache block of this application cannot be allocated to STT-RAM. If such applications are cache-size sensitive, then the partitioning decision of the applications cannot be applied to hybrid caches properly.

To resolve this situation, we devise an allocation switching technique where a block allocation on a load miss is done as if there were a store miss and vice versa within a small constant probability. By applying this switching technique, the concentration on either SRAM or STT-RAM can be relaxed.

The allocation switching technique works as follows. Consider the case where most of the misses occurring during the execution of an application are load misses. Then the application will put most of its blocks in the STT-RAM region even if some extra ways allocated to the application (or core) are in SRAM, making the STT-RAM crowded. It may not be able to move the extra ways in the SRAM to the STT-RAM side since other cores are already taking the STT-RAM space. In that case, it will be beneficial to place new blocks into the extra SRAM ways. Thus, sometimes on a load miss, placing the corresponding block into the SRAM region instead of the STT-RAM region helps alleviating the congestion in the STT-RAM region.

We set the probability of such allocation switching to a small constant value obtained empirically. But there is still room for further research on adjusting the probability value dynamically to achieve even higher performance because the characteristics of applications are very different.

## 4 Evaluation methodology

This section describes the experimental setup and methodology used to evaluate our hybrid cache partitioning technique.

### 4.1 Simulator

We evaluate our cache partitioning technique using a cycle accurate simulator MARSSx86 [11]. For off-chip memory model, we use DRAMSim2 simulator [13], which is integrated into MARSSx86. The details of our system configuration are listed in Table 1. The system has a 3.0 GHz, quad-core out-of-order processor based on x86 ISA. The cache hierarchy is configured with 32KB, 4-way set-associative L1 instruction/data caches and 4MB, 16-way set-associative L2 shared caches. We implement the shared L2 hybrid cache with a bank contention model. The L2 cache is a 16-way set associative cache consisting of 4 ways of SRAM and 12 ways of STT-RAM with asymmetric read/write latency. The off-chip DRAM is configured as a DDR3-1333 in which CL, tRCD, tRP timings are 10, 10, and 10, respectively.

The parameters of the hybrid cache model are calculated using NVSim [4] and CACTI 6.0 [10] under 45 nm technology. A tag of 16-way 4MB SRAM is borrowed for hybrid caches. 4-way 1MB SRAM and 4-way 1MB STT-RAM is configured for the data array. By combining one SRAM bank and three banks of STT-RAM, 16-way 4-bank data array is designed for the hybrid cache. Table 2 lists the energy consumption of the L2 cache.

**Table 1.** System configuration

| **Parameters** | **Configuration** |
|---|---|
| Processor | 3.0 GHz, 4-core CMP, 4-wide, out-of-order, 128-entry ROB, 48-entry LSQ. |
| L1 Caches | I-cache: 32kB, 64B lines, 4-way, 2-cycle latency<br>D-cache: 32kB, 64B lines, 4-way, 2-cycle latency |
| L2 Cache | Unified, 4MB, 64B lines, 16-way (4-way SRAM and 12-way STT-RAM), 10 cycle latency for SRAM, 10 cycle (read) and 38 cycle (write) latency for STT-RAM |
| DRAM | DDR3-1333 (10-10-10), 1 channel, 8 banks, 32-entry queue, open-page policy, FR-FCFS policy |

### 4.2 Workloads

We use SPEC 2006 [5] with reference input as workloads for the evaluation. For more precise analysis, the simulation method is changed from the work in [6]

**Table 2.** Energy Consumption of the L2 Cache

| | **Read Energy** | **Write Energy** | **Static Power** |
|---|---|---|---|
| SRAM Region | 0.217 nJ | 0.217 nJ | 14.682 mW |
| STT-RAM Region | 0.097 nJ | 0.670 nJ | 3.438 mW |

where 10 billion instructions were fast-forwarded per core to skip the initialization phase of code and multi-core workloads were mixed with high and low-MPKI applications. In the simulation of this chapter, we annotate a synchronization point in the program source code after some initialization code part. As a result, a multi-core simulation always starts at the same points of benchmarks in the MARSSx86 simulator. Because of this modification, the simulation region of a benchmark can be different from that in [6].

Workloads have also been selected differently; in this chapter, cache sensitivity of benchmarks is considered. We analyze the cache sensitivity of benchmarks by simulating benchmarks on a 16-way 4MB L2 cache and on a 2-way 512kB L2 cache. If the number of L2 misses per kilo instructions (MPKI) of a benchmark on the 16-way 4MB L2 cache is greater by more than 10% compared to that on the 2-way 512kB L2 cache, then we classify this benchmark as cache-sensitive. Otherwise, we classify it as cache-insensitive. The benchmarks are listed in Table 3.

Now we randomly assemble 20 quad-core workloads considering cache sensitivity. Workloads are categorized into five groups, four workloads in each group. For each quad-core workload, we mix four benchmarks. There is no cache-insensitive benchmark in the workloads in group1, one in each workload in group2, two in group3, three in group4, and group5 has only cache-insensitive benchmarks. Workloads are listed in Table 4.

After 5 million cycles cache warm up, 2 billion instructions are simulated for multi-programmed workloads on quad-core processors. For the partitioning technique, we use a 5 million cycle period for monitoring and partitioning decisions as was done in [12]. We use a 2% probability value for an allocation switching technique which is obtained empirically (the performance is not sensitive to the value around 2%).

## 5 Results

This section discusses the results of simulation. We compared our techniques with and without the allocation switching technique to the state-of-the-art migration-based hybrid cache management technique, called read-write aware hybrid cache architecture (RWHCA) [17].

### 5.1 Performance

A weighted speedup is used as a performance metric, which is the sum of per-application speedups in IPC compared to the baseline (RWHCA running a single

**Table 3.** Benchmark analysis of SPEC CPU2006

| | **L2 MPKI** | | **-Diff (%)** | **Type** |
|---|---|---|---|---|
| Benchmark | 512kB | 4MB | | |
| quantum | 24.7 | 24.7 | 0.0 | |
| milc | 24.3 | 24.3 | 0.1 | |
| sjeng | 56.2 | 56.1 | 0.1 | Insensitive |
| gobmk | 11.9 | 11.6 | 3.0 | |
| lbm | 35.2 | 33.5 | 4.8 | |
| namd | 0.8 | 0.8 | 7.0 | |
| soplex | 31.9 | 27.2 | 14.9 | |
| mcf | 78.8 | 66.3 | 15.9 | |
| h264 | 0.3 | 0.1 | 65.6 | |
| gromacs | 2.1 | 0.5 | 73.3 | |
| hmm | 2.5 | 0.5 | 78.7 | |
| deal2 | 0.3 | 0.1 | 80.3 | Sensitive |
| perl | 0.5 | 0.1 | 80.8 | |
| astar | 15.8 | 2.3 | 85.5 | |
| povray | 1.2 | 0.2 | 85.8 | |
| gcc | 1.6 | 0.2 | 87.5 | |
| bzip | 4.7 | 0.1 | 97.4 | |
| omnetpp | 102.3 | 2.6 | 97.5 | |

**Table 4.** Workloads from SPEC CPU2006

| | Set | Workloads | | | |
|---|---|---|---|---|---|
| group1 | 1 | soplex | perl | gcc | omnetpp |
| | 2 | soplex | povray | bzip | hmm |
| | 3 | gcc | h264 | hmm | soplex |
| | 4 | perl | povray | h264 | hmm |
| group2 | 5 | bzip | omnetpp | hmm | quantum |
| | 6 | omnetpp | gromacs | soplex | gobmk |
| | 7 | soplex | mcf | gromacs | quantum |
| | 8 | milc | monetpp | h264 | deal2 |
| group3 | 9 | omnetpp | quantum | soplex | sjeng |
| | 10 | omnetpp | h264 | namd | quantum |
| | 11 | sjeng | milc | deal2 | bzip |
| | 12 | quantum | sjeng | gromacs | bzip |
| group4 | 13 | milc | omnetpp | sjeng | quantum |
| | 14 | omnetpp | quantum | sjeng | gobmk |
| | 15 | quantum | milc | lbm | omnetpp |
| | 16 | soplex | quantum | milc | gobmk |
| group5 | 17 | milc | namd | gobmk | lbm |
| | 18 | lbm | milc | sjeng | gobmk |
| | 19 | quantum | sjeng | gobmk | milc |
| | 20 | sjeng | quantum | gobmk | lbm |

application is used as the baseline). Fig. 4 shows the weighted speedups normalized to those of RWHCA (the weighted speedups of RWHCA are also obtained by using the same baseline). The left bar in each workload set represents the speedup of the proposed approach without the switching technique and the right bar represents that with the switching technique.

In the hybrid caches partitioning technique without switching, the performance improvement is 4.9% in geometric mean over the total of 20 workloads. For the workloads in group3, where two cache-sensitive and two cache-insensitive benchmarks are mixed, the performance improvement is 8.7% on average; the useless preemption of cache ways by cache-insensitive benchmarks can degrade the performance of the cache-sensitive applications, and thus the partitioning technique can be very effective in this group. In the case of group5 where four cache-insensitive benchmarks are mixed, the partitioning scheme can improve cache utilization by assigning an optimal number of ways to each application, but the improvement is not as significant as group3.

The hybrid caches partitioning technique with allocation switching improves the performance by 6.7% in geometric mean over the total of 20 workloads. Some workloads show drastic performance improvement such as set1 and set10. These improvements come from some benchmarks that have little store access. So the SRAM is underutilized in these workloads. If both SRAM and STT-RAM caches are fully utilized, then an allocation switching scheme can harm the performance improvement of cache partitioning as shown in set5, but its degradation is not significant.

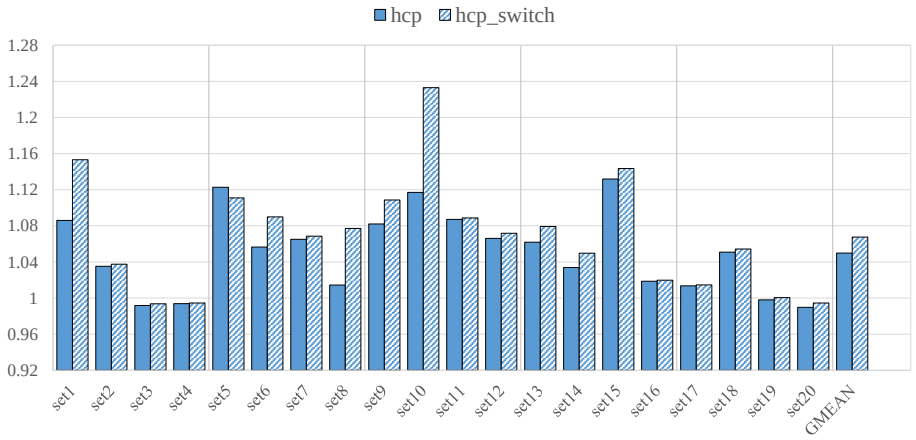


**Fig. 4.** Normalized weighted speedup.

### 5.2 Miss Rates

Fig. 5 reports the difference in L2 miss rates between the RWHCA and our techniques with and without allocation switching. Left bars are the results of the proposed technique without allocation switching, and right bars are the results with switching. The former case reduces the miss rate by 4.8% on average for the 20 workloads. The miss rate increases for set3, but the difference is ignorable. Among the five groups, group3 shows the highest reduction of miss rates (9.2% on average), which is similar to the trend of performance.

In the technique with switching, the miss rate reduction is 6.7% on average compared to the reference technique, which is 1.8% more reduction compared to the technique without switching. The miss rate is reduced by 13.3% for group2 compared to the reference technique.

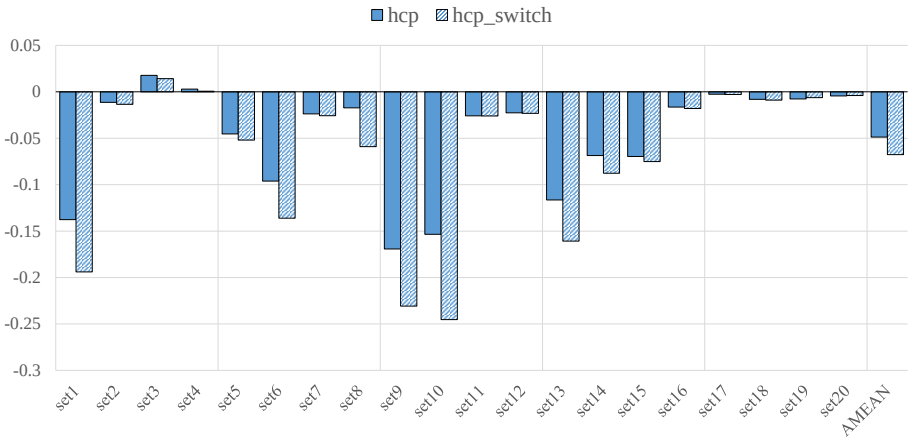


**Fig. 5.** Difference in miss rates.

### 5.3 Cache Energy Consumption

Fig. 6 compares the L2 energy consumption of our techniques with and without a switching scheme against the reference. In the technique without switching, the reduction of energy consumption is 10.0% on average for a total 20 of workloads and 11.7%, 10.4%, 11.0%, 12.0%, and 4.4%, respectively, for group1 to group 5. Workloads in group5 show little energy reduction compared to those of RWHCA because the partitioning technique is not efficient in these workloads.

The technique with switching reduces energy consumption by 16.6% on average over the total of 20 workloads, which corresponds to 6.6% more reduction compared to the technique without switching. All groups except group5 show more reduction of energy consumption. Compared to the reference technique, energy reductions by groups are 12.1%, 22.3%, 22.9%, 20.7%, and 3.2%.

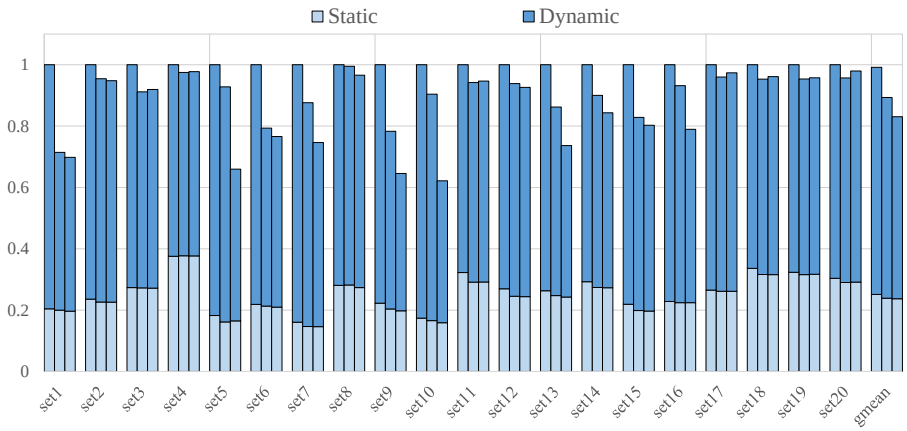


**Fig. 6.** Energy consumption of RWHCA (left), our architecture without allocation switching (middle), and our architecture with allocation switching (right).

### 5.4 DRAM Energy Consumption

Fig. 7 shows the DRAM energy consumption during the simulation. The results of our techniques with and without the switching scheme are normalized to that of RWHCA. In the technique without switching, the decrease of energy consumption is 5.9% on average for the 20 workloads. Group3 shows the highest energy saving of 9.6% on average and group5 shows the lowest of 3.4%.

The technique with allocation switching decreases DRAM energy consumption by 7.5% on average over the 20 workloads, which corresponds to 1.6% more reduction compared to the technique without the switching scheme.

### 5.5 Area Overhead

An area overhead of our technique comes from a partitioning scheme. For cache usage monitoring, every 32nd cache set is sampled, so one UMON has 16 of 4B counters and an ATD of 128 sets for a 4MB 16-way set-associative cache. One ATD consists of 16 entries, each has one valid bit, 24-bit tag and 4-bit LRU information. So the total overhead of one UMON is 7.3KB and the total overhead of cache usage monitoring for a quad-core processor is 29.25KB. In a tag of the cache, 2-bit is added to identify the core that owns each block, the sum of the overheads for the cache is 16KB. So the total overhead of the partitioning scheme for 4MB 16-way set-associative cache is 45.25KB, which is negligible compared to the size of the last-level cache.

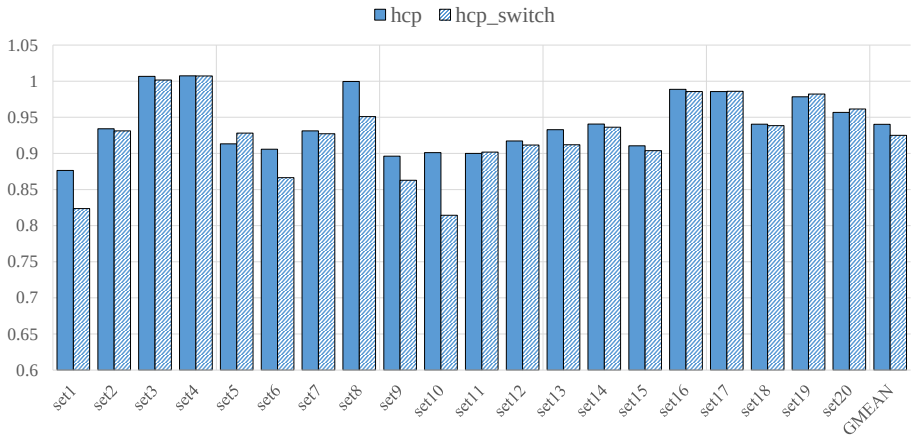


**Fig. 7.** Normalized energy consumption of DRAM.

# 6 Related work

## 6.1 Reducing Write Overhead of STT-RAM

A write intensity predictor [1] is proposed to find a write intensive block and allocate the block to SRAM. This approach achieves high energy reduction of hybrid caches, but does not consider cache partitioning, and thus it may increase miss rates of the cache in some workloads, worsening the DRAM energy efficiency. Obstruction-aware cache management technique (OAP) [16] increases the efficiency of a last-level STT-RAM cache by bypassing some application that has no merits of using the last-level cache. The technique collects information on latency, number of accesses, and miss rates of the applications in a period and exploits the data for the detection of bypassing applications. But the target of this technique is a pure STT-RAM cache, so it can not be applied directly to hybrid caches.

A lot of researches [3, 9, 8, 14, 18, 17] utilize a migration technique for adapting block placements, but in our proposed partitioning technique, a conventional migration scheme [17] can reduce the energy efficiency of hybrid caches by breaking the partitioning decision.

## 6.2 Cache Partitioning for Energy Saving

Cooperative partitioning technique [15] is proposed to save energy consumption of a shared cache. In this technique, a partition is aligned physically and unused ways are disabled to reduce static power consumption. But this technique does not consider hybrid caches, and thus the block placement problem should be solved to apply this technique to hybrid caches. Writeback-aware partitioning [19] assumes an off-chip memory of phase change RAM (PRAM) and reduces

the number of write operations in PRAM by dynamic partitioning of a shared cache to decrease the energy consumption of the write-inefficient memory.

## 7 Conclusion

We address the potential of the dynamic cache partitioning technique for reducing energy consumption of hybrid caches. We propose an energy-efficient partitioning technique for hybrid caches in which the number of blocks installed by a core is adaptively balanced between SRAM and STT-RAM while satisfying a partitioning decision. If a miss is caused by a store miss in the upper-level cache, the corresponding block is placed in SRAM. If the miss is originated from a load miss in the upper-level cache, a new block can be placed in either SRAM or STT-RAM according to the number of blocks installed by the core in SRAM. In addition, we apply the allocation switching technique to avoid too much unbalanced use of SRAM or STT-RAM. The simulation results show that our partitioning technique improves the performance of the multi-core system by 6.7% on average, saves the energy consumption of the hybrid cache by 16.6%, and reduces the DRAM energy by 7.5% compared to the state-of-the-art migration-based hybrid cache management technique.

**Acknowledgment.** This work was supported by the National Research Foundation of Korea (NRF) grants funded by the Korean government (MEST) (No. 2012R1A2A2A06047297).

## References

[1] Ahn, J., Yoo, S., Choi, K.: Write intensity prediction for energy-efficient non-volatile caches. In: Proceedings of the 2013 International Symposium on Low Power Electronics and Design. pp. 223–228. ISLPED '13, IEEE Press, Piscataway, NJ, USA (2013)

[2] Apalkov, D., Khvalkovskiy, A., Watts, S., Nikitin, V., Tang, X., Lottis, D., Moon, K., Luo, X., Chen, E., Ong, A., Driskill-Smith, A., Krounbi, M.: Spin-transfer torque magnetic random access memory (stt-mram). J. Emerg. Technol. Comput. Syst. 9(2), 13:1–13:35 (May 2013)

[3] Chen, Y.T., Cong, J., Huang, H., Liu, C., Prabhakar, R., Reinman, G.: Static and dynamic co-optimizations for blocks mapping in hybrid caches. In: Proceedings of the 2012 ACM/IEEE International Symposium on Low Power Electronics and Design. pp. 237–242. ISLPED '12, ACM, New York, NY, USA (2012)

[4] Dong, X., Xu, C., Xie, Y., Jouppi, N.: Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on 31(7), 994–1007 (July 2012)

[5] Henning, J.L.: Spec cpu2006 benchmark descriptions. SIGARCH Comput. Archit. News 34(4), 1–17 (Sep 2006)

[6] Lee, D., Choi, K.: Energy-efficient partitioning of hybrid caches in multi-core architecture. In: Very Large Scale Integration (VLSI-SoC), 2014 22nd International Conference on. pp. 37–42 (Oct 2014)

[7] Li, J., Xue, C., Xu, Y.: Stt-ram based energy-efficiency hybrid cache for cmps. In: VLSI and System-on-Chip (VLSI-SoC), 2011 IEEE/IFIP 19th International Conference on. pp. 31–36 (Oct 2011)
[8] Li, Q., Li, J., Shi, L., Xue, C.J., He, Y.: Mac: Migration-aware compilation for stt-ram based hybrid cache in embedded systems. In: Proceedings of the 2012 ACM/IEEE International Symposium on Low Power Electronics and Design. pp. 351–356. ISLPED '12, ACM, New York, NY, USA (2012)
[9] Li, Y., Chen, Y., Jones, A.K.: A software approach for combating asymmetries of non-volatile memories. In: Proceedings of the 2012 ACM/IEEE International Symposium on Low Power Electronics and Design. pp. 191–196. ISLPED '12, ACM, New York, NY, USA (2012)
[10] Muralimanohar, N., Balasubramonian, R., Jouppi, N.P.: Cacti 6.0: A tool to model large caches. HP Laboratories (2009)
[11] Patel, A., Afram, F., Chen, S., Ghose, K.: Marss: A full system simulator for multicore x86 cpus. In: Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE. pp. 1050–1055 (June 2011)
[12] Qureshi, M., Patt, Y.: Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches. In: Microarchitecture, 2006. MICRO-39. 39th Annual IEEE/ACM International Symposium on. pp. 423–432 (Dec 2006)
[13] Rosenfeld, P., Cooper-Balis, E., Jacob, B.: Dramsim2: A cycle accurate memory system simulator. Computer Architecture Letters 10(1), 16–19 (Jan 2011)
[14] Sun, G., Dong, X., Xie, Y., Li, J., Chen, Y.: A novel architecture of the 3d stacked mram l2 cache for cmps. In: High Performance Computer Architecture, 2009. HPCA 2009. IEEE 15th International Symposium on. pp. 239–249 (Feb 2009)
[15] Sundararajan, K., Porpodas, V., Jones, T., Topham, N., Franke, B.: Cooperative partitioning: Energy-efficient cache partitioning for high-performance cmps. In: High Performance Computer Architecture (HPCA), 2012 IEEE 18th International Symposium on. pp. 1–12 (Feb 2012)
[16] Wang, J., Dong, X., Xie, Y.: Oap: An obstruction-aware cache management policy for stt-ram last-level caches. In: Design, Automation Test in Europe Conference Exhibition (DATE), 2013. pp. 847–852 (March 2013)
[17] Wu, X., Li, J., Zhang, L., Speight, E., Xie, Y.: Power and performance of read-write aware hybrid caches with non-volatile memories. In: Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09. pp. 737–742 (April 2009)
[18] Wu, X., Li, J., Zhang, L., Speight, E., Rajamony, R., Xie, Y.: Hybrid cache architecture with disparate memory technologies. In: Proceedings of the 36th Annual International Symposium on Computer Architecture. pp. 34–45. ISCA '09, ACM, New York, NY, USA (2009)
[19] Zhou, M., Du, Y., Childers, B., Melhem, R., Mossé, D.: Writeback-aware partitioning and replacement for last-level caches in phase change main memory systems. ACM Trans. Archit. Code Optim. 8(4), 53:1–53:21 (Jan 2012)