Adaptive Placement and Migration Policy for an STT-RAM-Based Hybrid Cache

Zhe Wang[†] Daniel A. Jiménez[†] Cong Xu[‡] Guangyu Sun[♯] Yuan Xie[‡ §]

[†]Texas A&M University [‡]Pennsylvania State University [♯]Peking University [§]AMD Research

[†]{zhewang, djimenez}@cse.tamu.edu [‡]{czx102, yuanxie}@cse.psu.edu [♯]gsun@pku.edu.cn

Abstract

Emerging Non-Volatile Memories (NVM) such as Spin-Torque Transfer RAM (STT-RAM) and Resistive RAM (RRAM) have been explored as potential alternatives for traditional SRAM-based Last-Level-Caches (LLCs) due to the benefits of higher density and lower leakage power. However, NVM technologies have long latency and high energy overhead associated with the write operations. Consequently, a hybrid STT-RAM and SRAM based LLC architecture has been proposed in the hope of exploiting high density and low leakage power of STT-RAM and low write overhead of SRAM. Such a hybrid cache design relies on an intelligent block placement policy that makes good use of the characteristics of both STT-RAM and SRAM technology.

In this paper, we propose an adaptive block placement and migration policy (APM) for hybrid caches. LLC write accesses are categorized into three classes: prefetch-write, demand-write, and core-write. Our proposed technique places a block into either STT-RAM lines or SRAM lines by adapting to the access pattern of each class. An access pattern predictor is proposed to direct block placement and migration, which can benefit from the high density and low leakage power of STT-RAM lines as well as the low write overhead of SRAM lines.

Our evaluation shows that the technique can improve performance and reduce LLC power consumption compared to both SRAM-based LLC and STT-RAM-based LLCs with the same area footprint. It outperforms the SRAM-based LLC on average by 8.0% *for single-thread workloads and* 20.5% *for multi-core workloads. The technique reduces power consumption in the LLC by* 18.9% *and* 19.3% *for single-thread and multi-core workloads, respectively.*

1 Introduction

Emerging non-volatile memory (NVM) technologies, such as Phase-Change RAM (PCRAM), Resistive RAM (ReRAM), and Spin-Torque Transfer RAM (STT-RAM), are potential alternatives for future memory architecture design due to their higher density, better scalability, and lower leakage power. With all the NVM technology advances in recent years, it is anticipated that NVM technologies will closer to market in the near future.

Among these three memory technologies, PCRAM and ReRAM have slower performance and very limited lifetime (usually in the range of 10^8-10^{10}) and therefore can only be suitable as storage-class memory or at most as main-memory. STT-RAM, on the other hand, has better read/write latency and better lifetime reliability, so it can potentially be used as the SRAM replacement for last-level cache (LLC) design.

However, the latency and energy of write operations to STT-RAM are significantly higher than those of read operations. The long write latency can degrade performance by causing large write-induced interference to subsequent read requests. High write energy can increase power consumption of the LLC.

As both STT-RAM and SRAM have strengths and weaknesses, hybrid cache design [25, 9, 3, 2, 26] has been proposed in the hope of benefiting from the high density, low leakage power of STT-RAM as well as low write overhead of SRAM. In the hybrid cache, each set consists of a large portion of STT-RAM lines and a small portion of SRAM lines. Hybrid cache design depends on an intelligent block placement policy that can achieve high performance by making use of the large capacity of STT-RAM and maintain low write overhead using SRAM.

Recent work proposes [25, 9, 3, 2, 26] block placement and migration policies to map write-intensive blocks to SRAM lines. Prior proposals reduce the average write operations to STT-RAM lines. However, these proposals incur frequent block migration [9, 3], require the compiler to provide static hints [3], mitigate write overhead only for a subset of write accesses [25], or trade off performance for power or endurance [9, 2, 26].

This paper describes a low cost adaptive block placement policy for a hybrid LLC called APM (Adaptive Placement and Migration). The technique places a cache block in the proper position according to its access pattern. LLC write

978-1-4799-3097-5/14/$31.00

accesses are categorized into three distinct classes : *core-write*, *prefetch-write* and *demand-write*. *Core-write* is a write from the core. For a write-through core cache, it is the write directly from the core writing through to the LLC. For a write-back core cache, it is dirty data evicted from the core cache and written back to the LLC. *Prefetch-write* is a write from the LLC replacement caused by a prefetch miss. *Demand-write* is a write from the LLC replacement caused by a demand miss. The proposed technique is based on the observation that block placement is often initiated by a write access. The characteristics of each class of write enable them to be adapted to different block placement policies. A low overhead and low complexity pattern predictor is used to predict the access pattern for each class of accesses for directing the block placement.

Our contributions can be summarized as follows: (1) We analyze the LLC access pattern for distinct write access types: prefetch-write, demand-write and core-write. We propose that each class should be adapted to different block placement and migration policies for a hybrid LLC. We design a low overhead access pattern predictor. It predicts write burst blocks and dead blocks in the LLC that can be used to direct block placement and migration. (2) We propose an intelligent cache placement and migration policy for a hybrid STT-RAM and SRAM based LLC. The technique optimizes block placement by adapting to the access pattern of each class of write accesses. (3) We show the proposed technique can improve system performance and reduce LLC power consumption of hybrid LLC compared to both SRAM-based LLC and STT-RAM-based LLC with the same area. It outperforms an SRAM-based LLC on average by 8.0% for single-thread workloads and by 20.5% for multi-core workloads. The technique reduces the power consumption of the LLC by 18.9% and 19.3% respectively for single-thread and multi-core workloads compared with the baseline.

2 Background

2.1 Comparison of STT-RAM and SRAM Cache

Compared to SRAM, STT-RAM caches have higher density and lower leakage power, but higher write latency and write power consumption. Table 1 lists the technology features of various STT-RAM cache bank sizes and SRAM cache bank sizes used in our evaluation. The technology parameters are generated by NVSim [5], a performance, energy, and area model based on CACTI [8]. The cell parameters we used in NVSim are based on the projection from Wang *et al.* [28]. We assume a 22nm $\times$ 45nm MTJ built with 22nm CMOS technology. The SRAM cell parameters are estimated using CACTI [8].

The density of STT-RAM is currently 3$\times$-4$\times$ higher than SRAM. Another benefit for STT-RAM is its low leakage power. Leakage power can dominate the total power consumption for large SRAM-based LLCs [15]. Thus, the low leakage power consumption of STT-RAM makes it suitable for a large LLC. Disadvantages of STT-RAM are long write latency and high write energy.

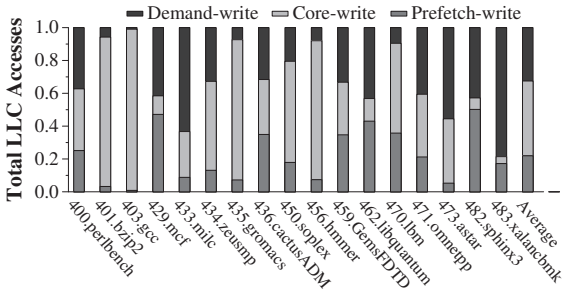

Figure 1. Distribution of LLC write accesses. Each type of write access accounts for a significant fraction of total write accesses

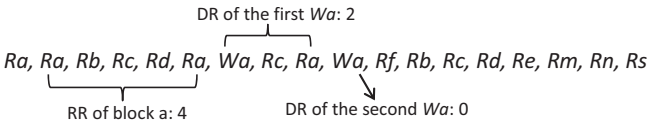

Figure 2. An example illustrating read range and depth range

2.2 Hybrid Cache Structure

The hybrid cache structure is composed of STT-RAM banks and SRAM banks. Each cache set consists of a large portion of STT-RAM cache lines and a small portion of SRAM cache lines distributed among multiple banks. The hybrid cache architecture relies on an intelligent block placement policy to bridge the performance and power gaps between STT-RAM and SRAM.

An intelligent block placement policy for hybrid cache design should be optimized for three requirements. First, the SRAM portion should service as many write requests as possible, thus minimizing the write overhead of STT-RAM portion. However, sending many write operations to SRAM without considering the access pattern can cause misses due to the small capacity of SRAM, leading to performance degradation. Thus, the second requirement is that reused cache blocks should be placed in the LLC to maintain performance by hiding the memory access latency. Finally, the block placement policy should be a low overhead and low complexity design without incurring frequent migration between cache lines.

3 Analysis of LLC Write Access Patterns

LLC block placement is often initiated by a LLC write access that can be categorized into three classes: *prefetch-write, core-write and demand-write*. Figure 1 shows the breakdown of each class of LLC write accesses for 17 memory intensive SPEC CPU2006 [7] benchmarks. The study is performed using the MARSSx86 [19] simulator

Memory Type	1M SRAM	2M SRAM	2M STT-RAM	4M STT-RAM
Area (mm^2)	0.825	1.650	0.518	1.035
Read Latency (ns)	1.751	2.017	2.681	2.759
Write Latency (ns)	1.530	1.663	10.954	10.993
Read Energy (nJ/access)	0.055	0.072	0.132	0.142
Write Energy (nJ/access)	0.039	0.056	0.608	0.618
leakage power (mW)	29.798	59.596	7.108	14.216

Table 1. Characteristics of SRAM and STT-RAM caches (22nm, temperature=350K)

with single-core configuration and a 4MB LLC. We implement a middle-of-the-road stream prefetcher that models the prefetcher of the Intel Core i7 [1]. From Figure 1, we can see that each type of write access accounts for a significant fraction of total write accesses. Prefetch-writes account for 21.9% on average while core-writes and demand-writes take 45.6% and 32.5%, respectively. In this section, we analyze the access pattern of each write access type and suggest a block placement policy that adapts to the access pattern for each class.

We first define the terminology that will be used later in this section for pattern analysis. To be clear, when we write "block" we mean a block of data apart from its physical realization. When we write "line" we mean the physical frame, whether in STT-RAM or SRAM, where a block may reside.

- **Read-range**: The read-range is a property of a cache block that fills the LLC by a demand-write or prefetch-write request. It is the largest interval between consecutive reads of the block from the time it is filled into the LLC until the time it is evicted.
- **Depth-range**: The depth-range is a property of a core-write access. It is the largest interval between accesses to the block from the current core-write access until the next core-write access to the same block. The "depth" refers to how deep the block descends into the LRU recency stack before it is accessed again.

We use an example to illustrate the read range and depth range. Figure 2 shows the behavior of a block a from the time it fills into a 8-way set until it is evicted. In the example, Ra represents "read block a" while Wa represents "write block a". The largest re-read interval of block a during the time it resides in the cache is 4 which is the read interval between the second Ra and the third Ra. Thus, the read range (RR) of block a is 4. The depth range (DR) of the first Wa access is 2 which is the access interval between the first Wa access and the fourth Ra access. The depth range of the second Wa access is 0 meaning a is not re-written from the second Wa until it is evicted from the LLC. We further classify the read/depth-range into three types: *zero-read/depth-range*, *immediate-read/depth-range* and *distant-read/depth-range*.

- Zero-read/depth-range: Data is filled into the LLC by a prefetch or demand request/core-write request, and it is never read/written again before it is evicted.
- Immediate-read/depth-range: The read/depth-range is smaller than a parameter m. We set $m = 2$ which is the same as the number of SRAM ways in our hybrid cache configuration as in Section 5.
- Distant-read/depth-range: The read/depth-range is larger than $m = 2$ and at most the associativity of the cache set which is 16 in our configuration.

3.1 Pattern Analysis of Prefetch-Write Block

Prefetching data into the cache before it is accessed can improve performance by hiding memory access latency. However, prefetching can also induce cache pollution by inaccurate prefetch requests.

We analyze the access pattern for the LLC prefetch-write blocks by using read-range. The first bar in Figure 3 shows each type of access pattern as a fraction of the total number of prefetch-write blocks. Zero-read-range prefetch-write blocks are inaccurately prefetched blocks accounting for 26% of all of prefetch blocks. Placing the zero-read-range prefetch-write blocks into the STT-RAM lines causes pollution and high write overhead. Thus, zero-read-range prefetch-write blocks should be placed in SRAM lines.

The immediate-read-range access pattern is related to cache bursts. After an initial burst of references, a cache block becomes dead, i.e. it is never used again prior to eviction. Of all prefetch-write blocks, 56.9% are immediate-read-range blocks. Immediate-read-range prefetch-write blocks should be placed in SRAM lines so they can be accessed by subsequent demand requests without incurring write operations to STT-RAM lines. Moreover, placing immediate-read-range prefetch-write blocks in SRAM allows them to be evicted when they are dead, reducing pollution.

Distant-read-range prefetch-write blocks should be placed in STT-RAM lines to make use of the large capacity to avoid cache misses. Distant-read-range prefetch-write blocks account for only 17.5% of all prefetch blocks.

Zero-read-range and immediate-read-range prefetch-write blocks account for 82.5% of all prefetch-write blocks.

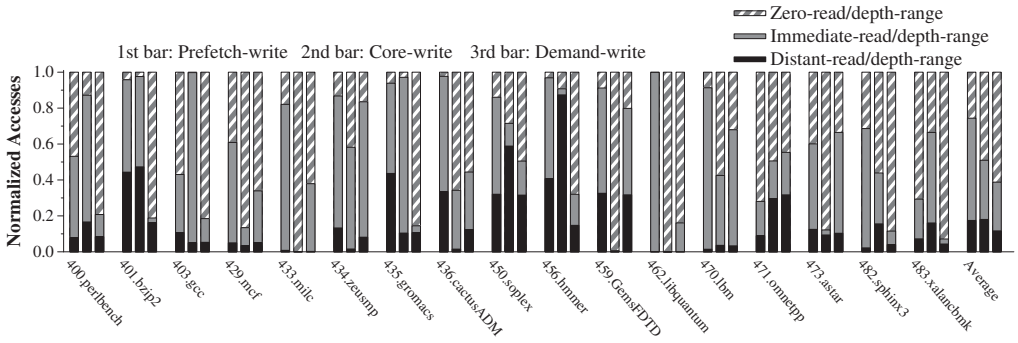

Figure 3. The distribution of access pattern for each type of LLC write access

Thus, we suggest the initial placement of the prefetch-write blocks in SRAM. Once a block is evicted from the SRAM, if it is a distant-read-range block, i.e. it is still live, it should be migrated to STT-RAM lines. Otherwise, the block is dead and should be evicted from the LLC. Since only 17.5% of prefetch blocks are distant-access prefetch blocks, the migration will not cause significantly increased traffic.

3.2 Pattern Analysis of Core-Write Access

In our design, if a core-write access misses in the LLC, the data will be written back to the main memory directly. Thus, our core-write placement policy is only designed for core-write hit accesses.

We analyze the access pattern of the LLC core-write access by using depth-range. The second bar in Figure 3 shows the access pattern for core-write accesses. Zero-depth-range accesses account for 49.1% of all core-write accesses. Though the data written by the zero-depth-range core-write access will not be rewritten before it is evicted, it still has some chance to be read again. Thus, we suggest leaving zero-depth-range data in its original cache line for avoiding read misses and block migrations.

Immediate-depth-range accesses account for 32.9% of total core-write accesses. The immediate-depth-range accesses are the write-intensive accesses with write burst behavior. Thus, the immediate-depth-range access data is preferred to be placed in the SRAM line for low write overhead. The distant-depth-range access data should remain in its original cache line, thus minimizing migration overhead.

3.3 Pattern Analysis of Demand-Write Block

The access pattern of demand-write blocks is analyzed using read-range. Zero-read-range demand-write blocks, also known in the literature as "dead-on-arrival" blocks, are brought to the LLC by a demand request and never referenced again before being evicted. It is unnecessary to place zero-read-range demand-write block into the LLC so the block should bypass the cache (assuming a non-inclusive cache). The third bar in Figure 3 shows dead-on-arrival blocks account for 61.2% of LLC demand-write blocks. Thus, bypassing dead-on-arrival blocks can significantly reduce write operations to LLC. Moreover, bypassing can improve cache efficiency by allowing the LLC to save space for other useful blocks in the cache.

The immediate-read-range and distant-read-range demand-write blocks account for 38.8% of the total demand-write blocks. We suggest placing them in the STT-RAM ways for making use of the large capacity of the STT-RAM portion and reducing pressure on the SRAM portion.

3.4 Pattern Analysis Conclusions

Each class of LLC write access can be applied to a different placement policy. The access pattern of each class type can be used to guide the block placement policy. From the analysis of the access pattern of each access class, we make the following conclusions: (1) The initial placement of prefetch-write blocks should be to SRAM lines. (2) Write-burst core-write data should be placed in SRAM lines while other types of core-write data should remain in their original cache lines. (3) Dead-on-arrival demand-write blocks should bypass the LLC while the other types of demand-write blocks should be placed in the STT-RAM lines. (4) When a block is evicted from SRAM, if it is live, it should be migrated to STT-RAM lines for avoiding LLC misses.

4 Adaptive Block Placement and Migration Policy

The design of the block placement and migration policy is guided by the access pattern of each write access type. An access pattern predictor is proposed to predict write-burst blocks and dead blocks. The information provided by the access pattern predictor is used to direct bypass and migration of blocks between STT-RAM lines and SRAM lines. The policy targets reducing write overhead by allowing the SRAM portion to service as many write requests as possible and attain high performance by benefiting from the high density of the STT-RAM portion.

4.1 Policy Design

Figure 4 shows a flow-chart for the technique. In our design, a prediction bit is associated with each block in the

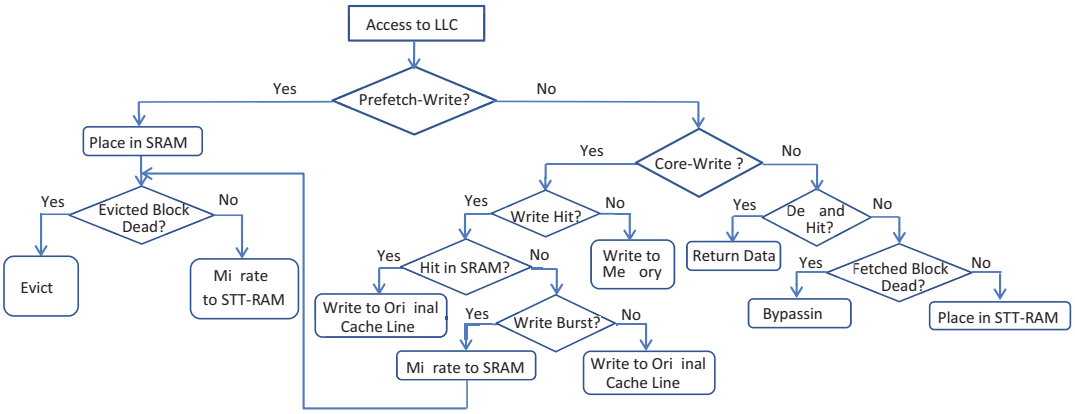

Figure 4. Flow-chart of the adaptive block placement and migration mechanism

LLC for representing whether the block is predicted dead. An access to the LLC searches all the STT-RAM lines and SRAM lines in the set. On a prefetch miss, the prefetched data is placed into an SRAM line and the prediction bit is set to 1 meaning we assume the prefetch block is dead on arrival. For every demand hit request in the SRAM lines, the access pattern predictor makes a prediction about whether the block is dead. Once the block is evicted from the SRAM lines, if it is predicted dead, it will be evicted from the LLC. Otherwise, it will be migrated to the STT-RAM lines. In this case, if a prefetch block is never accessed before it is evicted from the SRAM lines, it is taken as an inaccurately prefetched block and evicted based on the observation that accurately prefetched blocks are usually accessed soon by subsequent demand requests.

On a core-write request to the LLC, if it is an LLC miss, it will be written to main memory directly. For the core-write hit request in the STT-RAM lines, if it is predicted to be a write burst access, then data will be written to the SRAM lines with the prediction bit set to 0 indicating the block is predicted live; the prior position in the STT-RAM line is set to invalid. Then, as with prefetched blocks, once the data is evicted from the SRAM portion, a predicted dead block is evicted from the LLC while a live block is migrated to the STT-RAM portion.

On a demand miss to the LLC, the data is fetched from the main memory. If the fetched block is predicted to be a dead-on-arrival block, it will bypass the LLC. Otherwise, the block will be placed in the STT-RAM lines.

Minimizing Write Overhead The proposed scheme reduces write operations to STT-RAM portion in the following ways. First, bypass can reduce write operations to STT-RAM lines caused by dead-on-arrival requests. Second, SRAM lines filter the write operations caused by the inaccurate and immediate-read-range prefetch requests. Finally, the core-write-intensive blocks are placed in the SRAM lines, reducing the write operations to STT-RAM lines caused by write burst behavior.

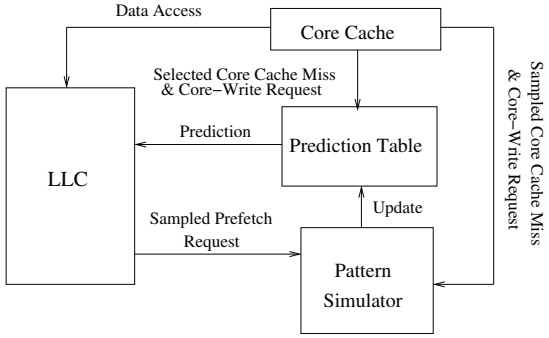

Figure 5. System structure

Attaining High Performance The block placement policy can attain high performance for the hybrid cache by benefiting from the high density of the STT-RAM portion. Specifically, the distant-read-range blocks are placed in STT-RAM lines that can reduce cache misses by making use of the large capacity of STT-RAM. Also, bypassing zero-range demand blocks saves space in the LLC for other useful data. Moreover, filtering zero-range prefetch blocks and immediate-read-range blocks using SRAM lines can improve cache efficiency by reducing inaccurately prefetched blocks and dead blocks in the LLC.

The technique relies on an access pattern predictor for directing block bypassing and migration.

4.2 Access Pattern Predictor

The goal of the access pattern predictor is to predict dead blocks and write burst blocks for guiding block placement. The predicted dead blocks are used to direct bypassing and block migration from SRAM lines to STT-RAM lines. Write burst blocks are used to guide block migration from STT-RAM lines to SRAM lines for core-write ac-

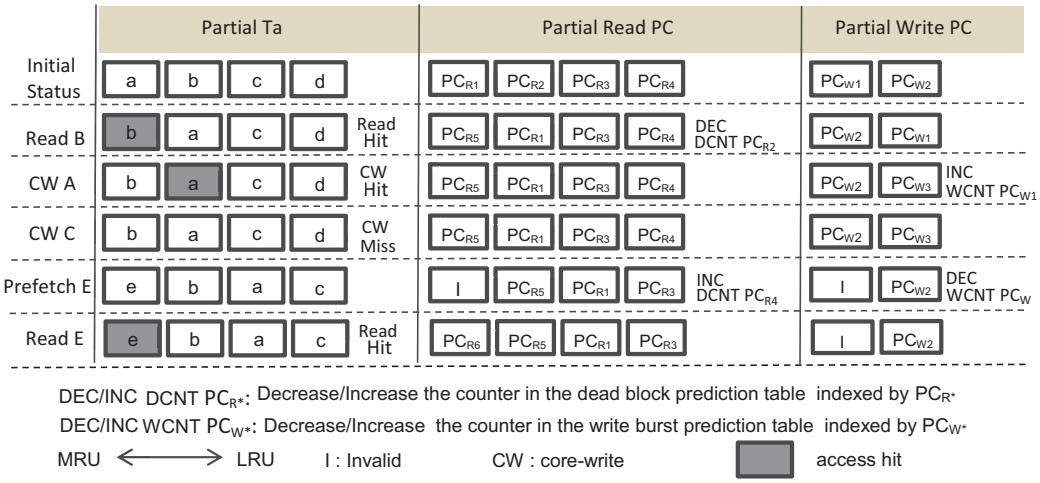

Figure 6. An example illustrating the set behavior of pattern simulator

cesses. The access pattern predictor consists of a prediction table and a pattern simulator as shown in Figure 5. The prediction table is composed of a dead block prediction table and a write burst prediction table having the same structure but making predictions for different types of accesses. The pattern simulator is used to learn the access pattern for updating the prediction table.

The design of the access pattern predictor is inspired by the sampling dead block predictor (SDBP) [13]. However, the SDBP predicts dead blocks only taking into account demand accesses, while the access pattern predictor predicts both dead blocks and write-intensive blocks by considering all types of LLC accesses. The predictor predicts access pattern using the Program Counter (PC) of memory access instructions. The intuition is the cache access pattern can be classified based on the instructions of the memory accesses. Specifically, if a given memory access instruction PC leads to some access pattern in previous accesses, then the future access pattern of same PC will be similar.

4.2.1 Making Prediction

The access pattern predictor makes a prediction in the following three conditions: (1) When a core-write request hits in the STT-RAM lines, the write burst prediction table will be accessed to predict whether it is a write burst request. (2) For each read hit request in the SRAM lines, the dead block prediction table will be accessed to predict whether it is a dead block. (3) On a demand-write request, dead block prediction table will be accessed to predict whether it is a dead-on-arrival request.

The dead block prediction and write burst prediction tables have the same structure. Each entry in a prediction table has a two-bit saturating counter. When making a prediction, bits from the related memory access instruction PC are hashed to index a prediction table entry. The prediction result is generated by thresholding the counter value in the prediction table entry. In our implementation, we use a skewed organization [29, 13, 18] to reduce the impact of conflicts in the hash table.

4.2.2 Updating Predictor

The pattern simulator samples LLC sets and simulates the access behavior of the LLC by using sampled sets. It updates the prediction table by learning the access pattern of the PCs from the simulated LLC access behavior. It targets learning the dead/live behavior and write burst behavior of LLC blocks.

Each simulated set in the pattern simulator has a tag field, a read PC field, an LRU field, a write PC field, a write LRU field and a valid bit field. The partial tag and partial read/write PC which are the lower 16-bit of the full tag and full read/write PC are stored in the tag field and read/write PC field. The read PC field is for learning the dead/live behavior of the block while the write PC field is for learning the write-burst behavior of the block.

A write burst occurs within a small number of cache lines. Thus the write PC field should have a small associativity. The pattern simulator consists of two parts: the tag array and its related read PC field, LRU field and valid bit field which have the same associativity while the write PC field and its write LRU field have a smaller associativity. In our implementation, we found 4-way associativity of the write PC field and 12-way associativity of the read PC field yield the best performance while the associativity of LLC is 18.

The behavior of a pattern simulator set is illustrated in Figure 6 using an example access pattern. In the example, the associativity of the tag and read PC fields is 4 while the associativity of the write PC field is 2. On each demand hit request, the pattern simulator updates the dead block pre-

diction table entry indexed with the related read PC by decreasing the counter value indicating "not dead." When a block is evicted from the simulator set, the pattern simulator updates the dead block prediction table entry indexed with the related read PC by increasing the counter value indicating "dead." The LRU recency is updated for every demand request and prefetch-write request. The write PC field is for learning the write-burst behavior for core-write requests. On each core-write hit request, the simulator increments the counter value stored in the write burst prediction table entry indexed with the related write PC indicating "write burst." When a block is evicted from the write field, the simulator decrements the counter value stored in the write burst prediction table entry indexed with the related write PC indicating "not write burst."

5 Evaluation

5.1 Methodology

Execution core	4.0GHZ, 1-core/4-core CMP, out of order, 128 entry reorder buffer, 48 entry load queue, 44 entry store queue, 4 width issue/decode
Caches	L1 I-cache: 64KB/2 way, private, 2-cycle 64 bytes block size, LRU L1 D-cache: 64KB/2 way, private, 2-cycle 64 bytes block size, LRU L2 Cache: shared, 64 bytes block size, LRU
DRAM	DDR3-1333, open-page policy, 2-channel, 8-bank/channel, FR_FCFS [21] policy, 32-entry/channel write buffer, drain_when_full write buffer policy

Table 2. System configuration

Name	Technique
SRAM	SRAM-based LLC
STT-RAM	STT-RAM-based LLC
OPT-STT-RAM	STT-RAM-based LLC Assuming symmetric read/write overhead
Sun-Hybrid	Hybrid LLC technique as described in [25]
APM	Adaptive placement and migration based hybrid cache as described in Section 4

Table 3. Legend for various LLC techniques.

We use MARSSx86 [19], a cycle-accurate simulator for the 64-bit x86 instruction set. The DRAMSim2 [22] simulator is integrated into MARSSx86 to simulate DDR3-1333 system. Table 2 lists the system configuration. We model a per-core LLC middle-of-the-road stream prefetcher [24, 1] with 32 streams for each core. The prefetcher looks up the stream table at each LLC request for issuing eligible prefetch requests. The LLC is configured with multiple banks. Requests to different banks are serviced in parallel. Within the same bank, requests are be pipelined. The LLC is implemented with single-port memory bitcell. We obtain STT-RAM and SRAM parameters using NVSim [5] and CACTI [8] as shown in Table 1.

Name	Benchmarks
Mix 1	milc gcc xalancbmk tonto
Mix 2	gamess soplex libquantum perlbench
Mix 3	gcc sphinx3 GemsFDTD tonto
Mix 4	lbm mcf cactusADM GemsFDTD
Mix 5	zeusmp bzip2 astar libquantum
Mix 6	mcf soplex zeusmp bwaves
Mix 7	omnetpp lbm cactusADM sphinx3
Mix 8	bwaves libquantum mcf GemsFDTD
Mix 9	omnetpp cactusADM tonto gcc
Mix 10	soplex mcf bzip2 gcc
Mix 11	perlbench sphinx3 libquantum lbm

Table 4. Multi-Core workloads

The SPEC CPU2006 [7] benchmarks are used for the evaluation. We evaluate five LLC techniques based on the same area configuration. Table 3 shows the legends for these techniques referred to in the graphs that follow. The OPT-STT-RAM technique assumes write operations have similar access latency to read operations which is the optimistic case. The Sun-Hybrid [25] technique assumes that a cache block that has been consecutively written to the LLC twice is a write-intensive block. The technique migrates the write-intensive blocks to SRAM lines for reducing the write operations to STT-RAM lines. The APM technique is our proposed adaptive block placement and migration policy based hybrid cache technique as described in section 4. We modify MARSSx86 to support all types of LLC listed in Table 3.

Single-Core Workloads and LLC Configuration Of the 29 SPEC CPU2006 benchmarks, 22 can be compiled and run with our infrastructure. We use all 22 of these benchmarks for evaluation including both memory-intensive benchmarks and non-memory-intensive benchmarks. For each workload, we simulate 250 million instructions from a typical phase identified by SimPoint [23].

Various LLC techniques are evaluated with the same area. A 2MB SRAM has similar area to a 6MB STT-RAM. Thus, we evaluate a 16-way SRAM with 2MB capacity and 24-way STT-RAM/OPT-STT-RAM with 6MB capacity. The hybrid cache design for the APM technique has 16 STT-RAM lines and 2 SRAM lines in each set and hence we evaluate a 4.5MB APM hybrid cache which has the same area with a 2MB SRAM. In the Sun-Hybrid technique, each cache set allocates 1 SRAM line. Thus we evaluate a 20 STT-RAM lines and 1 SRAM line hybrid cache with a 5.25MB capacity for Sun-Hybrid technique. We implement a 1MB SRAM cache bank and 2MB STT-RAM cache bank for a single-core configuration yielding the best trade off of access latency and bank level parallelism. If the capacity of SRAM is smaller than 1M, it is configured as one bank.

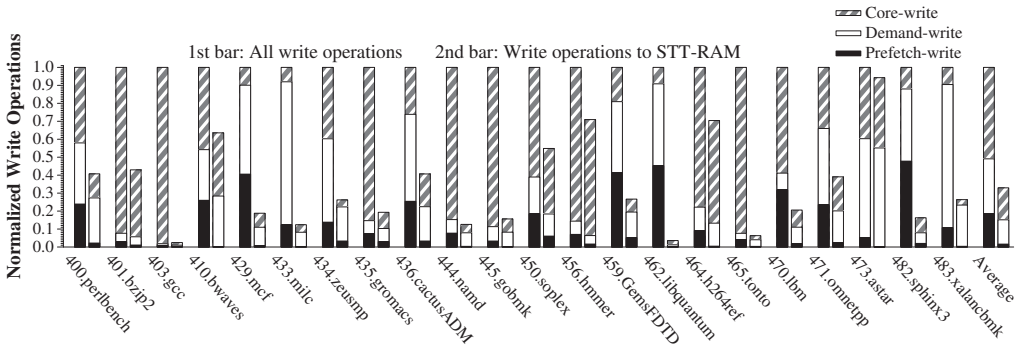

Figure 7. The distribution of write accesses to STT-RAM lines in APM LLC for single-core applications

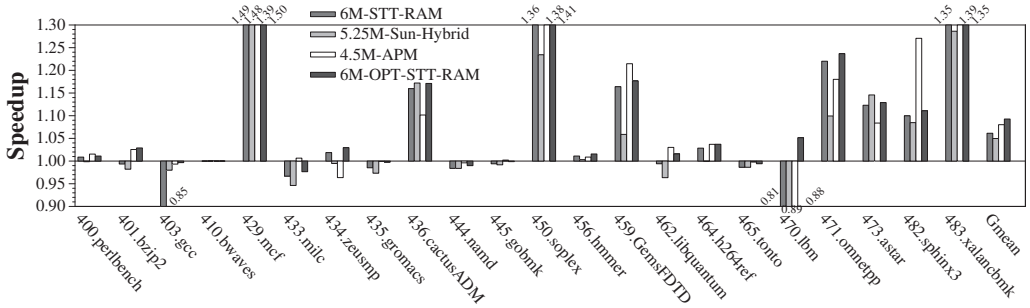

Figure 8. The comparison of IPC for single-core applications (normalized to 2M SRAM LLC)

Multi-Core Workloads and LLC Configuration We use quad-core workloads for evaluation. Table 4 shows eleven mixes of SPEC CPU2006 benchmarks with a variety of memory behaviors. For each mix, we run the experiment with 1 billion instructions total for all four cores starting from the typical phase. Each benchmark runs simultaneously with others. For the multi-core configuration, we evaluate a 16-way SRAM with 8MB capacity, 24-way STT-RAM/OPT-STT-RAM with 24MB capacity, 16-way STT-RAM and 2-way SRAM Hybrid APM technique with 18MB capacity, and 21MB 20-way STT-RAM and 1-way SRAM Sun-Hybrid technique. In the multi-core configuration, we use 2MB SRAM cache bank and 4MB STT-RAM cache bank which yield best performance. If the capacity of SRAM is smaller than 2M, it is configured as one bank.

5.2 Evaluation Results for Single-Core Configuration

5.2.1 Reduced Writes and Endurance Evaluation

The APM technique allows SRAM lines to service as many write requests as possible, thus reducing write operations to STT-RAM lines. Figure 7 shows the distribution of write operations to STT-RAM lines in the APM technique normalized to all write operations to the LLC for single-core workloads. We can see the APM LLC reduces write operations to STT-RAM lines for each type of write accesses. In APM LLC, only 32.9% of the total LLC write requests are serviced by the STT-RAM portion, significantly reducing the write overhead of the STT-RAM portion and translating into performance improvement and power reduction of the LLC.

5.2.2 Performance Evaluation

Figure 8 shows the speedup for various techniques compared with baseline technique which is 2MB SRAM LLC. The 6MB STT-RAM LLC has similar area to 2MB SRAM LLC. It improves the performance by 6.2% on average due to the increased capacity. Most of the benchmarks can benefit from the increased capacity of STT-RAM. However, several benchmarks such as `gcc`, `milc`, `libquantum` and `lbm` suffer more from the large write overhead of STT-RAM. The OPT-STT-RAM LLC assumes symmetric read/write latency meaning large write-induced interference to read request is removed. It yields a geometric mean speedup of 9.3%. The performance difference between OPT-STT-RAM and STT-RAM is caused by the long write latency of STT-RAM. The 4.5MB APM LLC reduces the write overhead of the STT-RAM portion, delivering a geometric mean speedup of 8.0%. It yields even higher speedup for benchmarks `462.libquantum`, `482.sphinx3`, and `483.xalancbmk` than the 6MB OPT-STT-RAM LLC. Because those workloads generate a large number of dead blocks or inaccurate prefetch blocks in the LLC and hence reducing the LLC efficiency. The APM technique reduces the LLC pollution cased by dead blocks

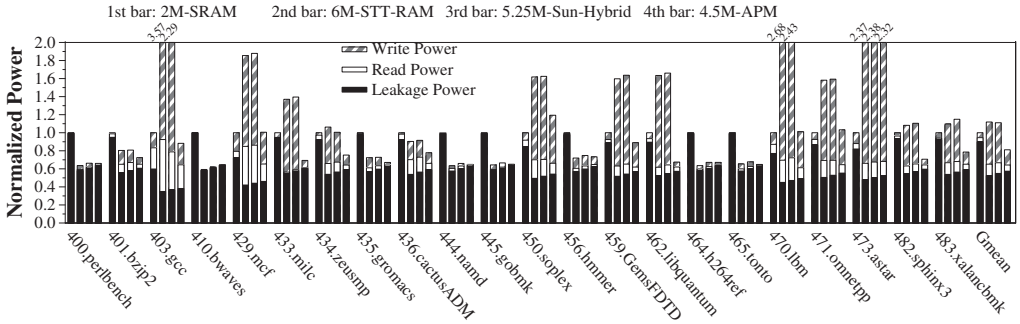

Figure 9. The power breakdown for single-core applications (normalized to 2MB SRAM)

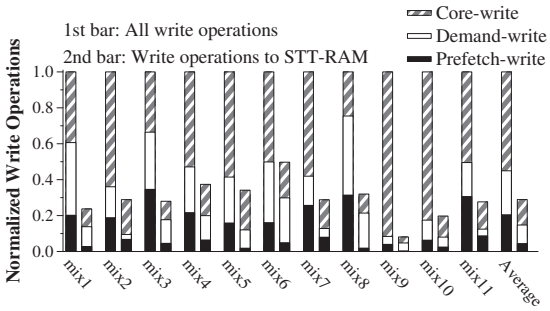

Figure 10. The distribution of write accesses to STT-RAM lines in APM LLC for multi-core applications

and inaccurate prefetch blocks, thus improving the LLC efficiency for those workloads. The 5.25MB Sun-Hybrid LLC improves the performance by 5.0% on average.

5.2.3 Power Evaluation

Figure 9 shows the normalized power consumption for various techniques due to leakage power, dynamic power caused by reads and dynamic power caused by writes. The baseline technique is a 2MB SRAM. For the SRAM technique, the leakage power dominates the total power consumption. The STT-RAM technique consumes lower leakage power. However, the dynamic power caused by writes is significantly increased due to the large write energy of STT-RAM. Thus the STT-RAM technique increases the overall power consumption by 11.9% on average. The APM technique reduces write operations to the STT-RAM portion, thus reducing the dynamic power caused by writes. It reduces the overall power consumption by 18.9% on average compared with the baseline. The Sun-Hybrid technique does not significantly reduce the write power because the Sun-Hybrid technique does not reduce write operations to STT-RAM caused by LLC replacement.

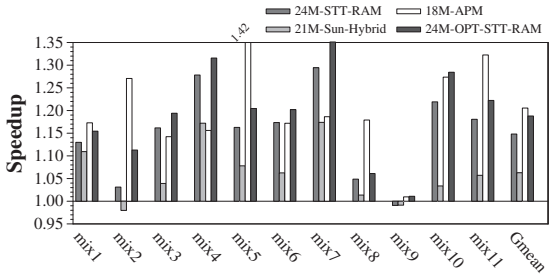

Figure 11. The comparison of IPC for multi-core applications (normalized to 8MB SRAM)

5.2.4 Extra LLC Traffic Evaluation

The APM technique migrates blocks between SRAM lines and STT-RAM lines. Migrating blocks from SRAM lines to STT-RAM lines causes extra cache traffic. We evaluate the LLC traffic caused by migration. Migration causes only 3.8% extra LLC traffic. Most of the blocks evicted from SRAM ways are dead blocks, thus only a small number of distant-read-range blocks need to be migrated to STT-RAM. Thus, the small percentage of traffic caused by migration will not cause significant traffic overhead.

5.3 Evaluation Results for Multi-Core Configuration

5.3.1 Reduced Writes, Endurance, Performance and Power Evaluation

Figure 10 shows the distribution of write operations to STT-RAM lines normalized to all write operations to LLC for the multi-core workloads. The APM technique reduces write operations to the STT-RAM portion to 28.9% on average of the total number of write operations.

Figure 11 shows the speedups of the various techniques for the multi-core workloads normalized to 8MB SRAM LLC. The 24MB STT-RAM LLC improves performance by 14.8% on average. Removing write-induced interference caused by asymmetric writes improves the average performance by 18.7% in the OPT-STT-RAM LLC. The 18M APM technique reduces write overhead of the STT-RAM portion. It achieves a geometric mean speedup of 20.5%

which is higher than the 24MB OPT-STT-RAM LLC. In multi-core workloads, the large number of dead or inaccurately prefetched blocks generated from one workload can also negatively affect the performance of other workloads, significantly reducing performance. The APM technique reduces cache pollution caused by dead blocks and inaccurately prefetched blocks. Our evaluation shows the 18MB APM LLC yields better performance and fewer misses for 5 out of 11 multi-core workloads compared with 24MB OPT-STT-RAM LLC.

Figure 12 shows the distribution of normalized power consumption for various techniques. The baseline is the 8MB SRAM cache. For a large LLC, the majority of power consumption comes from leakage power. The 24MB STT-RAM technique reduces overall power consumption to 87.8% of baseline due to low leakage power. The APM technique reduces dynamic power caused by the STT-RAM write operations. Thus, it further reduces power consumption to 80.7% of baseline on average.

5.3.2 Prediction Evaluation

We evaluate the access pattern predictor using false positive rate and coverage. Mispredictons can be false positives and false negatives. False positives are more harmful for two reasons: mispredicting a live block as a dead block can cause bypass or eviction of a live block from LLC early and generate LLC misses, and mispredicting a non-write-burst request as a write-burst request can cause extra migrations between STT-RAM lines and SRAM lines. False positive rate is measured as the number of mispredicted positive predictions divided by the total number of predictions. Among the 11 multi-core workloads, the access pattern predictor yields a low false positive rate ranging from 2.1% to 14.8%, with a geometric mean of 8.3%. The access pattern predictor achieves an average coverage of 71.7%. Thus, the majority of dead blocks and write burst blocks can be predicted by the access pattern predictor. The false positive and coverage rates are not illustrated with graphs for lack of space.

5.3.3 Memory Energy Evaluation

Figure 13 shows the memory energy evaluation results normalized to 8MB SRAM LLC. The 24MB STT-RAM LLC reduces the average memory energy to 72.9% of the baseline. The 18MB APM LLC technique reduces average memory energy to 72.4%. Compared with 24MB STT-RAM LLC, the 18MB APM LLC increases average memory traffic by 5.6% due to its smaller capacity. However, it does not increase the dynamic energy consumption because it consumes less activation/precharge energy. The APM LLC achieves a higher DRAM row-buffer hit rate for write requests than the STT-RAM LLC which can reduce

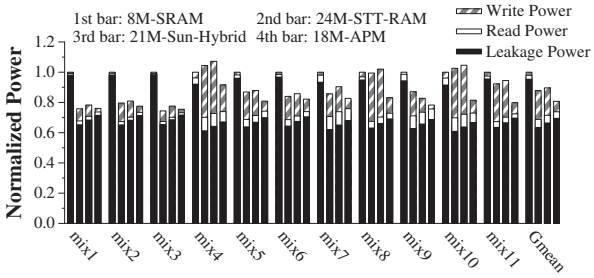

Figure 12. The LLC power breakdown for multi-core applications (normalized to 8MB SRAM)

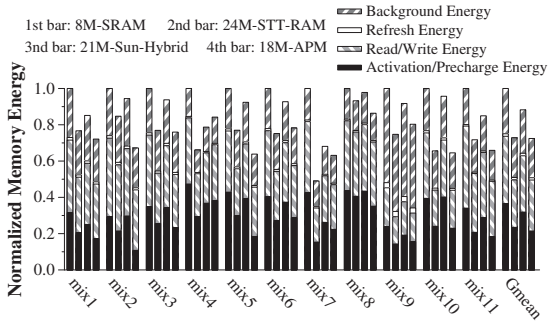

Figure 13. The memory energy breakdown for multi-core applications (normalized to 8MB SRAM)

the activation/precharge energy. The large LLC can filter the locality of dirty blocks, so the dirty blocks have low spatial locality when they are evicted from the LLC and written back to the main memory. However, in the APM LLC, a significant fraction of dirty blocks are written back to the main memory when they are evicted from the SRAM portion where the small capacity of SRAM allows the evicted dirty blocks to have higher spatial locality. Our evaluation result shows the DRAM row-buffer hit rates for writes are 21.1% and 35.6% for 24M STT-RAM LLC and 18M APM LLC respectively.

5.4 Storage Overhead and Power

The technique uses an access pattern predictor to predict the dead blocks and write burst blocks which cause extra storage and power overhead.

5.4.1 Storage Overhead

Each cache block in the LLC adds 1 bit for representing whether it is a dead block, using 9.2K storage total. For the pattern predictor, each prediction table has 4,096 entries with a 2-bit counter in each entry. There are 6 tables for the skewed structure for the dead block prediction table and write burst prediction table using a total of 6KB of storage. In the pattern simulator, one simulated set corresponding 32 LLC sets, each simulated set has 12-entry 16-bit partial read PC, 12-entry 16-bits partial tag, 12-entry

4-bit LRU position, 12-entry 1-bit valid flag, 4-entry 16-bits partial write PC and 4-entry 2-bits write LRU position. For a 4.5MB hybrid cache, it consumes 8.5K storage. Thus, the storage overhead for single-core configuration is 6K+8.5K+9.2K=23.7K which is only 0.53% capacity overhead of the hybrid 4.5MB LLC. For the quad-core configuration, the storage overhead of the APM technique is 6K+8.5K×4+9.2K×4=84.8K, which is 0.43% of the hybrid 18MB cache capacity.

5.4.2 Power Overhead

We evaluate the power overhead of our technique using NVSim [5] and CACTI [8]. For the single-core configuration, the extra dynamic and leakage power consumed by the access pattern predictor is 1.6% and 1.9% of the LLC dynamic and leakage power respectively. It induces a power overhead of 1.8% of the total LLC power consumption. For the quad-core system configuration, the extra dynamic and leakage power consumed by the access pattern predictor is 1.1% and 0.60% of the LLC dynamic and leakage power respectively. The overall power overhead caused by the access pattern predictor is 0.76% of the overall LLC power consumption for quad-core configuration.

6 Related Work

6.1 Related Work on Mitigating Write Overhead of STT-RAM

Many prior papers [12, 27, 17] focus on mitigating write overhead of an STT-RAM cache. Jog *et al.* [12] propose to improve the write speed of STT-RAM-based LLC by relaxing its data retention time. However, that technique requires large capacity buffers for a line level refreshing mechanism to retain reliability. Mao *et al.* [17] propose prioritization policies for reducing the waiting time of critical requests in the STT-RAM-based LLC. However, the technique increases the power consumption of LLC. Recently, researchers propose hybrid SRAM and STT-RAM techniques [25, 3, 9, 2, 26] for improving LLC efficiency. Sun *et al.* [25] take the first step introducing the hybrid cache structure. That technique uses a counter-based approach for predicting write-intensive blocks. Write-intensive blocks are placed in SRAM ways for reducing write overhead to STT-RAM portion. However, that technique is optimized only for core-write operations. It cannot reduce the prefetch-write and demand-write operations to STT-RAM. Jadidi *et al.* [9] propose reducing write variance of STT-RAM lines by migrating frequently written cache blocks to other STT-RAM lines or SRAM lines. However, frequently migrating data between cache lines incurs significant performance and energy overhead. Chen *et al.* [3] propose a combined static and dynamic scheme to optimize the block placement for hybrid cache. The downside of the technique is it requires the compiler to provide static hints for initializing the block placement.

6.2 Related Work on Intelligent Block Placement Policy

Much previous work [6, 10, 20, 11] focuses on improving cache efficiency by using intelligent block placement policy. Hardavellas *et al.* [6] propose a block placement and migration policy for a non-uniform cache. The technique classifies cache access patterns into instructions, private data and shared data. The cache design places data into its proper position based on data type. Our technique is orthogonal to LLC replacement policy. It can be combined with the LLC replacement policy applied to the STT-RAM portion to further improve the cache efficiency.

6.3 Related Work on Dead Block Prediction

Dead block predictors have been proposed [14, 4, 16, 13]. Lai *et al.* [4] propose a trace-based predictor used to prefetch data into dead blocks in the L1 data cache. The technique hashes a sequence of memory-access PCs related to the cache block into a signature used to predict when a block can be safely evicted. Kharbutli and Solihin [14] propose counting-based predictors used for cache replacement policy. Khan *et al.* [13] propose sampling-based dead block prediction for the LLC. None of the previous proposed dead block predictors take into account all data access types. Our proposed pattern predictor predicts dead blocks and write burst blocks in the presence of all types of LLC accesses.

7 Conclusion

Hybrid STT-RAM and SRAM LLC design relies on an intelligent block placement policy to leverage the benefits from the characteristics for both STT-RAM and SRAM technology. In this work, we propose a new block placement and migration policy for a hybrid LLC that places data based on access patterns. LLC writes are categorized into three classes: core-write, prefetch-write, and demand-write. We analyze the access pattern for each class of LLC writes and design a block placement policy that adapt to the access pattern of each class. A low cost access pattern predictor is proposed for guiding the block placement. Experimental results show our technique can improve performance and reduce LLC power consumption compared with both SRAM LLC and STT-RAM LLC with the same area configuration.

8 Acknowledgments

Yuan Xie and Cong Xu were supported in part by NSF 1218867, 1213052, and by DoE under Award Number DE-SC0005026. Guangyu Sun was supported by NSF China 61202072 and National High-tech R&D Program of China (Number 2013AA013201). Daniel A. Jiménez and Zhe Wang were supported in part by NSF CCF grants 1332654, 1332598, and 1332597.

References

[1] Intel core i7 processor. *http://www.intel.com/products /processor/corei7/.*

[2] Y.-T. Chen, J. Cong, H. Huang, B. Liu, C. Liu, M. Potkonjak, and G. Reinman. Dynamically reconfigurable hybrid cache: An energy-efficient last-level cache design. In *DATE'12*, pages 45–50, 2012.

[3] 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 *Proceedings of the 2012 ACM/IEEE international symposium on Low power electronics and design*, ISLPED '12, pages 237–242, New York, NY, USA, 2012. ACM.

[4] A. chow Lai, C. Fide, and B. Falsafi. Dead-block prediction and dead-block correlating prefetchers. In *In Proceedings of the 28th International Symposium on Computer Architecture*, pages 144–154, 2001.

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

[6] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki. Reactive nuca: near-optimal block placement and replication in distributed caches. In *Proceedings of the 36th annual international symposium on Computer architecture*, ISCA '09, pages 184–195, New York, NY, USA, 2009. ACM.

[7] J. L. Henning. Spec cpu2006 benchmark descriptions. *SIGARCH Comput. Archit. News*, 34:1–17, September 2006.

[8] HP Laboratories. CACTI 6.5. *http: //hpl .hp.com: research /cacti/*.

[9] A. Jadidi, M. Arjomand, and H. Sarbazi-Azad. High-endurance and performance-efficient design of hybrid cache architectures through adaptive line replacement. In *Low Power Electronics and Design (ISLPED) 2011 International Symposium on*, pages 79–84, 2011.

[10] A. Jaleel, K. Theobald, S. S. Jr., and J. Emer. High performance cache replacement using re-reference interval prediction. In *Proceedings of the 37th Annual International Symposium on Computer Architecture (ISCA-37)*, June 2010.

[11] D. A. Jiménez. Insertion and promotion for tree-based pseudolru last-level caches. In *MICRO*, pages 284–296, 2013.

[12] A. Jog, A. Mishra, C. Xu, Y. Xie, V. Narayanan, R. Iyer, and C. Das. Cache revive: Architecting volatile stt-ram caches for enhanced performance in cmps. In *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE*, pages 243–252, 2012.

[13] S. M. Khan, Y. Tian, and D. A. Jimenez. Sampling dead block prediction for last-level caches. In *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '43, pages 175–186, Washington, DC, USA, 2010. IEEE Computer Society.

[14] M. Kharbutli and Y. Solihin. Counter-based cache replacement and bypassing algorithms. *IEEE Trans. Comput.*, 57:433–447, April 2008.

[15] C. Kim, J.-J. Kim, S. Mukhopadhyay, and K. Roy. A forward body-biased low-leakage sram cache: device, circuit and architecture considerations. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 13(3):349–357, 2005.

[16] H. Liu, M. Ferdman, J. Huh, and D. Burger. Cache bursts: A new approach for eliminating dead blocks and increasing cache efficiency. In *Proceedings of the 41st annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 41, pages 222–233, Washington, DC, USA, 2008. IEEE Computer Society.

[17] M. Mao, H. H. Li, A. K. Jones, and Y. Chen. Coordinating prefetching and stt-ram based last-level cache management for multicore systems. In *Proceedings of the 23rd ACM international conference on Great lakes symposium on VLSI*, GLSVLSI '13, pages 55–60, New York, NY, USA, 2013. ACM.

[18] P. Michaud, A. Seznec, and R. Uhlig. Trading conflict and capacity aliasing in conditional branch predictors. In *Proceedings of the 24th International Symposium on Computer Architecture*, pages 292–303, June 1997.

[19] A. Patel, F. Afram, S. Chen, and K. Ghose. MARSSx86: A full system simulator for x86 CPUs. In *Proceedings of the 2011 Design Automation Conference*, June 2011.

[20] M. K. Qureshi, A. Jaleel, Y. N. Patt, S. C. Steely, and J. Emer. Adaptive insertion policies for high performance caching. In *Proceedings of the 34th annual international symposium on Computer architecture*, ISCA '07, pages 381–391, New York, NY, USA, 2007. ACM.

[21] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens. Memory access scheduling. In *Proceedings of the 27th annual international symposium on Computer architecture*, ISCA '00, pages 128–138, New York, NY, USA, 2000. ACM.

[22] P. Rosenfeld, E. Cooper-Balis, and B. Jacob. Dramsim2: A cycle accurate memory system simulator. *Computer Architecture Letters*, PP(99):1, 2011.

[23] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. In *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*, October 2002.

[24] S. Srinath, O. Mutlu, H. Kim, and Y. Patt. Feedback directed prefetching: Improving the performance and bandwidth-efficiency of hardware prefetchers. In *High Performance Computer Architecture, 2007. HPCA 2007. IEEE 13th International Symposium on*, pages 63–74, 2007.

[25] G. Sun, X. Dong, Y. Xie, J. Li, and Y. Chen. A novel architecture of the 3d stacked mram l2 cache for cmps. In *HPCA*, pages 239–249, 2009.

[26] 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. In *Proceedings of the 23rd ACM international conference on Great lakes symposium on VLSI*, GLSVLSI '13, pages 19–24, New York, NY, USA, 2013. ACM.

[27] J. Wang, X. Dong, and Y. Xie. Oap: An obstruction-aware cache management policy for stt-ram last-level caches. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, pages 847–852, 2013.

[28] X. Wang, Y. Chen, H. Li, D. Dimitrov, and H. Liu. Spin torque random access memory down to 22 nm technology. *IEEE Transactions on Magnetics*, 44(11):2479–2482, 2008.

[29] Z. Wang, S. M. Khan, and D. A. Jiménez. Improving writeback efficiency with decoupled last-write prediction. In *Proceedings of the 39th International Symposium on Computer Architecture*, ISCA '12, pages 309–320, Piscataway, NJ, USA, 2012. IEEE Press.