Statistical Cache Bypassing for Non-Volatile Memory

Guangyu Sun, *Member, IEEE*, Chao Zhang, *Student Member, IEEE*, Peng Li, *Member, IEEE*, Tao Wang, *Member, IEEE*, and Yiran Chen, *Member, IEEE*

Abstract—With the increasing data throughput requirement, non-volatile memories, such as STT-RAM, PCM and RRAM, have become very competitive designs as on-chip caches in chip-multi-processors (CMPs). Since the write operations are more expensive in an asymmetric-access cache, it is more valuable to justify the data allocation. However, the asymmetric-access property of non-volatile memory is not well addressed in prior bypassing approaches, which are not energy efficient and induce non-trivial operation overhead. In this paper, we propose cache-bypassing methods designed for non-volatile memory. The basic method, SBAC, is based on data locality statistics of the whole cache rather than a signature of each cache line. The multicore extensions, SBAC-C and SBAC-G, strengthen the SBAC by distinguishing data patterns in CMPs. We observe that the decision-making of SBAC and its multicore extensions is highly accurate. Experiments show that SBAC can reduce overall energy consumption by 22.3 percent, and reduce execution time by 8.3 percent on average. The energy consumption is reduced by 21.4 and 23.4 percent for SBAC-C and SBAC-G. And the performance is improved by 7.8 and 9.6 percent for SBAC-C and SBAC-G in multicore scenario. Compared to prior approaches, SBAC outperforms and induces trivial design overhead.

Index Terms—Statistics, bypass, asymmetric-access cache, data reuse count

1 Introduction

Driven by increasing data throughput requirement in chip-multi-processors (CMPs), more and more processing cores are integrated on a single chip. However, since progress of memory technology is slower than that of logic, the off-chip bandwidth can no longer fulfill the increasing requirement of data throughput. Thus, more caches are integrated in a processor to bridge the bandwidth gap. The Intel Haswell processor released in 2013 is an example: It employs embedded DRAM (eDRAM) as L4, in addition to previous three-level SRAM cache hierarchy [10]. Unfortunately, the poor scalability and high leakage consumption of volatile memory (SRAM and eDRAM) is challenging on-chip memory design in future.

To overcome those challenges, various non-volatile memory (NVM) technologies have been extensively studied to replace SRAM and eDRAM as on-chip caches. These emerging memory technologies include spin-transfer torque random access memory (STT-RAM), Phase Change Memory (PCM) and resistive random access memory (RRAM), etc., [11], [29], [41], [42]. Compared to traditional memory technologies, they have advantages of good scalability, low standby power, high storage density, and immunity to particle based soft errors. Prior research has shown that these emerging memories can be employed as L2 and L3 caches to improve performance, reduce power consumption, and even enhance reliability against soft errors [13], [23], [30], [34]. These NVMs demonstrate significant potential to bridge the bandwidth gap in future chip-multi-processors designs.

The cache designs based on these non-volatile memories are normally called *asymmetric-access caches*. It means that the read and write operations to these memories could be based on different mechanism and have different access latencies, energy consumptions, and reliability properties. From architecture perspective, the costs to read or write data in memory are not equal anymore, and the cost of write operation may not be ignored anymore. In most NVM techniques nowadays, the latency and energy consumption of write can be several times larger than those of read. This makes it more difficult to move write latency away from critical path. Low endurance of data programming makes writing unnecessary data costly: it not only degrades the performance, but also reduces the lifetime. Thus, the asymmetry should also be considered in architecture designs.

Most previous studies about asymmetric-access caches focus on how to mitigate the problems caused by access asymmetry between read and write operations. For example, write halt and PreSET techniques are proposed to hide long write latency of these asymmetric-access caches or main memory [26], [31], [33]. The hybrid cache architecture can improve write performance and reduce write energy at the same time by placing frequently updated data in the symmetric-access part (e.g., SRAM) [31], [39]. The replacement policy can also be tailored to reduce write latency or energy [43] by evicting cache lines with less updated bits. Wear-leveling and error correction codes are applied in some levels of asymmetric-access caches to improve their lifetimes and reliability [16], [32].

• *G. Sun, C. Zhang, P. Li, and T. Wang are with the CECA, Peking University, Beijing 100871, China.*
E-mail: {gsun, zhang.chao, wangtao}@pku.edu.cn, penli@cs.ucla.edu.
• *Y. Chen is with the ECE, University of Pittsburgh, PA 15261.*
E-mail: yic52@pitt.eduyic52@pitt.edu.

Manuscript received 11 Mar. 2015; revised 23 Dec. 2015; accepted 9 Jan. 2016. Date of publication 11 Feb. 2016; date of current version 14 Oct. 2016.
Recommended for acceptance by G. Min.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TC.2016.2529621

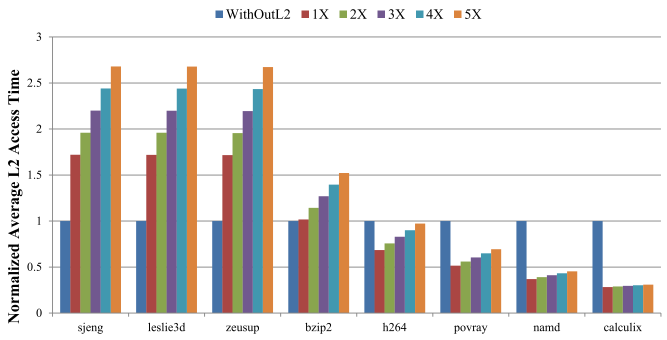


Fig. 1. Comparison of total cache access time with different L2 cache write latency.

These previous studies, however, still follow a fundamental management rule for traditional symmetric cache design: *recently accessed data should be placed in caches closer to processing cores to reduce data access latency*. In fact, such a rule may be easily violated with asymmetric-access caches. This conclusion can be observed from results shown in Fig. 1. In this Figure, some workloads from SPEC2006 are executed on a $i7$-like CMP with a three-level cache architecture illustrated in Fig. 6 (detailed configurations can be found in Table 3). Caches in the cache hierarchy are managed as traditional symmetric-access caches following the rule mentioned above. In other words, when processing cores request data from main memory, cache lines containing requested data are allocated in L3 cache first, then in L2 cache, and finally in L1 cache.

In the first set of results, we remove L2 cache and list total data access time when data are loaded from L3 cache to L1 cache directly. The second set of results is for the baseline case that read and write operations have the same access latency. In the next three sets of results, the write operation cost is increased to 2x, 3x, 5x of read operation, respectively. In order to highlight latency of loading cache lines from L3 cache to L2 cache, we assume that write back data from L1 cache can be completely hidden by a perfect write buffer. It is easy to find that the data access time increases when the cache become more asymmetric. When L2 cache write latency is large enough (5x of its read latency), removing L2 cache may even help improve performance. time can be reduced when L2 cache is removed for some benchmarks. It means that we cannot gain any benefits by moving recently used data from L3 cache to L2 cache.

Previous research has demonstrated that cache bypassing technique is efficient to mitigate cache contamination by allocating data into a cache selectively. There has been extensive research about bypassing techniques for traditional symmetric-access caches [1], [7], [8], [9], [14], [15], [19], [22], [24], [27], [28], [35], [36], [37], [40]. Lots of schemes designed for symmetric-access caches show significant benefit to performance improvement. Previous approaches, however, cannot work efficiently with asymmetric-access cache. Ignoring the high overhead of write operations leads to incorrect bypassing decisions. Moreover, the bypassing decision is cache-line oriented in previous approaches, where access history of every cache line needs to be tracked. It induces non-trivial design and run-time operation overhead. In addition, some bypassing techniques are designed for specific cache configurations (e.g., exclusive LLC only).

Extensive research has been proposed to mitigate write issues of asymmetric-cache. For example, write halt and PreSET techniques are proposed to hide long write latency of these asymmetric-access caches or main memory [26], [31], [33]. The hybrid cache architecture is explored by allocating frequently updated data to the symmetric-access cache (e.g., SRAM) [31], [39]. The replacement policy can also be tailored [43] by evicting cache lines with less updated bits.

Considering the expensive write operation in asymmetric-access caches, we think the data's locality should justify its cache allocation, or the data should be bypassed. In this work, we propose a novel bypassing method based on data reuse statistics. The method is called statistics-based cache bypassing for asymmetric-access caches (SBAC). The basic idea is to *estimate the run-time locality of data based on statistical distribution of data reuse numbers so that proper cache bypassing methods are applied*. Compared to traditional bypassing schemes, SBAC considers the read-write asymmetry to decide whether data should bypass a cache or not.

SBAC makes bypassing decision based on statistics from the entire cache, rather than partial history of specific datablock. Consequently, both design and run-time overhead is significantly reduced. More importantly, the bypassing decision-making can achieve high accuracy because the statistical behavior of data is stable and predictable for many applications (details are discussed in Section 3). The results show that our method induces trivial design overhead and can achieve better performance compared to previous approaches. The contributions of this work are summarized as follows:

- We provide theoretical analysis of cost (latency or energy consumption) for allocating or bypassing data into an asymmetric-access cache.
- Based on the theoretical principle, we propose a cache bypassing method, SBAC. It's easy to apply and induces trivial design overhead.
- Run-time bypassing prediction technique is employed to dynamically adjust bypassing policies.
- We propose core-based and group-based bypassing techniques to enhance SBAC in multi-core systems, where data with different localities are mixed together.
- We discuss the design issues to extend SBAC targeted on performance optimization.
- We further evaluate the proposed techniques by distinguishing loaded and write-back data.

The rest of this paper is organized as follows. The background of non-volatile memory is provided in Section 2. The theoretical analysis of statistics based cache bypassing is introduced in Section 3. The architecture and operation flow of SBAC and its extensions, core-based and group-based bypassing techniques, are introduced in Section 4. The issues to extend SBAC for other scenarios are discussed in Section 5. The experimental results and discussions are presented in Section 6. The related work is analyzed in Section 7, followed by conclusions in the last section.

2 NVM Background

In this section, we provide a brief review of various emerging NVM technologies to understand their asymmetry between read and write operations. These NVMs

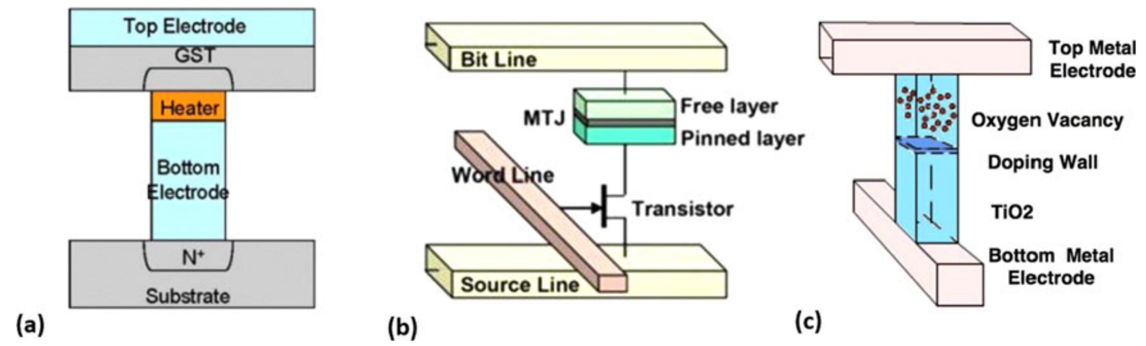


Fig. 2. Illustration of various non-volatile memory cells. (a) PCM, (b) STT-RAM, (c) RRAM.

include STT-RAM, PCM, and RRAM. Their cells are illustrated in Fig. 2.

STT-RAM: The magnetic tunneling junction (MTJ) is used to store data in a STT-RAM cell [11], [20], shown in Fig. 2a. MTJ is composed of a layer of tunneling dielectric (e.g., MgO) sandwiched between two ferromagnetic layers. These two layers are called “reference layer” and “free layer”, respectively. It is because the magnetization of reference layer is fixed and that for free layer can be changed by programming. When the magnetizations of two ferromagnetic layers have the same direction, the MTJ is in low resistance state, which can be used to represent bit ‘0’. On the contrary, the bit ‘1’ is represented with high resistance state of a MTJ. The read operation is processed by sensing the resistance of a MTJ. For a write operation, magnetization of the free layer is changed by the electrical current directly. Compared to a read operation, the programming current for a write is higher and longer.

PCM: For a PCM cell in Fig. 2b, the chalcogenide alloy (e.g., GST) material [2] to store data. Similar to STT-RAM, it also relies on resistance to represent different bits. And the reading operations is also achieved by sensing the resistance. The different is that PCM cell changes its states between the amorphous (high-resistance) and crystalline (low-resistance) phases of the chalcogenide material. Both processes are enabled by heating GST materials using an electrical pulse. In a SET operation, the state changes when the temperature is above its crystallization temperature. In a RESET operation, an electrical current pulse is applied and then cut off shortly to keep it in the amorphous state [4]. Due to the heating process, PCM demonstrates even more significant asymmetry between read and write operations.

RRAM: A typical cell structure of RRAM is shown in Fig. 2c. It is normally referred as those NVM technologies built on the resistance changing mechanisms, other than PCM and STT-RAM. Data is stored by changing the resistance across a dielectric solid-state material in the RRAM cell. The dielectric can be made to conduct through a filament or conduction path formed after application of a sufficiently high voltage. Once the filament is formed, it can be reset for high resistance by another voltage. RRAM can be categorized to unipolar switching [12] and bipolar switching [21]. In unipolar switching, the programming operations are executed by using the pulses with the same polarity but different durations or magnitudes. In bipolar switching, the operations are completed by applying pulses with opposite voltage polarities.

As a summary, read-write asymmetry is common for these emerging NVMs. Both latency and energy of a write operation can be several times higher than those of a read operation. Thus, compared to traditional memory technologies, a write operation is more “expensive”. In other words, it is less efficient to load data in caches made by NVMs. To this end, bypassing technique becomes necessary for those NVMs.

3 Theory Basis

In this section, we first introduce terminologies and definitions used in this paper. Based on them, we theoretically explore the benefit of data bypassing according to its locality.

3.1 Terminologies and Definitions

The terminologies used in this work are consistent with previous literature [8], [19] and are illustrated in Fig. 3. As shown in the figure, data A is brought into a cache line by either a read access or a prefetching operation. The life time of A in the cache is composed of live time (from allocation time to the last use) and dead time (from the last use to its eviction). The total number of accesses (hits) to data after the allocation is called *data reuse count* (DRC). The cache line A in Fig. 3 has a reuse count of five. The first allocation is called *initial placement*. The data having no live time ($DRC = 0$) is normally called *instant dead block*.

With DRCs for massive data, we introduce the definition of DRC probability. Let $N_{DRC=i}$ denotes the number of data that have their DRCs equal to i. Then, a DRC probability P_i is calculated in Equation (1)

$$P_i = \frac{N_{DRC=i}}{\sum_{j=0}^{\infty} N_{DRC=j}}. \tag{1}$$

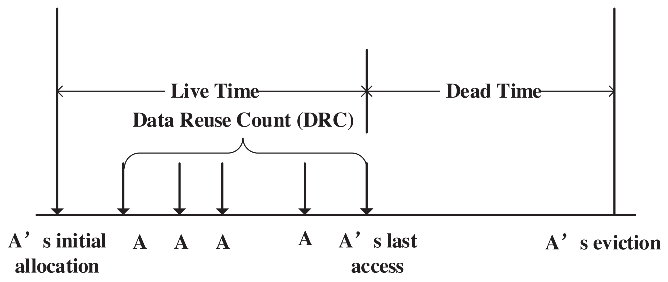


Fig. 3. Illustration of data A and related terms [8], [20].

TABLE 1
Terminologies and Definitions

Term	Definition
DRC	Data Reuse Count
P_i	Probability of $DRC = i$
R_2	Read energy of L2 cache
$R_{2_{tag}}$	Energy of reading L2 tag
$R_{2_{data}}$	Energy of reading L2 data
W_2	Write energy of L2 cache
R_3	Read energy of L3 cache
W_3	Write energy of L3 cache
d	Bypassing depth
λ	Bypassing feature
SI	Sample Interval of DRC

Other definitions and parameters of read and write operations to L2 and L3 caches used in this case study are listed in Table 1.

3.2 Theoretical Energy Saving of Bypassing

We first have a case study on loading data into a cache. Our goal in this case is *to reduce cache access energy*. In order to simplify the discussion, we make some assumptions. First, there are only read operations to the L2 cache. Second, L3 cache is large enough to allocate the working set. Third, the cache is non-inclusive, so the data coherence is still kept even data bypass L2 cache. As shown in Fig. 4, we focus on the case of loading data from L3 to L2. If data loaded from the L3 bypass the L2, they are loaded to L1 directly, as illustrated with path ❷. Otherwise, data will be loaded into L2 normally, shown with path ❶.

We derive the theoretical energy consumption as follows. Initially, the data A exists in the L3 only. When the processing core issues a request to access the data A, it generates cache miss at both L1 and L2 and finally receives a cache hit in the L3. If the data are loaded into the L2 without bypassing, the total access energy to L2 can be calculated in Equation (2)

$$E_{w/o\_bypass} = R_3 + W_2 + (DRC + 1) \times R_2. \tag{2}$$

From left to right, the terms on the right side of Equation (2) represent the energy of reading data from L3, writing data to L2, sending data from L2 to L1 after initial placement, and revisiting data for DRC times. Note that the L2 consumes $R_{2_{tag}}$ to detect the initial miss, and consumes $R_{2_{data}}$ to supply returning data to L1. If data A bypasses the L2, the total cost will be changed to that in Equation (3)

$$E_{bypass} = (DRC + 1) \times (R_{2_{tag}} + R_3). \tag{3}$$

It means that, for each data access, energy is consumed to detect a cache miss in L2 ($R_{2_{tag}}$) and load data from L3 (R_3). Obviously, we can reduce access energy with cache bypassing only when we have $E_{w/o\_bypass} > E_{bypass}$. Thus, we can obtain Equation (4) as the condition to enable cache bypassing. It means that the DRC should be large enough to ensure the benefits of data reuse can amortize the overhead of writing data into the L2

$$\begin{aligned} DRC &< \frac{W_2 + R_2 - R_{2_{tag}}}{R_3 + R_{2_{tag}} - R_2} \\ &= \frac{W_2 + R_{2_{data}}}{R_3 - R_{2_{data}}}. \end{aligned} \tag{4}$$

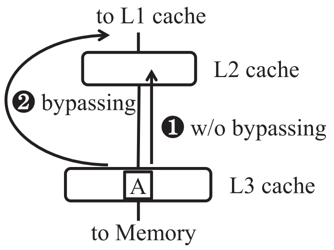


Fig. 4. SBAC for loading data.

Bypass or not? It becomes a new problem for asymmetric-access cache. Cache can benefit from the data allocation for symmetric-access cache, whenever there is at least once reuse of the data in L2. This is because W_2 is similar to R_2 and several times smaller than R_3, for SRAM/eDRAM caches. For the asymmetric-access cache, however, the W_2 is no longer much smaller than R_3. Thus, a higher DRC is expected to justify the data allocation. In order to achieve lowest access energy, data with DRC less than $\lceil \frac{W_2 + R_{2_{data}}}{R_3 - R_{2_{data}}} \rceil$ should bypass L2.

In order to demonstrate the impact of read-write asymmetry, we compare the DRC requirement of cache bypassing for SRAM and STT-RAM caches. Table 2 shows typical access energy consumption of caches based on SRAM or STT-RAM. To symmetric caches, loading data into L2 is more energy-efficient when DRC is higher than one. While in asymmetric caches, only very frequently accessed data with DRC higher than six should be loaded into L2.

3.3 Theory Basis of SBAC

In run-time execution, it is impractical to know exactly all the DRC values of all cache data before they come into cache. Thus, making bypass decisions based on future DRC of data, as shown by the simple method aforementioned, is not realistic. However, it is possible to filter out the data with specific DRC with a simple bypassing method. For example, we can assume the average DRC for unfiltered data is smaller than one, and make all initial placements bypass the L2 cache, so only the data with at least one reuse count can enter L2 cache. We call this simple SBAC as "initial placement" bypassing. Thus, the key is to ensure the benefits from dead blocks bypassing can amortize the bypassing of high DRC data.

The theoretical condition of employing the bypassing can be derived based on the probabilities of DRCs. And the probability distribution of DRC in L2 can be represented by statistics $\{P_0, P_1, P_2, \ldots\}$ ($\sum_{i=0}^{\infty} P_i = 1$). Without bypassing technique, the average access energy of these data is noted

TABLE 2
Typical Energy Consumption for 2 MB L2, and 8 MB L3 (Technology Node: 45 nm)

Cache Type	SRAM L2 STT-RAM L3	STT-RAM L2 STT-RAM L3
$R_{2_{data}}(nJ)$	0.066	0.127
$W_2(nJ)$	0.051	0.603
$R_3(nJ)$	0.246	0.246
$\lceil \frac{W_2 + R_{2_{data}}}{R_3 - R_{2_{data}}} \rceil$	1	6

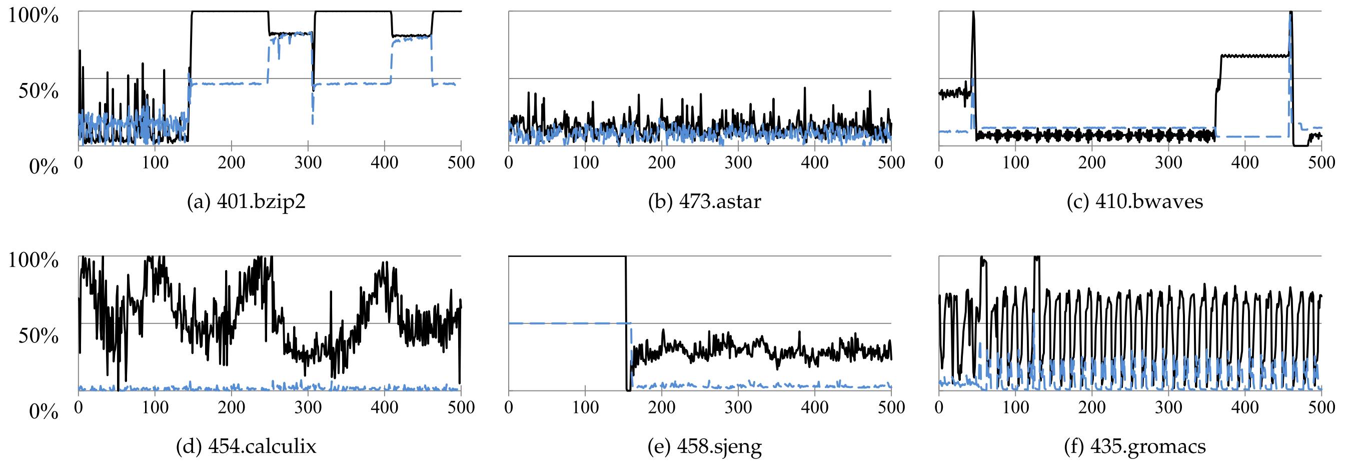


Fig. 5. Several typical distributions of P_0s (solid lines) and corresponding miss rates (dashed lines). X-axis represents cache accesses in the unit of $10K$.

as $\bar{E}_{w/o\_bypass}$. If initial placements of whole data bypass the L2 cache, the cache access energy consumption is noted as $\bar{E}_{bypass}$. The calculation of these two value is shown in Equations (5) and (6)

$$\begin{aligned}\bar{E}_{w/o\_bypass} &= \sum_{i=0}^{\infty}\{P_i \times [R_3 + W_2 + (i+1) \times R_2]\} \\ &= P_0 \times (R_3 + W_2 + R_2) \\ &\quad + P_1 \times (R_3 + W_2 + 2 \times R_2) + \cdots \\ &\quad + P_n \times [R_3 + W_2 + (n+1) \times R_2] + \cdots\end{aligned} \tag{5}$$

$$\begin{aligned}\bar{E}_{bypass} &= P_0 \times (R_3 + R_{2_{tag}}) \\ &\quad + \sum_{i=1}^{\infty}\left\{P_i \times \left[2 \times R_3 + R_{2_{tag}} + W_2 + i \times R_2\right]\right\} \\ &= P_0 \times (R_3 + R_{2_{tag}}) \\ &\quad + P_1 \times (2 \times R_3 + R_{2_{tag}} + W_2 + 1 \times R_2) + \cdots \\ &\quad + P_n \times \left[2 \times R_3 + R_{2_{tag}} + W_2 + n \times R_2\right] \cdots\end{aligned} \tag{6}$$

With these two equations, it is easy to understand that such an "initial placements" bypassing can only reduce average access energy when $E_{bypass} < E_{w/o\_bypass}$. After comparing Equations (5) and (6), we obtain the condition to trigger an "initial placement" bypassing, described as an Equation (7)

$$P_0 > \frac{R_3 + R_{2_{tag}} - R_2}{W_2 + R_3}. \tag{7}$$

Obviously, whether the bypass benefits can amortize the overhead comes related with the value of one statistic, P_0, which is the DRC probability of instant dead blocks. This is the reason why we call our technique as a statistics based cache bypassing method (SBAC).

In order to have a quantitative comparison, we calculate the bypass condition for symmetric- and asymmetric-access caches, respectively. Cache bypassing can gain benefits when $P_0 > 60.6$ percent for a symmetric-access cache. To an asymmetric-access cache, however, bypassing condition is satisfied with a significant lower value ($P_0 > 14.0$ percent). We follow the same parameters in Table 2.

It is easy to understand that the efficiency of "initial placement" bypassing highly depends on the P_0 of real applications. For example, P_0 should be larger than $\frac{R_3+R_{2_{tag}}-R_2}{W_2+R_3}$ to achieve substantial reduction of energy consumption using bypassing. To demonstrate feasibility of SBAC, we carefully study DRC distributions of benchmarks from SPEC2006. As shown in Fig. 5, we list several typical patterns of P_0. For the first two patterns (Figs. 5a bzip2 and 5b astar), P_0 is stable for most time so that it is easy to select a bypassing decision. For example, in the first workload, we can always let data bypass the initial placement to reduce energy consumption. For the third and fourth workloads (Figs. 5c bwaves and 5d calculix), the distribution of P_0 varies regularly. Thus, "initial placement" bypassing can also work well if we can dynamically adapt bypassing decisions according to the DRC distribution. For the fifth workload (Fig. 5e sjeng), the P_0 is close to the number $\frac{R_3+R_{2_{tag}}-R_2}{W_2+R_3}$, so that the benefit is not significant after bypassing "initial placement". For the last workload (Fig. 5f gromacs), the DRC distribution has serious transient vibration. Thus, it is difficult to select a proper bypassing decision, and "initial placement" bypassing may not work efficiently. Fortunately, DRC distribution is stable and predictable for most workloads. SBAC works very well for most workloads. More results can be found in Section 6. Note that we also include miss rates in these charts (blue dash lines). We can find the miss rate is not always consistent with the P_0, and it's less efficient to trigger bypass. Thus, miss rates are not used as trigger in this work.

3.4 Bypassing Depth

After the "initial placement" bypassing is applied, the original data with once reuse count becomes instant dead block since their first loads are filtered. Thus, it is reasonable to make these new instant dead blocks bypass L2 to further reduce access energy consumption. In other words, bypass the data with $DRC < 2$. Thus, we introduce the definition of *bypassing depth*, which means that data with DRC less than bypassing depth should bypass the cache. For example, when the bypassing depth is set to "1", only initial placements are bypassed. The theoretical calculation of bypassing depth is discussed as follows.

Similar to the derivation of "initial placement" bypassing decision, we can calculate the bypassing condition with "bypassing depth = 2" as in Equation (8)

$$\frac{P_1}{1-P_0} > \frac{R_3 + R_{2_{tag}} - R_2}{W_2 + R_3}. \tag{8}$$

And we can further calculate condition for any bypassing depth d as in the following equation:

$$\frac{P_{d-1}}{1-\sum_{j=0}^{d-2} P_j} > \frac{R_3 + R_{2_{tag}} - R_2}{W_2 + R_3}. \tag{9}$$

In this work, the $\lambda = \frac{R_3+R_{2_{tag}}-R_2}{W_2+R_3}$ is called *bypassing feature* of the system, which is the intrinsic cache attribute. The high write energy of asymmetric-access caches results in a small bypassing feature, making bypass more attractive to reduce energy consumption.

3.5 Bypassing with Both Read and Write Operations

As mentioned before, we have only considered the read operations in making decisions of bypassing. It's obvious not wise to ignore the write ones. However, the theory becomes more complicated, when both read and write operations are considered.

We still follow the terminologies and definitions introduced previously, while adding two new ones, P_r and P_w, to distinguish read and write. The P_r and P_w represent the average probability for read and write operation to L2 cache. We define the write operation here as the operation used to write data from upper level of cache. Thus, it does not count the data refill due to cache miss. In order to calculation the P_r and P_w, we represent the probability by statistics again. P_r can be calculated by the ratio of read operations compared with overall operations. P_w is calculated by the write ones. The numbers of write and read operation are also from Table 1.

With similar deduction flow (Equation (5) to (9)), we find that the condition of enabling cache bypassing can still be achieved by Equation (10). The only difference is the calculation of the bypassing feature λ

$$\frac{P_{d-1}}{1-\sum_{i=0}^{d-2} P_i} > \lambda. \tag{10}$$

The bypassing feature for L2 is calculated in Equation (11). It involves read and write energy in L2 and L3, and the probability of read and write operations

$$\lambda = \frac{P_r \times (R_{2_{tag}} + R_3 - R_2) + P_w \times (W_3 - W_2)}{P_r \times (W_2 + R_3) + P_w \times (W_3 + R_3)}. \tag{11}$$

The λ is actually affected by both P_r and P_w. After comparing the new bypassing feature with the one only considering read in Equation (9), we can find the part conrolled by P_r in Equation (11) is roughly the Equation (9). And the difference between these two equations is the P_w part. And when the P_r is 0, the bypassing feature is only determined by the write cost of L2 and L3 and read cost of L3. This means if we only consider the write operation, the bypassing feature value we get will be much smaller than what we got for read, due to the relatively large write energy of asymmetric cache.

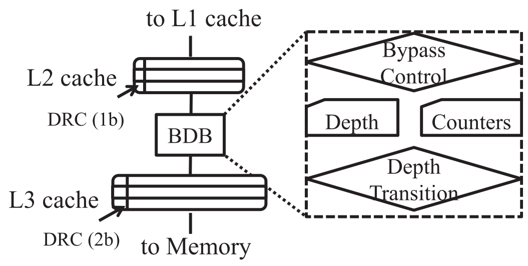


Fig. 6. Architecture for cache bypassing.

3.6 Other Practical Issues

In real cases, there are several factors that may affect the theoretical bypassing condition.

First, after some data have bypassed the cache, the probability distribution of DRC may change a little because cache bypassing can result in different cache replacements. Fortunately, we found that the distribution of DRC almost keeps the same, since it is calculated statistically based on a great quantity of data.

Second, the simplification that all requested data are cached in L3 cache can be inaccurate. Thus, the benefits of bypassing may be decreased and a higher probability is required in these equations to trigger a bypassing.

Third, data prefetching need to be treated separately from normal data access because data prefetched can only be avoided using "initial placement" bypassing.

Considering these factors, the real bypassing conditions for different workloads may have deviation from the theoretical λ introduced in Equation (9). Fortunately, as we have mentioned, the distribution of DRC is normally quite biased. It means that theoretical λ normally works well. This is supported by results in Section 6.

4 Design of SBAC

4.1 Overview

We still use the case of loading data from L3 cache to L2 cache to describe the architecture design of SBAC. As a pivot to select proper bypassing decisions, extra components are needed to monitor and predict the distribution of DRC for data in the L2 cache. As shown in Fig. 6, one extra bit is added to each cache line in L2, and two bits are added to each cache line of L3 to count the DRC of the line. In addition, they are also used to decide whether cache bypassing is needed. The extra function between L2 and L3 is called bypassing decision block *(BDB)*. BDB monitors cache lines transmitted on the data bus. It can track information of the DRC sent with data so that probability distribution of DRC is calculated.

As the major component to count the DRC distribution, a BDB includes three global DRC counters. Three DRC counters are denoted as $N_{\geq d-1}$, $N_{\geq d}$, and $N_{\geq d+1}$. They are used to count the number of DRC greater than $d-1$, d, and $d+1$, respectively. With these DRC counters, we can rewrite conditions in Equation (9) with Equation (12). The opposite condition is calculated in Equation (13). Without calculating the precise distribution value, we get effective DRC distribution to make bypassing decisions in Equations (12), (13). Read/Write energy numbers are used to calculate the λ.

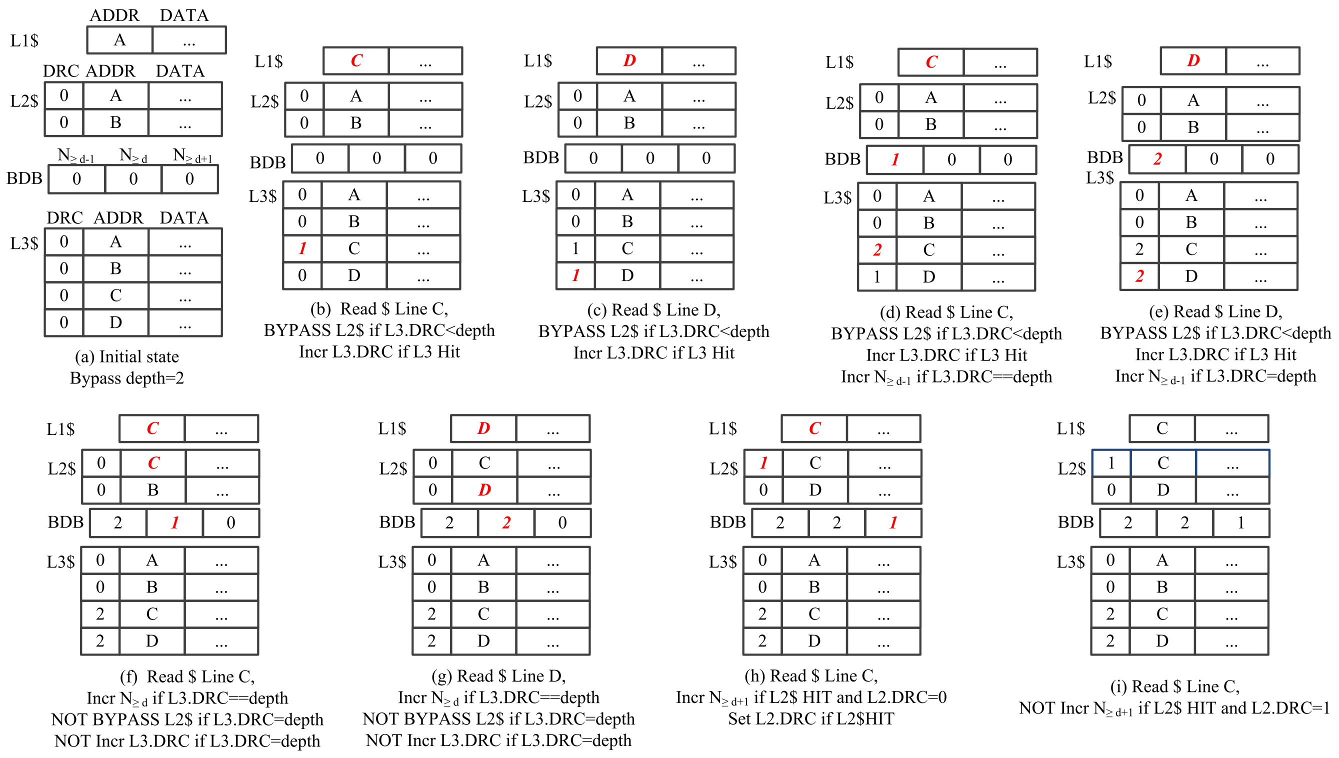

(a) Initial state
Bypass depth=2
(b) Read $ Line C, BYPASS L2$ if L3.DRC<depth Incr L3.DRC if L3 Hit
(c) Read $ Line D, BYPASS L2$ if L3.DRC<depth Incr L3.DRC if L3 Hit
(d) Read $ Line C, BYPASS L2$ if L3.DRC<depth Incr L3.DRC if L3 Hit Incr N≥d-1 if L3.DRC==depth
(e) Read $ Line D, BYPASS L2$ if L3.DRC<depth Incr L3.DRC if L3 Hit Incr N≥d-1 if L3.DRC=depth
(f) Read $ Line C, Incr N≥d if L3.DRC==depth NOT BYPASS L2$ if L3.DRC=depth NOT Incr L3.DRC if L3.DRC=depth
(g) Read $ Line D, Incr N≥d if L3.DRC==depth NOT BYPASS L2$ if L3.DRC=depth NOT Incr L3.DRC if L3.DRC=depth
(h) Read $ Line C, Incr N≥d+1 if L2$ HIT and L2.DRC=0 Set L2.DRC if L2$HIT
(i) Read $ Line C, NOT Incr N≥d+1 if L2$ HIT and L2.DRC=1

Fig. 7. An example of cache bypassing flow.

The bypassing control logic can make decision for data transferring on the bus. A cache block will bypass L2 cache if the DRC bit in the L3 cache is smaller than the bypass depth. The bypass depth transition logic is employed to calculate runtime bypassing depth. The bypass depth will be increased by one when Equation (12) is satisfied, and decreased by one when Equation (13) is satisfied

$$\frac{N_{\geq d} - N_{\geq d+1}}{N_{\geq d}} > \lambda, \tag{12}$$

$$\frac{N_{\geq d-1} - N_{\geq d}}{N_{\geq d-1}} < \lambda. \tag{13}$$

4.2 Operation Flow with Cache Bypassing

Having the SBAC architecture, we describe the flow for different cache operations with an example in Fig. 7. As shown in the figure, L1, L2, and L3 caches are illustrated with one, two, and four cache lines. The three DRC counters of BDB are also shown in the figure. There is one DRC bit for each cache line in L2 and two bits for each cache line in L3. The bypassing depth d is set to 2 in this example. The detailed operation flow is described as follows.

- Step (a): In the initial state, all three DRC counters are initialized as zero. The DRC bits of each line are also cleared as zero. We assume there are some initial data stored in cache lines.
- Step (b): L1 cache requests data C, since the DRC bit of data C in L3 cache is equal to zero, data C is bypassed to L1 cache directly because $DRC = 0 < d = 2$. At the same time, the DRC bit of data C in L3 cache is increased by one.
- Step (c): Similarly, when L1 cache requests data D, it is also moved from L3 to L1 directly for the same reason.
- Step (d): When L1 cache requests data C again, data C is bypassed again because we still have $DRC = 1 < d = 2$. Then, DRC of data C in L3 cache is increased to 2. At the same time, the first counter in BDB is increased by one because it counts the number of data with $DRC \geq d - 1 = 1$.
- Step (e): Similarly, when L1 cache requests data D again, it is bypassed again. And the first DRC counter of BDB is increased by one.
- Step (f): When L1 cache requests data C for the third time, data C is finally loaded to L2 cache because we have $DRC = d$ now. At the same time, the second counter in BDB is increased by one. Note that the DRC bits of data C in L3 cache are saturated now. They are only reset to zero when data C are evicted from L3 cache.
- Step (g): Similarly, data D is also loaded to L2 cache for the third request, and the second counter in BDB is increased by one.
- Step (h): When data C is first hit in L2 cache, the third counter in BDB is increased by one because C is requested for $d + 1 = 3$ times in total. At the same time, its DRC bit in L2 is set to one.
- Step (i): When data C gets hit again with DRC bit equal to one, the third counter in BDB remains the same.

4.3 Sensitivity Control

Since the probability distribution of DRC varies during runtime execution, the bypassing depth should also be updated periodically to reflect the distribution. The length of each period, in terms of cache accesses, is called *sampling* *interval (SI)* in this work. At the end of a sampling interval, the BDB counters are used to calculate the current probability distribution of DRC. The distribution is used to predict the bypassing depth for the next interval.

The choice of sampling interval has an impact on the prediction accuracy of bypassing depth. Since the bypassing depth is based on DRC, we use the amount of cache accesses to determine a SI. If the SI is too short, the poor sampled statistics cannot represent the probability distribution of DRC. On the other hand, if the SI is too long, it may not capture the changes of DRC distribution so that the efficiency of SBAC is degraded. In addition, the size of counters in BDB is also related to SI.

Experimental results show that the optimal SI varies in the range of $10k \sim 100k$ for different workloads. Thus, we propose an algorithm to dynamically adjust SI for different data patterns. The algorithm is described as follows:

- SI is initialized as the lower bound 2^{14}.
- After each SI, if the bypassing depth is not changed, SI is increased by $2\times$.
- After each SI, if the bypassing depth is changed, SI is decreased by $2\times$.
- The higher bound of SI is set to 2^{20}. Thus, a 20-bit counter is needed.

4.4 SBAC for Multi-Programmed Workload

The case becomes more complicated if multiple workloads are running concurrently. For example, when some programs are initializing with streaming data and the other are iterating inside some loops, two different data patterns may occupy different regions of the cache. Consequently, data in different regions of the cache can demonstrate various probability distributions of DRC. If these regions can be separated with their own bypassing decision, SBAC can work efficiently. Otherwise, when DRC for all cache lines are accumulated together, the efficiency of SBAC is decreased.

For the case that one program is bounded to a specific core, a possible solution is to calculate DRC distribution of each core separately. If the core ID is integrated in the cache tag, it can be leveraged to identify data from each program. Thus, bypassing decisions may be different for data requested by different cores. Such an extension of SBAC is called "core-based SBAC". Note that extra design overhead is induced because the number of counters increase proportionally with the number of cores. Unfortunately, for the processors not supporting core ID in caches, core-based SBAC cannot work. Consequently, we further propose the "group-based SBAC".

The basic idea is to cluster cache lines with similar DRC distributions into a group so that each group can select their own bypassing depth. Thus, cache lines in the same group can work efficiently with SBAC. However, we find that cache lines with similar DRC distributions may not locate close to each other physically. A naive partitioning policy cannot work efficiently. In order to achieve a proper group partitioning without introducing much design overhead, we propose "group-based SBAC" as follows.

First of all, a proper group number should be selected based on two rules: (1) group number should be less than the total number of bypassing depth to improve grouping efficiency; (2) group number should be the power of two to facilitate circuit design. Since the bypassing depth is limited in the range of $0 \sim 3$ in this work, possible group number are 2 and 4. To avoid inducing much design overhead, we set the group number to 2. Thus, an extra "group bit" is added to each cache line, and the two groups are denoted with g_0 and g_1. In addition, to count DRC distribution in these two groups, an extra set of counters is needed in BDB.

Having two groups, a clustering algorithm is needed to select cache lines in each group. The algorithm is described as follows.

- **Step-1** At the beginning, one half of cache lines are selected in g_0, and the other half is in g_1.
- **Step-2** By the end of the first sampling interval, each group has selected its own bypassing depth based on their own BDB counters. Then, k cache lines are selected in g_0 and migrated into g_1.
- **Step-3** By the end of the second SI, the k cache lines migrated can only be kept in g_1 if $\frac{N_{\geq d}-N_{\geq d+1}}{N_{\geq d}}$ increases in g_1. Otherwise, they are migrated back to g_0. Then, we randomly select k cache lines in g_1 and migrate them to g_0.
- **Step-4** By the end of the third sampling interval, cache lines migrated in Step-3 are only kept if $\frac{N_{\geq d-1}-N_{\geq d}}{N_{\geq d-1}}$ decreases in g_0. At the same time, migration in Step-2 is repeated.
- Step-3 is repeated by the end of any following even rounds of SI. Step-4 is repeated by the end of any following odd rounds of SI.

On one hand, this algorithm tries to find cache lines with high DRC in group g_0 and migrate them to group g_1 in the even rounds. On the other hand, the cache lines with lower DRC are moved from group g_1 to group g_0 in the odd rounds. Since both the $\frac{N_{\geq d}-N_{\geq d+1}}{N_{\geq d}}$ and $\frac{N_{\geq d-1}-N_{\geq d}}{N_{\geq d-1}}$ of each group are compared by the end of an SI, extra storage is required in BDB. Moreover, a randomizer based on circuit noise is needed to select cache lines for migration [17].

The number k has an impact on performance of clustering algorithm. If k is too large, cache lines with different DRC distributions may be migrated together so that the chance of successful migration is low. Thus, a small k is preferred although the convergence rate of clustering algorithm is decreased. In this work, we set k to 16.

Although there are many more efficient clustering algorithms, the problem is that they are difficult to be implemented in hardware. The advantage of our group-based SBAC implementation is simple to be designed with low overhead. Besides the DRC counters and group bits, we only need k registers to remember the selected lines for switching.

One drawback is that our clustering may converge very slowly or may not even converge because cache lines are randomly selected after each SI. In the worst case, two groups have the same bypassing depth. It means that group-based SBAC will work same as the original SBAC design.

5 Extending SBAC for Other Scenarios

We only discuss a simple case study in last two sections. As we have addressed, SBAC is a method that can be applied

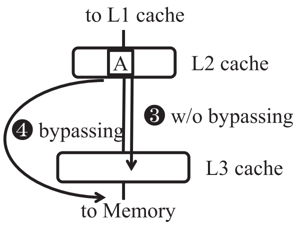


Fig. 8. SBAC for write-back data.

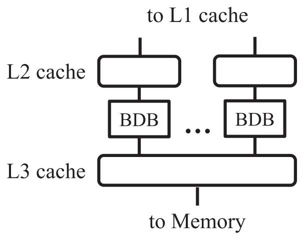


Fig. 9. SBAC for private L2 caches.

to various configurations and for different optimization goals. In this section, we discuss how to extend SBAC for other scenarios.

5.1 Cache Bypassing for Performance Optimization

It is also feasible to apply SBAC to optimize cache performance. The basic idea is quite similar to that for energy consumption optimization. For the theory derivation, we need to replace energy numbers of cache in Equations (1)-(9) with proper access latency numbers.

Different from energy consumption, the latency of a write operation cannot be directly added to the total execution time. For example, the data loaded from L3 cache can be forwarded to L1 cache at the same time when they are written into L2 cache. Thus, it is not accurate to calculate λ using write latency directly. Instead, we need to estimate the time that L2 cache is blocked due to loading data from L3. The blocking time is related to cache access intensity. When the access intensity is low, the blocking time may be well hidden. Previous research [31], [39], [43] pointed out that the blocking time varies from $0\times$ to $0.6 \times \mathit{write\ latency}$.

One solution to this problem is to calculate average runtime blocking time by monitoring the waiting time of read operations in the miss status holding registers (MSHRs). In fact, we find a simple but efficient alternative by fixing blocking time to $0.5 \times \mathit{write\ latency}$. The reason is in two folds: (1) the biased DRC distribution can help make SBAC tolerant to inaccurate blocking time; (2) for the applications that have a lower blocking time, the access intensity is low so that the effect of incorrect bypassing decision is not significant. This observation is proved by the experimental results in Section 6.

5.2 Bypassing for Write-Back Data

Cache bypassing for write-back data is different from that for load data. We use write-back from the L2 cache to the L3 cache as an example, as illustrated in Fig. 8. When a cache block in L2 is written back, it is evicted from L2 cache no matter whether it will bypass L3 cache or not. Thus, bypassing depth can only be set to either zero or one. Cache lines will bypass L3 cache when bypassing depth $d = 1$ and will not bypass L3 when $d = 0$. The method of increasing d from 0 to 1 is similar to that used in the case for data loading. We need to trace the reuse count of this cache line in L3 before it is evicted. Note that the cache line can be reused several times (e.g., there are several private L2 caches). The calculation of bypassing feature λ is similar to the process in Section 3 with read and write latency/enery of L3 cache and the read latency/enery of main memory. If P_0 of write back data is larger than λ after a sampling interval, cache bypassing is triggered. Then, write-back cache lines from L2 will be sent to main memory directly.

Apparently, we cannot use similar method for loading data to stop bypassing of write-back data because we cannot trace the reuse count of those cache lines in main memory. Instead, we use a simple restoration mechanism to overcome this problem. It works as follows.

- We define a new variable called restoration interval (RI). Restoration interval is initialized as same as SI.
- When the cache bypassing is triggered, it will be automatically turned off after it reaches the end of restoration interval.
- After the next sampling interval, if bypassing is triggered again, the restoration interval is increased by one SI. Otherwise, if restoration interval is larger than SI, it is decreased by one SI.

5.3 Private Caches & Data Inclusion

SBAC can be applied to both shared and private caches. For example, if L2 cache is private for each core and L3 cache is shared, we need a BDB between each L2 and L3, which is illustrated in Fig. 9.

For the case study in the last section, the data inclusion is relaxed by using a non-inclusive cache hierarchy. Our bypassing scheme, also works for exclusive caches. For example, we can use the same SBAC architecture in Fig. 6 for exclusive L2 and L3 exclusive caches. The operation flow is quite similar to that of non-inclusive one. Because we only trace data reuse count of a cache line in L3 cache before it is loaded into L2 cache, and the count is reset to zero when it is written back to L3 cache. Thus, whether the cache line is kept in L3 after it is loaded to L2 cache do not affect the bypassing decision.

6 Experimental Evaluation

In this section, we provide comprehensive evaluation to demonstrate the efficiency of SBAC for single and multiple applications under both shared and private L2 configurations.

6.1 Experiment Setup

We implement SBAC in a popular full-system simulator *gem5* [3]. It is configured to model a four-core Haswell like CMP. Each core is running at 2 GHz frequency. There are three levels of caches. The IL1/DL1 caches are SRAM based and the L2 and L3 are configured as asymmetric-access STT-RAM caches. Other details can be found in Table 3. We use cache latency and energy parameters from NVSim [6].

Both single and multiple applications workloads are evaluated. In order to provide a comprehensive evaluation

TABLE 3
Detailed Simulation Setup

Component	Configuration
Processor	4 cores, 2 GHz, 1-way issue
IL1/DL1	32/32 KB, 2-way, 64 B, private, LRU
SRAM	L.P.:47.7 mW, R/W Lat.: 2/2cycle, E.:6.2/2.3 pJ
L2	4 × 256 KB, 8-way, 64 B, LRU, L.P.:428 mW
STT-RAM	R/W Lat.: 6/36 cycle, E: 0.135/0.603 nJ
L3	8 MB, 16-way, 64 B, share, LRU, L.P.:1851 mW
STT-RAM	R/W Lat.: 25/60 cycle, E: 0.246/0.698 nJ
Memory	8 GB, DDR3, 1600 MHz, 120cycle, 12.8 GB/s.

with diversified distributions of DRC, we examine different code segments in both single and randomly mixed multi-programmed benchmarks. Both private and shared L2 configurations are used for experiments of mutli-programmed workloads. For the single application case, only the private cache with one core running is evaluated. The simulator captures all data operations such as loads, stores, and prefetching requests. The one block lookahead (OBL) approach is employed for prefetching in evaluation. All benchmarks come from SPEC CPU 2006. We fast forward one billion instructions at beginning, and execute ten billion instructions of a single benchmark. Then we construct the multi-program workloads by mixing the fast forwarded single programs. The formation of these multi-programs are shown in Table 4. Energy consumption includes leakage and dynamic power of entire cache hierarchy, based on operation statistics.

The labels used in the rest of this section are explained: (1) Baseline: baseline case without cache bypassing; (2) SBAC: case using SBAC; (3) SBAC-C: case using core-based SBAC; (4) SBAC-G: case using group-based SBAC; (5) Shared: case with shared L2 configuration; (6) Private: case with private L2 cache. We use the system overall execution time as performance metric, and overall cache system energy consumption as energy metric. We normalize the results to compare with each others.

6.2 DRC Prediction Accuracy

Since the access history of cache is used to decide the bypass depth in following periods, the performance of SBAC will rely on the prediction accuracy. We take the

TABLE 4
Multi-Programmed Workloads

Name	Benchmarks
mix1	401.bzip2, 410.bwaves, 429.mcf, 434.zeusmp
mix2	401.bzip2, 444.namd, 458.sjeng, 470.lbm
mix3	410.bwaves, 450.soplex, 459.GemsFDTD, 471.omnetpp
mix4	429.mcf, 453.povray, 462.libquantum, 473.astar
mix5	434.zeusmp, 454.calculix, 464.h264ref, 481.wrf
mix6	435.gromacs, 444.namd, 450.soplex, 453.povray
mix7	435.gromacs, 456.hmmer, 465.tonto, 401.bzip2
mix8	454.calculix, 456.hmmer, 458.sjeng, 459.GemsFDTD
mix9	462.libquantum, 464.h264ref, 465.tonto, 470.lbm
mix10	471.omnetpp, 473.astar, 481.wrf, 401.bzip2
mix11	429.mcf, 434.zeusmp, 470.lbm, 410.bwaves
mix12	450.soplex, 454.calculix, 456.hmmer, 458.sjeng
mix13	429.mcf, 470.lbm, 450.soplex, 454.calculix

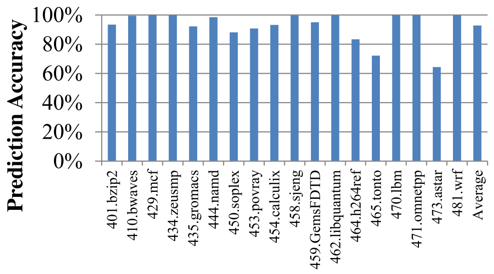


Fig. 10. Prediction accuracy for various single-programmed benchmarks.

relative match ratio of the run-time made decisions and the optimal decision after static profiling, as the prediction accuracy. As shown in Fig. 10, a high prediction accuracy of 92 percent on average is achieved for single program benchmarks. The prediction accuracy is about 86 percent on average for multiprogrammed applications (not shown due to page limit). Note that a correct bypassing decision may be obtained even with a mis-prediction, as long as the bypassing depth is not affected. On the other hand, a correct prediction of DRC distribution may also lead to an incorrect decision of cache bypassing due to inaccurate estimation of read/write energy.

6.3 Evaluation for Single Applications

The results of energy consumption are compared in Fig. 11. We can find that the reduction of energy is related to the prediction accuracy generally. With high prediction accuracy, SBAC has more potential to bypass useless data predicted to have low DRC. Incorrect prediction may cause SBAC to bypass useful data which offsets the benefit of its right doing. For some benchmarks, however, the energy reduction is insignificant even with high prediction accuracy (e.g., *GemsFDTD*). The reason is that for some benchmarks the cache bypassing is not triggered for most of execution time. On average the reduction of the total cache energy consumption is about 22.3 percent.

We also evaluate the performance after applying SBAC targeted for energy, as a side-effect of reducing energy consumption. As shown in Fig. 12, the results of performance improvement are similar to that of energy reduction, but less significant. On average, the total execution time is reduced by 8.3 percent. The result shows SBAC also improves the performance, which agrees with the fact that asymmetric-access caches shows larger write overhead both on energy consumption and time. The reason for the less significant improvement is two-fold: The energy consumption of each load operation is

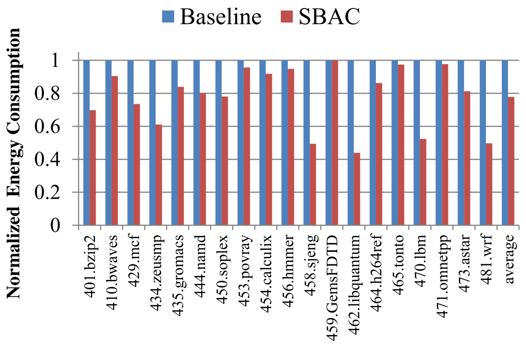


Fig. 11. Normalized energy consumption for single applications.

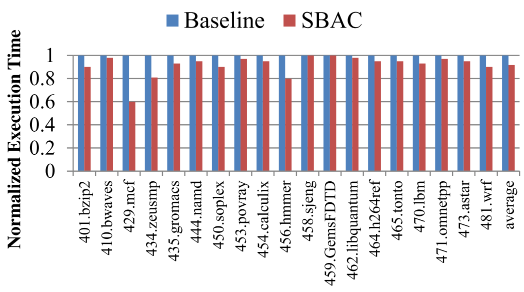


Fig. 12. Normalized execution time for single applications.

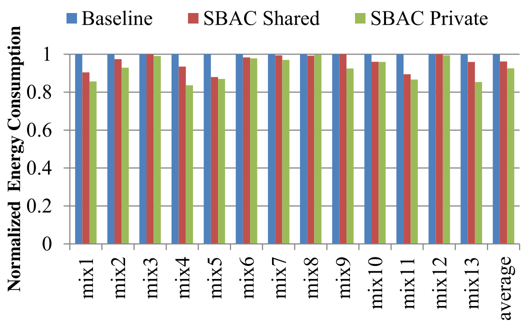


Fig. 13. Normalized energy consumption after using SBAC for two cache configurations.

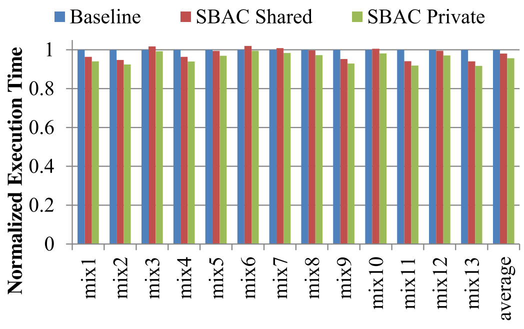


Fig. 14. Normalized execution time after using SBAC for two cache configurations.

reflected in total energy, but the loading time could be hidden by MSHR. The bypassing decisions calculated for optimized energy consumption may not lead to optimized ones in aspect of time. To improve the performance, we can recalculate the bypassing feature by cache time numbers. But this may also hurt the energy performance.

6.4 Evaluation for Multi-programmed Applications

We evaluate energy consumption after applying SBAC to two cache configurations against their corresponding baselines without cache bypassing. We show the normalized comparison in Fig. 13. The results demonstrate that, for private cache configuration, the energy consumption can be reduced after using SBAC. It is because each workload is bounded to a dedicated core and the DRC distribution is estimated separately. On average, SBAC can reduce energy consumption by 7.5 percent for private L2 cache, but 3.8 percent for shared L2 cache configuration. As addressed before, mixing data with different patterns from multiple workloads makes SBAC less efficient.

We also evaluate the results of execution time after using SBAC and compare them with the baseline. The results of

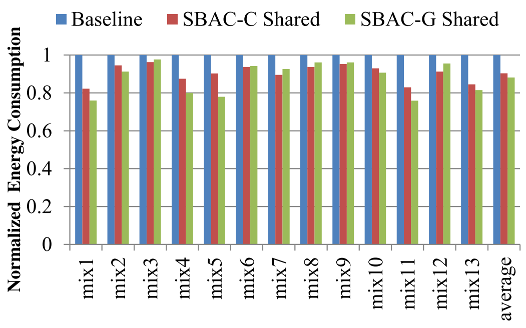


Fig. 15. Normalized energy consumption after using core-based SBAC and group-based SBAC for shared L2 cache.

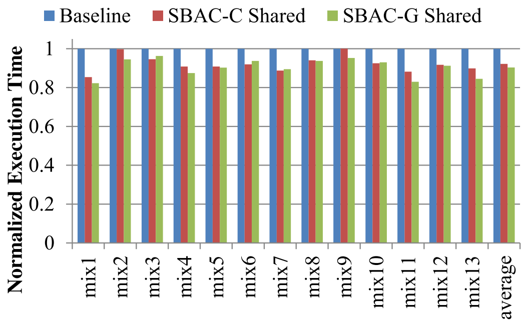


Fig. 16. Normalized execution time after using core-based SBAC and group-based SBAC for shared L2 cache.

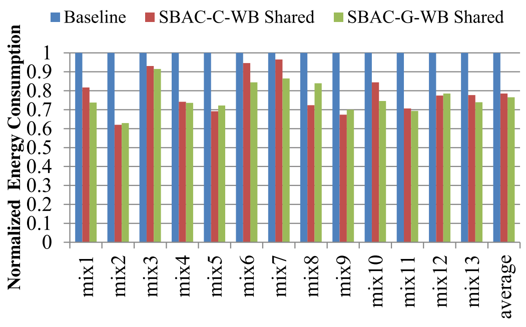


Fig. 17. Normalized cache energy consumption after using core-based SBAC and group-based SBAC on write-back data.

applying SBAC on data loaded from L3 to L2 are listed in Fig. 14. We can find that the trends of these results are similar to those for energy consumption optimization. On average, the performance is improved by 2.1 and 4.3 percent for shared and private configured L2 cache.

In order to improve SBAC for multi-programmed workloads, we propose core-based and group-based SBAC for shared L2. The results are shown in Fig. 15. It is easy to find that energy consumption is further reduced after using these techniques. Core-based SBAC can further reduce the energy consumption by about 9.6 percent, because it helps isolate the interference of data among different workloads, while shared cache supplies sufficient space. And the group-based one can further reduce the consumption by 11.9 percent, because it performs better on isolating the mixed data patterns.

The performance of core-based and group-based SBAC is similar to those of energy. The results are shown in Fig. 16. Core-based and group-based SBAC reduce the overall execution time by 7.8 and 9.6 percent, respectively.

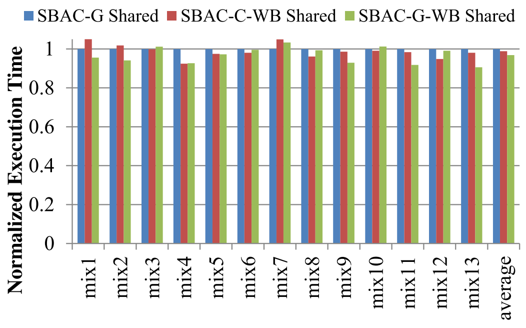


Fig. 18. Normalized execution time after using core-based SBAC and group-based SBAC on write-back data.

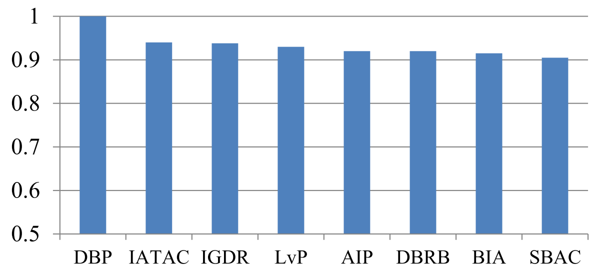


Fig. 19. Comparison of performance between our bypassing scheme and other approaches.

From the results in Figs. 13 and 15, we can find out that SBAC works well on cache bypassing of data loaded from L3 to L2. Now, we further apply SBAC to data written back from L2 to L3 and the results are shown in Fig. 17. On average, energy consumption can be reduced by 21.4 and 23.4 percent after using core-based SBAC and group-based SBAC, respectively. Unlike timing overhead of write-back data, which may be hidden for some workloads, energy consumption of write-back data is added to total energy consumption directly.

In Fig. 18, we show the results after applying SBAC on write-back data. On average, the performance can be further improved by about 1.1 and 3.2 percent for shared and private L2 caches respectively. The improvement is less significant compared to the case for data loaded from L3 to L2. The reasons is because write-back data is not on critical path and we simply set $0.5 \times \textit{write latency}$ as average blocking time. And due to the bypassing of several reused write back data, the miss of these data cause significant time overhead.

6.5 Comparison with Other Approaches

We compare normalized average cache access latency between our bypassing scheme and prior approaches for single application, shown in Fig. 19. Our bypassing scheme can outperform other approaches in respect of cache access latency. The main reason is that the asymmetry access operations are not considered in prior approaches. Note that we do not provide comparison for cases of energy optimization and multi-programmed application. It is because most prior approaches cannot work with these cases. We also compare design and operation overhead in Table 5. We estimate the design overhead by extra storage (per line and cache), area overhead of control logic, and extra cache operations. Area results are synthesized by Synopsys Design Compiler with TSMC 45 nm library. It is easy to find that SBAC costs much less storage, area, and operations.

7 Related Work

Most studies about asymmetric-access caches focus on how to mitigate the problems caused by write operations. For example, write halt and PreSET techniques are proposed to hide long write latency of these asymmetric-access caches or main memory [26], [31], [33]. The hybrid cache architecture places frequently updated data in the symmetric-access cache (e.g., SRAM) [31], [39]. The replacement policy can also be tailored by evicting cache lines with less updated bits [43]. Wear-leveling and error correction codes can be applied in last level asymmetric-access caches to improve their lifetimes and reliability [16], [32]. We can find that there lacks cache bypassing techniques for asymmetric-access caches.

Prior works for symmetric-access cache bypassing can be categorized into either static or dynamic approaches. Static approaches normally rely on profiling-guided compiler to identify cache lines, which should be avoided for placement [5], [40]. But the static methods may work unefficiently if the program rely largely on user input. Most of dynamic methods need to maintain a signature-like Cs requests [1], [7], [9], [14], [15], [19], [22], [24], [27], [28], [35], [36], [37]. Some traditional cache bypassing decisions can be made based on the program counter (PC) of the load instruction [7], [9], [37] or on the memory address of the access [14], [15], [27], [28]. A memory address based approach works well if the same memory address has similar locality behavior, while a PC-based approach can help

TABLE 5
Design Overhead Comparison

Approaches	Multiprogram Support	Storage Overhead (bits) per line	Storage Overhead (bits) global	Area Overhead (μm^2)	Operation Overhead
DBP [23]	No	-	2 M	102.78	2-level table lookup/update
IATAC [1]	No	31	288	293.63	6b comp + 31b update + 16-entry CAM lookup/update
IGDR [36]	No	-	42.5 K	4.12×10^4	5 table lookup/update
LvP [20]	No	17	40 K	62.27	5b comp + 17b update + 1 table lookup/update
AIP [20]	No	21	40 K	30.34	5b comp + 21b update + 1 table lookup/update
DBRB [19]	Yes	-	13.75 K	40.28	15b comp + 15b update + 3 table lookup/update
BIA [8]	Yes	3/L2+2/L3	1.8 K	192.6	5b update + 16-entry CAM lookup/update
SBAC	**Yes**	**1/L2+2/L3**	**73**	**36.16**	**3b update + 2b comp + 1 counter update**
SBAC-C	Yes	1/L2+2/L3	146	144.64	3b update + 2b comp + 1 counter update
SBAC-G	Yes	3/L2+2/L3	292	222.32	3b update+2b comp+1 counter update

when the data access patterns have a different locality behavior in different code segments. Abella et al. proposed IATAC [1], to predict dead cache lines by measuring a cache line's timing. Lai et al. proposed DBP [22], to predict dead blocks based on addresses of the last few accesses. Takagi and Hiraki [35] proposed IGDR to collect access interval distributions, but it is costly to implement in hardware. Kharbutli and Solihin proposed LvP and AIP to make bypassing decisions, using hybrid PC and address information [19]. Instead of using the individual access signature, *Instruction Sequence History Signature* and *cache bursts* can be traced to predict the behavior of grouped cache references [25], [38]. Such signatures are more regular than that of an individual cache request. Khan and et al. introduced DBRB to use statistical sampling of PCs of a small number of cache sets using partial tags to reduce overhead [18].

Recently, a bypassing algorithm based on statistical sampling of operation count, called BIA, is proposed for exclusive last level caches [8]. We differentiate this algorithm from other studies because its bypassing decision is made without using any signature-like structures. The percentages of dead blocks in different data categories are traced to judge the bypassing. Although this bypassing algorithm is statistics based, its high frequency of updating bypassing decisions causes considerable overhead. The algorithm only works for write back operations to last level cache. On the contrary, our scheme overcomes these shortages.

8 Conclusions

Emerging asymmetric-access caches are competitive for design of future cache hierarchy. Traditional cache bypassing techniques are not efficient for these asymmetric-access caches. In this work, we propose the statistics based cache bypassing method named SBAC. With the help of a theoretical model, we analyze the benefits of cache bypassing. Then, proper bypassing decisions are made based on DRC probability. In addition, we propose core-based and group-based SBAC to improve working efficiency of SBAC for multi-programmed workloads. Compared with prior approaches, SBAC has the advantages of low design overhead and compatibility for different cache configurations. The experimental results show improvement of cache performance and energy efficiency after using SBAC.

Acknowledgments

This work was supported by NSF CNS-1116171, ECCS-1202225, National Natural Science Foundation of China (No. 61572045, No. 61202072 and No. 61103028), and National High-tech R&D Program of China (No. 2013AA013201).

References

[1] J. Abella, A. González, X. Vera, and M. F. P. O'Boyle, "IATAC: A smart predictor to turn-off l2 cache lines," *ACM Trans. Archit. Code Optim.*, vol. 2, no. 1, pp. 55–77, Mar. 2005.

[2] F. Bedeschi, R. Fackenthal, C. Resta, E. M. Donzè, M. Jagasivamani, E. C. Buda, F. Pellizzer, D. W. Chow, A. Cabrini, G. M. A. Calvi, et al., "A bipolar-selected phase change memory featuring multi-level cell storage," *IEEE J. Solid-State Circuits*, vol. 44, no. 1, pp. 217–227, Jan. 2009.

[3] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, Aug. 2011.

[4] G. W. Burr, B. N. Kurdi, J. C. Scott, C. H. Lam, K. Gopalakrishnan, and R. S. Shenoy, "Overview of candidate device technologies for storage-class memory," *IBM J. Res. Develop.*, vol. 52, no. 4.5, pp. 449–464, Jul. 2008.

[5] C.-H. Chi and H. Dietz, "Improving cache performance by selective cache bypass," in *Proc. 22nd Annu. Hawaii Int. Conf. Syst. Sci., Archit. Track*, vol. 1, pp. 277–285, Jan. 1989.

[6] X. Dong, C. Xu, Y. Xie, and N. Jouppi, "Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 31, no. 7, pp. 994–1007, Jul. 2012.

[7] H. Dybdahl and P. Stenström, "Enhancing last-level cache performance by block bypassing and early miss determination," in *Proc. 11th Asia-Pacific Conf.*, 2006, pp. 52–66.

[8] J. Gaur, M. Chaudhuri, and S. Subramoney, "Bypass and insertion algorithms for exclusive last-level caches," in *Proc. 38th Annu. Int. Symp. Comput. Archit.*, 2011, pp. 81–92.

[9] A. González, C. Aliagas, and M. Valero, "A data cache with multiple caching strategies tuned to different types of locality," in *Proc. 9th Int. Conf. Supercomput.*, 1995, pp. 338–347.

[10] P. Hammarlund, A. J. Martinez, A. A. Bajwa, D. L. Hill, E. Hallnor, H. Jiang, M. Dixon, M. Derr, M. Hunsaker, R. Kumar, R. B. Osborne, R. Rajwar, R. Singhal, R. D'Sa, R. Chappell, S. Kaushik, S. Chennupaty, S. Jourdan, S. Gunther, T. Piazza, and T. Burton, "Haswell: The fourth-generation Intel core processor," *IEEE Micro*, vol. 34, no. 2, pp. 6–20, Mar./Apr. 2014.

[11] M. Hosomi, H. Yamagishi, T. Yamamoto, K. Bessho, Y. Higo, K. Yamane, H. Yamada, M. Shoji, H. Hachino, C. Fukumoto, et al., "A novel nonvolatile memory with spin torque transfer magnetization switching: Spin-RAM," in *Proc. IEEE Int. Electron Devices Meet., Tech. Dig.*, 2005, pp. 459–462.

[12] I. H. Inoue, S. Yasuda, H. Akinaga, and H. Takagi, "Nonpolar resistance switching of metal/binary-transition-metal oxides/metal sandwiches: Homogeneous/inhomogeneous transition of current distribution," *Phys. Rev. B*, vol. 77, no. 3, p. 035105, 2008.

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

[14] T. L. Johnson, D. A. Connors, M. C. Merten, and W.-m. W. Hwu, "Run-time cache bypassing," *IEEE Trans. Comput.*, vol. 48, no. 12, pp. 1338–1354, Dec. 1999.

[15] T. L. Johnson and W.-m. W. Hwu, "Run-time adaptive cache hierarchy management via reference analysis," in *Proc. 24th Annu. Int. Symp. Comput. Archit.*, 1997, pp. 315–326.

[16] Y. Joo, D. Niu, X. Dong, G. Sun, N. Chang, and Y. Xie, "Energy- and endurance-aware design of phase change memory caches," in *Proc. Eur. Des. Autom. Assoc.*, 2010, pp. 136–141.

[17] B. Jun and P. Kocher, "The Intel random number generator," *Cryptography Research Inc. white paper*, 1999.

[18] S. M. Khan, Y. Tian, and D. A. Jimenez, "Sampling dead block prediction for last-level caches," in *Proc. 43rd Annu. IEEE/ACM Int. Symp. Microarchit.*, 2010, pp. 175–186.

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

[20] T. Kishi, H. Yoda, T. Kai, T. Nagase, E. Kitagawa, M. Yoshikawa, K. Nishiyama, T. Daibou, M. Nagamine, M. Amano, et al., "Lower-current and fast switching of a perpendicular TMR for high speed and high density spin-transfer-torque MRAM," in *Proc. IEEE Int. Electron Devices Meet.*, 2008, pp. 1–4.

[21] M. N. Kozicki, M. Balakrishnan, C. Gopalan, C. Ratnakumar, and M. Mitkova, "Programmable metallization cell memory based on Ag-ge-s and Cu-ge-s solid electrolytes," presented at the Non-Volatile Memory Technology Symp., Dallas, TX, USA, 2005.

[22] A.-C. Lai, C. Fide, and B. Falsafi, "Dead-block prediction & dead-block correlating prefetchers," in *Proc. 28th Annu. Int. Symp. Comput. Archit.*, 2001, pp. 144–154.

[23] J. Li, P. Ndai, A. Goel, H. Liu, and K. Roy, "An alternate design paradigm for robust spin-torque transfer magnetic RAM (STT MRAM) from circuit/architecture perspective," in *Proc. Asia South Pacific Des. Autom. Conf.*, 2009, pp. 841–846.

[24] L. Li, I. Kadayif, Y.-F. Tsai, N. Vijaykrishnan, M. T. Kandemir, M. J. Irwin, and A. Sivasubramaniam, "Leakage energy management in cache hierarchies," in *Proc. Int. Conf. Parallel Archit. Compilation Tech.*, 2002, pp. 131–140.

[25] H. Liu, M. Ferdman, J. Huh, and D. Burger, "Cache bursts: A new approach for eliminating dead blocks and increasing cache efficiency," in *Proc. 41st Annu. IEEE/ACM Int. Symp. Microarchit.*, 2008, pp. 222–233.

[26] M. K. Qureshi, M. M. Franceschini, A. Jagmohan, and L. A. Lastras, "Preset: Improving performance of phase change memories by exploiting asymmetry in write times," in *Proc. 39th Annu. Int. Symp. Comput. Archit.*, vol. 40, no. 3, pp. 380–391, Jun. 2012.

[27] J. Rivers and E. S. Davidson, "Reducing conflicts in direct-mapped caches with a temporality-based design,"in *Proc. Int. Conf. Parallel Process. Software*, 1996, pp. 154–163.

[28] J. A. Rivers, E. S. Tam, G. S. Tyson, E. S. Davidson, and M. Farrens, "Utilizing reuse information in data cache management," in *Proc. 12th Int. Conf. Supercomput.*, 1998, pp. 449–456.

[29] Z. Shao, Y. Liu, Y. Chen, and T. Li, "Utilizing PCM for energy optimization in embedded systems," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI*, 2012, pp. 398–403.

[30] C. W. Smullen, V. Mohan, A. Nigam, S. Gurumurthi, and M. R. Stan, "Relaxing non-volatility for fast and energy-efficient STT-RAM caches," in *Proc. IEEE 17th Int. Symp. High Perform. Comput. Archit.*, 2011, pp. 50–61.

[31] G. Sun, X. Dong, Y. Xie, J. Li, and Y. Chen, "A novel architecture of the 3D stacked MRAM L2 cache for CMPs," in *Proc. IEEE 15th Int. Symp. High Perform. Comput. Archit.*, Feb. 2009, pp. 239–249.

[32] G. Sun, C. Xu, and Y. Xie, "Modeling and design exploration of FBDRAM as on-chip memory," in *Proc. Des. Autom. Test Eur. Conf. Exhib.*, Mar. 2012, pp. 1507–1512.

[33] G. Sun, Y. Zhang, Y. Wang, and Y. Chen, "Improving energy efficiency of wtire-asymmetric memories by log style write," in *Proc. ACM/IEEE Int. Symp. Low Power Electron. Des.*, 2012, pp. 173–178.

[34] Z. Sun, X. Bi, and H. Li, "Process variation aware data management for STT-RAM cache design," in *Proc. ACM/IEEE Int. Symp. Low Power Electron. Des.*, 2012, pp. 179–184.

[35] M. Takagi and K. Hiraki, "Inter-reference gap distribution replacement: An improved replacement algorithm for set-associative caches," in *Proc. 18th Annu. Int. Conf. Supercomput.*, 2004, pp. 20–30.

[36] E. S. Tam, J. A. Rivers, V. Srinivasan, G. S. Tyson, and E. S. Davidson, "Active management of data caches by exploiting reuse information," *IEEE Trans. Comput.*, vol. 48, no. 11, pp. 1244–1259, Nov. 1999.

[37] G. Tyson, M. Farrens, J. Matthews, and A. R. Pleszkun, "A modified approach to data cache management," in *Proc. 28th Annu. Int. Symp. Microarchit.*, 1995, pp. 93–103.

[38] C.-J. Wu, A. Jaleel, W. Hasenplaugh, M. Martonosi, S. C. Steely, Jr., and J. Emer, "Ship: Signature-based hit predictor for high performance caching," in *Proc. 44th Annu. IEEE/ACM Int. Symp. Microarchit.*, 2011, pp. 430–441.

[39] X. Wu, J. Li, L. Zhang, E. Speight, R. Rajamony, and Y. Xie, "Hybrid cache architecture with disparate memory technologies," in *Proc. 36th Annu. Int. Symp. Comput. Archit.*, 2009, pp. 34–45 .

[40] Y. Wu, R. Rakvic, L.-L. Chen, C.-C. Miao, G. Chrysos, and J. Fang, "Compiler managed micro-cache bypassing for high performance EPIC processors," in *Proc. 35th Annu. IEEE/ACM Int. Symp. Microarchit.*, 2002, pp. 134–145.

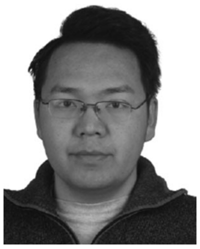
[41] C. Xu, X. Dong, N. P. Jouppi, and Y. Xie, "Design implications of memresistor-based RRAM cross-point structures," in *Proc. Des., Autom. Test Eur. Conf. Exhib.*, 2011, pp. 1–6.

[42] Y. Zhang, H. Wu, Y. Bai, A. Chen, Z. Yu, J. Zhang, and H. Qian, "Study of conduction and switching mechanisms in al/alox/wox/w resistive switching memory for multilevel applications," *Appl. Phys. Lett.*, vol. 102, no. 23, 2013.

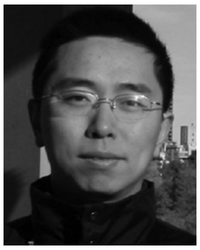
[43] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "Energy reduction for STT-RAM using early write termination," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des. - Dig. Tech. Papers*, 2009, pp. 264–268.

Guangyu Sun received the BS and MS degrees from Tsinghua University, Beijing, China, in 2003 and 2006, respectively. He received the PhD degree in computer science from Pennsylvania State University, State College, PA, USA, in 2011. He is currently an assistant professor of CECA at Peking University, Beijing, China. His research interests include computer architecture, VLSI Design, and electronic design automation (EDA). He has published more than 60+ journals and refereed conference papers in these areas. He has also served as a peer reviewer and technical referee for several journals, which include *IEEE Micro*, *IEEE Transactions on Very Large Scale Integration Systems* (TVLSI), *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (TCAD), etc. He is a member of the CCF and IEEE.

Chao Zhang received the BS degree in microelectronics from Peking University, Beijing, China, in 2012. He is currently working toward the PhD degree from Peking University. He is also a visiting scholar at the University of California, Santa Barbara, USA, since 2015. His current research interests include architectural optimization for STT-RAM and Domain wall racetrack memory. He is a student member of IEEE.



Peng Li received the BS degree in computer science and technology from the Harbin Institute of Technology, Harbin, China in 2002, and the PhD degree in computer science and technology from Tsinghua University, Beijing, China, in 2008. Then he joined Intel Labs China as a research scientist and later as senior research scientist. He was a postdoc in Peking University when he finished this work. He is a member of IEEE.



Tao Wang received the BS and PhD degrees from Peking University, Beijing, China, in 1999, and 2006, respectively. He is currently an associate professor in Peking University. His research interests include computer architecture, reconfigurable logic, wireless network architecture, and mobile cloud computing. He is a member of IEEE.

Yiran Chen received the BS and MS degrees in electrical engineering from Tsinghua University, China, in 1998 (Hons.), 2001 (Hons.), respectively and the PhD in ECE from Purdue University, W. Lafayette, IN, USA. He moved to the University of Pittsburgh as an assistant professor, Electrical and Computer Engineering Department in September 2010 and then promoted as an not a associate professor in September 2014. His research interests include VLSI design/CAD for nano-scale Silicon and non-Silicon technologies, low-power circuit design and computer architecture, emerging memory technologies and nano-scale reconfigurable computing system and sensor system. He is a member of IEEE.