DB-AMB: Dataset-Based Allocation, Migration, and Bypassing in hybrid non-volatile/SRAM caches

Gregory Vaumourin, Alexandra Jimborean, and David Black-Schaffer

Abstract—Non-volatile memories (NVRAM) have emerged as an appealing solution to reduce static energy and increase capacity in last-level caches (LLCs) but bring an important drawback with write operations. Hybrid caches mitigate this effect by combining both SRAM and NVM technologies to get the best of both. The key challenge in hybrid cache designs is the data allocation policy that must detect and map the write-intensive data to the SRAM array, without decreasing data reuse due to increased misses in the smaller SRAM. To address this challenge, we propose a Dataset-Based (DB) policy that groups cache lines into sets with uniform reuse and write-intensity properties called datasets. This allows us to rapidly learn and predict these two properties at a dataset granularity, as information from all accesses to that dataset are merged. With these properties, we can choose the *allocation*, *migration*, and *bypassing* strategies (AMB) for each dataset, by calculating the expected energy given a window of recent accesses to the dataset. The complete policy, DB-AMB, achieves a better trade-off between NVRAM writes and SRAM misses and improves performance by 7.9% and reduce LLC power consumption by 9.9% compared to previous policies.

Index Terms—Cache memories, hybrid caches, non-volatile memories, STT-MRAM.

1 Introduction

Non-Volatile Memories (NVRAM) both retain their state without power and also deliver much denser storage and lower leakage than traditional volatile (SRAM and DRAM) memories. These latter characteristics make them appealing for use across the memory hierarchy wherever memory capacity is at a premium. However NVRAM are not a panacea: they typically pay significant penalties over volatile memories for *writing* data in performance (latency), energy, and durability. The most mature NVRAM technologies today include Phase Change Memory (PCM), Resistive RAM (ReRAM) and Spin-Torque Transfer Magnetic RAM (STT-MRAM). STT-RAM is generally considered to be the best match for caches due to its low read latency (close to that of SRAM) and an endurance that is predicted to achieve 10^{15} writes per cell [1]. In contrast, PCM's and ReRAM's endurance is dramatically lower, between 10^{8}-10^{10} writes [2], [3]. While read operations in STT-RAM are comparable to SRAM, the latency and energy of write operations are typically an order of magnitude higher.

This asymmetry between reading and writing has led to a range of *hybrid* cache designs that combine traditional volatile SRAM storage with NVRAM storage in an attempt to get the best of both worlds: low-cost writes to a smaller SRAM array while benefiting from the higher capacities and lower leakage of a large NVRAM array. The challenge in such designs is developing allocation policies that learn how to allocate data to the two memory arrays Previous policies [4], [5], [6], [7], [8], [9] for hybrid NVRAM-SRAM caches usually predict the *write intensity* of cache lines. This information allows them to allocate write-intensive data in the cheap-to-write (but low-capacity) SRAM array, while putting all other read-only data in the high-capacity (but expensive-to-write) NVRAM. However, allocating data based only on write-intensity can lead to an increase in cache misses, as the SRAM holding write-intensive data is much smaller in a hybrid cache. Relaxing this policy leads to an increase in data written to the NVRAM, which increases latency and energy. The trade-off between increasing misses on write-intensive data on SRAM and decreasing the number of writes to the NVRAM is an important challenge in hybrid cache design.

To accurately trade-off increases in misses to write-intensive data in the SRAM with increases in writes to the NVRAM, the hybrid cache allocation policy must be able to predict both the *write-intensity* and the *reuse* of the data. Although most previous solutions [4], [7], [8], [10], [11] predict efficiently the write-intensity, they do not directly predict of the reuse of the data. Whether they do not predict the reuse assuming a systematic reuse of the cache line in the allocated array or measure it indirectly by using on thresholds to perform an efficient trade-off between SRAM misses and NVRAM writes.

One of the major implementation challenges of existing hybrid cache designs is learning data behavior (primarily write-intensity) *at the cache line granularity*, as the system must observe multiple accesses to each cache line over a long enough period to determine its behavior. To address this, we assign a *signature* to each cache line and group cache lines with the same signatures into *datasets*. This allows us to learn the behavior at the dataset level by aggregating what the system sees over many cache lines, which makes for faster learning and reduces metadata storage. Learning

• *G. Vaumourin, A. Jimborean and D. Black-Schaffer are with the Department of Information Technology, Uppsala University, Uppsala, Sweden. E-mails: {gregory.vaumourin,alexandra.jimborean,david.black-schaffer}@it.uu.se*

Manuscript received February 28, 2018.

behavior at the dataset level further simplifies the task of learning dataset reuse behavior, which allows us to better trade-off write-intensity and reuse when allocating data in either the SRAM or NVRAM. Our design uses this per-dataset reuse and write-intensity information to *estimate the energy* that each datasets' accesses would cost if placed in NVRAM or SRAM, and chooses the lowest-energy allocation. By including both write-intensity and reuse in this calculation, DB-AMB can find a better trade-off between increasing SRAM misses and decreasing NVRAM writes.

While data placement is the main challenge in hybrid caches, there are two other key optimizations that significantly improve efficiency: migration of data between the SRAM and NVRAM (to adapt to application's phases) and cache bypassing of dead blocks (to avoid NVRAM writes that will not lead to subsequent reuses). Traditionally these optimizations have been difficult to tune for hybrid caches as they can result in significant overheads due to excessive migrations or misses from bypassing. With the Dataset-Based Allocation, Migration and Bypassing policy (DB-AMB) policy, we show how having write-intensity and reuse information *at the dataset granularity* allows us to intuitively, and accurately, predict whether new data will not be reused at all (and thereby correctly bypass) and rapidly update policies for large numbers of cache lines (and thereby choose appropriate migration strategies).

This work makes the following contributions: (1) We use signature-based datasets to make it practical and efficient to collect combined write-intensity and reuse properties. (2) We develop the Datasets-Based Allocation Migration and Bypassing policy (DB-AMB) that chooses correct placement, migration, and bypassing for each dataset by estimating the energy cost for each option. On the evaluated benchmarks, DB-AMB improves performance by 7.9% and reduces LLC power consumption by 9.9% compared to a baseline saturation counter policy [8] for single-threaded workloads. For multi-threaded applications, the performance is improved by 6.5% and the energy consumption is reduced by 7.7%.

2 Datasets Analysis for hybrid caches

To accurately determine data allocation, hybrid caches need to be able to learn how an application behaves to predict the write intensity and reuse properties of the application's data. However, learning and predicting this behavior is difficult when tracking data at the cache line granularity. To address this, we propose forming cache lines into sets. If we can identify sets with uniform behavior, then we can then aggregate our learning and prediction across them to simplify the design.

2.1 Datasets for Identifying Write-Intensity and Reuse Properties

To build sets of cache lines, we first assign a *signature* to each cache line, and then define our sets as all cache lines with the same signature. For this to work, we need signatures that group cache lines with similar reuse and write-intensity behavior. Previous work has proposed all manner of methods for generating such signatures, including instructions, instruction sequences, memory regions, addresses, page offsets, instructions that first access data, and combinations of

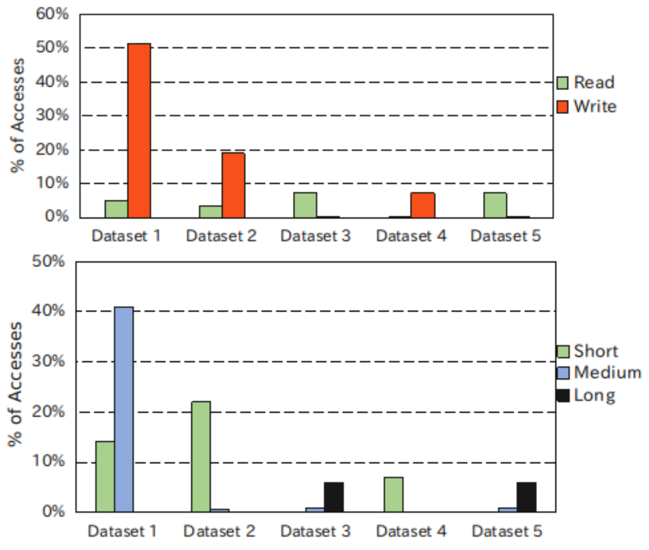


Fig. 1. Effectiveness of miss-PC signatures for dataset classification. (Top, a) write-intensity and bottom, b) reuse). The 5 most accessed datasets in the ferret benchmark demonstrate that the miss-PC signatures identify datasets with highly-uniform behavior.

the above [4], [5], [12], [13]. For our work we define a cache line's signature based on its *miss-PC*, that is the instruction (PC) that caused the cache line to be loaded into the cache. Using the miss-PC as a dataset signature has been shown to be correlated with both reuse behavior (Sembrant et al. [12]) and write intensity (Ahn et al. [4]). In addition, the miss-PC maintains the observed correlations in the last level cache, even after being filtered by the private caches in the hierarchy.

For dataset formation to be useful, the datasets need to have uniform behavior in terms of their write-intensity and reuse. Figure 1 shows the behavior of the 5 most accessed datasets (as identified by the miss-PC signature) at the last-level cache for the ferret application from PARSEC [14]. (Behavior for other applications in PARSEC is largely similar). In Figure 1a, we see that datasets 3 and 5 are mostly read, while the others are mostly written. This indicates that the miss-PC-based signature has been able to identify datasets with consistent behavior.

The second behavior we are after is how datasets are reused in the cache. In particular, we wish to know if the data is reused frequently enough to be a hit in the smaller SRAM array (short reuse), the larger NVRAM array (medium reuse), or if it is reused so infrequently as to be a miss altogether in the LLC (long reuse). For this analysis we use the position of the accessed block in the LRU stack to characterize reuse. In Figure 1b we evaluate a 16-way cache, and use the LRU position upon access to determine the reuse (0-3=short, 4-15=medium, ¿15, e.g., miss=long). Here we see that the same miss-PC-based signature identifies datasets with largely uniform behavior. Datasets 3 and 5 realize almost no reuse in the LLC, while datasets 2 and 4 show significant reuse even in the smaller SRAM array of the LLC, and dataset 1 would be best placed in the NVRAM, based on reuse. We can draw the same conclusion as for write-intensity: miss-PC-based signatures generate datasets that are well-formed in terms of reuse behavior.

2.2 Datasets for Faster Learning

Traditional approaches to learning write-intensity and reuse in hybrid caches do so at a cache line granularity. This requires that they observe for long enough to see each cache line's behavior, before they can make a reasonable decision as to how to allocate it, and that they store the behavior for each cache line. As a result, the learning rate is proportional to the number of cache lines in the cache. Furthermore, even after learning the behavior for a group of cache lines, it is not clear how to apply this to a new cache line brought into the cache. These challenges have typically lead to reactive policies that use a default placement for cache lines that only change later as they learn the behavior [6], [8], [10], [11].

For this work, we group multiple cache lines into datasets. This makes the learning rate proportional to the number of datasets in the cache. As programs typically accesses only a few datasets in any given phase, as opposed to many cache lines, we can therefore aggregate accesses to each dataset and learn their behavior far more rapidly. In addition, we need only to store the behavior on a per-dataset basis, which reduces the aggregate metadata overhead[1]. Perhaps most importantly, once we have learned the behavior of a dataset, we can trivially apply it to new cache lines that belong to that dataset without having to wait to learn their behavior. This allows us to apply the learned allocation policies much sooner, which increases efficiency, and allows us to design more efficient policies for allocation, migration and bypassing.

As datasets have already been studied in the past for standard caches, the novelty of our approach is to apply this technique for the hybrid cache data allocation problem as it allows to learn faster, the two data properties that are needed to take efficient allocation decision: reuse and write-intensity.

3 Dataset-Based Allocation, Migration and Bypassing

The proposed Dataset-Based Allocation, Migration and Bypassing policy (DB-AMB) identifies datasets using their miss-PC signature, learns the reuse and write intensity of each dataset over time, and then uses this information to calculate the expected energy for different allocations of the dataset. From this, it chooses allocation, migration, and bypass policies for the dataset, and applies the policy as data is brought into the hybrid cache.

3.1 Learning Datasets

In order to allocate datasets, the DB-AMB first has to learn the dataset behavior by collecting a window of accesses. Two pieces of information are collected per access: write-intensity (how much the dataset is written vs. only read) and reuse (how likely accesses to the dataset are to hit in the SRAM or NVRAM array of the cache). Write-intensity is easy to collect: for each access to a dataset, DB-AMB simply records whether the access was a read or write. For reuse, DB-AMB categorizes each access as short/medium/long reuse. Based on the reuse information, a prediction is made whether the access would hit in both SRAM and NVRAM arrays (short reuse), only hit in NVRAM (medium reuse) or miss in both arrays (long reuse).

1. Note that we do need to store per-cache line information as to which dataset it belongs to, but this can be cheap as discussed later.

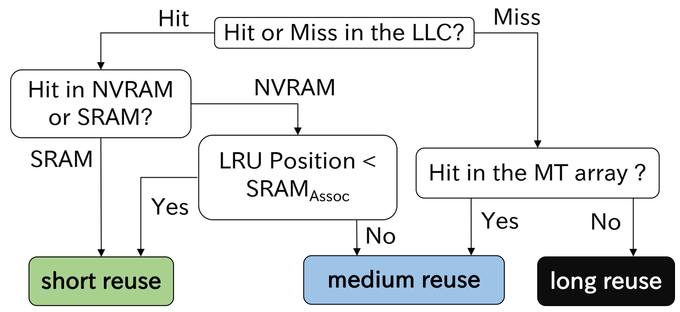


Fig. 2. Computation of the reuse information based on where the tag is found between short (predicted hit in SRAM and NVRAM array), medium (predicted hit in NVRAM, miss in SRAM) and long (miss in both).

Fig. 2 details how this information is computed. If the incoming access is a hit in the SRAM array, the access is directly classified as short reuse as we assume that this access would also hit in the NVRAM array. But if the hit occurs on the NVRAM array, then the LRU position of the block determines whether the reuse is short or medium. If the LRU position is inferior to the SRAM associativity, we predict a hit in SRAM (if the data had been allocated there), then it is categorized as a short reuse, and otherwise as a medium reuse. Upon misses, we want to be able to distinguish between blocks that have been evicted in the SRAM array but would have been a hit if allocated in NVRAM (i.e. a medium reuse) or just a regular miss (i.e. a long reuse). To perform this prediction, the SRAM array is extended with a missing tags (MT) array that keeps track of the last evicted tags that have been evicted from the SRAM array. Upon a miss, the missing tags (MT) array is checked, and the access is labeled as medium reuse in case of a hit in the MT array, and long reuse otherwise.

Note that the reuse information computed this way is not perfect as it does not reflect the increased cache pressure – if the data had been initially stored in the other data array – but still gives a good estimation of the access needs in terms of cache size.

Fig.3 illustrates this process on a simpler view of the hybrid cache: only one set is represented with the LRU stack of the different data arrays. Dataset X is composed of blocks A and B and is allocated in SRAM while dataset Y is composed of C and D and allocated in NVRAM. Upon the first access to A, the block is not present in neither of the data arrays nor the missing tag array, thus a miss occurs, classified as a long read. Block A is then inserted in the MRU position of the NVRAM array according to dataset X allocation policy. The next access to A occurs while the block is still at the top of the LRU stack of the NVRAM, so this access is labeled as a short write. B is then accessed while being in the LRU position which indicates a medium reuse as its LRU position is 4 and the SRAM associativity is 2 in this case. The access to C hits in the SRAM array which indicates a short read. The access to D misses in both data arrays, but hit in the missing tags array which means that

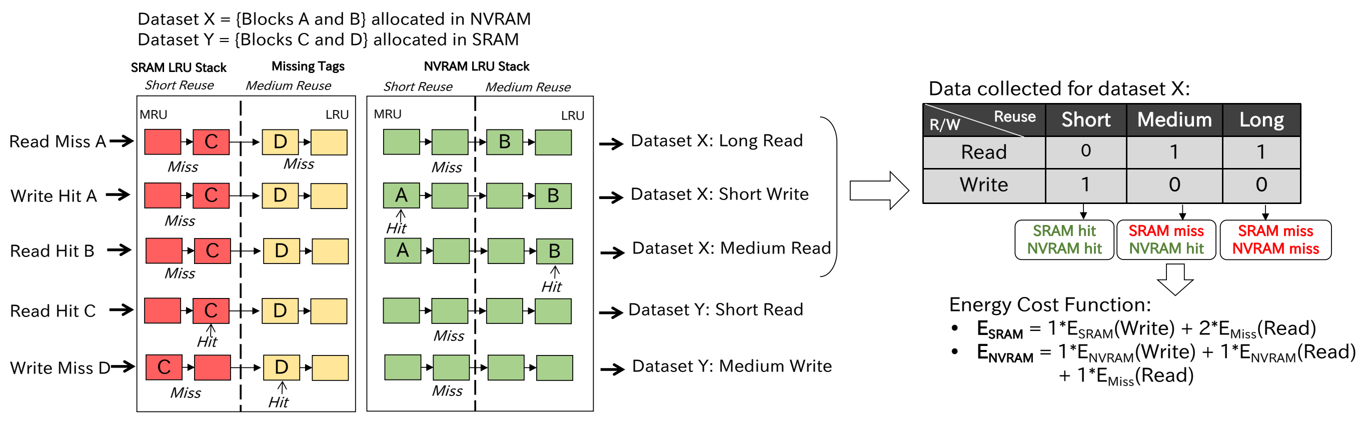


Fig. 3. Learning the write-intensity and reuse of a dataset. For each dataset (X and Y) the location of the data in the cache on an access (SRAM, SRAM Missing Tags, NVRAM, or miss) determines the expected reuse of the data (short, medium, or long). The reuse and the type of access (read or write) is recorded in a table to characterize the dataset. From this table, the energy for allocating the dataset in the NVRAM or SRAM can be calculated by using the reuse distance in order to estimate how many cache hits each allocation would lead to and the nature of the access (red/write) in order to determine the energy cost.

D just has been evicted from the SRAM array. The access is labeled as medium write as it is predicted to hit in the NVRAM array.

Reuse and write-intensity information of access are collected per dataset, which means that accesses performed on cache lines belonging to the same datasets fill the same dataset window. The size of the window is a parameter that has been tuned to 20 accesses in our solution. Once, 20 accesses have been collected per dataset, DB-AMB updates the dataset allocation policy based on the collected information and resets the window.

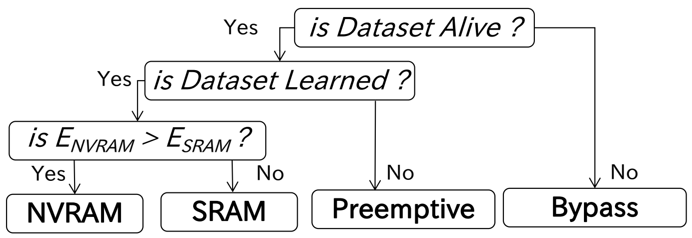


Fig. 4. Determining data allocation. Dead datasets are bypassed. If there is not enough information about a datasets' behavior, its cache lines are preemptively allocated, and otherwise allocation is based on estimating the minimum energy between NVRAM and SRAM allocation.

3.2 Datasets Allocation Policy

The allocation policy determines whether new data for a given dataset should be installed in the SRAM or NVRAM. The allocation is chosen by estimating the dynamic energy cost for NVRAM allocation (E_{NVRAM}) and SRAM allocation (E_{SRAM}) based on the accesses (write-intensity and reuse) observed in the last history window. We use the last window of accesses to predict the behavior in the future, which we have found to be an effective assumption.

The energy cost function uses the reuse information to compute if the access will hit/miss in the allocated array. For each short reuse access in the window, a hit is expected for both allocation to the SRAM and NVRAM. For those accesses, the energy estimation would be the access cost (read or write depending on the access) to the SRAM or NVRAM. A medium reuse access is predicted to be a miss if the dataset is allocated in the SRAM, and the energy would therefore include that of accessing the DRAM. If allocated in the NVRAM, however, the energy cost would only be that of an NVRAM hit (read or write). For long reuse accesses, the energy is counted directly as a DRAM access as we expect a miss. Based on this energy estimation, the allocation policy with the lower predicted energy is chosen.

An example of how the dynamic energy is computed is shown in Figure 3. For this example, we examine the access window of dataset X, we assume a window of 3 accesses for the sake of explanation[2]. The window is composed of one short write, one medium read and one long read. Therefore, 2 DRAM accesses are counted for E_{SRAM} and only one for E_{NVRAM}. The cost of accesses to the different elements are input data to the energy function cost that depend on the cache design and are obtained with the NVSim simulation [15]. In our simulations, as one NVRAM read is much cheaper than a DRAM read, E_{NVRAM} is smaller than E_{SRAM} and the allocation policy of dataset X remains set to NVRAM.

The data allocation policy can be determined and applied as soon as a history window has been observed for the dataset. However, before that point there is no information to guide placement. During this initial time, DB-AMB uses a *preemptive* policy that allocates each cache line based on its first access: if it is a write it is allocated in SRAM, and, if it is a read, in NVRAM.

3.3 Migration Policy

When the behavior of a dataset changes, the hybrid cache may contain a significant amount of data allocated based on the old behavior. This miss-allocation may result in lower-efficiency for future accesses that follow the new dataset behavior. However, there has been little consensus in previous

2. The final DB-AMB implementation tracks multiple datasets and uses a history window of 20 accesses.

work as to how aggressively this data should be migrated between SRAM and NVRAM caches or whether migration is even worth doing at all, due to the overhead of moving cache lines between the two arrays [11] [6] [10].

Also, migrations can be triggered by both legitimate changes in the data's behavior (phases) and by misclassifications of the allocation policy (errors). At best, migrations would be triggered only for the first reason, and in this situation it would concern only a small fraction of the cache lines that are inserted in the LLC. Despite being a small fraction of the LLC, these cache lines may be the target of numerous accesses (above average), and would actually generate a significant number of errors without migration.

Two approaches can be adopted for migration: pro-active migration, wherein all cache lines from a dataset whose placement policy has changed are moved when the policy changes, and lazy migration, wherein they are only moved upon access. For DB-AMB we adopt a lazy migration policy as it is simpler than having to search for all cache lines belonging to a given dataset, and avoids ping-pong effects for datasets whose policies change frequently.

3.4 Bypassing Policy

Much of the data in last-level caches is never reused and takes up space that could otherwise be used for storing useful data. Many approaches have been developed to *bypass* dead data to leave more space for live data and increase the overall hit rate [7], [16], [17], [18]. In the case of hybrid caches, however, bypassing becomes far more important as it not only leaves space for potentially live data, but can also eliminate the significant energy spent inserting data into the NVRAM.

We propose to adapt the technique developed by Sembrant et. al. [16] to learn whether a dataset is dead or alive, and appropriately bypass it. To do so we add a *reuse bit* to each cache line that is set when the line is reused. On each eviction, we look at the reuse bit to determine whether the line was dead. If it was not reused, we increment a per-dataset *dead counter*, and if it was reused, we decrement the counter. When the dead counter saturates, we determine that the dataset is dead, mark this in the datasets' policy, and bypass subsequent fetches of data from the dataset.

However, this approach relies on the dataset being installed in the cache to measure whether there is any reuse. This allows us to learn if a dataset transitions from live to dead, but once the dataset is marked as dead and is being bypassed, its cache lines will no longer be installed in the cache, and there is no way to learn if the dataset comes back to life. To address this we leverage *learning lines* [16], which takes a small fraction of all accesses and installs them in the cache regardless of the bypassing policy. By monitoring these cache lines, we can learn if a bypassed dataset is now seeing reuse in the cache and change policies.

The selected bypassing policy very cautiously enables bypassing (when the dead counter saturates) and aggressively disables bypassing (at the first reuse of a learning line). It is quite conservative as we want to avoid the overhead of increase LLC misses, but also because it affects the other policies (data allocation and migration policies) as accesses are not registered in the window of accesses of the dataset.

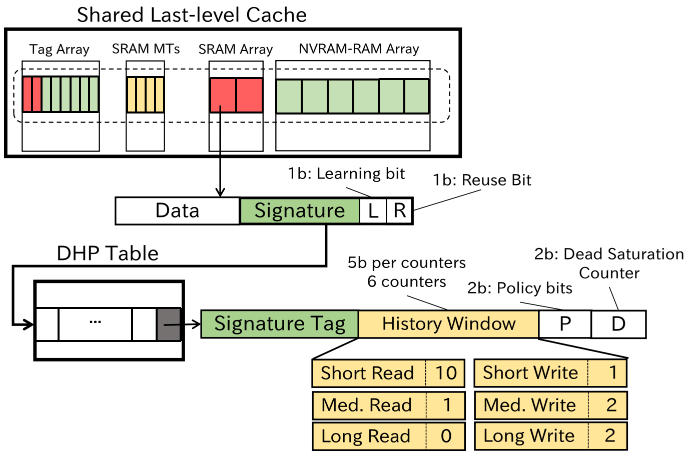


Fig. 5. Implementation. Each cache line is extended with a dataset *signature*, a *learning bit*, and a *reuse* bit. Upon access, the signature is used to access the *Dataset History and Policy Table* to update the history window based on the access type (reuse and read/write). The history window is evaluated periodically to set a new policy for the dataset.

The global allocation policy of a dataset is shown Fig.4. If the dataset is detected as dead by the bypassing policy, all blocks will be bypassed directly to the higher levels of memory. When the dataset is detected as alive, it has to be learned first by going through a window of accesses with a preemptive policy, then E_{SRAM} and E_{NVRAM} can be evaluated to determine the best allocation site between SRAM and NVRAM.

4 Design and Implementation

An overview of our DB-AMB solution is shown in Figure5. The hybrid cache is split between SRAM and NVRAM by ways (fewer for SRAM, more for NVRAM). Each cache line is extended with its PC-based signature, a reuse bit, and a learning bit. The PC-based signature is used to identify each cache line's dataset, which enables the lookup of the policy from a separate table. Counts of the classes of accesses over a short history window are updated on each access, and the policy is recomputed after each window.

4.1 The Dataset History and Policy Table

Each dataset's policy and access history is stored in the associative Dataset History and Policy Table (DHP Table). When installing cache lines in the cache, the cache line's dataset is used to lookup the appropriate allocation policy in the DHP Table.

For each access, the dataset is determined from the cache line's signature, and the history for that dataset is updated in the DHP Table based on the access type (read or write) and reuse (short, medium, or long). To learn the behavior of a dataset, these counters are reset and updated over a window of 20 accesses per dataset. After this history window, the policy is recomputed based on the new access counts from the history window.

The DHP Table needs to be sized to capture the important working sets of an application at any given time. The larger the table, the higher the overhead, but if the table is too small, it will have to re-learn datasets that have been

TABLE 1
Characteristics of the SRAM/STT-RAM hybrid cache simulated with 4-ways SRAM and 12-ways STT-MRAM simulated with NVSim [15] (22nm, 350K)

	SRAM	NVRAM	Hybrid
Read Latency (cycles)	15	15	15
Write Latency (cycles)	15	65	15/65
Read Energy (pJ)	25	25	25
Write Latency (pJ)	25	25	25/65
Static Power (mW)	2.83	0.56	1.10
Area (mm^2)	3.77	0.95	1.66

TABLE 2
Evaluated hybrid cache policies

Name	Technique
RWHCA	Read-Write aware Hybrid Cache Architecture [8]
PHC	Prediction Hybrid Cache with the profiled static threshold adjustment [4]
DB-A	Dataset-Based Allocation policy with no Migration and Bypassing
DB-AMB	Dataset-Based Allocation policy with Migration and Bypassing enabled

evicted from the table. A DHP Table with 256 entries is only 1% less energy-efficient overall compared to a perfect DHP Table. The metadata required is very small because usually benchmarks show a small number of used datasets at the same time.

4.2 Signature Storage

DB-AMB assumes that the cache line's signature is available when the LLC is accessed to be able to identify its dataset in the DHP Table. To avoid the overhead of storing the full PC, a hash can be used to reduce the number of bits. We observed that using only the 12 least significant bits achieves essentially the same precision (within 1%), consistent with previously reported results [4]. This reduces the signature field to 2.1% overhead.

The use of per-cache line signatures is a challenge in non-inclusive hierarchies, as we may have evictions from the L1 cache and need to know their signatures. Addressing this requires storing the signatures for cache lines in the L1, but as these are only used on eviction, they can be safely stored in a separate structure that is off the critical path for L1 accesses.

5 Methodology

5.1 Simulation

We evaluate DB-AMB for single- and four-core systems by using an in-house trace-driven simulator. We use Pin [19] to generate 5 instruction traces separated by 1 billion instructions for each PARSEC [14] benchmark, running with the medium input size. For each trace we warm the caches for 200 million instructions followed by 200 million instructions of measured execution. The simulated cache hierarchy includes a 32KB, 4-way set-associative private L1 instruction/data caches and a 2MB 16-way set-associative shared hybrid L2 cache. All caches have line sizes of 64 bytes. Cache coherence is managed by the MESI protocol without enforcing the inclusion property.

5.2 Design

Table 1 shows energy and delay characteristics of the hybrid cache at 22nm as modeled with NVSim [15]. An STT-RAM cell is used for the NVRAM array and Low Operating Power (LOP) devices are used for peripheral circuits and SRAM cells. The additional storage needed for the DB-AMB predictor including the DHP Table is also modeled. Upon every access, the DHP Table is accessed and the block needs to be accessed first in order to retrieve its signature for indexing the DHP Table. To limit this overhead, the DHP Table is modeled with High Performance (HP) devices and its tag and data arrays are accessed in parallel in 1 cycle. Each entry in the DHP Table consists of 6 counters to monitor the accesses to the window (6 counters of 5 bits), the dataset's allocation policy bits (2 bits) and the dead saturation counter (2 bits) for a total of 5 Bytes per entry. The DHP Table is designed as an LRU cache with 256 entries (128 sets times 2 ways) which leads to a 1.28kB cache.

5.3 Overhead

The extra hardware added for the DB-AMB policy are the DHP Table to store dataset information, a missing tags array to keep track of the last evicted tags for the SRAM array, and an extra-field per block to store the signature along with a learning and a reuse bit to each cache line. In our simulations, the DHP Table is a 2kB, 2-way LRU cache with 256 entries. The missing tag array for the SRAM is designed with 4 entries of 20 bits each per set, as adding more entries does not show significant improvements, which consumes 20.4kB storage. Storing the signature requires an extra 12 bits per cache line which leads to an extra 49kB capacity. The total capacity overhead is 72kB which is 3.4% of the capacity of the LLC. We evaluate the energy overheads of the different components using NVSim [15] and CACTI [20]. Accessing the DHP Table upon every access increases the geometric mean of the LLC dynamic energy consumption by 0.64% and adds a reduction in cycle time of only 0.46% as the DHP Table stays small and fast to access compared to the LLC. The missing tags array is accessed only upon misses and increases dynamic energy by 0.80%.

5.4 Baselines

The four hybrid cache allocation policies we compare are shown in Table 2. RWHCA [8] is a saturation counter based solution that allocates cache lines with a preemptive policy. RWHCA includes saturation counters on every line to detect and migrate write-intensive data to the SRAM array and read-intensive data to the NVRAM array. The saturation threshold is set to 3 after a design exploration to minimize the average energy consumption across PARSEC benchmarks. The PHC predictor proposed by Ahn et. al. [4] are the closest solution to this paper and performs the write intensity detection by detecting write-intensive miss-PC based datasets. The learning is performed at a cache line level, as each block is extended with a field that tracks the cost of the block during its lifetime in the LLC. Upon block's eviction, its cost is compared to a cost threshold to increment/decrement a saturation counter associated with the dataset. Block allocation is performed depending on the value of this saturation counter. The cost threshold can be tuned dynamically or profiled statically per application. The difference between the static and dynamic versions is reported to be negligible, and for the sake of implementation, in this evaluation, the static version is chosen and the cost threshold statically profiled per application in order to minimized the energy consumption. We present two versions of the DB-AMB predictor: the DB-A version that only include the datasets allocation policy, and DB-AMB, which adds migration and bypassing to DB-A in order to understand more precisely what those optimizations bring to the solution.

6 EVALUATION

6.1 Single-core Energy and Performance

The energy breakdown for the hybrid cache is shown in Figure 6-top. The reported migration energy is the energy associated with the extra accesses triggered by the migration policy. For performance we report Average Memory Access Time (AMAT) in Figure 6-midddle, as it indicates whether the DB-AMB policy is slowing down the cache by increasing latency through more NVRAM writes and LLC misses. The DRAM accesses per thousand instruction, Figure 6-bottom, indicates how much the hybrid cache misses and evicts cache lines. The baseline for each figure is the RWHCA predictor.

The PHC policy (second bar) is more conservative than the others with regard to the allocation of cache lines to the SRAM, and therefore pays more of an NVRAM write penalty in energy and performance. However, this leads to fewer misses, thereby reducing the number of DRAM accesses. Unfortunately, the increased latency of the NVRAM writes hurts performance for several of the benchmarks (*dedup*, *facesim*, *freqmine*, *swaptions*).

DB-A (third bar) is equivalent to RWHCA in terms of LLC energy but hurts less the miss rate, therefore reducing DRAM accesses by a geometric mean of 13.8% and reducing the AMAT by 2.0%. The limitation of DB-A is that while learning the behavior of the datasets, cache lines are allocated preemptively. Then, once the correct dataset behavior is learned, the allocation of the previously cached lines does not change (since DB-A does not include migration), but only newly brought cache lines will follow the allocation policy indicated for that dataset. Thus, the lines cached during the learning phase preemptively introduce a significant amount of errors, such as NVRAM writes and misses in the SRAM array. Addressing such issues, the DB-AMB yields a geometric mean of energy consumption reduction of 9.9% and achieves a reduction in AMAT of 7.9% compared to the RWHCA predictor while saving 27.3% of the DRAM dynamic energy. Generally speaking, DB-AMB is more aggressive at allocating write-intensive cache lines to the SRAM array, which results in a significant reduction of the NVRAM write energy from 11.3% for RWHCA to 2.4% for DB-AMB. As this allocation is performed taking reuse into account, DB-AMB is also able to reduce the DRAM accesses compared to the other predictors. Compared to DB-A, enabling migration and bypassing plays an important role in reducing the NVRAM write penalty in all benchmarks, as cache lines can be correctly reallocated.

6.2 Bypassing Impact

To investigate the effect of the bypass policy, Figure 7 shows the change in overall misses and hits (and their sources) as well as the proportion of dead cache lines residing in the cache, with DB-A and DB-AMB, respectively. The first bar for each benchmark shows the breakdown of hits and misses without bypassing, and the second with bypassing. The percentage of dead blocks in the cache is shown in black. When bypassing is employed, there are two additional cache behaviors that can be observed: bypassing errors and extra hits due to increased capacity from bypassing.

From this data we can see that bypassing successfully reduces the percentage of dead cache lines in the LLC for all applications from 23.5% with DB-A to 10.8% with DB-AMB. This directly translates into an energy reduction as the cost of fetching and writing the dead cache lines is eliminated.

Moreover, reducing the cache pollution due to dead blocks allows some benchmarks, such as *canneal*, *raytrace* and *streamcluster*, to also increase the reuse of blocks within the LLC as more space is available to the live blocks. In these 3 benchmarks, we can see a significant increase in the number of hits in the LLC compared to the DB-A policy, as reported by the extra hits. There is no general correlation between the percentage of dead cache lines that the DB-AMB is able to bypass and the increase of cache reuse, as this behavior is very application-specific.

To evaluate the efficiency of our bypassing scheme, we report the *bypass errors*, e.g., the misses caused by accesses to bypassed data, which would have been cache hits without bypassing. We can observe that the trade-off performed between the reduction in cache misses and the bypassed errors is always beneficial overall across these benchmarks. Moreover, even for applications such as *bodytrack* where our bypassing algorithm is not able to detect and bypass the dead blocks, it does not hurt the overall miss ratio.

6.3 Migration Impact

Migration allows to relocate cache lines when applications exhibit different phases with different write-intensity properties. It improves significantly the efficiency of the solution as blocks are systematically relocated based on the actual dataset allocation policy. However, the extra traffic due to migration leads to 1.9% energy overhead for DB-AMB. The Fig.8 shows the write requests distribution between the DB-A and DB-AMB policies. Due to the bypassing action, the total number of write requests performed by the LLC is reduced by a geometric mean of 10.3% between DB-A and DB-AMB, which brings small energy savings as writes account for only 20.4% of the total number of operations issued. However, the migration policy moves a significant fraction of the write traffic the SRAM array, the SRAM handling 83.5% of the total write traffic (geometric mean), compared to only 66.9% without migration. Furthermore, although the traffic in the SRAM array is increased, the SRAM miss rate is not affected, thanks to data reuse, as illustrated in figure 6-bottom.

Let us analyze *x264* as an example. Figure 7 shows that the number of dead blocks and LLC accesses remain almost constant with and without bypassing. However, figure 8 indicates that thanks to migration the number of writes

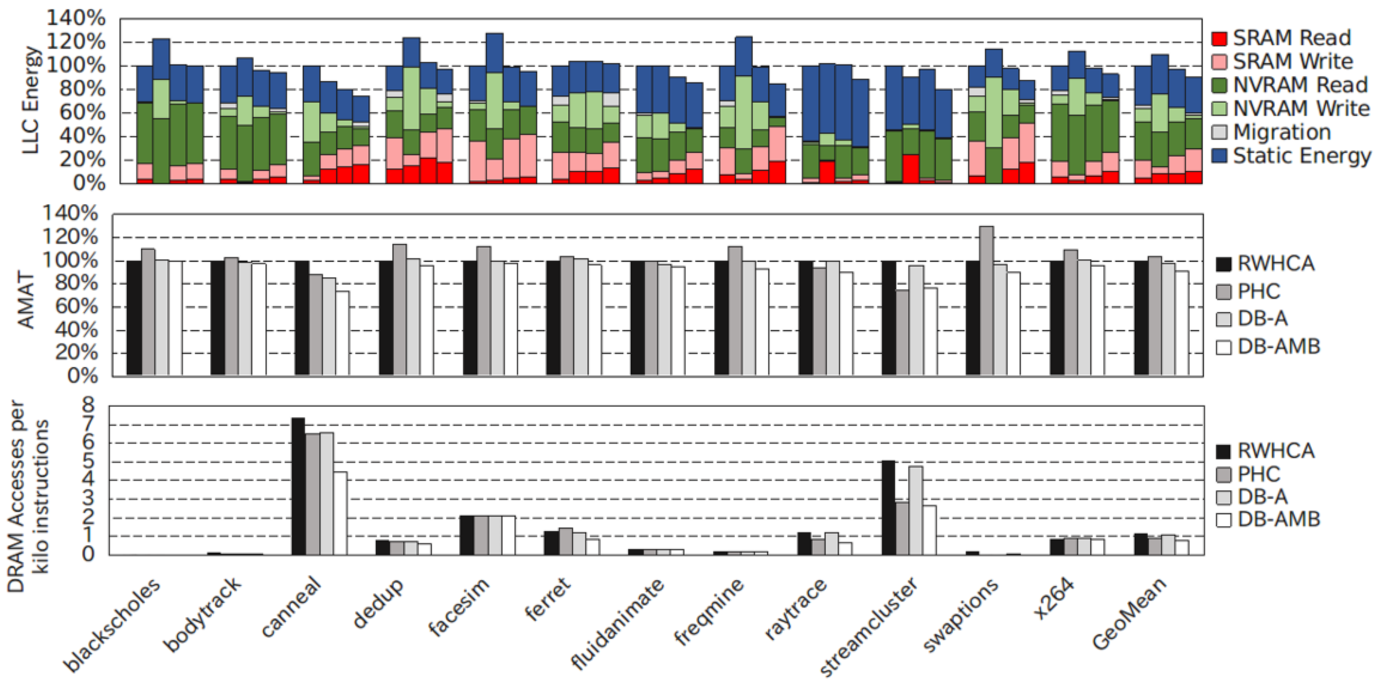


Fig. 6. Single-threaded results. Top: hybrid cache energy breakdown (left-to-right: RWHCA, PHC, DB-A, and DB-AMB) normalized to RWHCA. Middle: performance (Average Memory Access Time). Bottom: DRAM accesses per thousand instructions. Overall, DB-AMB significantly reduces the energy spent on NVRAM writes compared to existing techniques while not increasing misses, leading to a decrease in overall energy.

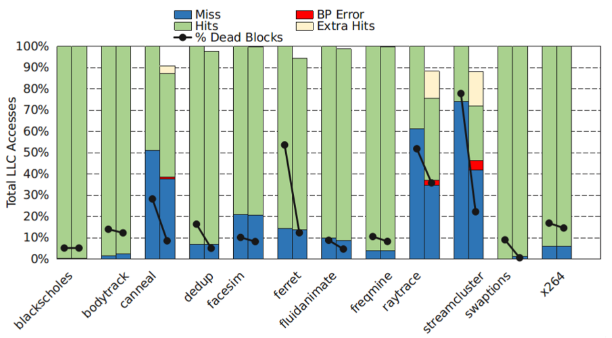


Fig. 7. Effects of bypassing (bars) and percentage of dead cache lines in the cache (lines) when enabling bypassing, DB-A (1st Bar) and DB-AMB (2nd Bar). Our bypassing policy decreases the number of accesses to the LLC without increasing the miss ratio. In several cases the additional space available due to bypassing actually increases the hit ratio over the baseline.

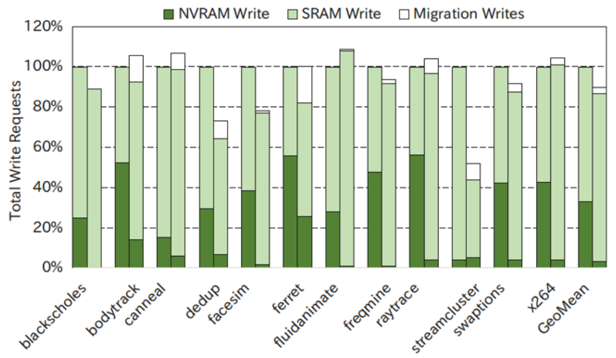


Fig. 8. Distribution of the write requests between DB-A (1st Bar) and DB-AMB (2nd bar). The bypassing policy reduces the total number of write requests issued by the LLC while the migration policy allows to perform most writes in the SRAM array reducing the NVRAM write penalty for a low extra-traffic overhead

issued to the SRAM increases from 54.7% to 92.1% between DB-A and DB-AMB while keeping a constant number of DRAM accesses (figure 6-bottom), which overall translates into 7.2% improvement in the LLC energy for *x264* as shown in figure 6-top.

6.4 Multi-core Energy and Performance

We evaluated the DB-AMB policy on a quad-core system using a simulation configuration similar to the single-threaded one. The parameters of RWHCA and static PHC have been calibrated to fit the new cache configuration. Figure 9 shows the hybrid cache energy, AMAT, and DRAM accesses per kilo-instruction for the multi-threaded workloads. The DB-AMB predictor reduces the AMAT by 6.5% (geometric mean) and the energy expenditure of the LLC by 7.7%, compared to the RWHCA predictor. This is a slightly smaller effect than observed with the single-threaded applications.

The conclusions are similar to the single-threaded case, as DB-AMB more aggressively allocates to SRAM while reducing the SRAM misses relative to RWHCA. The PHC predictor allocates more on the NVRAM side which increases the NVRAM write penalty but reduces the number of misses and therefore the DRAM accesses. For the multi-core workloads, a significant change is that the number of dead blocks is increased for all benchmarks, by 24.2% on average, allowing the DB-AMB policy to bypass more aggressively.

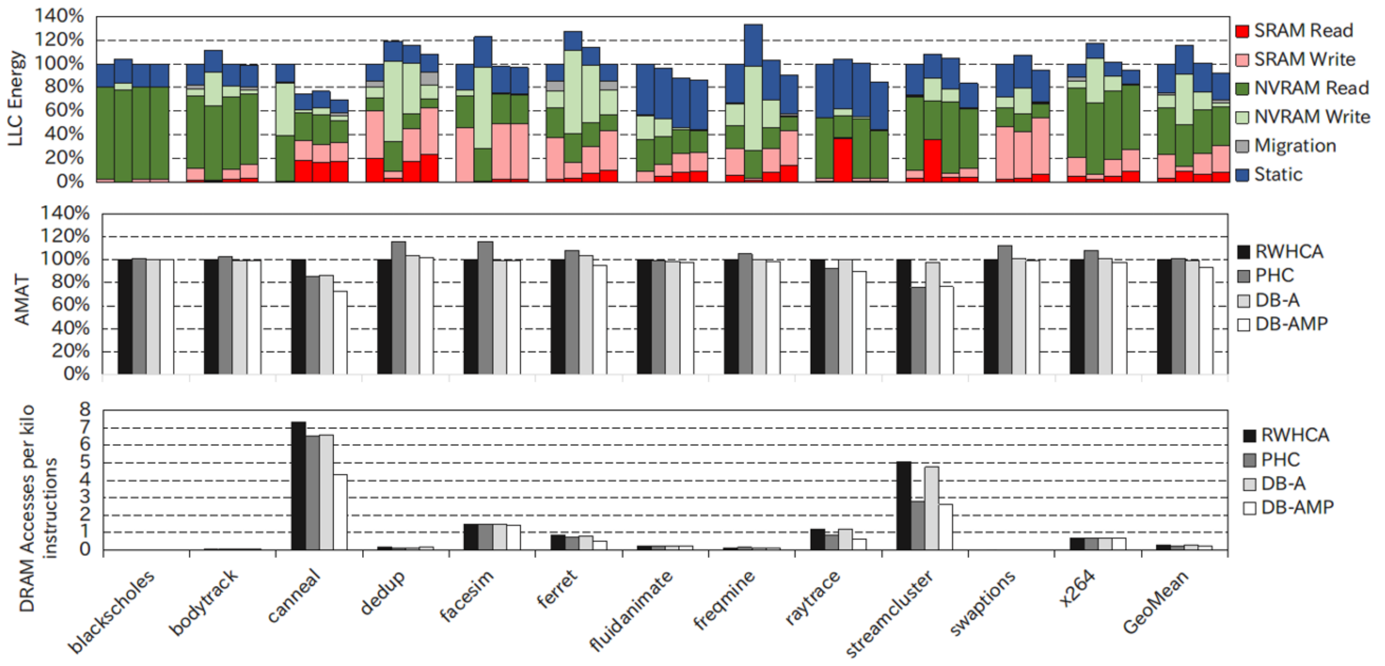


Fig. 9. Multi-threaded results. Top: hybrid cache energy breakdown (left-to-right: RWHCA, PHC, DB-A, and DB-AMB) normalized to RWHCA. Middle: performance (Average Memory Access Time). Bottom: DRAM accesses per thousand instructions. Overall savings are similar to those of single-threaded applications (Figure 6).

7 Related Work

Data Allocation for Hybrid Caches: Many approaches have been proposed to make the best use of NVRAM caches by trading off between performance, energy and endurance. The simplest approach, a direct replacement of SRAM with NVRAM, has been shown to be sub-optimal because of the write penalty overhead [21]. This has led to a significant interest in hybrid caches that combine SRAM and NVRAM, for which the data allocation policy is key. Most of the early approaches to hybrid cache allocation [6], [8], [10], [11], including the Read Write Hybrid Cache Architecture (RWHCA) detailed earlier, use counters to perform the write intensity prediction of cache lines. In these designs, the cache line is migrated when the counter saturates. Jadidi et al. [10] showed that write requests are distributed non-uniformly between sets, and even within sets, and proposed an allocation policy that maps the write-intensive cache lines to SRAM to avoid NVRAM writes. The remaining data blocks are remapped across the NVRAM array to distribute evenly the traffic and improve its endurance. The technique learns read/write behavior on a per-cache line basis and does not take reuse into account, which can lead to suboptimal placements. In contrast, DB-AMB leverages reuse information to optimize data placement, minimizing the LLC energy expenditure.

Compiler approaches have also been proposed to direct data allocation [9], [22]. For example, Chen et al. [9] proposed a hardware/software co-design where the compiler analyzes the read/write behavior of the data at loop-level granularity and guides the hardware to achieve the desired placement. The hardware updates the compiler hints based on runtime behavior. Li et al. [22] propose a compiler-assisted technique to reduce migration overhead by grouping write-intensive data in the same cache blocks in order to be allocated in the SRAM. However, predicting the write-intensity of cache lines in the LLC at compile-time has proven to be difficult, due to the filtering of accesses by the higher caches and the compiler being unaware of the effect of prefetches, write-backs, and coherence operations, all of which can cause LLC writes. DB-AMB builds on information collected at runtime, hence can easily capture the cache behavior inaccessible at compile-time.

The adaptive block placement and migration policy [7] (APM) exploits different classes of writes: prefetch, demand, and core writes. For each class, they design a specific allocation policy and add dead cache line bypassing for the NVRAM array. However, their classification does not show a clear distinction of classes of write accesses. These previous solutions learn the cache line behavior at the cache line granularity, which can require large amounts of metadata and a long time to learn the behavior of all cache lines. To alleviate this drawback, a dataset based classification of cache lines has been proposed by Anh et al. [4], exploiting the fact that applications tend to only have a limited number of active datasets at any given time. Their approach uses the signature of the cache line to predict its write intensity and exploits the correlation between signatures and write-intensity of cache lines to map them to the SRAM. In their design, the learning is also performed at a cache line level, as each block is extended with a field that tracks the energy cost of the block during its lifetime in the LLC. Upon a block eviction, its cost is used to increment/decrement the saturation counter of its corresponding signature (dataset) depending on its value. DB-AMB goes beyond this work by accounting for the *reuse* of the dataset in computing the energy cost function and performs learning at dataset granularity, which allows us to learn more quickly.

Reuse Prediction: Predicting the reuse of a cache line in standard caches has been studied for decades in the context of cache replacement policies [5], [23], [24]. Wu et al. [5] analyze different criteria to form datasets: memory regions (higher bits of data), signatures (the PC that triggered the miss), and hashing of the instruction sequence that accessed the cache line. They show that the miss-PC signature criteria performs better than the others for learning reuse. Jain et al. [24] also uses the miss PC of blocks to learn the reuse of blocks, however their predictor is not trained based on the last accesses issued by the dataset but by a sequence of accesses re-built based on the emulated Belady's optimal placement.

Dead Blocks Bypassing: Dead blocks predictors [13], [16], [17], [25] have targeted either bypassing blocks with zero reuse or detecting when a block becomes dead and then can be safely evicted. Lai et al. [13] hashes a sequence of memory-access PCs related to the cache block to produce an identifier used to predict the dead state of a block. The prefetch policy can avoid evicting live blocks. Sembrant et al. [16] use miss-PC based datasets to detect and bypass dead blocks on a per-level basis. Their monitoring is performed across the complete hierarchy so that datasets are only installed in cache levels where reuse is detected. Khan et al. [25] combines data-based and instruction-based dead block detections to build a higher level of confidence of the status (dead or alive) of the block. Kharbutli et al. [17] monitors each miss-PC dataset to rapidly detect when blocks become dead and therefore can be evicted by the replacement policy. Any of the bypass algorithms discussed here can be used for the DB-AMB policy as the bypassing policy is orthogonal to the allocation and migration policy, but the Sembrant et al. solution [16] uses the same idea of building datasets and allows a cheap implementation with the DHP Table.

8 Conclusion

Hybrid NVRAM/SRAM caches rely on intelligent data allocation policies to avoid the energy and latency penalties of writing to NVRAM and the increased miss ratio of over-allocating to the smaller SRAM. Our DB-AMB solution uses cache line signatures to identify datasets, and then applies allocation decisions for each cache line based on its dataset. By using datasets, DB-AMB can learn behaviors quickly and apply them to new cache lines. DB-AMB learns both the write-intensity of each dataset as well as its reuse properties. This learning is enabled by aggregating information across all the cache lines in the dataset. Based on this information, DB-AMB performs its data allocation using a cost function that optimizes for the dynamic energy consumption, but also includes a *migration policy* to adapt to application phases and *bypassing* to avoid installing dead data in the cache. DB-AMB outperforms previous policies by more aggressively allocating blocks to the SRAM array, which reduces the NVRAM write penalty significantly without increasing SRAM misses, as the allocation is performed taking into account the dataset reuse.

References

[1] S. Mittal, J. S. Vetter, and D. Li, "A survey of architectural approaches for managing embedded dram and non-volatile on-chip caches," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 6, pp. 1524–1537, June 2015.

[2] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "A durable and energy efficient main memory using phase change memory technology," *SIGARCH Comput. Archit. News*, vol. 37, no. 3, pp. 14–23, Jun. 2009.

[3] Y. B. Kim, S. R. Lee, D. Lee, C. B. Lee, M. Chang, J. H. Hur, M. J. Lee, G. S. Park, C. J. Kim, U. I. Chung, I. K. Yoo, and K. Kim, "Bi-layered rram with unlimited endurance and extremely uniform switching," in *2011 Symposium on VLSI Technology - Digest of Technical Papers*. New York, NY, USA: Elsevier, June 2011, pp. 52–53.

[4] J. Ahn, S. Yoo, and K. Choi, "Prediction hybrid cache: An energy-efficient stt-ram cache architecture," *IEEE Transactions on Computers*, vol. 65, no. 3, pp. 940–951, March 2016.

[5] 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 *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-44. New York, NY, USA: ACM, 2011, pp. 430–441.

[6] X. Wu, J. Li, L. Zhang, E. Speight, and Y. Xie, "Power and performance of read-write aware hybrid caches with non-volatile memories," in *2009 Design, Automation Test in Europe Conference Exhibition*, April 2009, pp. 737–742.

[7] Z. Wang, D. A. Jimnez, C. Xu, G. Sun, and Y. Xie, "Adaptive placement and migration policy for an stt-ram-based hybrid cache," in *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*. New York, NY, USA: IEEE, Feb 2014, pp. 13–24.

[8] X. Wu, J. Li, L. Zhang, E. Speight, R. Rajamony, and Y. Xie, "Hybrid cache architecture with disparate memory technologies," in *ACM SIGARCH computer architecture news*, vol. 37, ACM. New York, NY, USA: ACM, 2009.

[9] 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*, ser. ISLPED '12. New York, NY, USA: ACM, 2012.

[10] A. Jadidi, M. Arjomand, and H. Sarbazi-Azad, "High-endurance and performance-efficient design of hybrid cache architectures through adaptive line replacement," in *IEEE/ACM International Symposium on Low Power Electronics and Design*, Aug 2011, pp. 79–84.

[11] J. Li, C. J. Xue, and Y. Xu, "Stt-ram based energy-efficiency hybrid cache for cmps," in *2011 IEEE/IFIP 19th International Conference on VLSI and System-on-Chip*, Oct 2011, pp. 31–36.

[12] A. Sembrant, E. Hagersten, and D. Black-Schaffer, "Data placement across the cache hierarchy: Minimizing data movement with reuse-aware placement," in *2016 IEEE 34th International Conference on Computer Design (ICCD)*. New York, NY, USA: IEEE Press, Oct 2016, pp. 117–124.

[13] A.-C. Lai, C. Fide, and B. Falsafi, "Dead-block prediction amp; dead-block correlating prefetchers," in *Proceedings 28th Annual International Symposium on Computer Architecture*, 2001, pp. 144–154.

[14] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: Characterization and architectural implications," in *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, ACM. New York, NY, USA: Elsevier, 2008, pp. 72–81.

[15] X. Dong, C. Xu, N. Jouppi, and Y. Xie, "Nvsim: A circuit-level performance, energy, and area model for emerging non-volatile memory," in *Emerging Memory Technologies*. New York, NY, USA: Springer, 2014, pp. 15–50.

[16] A. Sembrant, E. Hagersten, and D. Black-Shaffer, "Tlc: A tag-less cache for reducing dynamic first level cache energy," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-46. New York, NY, USA: ACM, 2013, pp. 49–61. [Online]. Available: http://doi.acm.org/10.1145/2540708.2540714

[17] M. Kharbutli and Y. Solihin, "Counter-based cache replacement and bypassing algorithms," *IEEE Trans. Comput.*, vol. 57, no. 4, pp. 433–447, Apr. 2008. [Online]. Available: http://dx.doi.org/10.1109/TC.2007.70816

[18] H. Liu, M. Ferdman, J. Huh, and D. Burger, "Cache bursts: A new approach for eliminating dead blocks and increasing cache efficiency," in *Microarchitecture, 2008. MICRO-41. 2008 41st IEEE/ACM International Symposium on*. IEEE, 2008.

[19] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood, "Pin: building customized program analysis tools with dynamic instrumentation," in *Acm sigplan notices*, no. 6, ACM. New York, NY, USA: ACM, 2005, pp. 190–200.

[20] S. Thoziyoor, N. Muralimanohar, J. H. Ahn, and N. P. Jouppi, "Cacti 5.1," HP, Tech. Rep., 2008.

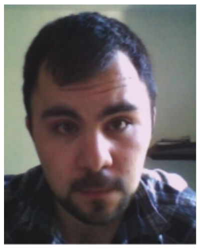
[21] Y. Xie, *Emerging Memory Technologies: Design, Architecture, and Applications*. New York, NY, USA: Springer Science & Business Media, 2013.

[22] Q. Li, J. Li, L. Shi, C. J. Xue, and Y. He, "Mac: Migration-aware compilation for stt-ram based hybrid cache in embedded systems," in *Proceedings of the 2012 ACM/IEEE International Symposium on Low Power Electronics and Design*, ser. ISLPED '12. New York, NY, USA: ACM, 2012, pp. 351–356.

[23] A. Jaleel, K. B. Theobald, S. C. Steely Jr, and J. Emer, "High performance cache replacement using re-reference interval prediction (rrip)," in *ACM SIGARCH Computer Architecture News*, vol. 38, no. 3. ACM, 2010, pp. 60–71.

[24] A. Jain and C. Lin, "Back to the future: Leveraging belady's algorithm for improved cache replacement," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, June 2016, pp. 78–89.

[25] S. M. Khan, Y. Tian, and D. A. Jimenez, "Sampling dead block prediction for last-level caches," in *2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, Dec 2010, pp. 175–186.



Vaumourin Gregory received his PhD in Electrical Engineering from the University of Bordeaux, France in 2016. He is a guest researcher in computer architecture in the Department of Information Technology at Uppsala University. His current research interest include memory system design and emerging memory technologies.

Alexandra Jimborean is Assistant Professor at Uppsala University since 2015. She obtained her PhD from the University of Strasbourg, France in 2012, was awarded the Anita Borg Memorial Scholarship offered by Google in recognition of excellent research, along with other 25 distinctions, awards and grants. Her research focuses on compile-time and run-time code analysis and optimization for performance and energy efficiency and on software-hardware co-designs.

David Black-Schaffer is a Professor in the Department of Information Technology at Uppsala University. His research focuses on approaches for moving data more efficiently in heterogeneous computer systems, using both software and hardware techniques. His results have led to multiple patents and a startup. Prior to joining the faculty of Uppsala University, he contributed to the design and development of the OpenCL standard for heterogeneous computation at Apple. He received his PhD in Electrical Engineering from Stanford University in 2008 for work on power-efficient embedded processing systems. Dr. Black-Schaffer is also an award-winning teacher, and his educational tools are used by tens of thousands of students.