2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture

LAP: Loop-Block Aware Inclusion Properties for Energy-Efficient Asymmetric Last Level Caches

Hsiang-Yun Cheng*¶, Jishen Zhao†, Jack Sampson*, Mary Jane Irwin*, Aamer Jaleel‡, Yu Lu§, and Yuan Xie¶
*Pennsylvania State University, †University of California, Santa Cruz, ‡NVIDIA Research,
§Qualcomm Research, ¶University of California, Santa Barbara
*{hoc5108, sampson, mji}@cse.psu.edu, †jishen.zhao@ucsc.edu, ‡ajaleel@nvidia.com,
§yulu@qti.qualcomm.com, ¶yuanxie@ece.ucsb.edu

***Abstract*—Emerging non-volatile memory (NVM) technologies, such as spin-transfer torque RAM (STT-RAM), are attractive options for replacing or augmenting SRAM in implementing last-level caches (LLCs). However, the asymmetric read/write energy and latency associated with NVM introduces new challenges in designing caches where, in contrast to SRAM, dynamic energy from write operations can be responsible for a larger fraction of total cache energy than leakage. These properties lead to the fact that no single traditional inclusion policy being dominant in terms of LLC energy consumption for asymmetric LLCs.**

We propose a novel selective inclusion policy, Loop-block-Aware Policy (*LAP*), to reduce energy consumption in LLCs with asymmetric read/write properties. In order to eliminate redundant writes to the LLC, LAP incorporates advantages from both non-inclusive and exclusive designs to selectively cache only part of upper-level data in the LLC. Results show that LAP outperforms other variants of selective inclusion policies and consumes 20% and 12% less energy than non-inclusive and exclusive STT-RAM-based LLCs, respectively. We extend LAP to a system with SRAM/STT-RAM hybrid LLCs to achieve energy-efficient data placement, reducing the energy consumption by 22% and 15% over non-inclusion and exclusion on average, with average-case performance improvements, small worst-case performance loss, and minimal hardware overheads.

I. Introduction

Advances in emerging technologies have made it increasingly plausible to consider employing SRAM alternatives in last-level caches (LLCs). From a comparative technology standpoint, SRAM caches have limited density and relatively high leakage currents, resulting in SRAM LLCs occupying as much as 50% of chip area [1, 2], and accounting for a significant fraction (15% of peak and 30% of standby) of on-chip power [3, 4]. Technologies such as spin-transfer torque RAM (STT-RAM) [5], phase change memory (PCM) [6, 7], and resistive RAM (R-RAM) [8–10], have been proposed as candidates for replacing or augmenting SRAM in LLCs. STT-RAM, in particular, due to its relatively high endurance among NVM technologies, is considered as a feasible replacement for SRAM [11, 12]. For example, recent studies show that replacing SRAM with STT-RAM could save 60% of LLC energy with less than 2% performance degradation [12].

Although NVM technologies can offer significant overall energy savings, a common feature across these technologies is substantial read-write asymmetry, with write operations taking longer time and more energy than read operations and often more time and energy than the equivalent write operations to an SRAM LLC. The range of this asymmetry varies significantly with both technology [6, 8, 13] and optimization goal (e.g. performance/robustness-oriented [11, 14] vs. density-oriented bit cells [15, 16]). However, a clear trend emerges across all of these proposed circuit-technology combinations: Unlike SRAM LLCs, dynamic energy consumption frequently comprises as large a portion of total LLC energy as leakage. Thus, many efforts to optimize the energy-efficiency of caches built with NVMs tend to focus on write-reduction strategies such as utilizing hybrid-technology caches [17–19] to avoid writes being served by the NVM portions of a cache and data-driven re-encoding and bit-level write-masking [20, 21] to limit the number of cells written per write operation.

This paper examines a novel angle for improving the efficiency of NVM-based LLCs, namely, *the redesign of inclusion properties and associated replacement policies to explicitly include write-energy asymmetry awareness and repeated migration awareness as a primary means for write reduction*. Our explorations for one particular NVM technology, STT-RAM, show that none of the commonly employed inclusivity models (inclusive [22], non-inclusive [23], or exclusive [24]) are dominant from an energy-efficiency perspective for technologies where dynamic write energy can exceed leakage energy. The lack of a single preferred policy persists even when several technology parameters are varied significantly, strongly tracks the degree of read/write asymmetry, and is noticeable even for low (2x) degrees of read/write asymmetry, implying that this lack of dominance is likely present in read/write asymmetric technologies beyond STT-RAM. Likewise, we will also show that dynamic inclusivity switching approaches designed for SRAM-based caches [25] do not fully capture opportunities for optimizing the energy consumption of NVM LLCs through inclusion management, necessitating the development of NVM-specific adaptive inclusion policies.

1063-6897/16 $31.00 © 2016 IEEE
DOI 10.1109/ISCA.2016.19

This lack of a dominant inclusion policy motivates us to propose a new selective inclusion policy for better energy-efficiency in LLCs built from STT-RAM or other technologies with read-write asymmetry. Distinct from prior SRAM techniques that switch between existing inclusion properties [25], our loop-block-aware (LAP) policy introduces a new inclusion model. LAP exploits the advantages of both non-inclusion and exclusion to selectively cache in the LLC only part of the upper-level data in order to eliminate redundant writes. LAP detects the frequently reused clean data (with small hardware overhead) and keeps a copy of this data in the LLC to avoid redundant insertions, while evicting other data earlier to increase available cache capacity. LAP is orthogonal to and compatible with data-driven bit-level write reducing schemes for NVMs [20, 21]. Furthermore, we show that LAP can be easily and successfully extended to hybrid SRAM/STT-RAM LLCs where placement within, as well as among, hierarchy levels impacts energy efficiency.

By effectively utilizing the cache capacity to reduce redundant writes, LAP outperforms other variants of selective inclusion policies, including one designed with write-awareness [26], and can provide up to 51% (average 20%) and 47% (average 12%) energy savings over non-inclusive and exclusive STT-RAM LLCs, respectively (73% and 61% over non-inclusive and exclusive SRAM caches, respectively). For hybrid SRAM/STT-RAM LLCs, energy consumption is reduced by 22% and 15% on average, compared to the non-inclusive and exclusive policies.

This paper offers the following contributions:

- We show that, among traditional inclusion properties, there is no dominant inclusion property in STT-RAM LLCs for any point within a wide range of STT technology assumptions. To the best of our knowledge, no previous study analyzes the energy efficiency of different inclusion properties in STT-RAM LLCs.
- We propose LAP, a novel selective inclusion policy to reduce the energy consumption of LLCs employing asymmetric read-write technologies. To reduce redundant writes, LAP selectively caches the frequently reused clean data in particular levels of the hierarchy. On average, LAP reduces LLC write traffic by 35% / 29% while improving throughput by 12% / 2% over non-inclusion and exclusion, respectively.
- We develop a variant of LAP for systems with hybrid SRAM/STT-RAM LLCs. We show that detection of frequently reused clean data can help to achieve more energy-efficient data placement in hybrid caches, and that the hardware overheads for doing so are small.
- Evaluations of our mechanisms with cycle-accurate simulations running both multi-programmed and multi-threaded workloads show that LAP provides significant energy savings over both non-inclusive and exclusive STT-RAM LLCs, as well as outperforming other dynamic inclusion-switching policies, with small worst-case throughput loss, average throughput improvements, and minimal hardware overhead.
- We perform a range of sensitivity studies showing that LAP continues to offer benefits for systems with different LLC sizes, number of cores, and LLCs built on different STT-RAM technologies. In particular, we show that the dominant parameter for predicting the utility of LAP for a given STT-RAM technology is simply the write/read energy ratio rather than any parameter distinct to spin-transfer torque devices. This indicates that our method should benefit all LLCs with asymmetric write/read energy, and energy savings can be predicted by the write/read energy ratio.

The rest of the paper is organized as follows. Section II provides background on STT-RAMs, analyzes the pros and cons of different inclusion properties in STT-RAM LLCs, and highlights the motivating data driving this research. The LAP model is explained in detail in Section III, while Section IV adapts the reuse-aware data placement to hybrid LLCs. Section V and Section VI describe evaluation methodology and results, followed by a summary of related work in Section VII. Finally, we conclude the paper in Section VIII.

II. Background and Motivation

In this section, we first overview the characteristics of the representative asymmetric memory technology (STT-RAMs) that we will use to evaluate inclusivity in asymmetric memories and discuss the write traffic to the LLC for previously proposed inclusion properties. We then use some motivating examples to illustrate that there is no dominant inclusion property - that the energy-efficiency of an inclusion property depends on the workload's working set size and the number of writes. We further show that the number of writes to the LLC can be reduced by eliminating redundant data insertions when various inclusion properties are applied.

A. STT-RAM

Unlike traditional SRAM technology that uses electric charges to store information, an STT-RAM [5] memory cell stores data in a magnetic tunnel junction (MTJ). Each MTJ is composed of two ferromagnetic layers (free and reference layers) and one tunnel barrier layer (MgO). The relative direction of the free and reference layer is used to represent a digital "0" or "1". Compared to conventional SRAM caches, STT-RAM caches provide higher density and lower leakage power, but higher write latency and write energy.

Table I compares characteristics, as modeled by CACTI [27] and NVSim [28], of a 2MB STT-RAM cache bank in 22nm technology with its SRAM counterpart. As shown in the table, the read latency and read energy of the STT-RAM is comparable to that of the SRAM. Moreover, the 3x higher density and 7x less leakage power consumption in the STT-RAM makes it suitable for replacing SRAMs to build large LLCs. However, a write operation in an STT-RAM cache consumes 8x more energy and takes 6x longer latency than an SRAM. Therefore, minimizing the impact of inefficient writes is critical for successful application of STT-RAM caches. While we utilize STT-RAM as our technology

	SRAM	STT-RAM
Area (mm^2)	1.65	0.62
Read latency (ns)	2.09	2.69
Write latency (ns)	1.73	10.91
Read energy (nJ/access)	0.072	0.133
Write energy (nJ/access)	0.056	0.436
Leakage power (mW)	50.736	7.108

Table I: Characteristic of 2MB SRAM and STT-RAM cache bank (22nm, temperature=350K). Based on CACTI [27] and NVSim [28].

of choice, as seen in Figure 23, we will show that the key indicator of sensitivity to our proposed inclusion-policy optimizations will be the abstracted write/read energy ratio of the technology considered, indicating that the proposed policies should be applicable to other asymmetric memory technologies as well.

B. Write Traffic in Inclusion Properties

Different inclusion properties lead to different write traffic at the LLC, and for LLCs implemented with asymmetric read/write technologies, crafting an energy optimized inclusion policy must carefully consider the write traffic. Figure 1 illustrates the data flow of three representative inclusion properties[1]: inclusive [22], non-inclusive [23] and exclusive [24]. The former two are used in many commercial processors [29, 30] and the latter in designs with large non-LLC capacity [31]. We focus on non-inclusive and exclusive LLCs in this paper[2]. In the non-inclusive LLC, writes come from the data filled into L3 on L3 misses and from the dirty victims from L2 evictions. Writes to the exclusive LLC come from the clean and dirty victims evicted by the L2 cache. Since the number of dirty victims evicted from the L2 is a function of the parameters of the L2 and the program behavior, it is not, in the absence of back-invalidation, affected by the L3 cache design. As a result, the difference in write traffic only comes from the number of LLC misses in the non-inclusive mode and the number of clean L2 victims in the exclusive mode.

Figure 1 also illustrates the following observations. A non-inclusive LLC forces the cache to hold most cache blocks that are also residing in the upper levels. On an L3 miss, the requested block is brought into both the L2 and L3 caches (Figure 1 (b)). On an L3 hit, the block is brought into the L2 and the duplicate copy remains in the L3. Whenever there is an L2 miss, one victim block will be evicted from the L2. A clean victim block will be silently dropped while a dirty victim block will be inserted into the L3. When a block is evicted from the L3 during L3 insertion, the same block in the upper level caches is not back-invalidated to avoid serious performance degradation [33]. In an exclusive LLC, only the data that is evicted from upper levels is stored. On

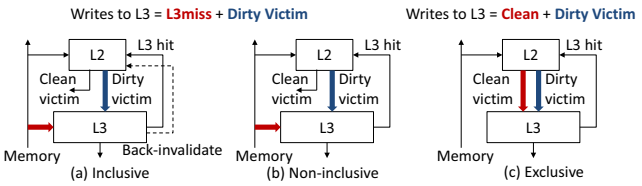


Figure 1: Block insertion/eviction flows for the (a) inclusive, (b) non-inclusive, and (c) exclusive LLC in a three-level cache hierarchy.

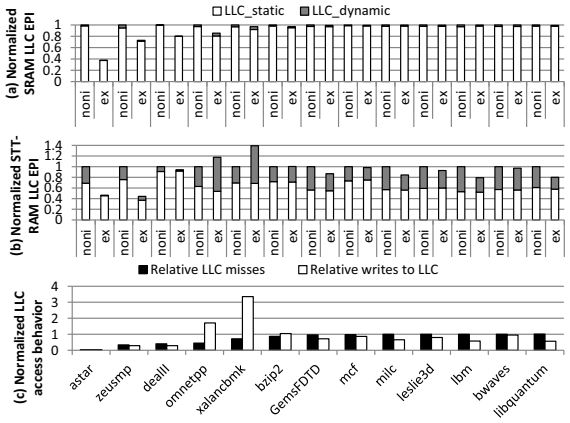


Figure 2: (a) LLC-energy/instruction (EPI) for the exclusive policy (ex) normalized to the non-inclusive policy (noni) in (a) SRAM and (b) STT-RAM LLC. (c) LLC misses and relative write traffic of the exclusive policy normalized to the non-inclusive policy.

an L3 miss, the requested block is inserted only into the L2 (Figure 1 (c)). On an L3 hit, the block is inserted into the L2 while the block in L3 is invalidated. Contrary to the non-inclusive LLC, an L2 eviction would always install the victim block into the L3 regardless of its dirty status.

Figure 2[3] shows that, from an energy perspective, while traditional SRAM LLCs appear to always benefit from the larger effective capacity provided by an exclusive policy (Figure 2 (a)), STT-RAM LLCs, which consume less leakage power but have higher write energy than SRAM LLCs [17, 34], may favor different inclusion properties depending on an application's behavior. *In short, there is no efficiency-dominant inclusion property for STT-RAM.*

Figure 2 (b) and (c) illustrate that the energy efficiency of the exclusive STT-RAM LLC compared to the non-inclusive STT-RAM LLC depends on the capacity benefit of exclusion, which can be captured by the relative LLC misses (M_{rel}), and the relative write traffic (W_{rel}). As shown in the figure, the exclusive LLC incurs fewer LLC misses due to its larger effective capacity and, thus, better performance, and lower static energy consumption than the non-inclusive LLC. However, the overall energy consumption is highly related to the number of writes. For example, *astar*, *zeusmp*, and *libquantum* are more energy-efficient in the exclusive STT-RAM LLC, as the relative number of writes is lower. On the other hand, *omnetpp* and *xalancbmk* are less energy-efficient

[1]Since STT-RAMs are mainly used to implement the shared LLC, we assume that the L2 is non-inclusive, as in Intel processors [29], and target only the inclusion property between L2 and L3.

[2]In this paper, we will not consider further using a strictly inclusive LLC, as industry is trending towards non-inclusive and exclusive cache designs [31, 32], and because bypassing redundant writes is not possible when strict inclusion is enforced.

[3]Results are evaluated on a 4-core system with an 8MB shared L3, running duplicate copies of SPEC2006 benchmarks. Detailed system configuration is described in Section V.

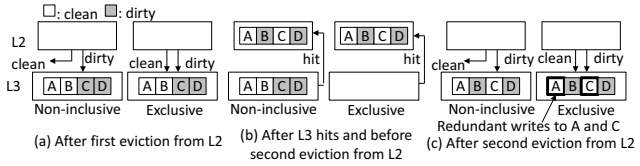


Figure 3: Illustration of redundant clean data insertion.

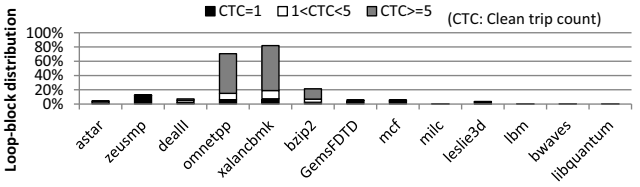


Figure 4: Loop-block distribution in different workloads.

in the exclusive STT-RAM LLC, due to the higher number of clean L2 victims in the exclusive mode compared to number of LLC data-fills in the non-inclusive mode.

C. Redundant Data Insertion

The number of writes to the LLC highly depends on the cache access behavior and the inclusion property. In this subsection, we show that the number of writes to the exclusive LLC can be reduced by eliminating redundant insertions of frequently reused clean data. Moreover, portions of LLC data-fills in the non-inclusive LLC are redundant and can be bypassed. When crafting an inclusion property for asymmetric LLCs, eliminating these redundant insertions is a source of significant energy savings.

1) Redundant Clean Data Insertion:

The exclusive LLC provides larger effective capacity by invalidating the duplicate upper-level data during LLC hits, but the clean data are redundantly inserted into the LLC during eviction from upper-level caches, as illustrated in Figure 3. Cache blocks A to D represent four different types of access behavior. During its initial life in L2, A and B are clean, while C and D are dirty. After their first eviction from L2, as illustrated in Figure 3 (a), C and D are updated in the non-inclusive LLC, and all four cache blocks are inserted into the exclusive LLC. Suppose that all four cache blocks hit and are brought back into L2, as shown in Figure 3 (b). The cache blocks in the exclusive LLC are invalidated giving larger effective capacity, while the data are maintained in the non-inclusive LLC. Figure 3 (c) illustrates that after their second eviction from L2, the two now dirty blocks, B and D, are written to the non-inclusive LLC, while all four cache blocks are inserted into the exclusive LLC. Compared to the non-inclusive LLC, the exclusive LLC needs two additional writes (insertions of clean data A and C) due to their invalidation during LLC hits. We define these cache blocks that are not modified during their travel between the upper-level caches and the LLC, causing redundant clean data insertions in the exclusive LLC, as *loop-blocks*. By keeping these loop-blocks in the LLC and evicting non-loop-blocks, we can more effectively utilize the cache capacity to reduce the number of writes.

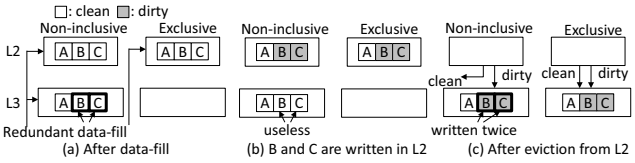


Figure 5: Illustration of redundant LLC data-fill.

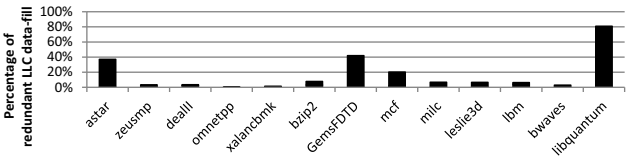


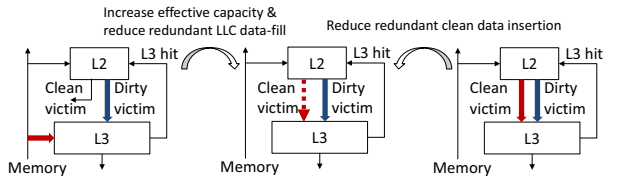
Figure 6: Distribution of redundant LLC data-fill.

Figure 4 shows the distribution of loop-blocks in SPEC2006 workloads. We observe that *omnetpp*, *xalancbmk*, and *bzip2* are the three workloads with higher than 20% loop-blocks. The number of loop-blocks in *omnetpp* and *xalancbmk* are even higher than 60%. These three workloads have a set of frequently-read data whose working set size is larger than L2 but smaller than the LLC. A large number of loop-blocks implies a larger opportunity to reduce redundant writes and, thus, dynamic energy consumption. In Figure 4, we also show the clean trip count (CTC) of loop-blocks to gain insight on how to predict loop-blocks that would experience redundant writes if they are invalidated during LLC hits. CTC represents the number of clean trips experienced by a loop-block between L2 and the LLC before it becomes a non-loop-block. We observe that the majority of loop blocks have CTC$\geq$5 which indicates that a block with consecutive clean evictions is highly likely to continue its clean loop between L2 and the LLC.

2) Redundant LLC Data-Fill:

In non-inclusive LLCs, the requested block is brought into both the L2 and the LLC on a L3 miss. Not all of these data-fills are useful, and the useless data-fills incur additional writes to the LLC, as illustrated in Figure 5. Suppose that cache blocks A, B, and C are brought into L2 and the LLC, and cache blocks B and C are written during their first lifetime in L2, as shown in Figure 5 (a) and (b). Since the cache blocks B and C are modified in L2, the duplicate copies in the LLC are now useless. After these cache blocks are evicted from L2, the clean data A is discarded in the non-inclusive LLC, while the dirty data B and C are written back to the LLC, as illustrated in Figure 5 (c). The data-fill of B and C in the non-inclusive LLC, as shown in Figure 5 (a), are redundant, because these data are not reused before being modified. Thus, the non-inclusive LLC experiences an additional two writes compared to the exclusive LLC.

Figure 6 shows the distribution of redundant LLC data-fills, i.e., the number of redundant LLC data-fills divided by the total number of LLC data-fills, in different SPEC2006 workloads. From the result, we observe that *astar*, *GemsFDTD*, *mcf*, and *libquantum* have a large number of redundant LLC data-fills when the non-inclusive policy is



(a) Non-inclusive	(b) LAP	(c) Exclusive
✓ No redundant clean insertion	✓ Insert only non-duplicate data	➢ Redundant clean insertion
➢ Useless duplicate data	✓ Duplicate only useful clean data	✓ No duplicate data
➢ Redundant LLC data-fill	✓ No redundant LLC data-fill	✓ No redundant LLC data-fill

✓ : pros ➢ : cons

Figure 7: Pros and cons of non-inclusive and exclusive LLC implementations, and an overview of LAP.

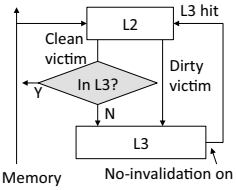
used. The percentage of redundant LLC data-fill at *libquantum* is even higher than 80%. By inserting data only into upper level caches on LLC misses, as in the exclusive LLC, all the redundant LLC data-fills can be eliminated.

III. LAP: Loop-Block-Aware Policy

As shown, no existing inclusion property is dominant in terms of energy efficiency for STT-RAM LLCs. One strategy is to attempt to dynamically switch among traditional inclusion properties to select the more energy efficient one for the current program phases, similar to the idea of the selective inclusion approach in SRAM [25]. However, this class of switching solutions misses some energy saving opportunities because both non-inclusion and exclusion incur their own type of redundant writes that cannot be eliminated simply by switching to the other one.

An energy-efficient inclusion property for asymmetric read/write LLCs should (1) trade off some LLC capacity for reducing the number of writes to the LLC, and (2) eliminate redundant clean data insertions and LLC data-fills. Following these two design principles, we propose a novel loop-block-aware policy (*LAP*) for LLCs built from STT-RAM or other technologies with asymmetric write and read energy. LAP discovers more energy saving opportunities than switching policies by selectively caching only part of the upper-level data in the LLC to reduce redundant writes.

By exploiting the desirable properties of both non-inclusive and exclusive policies, LAP aims to (1) reduce redundant insertion of frequently reused clean data, i.e., loop-blocks, by storing duplicate copies in the LLC, (2) increase available capacity by evicting less useful non-loop-blocks earlier, and (3) eliminate redundant LLC data-fill by inserting data into only upper level caches on LLC misses. Figure 7 shows the pros and cons of different policies, and an overview of LAP. Non-inclusive LLCs sacrifice cache capacity to store all the duplicate data, thus eliminating redundant clean data insertion. On the other hand, exclusive LLCs provide large cache capacity and eliminate redundant LLC data-fill, but incur redundant insertion of clean victims from upper level caches. In the following subsections, we describe in detail the selective inclusion of LAP, the loop-block-aware replacement policy, and how to identify loop-blocks with minimal hardware overheads.



	Invalidate on hit	L3 fill	Clean Writeback
noni	N	Y	N
ex	Y	N	Y
LAP	N	N	Y if not in L3

Figure 8: Data flow of LAP and the comparison to the non-inclusive (noni) and exclusive (ex) policy.

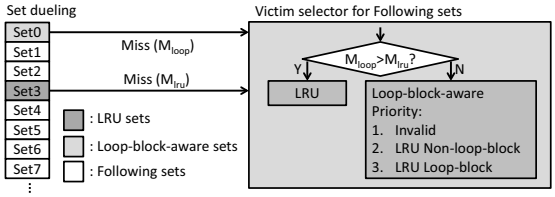


Figure 9: Loop-block-aware replacement policy.

A. Selective Inclusion Data Flow

Figure 8 shows the data flow of LAP in comparison to the non-inclusive and exclusive policies. As discussed in Section II-C1, if loop-blocks that experienced consecutive reads are invalidated during LLC hits in an exclusive LLC, there would be redundant writes compared to a non-inclusive LLC. Thus, LAP does not invalidate data on LLC hits. Although a non-inclusive LLC keeps a duplicate copy of all the upper-level blocks to reduce redundant insertion of loop-blocks, not all the duplicate data are useful, as the non-loop-blocks may not be reused later or would be modified anyway at their next eviction from the upper-level caches. Therefore, LAP does not insert data into the LLC during LLC misses, but instead relies on the no-invalidation policy during LLC hits and the replacement policy to keep loop-blocks in the LLC. Since each loop-block may have a duplicate copy in the LLC, only the clean victims that do not have a duplicate copy need to be inserted into the LLC during L2 evictions. Therefore, the number of writes to the LLC reduces to the number of exclusive clean victims plus the dirty victims from L2. By changing the data flow, LAP can lazily invalidate non-loop-blocks for larger LLC capacity, and keep as many loop-blocks as possible in the LLC to reduce redundant insertions of clean data.

B. Loop-Block-Aware Replacement Policy

To avoid write-intensive non-loop-blocks occupying LLC capacity, our loop-block-aware replacement policy gives loop-blocks higher priority to stay in the LLC, unless evicting non-loop-blocks earlier incurs more LLC misses than the loop-block-agnostic baseline replacement policy. This principle can be easily applied to any baseline policy. In this paper, we illustrate the design of loop-block-aware LRU, as shown in Figure 9. We use set-dueling [35] to avoid incurring high numbers of LLC misses and degrading performance. When selecting the replacement victim from a loop-block-aware sampling set, we first select an invalid block. If there is no invalid block, we then select the LRU non-loop-block. A loop-block would be evicted only when

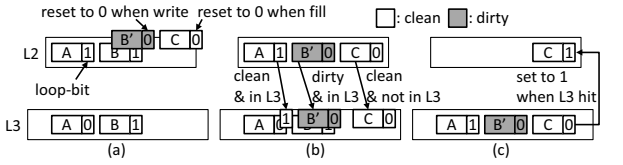


Figure 10: The update of loop-bits during (a) L2 insertions, L2 modifications, (b) L2 evictions, and (c) L3 hits.

there is no non-loop-block in the cache set. The LRU sampling sets follow the baseline LRU replacement policy. In our evaluation, we dedicate $1/64$ sets as loop-block-aware sets, and another $1/64$ sets as LRU sets. The number of LLC misses in these sampling sets are compared at the end of every 10M cycles, and the other sets follow the policy that incurs fewer LLC misses. By using set-dueling and following the loop-block-aware principle, we not only can keep as many loop blocks as possible in the LLC to reduce redundant writes, but also can avoid incurring performance degradation when evicting non-loop-blocks earlier is not beneficial for a particular phase or program.

C. Loop-Block Identification

We propose a low-overhead approach to identify loop-blocks, as illustrated in Figure 10. Based on our observation in Section II-C1, a loop-block is highly likely to continue its clean trips between L2 and L3. Therefore, we can add only one bit, called loop-bit, per cache block in both L2 and L3 to record whether the cache block has experienced a clean trip and predict whether it will incur redundant writes if it is invalidated on LLC hits. When the data is inserted into the L2 from the main memory, the loop-bit is reset to zero, as shown in Figure 10 (a). If the block is written, the loop-bit would also be reset to zero as it becomes dirty. When a cache block is evicted from L2, it first checks whether there is a copy in the LLC. If so, the clean data would be discarded but the loop-bit stored in the SRAM tag array would be updated, as illustrated in Figure 10 (b). The dirty data and its loop-bit would both be updated. If there is no duplicate copy in the LLC, the clean block and its loop-bit would be inserted into the LLC. When there is a LLC hit, the block would not be invalidated, as shown in Figure 10 (c). To indicate that this block may experience consecutive clean trips if it is not modified during its following life in the L2, the loop-bit of the inserted L2 block on the LLC hit would be set to one. Our replacement policy then uses the loop-bit information to keep as many loop-blocks (loop-bit=1) as possible in the LLC to reduce redundant clean insertions.

D. Summary

We propose a new inclusion model and associated replacement policy to reduce redundant writes by exploiting the loop-block characteristic, which is not discovered in prior selective inclusion [25] and deadblock bypassing [34, 36] approaches. The proposed LAP requires only small hardware overhead and design changes. We need one loop-bit per L2 and L3 cache block, and the overhead is negligible compared

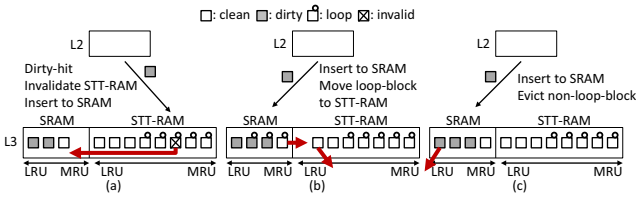


Figure 11: Data placement policy in hybrid SRAM/STT-RAM LLC: (a) dirty-hit to STT-RAM, (b) loop-block exists in SRAM, and (c) no loop-blocks in SRAM.

to the 64B cache block size. For the loop-block-aware replacement policy, we only need two miss counters for the entire cache and a simple comparator to decide the winning replacement policy. All the required data flows already exist in traditional exclusive and non-inclusive caches, so our loop-based approaches simply leverage these pre-existing data paths.

IV. LHYBRID: LAP FOR HYBRID-LLCS

Recent studies have proposed hybrid SRAM/STT-RAM LLCs to make use of the high density and low leakage power of STT-RAM and maintain low write overhead using SRAM [17–19]. LAP can also be applied to such a hybrid LLC architecture, and helps to save energy by reducing redundant writes to the STT-RAM portion of the LLC. Moreover, based on the characteristic of loop-blocks, we design a novel data placement policy (Lhybrid) for the hybrid LLC to further reduce the energy consumption. Since loop-blocks would not be modified at their following evictions from upper-level caches, our Lhybrid policy aims to store as many loop-blocks as possible in the STT-RAM portion. The non-loop-blocks that need to be frequently updated are given higher priority to be stored in the SRAM portion.

Figure 11 shows how the Lhybrid policy is designed to keep most of the loop-blocks in STT-RAM while most of the non-loop-blocks are stored in SRAM, assuming the baseline replacement policy is LRU. If the victim from L2 is dirty and it hits a block in STT-RAM, as shown in Figure 11 (a), our Lhybrid policy would invalidate the copy in STT-RAM and insert the dirty block in SRAM to reduce the number of writes in STT-RAM. If the victim from L2 does not have a duplicate copy in the L3, it would also be inserted into the SRAM portion of the LLC. When inserting a block to SRAM, the replacement policy would first select an invalid block from SRAM. If there is no invalid block in SRAM, one block in SRAM would be migrated to STT-RAM or evicted from the L3. If the incoming block is a loop-block or there are loop-blocks in SRAM, the MRU loop-block would be migrated to STT-RAM, as illustrated in Figure 11 (b). If there is an invalid entry in STT-RAM, the migrated block would replace the invalid block. Otherwise, an LRU non-loop-block would be evicted from STT-RAM, or an LRU loop-block would be evicted if there is no non-loop-block in STT-RAM. On the other hand, if there are no loop-blocks in SRAM, including the incoming block, the LRU block in SRAM would be selected as the replacement victim, as

Cores	3GHz, OoO, issue-width=4, 192 ROB, 32 LSQ
L1 I&D	private, 32KB per core; 4-way LRU; 64B cache block; write-back; 2-cycle read/write
L2	private, 512KB per core; 8-way LRU; 64B cache block; write-back; 4-cycle read/write
L3	shared, 8MB; 16-way; 4 banks; 64B cache block; write-back; write-allocate
	hybrid: 2MB SRAM (4-way) + 6MB STTRAM (12-way)
	SRAM: 8-cycle read, 8-cycle write tag: leakage = 17.73 mW, dynamic = 0.015 nj/access data: leakage = 202.94 mW, r|w energy = 0.072|0.056 nj/access
	STT-RAM: 8-cycle read; 33-cycle write data: leakage = 28.41 mW, r|w energy = 0.133|0.436 nj/access
Memory	4GB DRAM; DDR3-1600-x64

Table II: System configuration.

Name	benchmarks	Name	benchmarks
WL1	zeusmp,leslie3d,omn,dealII	WH1	omn,xalan,zeusmp,lib
WL2	lbm,xalan,lib,Gems	WH2	milc,omn,bzip2,xalan
WL3	Gems,Gems,Gems,mcf	WH3	omn,omn,dealII,leslie3d
WL4	milc,lib,leslie3d,bwaves	WH4	mcf,omn,leslie3d,xalan
WL5	bzip2,xalan,Gems,Gems	WH5	xalan,xalan,xalan,bzip2

Table III: Selected workload mixes of SPEC CPU2006. WL/WH: workloads with fewer/more writes in the exclusion than non-inclusion. (xalan:xalancbmk; omn:omnetpp; GemsFDTD:Gems; libquantum:lib)

shown in Figure 11 (c). Our data placement principle can also be combined with other replacement policies, such as RRIP [37]. Selecting an LRU block is just like selecting a block with *distant* re-reference interval in RRIP, while selecting an MRU block is just like selecting a block with *immediate* re-reference interval.

V. Experimental Setup

We evaluate our selective inclusion policies by modifying the gem5 simulator [38]. The cache access latency and energy are modeled by NVSim [28], with parameters listed in Table I. Table II shows our baseline configuration for the CMP system. We use a set of SPEC CPU2006 benchmarks for multi-programmed workloads and PARSEC benchmarks for multi-threaded workloads. We randomly choose 50 combinations of SPEC CPU2006 benchmarks, sorted by the number of writes in the exclusive LLC normalized to the non-inclusive LLC, as multi-programmed workloads. Among these 50 workloads, 10 workloads with diverse write behavior, as noted in Table III, are selected. We report the results from all 50 workload mixes, but analyze these 10 representative workloads in detail. For the SPEC benchmarks, we fast forward 6 billion instructions in atomic mode, and run in detailed mode for 2 billion cycles. When reporting performance and energy results, we evaluate throughput ($\sum IPC_i$) and LLC energy per instruction (EPI). For the PARSEC benchmarks, we run the simulation starting from the Region of Interest (ROI) [39], using the simlarge input set and report latency and total LLC energy.

To evaluate energy-efficiency, we compare LAP against the policies listed in Table IV. Since neither non-inclusion nor exclusion is dominant in terms of energy efficiency, a straightforward way to build a dynamic solution is to dynamically select between one of these two traditional inclusion properties. We compare LAP with two such policies, *FLEXclusion* and Dswitch. *FLEXclusion* [25] is a prior

Non-inclusive	The baseline inclusion property.
Exclusive	Exclusive policy that is widely used in comercial products.
FLEXclusion	Dynamically selects between non-inclusion and exclusion according to capacity demands and bandwidth consumption.
Dswitch	Dynamically selects between non-inclusion and exclusion according to capacity demands and writes to the LLC.
LAP-LRU	LAP but always use LRU as the replacement policy.
LAP-Loop	LAP but always evict non-loop-blocks earlier.
LAP	LAP that uses set-dueling to decide the replacement policy.
Lhybrid	LAP with loop-block-aware data placement for hybrid LLC.

Table IV: Evaluated Policies.

SRAM-based approach, which dynamically configures the LLC for non-inclusion or exclusion to improve performance and reduce bandwidth consumption. Since *FLEXclusion* does not consider the impact of asymmetric write energy when deciding the inclusion mode, we also compare with Dswitch [26], an alternative approach that also switches between non-inclusion and exclusion mode but considers the number of writes to the LLC as a decision input. We also evaluate variants of our LAP with different replacement policies, including LAP-LRU and LAP-Loop.

VI. Experimental Results

In this section, we evaluate the energy-efficiency of existing inclusion properties and compare them against both LAP and two variants of selective inclusion policies. We consider both multi-programmed and multi-threaded workloads and perform sensitivity studies of different system configurations. Finally, we extend LAP to hybrid-technology LLCs.

A. Non-inclusion vs. Exclusion

Figure 12 shows the energy consumption of the exclusive policy normalized to the non-inclusive policy in the SRAM and STT-RAM LLC. Consistent with our motivational studies, in the SRAM LLC, the energy consumption is dominated by the leakage and the exclusive policy incurs fewer LLC misses and consumes less energy than the non-inclusive policy by providing larger effective capacity in all workloads. For STT-RAM LLC, neither exclusion nor non-inclusion is dominant in terms of energy consumption reduction, as shown in Figure 12 (c) and (d).

Figure 13 shows workload diversity. Workloads that favor exclusion are separated from workloads that favor non-inclusion by a borderline, and the slope of the borderline is -0.8: As the relative writes for exclusivity increase, the exclusive LLC becomes less energy-efficient. For WL workloads with fewer writes in the exclusive LLC, the exclusive policy is 18% more energy-efficient than the non-inclusive policy on average, since the exclusive LLC benefits from both larger effective capacity and fewer writes. However, exclusion is less energy-efficient than non-inclusion in WH workloads, when the drawback of a large number of redundant data insertions outweighs the benefit of large effective capacity. For workloads, such as WH1, with significant capacity benefit from exclusivity, non-inclusion consumes slightly more energy than exclusion. On average, the exclusive policy is 12% less energy-efficient than the non-inclusive policy for WH workloads.

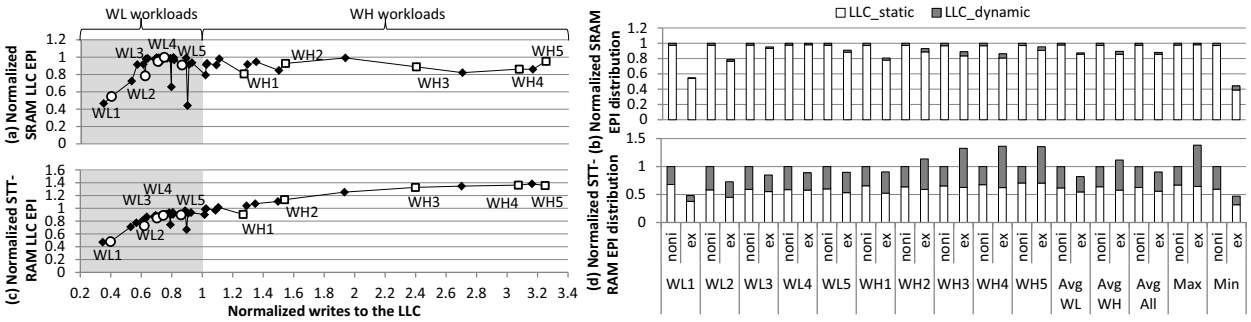


Figure 12: EPI of the exclusive policy (ex) normalized to the non-inclusive policy (noni) in the (a) SRAM and (c) STT-RAM LLC. The energy breakdown of sampled workloads are shown in (b) and (d). WL/WH workloads: fewer/more writes in the exclusive policy.

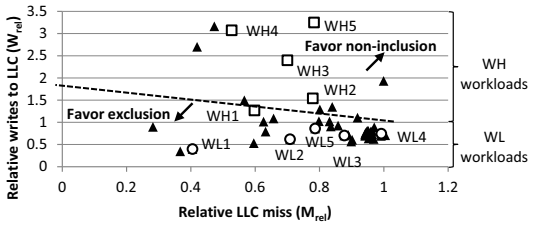


Figure 13: Workload characteristics: Relative misses and write traffic of the exclusive LLC compared to the non-inclusive LLC.

B. Multi-programmed Workloads

In this subsection, we provide detailed analysis of the selected inclusion policies for multi-programmed workloads, focusing on the reduction of redundant data insertion and the effective capacity. We then analyze the energy efficiency of variants of LAP with different replacement policies.

Energy-Efficiency. Figure 14 (a) shows the energy efficiency of different inclusion properties. Since *FLEXclusion* [25] does not consider the write traffic and the high write energy in STT-RAM LLC, it sometimes selects the less energy-efficient inclusion property at program phases and overall consumes more energy than exclusion on average. Dswitch [26] reduces the energy consumption of STT-RAM LLC by dynamically selecting the better inclusion property, leading to 10% and 2% energy reduction on average over non-inclusion and exclusion. By effectively utilizing cache capacity to reduce redundant writes to the LLC, LAP is more energy efficient for all workloads, and can reduce the energy consumption by 20% and 12% on average (up to 51% and 47%) compared to non-inclusion and exclusion. For WL workloads, the energy savings of LAP over non-inclusion increases as the relative writes decrease. On the other hand, the energy savings of LAP over exclusion increases as the relative writes increases for WH workloads. Energy reductions mainly come from decreased dynamic energy consumption, as shown in Figure 14 (b).

Performance. We also evaluate the performance of different inclusion properties, as illustrated in Figure 14 (c). On average, the exclusive LLC provides 12% better performance than the non-inclusive LLC by increasing effective capacity. Since *FLEXclusion* is optimized for better performance, it provides performance similar to exclusion. In WH workloads, Dswitch selects the non-inclusion mode, which consumes less energy but provides worse performance, leading to 4% performance loss on average over exclusion. LAP sacrifices cache capacity to reduce write counts, and thus performs slightly worse than the exclusive LLC in some workloads with large capacity demands. Nevertheless, for some workloads, LAP performs better by reducing the number of long-latency writes and keeping frequently reused loop-blocks in the LLC, as indicated by arrows in Figure 14 (c). As a result, LAP can perform slightly better than exclusion, with 2% performance speedup on average.

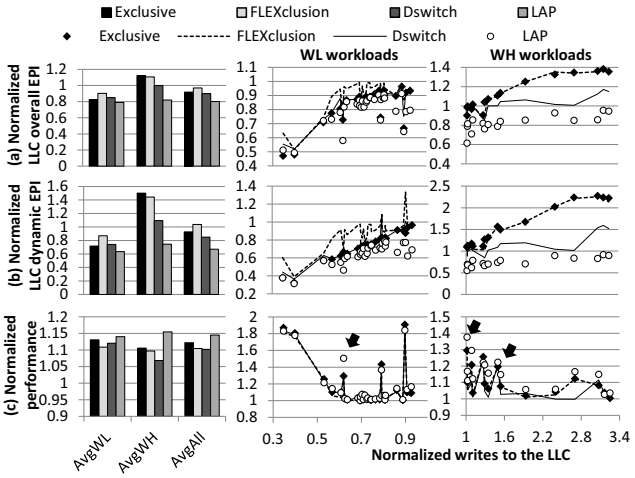


Figure 14: Comparison of different policies in STT-RAM LLC. (a) LLC overall EPI, (b) LLC dynamic EPI, and (c) system performance normalized to the non-inclusive LLC.

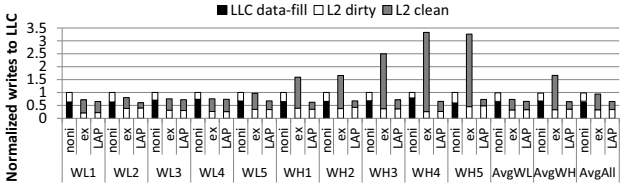


Figure 15: Distribution of writes to the STT-RAM LLC with different policies normalized to the non-inclusive policy.

Number of Writes to the LLC. Figure 15 shows that LAP reduces the number of writes by eliminating redundant LLC data-fills from the main memory and clean insertions from upper level caches, leading to 35% and 29% reduction in write traffic on average over non-inclusion and exclusion.

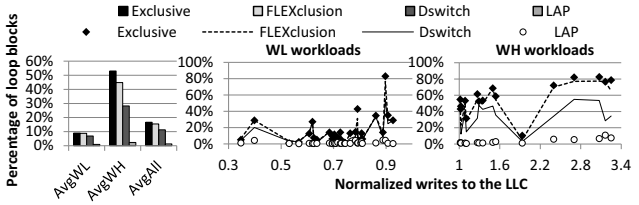


Figure 16: Distribution of loop-blocks in the STT-RAM LLC with different policies.

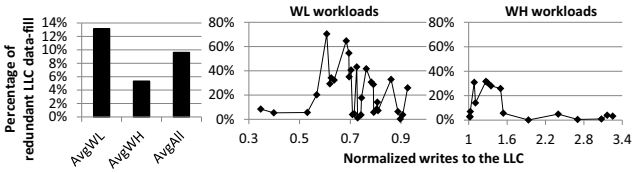


Figure 17: Distribution of redundant data-fill in the non-inclusive LLC.

Since LAP only inserts data from the main memory to upper level caches on LLC misses, the writes incurred by LLC data-fill are eliminated. By keeping duplicate copies of loop-blocks in the LLC, LAP reduces the insertion of clean victims from upper level caches by 30% on average over the exclusive policy. As the number of writes in the exclusive policy increases, such as in WH workloads, the amount of writes eliminated by LAP increases.

Loop-block Elimination. Figure 16 illustrates the distribution of loop-blocks in the STT-RAM with different inclusion properties. WH workloads have a large number of loop-blocks, so exclusion is less energy-efficient than non-inclusion for WH workloads. *FLEXclusion* and Dswitch can reduce the number of loop-blocks by switching to non-inclusion mode at some program phases, thus eliminating the energy-harmful loop-blocks by 1% and 5% on average. LAP can further reduce the number of loop-blocks by 15% on average, leading to higher dynamic energy savings.

Elimination of Redundant Data-Fill. LAP reduces redundant LLC data-fills by only inserting data from the main memory into upper level caches on LLC misses. Figure 17 illustrates that 9.6% of LLC data-fill are redundant on average under non-inclusion. In some workloads, the number of redundant LLC data-fill is higher than 30%. By eliminating these redundant LLC data-fills, LAP consumes less dynamic energy than the non-inclusive policy in STT-RAM LLCs.

Effective Capacity. Figure 18 shows the LLC MPKI of different inclusion properties, indicating the effective capacity provided by different policies. Since there are no duplicate upper-level data in the exclusive LLC, LLC MPKI is reduced by 23% on average over the non-inclusive LLC, leading to better performance and lower leakage energy consumption. LAP aims to store only the duplicate copy of loop-blocks in the STT-RAM LLC to reduce redundant clean data insertions, so the LLC MPKI is 22% lower than the non-inclusive policy on average. By using set-dueling and loop-block-aware replacement policy, LAP only incurs 1% more LLC misses than exclusion on average.

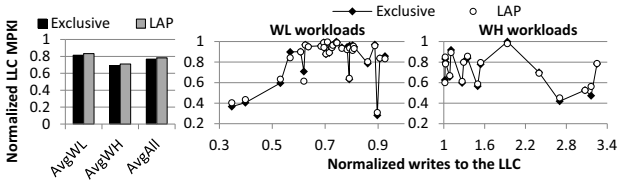


Figure 18: LLC MPKI of different policies normalized to the non-inclusive policy.

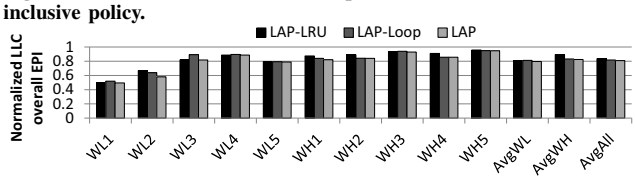


Figure 19: LLC overall EPI of LAP-LRU, LAP-Loop, and LAP normalized to the non-inclusive LLC.

Variants of LAP. Figure 19 analyzes the impact of replacement policy on LAP energy efficiency. For some workloads, such as WL1 and WL3, evicting non-loop-blocks earlier (LAP-Loop) helps reduce redundant writes but incurs a significant number of LLC misses. Thus, the LRU replacement policy (LAP-LRU) consumes less energy than LAP-Loop. On the other hand, the LAP-LRU may evict loop-blocks earlier and elide fewer redundant clean data insertions, such as in WL2 and WH1, leading to higher energy consumption than LAP-Loop. By using set-dueling, LAP dynamically switches between the LRU and the loop-block-aware policy to achieve the best energy efficiency on average.

C. Multi-threaded Workloads

LAP also reduces energy in systems running multi-threaded workloads. Figure 20 (a) shows the energy efficiency of different inclusion properties when running PARSEC [40]. Although some benchmarks, such as *blackscholes*, *bodytrack*, and *swaptions*, have small memory footprints and are compute-intensive with few memory requests, different inclusion properties still lead to variant energy efficiency on average. Since Dswitch takes the impact of high write energy into consideration, it often selects a more energy-efficient inclusion mode than *FLEXclusion* and consumes 3% lower energy than *FLEXclusion* on average. LAP can further reduce redundant LLC data-fill and clean data insertions, resulting in 11% and 7% energy savings over non-inclusion and exclusion on average. The energy savings for *streamcluster* are especially high (53% over non-inclusion and 18% over exclusion) as it demands high cache capacity and frequently reuses clean data with a footprint larger than L2 but smaller than the LLC. Figure 20 (b) shows that with the capacity benefit provided by evicting non-loop-blocks earlier in LAP, the performance is improved by 7% on average than the non-inclusion.

Figure 20 (c) analyzes the amount of coherence traffic in the cache hierarchy when different inclusion policies are applied. Our baseline cache coherence protocol is MOESI snooping protocol. As the exclusive LLC incurs fewer LLC misses than the non-inclusive LLC, the coherence traffic of

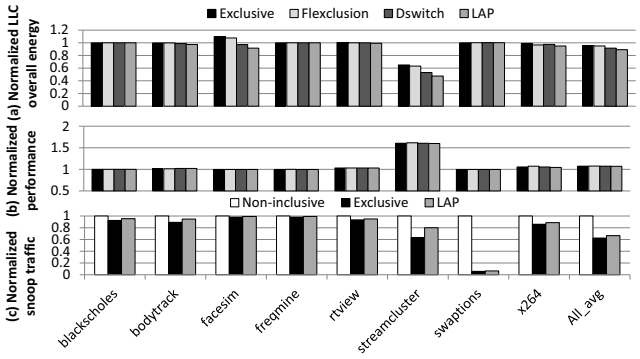


Figure 20: Comparison of different policies in STT-RAM LLC when running PARSEC. (a) LLC overall energy, (b) system performance, and (c) coherence traffic normalized to the non-inclusive LLC.

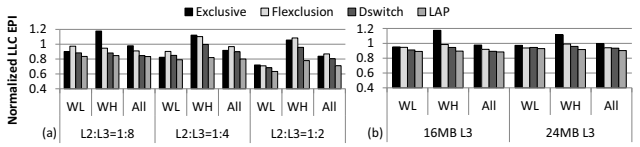


Figure 21: LLC EPI normalized to the non-inclusive policy in systems with different L2:L3 ratios by varying (a) private L2 size and (b) shared L3 size.

exclusion is 38% lower than the non-inclusion on average. LAP sacrifices cache capacity to reduce redundant clean data insertions, leading to 33% less coherence traffic over non-inclusion, but generating 5% more snoop traffic than exclusion. The coherence traffic at *swaptions* is low because most of data accesses hit in the LLC.

D. Sensitivity Studies

We evaluate the energy efficiency of our proposed inclusion policies on different systems, and show that our mechanisms can provide energy savings over existing policies for systems with various cache sizes, number of cores, and range broadly in the technology-driven write/read energy ratio.

L2:L3 Ratio. Figure 21 illustrates that our selective inclusion policies are energy-efficient in systems with different cache size configurations. We first evaluate the systems with different L2:L3 ratios by varying the size of private L2 from 256KB to 1MB ($\sum$(L2):L3 = $1/8$ to $1/2$) with shared 8MB L3, as shown in Figure 21 (a). Since the duplicate data in the non-inclusive LLC waste capacity, the benefit of larger effective capacity under exclusion is more significant at higher L2:L3 ratio. With larger upper-level caches, the clean victim from upper-level caches also decreases, resulting in lower dynamic energy consumption in the exclusive LLC. We observe that the energy savings of the exclusive LLC increases from 2% to 16% when the L2:L3 ratio increases from $1/8$ to $1/2$. The energy savings of LAP over non-inclusion also increases as the L2:L3 ratio increases, by eliminating redundant LLC data-fill and evicting seldom reused non-loop blocks earlier. As the L2:L3 ratio is likely to become larger in future many-core systems [25], LAP is an attractive design choice in terms of energy efficiency. In Figure 21 (b), we also evaluate the systems with smaller L2:L3 ratio by enlarging the LLC capacity, as the high density of STT-RAM could be utilized to provide larger capacity within the same chip area. With larger LLC capacity, the benefit of exclusion becomes less significant. However, even for an iso-area 24MB STT-RAM replacement for an 8MB SRAM LLC, LAP continues to provide an average energy savings of 10% over both non-inclusion and exclusive by reducing redundant LLC insertions.

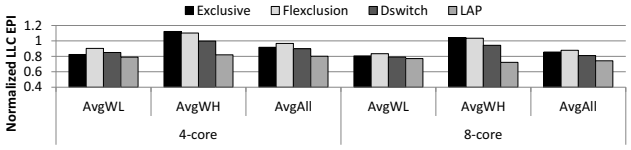


Figure 22: LLC EPI normalized to the non-inclusive policy in the systems with different number of cores.

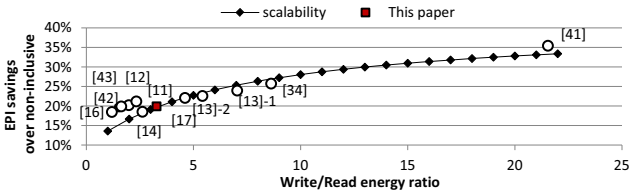


Figure 23: EPI savings of LAP over non-inclusive STT-RAM LLC across variant write/read energy ratio. scalability: energy savings when fixing the read energy and scaling the write energy.

8-Core System. LAP retains energy efficiency across core counts, as shown in Figure 22. We fix the size of caches in the cache hierarchy and vary the number of cores from 4 to 8 in our evaluation to observe the impact of thread contention. As the number of cores increases, exclusion shows greater benefits due to workloads' larger capacity demand. The increase in LLC misses (redundant LLC data-fill) in the non-inclusive LLC outweighs the redundant clean data insertion in the exclusive LLC. Therefore, the average energy savings of exclusion over non-inclusion increase from 8% to 15% when the number of cores increases from 4 to 8. Even in the system with higher core count, LAP can save 25% and 12% energy over non-inclusion and exclusion.

STT-RAM Write/Read Energy Ratio. Figure 23 shows that LAP continues to provide energy benefits across STT-RAM LLCs with variant write/read energy ratios. Different STT-RAM circuit designs consume variant amount of write energy, with writes taking more time and energy than reads. The range of asymmetric write/read energy ratio varies significantly across different optimization goals. For example, researchers have proposed techniques to reduce the write latency or energy of STT-RAM by sacrificing the retention time or cell density [12, 13]. Bit-flip reduction techniques [20, 21] effectively alter the write/read ratio for a given technology with significant reductions in write energy for some cost in read. In Figure 23, we fix the read energy and leakage power consumption, but scale the write energy to show that LAP provides higher energy savings as the write/read energy ratio increases. The exacerbated impact of redundant LLC data-fill and clean victim insertions leads

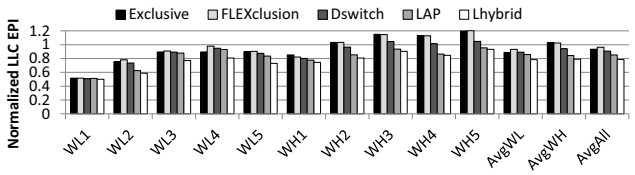


Figure 24: Hybrid LLC EPI normalized to non-inclusive policy.

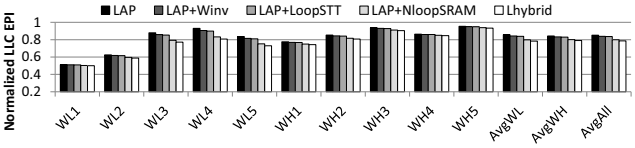


Figure 25: LLC EPI of different data placement policies in the hybrid LLC, normalized to the non-inclusive policy. Winv: invalidating STT-RAM block when write-hit. LoopSTT: storing loop-blocks in STT-RAM. NloopSRAM: storing non-loop-blocks in SRAM.

to the significant energy savings at high write/read energy ratio. Even if the write energy of STT-RAM is only 2x read energy, LAP can still provide 17% energy savings. We also show that the predicted scalability curve matches the energy savings when the configurations from published data [11–14, 16, 17, 34, 41–43] are applied. The energy savings of LAP at some of the configurations [11, 12, 16, 42, 43] is slightly different from our predicted curve due to variant settings of access latency and leakage power, but the write/read energy ratio is the key factor that impacts the amount of energy savings provided by LAP, ***indicating that LAP is also applicable to other asymmetric memory technologies***.

E. Hybrid SRAM/STT-RAM LLC

For a hybrid SRAM/STT-RAM LLC, we evaluate the energy-efficiency of different policies, including our Lhybrid policy extension of LAP that utilizes the characteristic of loop-blocks to guide the data placement in the hybrid LLC. Figure 24 shows that LAP can also help to reduce the energy consumption of hybrid LLCs. Compared to non-inclusion and exclusion, Dswitch saves 10% and 3% energy on average. LAP can save more energy than Dswitch by eliminating redundant writes to the LLC to achieve 15% and 8% average energy savings over non-inclusion and exclusion. By keeping frequently written non-loop-blocks in the SRAM, our Lhybrid policy can further reduce the energy consumption by 22% and 15% on average (up to 50% and 41%), compared to non-inclusion and exclusion.

Figure 25 analyzes the improvement in energy-efficiency provided by different stages in Lhybrid. We can observe that invalidating STT-RAM blocks and writing to the SRAM when write-hits occur (LAP+Winv) slightly reduces the energy consumption of the hybrid LLC. Forcing the loop-blocks to be stored in STT-RAM banks (LAP+LoopSTT) also slightly improves the energy efficiency. For some workloads, such as WL3, WL4, and WL5, the energy reduction mainly comes from forcing the frequently written non-loop-blocks to be stored in the SRAM (LAP+NloopSRAM). Combining these three stages, the Lhybrid policy can provide 7% higher energy savings on average over LAP.

VII. Related Work

Researchers have studied the inclusion property design in traditional SRAM LLCs to trade-off capacity benefit, complexity of the coherence protocol, and on-chip bus traffic [22–25, 32, 33, 36, 44]. Baer et al. [22] studied inclusive cache hierarchies to simplify the cache coherence protocol. Zahran et al. [23] discussed the benefits of non-inclusive caches, and Jaleel et al. [33] proposed methods to achieve the performance of non-inclusive caches without violating the inclusion property. Jouppi et al. [44] first proposed exclusive caches to reduce LLC misses. The benefits of exclusive caches have been studied in both general-purpose [24] and server systems [32], and several prior studies proposed bypass and replacement policies [32, 36] to improve exclusive cache hierarchy performance. Sim et al. [25] further proposed *FLEXclusion* to dynamically configure the LLC to non-inclusion or exclusion for better performance and lower on-chip bandwidth consumption. However, these prior approaches targeted traditional SRAM LLCs and did not consider the impact of write traffic, due to the inclusion property of the LLC, on energy consumption. In this paper, we demonstrate that these existing policies are not energy-efficient when being directly applied to STT-RAM LLCs.

Several prior studies have proposed different methods to reduce unnecessary writes to an STT-RAM-based LLC [17–21, 34]. Bit-flip techniques [20, 21] are proposed to save energy in NVM by reducing the number of written bits. Our work is orthogonal to these bit-flip approaches, and these bit-flip techniques can be applied on top of our design. Ahn et al. [34] proposed a mechanism to predict and bypass dead writes to STT-RAM LLCs for energy reduction. Their deadblock bypassing technique is orthogonal to our selective inclusion policies and can be combined with our approaches to further reduce the dynamic energy consumption. Other prior researchers utilized hybrid SRAM/STT-RAM cache organization to service some writes in faster and less energy-consuming SRAM banks for better energy efficiency [17–19]. They proposed different approaches to migrate write-intensive blocks to SRAM for reducing STT-RAM writes. Nevertheless, these prior write-reduction methods are all based on non-inclusive or inclusive LLCs, and the LLC access pattern would be different under different inclusion properties. To the best of our knowledge, no prior studies have designed inclusion properties for STT-RAM LLCs targeting energy-efficiency with comparable performance.

VIII. Conclusion

No existing inclusion property applied in SRAM LLCs is dominant in terms of energy-efficiency in STT-RAM LLCs, as different inclusion properties generate different number of writes to the LLC and the high write energy in STT-RAM cannot be ignored. We propose and evaluate a Loop-block-Aware Policy (LAP) to effectively utilize cache capacity to reduce redundant writes as an alternative to

traditional inclusion properties, and extend LAP to serve hybrid SRAM/STT-RAM LLCs. Compared to non-inclusive and exclusive policies, LAP saves 20% and 12% energy, respectively, for STT-RAM LLCs, and 22% and 15% for hybrid LLCs, while improving average case performance, imposing small worst-case performance degradation, and requiring minimal hardware overheads. For the workloads examined, LAP outperforms techniques [25, 26] that dynamically switch between traditional exclusive and non-inclusive modes. Furthermore, we find that the key indicator of energy-sensitivity to the choice of inclusion property is the write/read energy ratio of the memory technology, so approaches developed in this work should apply broadly across other asymmetric memory technologies.

ACKNOWLEDGMENT

We thank anonymous reviewers, Meng-Fan Chang, Lei Wang, and SEAL group members for their valuable feedback and suggestion. This work was supported in part by NSF grants 1409798, 1461698, 1500848, 1533933, a grant from Qualcomm, and Department of Energy under Award Number DE-SC0013553.

REFERENCES

[1] N. Kurd *et al.*, "Westmere: A family of 32nm IA processors," in *ISSCC*, 2010.

[2] D. Wendel *et al.*, "The implementation of POWER7™: A highly parallel and scalable multi-core high-end server processor," in *ISSCC*, 2010.

[3] S. Li *et al.*, "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *MICRO*, 2009.

[4] U. Nawathe *et al.*, "Implementation of an 8-core, 64-thread, power-efficient SPARC server on a chip," *JSSCC*, vol. 43, no. 1, 2008.

[5] D. Apalkov *et al.*, "Spin-transfer torque magnetic random access memory (STT-MRAM)," *JETC*, vol. 9, no. 2, 2013.

[6] B. C. Lee *et al.*, "Architecting phase change memory as a scalable DRAM alternative," in *ISCA*, 2009.

[7] M. K. Qureshi *et al.*, "Scalable high performance main memory system using phase-change memory technology," in *ISCA*, 2009.

[8] C. Xu *et al.*, "Overcoming the challenges of crossbar resistive memory architectures," in *HPCA*, 2015.

[9] P. Chi *et al.*, "A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory," in *ISCA*, 2016.

[10] L. Zhang *et al.*, "Mellow writes: Extending lifetime in resistive memories through selective slow write backs," in *ISCA*, 2016.

[11] H. Noguchi *et al.*, "A 3.3ns-access-time 71.2 uW/MHz 1Mb embedded STT-MRAM using physically eliminated read-disturb scheme and normally-off memory architecture," in *ISSCC*, 2015.

[12] H. Noguchi *et al.*, "Highly reliable and low-power nonvolatile cache memory with advanced perpendicular STT-MRAM for high-performance CPU," in *VLSIC*, 2014.

[13] C. W. Smullen *et al.*, "Relaxing non-volatility for fast and energy-efficient STT-RAM caches," in *HPCA*, 2011.

[14] H. Noguchi *et al.*, "A 250-MHz 256b-I/O 1-Mb STT-MRAM with advanced perpendicular MTJ based dual cell for nonvolatile magnetic caches to reduce active power of processors," in *VLSIT*, 2013.

[15] S. Chung *et al.*, "Fully integrated 54nm STT-RAM with the smallest bit cell dimension for high density memory application," in *IEDM*, 2010.

[16] K. Tsuchida *et al.*, "A 64Mb MRAM with clamped-reference and adequate-reference schemes," in *ISSCC*, 2010.

[17] Z. Wang *et al.*, "Adaptive placement and migration policy for STT-RAM-based hybrid cache," in *HPCA*, 2014.

[18] X. Wu *et al.*, "Hybrid cache architecture with disparate memory technologies," in *ISCA*, 2009.

[19] Y.-T. Chen *et al.*, "Static and dynamic co-optimizations for blocks mapping in hybrid caches," in *ISLPED*, 2012.

[20] B. C. Lee *et al.*, "Phase-change technology and the future of main memory," *IEEE Micro*, vol. 30, no. 1, 2010.

[21] S. Cho and H. Lee, "Flip-N-Write: A simple deterministic technique to improve PRAM write performance, energy and endurance," in *MICRO*, 2009.

[22] J.-L. Baer and W.-H. Wang, "On the inclusion properties for multi-level cache hierarchies," in *ISCA*, 1988.

[23] M. M. Zahran *et al.*, "Non-inclusion property in multi-level caches revisited," *IJCA*, 2007.

[24] Y. Zheng *et al.*, "Performance evaluation of exclusive cache hierarchies," in *ISPASS*, 2004.

[25] J. Sim *et al.*, "FLEXclusion: Balancing cache capacity and on-chip bandwidth via flexible exclusion," in *ISCA*, 2012.

[26] H.-Y. Cheng *et al.*, "Dswitch: Write-aware dynamic inclusion property switching for emerging asymmetric memory technologies," PSU CSE16-004, Tech. Rep., 2016.

[27] N. Muralimanohar *et al.*, "CACTI 6.0: A tool to model large caches," HP Lab, Tech. Rep., 2009.

[28] X. Dong *et al.*, "NVSim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *TCAD*, vol. 31, no. 7, 2012.

[29] Intel, "Intel Core i7 Processors," http://www.intel.com/content/www/us/en/processors/core/core-i7-processor.html.

[30] Intel, "First the tick, now the tock: next generation Intel microarchitecture (Nehalem)," in *Intel Whitepaper*, 2008.

[31] AMD, "Phenom II processor model," http://www.amd.com/en-us/products/processors/desktop/phenom-ii.

[32] A. Jaleel *et al.*, "High performing cache hierarchies for server workloads: Relaxing inclusion to capture the latency benefits of exclusive caches," in *HPCA*, 2015.

[33] A. Jaleel *et al.*, "Achieving non-inclusive cache performance with inclusive caches: Temporal locality aware (TLA) cache management policies," in *MICRO*, 2010.

[34] J. Ahn *et al.*, "DASCA: Dead write prediction assisted STT-RAM cache architecture," in *HPCA*, 2014.

[35] M. K. Qureshi *et al.*, "Adaptive insertion policies for high performance caching," in *ISCA*, 2007.

[36] J. Gaur *et al.*, "Bypass and insertion algorithms for exclusive last-level caches," in *ISCA*, 2011.

[37] A. Jaleel *et al.*, "High performance cache replacement using re-reference interval prediction (RRIP)," in *ISCA*, 2010.

[38] N. Binkert *et al.*, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, 2011.

[39] M. Gebhart *et al.*, "Running PARSEC 2.1 on M5," UT Austin, CS Dept., Tech. Rep., 2009.

[40] C. Bienia *et al.*, "The PARSEC benchmark suite: Characterization and architectural implications," in *PACT*, 2008.

[41] M.-T. Chang *et al.*, "Technology comparison for large last-level caches (L3Cs): Low-leakage SRAM, low write-energy STT-RAM, and refresh-optimized eDRAM," in *HPCA*, 2013.

[42] D. Halupka *et al.*, "Negative-resistance read and write schemes for STT-MRAM in 0.13um CMOS," in *ISSCC*, 2010.

[43] T. Ohsawa *et al.*, "1Mb 4T-2MTJ nonvolatile STT-RAM for embedded memories using 32b fine-grained power gating technique with 1.0ns/200ps wake-up/power-off times," in *VLSIC*, 2012.

[44] N. Jouppi and S. Wilton, "Tradeoffs in two-level on-chip caching," in *ISCA*, 1994.