THE UNIVERSITY *of* EDINBURGH

Edinburgh Research Explorer

Architecting Selective Refresh based Multi-Retention Cache for Heterogeneous System (ARMOUR)

Citation for published version:
Agarwal, S, Chakraborty, S & Själander, M 2023, Architecting Selective Refresh based Multi-Retention Cache for Heterogeneous System (ARMOUR). in *DAC '23: Proceedings of the 60th ACM/IEEE Design Automation Conference.* IEEE, pp. 1-6, 60th Design Automation Conference, San Francisco, United States, 9/07/23. https://doi.org/10.1109/DAC56929.2023.10247878

Digital Object Identifier (DOI):
10.1109/DAC56929.2023.10247878

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Peer reviewed version

Published In:
DAC '23: Proceedings of the 60th ACM/IEEE Design Automation Conference

General rights
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 07. Aug. 2024

Architecting Selective Refresh based Multi-Retention Cache for Heterogeneous System (ARMOUR)

Sukarn Agarwal
The University of Edinburgh
Edinburgh, UK
sagarwa2@ed.ac.uk

Shounak Chakraborty
Norwegian University of Science and Technology, Trondheim, Norway
shounak.chakraborty@ntnu.no

Magnus Själander
Norwegian University of Science and Technology, Trondheim, Norway
magnus.sjalander@ntnu.no

***Abstract*—The increasing use of chiplets, and the demand for high-performance yet low-power systems, will result in heterogeneous systems that combine both CPUs and accelerators (e.g., general-purpose GPUs). Chiplet based designs also enable the inclusion of emerging memory technologies, since such technologies can reside on a separate chiplet without requiring complex integration in existing high-performance process technologies. One such emerging memory technology is spin-transfer torque (STT) memory, which has the potential to replace SRAM as the last-level cache (LLC). STT-RAM has the advantage of high density, non-volatility, and reduced leakage power, but suffers from a higher write latency and energy, as compared to SRAM. However, by relaxing the retention time, the write latency and energy can be reduced at the cost of the STT-RAM becoming more volatile. The retention time and write latency/energy can be traded against each other by creating an LLC with multiple retention zones. With a multi-retention LLC, the challenge is to direct the memory accesses to the most advantageous zone, to optimize for overall performance and energy efficiency. We propose *ARMOUR*, a mechanism for efficient management of memory accesses to a multi-retention LLC, where based on the initial requester (CPU or GPU) the cache blocks are allocated in the high (CPU) or low (GPU) retention zone. Furthermore, blocks that are about to expire are either refreshed (CPU) or written back (GPU). In addition, *ARMOUR* evicts CPU blocks with an estimated short lifetime, which further improves cache performance by reducing cache pollution. Our evaluation shows that *ARMOUR* improves average performance by 28.9% compared to a baseline STT-RAM based LLC and reduces the energy-delay product (EDP) by 74.5% compared to an iso-area SRAM LLC.**

***Index Terms*—STT-RAM, NVM, retention time, multi-retention cache, heterogeneous system, GPU, energy efficiency**

I. Introduction

The demand for high performance yet low power has led to an increasing use of heterogeneous systems that integrates CPUs and general-purpose GPUs (GPGPUs). With the increasing use of chiplets for high-performance systems, CPUs and GPGPUs are becoming tightly integrated, such as in Intel's proposed Meteor Lake CPUs [1]. For such systems, it is desirable to have a single shared last-level cache (LLC) to simplify data sharing in CPU-GPGPU workloads. However, a sufficiently large SRAM-based LLC requires a large amount of area and comes with a high static-energy usage. To address these issues, computer architects are investigating emerging memory technologies, such as spin-transfer torque RAM (STT-RAM) [2], phase change memory (PCM) [3], and flash memory [4]. Out of these, STT-RAM has the potential to replace conventional SRAM in cache designs due to its low leakage power, high endurance, and comparable read access times.

STT-RAM, however, suffers from longer write latency and increased write energy as compared to SRAM [5]. Both write latency and energy related issues can be mitigated by lowering the retention time of the STT-RAM cell [6], but a lower retention time can increase the required number of writebacks or refreshes due to early expiry of cache blocks. The increased writebacks or refreshes give rise to increased power and energy overheads [7], [8]. A higher retention time reduces the refreshes and writebacks by enhancing the lifetime of the individual blocks, at the cost of increased write energy and latency. However, the lifetime of the cache blocks is drastically curtailed when executing CPU and GPU workloads in parallel, in a heterogeneous system, due to the increased cache contention caused by interference [9], [10]. A plethora of prior arts [9]–[11] tackle cache contention and interference, but none of the previous studies considered multi-retention STT-RAM based caches, which balance the benefits and disadvantages of different retention times by allocating blocks with shorter (longer) cache lifetime in the lower (higher) retention zone.

By simulating a set of CPU-GPGPU workload combinations, we concluded that the cache lifetimes of both CPU and GPGPU applications are drastically reduced while executing together. Additionally, the cache lifetime of GPU cache blocks is significantly lower than of CPU cache blocks, on average, which motivates us to writeback throughput oriented GPU cache blocks on expiry, whereas, the performance critical CPU blocks are refreshed. We propose *ARMOUR*, a mechanism for efficient management of multi-retention STT-RAM LLCs. *ARMOUR* differentiates between CPU and GPU blocks, allocates GPU (CPU) blocks in the lower (higher) retention zone, writes back GPU blocks, and dynamically refreshes CPU blocks on expiry.

As the name of the mechanism suggests, *ARMOUR* protects live CPU blocks against large bursts of GPGPU memory operations and dead CPU blocks. The main contributions of *ARMOUR* are as follows:

- We present an analysis on how the average lifetime of cache blocks are significantly reduced when executing both CPU and GPGPU workloads in parallel than when executing them in isolation;
- *ARMOUR* identifies dead CPU blocks and proactively evicts them from the LLC to free up cache space;

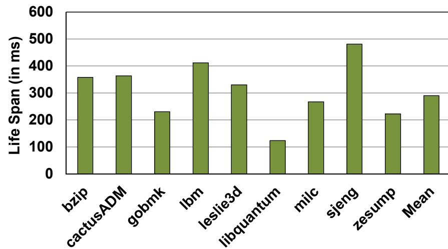

Fig. 1: Average lifetime of CPU-only workload.

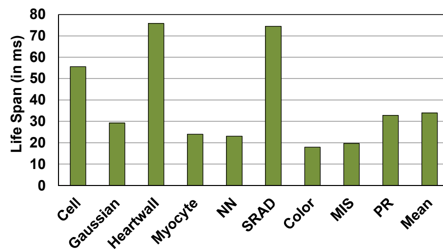

Fig. 2: Average lifetime of GPU-only workloads.

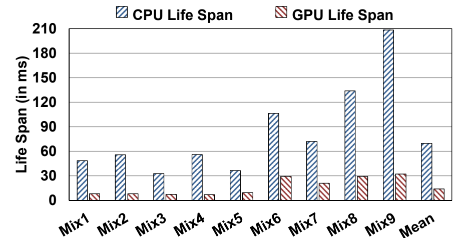

Fig. 3: Average lifetime of mixed CPU-GPGPU workloads.

TABLE I: Comparison of *refresh* based vs *writeback* based relaxed cache (normalized to 8 MiB SRAM cache)

	Type	CPI	Energy	Refresh	MPKI	EDP
20 ms	Refresh	1.48	8.00	23.6%	1.01	0.45
	Writeback	1.31	6.00		1.53	0.35
50 ms	Refresh	1.37	7.42	15.5%	1.02	0.39
	Writeback	1.29	5.91		1.21	0.33
100 ms	Refresh	1.26	6.93	7.43%	1.04	0.34
	Writeback	1.35	6.22		1.10	0.37
500 ms	Refresh	1.42	6.85	1.75%	1.01	0.41
	Writeback	1.40	6.56		1.07	0.39

- *ARMOUR* protects the residency of live CPU blocks against GPU blocks with poor temporal locality and dead CPU blocks.

The *ARMOUR* architecture is described in Sec. III after discussing our motivation in Sec. II. Our results (Sec. V) show that *ARMOUR* reduces the misses per kilo instructions (MPKI) by up to 51% over the baselines, resulting in a reduction in cycles per instruction (CPI) by more than 20% and more than 15% over iso-area SRAM and iso-capacity STT-RAM based LLCs, respectively. Further, *ARMOUR* achieves a significant reduction in energy delay product (EDP) of 74.5% compared to an SRAM based iso-area LLC.

II. Motivation

We begin with analyzing a set of CPU and GPGPU workloads by executing them in isolation and in parallel to study the difference in LLC usage behavior. We use SPEC (CPU) [12], Rodinia [13], and Pannotia [14] (GPGPU) workloads with an 8 MiB SRAM LLC in gem5-gpu [15] for 80M cycles. Sec. IV details our simulation framework and selected workloads. The details for individual applications and average cache lifetime for CPU workloads are shown in Figure 1, for GPU workloads in Figure 2, and for mixed workloads in Figure 3. From the results, we conclude that the average lifetime for CPU blocks is 290 ms and for GPU blocks is 34 ms when executed in isolation. *When executing both CPU and GPGPU workloads in parallel, the average cache lifetime reduces significantly due to cache contention caused by interference.* The average lifetime of CPU blocks is reduced to 70 ms and for GPU blocks to 14 ms, which can aggravate performance by premature eviction of the cache blocks.

We further study the power-performance tradeoffs with an 8 MiB relaxed STT-RAM that either refreshes or writes back blocks for different retention times of 20 ms, 50 ms, 100 ms, and 500 ms, which are listed in Table I. All the result metrics in the table are normalized to an 8 MiB SRAM-based LLC. As evident from the table, on a lower retention zone of 20 ms, the increased refreshes lead to increased energy usage than the writeback approach. However, the writeback approach increases the miss rate due to premature eviction of cache blocks, which degrades the CPI. On the other hand, a higher retention time drastically reduces writeback and/or refresh overheads, which improves the MPKI and energy overheads. But, higher retention time can incur higher write latency and write energy. Hence, there is a trade-off between the refresh and writeback approach regarding the application and core behavior. Thus, our empirical evaluation motivates us to write back GPU cache blocks due to their short lifetimes, and to refresh the performance-critical CPU blocks that benefit from longer cache residency. This reduces premature eviction of performance critical cache blocks, a prime objective of *ARMOUR*.

III. *ARMOUR*: Proposed Architecture

In this section, we present the core idea of *ARMOUR*. We assume a system equipped with a shared LLC and a set of CPU and GPU cores, where each core has its own private L1 caches. *ARMOUR* uses a multi-retention LLC architecture that is partitioned way-wise. Figure 4 illustrates a 16-way LLC with four retention zones, each consisting of four ways.

A requested cache block can be placed anywhere within its designated cache set, but, during block allocation, the search direction for a suitable block to evict is based upon the requester, i.e., if it is coming from a CPU or GPU core. If the requester is a CPU, then the search for an invalid or clean non-most-recently-used (non-MRU) entry starts from the higher retention zone (i.e., from way 15) and ends at the lower retention zone. If the requester is a GPU, then the search starts from the lower retention zone (i.e., from way 0) and ends at the higher retention zone. Figure 4 illustrates this allocation strategy. On absence of a clean non-MRU block in a set, the first non-MRU block found during the search is evicted. For identifying, if a block allocation has been done by a CPU or a GPU core, a single *id*-bit is added to each block entry.

ARMOUR further introduces a simple block management strategy by considering the lifetime of CPU and GPU cache blocks. Our preliminary evaluation shows that, the lifetime of CPU blocks in the LLC is much higher than for GPU blocks. Hence, our proposition is to populate the LLC with more CPU blocks than GPU blocks. On the expiry of a cache block, the *id*-bit is checked to identify the initiator of the block. If the

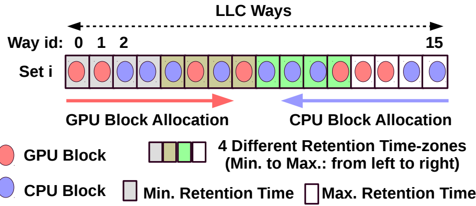

Fig. 4: Allocation strategy of *ARMOUR* for the CPU and GPU blocks in the multi-retention STT-RAM LLC.

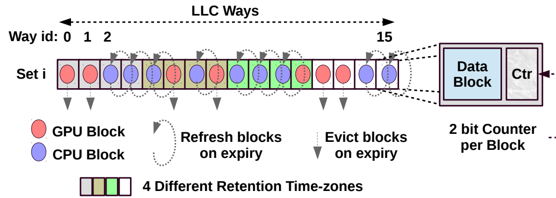

Fig. 5: Handling of CPU and GPU blocks in a multi-retention STT-RAM LLC.

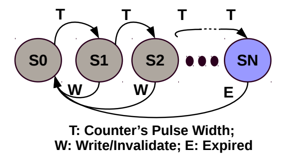

Fig. 6: Counter for monitoring block expiry.

initiator is a GPU core, then the block is evicted from the LLC, with dirty blocks being written back to memory. Whereas, if the initiator is a CPU core, then the block is refreshed and retained in the LLC. Figure 5 depicts the block management strategy on the verge of expiry. As seen, the CPU blocks at cache ways 2, 3, 4, etc. are refreshed (shown with feedback arrow), whereas, the GPU blocks at cache ways 0, 1, 5, etc. are written back (shown with downward arrow).

At the end of the retention times, the individual block's expiry needs to be tracked with the help of a counter, as shown in Figure 6. A clock period is defined that is N times smaller than the retention time, where N implies the number of states used by the counter. The counter advances to the next state on a fixed time-out (T) unless there is a write (W) to the block. Once the counter reaches the saturation point (i.e., state SN), the expiry of the block (E) is detected and subsequently the block is either evicted or refreshed based upon the *id*-bit and the counter is reset ($S0$). On a write or invalidation (W), the counter is always reset ($S0$). The implementation of the counter can be adopted by employing a prior technique [16].

An increased number of CPU blocks in the LLC might lead to cache pollution by containing a higher number of dead or unused blocks, i.e., blocks that are never accessed again before they get evicted. Our results (Table VI) show that dead blocks are responsible for 64% of all blocks that are refreshed, on average. To address this issue, *ARMOUR* tracks read accesses of the individual blocks, for which a 2-bit saturating counter, attached to each LLC block, is used. Figure 7 shows the associated 2-bit counter. The counter for a block is reset every time it is refreshed or written. On every read operation, the counter is incremented. The counter value for a block is assessed on the expiry of the cache block. If the counter value is above a set threshold (ThX) for the given retention zone, then the block is refreshed, else the block is evicted. Figure 7 shows an example where CPU blocks at cache ways 2, 4, 8, 9, and 15 are refreshed as their counter value is larger than the respective retention threshold. Whereas, due to lower counter value than the retention threshold, the CPU blocks at cache ways 3, 6, 10, and 14 are evicted.

Implementation Overhead: To implement *ARMOUR*, we need a 2-bit counter for each block, and an *id*-bit for identifying if the block is allocated by a CPU or a GPU core. Hence, each block has three additional bits, which constitutes a negligible area overhead of less than 1% [17]. Other implementation overheads include the time tracking for each block before triggering a transition to another state, which is implemented

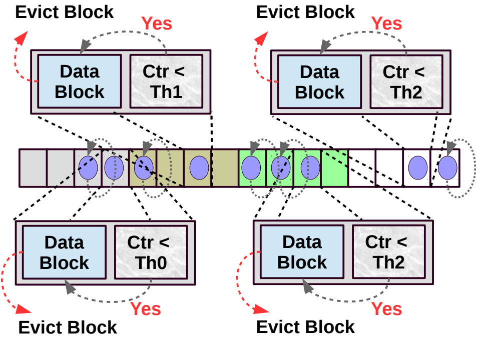

Fig. 7: Eviction of CPU-blocks based on the read counter and respective thresholds of different retention time zones.

TABLE II: System Configuration

System Components	Configuration
GPU Core	16 shader cores, 1.4 GHz, 48 warps/core, 32 threads/warp
CPU Core	x86 quad-core, 2GHz
GPU L1 SRAM Cache	16 KiB, 4-way I/D cache, 12 KiB 24-way texture cache, 8KiB 2-way constant cache, 128B cache block
CPU L1 SRAM Cache	32 KiB, 4-way I/D, 128 B cache block
Shared STT/SRAM LLC	8 × 1MiB / 8 × 512 KiB, 16-way, 128 B line
Main Memory	8 GiB DRAM

by incorporated a prior technique [16]. Note that, the CPU blocks are selectively refreshed on expiry, for which we adopt the hardware mechanism discussed in CacheRevive [18].

IV. Simulation Framework and Benchmarks

We evaluate *ARMOUR* on gem5-gpu [15]. Table II reports the system configuration consisting of four x86 CPU cores and 16 GPU cores. Each CPU and GPU core has its own private L1 instruction and data caches. The multi-retention STT-RAM LLC is shared between the CPU and GPU cores. The shared LLC is divided into eight banks of 1 MiB each. Each bank is partitioned into four different retention time zones of 20 ms, 50 ms, 100 ms, and 500 ms. The retention times and energy parameters are detailed in Table III, obtained from CACTI-STT [17]. To measure the efficacy of *ARMOUR*, we executed a range of CPU and GPGPU workloads from the SPEC CPU 2006 [12], Rodinia [13], and Pannotia [14] benchmark suits, as given in Table IV. Based on the MPKI of respective CPU workload, different mixes are constructed, as reported in Table V. Each mix consists of three CPU workloads of high, mid, and low MPKI, and one GPGPU workload. We ran each mix of workloads for 500 million instructions, with both CPU and GPGPU workloads executed in parallel.

TABLE III: Timing, Energy and Area values for Iso-Area SRAM and STT-RAM cahes (4/8MiB, 16-way) for multiple retention times [17]

Memory Device	SRAM	STT-RAM			
Retention Time/Feature Size	22nm CMOS (A)	20ms	50ms	100ms	500ms
Wr Energy (pJ) (per access/bit)	0.239	5.01	5.22	5.37	5.61
Rd Energy (pJ) (per access/bit)	0.239	0.209	0.211	0.213	0.222
Leakage Power (at 350K)	326.71 mW	84.409 mW			
Rd Latency (cycles)	2	2	2	2	2
Wr Latency (cycles)	2	6	7	8	9
Area	1.8116	0.425	0.432	0.436	0.467

TABLE IV: Benchmark Suites and Applications (L: Low MPKI, M: Mid MPKI and H: High MPKI)

Benchmark Suites		Applications
CPU	**SPEC**	bzip (L), cactusADM (H), gobmk (L), lbm (M), leslie3d (M), libquantum (L), milc (H), sjeng (H), zesump (M)
GPU	**Rodinia**	Cell, Gaussian, Heartwall, Myocyte, NN, SRAD
	Pannotia	Color, MIS, Page Rank (pr)

We evaluate the following system configurations: The **SRAM** configuration has an iso-area SRAM-based LLC of 4 MiB with the same area budget as an 8 MiB STT-RAM based LLC, as evident from Table III. Whereas the **STT-REF** and **STT-WB** configurations are the iso-capacity multi-retention STT-RAM LLC that either refreshes or writes back the cache blocks on expiry, respectively. The **STT-REF&WB** configuration has an iso-capacity multi-retention STT-RAM LLC that refreshes CPU blocks and writes back GPU blocks on expiry. ***ARMOUR*** is similar to STT-REF&WB with the addition of evicting CPU blocks that have not reached the read threshold (ThX) as described in Sec. III. Note that, in our simulation, we set $Th0 = 1$, $Th1 = 2$, and $Th2 = 3$[1].

V. Results and Analysis

In this section, we first discuss the allocation of CPU and GPGPU blocks across the retention zones, and how this allocation in combination with proactive dead CPU block eviction improve the performance for each application. Next, the impact of *ARMOUR* on write endurance and energy usage is discussed. Lastly, partitioned vs. shared caches are compared, while implementing partitioned cache with restricted allocation.

A. Performance Analysis

We first present the refresh percentages of the dead blocks for different mixes of CPU and GPGPU workloads in Table VI. Table VI also presents the GPU block allocation percentage across different retention zones. On average, 62% of GPU blocks are allocated to lower retention zones (i.e., 20 ms and 50 ms), which shows that *ARMOUR* effectively accounts for the lower temporal locality of the GPU applications.

Figure 8 presents the CPI normalized to the SRAM configuration. *ARMOUR* achieves a CPI improvement of 28.9%, 22.3%, 15.3% and 7.5% over STT-WB, SRAM, STT-REF, and STT-REF&WB, respectively. Whereas the respective CPI improvements by STT-REF&WB are 23.1%, 16%, and 8.4%

[1]We performed the experiments on different ThX values and found these as stable ones. The results are not shown due to the space limitation.

TABLE V: CPU-GPGPU Workload Mixes

Mix	Applications
Mix1	sjeng, leslie3d, libquantum, cell
Mix2	cactusADM, zesump, gobmk, gaussian
Mix3	milc, leslie3d, bzip, heartwall
Mix4	milc, zesump, libquantum, myocyte
Mix5	sjeng, lbm, bzip, nn
Mix6	sjeng, leslie3d, bzip, srad
Mix7	cactusADM, leslie3d, bzip, color
Mix8	milc, zesump, gobmk, mis
Mix9	cactusADM, lbm, libquantum, pr

TABLE VI: Dead block refresh percentage across different mixes and GPU block allocation percentage in different retention zones

Mixes	**Dead Block Refresh (in %)**	**GPU Block Allocation at each retention zone (in %)**			
		20ms	**50ms**	**100ms**	**500ms**
Mix1	65.2%	31.8%	29.6%	19.9%	18.6%
Mix2	65%	33.7%	29%	19.9%	17.4%
Mix3	61.1%	34.9%	31.6%	18.4%	15%
Mix4	60.1%	32.3%	29.1%	20.8%	17.8%
Mix5	60.8%	32.2%	28.5%	20.5%	18.7%
Mix6	71.1%	32.1%	29.8%	18.7%	19.4%
Mix7	60.1%	31.6%	31.3%	19.4%	17.6%
Mix8	64.8%	31%	31%	19.6%	18.4%
Mix9	69.7%	31.5%	33.8%	20.2%	14.4%
Mean	64.1%	32.3%	30.4%	19.7%	17.5%

over STT-WB, SRAM, and STT-REF, respectively. These improvements are due to the reduction in refreshes and writebacks, which results in a significant reduction in MPKI, as shown in Figure 9.

We present the normalized reduction in refresh and writeback counts over STT-REF and STT-WB in Figure 10 and 11, respectively. In particular, the refresh counts of *ARMOUR* are reduced by 68.1%, and 56.5% over STT-REF, and STT-REF&WB, respectively. Whereas, the writeback count for *ARMOUR* is reduced by 59.9% for STT-WB, but it has been increased by 74.7% over STT-REF&WB. Hence, the selective refresh scheme of *ARMOUR* lowers both the refresh and writeback counts, and the proactive evictions of dead CPU blocks improve performance. However, the increase in writebacks of *ARMOUR* over STT-REF&WB is due to an increased number of writebacks of CPU dead blocks, which increases the live block count on-chip and ultimately improves performance.

B. Endurance Analysis

Figure 12 presents the intra-set write variation, which is the write variation between the blocks inside a cache set [19]. *ARMOUR* reduces the write-variation by 10.67%, 9.1%, and 3.5% over STT-REF, STT-WB and STT-REF&WB, respectively. The respective improvements in the write variation by *ARMOUR* is due to the allocation strategy that allocates CPU blocks from higher to lower retention zones and GPU block from lower to higher retention zones, as shown in Figure 4. Furthermore, writing back the cache block to the next level of memory on expiry ensures that writes are not concentrated on one particular physical cache location. Hence, with the reduction of write variation, the lifetime improvement (the inverse of the maximum writes to a cache block) normalized to STT-REF are 2.14, 1.78, and 1.38 times over STT-REF, STT-WB, and STT-REF&WB, respectively, as shown in Figure 13.

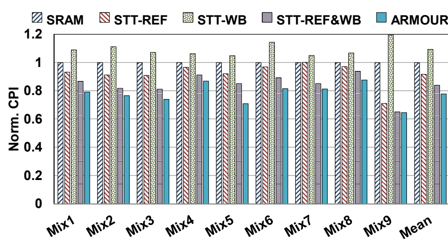

Fig. 8: Normalized CPI wrt. SRAM. (Lower is better)

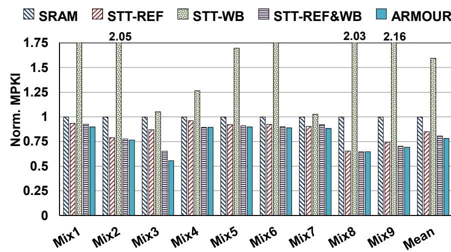

Fig. 9: Normalized MPKI wrt. SRAM. (Lower is better)

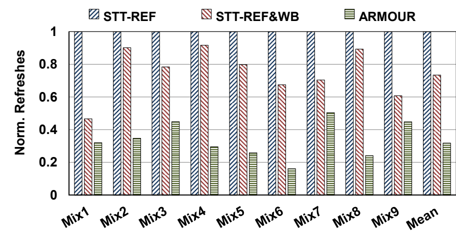

Fig. 10: Normalized Refreshes wrt. STT-REF. (Lower is better)

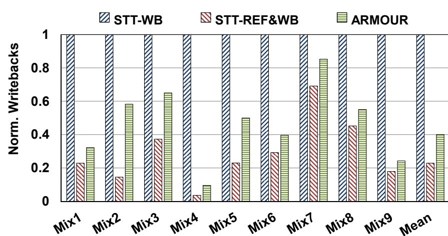

Fig. 11: Normalized writebacks wrt. STT-WB. (Lower is better)

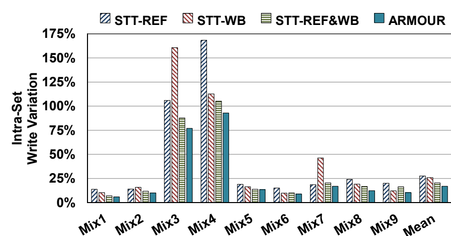

Fig. 12: Intra-Set Write Variation wrt. STT-REF. (Lower is better)

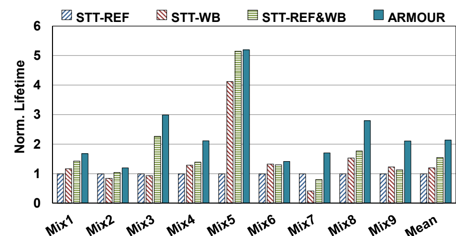

Fig. 13: Lifetime Improvement wrt. STT-REF. (Higher is better)

C. Energy Analysis

Figure 14 reports the energy usage normalized to the SRAM baseline. Due to fewer refreshes and writebacks, *ARMOUR* achieves a total energy saving of 66.7%, 14.4%, 22.3%, and 6.71% over SRAM, STT-REF, STT-WB, and STT-REF&WB, respectively. Figure 15 presents the energy delay product (EDP) normalized against the SRAM baseline. The EDP gains by name over SRAM, STT-REF, STT-WB, and STT-REF&WB are 74.5%, 26.3%, 41.6%, and 13% respectively. Basically, the significant reduction in writebacks and refreshes by *ARMOUR* improves the EDP.

D. Partitioned vs. Shared Cache Analysis

We evaluate two cache architectures: partitioned and shared on *ARMOUR*. In the partitioned approach, the cache is partitioned across different cache ways based on the core types. For instance, *ARMOUR*_8C_8G represents eight ways of lower retention (i.e., 20 ms and 50 ms) allocated to GPU applications, whereas eight ways of higher retention (i.e., 100 ms and 500 ms) are allocated to CPU applications. Similar, with *ARMOUR*_10C_6G, four ways of 20 ms and two ways of 50 ms are reserved for GPU, and two 50ms ways, four 100 ms ways, and four 500 ms ways are reserved for the CPU. In the partitioned approach, the search direction for an invalid entry during allocation is the same as given in Figure 4. However, if an invalid entry is not present within the partition assigned to the respective core, then a non-MRU block is evicted. Whereas, with the shared approach, the multi-retention cache is shared irrespective of the core type (same as the analysis in Sec. V-A, Sec. V-B, and Sec. V-C). In particular, any core can allocate its block to any multi-retention partition based upon the availability of the first invalid entry in their search direction. Figure 16 presents the comparative analysis of EDP for a statically partitioned cache vs. a shared cache with *ARMOUR*. As evident from Figure 16, the shared cache outperforms the partitioned cache. This is due to better cache space utilization for the CPU blocks. Additionally, the temporal locality-based refreshes and write backs of the GPU blocks ensure that the cache holds mostly live blocks.

VI. State-of-the-Art

There exists plenty of prior techniques aimed at reducing interference in the shared LLC for CPU-GPGPU workloads. OSCAR, proposed by Zhan et al. [9], exploits a dense STT-RAM cache architecture to reduce LLC contention, and optimize cache traffic by prioritizing CPU packets over GPU packets. Holey et al. [10], reduced cache contention by directly accessing the off-chip memory, while bypassing the LLC, in a GPU based system, by taking into account the available thread level parallelism of GPU workloads that are latency tolerant. Rai et al. [11] proposed a data criticality aware scheduling algorithm for GPU cache accesses to accelerate the workloads.

Another group of prior art tries to exploit the benefits of a multi-retention LLC with homogeneous systems. Dynamic refresh scheme (DRS) based techniques, like CacheRevive [18], first writes the cache blocks to a buffer when they expire and then writes back the blocks to the cache. However, CacheRevive limits the number of refreshes to only MRU blocks. The overhead with DRS-based schemes is the extra energy and performance aggravation due to data movement between the cache and buffer. To overcome this, MirrorCache [20] employs an extra cache to avoid regular data movement for refreshes, but it negates the cell density advantage of the NVMs. Some compiler-based techniques attempt to reduce the number of refreshes [21], [22]. Along with the DRS-based approaches, some studies propose writeback-based techniques [8], [23], [24]. One such approach, HALLS [8], writes back the blocks on expiry. HALLS improves the runtime EDP with a training-based mechanism in a set of multi-retention multiple virtual

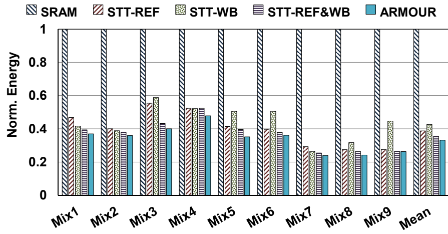

Fig. 14: Normalized energy wrt. SRAM.

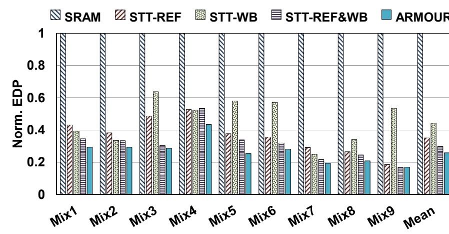

Fig. 15: Normalized EDP wrt. SRAM.

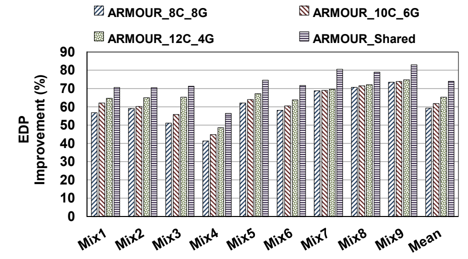

Fig. 16: EDP comparative analysis of partitioned caches vs shared cache. (Higher is better)

bank setup. Based on the training analysis, HALLS binds the application to one particular virtual bank. However, HALLS do not consider the dynamic changes in write accesses across execution phases of individual applications.

***ARMOUR* vs. Prior arts:** Most of the prior works tackle the interference in a shared SRAM LLC for heterogeneous systems. Although, OSCAR [9] considers an NVM cache, this work is limited to network optimizations and uses the NVM cache to reduce capacity misses. To the best of our knowledge, *ARMOUR* is the first work that employs a multi-retention NVM LLC by considering the lifetime of cache blocks in a CPU-GPU-based heterogeneous system. Prior techniques evaluate multi-retention caches only for homogeneous systems, while a few of might have limited applicability for heterogeneous systems with their vastly different cache characteristics (as shown in Sec. II). This makes meaningful comparisons with earlier techniques challenging to perform.

VII. Conclusion

We consider a multi-retention STT-RAM-based LLC and analyze the cache lifetime and interference of CPU-GPGPU workloads in a heterogeneous platform. Our analysis shows that the cache lifetime of CPU blocks are higher than for GPU blocks, but the lifetime of both types of cache blocks drop significantly when executing both workloads concurrently. Still, CPU blocks show more extended cache residency than their GPU counterparts. Hence, to mitigate the power-performance overheads of an STT-RAM based LLC, we propose *ARMOUR*, a hybrid mechanism that refreshes CPU blocks and writes back GPU blocks when they expire. To reduce the refresh overhead, *ARMOUR* selectively refreshes CPU blocks based on their temporal locality and retention zone. *ARMOUR* also prudentially selects and proactively evicts dead CPU blocks to improve LLC utilization. Our evaluation shows significant reduction of up to 28.9% in CPI with up to 51% reduction in MPKI compared to an 8 MiB STT-RAM based LLC, integrated in a heterogeneous system. *ARMOUR* also achieves an EDP gain of 74.5% over an iso-area SRAM LLC.

Acknowledgment

This work was supported in part by a *Marie Curie Individual Fellowship (MSCA-IF), EU (Grant Number 898296)*, and in part by a *EPSRC grant EP/V028154/1 to the University of Edinburgh*.

References

[1] W. Gomes, "Meteor Lake and Arrow Lake: Intel Next Gen 3D Client Architecture Platform with Foveros," in *Hot Chips*, 2022.

[2] D. Apalkov *et al.*, "Spin-transfer torque magnetic random access memory (STT-MRAM)," *ACM JETC*, 2013.

[3] M. K. Qureshi *et al.*, "Phase change memory: From devices to systems," *Synthesis Lectures on Computer Architecture*, 2011.

[4] R. Bez *et al.*, "Introduction to flash memory," *Proceedings of the IEEE*, 2003.

[5] M. V. Beigi and G. Memik, "TAPAS: temperature-aware adaptive placement for 3D stacked hybrid caches," in *MEMSYS*, 2016.

[6] Z. Sun *et al.*, "STT-RAM cache hierarchy with multiretention MTJ designs," *IEEE TVLSI*, 2014.

[7] ——, "Multi retention level STT-RAM cache designs with a dynamic refresh scheme," in *MICRO*, 2011.

[8] K. Kuan and T. Adegbija, "HALLS: an energy-efficient highly adaptable last level STT-RAM cache for multicore systems," *IEEE TC*, 2019.

[9] J. Zhan *et al.*, "OSCAR: Orchestrating STT-RAM Cache Traffic for Heterogeneous CPU-GPU Architectures," in *MICRO*, 2016.

[10] A. Holey *et al.*, "Performance-energy considerations for shared cache management in a heterogeneous multicore processor," *ACM TACO*, 2015.

[11] S. Rai and M. Chaudhuri, "Using criticality of GPU accesses in memory management for CPU-GPU heterogeneous multi-core processors," *ACM TECS*, 2017.

[12] J. L. Henning, "SPEC CPU2006 Benchmark Descriptions," *SIGARCH Comput. Archit. News*, 2006.

[13] S. Che *et al.*, "Rodinia: A benchmark suite for heterogeneous computing," in *IISWC*, 2009.

[14] ——, "Pannotia: Understanding irregular gpgpu graph applications," in *IISWC*, 2013.

[15] J. Power *et al.*, "gem5-gpu: A heterogeneous CPU-GPU simulator," *IEEE CAL*, 2015.

[16] Z. Sun *et al.*, "Multi retention level STT-RAM cache designs with a dynamic refresh scheme," in *MICRO*, 2011.

[17] S. Arcaro *et al.*, "Integration of STT-MRAM model into CACTI simulator," in *IDT*, 2014.

[18] A. Jog *et al.*, "Cache revive: Architecting volatile STT-RAM caches for enhanced performance in CMPs," in *DAC*, 2012.

[19] J. Wang *et al.*, "i2WAP: Improving non-volatile cache lifetime by reducing inter- and intra-set write variations," in *HPCA*, 2013.

[20] K. Kuan and T. Adegbija, "MirrorCache: an energy-efficient relaxed retention L1 STTRAM cache," in *GLS-VLSI*, 2019.

[21] K. Qiu *et al.*, "Refresh-aware loop scheduling for high performance low power volatile STT-RAM," in *ICCD*, 2016.

[22] Q. Li *et al.*, "Compiler-assisted refresh minimization for volatile stt-ram cache," *IEEE Transactions on Computers*, 2015.

[23] M. Baranwal *et al.*, "DAMUS: Dynamic Allocation based on Write Frequency in MUlti-Retention STT-RAM based Last Level Caches," in *ISQED*, 2021.

[24] S. Agarwal and S. Chakraborty, "ABACa: access based allocation on set wise multi-retention in STT-RAM last level cache," in *2021 ASAP*, 2021.