2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture

Density Tradeoffs of Non-Volatile Memory as a Replacement for SRAM based Last Level Cache

Kunal Korgaonkar[‖]*, Ishwar Bhati[§], Huichu Liu[†], Jayesh Gaur[†], Sasikanth Manipatruni[‡],
Sreenivas Subramoney[†], Tanay Karnik[†], Steven Swanson[‖], Ian Young[‡] and Hong Wang[†]
[‖]*University of California, San Diego*
(kkorgaon@ucsd.edu,swanson@cs.ucsd.edu)
[§] *Intel Labs, Intel;* [†]*Microarchitecture Research Lab (MRL), Intel*
(ishwar.s.bhati, huichu.liu, jayesh.gaur, sreenivas.subramoney, tanay.karnik, hong.wang)@intel.com
[‡]*Components Research (CR), Intel*
(sasikanth.manipatruni, ian.young)@intel.com

***Abstract*—Increasing the capacity of the *Last Level Cache* (LLC) can help scale the memory wall. Due to prohibitive area and leakage power, however, growing conventional SRAM LLC already incurs diminishing returns. Emerging *Non-Volatile Memory* (NVM) technologies like *Spin Torque Transfer RAM* (STTRAM) promise high density and low leakage, thereby offering an attractive alternative for building large capacity LLCs. However these technologies have significantly longer write latency compared to SRAM, which interferes with reads and severely limits their performance potential. Despite the recent work showing the write latency reduction at NVM technology level, practical considerations like high yield and low bit error rates will result a significant loss of NVM density when these techniques are implemented. Therefore, improving the write latency while compromising on the density results in sub-optimal usage of the NVM technology.**

In this paper we present a novel STTRAM LLC design that mitigates the long write latency, thereby delivering SRAM like performance while preserving the benefits of high density. Based on a light-weight learning mechanism, our solution relieves LLC congestion through two schemes. Firstly, we propose *write congestion aware bypass* that eliminates a large fraction of writes. Despite dropping LLC hit rates which could severely degrade performance in a conventional LLC, our policy smartly modulates the bypass, overcomes the hit rate loss and delivers significant performance gain. Furthermore, our solution establishes a *virtual hybrid cache* that absorbs and eliminates the redundant writes, which otherwise might be repeatedly and slowly written to the NVM LLC. Detailed simulation of traditional SPEC CPU 2006 suite as well as important industry workloads running on a 4-core system shows that our proposal delivers on an average 26% performance improvement over a baseline LLC design using 8MB STTRAM, while reducing the memory system energy by 10%. Our design outperforms a similar area SRAM LLC by nearly 18%, thereby making NVM technology an attractive alternative for future high performance computing.

***Keywords*-sttram; llc; non-volatile;**

I. Introduction

Despite advances in CPU architecture and DRAM memory technology, the memory wall continues to remain a bottleneck to application performance. The ever growing working set of applications exacerbates this problem even further. Increasing the capacity of the Last Level Cache (LLC) can help scale the memory wall. However, traditional SRAM based LLCs have limited density and high leakage power. Hence increasing the capacity of such LLCs is prohibitive in area, cost and power.

Emerging Non-volatile memories (NVM) based on technologies like Spintronics (e.g. Spin-Transfer-Torque (STT) RAM) and Resistive RAM offer energy efficiency improvement for normally-off computing [39], [40]. These memories also offer higher density and lower leakage power over SRAMs [27], [25], and hence are an attractive alternative to build large LLCs [1], [2], [3]. However, these technologies often suffer from long write latency that can degrade performance by causing write induced interference to subsequent critical reads. Additionally, the long latency of writes puts pressure on the request queues and the resultant congestion at the LLC can stall the core. As a result, LLCs based on NVM technologies underperform when compared to traditional SRAM LLCs for high performance applications.

Many techniques in devices, circuits and architecture have been proposed in both industry and academia to mitigate the performance impact of write latency as well as reduce write energy [37], [38], [31], [41], [42], [28], [1], [2], [3]. For instance, spin-hall-effect memory is another spintronic memory candidate that can reduce the write latency but at the cost of lower density [37], [38]. While technologists have been exploring material and device options to reduce the performance gap between emerging memories and SRAMs, circuit and architectural techniques have also been investigated. The write latency can be reduced by upsizing the write access transistors or using differential bits [31], [41] at the circuit level. However, these will result in higher overall power and lower density [25], [31], [30], thereby negating the density advantage offered by the emerging NVM technologies.

Another approach to reduce write latency is by relax-

*The author contributed to the work as an intern with Microarchitecture Research Lab (MRL), Intel India

2575-713X/18/$31.00 ©2018 IEEE
DOI 10.1109/ISCA.2018.00035

ing NVM retention to enable low-energy and fast write but degrading reliability [44], [42], [29]. From industrial perspective, high performance on-chip NVM LLCs need to meet the stringent reliability requirement for high volume manufacturing (e.g. error rate less than 10^{-9} at temperature of 85C and above [43]). For example, to ensure correct write in some emerging memory circuit prototypes [42], write-verify operation is required which results in higher effective write latency. Lastly, some recent architectural techniques [1], [2], [3] propose intelligent ways to reduce the number of writes happening in NVM LLC. Our results and analysis show that even after applying these techniques there remains significant performance gap from an NVM LLC solution that can fully mitigate write latency impact while maintaining high density.

To summarize, multiple device and circuit techniques have been proposed to reduce NVM write latency, however these techniques either reduce LLC density or make NVM LLCs unreliable for industry adoption. Therefore, we need low cost architecture techniques to overcome the expensive writes without sacrificing the high density advantages the NVM technology offers. In this paper we propose new architecture for NVM LLCs that dynamically observes the write congestion at the LLC and mitigates the performance impact of long latency writes. Specifically we make the following contributions:

- We make the key observation that for NVM LLCs it is possible to trade-off some drop in LLC hit rate in order to aggressively bypass writes and still gain overall performance. To accomplish this, we propose write congestion aware bypass (WCAB) that uses a dynamic learning mechanism in order to achieve an optimal bypass fraction in a given phase of execution, that would best balance the conflicting goals of relieving LLC congestion (through aggressive write bypass) and maintaining high LLC hit rates. We should note that WCAB is very different from traditional LLC bypass schemes where the motivation for bypass is to improve LLC hit rate by bypassing writes that have low priority of future reuse. WCAB, on the other hand, may end up reducing LLC hit rates while still delivering overall higher performance.
- We also observed that a lot redundant writes add to the LLC write traffic. We hence propose Virtual Hybrid Cache (VHC) that uses a portion of the L2 SRAM and the NVM LLC to preserve cache lines that frequently make trips between the L2 and the LLC. By intelligent placement and replacement management of cache lines, VHC eliminates a signifant fraction of writes.
- We perform detailed simulations and evaluate our policies on different LLC architectures (inclusive and exclusive) and evaluate the sensitivity of our proposals to varying NVM write latency and capacity. We show that, our policies gain significant performance at all configurations, while improving overall memory subsystem energy of the baseline system.
- Finally, we perform a thorough analysis of the trade-off between the NVM write latency and density and show that a combination of circuit techniques to reduce write latency coupled with our architecture provides the best trade-off between density and latency for an NVM LLC design. By mitigating the long write latency, our proposals help the NVM LLC outperform a similar area SRAM LLC by nearly 18%, thereby enabling large NVM LLCs in the future that can deliver an SRAM like performance but at a significantly lower area cost.

II. Background and Motivation

In this section, we first overview the characteristics of a representative NVM technology namely the Spin Torque Transfer RAM or STTRAM. We will then show that although STTRAM LLC can potentially provide significant capacity benefits owing to its high density, the interference caused by high latency writes makes it perform poorly compared to existing SRAM LLCs. This motivates the need for architectural techniques to mitigate the write latency in STTRAM LLC.

A. STTRAM based LLC

STTRAM offers density advantages over conventional SRAM as well as fast read and write time compared to other NVM technologies [27]. STTRAM utilizes a magnetic tunnel junction (MTJ) [25], [4], which is composed of a thin insulator layer sandwiched between a fixed ferromagnetic layer (polarization reference layer) and a free layer, to store the spin orientations as memory states (logic 0 or 1). The control of the MTJ states is through an access transistor, where the current flow through the MTJ can generate a spin torque based on the current direction and switch the magnetization direction in the free layer. The parallel and antiparallel magnetization states of the free layer (with respective to the reference layer) result in low resistance and high resistance states of the MTJ.

The main challenge for STTRAM based near logic memory application (e.g. LLCs) is its high write latency and high write energy compared with SRAM. This is due to the high thermal stability factor (Δ) in STTRAM design, which is required not only for reliable operation but also for longer retention time to avoid refresh overhead. A typical thermal stability value of 60kT (k is the Boltzmann constant, T is the temperature) or up is desired [43]. In addition, for high yield requirement, a minimum design margin up to 6σ is required (failure probability of 10^{-9}) [43]. The write time of STTRAM bitcell increases with the thermal stability and 6σ design corner requires 2×-3× higher write time compared to low margin mean value corner [43]. At 6σ design corner and 60kT thermal stability, a bitcell write time of more than 50ns is required [43]. Furthermore, retention time for a

given thermal stability value degrades with temperature. At a realistic operation temperature of 85C or above, higher thermal stability (60kT or higher) is needed to maintain non-volatility. Therefore, long write latency of 10 to 50 ns is required for STTRAM to meet the high reliability LLC design requirement for industry adoption.

Although the write latency issue has been addressed in prior works, such as reducing the write pulse time [28], tailoring the access transistor sizes [37], [38] and relaxing the non-volatility with refreshing techniques [29], these techniques come at the cost of higher error rates and implementation overheads, and become even more challenging with advanced logic node scaling and increase in process variation [25], [31], [30], [43]. In this work, we focus on the architectural solutions which can meet industrial design requirements, and explore the design space of density and write latency trade-offs for STTRAM LLCs in Section VI-H. We assume a write error rate (WER) less than 10^{-9} with SRAM-like ECC overhead and a thermal stability of 60kT.

B. *Write Latency vs. Cache Capacity*

Figure 1 shows the performance gains when the capacity of an SRAM LLC is increased from 4 MB to 16 MB, while the latency is kept constant, for a 4 core system [1]. On an average (geometric mean of all the 64 benchmarks described in Section V), 2× (8 MB) and 4× (16 MB) capacity increase brings 15% and 23% performance improvement over the 4MB baseline. These results clearly show significant performance potential of increasing the LLC capacity, which can be achieved by building the LLC using high density STTRAM technology.

Figure 2 shows the performance impact of increasing the write latency for an 8 MB STTRAM-LLC, compared to a 4 MB SRAM baseline. For this result we assume that STTRAM is 2× denser than SRAM and hence the 8 MB STTRAM LLC has the same area as the 4 MB SRAM LLC. We increase the write latency from 0 ns to 20 ns. As is evident from Figure 2, as the write latency of STTRAM increases, performance drops rapidly as long latency writes interfere with reads. A hypothetical SRAM-like write latency STTRAM LLC (+0 ns) would gain 15% over the 4 MB SRAM baseline, which drops to 13% and 5% as the write latency is increased to 5 ns and 10 ns respectively. But when the write latency is further increased to 20 ns, all the capacity benefits are lost, and overall the performance drops by 15% when compared to the 4 MB SRAM baseline. As discussed earlier in Section II-A, lowering the write latency reduces the density advantages of STTRAM, and is not desirable.

In this paper, we hence explore architecture techniques to mitigate the long write latency and build an STTRAM LLC that can deliver SRAM like performance. We will first discuss briefly the existing solutions that have been proposed for STTRAM based LLC in Section III. Thereafter, we will describe our proposed architecture in detail in Section IV.

[1]Simulation framework and workloads are discussed in Section V

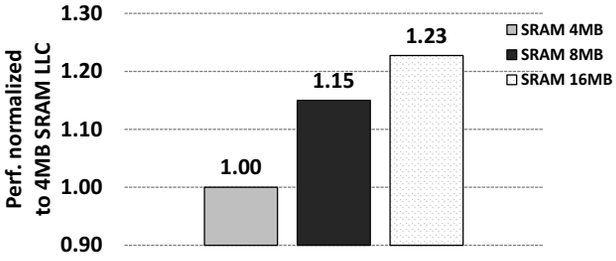

Figure 1: Performance benefit of higher capacity LLC

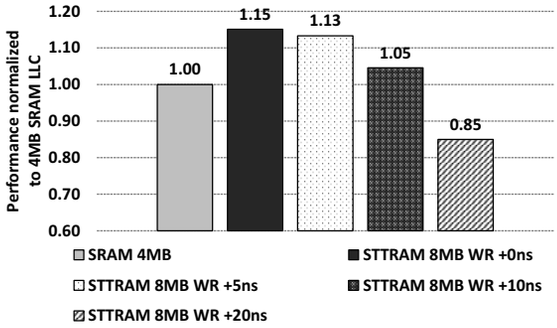

Figure 2: Impact of high STTRAM write latency

C. *Sources of Writes in the LLC*

LLC architecture can be Exclusive, Inclusive, or Non-Inclusive [10]. Inclusive LLC duplicates every line in the inner level caches in the LLC. This helps simplify coherence flows, although it reduces capacity. However, as the L2 size continues to grow, LLCs are also being designed as exclusive [23], [5], [2], primarily because of the capacity benefits due to not replicating cache lines between the L2 and the LLC. Sometimes strict exclusion may not be possible for design simplicity, and such architectures are called as non-inclusive [2].

Different inclusion properties lead to different sources of write traffic at the LLC. Writes to an inclusive LLC originate from fills coming from the memory due to LLC read misses and dirty victims from L2. In exclusive LLC, both clean and dirty L2 victims are written to the LLC, while memory fills only happen to the L2. Write traffic in the exclusive LLC can be substantially higher as cache lines are deallocated from LLC on hits and are always written back when evicted from L2, irrespective of dirty or clean. On the other hand, in an inclusive LLC, clean L2 victims are dropped, as a copy of these cache lines is already present in the LLC. As a result of the higher write volume, designing an exclusive STTRAM LLC is more challenging, though exclusion brings in higher effective cache capacity. In this paper we will evaluate our proposal on both inclusive and exclusive LLC architectures.

III. Related Work

A large body of work [14], [12], [7], [5], [15] has addressed the capacity management problem in SRAM based LLCs. Managing the demand for LLC bandwidth has been addressed in the context of SRAM LLC [10], [5] mainly to reduce on-chip traffic and power. NVM LLCs can also benefit from these techniques. However, NVM has long write latency and therefore needs solutions to mitigate the LLC congestion arising because of these long latency writes. Some of the solutions explored in the past to this problem include [9], [8], [1], [2], [3], [11], [6]. These solutions propose heuristics to either stall writes or completely bypass the writes. These techniques eliminate some of the writes while making sure there is no loss in hit rate because of the bypass. Bypassing writes helps relieve some of the LLC congestion. However increasing the aggressiveness of bypass to improve LLC congestion further is very risky as it can reduce hit rates significantly and severely degrade performance.

Hybrid cache [1] proposes to convert some portion of the NVM LLC into an SRAM. Frequently written to cache lines are allocated in the SRAM portion to reduce writes to the primary NVM portion. However, as SRAM has lower density than NVM, this results in overall lower capacity than a pure NVM based LLC and hence bounds the performance of capacity sensitive applications. The Hybrid cache proposal uses a predictor for placement and migration between the SRAM portion and NVM to reduce write congestion.

LAP [2] was proposed for exclusive STTRAM LLCs. It duplicates cache lines, that make frequent trips between the LLC and the L2, in the LLC. However, this duplication results in LLC capacity loss limiting the performance potential. Moreover a significant fraction of writes to the LLC originate from dirty writebacks from the L2 which cannot be addressed by this scheme. We compare the Hybrid cache proposal and LAP with our architecture in Section VI.

Optimizations have also been explored in the context of memories built using NVM technology by [6], [33], [34]. The focus of these works is primarily to improve energy efficiency and resilience of NVM memory. FIRM [35] proposes to reduce memory bus turnarounds and utilizes bank level parallelism to improve fairness and performance in NVM based persistent main memory systems. A recent work, OSCAR [4], tries to solve high bandwidth demand on on-chip network of shared GPU-CPU NVM based LLC.

Overall, although past efforts have shown improvement over baseline NVM LLCs, none have explored techniques that delicately balance the conflicting goals of managing contention while retaining the benefits of higher capacity. Through a comprehensive and synergistic set of architectural techniques we show a path to get to performance that is within striking distance of that of an LLC with SRAM-like write latency and of the same capacity.

IV. Proposed LLC Architecture

Our proposal is composed of two main components - Write Congestion Aware Bypass and Virtual Hybrid Cache, which are explained in the subsequent sections.

A. *Write Congestion Aware Bypass*

One of the ways to reduce LLC congestion is to bypass some of the writes at the LLC. Many bypassing schemes have been proposed in the context of SRAM LLC [5], [12]. Traditional LLC bypass schemes employ a dead block predictor to classify lines which are less likely to be accessed again and therefore are not filled in the LLC. Bypassing such dead lines retains more live (more likely to be accessed again) cache lines in the LLC and therefore improves hit rate.

In the case of NVM LLC, bypassing not only improves hit rate but also reduces write congestion, thereby having a greater impact on performance. Bypass policies proposed in [1], [3] adapt SRAM bypass schemes to NVM LLC and demonstrate superior performance. Unfortunately since these bypass schemes are inherently designed for improving hit rates, the amount of bypass accomplished by them is fairly limited. Moreover, as the LLC capacity grows, the fraction of writes that will not be reused drops as larger capacity, enabled by high NVM density, allows more cache lines with large reuse distances to be retained. As a result, more aggressive bypass policies are needed to relieve the LLC congestion because of long latency writes. Unfortunately just increasing the aggressiveness of bypass can significantly affect LLC hit rates, thereby negating the capacity benefits offered by the NVM LLC.

We hence need to strike a balance between the conflicting goals of bypassing writes to relieve LLC congestion and the need to minimize the hit rate loss in the LLC because of bypass. To understand this trade-off better we first try to analyze our workloads and develop an understanding of the parameters that effect this trade-off. Based on this learning, we will then propose a novel algorithm that we call as the Write Congestion Aware Bypass (WCAB) in Section IV-A2.

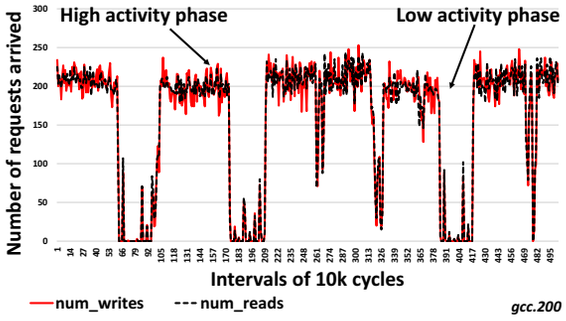

Figure 3: Request periods for a snippet of *gcc.200*

1) Write Congestion vs. Hit Rate: For NVM LLC, the bandwidth delivered by the LLC depends on the write

latency as long write latency blocks an LLC bank from accepting future requests, while the write is being performed to the bank. Reducing the latency of the writes or the fraction of writes will improve the NVM LLC bandwidth, thereby reducing queuing latency at the LLC and improving performance. As write latency is fixed for a given configuration, we need to reduce the fraction of writes by write bypassing. Unfortunately write bypass will result in increased misses to memory, thereby impacting performance. The cost of bypass is dictated by the latency of the main memory and the fraction of write bypasses that will result in future memory reads (*liveness* of the application). Slow memories can increase the cost of a wrong bypass. Likewise application phases with high liveness will suffer more from wrong bypass.

Figure 4 shows the performance (left Y axis) and hit rate loss (right Y axis) for two representative benchmarks *bzip2.combined* and *gcc.g23*, as the bypass aggressiveness is increased for an SRAM LLC. Figure 5 shows the same graph when applied to an STTRAM LLC. In case of SRAM we clearly see that as the bypass aggressiveness is increased, the hit rate drops and overall performance degrades. This is expected because SRAM does not have any congestion and hence bypassing is detrimental to hit rate and consequently to performance. For STTRAM in *bzip.combined* we see that, like in the case of SRAM, the hit rate continues to drop as the bypass fraction is increased. Interestingly, despite of the hit rate loss, the performance initially increases as the bypass helps relieve LLC congestion. However beyond a point, the loss in hit rate outweighs the improvement in queuing latency and the performance eventually drops. Similar behavior is seen for the *gcc.g23* workload. This clearly matches our intuition that a we need to derive an optimal bypass that balances the conflicting goal of capacity management and reducing LLC congestion.

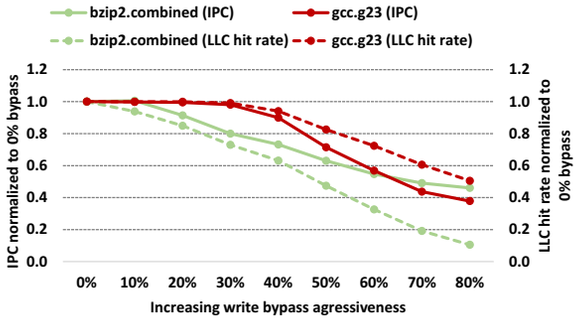

Figure 4: Impact of write bypass on SRAM LLC

To summarize, our goal is to find an optimal bypass that reduces the LLC queuing latency while minimizing the cost of bypass. Combining all the learning above, we can recapitulate that the optimal bypass depends on request bandwidth demand, fraction of writes, write latency of STTRAM, main memory latency and the liveness of the

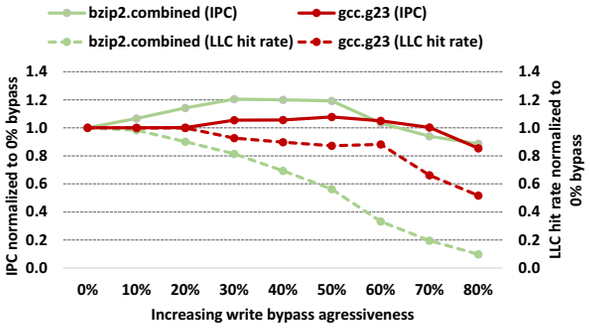

Figure 5: Impact of write bypass on STTRAM LLC

application. Of these, the latencies at the LLC and the memory are fixed for a given system design, whereas the liveness, write fraction and request bandwidth need to be learnt dynamically, for a given phase of execution of an application. We now describe our proposal, write congestion aware bypass (WCAB), that learns these parameters through a simple lightweight learning mechanism, and then uses it to modulate the fraction of bypass.

2) WCAB Algorithm: Our LLC controller models a Request Queue, which stores LLC requests as they arrive. Requests are taken out of the request queue in order and sent to the LLC. Misses and victims may need subsequent passes through the request queue.

Learning LLC congestion: If the request bandwidth demand is higher than the bandwidth supplied, the request queues will have a high occupancy. Hence the occupancy of the request queue can be used as an indication for LLC congestion. We should note that the request queues can also have high occupancy because of a large number of pending memory requests. To differentiate from this scenario, we also look at the number of writes pending in the request queue. If the occupancy of the request queue is high and the fraction of writes is above a threshold (*write_th*), we infer it to be a scenario for LLC congestion. We keep a running counter that counts the average number of writes as well as the average occupancy seen in previous K windows (*int_write_occ*). Based on the values of these counters we decide the amount of bypass that needs to be done in the current window. Windows of high write congestion will have more bypasses, and those of low congestion will have fewer WCAB induced bypasses. A running average also makes sure that small spikes in bandwidth demand are not treated as phases of LLC congestion and unnecessary bypass is prevented. We found $K = 5$ and interval window length of 100K cycles, to be the best suited for our work.

Learning the cost of bypass: Write bypass will reduce hit rates and increase traffic at the memory. To estimate the cost of bypass, WCAB learns the reuse probability (liveness) of a given write that needs to be allocated at the LLC. We allocate small number (32) of observer sets in the L2 and the LLC, similar to [5], [12] for learning. A table indexed by

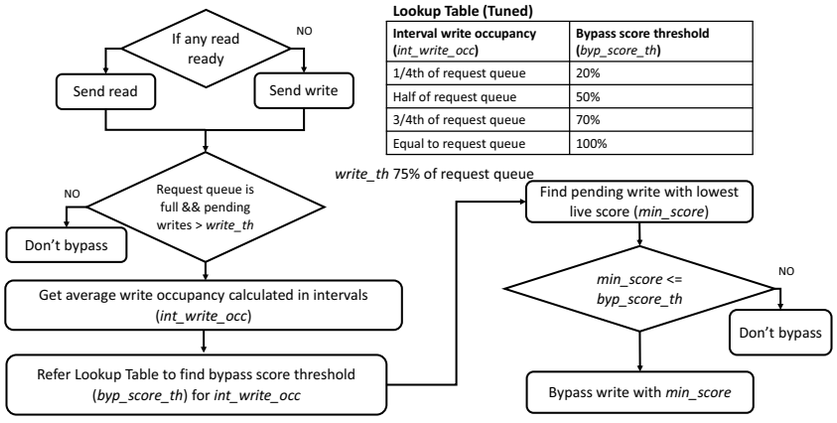

Figure 6: Detailed flowchart of the WCAB algorithm.

hashed *L2 access PCs* (Instruction address that last accessed the cache line in the L2) is maintained at the L2. The table has four 10b signed, saturating counters corresponding to the liveness buckets $0-20\%, 21-50\%, 51-70\%$ and $71-100\%$ (we experimented with more buckets, but did not see much sensitivity). We define liveness as the fraction of writes corresponding to an access PC that are recalled from the LLC. So for instance, 20% liveness would mean that out of every 100 evictions from an access PC in the observer sets that were filled to the LLC, 20 were hit by subsequent reads. The observer sets are chosen similar to [5], [12].

Whenever a cacheline is evicted from the L2 observer set we decrement the liveness counters corresponding to its access PC. Note similar to the sampler of [12], we maintain partial-PC tags for lines in observer sets. If the line is recalled from the LLC in the future (LLC hit), we increment the liveness counters of the access PC. For example, for the $21-50\%$ liveness counter, on eviction we decrement by 2 and on a recall we increment the counter by 10 (corresponding to 20%). If this counter is positive, it will mean that the liveness was at least 20%.

Whenever a cacheline is filled in the non-observer sets of the L2, we check the live counters corresponding to the eviction and assign it a 2 bit live score. This live score is stored for each L2 cacheline. The live score is basically the highest liveness bucket that was positive for its access PC. For instance if both $71-100\%$ and $51-70\%$ liveness counters are positive, it is assigned the $71-100\%$ live score. Since we are only tracking four buckets, we need just 2 bits. On eviction, the live score is sent along with the write to the LLC and is stored in the LLC request queue. We should note that the default live score for all writes (when none of the counters are positive) corresponds to the $0-20\%$ liveness bucket.

Write bypass: WCAB checks the LLC congestion counters and based on the congestion decides a target live score (*byp_score_th*). Writes that have a live score below the target live score are bypassed. The bypasses are done as long as the occupancy of the request queues do not drop below a threshold, that is, bypasses are done as long as there is write congestion. If the amount of congestion is high, WCAB has a higher target live score. If the congestion is low, we do conservative bypassing by reducing the target live score. WCAB always ranks pending writes in order of their *live scores*. Writes with low live scores are bypassed before writes with high live scores. This helps retain writes that had higher likelihood of producing hits in the future, thereby minimizing the hit rate loss because of bypassing potentially *live* writes. Figure 6 depicts the WCAB algorithm in more detail.

Tuning the thresholds: The amount of bypass and hence the corresponding thresholds depend on the write latency of the LLC and the latency of the main memory. Therefore the thresholds need to be tuned for a given system. We ran various simulations for our target system to identify the best average thresholds. Figure 6 also shows (in the table on top right) the tuned parameters for our target system described in Section V.

B. Virtual Hybrid Cache

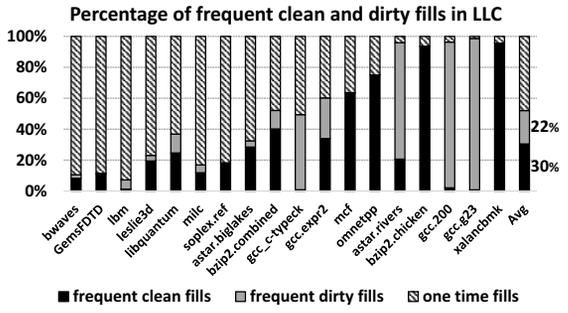

Figure 7: Frequent clean and dirty fills in LLC

In inclusive LLCs, many cache lines are repeatedly recalled from the LLC, modified in the L2, and written back to the LLC. We call such cache lines that frequently move

between the LLC and the L2 as frequent dirty fills. In exclusive LLC, a read hit deallocates the cache line and moves it to the L2. On an L2 eviction, this line has to be written back to the LLC, irrespective of whether it was clean or dirty. A subsequent hit will move this line back to the L2. Therefore exclusive caches also tend to have significant amount of frequent clean fills. Figure 7 shows the average (arithmetic mean) percentage of such clean and dirty frequent fills for an exclusive LLC. We see that such fills contribute to a significant portion of the overall writes and can be as high as 50% of the total writes at the LLC.

To reduce frequent fills at the NVM LLC, we propose a solution that we call as the Virtual Hybrid Cache (VHC). Unlike true hybrid caches proposed in [1] that use a dedicated SRAM cache for such frequent fills, the VHC simply borrows some capacity from the L2 and the LLC. To reduce writebacks because of dirty L2 evictions, VHC retains cachelines, that create frequent dirty fills, in the L2 so that multiple L1 writebacks may merge in the L2. For exclusive caches, VHC duplicates some lines in the LLC for reducing frequent clean fills. A recent proposal called LAP [2], attempts to tackle the clean eviction problem with exclusive caches using a similar approach. However, unlike LAP, VHC minimizes the LLC capacity loss because of duplication, through smart optimizations in the LLC. We next describe these two components of VHC in more detail.

Write Merges in L2. To reduce frequent dirty trips in the LLC, the cache lines classified as getting frequently dirty are given preference to stay in the L2. To classify frequent dirty lines we used the predictor proposed in [1]. By keeping the frequent dirty lines in L2 for longer, many writes merge in the L2 without the need of making fills in the LLC. To implement this technique, we keep a count of how many frequent dirty lines are in each L2 set. If this count is lower than W ways, then the frequent dirty lines do not participate in victim selection. Otherwise, if the count is equal or higher than W, then all the lines, including frequent dirty lines, participate in victim selection. For our workloads and configurations, W of 2 performs the best. This means we end up dedicating up to 2 ways for frequent dirty lines. Note that this trades off some L2 hit rate as some ways are dedicated to frequent dirty lines and therefore clean lines are evicted to the LLC more often.

Relaxed Exclusivity. As described earlier in Section II, in exclusive LLCs even the clean victims from L2 are filled in the LLC, creating write congestion problem due to frequent clean fills in NVM LLC. One solution, as proposed in LAP [2], is to not deallocate lines from the LLC on a read hit thereby duplicating some lines between the L2 and the LLC. This will eliminate all the clean fills after their first trip. However, the disadvantage of such a scheme is the LLC capacity loss due to duplication. To minimize the hit rate loss from duplication, we propose the following optimizations.

- We retain the duplicated lines in the LLC closer to the LRU position so that the replacement policy in the LLC does not degrade. If a line moves from the LLC to L2 and is retained in the L2 for a long time, it will reduce effective capacity because of duplication for a long time. To prevent such lines from being duplicated in the LLC, we do not update the LRU on duplication. The LAP policy of [2] on the other hand, moves duplicated lines to the most recently used (MRU) position to reduce clean fills. As a result our policy reduces clean fills by a lesser amount as compared to LAP, but has a lower hit rate impact on the LLC.
- If a cache line is hit by a store request in the LLC, then there is no value in duplicating the line as it would be made dirty in the L2 and the LLC copy will be stale. Hence such lines are not duplicated in the LLC. However, there are many cases when a read request brings the line into the L2, and a subsequent store makes this line dirty in the L1. In such cases, as soon as the line get dirty in L1, we send a hint with the coherence packet to de-allocate its copy in the LLC, thereby reducing an unnecessary duplication.

Coherence in VHC is handled similar to the exclusive LLC. Exclusive LLCs typically keep a snoop filter, which stores owner core for all the cachelines filled in core caches. For a duplicated line between the L2 and the LLC, the corresponding L2 is stored as owner in the snoop filter. The LLC copy is not regarded as latest.

C. Area Overheads

For WCAB, 2 bit live score is stored for each L2 cacheline and assigned for every write to LLC. For 256KB L2 and 64 entry LLC queue, additional storage of around 1 KB per core is required. We use 256 entries in the hashed table (10 bits per PC) and 4 10 b counters. That needs an area of 1.5 KB. Duplication of lines in the L2 or L3 would need a bit to store the duplication in the L2. This needs 0.5 KB of L2 area. For observer sets (32), partial-PC tags account for another 0.5KB area. Overall our architecture adds an additional 3.5 KB of area per core. As compared to the 8 MB LLC and 256 KB L2, this constitutes less than 0.1% of the total cache area.

V. Evaluation Methodology

For our simulations, we model four dynamically scheduled x86 cores with an in-house modified version of the Multi2Sim simulator [20]. Each core is four-wide with 224 ROB entries and clocked at 4 GHz. The core microarchitecture parameters are taken from the Intel Skylake processor [21]. Each core has 32 KB, 8-way L1 instruction and data caches with latency of three cycles and a private 256 KB 8-way L2 cache with a round-trip latency of eleven cycles. In our SRAM baseline configuration, all the cores share a 4 MB, 4 banks, 16-way LLC with round-trip latency of twenty cycles. For STTRAM configurations, we increase LLC capacity to 8 MB (with 8 banks) and add an additional

20ns of write latency. We will sweep this write latency in Section VI-E2 and study the sensitivity of write latency to our policies.

Each core is equipped with an aggressive multi-stream stride prefetcher that prefetches into the L2 and LLC caches. The main memory DRAM model includes two DDR4-2400 channels (total bandwidth 38.4 GB/s), two ranks per channel, eight banks per rank, and burst length of eight. Each DRAM device has a 2 KB row buffer with 15-15-15-39 (tCAS-tRCD-tRP-tRAS) timing parameters. An additional ten-cycle I/O delay (at 1.2 GHz) is charged for each access to account for floorplan, board delays, etc. Writes are scheduled in batches to reduce channel turn-arounds.

We select 20 workloads from the SPEC CPU 2006 [19] and HPCG [22] benchmark suites. These are traces that have high L2 MPKI (more than 10) and will hence be sensitive to the LLC optimizations proposed in this paper. Table I lists these workloads along with their LLC MPKI. We first created 20 homogenous, RATE-4 traces (each core runs the same copy of the benchmark) for these workloads. In addition to these 20 homogeneous multi-programmed mixes, we prepare 44 four-way heterogeneous, multi-programmed mixes by randomly combining these 20 traces. In all these 64 multi-programmed mixes, each core simulates 250 million dynamic instruction and all the cores combined together simulate at least one billion instructions. Cores that finish early continue to run. We use weighted speedup [16] as the metric of performance.

Benchmarks	LLC MPKI
gcc.g23, xalancbmk, gcc.200, astar.rivers2, bzip2.chicken, gcc.c-typeck,omnetpp	0-10
libquantum, gcc.expr, leslie3d, bwaves, GemsFDTD, milc, mcf	10-20
soplex.ref, bzip2.combined, astar.rivers8, astar.biglakes, lbm, hpcg	20-100

Table I: Benchmark selection and characterization

VI. Simulation Results

In this section, we evaluate the performance of our proposals. We first describe our baseline LLC policy in Section VI-A and then summarize the performance gain of our features on top of this baseline for both exclusive and inclusive STTRAM LLC in Section VI-B. After that, we will present a detailed analysis of our features in Section VI-C, comparison with prior art (Section VI-D), sensitivity to different parameters like write latency, banking, memory bandwidth and latency (Section VI-E) and a brief energy analysis in Section VI-F. Further, in Section VI-G we present performance benefits of our techniques on a subset of industry workloads. Finally, in Section VI-H we compare different density and write latency options in STTRAM technology.

A. Baseline Bypass

We optimize the baseline LLC by deploying a PC based dead block predictor to bypass writes while improving the hit rate in the LLC. This policy is similar to the bypass policy proposed in DASCA [3]. This bypass predictor improves SRAM performance by 1% over an SRAM baseline that used the replacement and bypass schemes as proposed by [5]. We consider this predictor as prior art and employ it in the baseline of all the results in subsequent sections.

B. Performance Summary

1) Exclusive LLC: Figure 8 shows the performance gain of our proposal on top of an 8MB, 8 banks, exclusive STTRAM LLC baseline. Overall our proposal improves the performance (measured as geometric mean over all 64 traces) of the baseline STTRAM LLC design by 26%. The WCAB component of our policy gains 16% and VHC contributes additionally 10% more performance improvement. Several benchmarks gain significant performance. However *hpcg*, *lbm* and *astar.biglakes* do not show much response to our policies as these are streaming in nature and have very high LLC MPKI of 50, 28 and 27 respectively. Hence they are insensitive to LLC latency and bandwidth improvements.

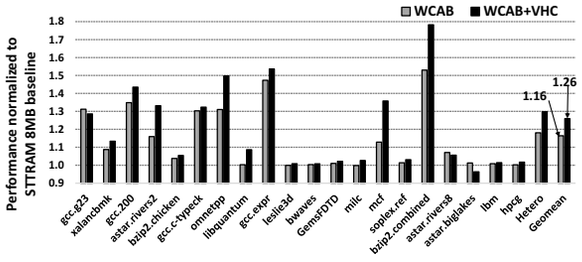

Figure 8: Performance in exclusive STTRAM LLC

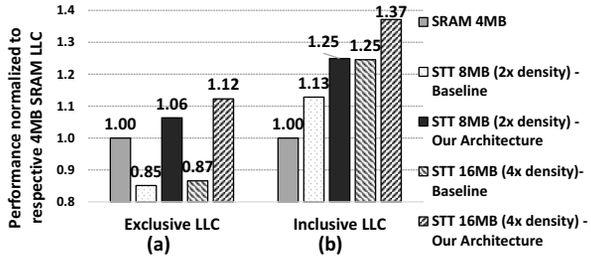

Figure 9: Performance compared to 4MB SRAM LLC

Figure 9(a) compares our policies with a similar area SRAM. For these results we assume two density points for STTRAM in comparison to SRAM. Overall for exclusive LLCs we see that 8MB (2X density) and 16MB (4X density) STTRAM, despite of a larger capacity, lose 15% and 13% performance as compared to the 4MB SRAM LLC. However with our proposals added to the STTRAM LLC, we gain

6% and 12% performance in 8MB and 16MB STTRAM respectively.

2) Inclusive LLC: Figure 10 shows the performance gain of our features on top of an 8MB, 8 banks, inclusive STTRAM LLC baseline. For the inclusive baseline our features gain a significant 11% overall performance. We should note that our performance gains were expected to be lower in case of inclusive LLCs because inherently inclusive LLCs have less write pressure as the clean writebacks from the L2 are simply dropped.

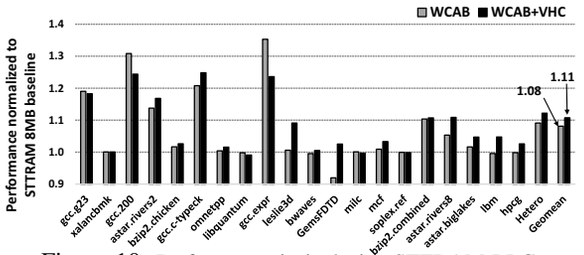

Figure 10: Performance in inclusive STTRAM LLC

Figure 9(b) compares our policies with a similar area SRAM. As described in Section II-C, inclusive LLCs see fewer writes, therefore the baseline STTRAM with 8MB and 16MB provide 13% and 25% performance gains. Our proposals further improve performance to 25% and 37% for 8MB and 16MB STTRAM respectively.

We now analyze our policies in more detail in subsequent sections. For sake of brevity, we will show results only on the exclusive LLC baseline.

C. Performance Analysis

1) WCAB: Figure 11 shows the gains delivered by WCAB for every trace used in our simulation (left Y axis). Also plotted is the miss rate difference from this baseline on the right Y axis. On an average, WCAB reduces the writes to the LLC by 11%, while increasing the miss rate by 9%. This trade-off helps it deliver performance. We should note that there are some benchmarks that lose because of WCAB. These benchmarks typically suffer a higher miss rate that is not offset by the improved congestion at the LLC. Overall WCAB performance ranges from -3% to +53%.

To verify the significance of PC based liveness, we changed WCAB bypass algorithm to not use liveness score. Instead it selects the oldest write for bypass when there is congestion. This scheme degrades the performance by 3% compared to our default WCAB algorithm using PC based liveness.

2) Virtual Hybrid Cache: VHC further improves the performance gain by 10%. Figure 12 shows the gains delivered by VHC on top of WCAB, for every trace. Also plotted is the difference in miss rate. Overall VHC reduces the number of writes to the LLC and hence helps reduce the aggressiveness of WCAB, thereby improving the overall hit rate. Some

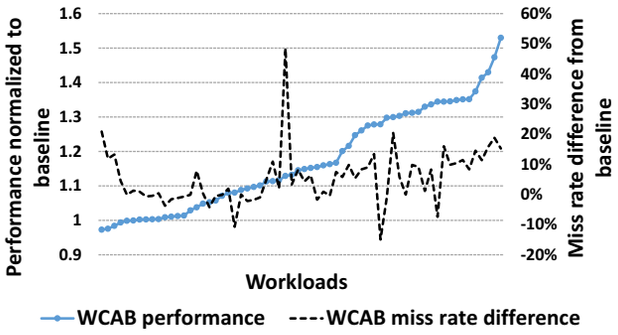

Figure 11: Performance vs. miss rate difference of WCAB.

losses seen with VHC are primarily because of additional hit rate loss as a result of duplication (that reduced the effective capacity of LLC + L2). VHC is able to eliminate nearly 40% of writes at the LLC, while increasing the miss rate further by 2.2%.

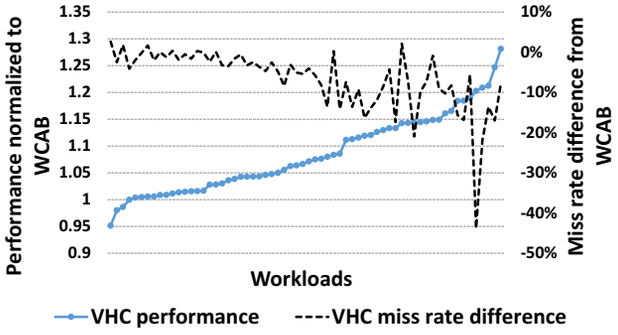

Figure 12: Performance vs. miss rate difference of VHC on top of WCAB.

D. Comparison to Prior Art

We compare our architecture to two existing policies namely the Hybrid Cache (as used in [1]) and LAP [2]. These two policies were discussed in Section III. Figure 13 compares the proposed architecture with these two existing policies. For the hybrid cache we used two configurations where the SRAM portion is $1/4^{th}$ and $1/8^{th}$ of the LLC capacity. If we assume STTRAM has 2x density as compared to SRAM, the two configurations will have have 4MB and 6MB of STTRAM respectively. On an average the best performing Hybrid cache improves the performance by 9%, whereas LAP improves the performance by 11%. On the other hand, the proposed architecture improves the average performance by 26%.

As discussed earlier, Hybrid caches result in an overall smaller capacity, as SRAM has a lower density than STTRAM. Our simulations show that the best performing Hybrid cache (with 2MB of SRAM and 4MB of STTRAM) has an average 11% higher miss rate, as compared to the STTRAM baseline. On the other hand the fraction of writes

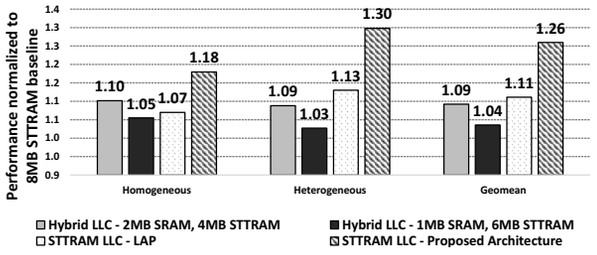

Figure 13: Performance comparison to prior art.

drop by 19%. The capacity loss tends to bring down the gains of Hybrid cache.

LAP shows a performance improvement of 11%. Since cache lines are duplicated by LAP there is a loss in cache capacity. To overcome this capacity loss, the proposed Virtual Hybrid Cache uses intelligent replacement optimizations as described in Section IV-B. Moreover, unlike LAP that cannot target dirty fills, VHC uses a small subset of the L2 cache to increase write merging and cut down on dirty fills. To show the advantages of the proposed optimizations by VHC we compare just the VHC component of our policy with LAP in Figure 14. As can be seen for several traces, VHC has a lower miss rate loss as compared to LAP.

As a standalone feature itself, VHC outperforms LAP by 2.6% on the same baseline. VHC has a 1.5% lower miss rate as compared to LAP, though LAP has 2.8% lower writes to the LLC. Overall our proposed architecture delivers 26% performance as compared to LAP that delivers 11% performance and Hybrid cache that delivers 9% performance gain for an 8MB STTRAM, having a 20 ns write latency.

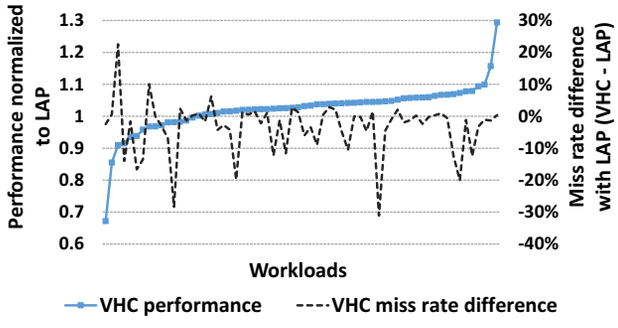

Figure 14: Performance and miss rate difference of VHC compared to LAP.

E. Sensitivity Studies

1) Sensitivity to LLC Banking: Banking the LLC can increase the overall LLC bandwidth and help mitigate the LLC congestion. However banking results in substantial area and power overheads because of duplication of peripheral circuit, interconnect etc. [18]. Also floorplan considerations are important to decide the number of LLC banks that are possible. Figure 15 shows the impact of our policy when the number of banks are increased in the baseline. For 8 banks, we gain 26% performance. With 16 banks (and assuming no area overheads of banking), the STTRAM baseline improves by 18%, and our policies improve the gain further by 15%. However banking costs area. A 10% overhead reduces the performance gain of 16 banks over 8 banks to 16%, whereas a 20% overhead would further reduce the performance gain to 15%. Our policies gain 12% and 11% performance on top of these area compensated banked baselines.

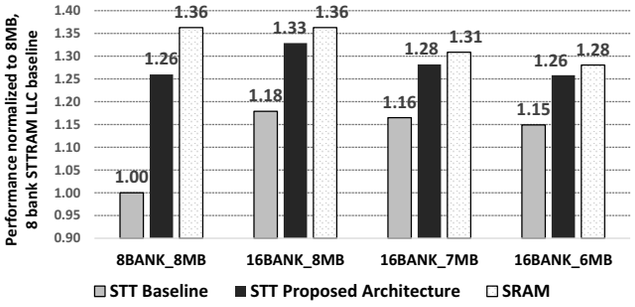

Figure 15: Impact of LLC banking

2) Sensitivity to write latency: Figure 16 shows the performance of our policies at different write latencies of an 8MB STTRAM LLC. Our policies gain 1%, 7%, 26% and 53% performance gains at STTRAM LLC write latency of 5ns, 10ns, 20ns and 40ns respectively. Write congestion problem is less severe at lower STTRAM write latencies and hence the sensitivity of our proposals increase as the write latency of the STTRAM is increased. However we should note that reducing write latency at the circuit level reduces the density of the STTRAM, and may not be desirable. We will evaluate this interesting trade-off in a separate section(Section VI-H).

Figure 16 also shows the performance gain of our policies on an 8MB SRAM (+0ns write latency), where there is no write congestion. As expected our policies overall degrade the performance of such a baseline by almost 2%. This shows the uniqueness of our problem space and our solution - conventional SRAM LLCs expectedly lose performance when we degrade hit rates, whereas for STTRAM LLCs we can actually lose some amount of hit rates as long as we can use that loss in order to improve LLC characteristics and gain overall higher performance.

3) Sensitivity to algorithm parameters: Our thresholds are dependent on the memory latency (the cost of bypass) and the write latency of the STTRAM LLC. Therefore these need to be tuned for a given system. We ran several experiments to arrive at the best possible thresholds. For our default system configuration the best thresholds are mentioned in Figure 6.

4) Sensitivity to LLC capacity: Figure 17 shows the performance delivered by our optimization at various LLC sizes by keeping the write latency constant at 20ns. For

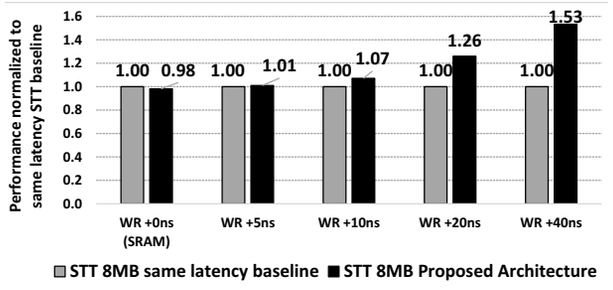

Figure 16: Impact of STTRAM write latency

each capacity point we also show the performance that an SRAM would deliver at the same capacity. Note that SRAM would need a much higher area than STTRAM for a given capacity, so this comparison is primarily to gauge the effectiveness of our policy in mitigating write latency. Overall we see that a similar capacity SRAM would deliver 20%, 36% and 42% performance on top of 4 MB, 8 MB and 16 MB STTRAM baseline respectively. However, with our architecture applied to the baseline STTRAM, we bridge this gap by almost 65-70% across the various capacity points. The remaining gap between SRAM and STTRAM can potentially be bridged by improving the bypass decisions of WCAB so that capacity is not sacrificed while writes are still bypassed. Also addressing dirty fills and write merging in the core can significantly bridge this gap.

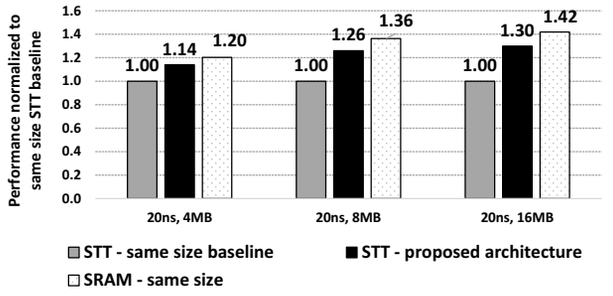

Figure 17: Our techniques vs same capacity SRAM

5) Sensitivity to Memory Bandwidth and Latency: We evaluated different main memory bandwidth and latency configurations with an 8 MB STTRAM LLC baseline. Compared to the default DDR-2400 memory, using DDR-1600 lowers performance by 2%, whereas higher bandwidth DDR-3200 improves by 1%. On the other hand increasing memory latency by 50% reduces performance by 4%, whereas reducing latency by 30% increases performance by 5%.

Figure 18 shows the performance delivered by our proposal at different memory bandwidth and latency points. Expectedly our performance gains increase when the baseline memory bandwidth is increased. This is because higher bandwidth memory will be able to absorb the extra pressure our write bypassing scheme creates at the memory. Our proposal gains 27% when the baseline uses a higher bandwidth DDR4-3200, whereas the gains are somewhat lower at 24% for lower bandwidth DDR4-1600. Likewise better latency memory baseline improves our gains to 29%, whereas high memory latency baseline reduces our gains to 22%. In summary, our proposals deliver substantial performance at different memory configurations.

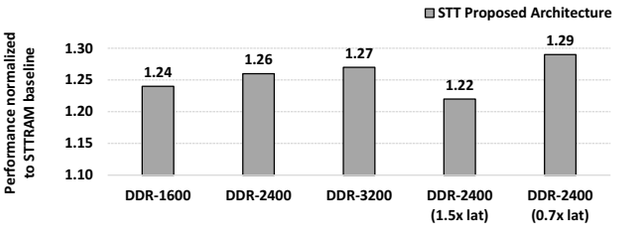

Figure 18: Impact of memory bandwidth and latency

F. Energy Considerations

STTRAM has high write energy requirements [2], [3]. Our proposed architecture reduces this write energy by eliminating writes. However, this comes at a higher miss rate which costs power in the off-chip DRAM memory. To estimate the various components of energy in the proposed architecture, we use the Micron power calculator [17] for estimating the DRAM array energy in the main memory. For STTRAM read/write energy we use the same numbers as used in [2]. Figure 19 shows the LLC and main memory energy of our proposed architecture, normalized to the baseline STTRAM. Our architecture, by eliminating writes to the STTRAM, reduces the energy across all workloads by an average of 8%. However in some workloads (for instance *gcc.g23*, *gcc.expr*) the energy increases due to additional bypasses that lower the hit rate and result in more requests at the DRAM memory.

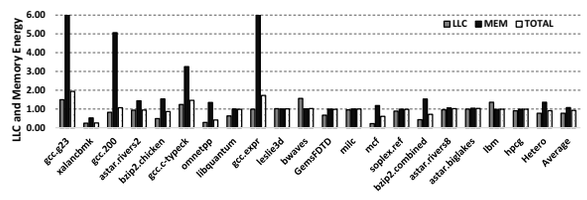

Figure 19: Combined LLC and main memory energy

G. Industry Workloads

Apart from the workloads described in Section V, we evaluated our proposal on 15 important industry applications from Enterprise, Database, BigData, Desktops, Mobile and HPC domains. Most of these applications run on real world commercial systems and some of them represent emerging usage like machine learning. Moreover, these workloads

exhibit a wide range of characteristics such as LLC MPKI and write intensity, stressing different tradeoffs in the cache hierarchy. Figure 20 shows the performance of STTRAM baseline and with our optimization on these industry workloads compared to the SRAM baseline. Overall, STTRAM with our optimization provides 9% performance advantage over the SRAM baseline showing that our proposal is applicable on real world workloads with broad range of characteristics.

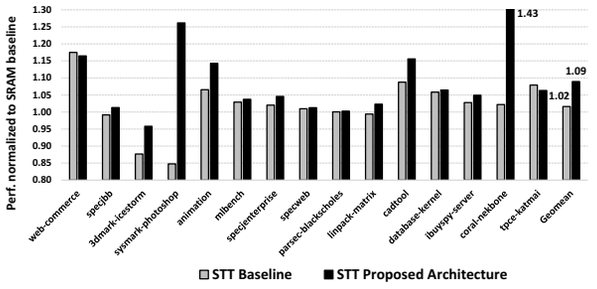

Figure 20: Evaluation on industry workloads

H. STTRAM Density vs Write Latency

As discussed earlier in Section II-A, write latency determines the density for STTRAM technology. Higher write latency results in higher density and vice versa. We consider two different density scaling characteristics based on the data available from [25] and summarized in Table II. The two capacity-latency curves respectively represent an aggressive and conservative density scaling STTRAM.

Write Latency	5ns	10ns	20ns	30ns
Conservative density	4MB	6MB	8MB	12MB
Aggressive density	7MB	12MB	16MB	20MB

Table II: **STTRAM write latency vs cell density**

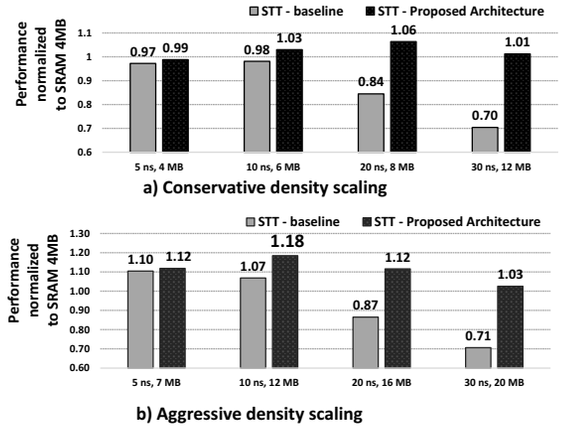

Figure 21: STTRAM density vs write latency scaling

Figure 21 shows the performance with conservative and aggressive capacity scaling as the write latency increases from 5 ns to 30 ns. In conservative scaling (Figure 21(a)), the baseline STTRAM shows performance loss and the loss increases further with the write latency. On the other hand, our proposed architecture is able to mitigate higher write latency and take advantage of larger capacity, thereby delivers 6% performance gain at (20ns, 8MB) compared to same area SRAM. Performance advantage of our architecture further increases to 18% when applied to aggressive density scaling as shown in Figure 21(b). Moreover, note that our proposed architecture sees a more gradual decline in performance for (20ns, 16MB), and (30ns, 20MB), when compared to baseline which falls off a cliff.

The results show that in presence of our optimizations, the best performing write latency is 20ns for conservative and 10ns for aggressive scaling. Without our optimizations, circuit techniques would need to reduce latency further to 10ns and 5ns for conservative and aggressive scaling configurations, thereby sacrificing significant LLC capacity. This clearly shows that a combination of circuit techniques to reduce write latency, coupled with our proposed architecture techniques, helps create a more optimal NVM LLC design.

VII. Conclusions

In this paper we showed that LLCs built with emerging NVM memory technologies like STTRAM give sub-optimal performance as compared to SRAM because of long write latency. We hence proposed a new, low cost, architecture that mitigates this write latency induced performance degradation and improves the performance of a 4 core system with an 8MB of STTRAM based exclusive LLC by an average of 26%. Moreover, we show that the proposed architecture can tolerate high asymmetry in write latency and delivers significant performance improvements over a traditional SRAM LLC. This can pave the path for the creation of future large LLCs that can effectively utilize the high density offered by these new NVM technologies, while still delivering near SRAM-like performance.

Acknowledgments

The authors thank Mainak Chaudhuri (IIT Kanpur) and our colleagues at MRL, Shankar Balachandran and Lavanya Subramanian for their valuable feedback. We also thank the anonymous reviewers for their insightful comments and suggestions.

References

[1] Zhe Wang, Daniel A. Jimenez, Cong Xu, Guangyu Sun and Yuan Xie .Adaptive placement and migration policy for an STT-RAM-based hybrid cache. HPCA 2014

[2] Hsiang-Yun Cheng, Jishen Zhao, Jack Sampson, Mary Jane Irwin, Aamer Jaleel, Yu Lu and Yuan Xie. LAP: loop-block aware inclusion properties for energy-efficient asymmetric last level caches. ISCA 2016

[3] Junwhan Ahn, Sungjoo Yoo and Kiyoung Choi. DASCA: Dead Write Prediction Assisted STT-RAM Cache Architecture. HPCA 2014

[4] Jia Zhan, Onur Kayiran, Gabriel H. Loh, Chita R. Das and Yuan Xie. OSCAR: Orchestrating STT-RAM Cache Traffic for Heterogeneous CPU-GPU Architectures. MICRO2016

[5] J. Gaur, M. Chaudhuri and S. Subramoney. Bypass and insertion algorithms for exclusive last-level caches. In *Proceedings of the 38th Annual International Symposium on Computer Architecture*, pages 81–92, June 2011.
[6] Lunkai Zhang, Brian Neely, Diana Franklin, Dmitri Strukov, Yuan Xie and Frederic T. Chong. Mellow Writes: Extending Lifetime in Resistive Memories through Selective Slow Write Backs. ISCA 2016
[7] M. Chaudhuri, J. Gaur, N. Bashyam, S. Subramoney and J. Nuzman. Introducing Hierarchy-awareness in Replacement and Bypass Algorithms for Last-level Caches. In *Proceedings of the 21st International Conference on Parallel Architecture and Compilation Techniques*, pages 293–304, September 2012.
[8] Jue Wang, Xiangyu Dong, Yuan Xie, OAP: An Obstruction-Aware Cache Management Policy for STT-RAM Last-Level Caches. DATE 2013
[9] G. Sun, X. Dong, Y. Xie, J. Li and Y. Chen, A novel architecture of the 3D stacked MRAM L2 cache for CMPs. HPCA 2009
[10] Jaewoong Sim, Jaekyu Lee, Moinuddin K. Qureshi, and Hyesoon Kim. FLEXclusion: balancing cache capacity and on-chip bandwidth via flexible exclusion. ISCA 2012
[11] Dongwoo Kang, Seungjae Baek, Jongmoo Choi, Donghee Lee, Sam H. Noh and Onur Mutlu. Amnesic Cache Management for Non-Volatile Memory. MSST 2015
[12] Samira Khan, Yingying Tian and Daniel A. Jimenez. Sampling Dead Block Prediction for Last-Level Caches. MICRO 2010
[13] Carole-Jean Wu, Aamer Jaleel, Will Hasenplaugh, Margaret Martonosi, Simon C. Steely Jr. and Joel Emer. SHiP: Signature-based Hit Predictor for High Performance Caching. MICRO 2011
[14] Mu-Tien Chang, Paul Rosenfeld, Shih-Lien Lu and Bruce Jacob. Technology Comparison for Large Last-Level Caches (L3Cs): Low-Leakage SRAM, Low Write-Energy STT-RAM, and Refresh-Optimized eDRAM. HPCA 2013
[15] Aamer Jaleel, William Hasenplaugh, Moinuddin Qureshi, Julien Sebot, Simon C. Steely Jr. and Joel Emer. Adaptive Insertion Policies for Managing Shared Caches. International Conference on Parallel Architecture and Compilation Techniques (PACT'08), pp. 208–219, 2008.
[16] Aamer Jaleel, Kevin Theobald, Simon C. Steely Jr and Joel Emer. High Performance Cache Replacement Using Re-Reference Interval Prediction (RRIP). International Symposium on Computer Architecture (ISCA-37), pp. 60–71, Saint Malo, France, June 2010.
[17] Micron Technology Inc. Calculating Memory System Power for DDR3. Micron Technical Note TN-41-01. http://www.micron.com/products/support/power-calc.
[18] Naveen Muralimanohar, Rajeev Balasubramonian and Norman P. Jouppi. CACTI 6.0: A Tool to Model Large Caches. HP Labs Technical Report HPL-2009-85, HP Laboratories, 2009.
[19] SPEC CPU 2006 Benchmarks. http://www.spec.org/cpu2006
[20] R. Ubal, B. Jang, P. Mistry, D. Schaa, and D. Kaeli. Multi2Sim: A Simulation Framework for CPU-GPU Computing. In Proceedings of the 21st International Conference on Parallel Architecture and Compilation Techniques, pages 335344, September 2012.
[21] I. Cutress. The Intel Skylake Mobile and Desktop Launch, with Architecture Analysis. September 2015. Available at http://www.anandtech.com/show/9582/intel-skylakemobile-desktop-launch-architecture-analysis/5.
[22] HPCG Benchmark. Available at http://hpcg-benchmark.org/.
[23] AMD. Phenom II processor model. http://www.amd.com/en-us/products/processors/desktop/phenom-ii
[24] ITRS. (2013) Process Integration, Devices, and Structures (PIDS). http://www.itrs.net/Links/2013ITRS/2013Tables/PIDS 2013 Tables.xlsx
[25] A V Khvalkovskiy1, D Apalkov, S Watts, R Chepulskii, R S Beach, A Ong, X Tang, A Driskill-Smith, W H Butler, P B Visscher, D Lottis, E Chen, V Nikitin and M Krounbi. Basic principles of STT-MRAM cell operation in memory arrays. Journal of Physics D: Applied Physics, Volume 46, Number 7, 2013
[26] A. Chintaluri, H. Naeimi, S. Natarajan and A. Raychowdhury. Analysis of Defects and Variations in Embedded Spin Transfer Torque (STT) MRAM Arrays. in IEEE Journal on Emerging and Selected Topics in Circuits and Systems, vol. 6, no. 3, pp. 319-329, Sept. 2016.
[27] H.-S. P. Wong, C. Ahn, J. Cao, H.-Y. Chen, S. W. Fong, Z. Jiang, C. Neumann, S. Qin, J. Sohn, Y. Wu, S. Yu, X. Zheng, H. Li, J. A. Incorvia, S. B. Eryilmaz, K. Okabe. Stanford Memory Trends. https://nano.stanford.edu/stanford-memory-trends, accessed November 8, 2016.
[28] Ping Zhou, Bo Zhao, Jun Yang, and Youtao Zhang. 2009. Energy reduction for STT-RAM using early write termination. In Proceedings of the 2009 International Conference on Computer-Aided Design(ICCAD '09). ACM, New York, NY, USA, 264-268.
[29] C. W. Smullen, V. Mohan, A. Nigam, S. Gurumurthi and M. R. Stan. Relaxing non-volatility for fast and energy-efficient STT-RAM caches. 2011 IEEE 17th International Symposium on High Performance Computer Architecture, San Antonio, TX, 2011, pp. 50-61.
[30] K. C. Chun, H. Zhao, J. D. Harms, T. H. Kim, J. P. Wang and C. H. Kim. A Scaling Roadmap and Performance Evaluation of In-Plane and Perpendicular MTJ Based STT-MRAMs for High-Density Cache Memory. in IEEE Journal of Solid-State Circuits, vol. 48, no. 2, pp. 598-610, Feb. 2013.
[31] Mustafa Shihab, Jie Zhang, Shuwen Gao, Joseph Callenes-Sloan and Myoungsoo Jung. Couture: Tailoring STT-MRAM for Persistent Main Memory. 4th Workshop on Interactions of NVM/Flash with Operating Systems and Workloads, 2016
[32] B. Jacob, S. W. Ng, and D. T. Wang. Memory Systems - Cache, DRAM, Disk. Elsevier, 2008
[33] Emre Kultursay, Mahmut Kandemir, Anand Sivasubramaniam and Onur Mutlu. Evaluating STT-RAM as an Energy-Efficient Main Memory Alternative. ISPASS 2013.
[34] Ren-Shuo Liu, De-Yu Shen, Chia-Lin Yang, Shun-Chih Yu and Cheng-Yuan Michael Wang. NVM duet: unified working memory and persistent store architecture. ASPLOS 2014.
[35] Jishen Zhao, Onur Mutlu and Yuan Xie. FIRM: Fair and High-Performance Memory Control for Persistent Memory Systems. MICRO 2014
[36] Shaodi Wang, Hochul Lee, Farbod Ebrahimi, P. Khalili Amiri, Kang L. Wang, and Puneet Gupta. Comparative Evaluation of Spin-Transfer-Torque and Magnetoelectric Random Access Memory. Journal on Emerging and Selected Topics in Circuits and Systems, June 2016
[37] Sasikanth Manipatruni, Dmitri Nikonov and Ian Young. Energy-delay performance of giant spin Hall effect switching for dense magnetic memory. Applied Physics Express, Vol 7, Num 10, 2014
[38] Jongyeon Kim, Bill Tuohy, Cong Ma, Won Ho Choi, Ibrahim Ahmed, David Lilja and Chris H. Kim. Spin-Hall effect MRAM based cache memory: A feasibility study. 2015 73rd Annual Device Research Conference (DRC), Columbus, OH, 2015, pp. 117-118
[39] E. Kitagawa, S. Fujita, K. Nomura, H. Noguchi, K. Abe et al. Impact of ultra low power and fast write operation of advanced perpendicular MTJ on power reduction for high-performance mobile CPU. IEDM 2012
[40] Shinobu Fujita, Hiroki Noguchi, Kazutaka Ikegami, Susumu Takeda, Kumiko Nomura and Keiko Abe. Novel memory hierarchy with e-STT-MRAM for near-future applications. VLSI-DAT 2017
[41] Hiroki Noguchi, Kazutaka Ikegami, Keiichi Kushida, Keiko Abe et al. A 3.3ns-Access-Time 71.2uW/MHz 1Mb Embedded STT-MRAM Using Physically Eliminated Read-Disturb Scheme and Normally-Off Memory Architecture. ISSCC 2015
[42] Hiroki Noguchi, Kazutaka Ikegami, Satoshi Takaya, Eishi Arima et al. 4Mb STT-MRAM-Based Cache with Memory Access-Aware Power Optimization and Write-Verify-Write / Read-Modify-Write Scheme. ISSCC 2016
[43] A. Chintaluri, H. Naeimi, S. Natarajan and A. Raychowdhury. Analysis of Defects and Variations in Embedded Spin Transfer Torque (STT) MRAM Arrays. in IEEE Journal on Emerging and Selected Topics in Circuits and Systems, vol. 6, no. 3, pp. 319-329, Sept. 2016.
[44] Akifumi Kawahara, Ken Kawai, Yuuichirou Ikeda, Yoshikazu Kato et al. Filament Scaling Forming Technique and Level-Verify-Write Scheme with Endurance Over 107 Cycles in ReRAM. ISSCC 2013