Cloak: Tolerating Non-Volatile Cache Read Latency

Apostolos Kokolis
University of Illinois
Urbana-Champaign
Illinois, USA
kokolis2@illinois.edu

Namrata Mantri*
NVIDIA
California, USA
nmantri@nvidia.com

Shrikanth Ganapathy†
Rivos Inc.
California, USA
shrikanth.ganapathy@gmail.com

Josep Torrellas
University of Illinois
Urbana-Champaign
Illinois, USA
torrella@illinois.edu

John Kalamatianos
AMD Inc.
Massachusetts, USA
john.kalamatianos@amd.com

ABSTRACT

The increased memory demands of workloads are putting high pressure on Last Level Caches (LLCs). In general, there is limited opportunity to increase the capacity of LLCs due to the area and power requirements of the underlying SRAM technology. Interestingly, emerging Non-Volatile Memory (NVM) technologies promise a feasible alternative to SRAM for LLCs due to their higher area density. However, NVMs have substantially higher read and write latencies, which offset their density benefit. Although researchers have proposed methods to tolerate NVM's higher write latency, little emphasis has been placed on the critical NVM read latency.

To address this problem, this paper proposes *Cloak*. Cloak exploits page-level data reuse in the LLC, to hide NVM read latency. Specifically, on certain L1 DTLB misses, Cloak transfers LLC-resident data belonging to the TLB-missing page from the LLC NVM array to a set of small SRAM Page Buffers that will service subsequent requests to this page. Further, to enable the high-bandwidth, low-latency transfer of lines of a page to the page buffers, Cloak uses an LLC layout that accelerates the discovery of LLC-resident cache lines from the page. We evaluate Cloak with full-system simulations of a 4-core processor across 14 workloads. We find that, on average, a machine with Cloak is faster than one with an SRAM LLC by 23.8% and one with an NVM-only LLC by 8.9%—in both cases, with negligible change in area. Further, Cloak reduces the ED^2 metric relative to these designs by 39.9% and 17.5%, respectively.

CCS CONCEPTS

• **Hardware → Non-volatile memory**.

KEYWORDS

Non-volatile memory, STT-RAM, Last level cache, Cache hierarchy

*Work performed while at the University of Illinois Urbana-Champaign.
†Work performed while at AMD. AMD, the AMD Arrow logo and combinations thereof are trademarks of Advanced Micro Devices, Inc.

ICS '22, June 28–30, 2022, Virtual Event, USA

ACM Reference Format:
Apostolos Kokolis, Namrata Mantri, Shrikanth Ganapathy, Josep Torrellas, and John Kalamatianos. 2022. Cloak: Tolerating Non-Volatile Cache Read Latency. In *2022 International Conference on Supercomputing (ICS '22), June 28–30, 2022, Virtual Event, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3524059.3532381

1 INTRODUCTION

The popularity of data intensive workloads, such as HPC applications and databases, has intensified capacity pressure on Last-Level Caches (LLCs). While much larger LLCs are desired, SRAM technology suffers from a high area overhead (exacerbated by the increasing manufacturing costs at leading-edge technologies [2, 4]), substantial leakage power, and scalability problems [14].

Researchers have examined alternative memory technologies, such as eDRAM and Non-Volatile Memory (NVM). In particular, NVM technologies such as STT-RAM [10] are promising candidates to replace SRAM in LLCs. Compared to SRAM, STT-RAM offers higher density and lower leakage power [33]. Compared to eDRAM, STT-RAM offers lower complexity (no refresh, activate, or precharge operations), comparable read access time, and improved power-efficiency due to its low leakage power [14]. Moreover, STT-RAM is not volatile.

However, NVMs have two main shortcomings over SRAM, namely, higher latency for both read and write operations, and a higher dynamic energy consumption per access. Moreover, read and write latencies in NVMs change based on the targeted lifetime endurance (wear-out) of the device. Therefore, replacing an SRAM LLC with an NVM one becomes a trade-off between latency, capacity, reliability and energy consumption. In this paper, we focus on mitigating the longer NVM read latency for highly-reliable NVM caches.

Table 1 compares the characteristics of SRAM and STT-RAM cells. We can see that STT-RAM cells are ~4x smaller in area, while their read and write latencies are 10–30x and 25–100x higher, respectively, than SRAM's. The table does not include the energy and power numbers because the literature provides wide ranges of values, dependent on implementation and manufacturing technology [8, 61, 63, 65, 69]. Specifically, STT-RAM's leakage power is 0.15–0.48x that of SRAM's, and its dynamic access energy is 0.8–2.5x higher than SRAM's for reads and 1.5–15x higher for writes.

Characteristic	SRAM [7, 31, 56, 58] [34, 49, 55]	STT-RAM [16, 32, 44, 47, 54] [23, 24, 41, 46, 48]
Area (F^2)	70-150	15-40
Read Latency (ns)	0.3	3 - 10
Write Latency (ns)	0.3	8 - 30

Table 1: Comparing SRAM and STT-RAM characteristics.

Prior work has tried to overcome NVM's problems of higher latency—primarily write latency—and dynamic power using solutions spanning the device, circuit, and architecture levels. At the device and circuit levels, the write access latency (primarily) can be reduced by sacrificing the retention time and non-volatility of the STT-RAM cells [29, 30, 57, 60]. Also, the transistor size can be adjusted for faster write operation [44] at the cost of higher power, lower density, and lower reliability [33]. Such approaches limit the full potential of NVM caches and do not solve the increased read latency problem. Indeed, degrading NVM characteristics to reduce the write latency introduces the need for periodic refresh, which increases design complexity and energy consumption, and hinders non-volatility [36, 50]. Additionally, adjusting the NVM cell size to reduce write latency limits NVM capacity and introduces higher error rates [19, 33].

At the architecture level, the most popular solutions to address NVM's higher latency and dynamic power involve hybrid caches that combine SRAM and NVM storage [15, 59, 63, 65]. However, these solutions focus mostly on write latency (not the focus of this paper) or use inclusive caches (not so popular today). In addition, they use a considerable amount of SRAM storage, plus complex logic to decide which cache lines to swap between SRAM and NVM. As a result, they limit the area savings from NVM and increase the energy consumption.

In terms of access latencies, several proposals mitigate the performance impact of long NVM write latencies [8, 17, 37, 61, 67, 68]. However, little emphasis has been placed on mitigating the NVM read latency, based on the common assumption that the SRAM and NVM read latencies are similar. However, measurements on fabricated STT-RAM caches and observations by industry vendors show a significant difference in read latency between STT-RAM [16, 23, 24, 32, 41, 44, 46–48, 65, 66, 69] and SRAM [7, 31, 34, 49, 55, 58]. Specifically, as shown in Table 1, FinFET-based 6T SRAM arrays perform read operations with a 300ps latency, while the fastest STT-RAMs can only attain 3ns latencies at best.

To take advantage of NVM for LLCs, we need a low-cost architectural solution that can tolerate the higher read latency of STT-RAM without sacrificing capacity, reliability, or non-volatility. This paper proposes such an architectural solution, which we call *Cloak*. Cloak exploits page-level data reuse in the LLC to hide NVM read latency. Specifically, on certain L1 DTLB misses, the hardware transfers LLC-resident lines of the TLB-missing page from the LLC NVM array to a set of small SRAM page buffers. Such buffers will service future requests to this page. To enable the low latency detection and high-bandwidth transfer of lines of a page from the LLC NVM array to the SRAM page buffers, Cloak uses an LLC layout that accelerates the discovery of LLC-resident cache lines from the page. Further, we develop an adaptive replacement policy for the page buffers to increase their utilization and achieve better performance and lower energy consumption.

We evaluate Cloak with full-system simulations of a four-core processor running 14 workloads. On average, a machine with Cloak is faster than one with an SRAM LLC by 23.8% and one with an NVM-only LLC by 8.9%—in both cases, with negligible change in area. Further, Cloak reduces the ED^2 metric relative to these designs by 39.9% and 17.5%, respectively.

2 BACKGROUND

2.1 STT-RAM Limitations and Opportunities

STT-RAM has emerged as a promising candidate to replace SRAM in LLCs [37, 45, 66] because it provides higher density and lower leakage than SRAM. However, as indicated before, the viability of STT-RAM is inhibited by higher read and write access latencies, and by higher dynamic energy than SRAM.

Past research has exploited a trade-off that exists between retention time and write access latency, to design STT-RAM cells whose write access latency is tolerable for practical on-chip integration [29, 30, 57, 60]. STT-RAM write latency is constrained by bit-level error guarantees to ensure reliable operation across the lifetime of the chip. In our case, with 16MB of STT-RAM per LLC slice (Table 2), the Bit Error Rate (BER) of the STT-RAM cell needs to be lower than 10^{-10} to ensure a 99.99999% yield with SECDED ECC. This is based on the assumption that a cache line is fetched from a single STT-RAM array block of 2MB.

Based on results from prototype devices [9, 24, 44], STT-RAMs with such error rate guarantees can achieve a bitcell write latency of ≈8ns and a read latency of ≈3.2ns. Furthermore, recent innovations in the quality of magnesium oxide (MgO), which acts as the dielectric material, pave the way for high endurance STT-RAM cells in future designs [62]. As a result, the literature reports that STT-RAM is a competitive alternative to SRAM for LLC caches. It can have an endurance in the order of 10^{12} to 10^{13} write cycles [15, 37, 62, 67], especially under normal temperature environments as the one we target [18].

An important consideration is that the access latency to an STT-RAM array *cannot* be pipelined. During a cell access, the STT-RAM array is blocked from servicing other requests. In contrast, SRAM array accesses are pipelined, and data can move to/from the cache array every cycle, achieving higher throughput than STT-RAM.

2.2 NVM Cache as an SRAM Replacement

Prior work on NVM caches [8, 17, 37, 61, 67, 68] has focused on mitigating the effect of long-latency write operations rather than read operations. The work can be categorized into three groups: methods to reduce, stall or bypass writes to NVM caches, NVM cell optimizations, and hybrid SRAM/NVM caches.

Solutions in the first group identify write contention in the NVM cache that can stall latency-critical reads, and try to take writes off the critical path of subsequent reads [8, 37, 61, 68]. These techniques assume the same read access latency for NVM and SRAMs.

Proposals that optimize NVM cells improve NVM cache write performance at the expense of retention time and area [29, 44, 57, 60]. These optimizations are not trivial, given the trade-offs between access latency, area, and retention time of NVMs [10, 19,

66]. In addition, decreasing the retention time of NVMs introduces refreshes, similar to DRAM, which increase energy consumption and complicate the design. Moreover, it limits the capacity of NVMs and introduces higher error rates. Importantly, it does not address the problem of the non-pipelined and higher read latency.

Hybrid cache proposals [15, 59, 63, 65] split a cache into an NVM and an SRAM portion, typically by partitioning a cache set into SRAM and NVM ways. These proposals monitor address reuse [59, 63, 65] and migrate frequently-used data to the SRAM portion of the cache, and rarely-used data to the NVM portion. However, these techniques have several shortcomings. First, they only target inclusive caches, which are not widely used nowadays. Second, they dedicate a large portion of cache capacity to SRAM, therefore reducing the density benefit of NVM and increasing leakage power. The area overhead of the SRAM portion is 25–100%, assuming a 4:1 density between NVM and SRAM [59, 63, 65]. Third, they need large structures of several KBs to accurately monitor cache line activity. Fourth, they must migrate data between SRAM and NVM, which further increases the number of writes to NVM, the energy consumption, and the area overhead. Fifth, in exclusive and victim LLCs, it is hard to monitor the reuse of individual addresses because LLC hits result in the removal of lines from the LLC; as a result, the overhead and complexity of recording data reuse increases. Finally, these techniques do not consider that the non-pipelined nature of NVM accesses introduces higher overhead to line migration.

Given that past work has focused on mitigating the effect of high write latency, a pressing problem now is to mitigate the impact of the high read latency of large NVM LLCs. This is the goal of this paper.

3 MOTIVATION

To increase the cache capacity in multi-cores, one can architect the LLC as a victim cache. In this environment, we propose using NVM in the LLC, to enable a higher LLC capacity for the same area. To improve on this design, in this paper, we observe that an L1 DTLB miss on a page *that has already been referenced in the past* is a good hint that some LLC-resident cache lines of the page will be re-referenced soon. Consequently, we identify such DTLB refills and bring likely-to-reuse lines from the LLC into a small SRAM structure next to the LLC. After that, the processor can read lines from this small SRAM structure with low latency.

To evaluate the potential of this idea, we use simulations. We model a conventional 3-level cache hierarchy with a 16MB LLC and 4KB pages, and run a set of workloads that will be discussed in Section 6.2. Figure 1 shows the percentage of LLC hits that originate from accesses to pages that were re-filled into the L1 DTLB. The figure shows that, on average, 94.9% of LLC hits originate from these accesses. This data implies that, upon an L1 DTLB page re-fill, there should be a significant number of LLC-resident lines from this page that will be re-referenced.

We also measure the number of LLC-resident cache lines belonging to a 4KB page at the point when the page is re-filled into the L1 DTLB. We use 4MB and 16MB LLCs. Figure 2 shows the frequency of the number of such lines. Even though it is common to have 0–3 resident lines, there is a long tail of up to 60-64 resident lines, which increases with larger LLC size (16MB). Therefore, we conclude that

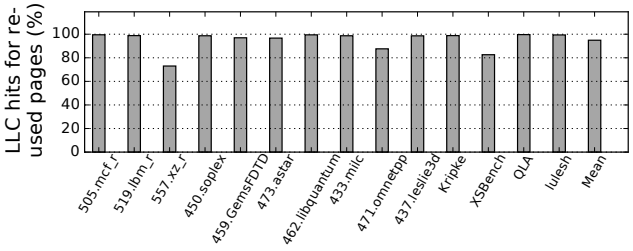


Figure 1: Percentage of LLC hits that originate from accesses to pages that were re-filled into the L1 DTLB.

the number of requests hitting in the LLC and originating from an L1 DTLB-refilled page is likely to be sizable, and increase with LLC size. Cloak builds on these observations to architect a solution that hides the increased read latency of NVM caches and increases overall request throughput.

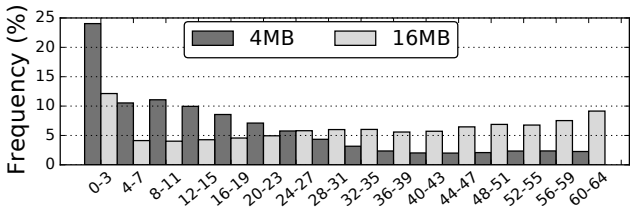


Figure 2: Frequency of the number of LLC-resident cache lines from a 4KB page that is re-filled into the L1 DTLB. The figure shows data for 4MB and 16MB LLCs.

4 DESIGN OVERVIEW OF CLOAK

4.1 Main Idea

Cloak is a hardware mechanism that takes advantage of certain L1 DTLB misses to exploit data re-use in large NVM LLC caches, and hide NVM's higher non-pipelined read latency. The NVM LLC is augmented with small SRAM buffers, which we call *Page Buffers (PB)*. PBs hold data transferred from the NVM LLC. Each PB can hold a copy of LLC-resident cache lines originating from a given page. To trigger the copy of lines into a PB, Cloak leverages the L1 DTLBs. When a miss in the L1 DTLB occurs for a previously-accessed page, the hardware passes this information to the LLC, which finds and copies the LLC-resident lines of this page to a PB.

Previously-accessed lines from the page have a high chance of being accessed again when the page translation is re-filled in the L1 DTLB—due to temporal locality. To facilitate the retrieval of a page's cache lines from the LLC, we introduce a new LLC data layout that places the lines of a given page in the same physical cache row.

Figure 3 shows the architecture of Cloak, where the new or modified hardware structures are colored, and the added connections between the L1 DTLBs and the L3 Controllers are shown in color. In this section, we describe the overall operation of Cloak to fetch data to the PBs and service memory requests. Subsequently, Section 5 discusses the architectural details of the Cloak design.

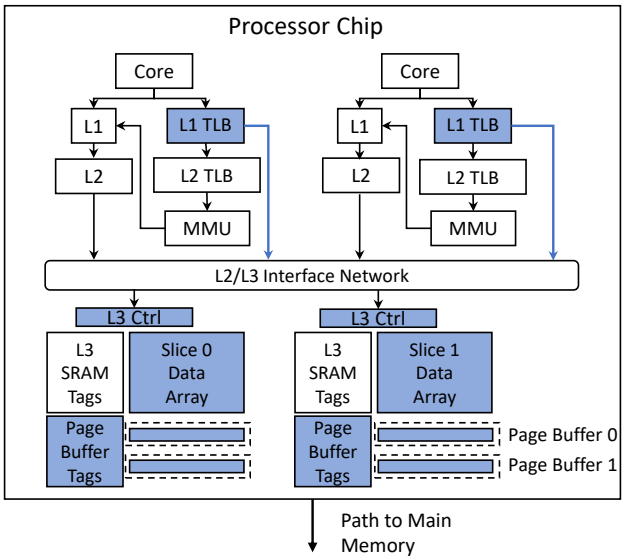


Figure 3: Cloak architecture, with the new or modified hardware structures and connections colored.

4.2 Cloak Overview

The core operations of Cloak consist of data movement from the NVM LLC data array to the PBs on a DTLB signal, and potentially servicing subsequent processor requests to the LLC from the PBs. Figure 4 shows the control flow diagrams of these actions.

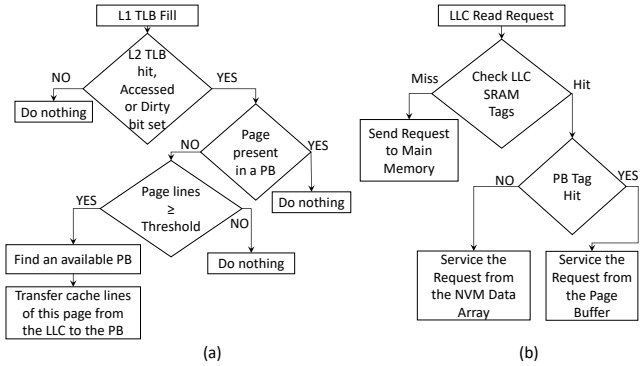


Figure 4: Control flow diagrams: (a) promotion of lines of a page to a PB, and (b) servicing a read request from the LLC.

4.2.1 TLB-triggered Page Buffer Transfer. PBs are small SRAM-based cache structures which act as fast access buffers to the NVM LLC. Each PB can hold a set of NVM-resident cache lines of a given page. Promoting NVM-resident lines of a page into a PB can reduce the read latency of future LLC accesses: thanks to intra-page spatial locality [21], it is likely that a number of future accesses will be intercepted by the PB and not need to access the NVM data array.

The algorithm to promote a page's cache lines to one of the PBs is depicted in Figure 4a. When an L1 DTLB miss occurs, the PTE for the page is fetched and Cloak determines whether this page was previously referenced. To determine whether the page was referenced in the past, Cloak checks if the page is in the L2 DTLB or, if it is not, if either the *Accessed* or *Dirty* bits of its PTE [5] is set. A set Accessed bit indicates that the page was accessed in the past. This bit is set by hardware when the page is first read or written, and is only reset by the OS to track the frequency of accesses to the page. A set Dirty bit indicates that the page was written, and hence, was referenced before. The Dirty bit is set by the processor the first time that the page is written to, and is only cleared by software.

If Cloak concludes that the page was used in the past, it sends a signal to the controller of the LLC slice that contains the physical address of the request that caused the DTLB refill. The LLC controller checks if the local PBs contain lines from this page. If they do not, Cloak decides whether to copy the lines of the page to a PB and, if so, which PB to use. To decide whether to copy the page's lines, Cloak checks the tags of the LLC slice to calculate the population of NVM-resident lines from this page. A copy to a PB occurs only if the population size exceeds a programmable threshold, so that the cost of fetching the lines to a PB can be amortized across the expected number of future PB hits. In this case, Cloak selects an available PB to copy the page's cache lines according to the PB replacement policy (Section 5.4).

4.2.2 Servicing Requests to the LLC. In Cloak, a hit in the LLC can obtain the data from the NVM data array or from a PB. Figure 4b shows the algorithm to service a read request to the LLC. The LLC controller checks in parallel the LLC SRAM tags and the PB Tags, to determine if there is a hit or a miss. If the LLC tag check misses, the request is forwarded to main memory. If the LLC tag check hits and the PB tag check misses, the request is serviced from the NVM data array. Finally, if both the LLC tag check and the PB tag check hit, the request is serviced from the PB.

The PB contents are kept coherent with the NVM data array. Specifically, writes to the LLC (e.g., due to an L2 eviction) also check the PBs and, on a hit, update both the NVM data array and the PB. Completion of write requests is signaled to L2 when the request is buffered in the LLC queues. It does not wait until when the write updates the NVM data array and PB.

A hit in the LLC NVM data array can be serviced in parallel with a PB hit to a different address. As a result, in-flight read accesses to the slow, non-pipelined NVM data array do not block younger reads to the PBs.

5 CLOAK IMPLEMENTATION

5.1 Data Layout

Copying the lines of a page from the LLC to a PB requires finding all NVM-resident lines of that page. To avoid massive LLC tag lookups and NVM cache read operations, we introduce a new data layout for the LLC. The proposed layout forces all the lines of a given page to be mapped to a single physical row of the LLC. A physical row contains multiple cache sets, each with multiple cache lines. Each physical row may contain lines from multiple pages.

For the Cloak LLC, we assume a physically distributed, logically shared LLC cache that acts as a victim of private L2 caches. While we evaluate this specific design point, the design of Cloak itself does not preclude other potential organizations of the cache hierarchy.

The LLC is split into equally-sized slices. Each slice has its own controller and can independently service any type of request. We use STT-RAM for the data array and SRAM for the tag array for two reasons. First, the tag array is much smaller than the data array and so the area overhead of using SRAM is small. Second, LLCs are typically highly set-associative, and tags are often accessed before data to minimize the dynamic energy of the data array access. Having NVM tags would add significant latency to LLC accesses.

The LLC tag array supports both conventional accesses (e.g., triggered by L2 misses) and page-level data copies to the PBs triggered by L1 DTLB fills. We distinguish the two by referring to the former as Cache Line Requests (CLR) and to the latter as Page Transfer Requests (PTR).

We map all the lines of a given page to the same LLC physical row. To do so, we alter the LLC cache indexing as presented in Figure 5. To pick the physical row, we use some bits of the Physical Page Number (PPN) called Row Index. Once the row is selected, we use a subset of the Physical Page Offset (PPO) bits called Set Index to select a set within the physical row. Then, some of the PPN bits (*Tag-High*) and of the PPO bits (*Tag-Low*) are used as tag. Finally, the remaining PPO bits are used as block offset (Figure 5).

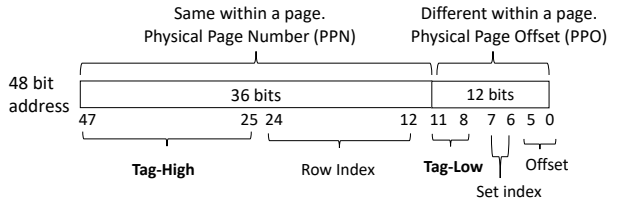


Figure 5: Cloak LLC addressing scheme.

PTRs and CLRs differ in the tag match logic. Specifically, for tag matching, a PTR access ignores the page offset bits and uses *Tag-High* bits only. In contrast, a CLR access uses both *Tag-High* and *Tag-Low* bits for tag matching.

The lines of a page could be split across LLC slices, mapping to the same physical row in each slice. However, in order to simplify the tag hit logic and NVM to PB data movement, we choose to map the entire page in the same LLC slice. Note that our layout does not impose any restrictions on the LLC organization (e.g., line size, associativity, etc.).

Example. To illustrate the proposed layout, we show an example with a single-slice of 32MB size, 16-way set-associative LLC with 64B cache lines and 4KB pages. The LLC has 8192 physical rows, each organized in 4 sets, 16 ways each. Each physical row is 4KB and can be banked if needed. As shown in Figure 5, the physical address (PA) has 48 bits, the 12 least significant ones are the PPO, and bits 0-5 form the line offset.

The cache index bits include the row index bits (bits 24:12), which select the row of the cache, and the set index bits (bits 7:6), which select the cache set within a row. Note that the row index bits (bits 24:12) do not include any PPO bits. Bits 11:8 and 47:25 form the tag, split into Tag-Low and Tag-High parts, respectively. For tag matching, a CLR selects a row and a set using indexing bits 24:12 and 7:6, respectively, and finds a match using the tag bits (bits 47:25 and 11:8). For tag matching, a PTR selects a row using the row index bits (bits 24:12) and finds all matches using the subset of tag bits lying outside the PPO, namely the Tag-High (bits 47:25). Thanks to this layout, the PTR does not search the entire cache; it only checks the Tag-High (bits 47:25) of the 64 lines in the selected cache row.

The dynamic energy of a CLR tag access is proportional to the 16 lines x 27 tag bits comparison (432 bits). The dynamic energy of a PTR tag access is proportional to the 64 lines x 23 tag bits comparison (1,472 bits). A PTR tag access consumes 3.4x more energy than a conventional CLR tag access. Triggering PTR tag searches only on DTLB re-filled pages whose lines are not already in a PB keeps the total energy cost of these operations low.

Our data layout could be extended to optimize for huge pages. However, we find that such a design is not efficient, as huge pages increase the overhead of tag lookup and data movement, while it is unlikely that a large fraction of their lines will be LLC-resident. In Section 5.5, we discuss how to efficiently handle huge pages.

5.2 Promotion of Cache Lines to Page Buffers

Cloak populates each PB with LLC-resident cache lines from one page. The process is as follows. Once a PTR reaches the LLC controller, the hardware checks if any PB already has lines from the accessed page. If not, the hardware checks the LLC tags to find if the LLC holds more than a threshold number of cache lines of the accessed page. If so, the LLC-resident lines of the page are transferred to a PB. Since the NVM cache is inclusive of the PBs, the transferred lines are not invalidated from the NVM data array.

Cloak provides hardware to bring cache lines to the PBs. According to Figure 2, the LLC may only contain a fraction of the lines from a given page. Hence, PBs will be sparsely populated. In order to increase PB utilization, we use PBs that are smaller than a page. However, given that the data in a physical row is equal to a page (4KB), steering the data from a row to a PB is not trivial and may require additional metadata and complex routing logic. To simplify both the metadata and the routing overhead, we propose using PBs of size equal to half a page (2KB). Moreover, we logically partition a physical row into two 2KB regions, and multiplex the two regions into the same PB.

Example. Figure 6 shows an example that promotes cache lines from page *A* into a PB. The two logical partitions of a physical row are formed as follows. The first partition contains the first set of the LLC row with all its 16 ways, in order, and the second set with all its 16 ways. The second partition contains the third and fourth sets. The top left part of Figure 6 shows the state of a physical row with its 64 cache lines ordered and partitioned as described. The entries with A_i values are lines with data from page *A*. Promoting the lines of the page to the PB proceeds in two steps: first from the leftmost region of the physical row (Step 1 in Figure 6), and then from the rightmost region of the row (Step 2 in Figure 6).

In Step 1 (left side of Figure 6), we process the leftmost region, which has lines in its first position (*A1*) and in the one before last (*A2*). The hardware promotes these lines into the PB. To simplify the routing, as shown in the figure, the lines are placed in the PB in the same slots that they use in the 2KB region. The PB does not need to store any address tags because any memory access will check the SRAM LLC tags first, to decide whether the corresponding PB line

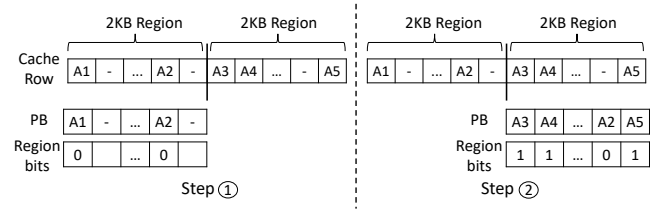


Figure 6: Promotion of the lines of a page to a PB.

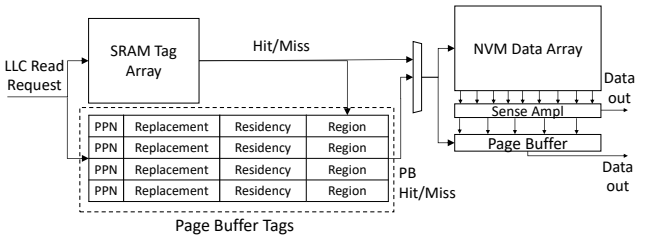


Figure 7: LLC read request path in Cloak.

is valid. However, each PB slot has a bit to identify which region the line comes from. In our example, we need 32 such bits, which we call *Region* bits. This bit is needed to fully identify the line. In the example, since the two lines come from the leftmost region, the bits are 0.

In Step 2 (right side of Figure 6), we process the rightmost region, which has lines in its first (*A3*), second (*A4*), and last (*A5*) positions. The hardware promotes these lines into the PB and sets the Region bits of the entries to 1. The figure shows the final state of the PB.

Note that the two lines in the first position of the two regions wanted to use the same PB slot, and we had to pick a winner. In the example, we picked *A3* over *A1*. To pick a winner, Cloak uses a simple algorithm that guesses which of the two lines is more likely to be used in the future. Specifically, Cloak records if the address a_{ref} that triggered the DTLB miss belongs to the first or second half of the page. Then, when populating a PB, when two lines want to use the same PB entry, the line from the same half of the page as a_{ref} is the one that wins. This algorithm guesses that, because of spatial locality, this line is more likely to be accessed soon that the other line.

Given Cloak's proposed LLC organization, the operation of promoting the lines from the two regions (and, in another design, from potentially more regions) into a PB does not stall the LLC pipeline more than a single read access would. Indeed, all the cache lines of a page are on the *same physical row*, and thus they are promoted to a PB from the NVM data array with *a single read operation*. Finally, the writes into the PB are pipelined: as the first region is written, the second region performs the checks.

5.3 Tag Checks

To keep track of the pages and cache lines that are present in the PBs, Cloak employs an array called the *Page Buffer Tags* (PB Tags) (Figure 7). Each entry in the PB Tags corresponds to one PB. An entry has: (a) the PPN of the page whose lines are stored in the PB, (b) a *Replacement* counter to manage PB replacement, (c) a *Residency* counter that tracks the number of valid lines in the PB, and (d) the *Region* bits discussed above.

Both PTR and CLR use the PB Tags to determine whether a PB contains data for the page requested. In the case of a PTR, the PB Tags are searched using the PPN of the requested page. In a CLR, the PB Tags are searched using the PPN of the requested page and the correct bit in the Region field corresponding to the requested address.

A CLR operation starts by accessing both the LLC SRAM tag array and the PB Tags simultaneously (Figure 7). It uses the PPN bits of the PB Tags to identify if a PB contains lines from the page accessed. It uses the LLC tag array to identify if and where the requested line resides in the LLC. If the address of the line is not found in the LLC tag array, an LLC miss is declared.

However, if the LLC tag array indicates a cache hit, Cloak checks for a PB hit. A PB hit will occur if: (i) there is a PB with lines of the page accessed and (ii) the region of the LLC physical row with the matching address is the same as the one indicated by the Region bit of the corresponding location of the PB. In this case, the line is accessed from the PB in the same position. Recall that, during data promotion, lines were moved from the LLC to the PB without reordering. If the Region bit does not match or the PPN bits do not match, the line is accessed from the NVM-LLC data array.

Note that the access to the PPNs in the PB tags overlaps with the access to the LLC tag array. The access to the Region bit in the PB tags is only performed after the LLC tag array access (Figure 7). However, accessing the Region bit only extends the critical path by one cycle—when both the PPN and the LLC tag array hit.

The PB contents are always kept synchronized with the contents of the LLC NVM data array. When a line is written to the LLC, the corresponding line in the PB, if present, is updated. For this reason, there is no need to write back PBs to the NVM data array. There is also no need to keep valid or coherence state bits in PB Tags because the LLC tags provide such information. Whenever an LLC line is invalidated (due to an external probe or L2 promotion), or evicted (due to an LLC replacement), the corresponding valid bit of the line in the LLC tags is reset. No other action is needed: given that the PB hit logic waits for the LLC tag search to complete, a CLR will not read the PB slot data if its corresponding LLC line is invalid, even if the data is still resident in the PB entry.

The Residency counter in the PB Tags tells how many cache lines are valid in a PB. This counter is set when the lines of a page are moved from the NVM data array to the PB. It is decremented when one of the lines is invalidated or evicted from the LLC. It is incremented when an L2 victim is installed in the LLC and copied into the PB. The Residency and Replacement counters are used to handle PB replacement, as we discuss in Section 5.4.

Example. The PBs add little area overhead to the LLC. To see why, consider an example based on Figure 5. A PB is composed of tag and data. For the tag (Figure 7), we have a 36-bit PPN and assume a 10-bit Replacement counter. The Residency counter needs $log_2(PBsize/linesize)$ bits, which is 5 in our example. The Region bits are $(PBsize/linesize) * (log_2(Pagesize/PBsize))$, which is 32 in our case. The total comes to 83 bits per PB tag entry. The size of the PB data per entry is 2KB. Based on this, if we assume a 4:1 area

ratio between NVM and SRAM, we estimate that each PB adds a 0.046% area overhead over the 16MB LLC slice.

5.4 Page Buffer Replacement Policy

Cloak needs to find an available PB to promote a page's cache lines, when all PBs are in use. To pick a PB, Cloak uses a PB replacement algorithm that considers: (i) how many cache lines are resident in a PB, and (ii) the frequency of accesses to the page in the PB. The goal is to capture the dynamic behavior of accesses to each PB, and neither replace a PB too early (before its entries are accessed), nor keep a page resident in a PB if it is not being accessed.

Specifically, when a PB is loaded, its Replacement counter is set to the product of the Residency counter and a programmable constant called *Activation Period*. At every cycle, the Replacement counter is decreased by one. When a PB entry is accessed, either for a read or a write operation, its Replacement counter is recalculated by multiplying the current value of the Residency counter with the Activation Period.

Note that PB accesses also change the Residency counter. On a PB read, Cloak decrements the Residency counter because a line from the PB is moved to the private caches. On a PB write because of a line eviction from the private caches, Cloak increments the Residency counter. Once the Replacement counter reaches zero, the PB entry is subject to replacement.

5.5 Huge Page Management

Modern systems support huge pages, such as 2MB and 1GB, to alleviate TLB pressure. Even though we described Cloak in the context of 4KB pages, Cloak can support huge pages without any modification to the NVM cache layout.

We envision a physical row in the LLC cache to still hold 4KB of data. However, if we chose to transfer a whole 2MB page into a PB, we would need to search many rows. Moreover, larger PBs would likely be underutilized.

Consequently, Cloak only transfers lines from individual 4KB chunks of a huge page at a time into a PB. Specifically, when a huge page entry is re-filled into the L1 DTLB, Cloak only promotes lines from the 4KB chunk of this huge page that contains the address that triggered the DTLB miss. In addition, the L1 DTLB records this 4KB chunk that triggered the DTLB miss. Subsequently, when an access occurs to the same huge page, but a different 4KB chunk, Cloak triggers the transfer of lines from the new 4KB chunk, and again records the chunk in the DTLB. In this way, Cloak can have multiple 4KB chunks of the same huge page active in the PBs.

Cloak adds this support for 2MB and 1GB pages. For the 2MB pages, Cloak needs to add 9 bits per L1 DTLB entry to record the most-recently-promoted 4KB chunk. For the 1GB pages, Cloak needs to add 18 bits per L1 DTLB entry. These are minimal area overheads.

6 EVALUATION METHODOLOGY

6.1 Modeled Architecture and Infrastructure

We use full-system cycle-level simulations using the SST [52] framework to model a server architecture with 4 cores and 64 GB of main memory. The main architecture parameters are shown in Table 2. Each core is out-of-order with private L1 and L2 caches, and a shared LLC. The L1 and L2 caches use stride and next-line prefetchers, respectively, as implemented by SST. The L2 cache is inclusive of L1, while the LLC is populated by L2 victims. The baseline system uses an SRAM-based physically-distributed logically-shared LLC. For Cloak, the LLC has SRAM tags and an increased-latency, non-pipelined, STT-RAM-based data array.

We use published data (Table 1) to set the read and write latencies to access the STT-RAM LLC data array to be 3 ns and 8 ns, respectively. Note that this latency is not the *total round trip latency* to access the LLC from the core. Such latency is shown in Table 2 to be 63 cycles for reads and 78 cycles for writes.

There are private L1 and L2 DTLBs, and a page walker per core. For our evaluation, we integrate the Simics full-system simulator [42] with the SST [52] framework. Simics is a full-system functional simulator that provides SST with the necessary information about the executed instructions. SST gets the executed instructions alongside information about virtual and physical addresses, page walk addresses, and registers. It then simulates our system architecture with the parameters of Table 2.

To model main memory, we use the DRAMSim2 [53] memory simulator. We use Intel SAE [13] on top of Simics for OS instrumentation. Finally, we use CACTI [11] and McPAT [39] to calculate the timing and energy parameters of our processor and SRAM-based tags, data arrays, and buffers. For Cloak's STT-RAM data array, we scale the SRAM energy values according to prior work [61, 63]. After the hardware checks the LLC tags, we add one extra cycle for the hardware to determine whether it is a PB hit or miss. This is necessary because the LLC tag search determines the location of the line inside a PB.

6.2 Configurations and Workloads

We compare four different design configurations.

Baseline: SRAM-based LLC with the latency and size parameters described in Table 2.

NVM-Only: LLC with STT-RAM for the data array and SRAM for the tag array with the parameters of Table 2, but without PB support and with conventional indexing.

Cloak: LLC with STT-RAM for the data array and SRAM for the tag array with the proposed data layout and PB support.

O-SRAM: Optimistic LLC hybrid design with conventional indexing, pipelined access latency and energy characteristics of *Baseline*, combined with STT-RAM area density.

To evaluate the efficacy of our design, we run 14 workloads. They are shown in Table 3 with their memory footprint and L2 misses per kilo instructions (MPKI). We chose ten benchmarks from the SPEC CPU® 2017 (Group A) [6] and SPEC CPU® 2006 (Group B) [22][1] benchmark suites with high MPKI to stress the memory subsystem. We also run four benchmarks from the CORAL [1] and CORAL2 [3] suites (Group C), as representative HPC applications. The memory footprint of our workloads does not fit in the L2 and can stress the LLC. We select the region of interest (ROI) with SimPoint [20] for the SPEC® workloads with intervals of 200 million instructions for each of 10 different regions, and we instrument the source code for the other workloads. In all cases, we warm-up the architectural

[1]SPEC® and SPEC CPU® are registered trademarks of the Standard Performance Evaluation Corporation. See www.spec.org for more information.

Table 2: Architectural parameters used for the evaluation. In the table, RT means round trip latency from the core.

Processor Parameters	
Multicore chip	4 OoO cores, 4-issue, 22nm, 3.2GHz
Ld-St queue; ROB	92 entries; 192 entries
L1 cache	32KB, 8-way, 2 cycles round trip latency (RT), 64B line
L2 cache	512KB, 8-way, 14 cycles RT, 64B line
Prefetchers	L1 cache: stride prefetch. L2 cache: next-block prefetch
Baseline LLC cache	4MB/core, 16-way, 1 slice/core, 64B line 53 cycles RT, 2 cycles tag latency, 12 cycles data latency Energy: Read/Write 0.47/0.48nJ, Tag 4pJ, Leak: 1.4W
L1 DTLB	64 entries, 4-way, 2 cycles RT
L2 DTLB	1024 entries, 12-way, 12 cycles RT
NVM cache parameters	
LLC cache (SRAM tag + NVM data)	16MB/core, 16-way, 1 slice/core, 64B line 63 cycles RT read latency, 78 cycles RT write latency 2 cycles tag access latency, 22 cycles data access latency (of which 10 cycles are not pipelined) Energy: Read/Write 0.95/6.3nJ, Tag 7pJ , Leak: 829mW
Page Buffers (PB)	20 PBs, 2KB/each, 43 cycles RT Energy: Read/Write 12/13pJ, Tag 12pJ, Leak: 4.1mW
PTR signal latency STT-RAM cache to PB threshold	6 cycles 6 cache lines
PB activation period	20 cycles per active cache line
PB area overhead	0.92% area overhead over an LLC slice (0.046% per PB)
Main-Memory Parameters	
Capacity	64GB
Channels; Banks	2; 8
Latency	190 cycles RT (on average)
Freq; Bus width	1.6GHz DDR; 64 bits per channel
System Parameters	
Host OS	Ubuntu Server 16.04

Table 3: Workloads.

Workload	Footprint (MB)	L2 MPKI	Workload	Footprint (MB)	L2 MPKI
Group A			**Group B**		
505.mcf_r	613	39			
519.lbm_r	409	10	450.soplex	436	10
557.xz_r	800	3	459.GemsFDTD	146	4
Group C			473.astar	372	18
Kripke	608	39	462.libquantum	267	11
XSBench	110	63	433.milc	123	7
QLA	375	11	471.omnetpp	388	12
lulesh	110	15	437.leslie3d	62	3

state by running one quarter of the instructions before simulating the remaining three quarters of the instructions.

7 EVALUATION

7.1 Cloak Performance and Energy

In this section, we evaluate the performance of Cloak. When replacing an SRAM-based cache with NVM (STT-RAM), there are two factors that affect application performance. The first is the higher read and write latencies of STT-RAM. The second is the lower cache miss rate due to the higher area density of NVM technology. We consider two different metrics in Figure 8 to show the performance impact of Cloak. Figures 8a and 8c show the L2 miss response times for read CLRs, while Figures 8b and 8d depict the application speedup. All figures are normalized to the Baseline configuration. We conduct experiments on a system with 4KB pages only, and a system with Transparent Huge Pages enabled (2MB and 1GB pages).

Figures 8a and 8c show the L2 miss response time, which is calculated as the total number of cycles from issuing an L2 miss until the miss response reaches back to the L2. On average, Cloak reduces the L2 miss response time by 30.0% and 30.5% over Baseline, with and without Huge Pages. This impact is really close to that of the O-SRAM configuration. The NVM-Only configuration lowers the L2 miss response time by only 15.8% and 15.9%. It does not achieve the same reduction as Cloak because, without PBs, it has high and non-pipelined LLC hit latency. These results indicate that the PBs are effective at reducing the NVM cache read latency—practically as much as O-SRAM.

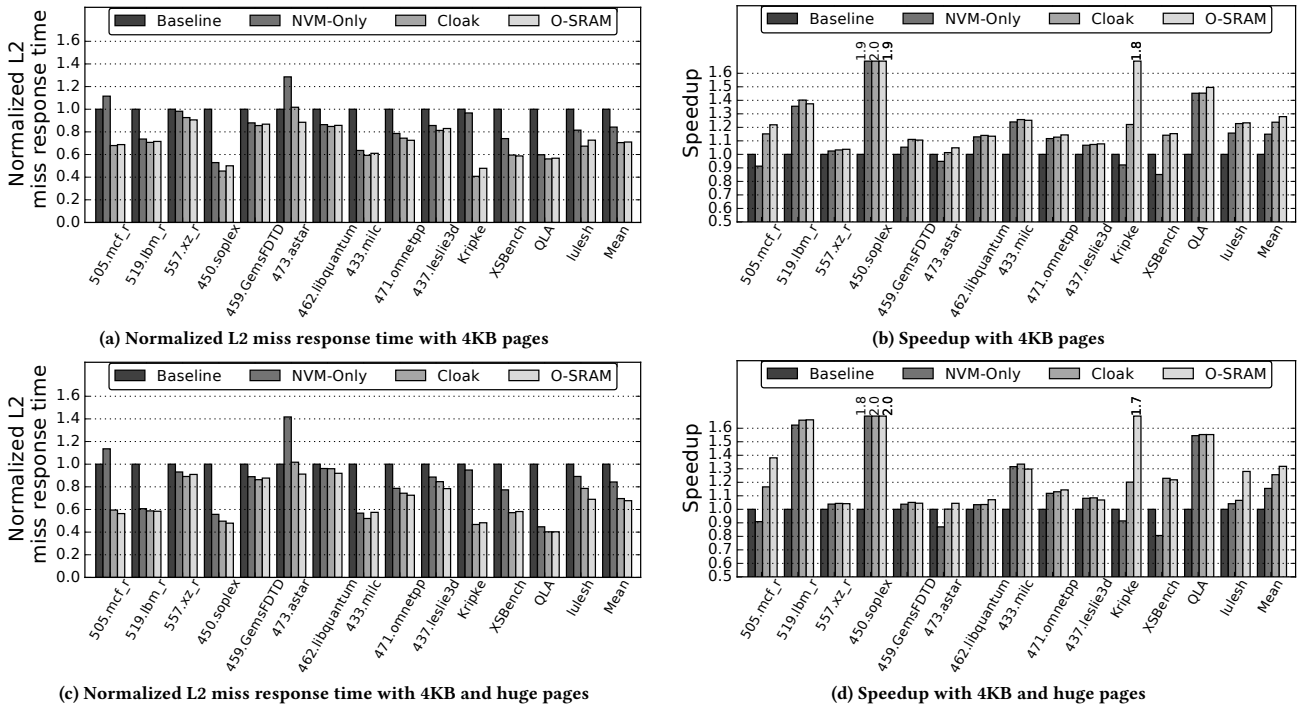
Figures 8b and 8d show the application speedup over Baseline. We see that NVM-Only LLCs can increase performance. The reason is the larger LLC capacity achieved via NVM technology, which can greatly decrease the LLC miss rate. However, there are benchmarks where NVM-Only experiences a performance degradation compared to Baseline (505.mcf_r, 473.astar, Kripke, and XSBench). This is because the lower LLC miss rate cannot compensate for the higher LLC hit latency. Benchmarks with high L2 MPKI and high LLC hit rate suffer more from the increased read latency of an NVM-based LLC. For instance, 473.astar and XSBench with Huge Pages experience 13% and 19% lower performance than Baseline, respectively.

On the other hand, Cloak consistently attains higher performance than Baseline and NVM-Only. There are times when it even outperforms O-SRAM. This can happen for benchmarks with high PB hit rate, because a PB hit has a lower access latency than a hit in an SRAM-based LLC. This is due to the lower routing latency observed when retrieving data from the PB data array compared to the much larger SRAM-based LLC slice. On average, Cloak is 25.6% and 23.8% faster than Baseline with and without Huge Pages, respectively, while NVM-Only is 15.5% and 14.9% faster than Baseline. Cloak outperforms Baseline by up to 97%, effectively hiding the increased read latency of STT-RAM.

We also tested Cloak's efficacy by running mixes of four benchmarks at a time. It can be shown that NVM-Only and Cloak outperform baseline by 24% and 31% without Huge pages and by 27% and 33% with Huge pages, respectively. Additionally, we also tested all the low L2-MPKI SPEC® benchmarks and found that Cloak consistently achieves the same or slightly higher performance (1-2%) than the Baseline and NVM-Only cases.

The rest of our evaluation focuses on a system that utilizes only 4KB pages. However, the performance trends remain the same when huge pages are enabled.

To further understand the performance impact of Cloak, we present two more performance metrics in Figures 9 and 10. In Figure 9, we show the LLC Misses per Kilo-Instructions (MPKI) for the four configurations. The increased capacity with STT-RAM greatly reduces the LLC MPKI of the applications. The MPKI drops across all benchmarks—including those with the highest MPKI such as XSBench, which shows a 50% drop. In most cases, Cloak achieves an LLC MPKI close to that of O-SRAM.



(a) Normalized L2 miss response time with 4KB pages

(b) Speedup with 4KB pages

(c) Normalized L2 miss response time with 4KB and huge pages

(d) Speedup with 4KB and huge pages

Figure 8: Performance impact of Cloak normalized to the Baseline configuration, when huge pages are disabled (a,b) and enabled (c,d). The figures show the normalized L2 miss response times (a,c) and the application speedup (b,d).

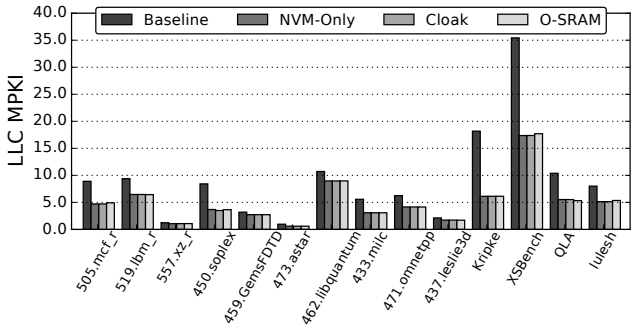


Figure 9: LLC misses per Kilo-Instructions (MPKI).

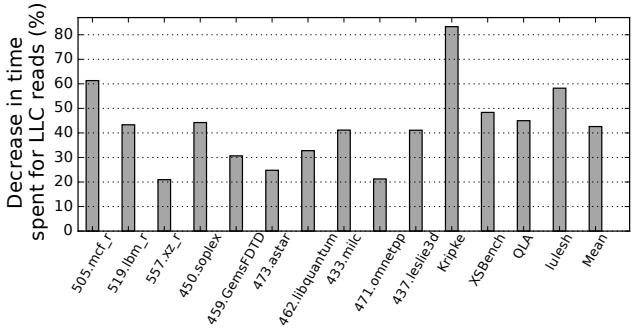


Figure 10: LLC read latency reduction of Cloak over NVM-Only.

To isolate the performance impact of PBs, Figure 10 compares the total time that read requests spend in the LLC in NVM-Only and Cloak. This time is calculated as the total number of cycles from the time that an L2 miss is issued until the response is sent back to L2 by the LLC data array or a PB (in case of a hit), or until the LLC declares a miss. Note that the two configurations have similar LLC MPKIs. Therefore, their cycle count difference depends on the PB hit rate in Cloak. Figure 10 shows that Cloak notably reduces the LLC read latency cycles and, therefore, accelerates the LLC read requests. On average, LLC CLRs spent 42.5% less time in the LLC with Cloak than with NVM-Only. The PBs are able to speed-up Cloak because they service CLRs much faster than the LLC NVM-based data array (both in latency and bandwidth). Moreover, when CLRs are serviced from the PBs, they do not block the LLC data array pipeline, giving the opportunity to subsequent CLRs that do not access the PBs, to proceed in parallel. As a result, the PBs not only service requests faster, but also increase the overall throughput of the LLC.

Figure 11 shows the ED^2 of the different configurations normalized to Baseline. The bars are broken down into the contributions of the core plus private caches, the LLC, and main memory. Overall, we see that all STT-RAM designs have a lower ED^2 than Baseline. On average, NVM-Only reduces the ED^2 by 22.4%, while Cloak reduces it by 39.9%. The reasons are the lower execution times

of the STT-RAM configurations, the lower leakage power of the STT-RAM data array (which is the main energy contributor of the LLC), and the reduced number of accesses to main memory.

Compared to the NVM-Only design, Cloak consumes more dynamic energy in the LLC because it performs more tag checks (LLC and PB) and read accesses to the STT-RAM data array when fetching data to the PBs. Cloak also consumes more static energy due to the PB leakage. Specifically, it can be shown that: (i) PB static energy accounts for 6.6% of total LLC energy consumption, (ii) cache line promotion to the PBs accounts for 8.5% of total LLC energy, and, (iii) the energy of PB tag checks and PB data accesses accounts for 0.04% of the total LLC energy.

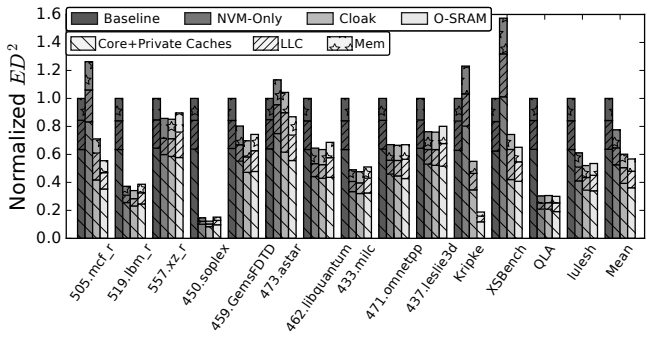


Figure 11: ED^2 normalized to Baseline.

However, this extra energy is compensated by Cloak's faster execution because it services many requests from the PBs. O-SRAM reduces the ED^2 by 43.3% over Baseline on average, delivering the best energy efficiency on average. Yet, in some cases, Cloak achieves better energy efficiency because O-SRAM's faster execution time can not compensate for its higher leakage.

7.2 Cloak Characterization

To achieve high performance, it is crucial to maximize the use of PBs. We find that 84.4% of the PTRs sent by Cloak to the LLC are for pages that have at least 6 cache lines in the LLC. Moreover, 99% of the PTRs are able to find a PB to promote the lines to.

To get further insight, Figure 12 shows the percentage of LLC hits serviced from the PBs (instead of from the STT-RAM data array). On average, 54% of the hit CLRs are serviced from the PBs instead of the STT-RAM data array. The benchmarks with the highest LLC hits per kilo-instruction (HPKI) such as XSBench (45 HPKI) and Kripke (32 HPKI) hit in the PB 57% and 48% of the time, respectively (Figure 12). This leads to substantial performance gains of Cloak over NVM-Only, as shown in Figure 8b.

We now quantify the coverage of PTRs to the LLC in Figures 13a and 13b. Figure 13a shows the percentage of cache lines promoted into PBs that are actually accessed from the PBs. In the figure, Group A, Group B, and Group C are the mean of the SPEC CPU® 2017, SPEC CPU® 2006, and Coral benchmarks in Table 3, respectively. We see that, on average, CLRs reference 51.1% of the cache lines promoted to the PBs. This relatively high hit ratio is helped by our PB replacement algorithm that favors the victimization of PBs with few lines or with a low number of hits.

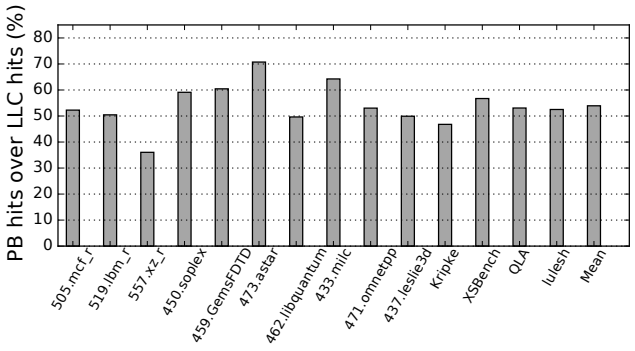


Figure 12: Percentage of LLC hits that are serviced by PBs.

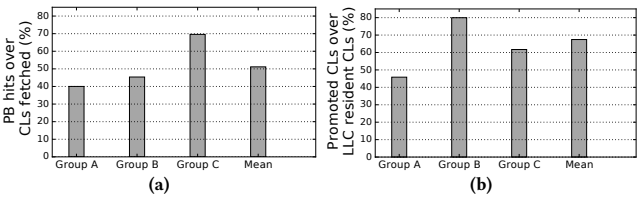


Figure 13: Characterizing PB use: (a) percentage of cache lines (CLs) promoted into PBs that are actually accessed from the PBs, and (b) percentage of LLC-resident CLs of a page that are promoted to a PB in a PTR.

Besides our adaptive replacement policy, we tried the LRU replacement policy that is typically used in caches. With LRU, we find that PB hits drop by 10.1%, PB hits per cache line fetched to PBs drops by 14.2%, and Cloak is 2% slower. Thus, LRU introduces more data movement and delivers lower performance. We also tried a greedy replacement policy that always replaces the PB that contains the smallest number of valid cache lines. We find that this policy reduces the number of PB hits by 7%, decreases the PB hits per cache line fetched to PBs by 15%, and reduces Cloak's speed by 1.2%. Note that an important aspect of the PB design, besides the replacement policy, is that when we promote cache lines into a PB, we do not pollute the L2. This is because the PBs simply hold a copy of the data present in the STT-RAM data array.

Figure 13b shows the percentage of LLC-resident cache lines of a page that are promoted to a PB in a PTR. This number is not 100% for two reasons. First, for a given page, some of the lines from different regions in the same physical row may conflict with each other, and cannot all be promoted to the PB. Second, if the number of LLC-resident lines is less than a threshold, Cloak does not promote the page. On average, Cloak promotes 68% of the LLC-resident cache lines of a page to a PB—or about 26 lines.

7.3 Alternative Cloak Design

To further highlight the benefits of Cloak, we evaluate a scheme that fetches NVM-resident cache lines to the L2 cache instead of the PBs, using the same trigger as Cloak. For this experiment, we keep the data layout of the LLC that we introduced in this paper,

so that we can identify the cache lines of a page with a single LLC read operation. Moreover, as an optimization, we make sure that the L2 cache always prioritizes read requests from the core over LLC-to-L2 prefetches. In addition, when the L2 MSHR entries are heavily utilized (i.e., 90% or higher), we drop outstanding LLC-to-L2 prefetches.

We find that this design is not competitive with Cloak: on average, it is 19.8% slower than Cloak and increases the writes to NVM by 183%. This is because bulk data moves from LLC to L2 saturate the interconnect, causing core requests to stall while arbitrating for the network. Moreover, fetching many lines to L2 causes L2 thrashing, which in turn increases L2 misses. This is especially the case for benchmarks with a high L2 MPKI such as XSBench. This benchmark takes 90% longer to complete with the new design than with Cloak because of the increased traffic between the L2 and the LLC. Only benchmarks with a small L2 MPKI and a high PB hit ratio, such as 450.soplex and 437.leslie3d, can benefit from this design, and attain a performance that is comparable to Cloak's.

An aggressive L2 prefetcher that tries to prefetch the same cache lines as Cloak faces the same performance bottleneck. Furthermore, if the Cloak LLC data layout is not used, read requests from the core suffer from low LLC read bandwidth due to the non-pipelined nature of STT-RAM data array accesses.

7.4 Sensitivity Analysis

Finally, we perform two sensitivity analyses. First, we examine the sensitivity of Cloak to the LLC size, which is the primary parameter dictating the LLC hit rate. Figures 14a and 14b show the average L2 miss response time and the average workload speedup, respectively, across all benchmarks, as the size of the LLC increases from 4MB to 32MB per core. All results are normalized to Baseline, which has an SRAM-based LLC with 4MB per core.

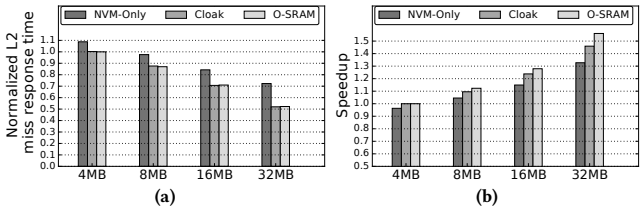


Figure 14: Sensitivity analysis of different LLC sizes per core over Baseline with an SRAM-based LLC of 4MB per core: (a) normalized L2 miss response time and (b) speedup.

Figure 14a shows that the relative L2 miss response time drops with the increase in LLC size for all the schemes. Cloak has lower L2 miss response time than NVM-Only for all configurations. It has practically the same L2 miss response time as O-SRAM because the PBs provide even faster access than a larger SRAM LLC slice due to their smaller routing overhead.

Figure 14b shows that the speedup of all the schemes increases with the LLC size. This is because of the increasingly lower LLC miss rate. For all LLC sizes beyond 4MB per core, Cloak delivers higher speedups than NVM-Only and lower speedups than O-SRAM. Interestingly, Cloak can tolerate the higher read latency of STT-RAM and achieve equal performance to Baseline with a 4MB LLC.

We also analyze the effects of increasing the read latency of STT-RAM LLC caches, while keeping the cache size at 16MB per core. Figure 15a and Figure 15b show the average L2 miss response time and the average workload speedup, respectively, across all benchmarks, as the LLC read latency is increased. We increase the latency by lengthening the NVM-based LLC data array read latency by 10, 20 and 30 cycles over the SRAM Baseline. The configuration with +10 cycles is the NVM cache configuration we have simulated in our prior experiments. The three designs are an STT-RAM with a read latency of 3ns, 6ns and 9ns. All results are normalized to Baseline, which has an SRAM-based LLC of 4MB per core.

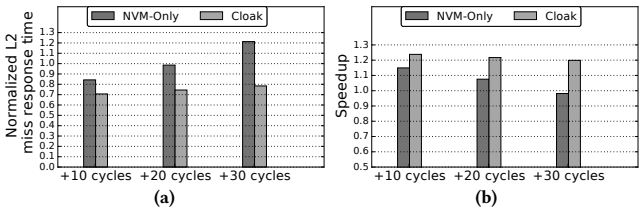


Figure 15: Sensitivity analysis of different LLC read latencies over Baseline with an SRAM-based LLC of 4MB per core: (a) normalized L2 miss response time and (b) speedup.

As the STT-RAM LLC read latency increases, the relative L2 miss response time increases, and the speedup drops. These trends occur for both NVM-Only and Cloak, although they are less prominent for Cloak. In all cases, Cloak has a lower L2 miss response time and a higher speedup than both NVM-Only and Baseline. This is because the PBs can tolerate part of the higher STT-RAM array read latency.

8 OTHER RELATED WORK

Page Caches. Prior work has looked into the use of die-stacked eDRAM as large LLCs [25–28, 38, 40, 51, 64]. eDRAM-based caches are typically organized in pages (i.e., Page Caches) instead of blocks to avoid massive tag storage. When a request reaches a page cache and the page is not cached, the whole or a subset of the page [26, 27] is brought from main memory, generating off-chip traffic. The capacity of page caches is underutilized, since a page allocates cache space even for lines that are not fetched. This reduces cache capacity. In addition, page caches add extra overhead to keep track of a page's useful footprint. Cloak does not sacrifice any LLC capacity, does not need to track any footprint, and does not generate any off-chip traffic. Instead, it brings the LLC-resident lines into the PBs. Efficient page caches cannot be easily designed as victim or non-inclusive LLCs—e.g., storing a victim line requires the allocation of space for the whole page. Instead, Cloak can be integrated with LLCs of different inclusion properties.

Techniques to Hide High Latency. Conventional techniques such as prefetching and dead block elimination [8] are orthogonal to Cloak and can be used in conjunction with it. However, LLC

prefetchers incur increased complexity and can saturate memory bandwidth (Section 7.3) when using NVM caches [43, 61, 67].

The advantage of using the address translation to make early decisions has been demonstrated before for page walks. Specifically, TEMPO [12] uses PTE page walk requests that miss in the cache hierarchy to prefetch the cache line that caused the page walk to LLC from main memory. PageSeer [35] uses page walk information to swap pages in a DRAM-NVM hybrid main memory system. Cloak is different because it hides the higher read latency of NVM-based LLCs. Second, it uses a different trigger, namely, the DTLB miss on a page used in the past.

9 CONCLUSION

This paper presented Cloak, a novel, low cost NVM LLC architecture that uses small SRAM-based page buffers to tolerate the higher and non-pipelined latency of NVM reads. An L1 DTLB miss on certain pages triggers the data transfer of LLC-resident lines belonging to the page from the NVM LLC to the page buffers. The buffers will service subsequent requests for this page, and use a novel replacement algorithm to achieve high performance and low energy consumption. Further, to enable the high-bandwidth, low-latency transfer of lines of a page to the page buffers, Cloak uses an LLC layout that accelerates the discovery of LLC-resident cache lines from the page.

Cloak effectively hides the higher latency of NVM reads. We find that, on average, a machine with Cloak is faster than one with an SRAM LLC by 23.8% and one with an NVM-only LLC by 8.9%—in both cases, with negligible change in area. Further, Cloak reduces the ED^2 metric relative to these designs by 39.9% and 17.5%, respectively.

ACKNOWLEDGMENTS

The inventions in this publication were made with government support under a PathForward Project with Lawrence Livermore National Security (Prime Contract No. DE-AC52-07NA27344, Subcontract No. B620717) awarded by DOE, and under NSF grants CNS-1763658 and CCF-2107470. The government has certain rights in the inventions.

REFERENCES

[1] 2014. CORAL Benchmark Codes. https://asc.llnl.gov/CORAL-benchmarks/.
[2] 2018. Big Trouble At 3nm. https://semiengineering.com/big-trouble-at-3nm/.
[3] 2018. CORAL2 Benchmarks. https://asc.llnl.gov/coral-2-benchmarks/.
[4] 2019. Apple A13 & Beyond: How Transistor Count And Costs Will Go Up. https://wccftech.com/apple-5nm-3nm-cost-transistors/.
[5] 2021. AMD64 Architecture Programmer's Manual Volume 2: System Programming. https://www.amd.com/system/files/TechDocs/24593.pdf.
[6] 2021. SPEC CPU2017. https://www.spec.org/cpu2017.
[7] Hassan Mostafa Ahmed T. El-Thakeb, Hamdy A. Elhamid and Yehea Ismail. 2014. Performance evaluation of FINFET based SRAM under statistical VT variability. In *Proceedings of the 2014 International Conference on Microelectronics* (Doha, Qatar) *(ICM'14)*. IEEE.
[8] J. Ahn, S. Yoo, and K. Choi. 2014. DASCA: Dead Write Prediction Assisted STT-RAM Cache Architecture. In *2014 IEEE 20th International Symposium on High Performance Computer Architecture*.
[9] J. G. Alzate, U. Arslan, P. Bai, J. Brockman, Y. J. Chen, N. Das, K. Fischer, T. Ghani, P. Heil, P. Hentges, R. Jahan, A. Littlejohn, M. Mainuddin, D. Ouellette, J. Pellegren, T. Pramanik, C. Puls, P. Quintero, T. Rahman, M. Sekhar, B. Sell, M. Seth, A. J. Smith, A. K. Smith, L. Wei, C. Wiegand, O. Golonzka, and F. Hamzaoglu. 2019. 2 MB Array-Level Demonstration of STT-MRAM Process and Performance Towards L4 Cache Applications. In *2019 IEEE International Electron Devices Meeting (IEDM)*. 2.4.1–2.4.4.
[10] Dmytro Apalkov, Alexey Khvalkovskiy, Steven Watts, Vladimir Nikitin, Xueti Tang, Daniel Lottis, Kiseok Moon, Xiao Luo, Eugene Chen, Adrian Ong, Alexander Driskill-Smith, and Mohamad Krounbi. 2013. Spin-transfer Torque Magnetic Random Access Memory (STT-MRAM). *J. Emerg. Technol. Comput. Syst.* (May 2013).
[11] Rajeev Balasubramonian, Andrew B. Kahng, Naveen Muralimanohar, Ali Shafiee, and Vaishnav Srinivas. 2017. CACTI 7: New Tools for Interconnect Exploration in Innovative Off-Chip Memories. *ACM Trans. Archit. Code Optim.* 14, 2, Article 14 (June 2017), 25 pages. https://doi.org/10.1145/3085572
[12] Abhishek Bhattacharjee. 2017. Translation-Triggered Prefetching. In *the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*.
[13] Nadav Chachmon, Daniel Richins, Robert Cohn, Magnus Christensson, Wenzhi Cui, and Vijay Janapa Reddi. 2016. Simulation and Analysis Engine for Scale-Out Workloads. In *2016 International Conference on Supercomputing*. Article 22, 13 pages.
[14] M. Chang, P. Rosenfeld, S. Lu, and B. Jacob. 2013. Technology comparison for large last-level caches (L3Cs): Low-leakage SRAM, low write-energy STT-RAM, and refresh-optimized eDRAM. In *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*.
[15] Y. Chen, J. Cong, H. Huang, B. Liu, C. Liu, M. Potkonjak, and G. Reinman. 2012. Dynamically reconfigurable hybrid cache: An energy-efficient last-level cache design. In *2012 Design, Automation Test in Europe Conference Exhibition (DATE)*.
[16] Y. Chen, H. Li, X. Wang, W. Zhu, W. Xu, and T. Zhang. 2012. A 130 nm 1.2 V/3.3 V 16 Kb Spin-Transfer Torque Random Access Memory With Nondestructive Self-Reference Sensing Scheme. *IEEE Journal of Solid-State Circuits* (Feb 2012).
[17] H. Cheng, J. Zhao, J. Sampson, M. J. Irwin, A. Jaleel, Y. Lu, and Y. Xie. 2016. LAP: Loop-Block Aware Inclusion Properties for Energy-Efficient Asymmetric Last Level Caches. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*.
[18] Y. Chih, Y. Shih, C. Lee, Y. Chang, P. Lee, H. Lin, Y. Chen, C. Lo, M. Shih, K. Shen, H. Chuang, and T. J. Chang. 2020. 13.3 A 22nm 32Mb Embedded STT-MRAM with 10ns Read Speed, 1M Cycle Write Endurance, 10 Years Retention at 150 C and High Immunity to Magnetic Field Interference. In *2020 IEEE International Solid- State Circuits Conference - (ISSCC)*. 222–224.
[19] A. Chintaluri, H. Naeimi, S. Natarajan, and A. Raychowdhury. 2016. Analysis of Defects and Variations in Embedded Spin Transfer Torque (STT) MRAM Arrays. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* (Sep. 2016).
[20] Greg Hamerly and E. Perelman and Jeremy Lau and B. Calder. 2005. SimPoint 3.0: Faster and More Flexible Program Phase Analysis. *J. Instr. Level Parallelism* 7 (2005).
[21] S. Gupta and H. Zhou. 2015. Spatial Locality-Aware Cache Partitioning for Effective Cache Sharing. In *2015 44th International Conference on Parallel Processing*.
[22] John L. Henning. 2006. SPEC CPU2006 Benchmark Descriptions. *SIGARCH Comput. Archit. News* 34, 4 (Sept. 2006). https://doi.org/10.1145/1186736.1186737
[23] K. Ikegami, H. Noguchi, S. Takaya, C. Kamata, M. Amano, K. Abe, K. Kushida, E. Kitagawa, T. Ochiai, N. Shimomura, D. Saida, A. Kawasumi, H. Hara, J. Ito, and S. Fujita. 2015. MTJ-based "normally-off processors" with thermal stability factor engineered perpendicular MTJ, L2 cache based on 2T-2MTJ cell, L3 and last level cache based on 1T-1MTJ cell and novel error handling scheme. In *2015 IEEE International Electron Devices Meeting (IEDM)*. 25.1.1–25.1.4.
[24] G. Jan, L. Thomas, S. Le, Y. Lee, H. Liu, J. Zhu, R. Tong, K. Pi, Y. Wang, D. Shen, R. He, J. Haq, J. Teng, V. Lam, K. Huang, T. Zhong, T. Torng, and P. Wang. 2014. Demonstration of fully functional 8Mb perpendicular STT-MRAM chips with sub-5ns writing for non-volatile embedded memories. In *2014 Symposium on VLSI Technology (VLSI-Technology)*.
[25] H. Jang, Y. Lee, J. Kim, Y. Kim, J. Kim, J. Jeong, and J. W. Lee. 2016. Efficient footprint caching for Tagless DRAM Caches. In *2016 IEEE International Symposium on High Performance Computer Architecture*. 237–248.
[26] D. Jevdjic, G. H. Loh, C. Kaynak, and B. Falsafi. 2014. Unison Cache: A Scalable and Effective Die-Stacked DRAM Cache. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*. 25–37.
[27] Djordje Jevdjic, Stavros Volos, and Babak Falsafi. 2013. Die-Stacked DRAM Caches for Servers: Hit Ratio, Latency, or Bandwidth? Have It All with Footprint Cache. In *Proceedings of the 40th Annual International Symposium on Computer Architecture*. https://doi.org/10.1145/2485922.2485957
[28] X. Jiang, N. Madan, L. Zhao, M. Upton, R. Iyer, S. Makineni, D. Newell, Y. Solihin, and R. Balasubramonian. 2010. CHOP: Adaptive filter-based DRAM caching for CMP server platforms. In *HPCA - 16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture*. 1–12.
[29] A. Jog, A. K. Mishra, C. Xu, Y. Xie, V. Narayanan, R. Iyer, and C. R. Das. 2012. Cache revive: Architecting volatile STT-RAM caches for enhanced performance in CMPs. In *DAC Design Automation Conference*.
[30] Dongwoo Kang, Seungjae Baek, Jongmoo Choi, Donghee Lee, Sam H. Noh, and Onur Mutlu. 2015. Amnesic cache management for non-volatile memory. In *31st Symposium on Mass Storage Systems and Technologies (MSST)*. 1–13. https://doi.org/10.1109/MSST.2015.7208291

[31] E. Karl, Z. Guo, J. Conary, J. Miller, Y. Ng, S. Nalam, D. Kim, J. Keane, X. Wang, U. Bhattacharya, and K. Zhang. 2016. A 0.6 V, 1.5 GHz 84 Mb SRAM in 14 nm FinFET CMOS Technology With Capacitive Charge-Sharing Write Assist Circuitry. *IEEE Journal of Solid-State Circuits* 51, 1 (2016), 222–229.

[32] T. Kawahara, R. Takemura, K. Miura, J. Hayakawa, S. Ikeda, Y. Lee, R. Sasaki, Y. Goto, K. Ito, T. Meguro, F. Matsukura, H. Takahashi, H. Matsuoka, and H. Ohno. 2007. 2Mb Spin-Transfer Torque RAM (SPRAM) with Bit-by-Bit Bidirectional Current Write and Parallelizing-Direction Current Read. In *2007 IEEE International Solid-State Circuits Conference*.

[33] A V Khvalkovskiy, 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. 2013. Basic principles of STT-MRAM cell operation in memory arrays. *Journal of Physics D: Applied Physics* (feb 2013).

[34] Youngbae Kim, Shreyash Patel, Heekyung Kim, Nandakishor Yadav, and Kyuwon Choi. 2021. Ultra-Low Power and High-Throughput SRAM Design to Enhance AI Computing Ability in Autonomous Vehicles. *Electronics* 10 (01 2021), 256. https://doi.org/10.3390/electronics10030256

[35] A. Kokolis, D. Skarlatos, and J. Torrellas. 2019. PageSeer: Using Page Walks to Trigger Page Swaps in Hybrid Memory Systems. In *2019 IEEE 25th International Symposium on High Performance Computer Architecture*.

[36] Aasheesh Kolli, Jeff Rosen, Stephan Diestelhorst, Ali Saidi, Steven Pelley, Sihang Liu, Peter M. Chen, and Thomas F. Wenisch. 2016. Delegated Persist Ordering. In *The 49th Annual IEEE/ACM International Symposium on Microarchitecture* (Taipei, Taiwan).

[37] K. Korgaonkar, I. Bhati, H. Liu, J. Gaur, S. Manipatruni, S. Subramoney, T. Karnik, S. Swanson, I. Young, and H. Wang. 2018. Density Tradeoffs of Non-Volatile Memory as a Replacement for SRAM Based Last Level Cache. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture*.

[38] Y. Lee, J. Kim, H. Jang, H. Yang, J. Kim, J. Jeong, and J. W. Lee. 2015. A fully associative, tagless DRAM cache. In *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture*. 211–222.

[39] Sheng Li, Jung Ho Ahn, Richard D. Strong, Jay B. Brockman, Dean M. Tullsen, and Norman P. Jouppi. 2009. McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*. 469–480.

[40] G. H. Loh and M. D. Hill. 2011. Efficiently enabling conventional block sizes for very large die-stacked DRAM caches. In *2011 44th Annual IEEE/ACM International Symposium on Microarchitecture*. 454–464.

[41] Y. Lu, T. Zhong, W. Hsu, S. Kim, X. Lu, J. J. Kan, C. Park, W. C. Chen, X. Li, X. Zhu, P. Wang, M. Gottwald, J. Fatehi, L. Seward, J. P. Kim, N. Yu, G. Jan, J. Haq, S. Le, Y. J. Wang, L. Thomas, J. Zhu, H. Liu, Y. J. Lee, R. Y. Tong, K. Pi, D. Shen, R. He, Z. Teng, V. Lam, R. Annapragada, T. Torng, P. Wang, and S. H. Kang. 2015. Fully functional perpendicular STT-MRAM macro embedded in 40 nm logic for energy-efficient IOT applications. In *2015 IEEE International Electron Devices Meeting (IEDM)*.

[42] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner. 2002. Simics: A full system simulation platform. *Computer* 35, 2 (Feb 2002), 50–58. https://doi.org/10.1109/2.982916

[43] M. Mao, G. Sun, Y. Li, A. K. Jones, and Y. Chen. 2014. Prefetching techniques for STT-RAM based last-level cache in CMP systems. In *2014 19th Asia and South Pacific Design Automation Conference*.

[44] H. Noguchi, K. Ikegami, K. Kushida, K. Abe, S. Itai, S. Takaya, N. Shimomura, J. Ito, A. Kawasumi, H. Hara, and S. Fujita. 2015. A 3.3ns-access-time 71.2μW/MHz 1Mb embedded STT-MRAM using physically eliminated read-disturb scheme and normally-off memory architecture. In *2015 IEEE International Solid-State Circuits Conference*.

[45] H. Noguchi, K. Ikegami, N. Shimomura, T. Tetsufumi, J. Ito, and S. Fujita. 2014. Highly reliable and low-power nonvolatile cache memory with advanced perpendicular STT-MRAM for high-performance CPU. In *2014 Symposium on VLSI Circuits*.

[46] H. Noguchi, K. Ikegami, S. Takaya, E. Arima, K. Kushida, A. Kawasumi, H. Hara, K. Abe, N. Shimomura, J. Ito, S. Fujita, T. Nakada, and H. Nakamura. 2016. 4Mb STT-MRAM-based cache with memory-access-aware power optimization and write-verify-write / read-modify-write scheme. In *2016 IEEE International Solid-State Circuits Conference*.

[47] T. Ohsawa, H. Koike, S. Miura, H. Honjo, K. Kinoshita, S. Ikeda, T. Hanyu, H. Ohno, and T. Endoh. 2013. A 1 Mb Nonvolatile Embedded Memory Using 4T2MTJ Cell With 32 b Fine-Grained Power Gating Scheme. *IEEE Journal of Solid-State Circuits* (June 2013).

[48] T. Ohsawa, S. Miura, K. Kinoshita, H. Honjo, S. Ikeda, T. Hanyu, H. Ohno, and T. Endoh. 2013. A 1.5nsec/2.1nsec random read/write cycle 1Mb STT-RAM using 6T2MTJ cell with background write for nonvolatile e-memories. In *2013 Symposium on VLSI Circuits*.

[49] S. Panda, N. M. Kumar, and C.K. Sarkar. 2009. Power, delay and noise optimization of a SRAM cell using a different threshold voltages and high performance output noise reduction circuit. In *2009 4th International Conference on Computers and Devices for Communication (CODEC)*. 1–4.

[50] Steven Pelley, Peter M. Chen, and Thomas F. Wenisch. 2014. Memory Persistency. In *Proceeding of the 41st Annual International Symposium on Computer Architecuture* (Minneapolis, Minnesota, USA).

[51] M. K. Qureshi and G. H. Loh. 2012. Fundamental Latency Trade-off in Architecting DRAM Caches: Outperforming Impractical SRAM-Tags with a Simple and Practical Design. In *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*. 235–246.

[52] A. F. Rodrigues, K. S. Hemmert, B. W. Barrett, C. Kersey, R. Oldfield, M. Weston, R. Risen, J. Cook, P. Rosenfeld, E. CooperBalls, and B. Jacob. 2011. The Structural Simulation Toolkit. *SIGMETRICS Perform. Eval. Rev.* 38, 4 (March 2011).

[53] P. Rosenfeld, E. Cooper-Balis, and B. Jacob. 2011. DRAMSim2: A Cycle Accurate Memory System Simulator. *IEEE Computer Architecture Letters* (Jan 2011).

[54] S. Sakhare, M. Perumkunnil, T. H. Bao, S. Rao, W. Kim, D. Crotti, F. Yasin, S. Couet, J. Swerts, S. Kundu, D. Yakimets, R. Baert, H. Oh, A. Spessot, A. Mocuta, G. S. Kar, and A. Furnemont. 2018. Enablement of STT-MRAM as last level cache for the high performance computing domain at the 5nm node. In *2018 IEEE International Electron Devices Meeting (IEDM)*. 18.3.1–18.3.4. https://doi.org/10.1109/IEDM.2018.8614637

[55] Shikha Saun and Hemant Kumar. 2019. Design and performance analysis of 6T SRAM cell on different CMOS technologies with stability characterization. *IOP Conference Series Materials Science and Engineering* 561 (11 2019), 1–9. https://doi.org/10.1088/1757-899X/561/1/012093

[56] A. Shafaei, Y. Wang, X. Lin, and M. Pedram. 2014. FinCACTI: Architectural Analysis and Modeling of Caches with Deeply-Scaled FinFET Devices. In *2014 IEEE Computer Society Annual Symposium on VLSI*. 290–295.

[57] C. W. Smullen, V. Mohan, A. Nigam, S. Gurumurthi, and M. R. Stan. 2011. Relaxing non-volatility for fast and energy-efficient STT-RAM caches. In *2011 IEEE 17th International Symposium on High Performance Computer Architecture*.

[58] Sanjana S.R., Balaji Ramakrishna, Samiksha, Roohila Banu, and Prateek Shubham. 2017. Design and Performance Analysis of 6T SRAM Cell in 22nm CMOS and FINFET Technology Nodes. In *Proceedings of the 2017 International Conference on Recent Advances in Electronics and Communication Technology* (Bangalore, India) *(ICRAECT'17)*. IEEE.

[59] G. Sun, X. Dong, Y. Xie, J. Li, and Y. Chen. 2009. A novel architecture of the 3D stacked MRAM L2 cache for CMPs. In *2009 IEEE 15th International Symposium on High Performance Computer Architecture*.

[60] Z. Sun, X. Bi, H. Li, W. Wong, Z. Ong, X. Zhu, and W. Wu. 2011. Multi retention level STT-RAM cache designs with a dynamic refresh scheme. In *2011 44th Annual IEEE/ACM International Symposium on Microarchitecture*.

[61] J. Wang, X. Dong, and Y. Xie. 2013. OAP: An obstruction-aware cache management policy for STT-RAM last-level caches. In *2013 Design, Automation Test in Europe Conference Exhibition*.

[62] Z. Wang, X. Hao, P. Xu, L. Hu, D. Jung, W. Kim, K. Satoh, B. Yen, Z. Wei, L. Wang, J. Zhang, and Y. Huai. 2020. STT-MRAM for Embedded Memory Applications. In *2020 IEEE International Memory Workshop (IMW)*. 1–3.

[63] Z. Wang, D. A. Jimenez, C. Xu, G. Sun, and Y. Xie. 2014. Adaptive placement and migration policy for an STT-RAM-based hybrid cache. In *2014 IEEE 20th International Symposium on High Performance Computer Architecture*.

[64] Dong Hyuk Woo, Nak Hee Seong, Dean L. Lewis, and Hsien-Hsin S. Lee. 2010. An optimized 3D-stacked memory architecture by exploiting excessive, high-density TSV bandwidth. In *HPCA - 16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture*. 1–12. https://doi.org/10.1109/HPCA.2010.5416628

[65] Xiaoxia Wu, Jian Li, Lixin Zhang, Evan Speight, Ram Rajamony, and Yuan Xie. 2009. Hybrid Cache Architecture with Disparate Memory Technologies. In *Proceedings of the 36th Annual International Symposium on Computer Architecture*.

[66] W. Xu, H. Sun, X. Wang, Y. Chen, and T. Zhang. 2011. Design of Last-Level On-Chip Cache Using Spin-Torque Transfer RAM (STT RAM). *IEEE Transactions on Very Large Scale Integration Systems* (March 2011).

[67] J. Zhan, O. Kayiran, G. H. Loh, C. R. Das, and Y. Xie. 2016. OSCAR: Orchestrating STT-RAM cache traffic for heterogeneous CPU-GPU architectures. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture*.

[68] L. Zhang, B. Neely, D. Franklin, D. Strukov, Y. Xie, and F. T. Chong. 2016. Mellow Writes: Extending Lifetime in Resistive Memories through Selective Slow Write Backs. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture*.

[69] Y. Zhang, Y. Li, Z. Sun, H. Li, Y. Chen, and A. K. Jones. 2015. Read Performance: The Newest Barrier in Scaled STT-RAM. *IEEE Transactions on Very Large Scale Integration Systems* (June 2015).