Energy and Performance Analysis of STTRAM Caches for Mobile Applications

Kyle Kuan and Tosiron Adegbija
Department of Electrical & Computer Engineering
University of Arizona, Tucson, AZ, USA
Email: {ckkuan, tosiron}@email.arizona.edu

***Abstract*—Spin-Transfer Torque RAMs (STTRAMs) have been shown to offer much promise for implementing emerging cache architectures. This paper studies the viability of STTRAM caches for mobile workloads from the perspective of energy and latency. Specifically, we explore the benefits of reduced retention STTRAM caches for mobile applications. We analyze the characteristics of mobile applications' cache blocks and how those characteristics dictate the appropriate retention time for mobile device caches. We show that due to their inherently interactive nature, mobile applications' execution characteristics—and hence, STTRAM cache design requirements—differ from other kinds of applications. We also explore various STTRAM cache designs in both single and multicore systems, and at different cache levels, that can efficiently satisfy mobile applications' execution requirements, in order to maximize energy savings without introducing substantial latency overhead.**

***Index Terms*—Spin-Transfer Torque RAM (STTRAM) cache, mobile applications; performance analysis; retention time; non-volatile memory; energy efficient; multicore processor.**

I. Introduction

The past few years have witnessed a mobile tipping point wherein more mobile devices (e.g., smartphones and tablets) are being sold and used than all other kinds of computers combined. In addition, mobile devices continue to run increasingly complex applications in line with users' increasing demands for high-performance, low-energy systems. As a result, the processors featured in mobile devices are becoming more sophisticated, with advanced microarchitecture optimizations, such as out-of-order execution, deep pipelines, multi-level cache hierarchies, asymmetric configurations, etc. [1].

While mobile devices contain several components that impact energy and performance, such as the display and radios, the cache subsystem remains one of the most important components of mobile device processors. The cache bridges the processor-memory performance gap, and is consequential to the processor's overall energy efficiency [2]. As such, there is much ongoing research into technologies for enabling energy efficient caches for resource-constrained processors.

An increasingly popular approach for improving caches' energy efficiency involves replacing the traditional SRAM with emerging non-volatile memory (NVM) technologies. The spin-transfer torque RAM (STTRAM) [3], especially, is attractive for implementing on-chip memories, due to several characteristics, such as high density, higher reliability (compared to other NVMs), etc. There is also much ongoing research to mitigate STTRAM's overheads of high write latency, high write energy, reliability issues, etc [4]. However, to maximize the benefits of STTRAM caches in mobile devices, we must first understand the execution characteristics of mobile applications within the context of STTRAM caches' unique characteristics.

This paper aims to explore and analyze the energy and performance benefits of STTRAM caches for mobile applications. Specifically, we approach our analysis from the perspective of *reduced retention STTRAM caches* [5]. Prior work showed that STTRAM's long write latency and high write energy can be attributed to the long retention time—the duration for which data is maintained in the memory in the absence of power. For caches, the intrinsic STTRAM retention time of up to 10 years is unnecessary, since most cache blocks need to be retained in the cache for no longer than $1s$ [6]. As such, the retention time can be substantially reduced to mitigate the write overheads. We note that other techniques for addressing STTRAM's write overheads have been proposed, but we limit the studies herein to reduced retention STTRAM caches, hereafter simply referred to as 'STTRAM caches.'

In order to maximize the benefits of STTRAM caches for mobile applications, the retention time must suffice for the applications' *cache block lifetimes*. Similar to prior studies on SRAM caches [7], we define cache block lifetimes as the duration for which cache blocks must remain in the cache before they are evicted or invalidated. Given that most mobile applications are intrinsically interactive, and thus exhibit execution characteristics that may differ from other non-interactive applications, it is imperative to study STTRAM caches in the context of mobile applications' characteristics.

In this paper, we analyze mobile applications with respect to their cache block lifetimes, and other execution characteristics (e.g., read-write behavior) that impact how well STTRAM caches are matched to mobile applications' execution needs. Furthermore, we explore the behavior of mobile applications on STTRAM caches in single core and multicore processors, and derive new insights, based on our analysis, on the tradeoffs of STTRAM caches within these processor contexts.

II. Background and Related Work

A. Overview of STTRAM

Similar to other resistive memories, STTRAM uses non-volatile, resistive information storage in a cell. Fig. 1 illustrates

This work was supported in part by NSF Grant CAREER 1844952.

STTRAM's basic structure, comprising of a magnetic tunnel junction (MTJ) and a transistor. The MTJ cell, which is used as the binary storage cell, contains two ferromagnetic layers (the free and fixed layers) separated by an oxide barrier/tunnel layer. The free layer's direction with respect to the fixed layer (parallel or anti-parallel) generates low resistance and high resistance states of the MTJ cell, to indicate the "0" or "1" bit. The magnetization change in the free layer is controlled by a transistor, which allows current to flow through the MTJ cell and creates a spin torque that switches the magnetization in the free layer. We direct the reader to [8] for additional low-level details of STTRAM's basic structure.

B. Overview of STTRAM-based Analysis

There has been much prior work on the benefits of replacing SRAM with STTRAM. For instance, Noguchi et al. [9] showed that replacing SRAM with STT-RAM reduced the last level cache (LLC) energy by 60%, with a 2% performance degradation. However, STTRAM still has drawbacks that have slowed down the adoption of STTRAM in emerging processor architectures. Notably, STTRAMs' high write energy and latency overheads have attracted much research attention. One research thread involves relaxing STTRAM's thermal stability to reduce the retention time and thus, the write energy and latency [10], [5]. Alternatively, prior work has explored other techniques such as dynamic block allocation [11] for use in hybrid (SRAM+STTRAM) caches, wherein write-active cache blocks are written into SRAM, while read-active blocks are written into STTRAM. Other techniques involve identifying and eliminating redundant writes or utilizing opportunistic replacement policies in order to minimize the write overheads [12], [13], [14], [15]. Recently, Kuan et al. [16] studied the characteristics of different SPEC 2006 benchmarks [17] and showed that the STTRAM cache energy could be further improved by adapting the retention time to different applications' characteristics, without incurring much optimization overhead.

Yan et al. [18] proposed to partition the L2 cache for user and kernel accesses in mobile devices based on in the variability in access patterns between these two kinds of accesses. However, to the best of our knowledge, there is no prior work that has extensively analyzed the benefits of reduced retention STTRAM caches for mobile applications. Most of the aforementioned prior works used desktop or high performance benchmark suites like SPEC2006 and PARSEC [19] in their analysis. Given that STTRAMs offer multiple advantages (e.g., low leakage, high density) for resource-constrained systems, we anticipate that STTRAM caches will play an important role in emerging mobile computing systems. Furthermore, since mobile applications' execution characteristics differ drastically from traditional benchmarks, due in part to mobile applications' interactive nature [20], it is imperative to analyze STTRAM caches with relevant benchmarks that represent mobile applications' characteristics [18].

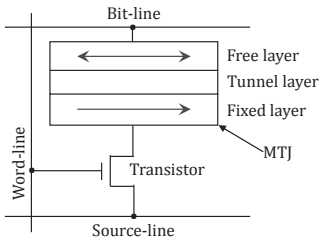


Fig. 1: STT-RAM basic cell structure

III. Methodology

This section describes our methodology for gathering the data for the analysis presented herein. We first describe some design assumptions made, and thereafter, present our experimental setup.

A. Design Assumptions

We assume that STTRAM caches can be fabricated as desired with different retention times. We note that reducing the retention time trades off other factors like the reliability [21]. As such, there is much on-going research to address this and other fabrication and device challenges of STTRAMs (e.g., process variation) [22], but addressing these challenges is outside the scope of this paper. This subsection summarizes modeling techniques that we have employed for achieving reduced retention times and for preventing data corruption in reduced retention STTRAM caches.

1) Achieving Reduced Retention Times: Prior work showed that STTRAM's thermal stability (Δ) can be substantially reduced to lower the write energy and latency [5]. This is more so beneficial for storage media that don't require long retention, such as CPU caches. Thus, to model the STTRAM caches analyzed in this paper, we followed the technique proposed in [8]. We decreased the MTJ's planar area to obtain the desired retention times and lower thermal stability, and used the models described in [8] to determine the MTJ characteristics for different retention times. Based on these characteristics, we calculated write pulse, write current, and MTJ resistance values R_{AP} and R_P.

2) Preventing Data Corruption: A challenge that arises after reducing STTRAM cache's retention time is that some cache blocks may need to remain in the cache beyond the cache's predetermined retention time. As such, the data could become unstable or corrupt, and could cause incorrect results if reused by the CPU. Thus, to prevent data corruption, we incorporate a per-block counter to keep track of the cache blocks' lifetimes [6], [7]. The counter detects the expiration of a cache block and evicts the block just before the retention time elapses. Dirty blocks are first written to main memory before eviction. We implemented the counter as a finite state machine, with a clock period defined as the retention time divided by N, where N dictates the granularity of block eviction. When a block is written to the cache, the counter's state advances from the initial state until it reaches the maximum state. The block is then evicted, and the counter is reset to the initial state whenever a new fetch operation occurs for the block. We note that this is a low overhead technique that only requires a few bits per block. The implementation in our experiments only required two bits per block for a four-state (N = 4) counter.

B. Experimental Setup and Workloads

For the analysis presented herein, we used an in-house modified version of the GEM5 simulator [23]. The modified GEM5[1] models the behavior of relaxed retention STTRAM caches with specified retention times. We modeled single- and quad-core processors with base configurations similar to those featured in modern mobile devices (e.g., ARM Cortex-A76). The processor featured a 1.9GHz clock frequency, out-of-order execution, private level one (L1) instruction and data caches, and shared unified L2 caches (for the multicore experiments). The L1 caches had 32KB size, 4-way set associative, and 64B line size, while the L2 cache had 2MB size, 16-way set associative, and 64B line size.

We considered retention times ranging from $1\mu s$ to $100ms$ in *10 increments. Later in our analysis, we focus on only the $1ms$, $10ms$, and $100ms$ retention times, since we found that smaller retention times were severely under-provisioned for all the mobile applications considered with respect to energy and latency. To model STTRAM and SRAM cache energy, we used NVSim [24] integrated with the GEM5 statistics. We used the MTJ cell modeling technique proposed in [8] to obtain essential parameters, such as the write pulse, write current, and MTJ resistance value R_{AP}, and then applied these parameters to NVSim to construct the STTRAM caches.

To represent mobile applications, we used ten benchmarks from the *Moby* benchmark suite [25]. Moby comprises of a variety of popular android applications, including web browser, social networking, email client, music player, map, document processing, etc., all of which are selected from the Google Play Store. The benchmarks were run in GEM5 Full System mode using Android Ice Cream Sandwich (ICS) as the operating system. We ran all the benchmarks to completion after skipping the boot process using checkpointing.

IV. STTRAM-Aware Characterization

In the analysis presented in this section, we focus on STTRAM-specific insights, since the mobile applications have been analyzed with respect to SRAM caches in prior work [25]. We first analyze the characteristics of mobile applications in the context of STTRAM caches, with respect to the *read-write activity*, *cache block lifetimes*, and *expiration misses*; these characteristics directly affect the applications' performance on STTRAM caches. In the following section, we analyze the energy and performance (latency) of mobile applications in different STTRAM cache design scenarios.

A. Read-Write Activity

An application's read-write activity—the ratio of reads to writes—can substantially affect the efficiency of executing those applications on STTRAM memories. Since STTRAMs suffer the most overheads during write operations (due to the high write energy and latency), applications with more reads than writes tend to benefit more from STTRAM caches. Thus, we analyzed the read-write behavior of the mobile applications in the context of single and multicore systems.

[1]The modified GEM5 version can be found at www.ece.arizona.edu/tosiron/downloads.php

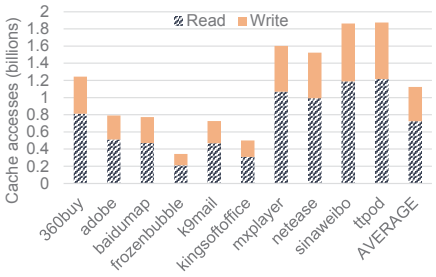


Fig. 2: Read-write ratio for data cache memory accesses

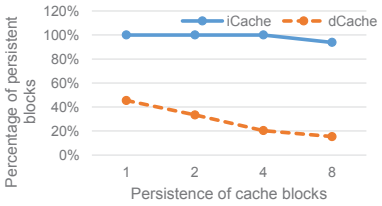


Fig. 3: Percentage of persistent blocks. The results depicted are on average across all the benchmarks.

Figure 2 depicts the read-write ratio of data cache accesses (in billions of instructions) of the mobile benchmarks in a single core execution. Even though the total number of accesses differed substantially across the different applications, we observed that the read-write ratio was relatively stable across the applications. Overall, the reads were, on average, 67% of total accesses, ranging from 66% to 70%.

It is possible for an application's threads to exhibit different read-write behaviors when they are distributed across different cores vs when the application is running on a single core. Thus, we also analyzed the read-write behaviors across different threads in a multicore scenario. The trends were consistent across all the threads running on different cores: on average across all the benchmarks, the reads were 64% of the total accesses, ranging from 59% to 73%. This high proportion of reads suggests that the considered mobile applications, in general, would consistently be less susceptible to the overheads of STTRAM caches than applications that exhibit high variability in the read-write ratios, such as SPEC benchmarks [6].

B. Cache Block Lifetimes and Persistence

Cache block lifetimes and reuse are interrelated and refer to how long a cache block must remain in the cache before it is evicted or invalidated. The cache block lifetime directly affects an application's retention time requirements and, in effect, an STTRAM cache's performance for the application. We studied the mobile applications' cache block lifetimes to derive insights into their retention time requirements. We observed that, on average, the cache block lifetimes of the mobile applications varied across the different applications, but mostly ranged from $1ms$ to $100ms$. These observations about the cache block lifetimes dictate the retention times that perform best for the different applications, as will become more evident in subsequent sections.

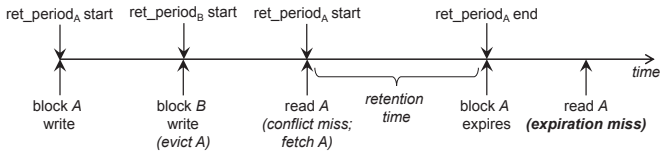


Fig. 4: Illustration of expiration miss.

To further understand the behavior of mobile applications in the context of STTRAM caches, we analyzed the percentage of unique cache blocks that were persistent, given a *persistence threshold (thd)*. We define a block as persistent if it is loaded into cache at least *thd* times after it is evicted from the cache. The persistence of a cache block can provide a sense of how long the block must *ideally* remain in the cache to service all its future references. We analyzed the blocks' persistence for both instruction and data caches with *thd* values of 1 to 8, in power-of-two increments.

Figure 3 depicts the persistence of instruction and data cache blocks for persistence thresholds of 1, 2, 4, and 8. In the instruction cache, 100% of cache blocks were loaded more than four times, and 94% were loaded more than eight times, revealing a high degree of persistence. In the data cache, on the other hand, only 45% of cache blocks were loaded more than once, and 15% were loaded more than eight times. We also observe a much sharper drop in the blocks' persistence as the persistence threshold increases from 1 to 8 for the data cache. On average across all the benchmarks, the data cache had substantially more (72x, on average) unique blocks referenced than the instruction cache. However, the instruction cache's total accesses exceeded the data cache's by more than 2x, due to the high persistence of the instruction cache blocks.

C. Expiration Misses

In STTRAM caches, an important characteristic that can significantly impact the energy and latency is what we call the *expiration misses*. The expiration misses are introduced by reduced retention times and refer to the misses that result from references to a block that was prematurely evicted due to elapsed retention time. Figure 4 illustrates the occurrence of an expiration miss. Assuming a block A and B reside in the same cache set location, and a write of A, followed by a write of B evicts A, a subsequent read request for A would result in a conflict miss. A is then fetched and its retention period is started. When the retention time elapses, A expires, and a subsequent reference to A results in an expiration miss.

Figure 5 depicts the average number of expiration misses for different retention times. The figure depicts the average across all the benchmarks, since the trends were similar for the different benchmarks. Unsurprisingly, as the retention time increased, the number of expiration misses decreased substantially. However, we observed a much sharper decrease for the data cache than for the instruction cache. For instance, the decrease from $1\mu s$ to $10\mu s$, $10\mu s$ to $100\mu s$, $100\mu s$ to $1ms$, and $1ms$ to $10ms$ were approximately 3x, 9x, 13x, and 19x,

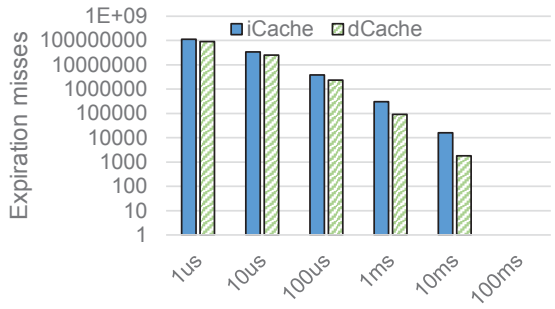
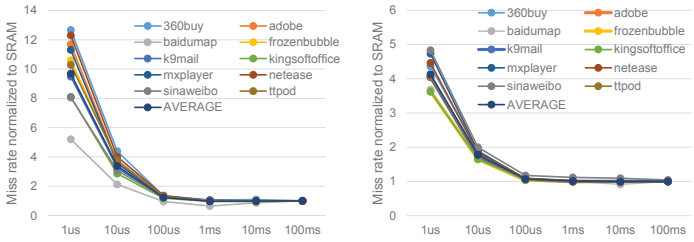


Fig. 5: Average expiration misses for different retention times.



(a) Instruction cache (b) Data cache

Fig. 6: Instruction and data cache miss rates for different retention times normalized to SRAM

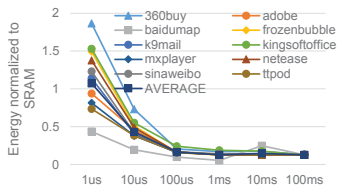
respectively, for the instruction cache; for the data cache, the decreases were 4x, 11x, 25x, and 51x, respectively. Also, there were substantially more expiration misses in the instruction cache than the data cache, and the difference increased as the retention time increased—ranging from 1.2x at $1\mu s$ to 9x at $10ms$. For both the instruction and data caches, the number of expiration misses was 0 on the $100ms$ retention time.

In general, expiration misses of zero is ideal. However, we also found that the expiration misses is an insufficient criterion for evaluating the benefits of STTRAM cache and different retention times for executing applications. Some applications that have low persistence blocks as majority may tolerate expiration misses better than others, depending on the kinds of misses that occurred. An increase in expiration misses did not necessarily increase energy or latency. For instance, a single miss may accrue less energy and latency overhead than another miss due to future accesses that depend on the miss. Overall, we found that most of the applications were able to tolerate some expiration misses. As such, even though $100ms$ resulted in zero expiration misses, it was not necessarily the best for all applications, as will become clear in the following sections.

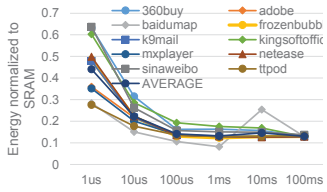
V. Single Core Evaluation

A. Cache Miss Rates and Energy

First, we analyze the benefits of STTRAM caches in the context of a single core processor. Since mobile applications typically execute in energy-constrained environments (i.e., battery-powered mobile devices), the goal is an STTRAM cache that minimizes the energy without substantially increasing the latency compared to SRAM. Thus, we first explore how the instruction and data cache miss rates and energy

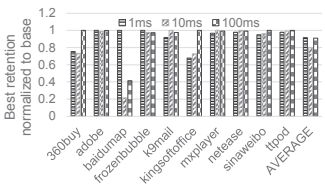


(a) Instruction cache

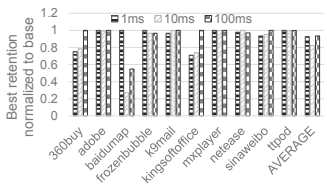


(b) Data cache

Fig. 7: Instruction and data cache energy consumption for different retention times normalized to SRAM



(a) Cache energy



(b) Execution time

Fig. 8: Benefits of variable retention times. The best retention time for (a) energy and (b) execution time for each benchmark is normalized to a base of $1ms$, $10ms$, and $100ms$. The instruction and data caches exhibited similar energy trends.

consumption change for different retention times, and in comparison to SRAM.

Figure 6 depicts the instruction and data cache miss rates of different retention times normalized to SRAM. As expected, as the retention time increased, the cache miss rates trended towards SRAM for both the instruction and data caches. Importantly, we observed that both the data (Figure 6b) and instruction (Figure 6a) caches exhibit high variability in cache miss rates. That is, different applications require different retention times—for both the instruction and data caches—to achieve the lowest miss rates. This observation is contrary to prior that studied SPEC benchmarks and showed that variability only exists in the data cache, while a single retention time sufficed for the instruction cache across all the SPEC benchmarks [16]. We attribute our observation to the interactive nature of mobile applications, which introduces variability to the instruction cache behavior [20].

For the energy consumption, we observed similar trends between the instruction and data caches. However, there were a few differences. Figure 7 depicts the instruction and data cache energy consumption trends for different retention times normalized to SRAM. Even though lower retention times would consume less energy per access than SRAM, the behavior is different in the context of application execution. For both the instruction (Figure 7a) and data (Figure 7b) caches, the smallest retention times ($1\mu s$ and $10\mu s$) substantially increased the cache accesses, and in effect the energy consumption.

In the instruction cache, despite the substantial energy savings of low retention times, the cache accesses due to expiration misses (Section IV-C) were so substantially increased that the energy consumption at $1\mu s$ was *more* than SRAM for some applications (e.g., $360buy, kingsoftoffice, netease,$ etc.). For instance, on average[2], the $1\mu s$ retention time *increased* the energy consumption by 7.1%, and by up to 86.4% for $360buy$. The average energy savings for the other retention times ranged from 57% for $10\mu s$ to 87.3% for $1\mu s$, with little variance for the $100\mu s$, $1ms$, $10ms$, and $100ms$ retention times. For the data cache, on the other hand, none of the retention times exceeded SRAM in energy consumption despite the increase in cache accesses at lower retention times. The average energy savings ranged from 55.9% for $1\mu s$ to 87% for $100ms$.

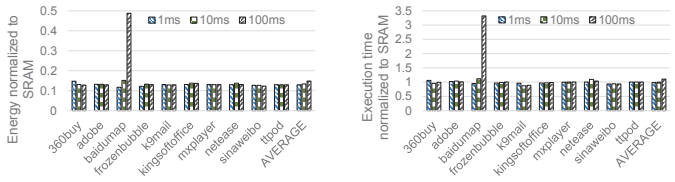
[2]We used the geometric mean for the averages discussed in the evaluations.

These results illustrate the variable retention time needs of different mobile applications. Due to the cache block lifetimes (Section IV-B), the best retention times for energy ranged from $1ms$ to $100ms$, with a majority at $10ms$—five and seven benchmarks for the instruction and data cache, respectively. None of the benchmarks required less than $1ms$ for either cache for minimum energy.

B. Benefits of Retention Time Specialization

Prior work (e.g., [16]) suggested using a variety of cache units with different retention times in a single chip to enable close specialization to applications' retention time needs. Thus, we also explored the benefits of specializing the retention time to individual benchmarks' needs vs. using a base configuration of $1ms$, $10ms$, or $100ms$. We selected these three retention times as base, since they covered the best options for all the benchmarks. A design option for such a variable retention time system is described in [16], wherein the cache chip is designed with different cache units featuring different retention times to satisfy a variety of applications' needs. During runtime, the applications' cache accesses are serviced by the cache unit that best satisfies the applications' needs as determined via sampling or a tuning algorithm.

Figure 8 depicts the benefits of such a system in a single core. We assumed a design similar to [16], with a sampling technique, which samples the application on each cache unit for a sampling interval of 10M instructions to determine the best retention time for different objective functions (energy or latency). The energy savings were similar for both the instruction and data caches. On average across the benchmarks, the best retention times improved the energy for the instruction cache by 8.5%, 20.2%, and 9.1%, compared to the $1ms$, $10ms$, and $100ms$ base retention times, respectively. The data cache energy savings were 7%, 16%, and 5.7%, respectively (Figure 8a is used to illustrate the cache energy due to the similar trends). As depicted in Figure 8b, on average across the benchmarks, the best retention times improved the performance (execution time) by 7.3%, 17.7%, and 6.5%, respectively.



(a) Total energy (b) Execution time

Fig. 9: Energy and execution time of STTRAM cache normalized to SRAM in a quad-core system

VI. Multicore Evaluation

A. Single Level Cache

Next, we explore the benefits of STTRAM caches in a quad-core system with a single level cache. We assumed that a single multi-threaded mobile application was running with the threads distributed among the four cores. Similar to the single core (Figure 6), as the retention time increased, the miss rates trended toward SRAM. Overall, no retention time below $1ms$ sufficed for any of the considered applications. For instance, for the instruction cache, the $100\mu s$ retention time increased the cache miss rates by 30.3%, on average, whereas $1ms$ achieved similar cache miss rates to SRAM. We observed similar behaviors for the data cache. Thus, we limit the following discussions to only include the $1ms$, $10ms$, and $100ms$ retention times.

Just like the single core system, the multicore STTRAM caches achieved substantial energy savings compared to SRAM. We observed similar trends for both the instruction and data caches, so show results for total cache energy savings. Figure 9a depicts the total cache energy of STTRAM normalized to SRAM in a quad-core system. On average across all the benchmarks, $1ms$, $10ms$, and $100ms$ STTRAM caches reduced the energy by 87.0%, 86.6%, and 85.2%, respectively, compared to SRAM, with energy savings ranging from 82% to up to 87%. The energy savings were consistent across the benchmarks, except for *baidumap*, which was an outlier on the $100ms$ retention time—due to its sensitivity to the increased write access energy—with 51.3% energy savings.

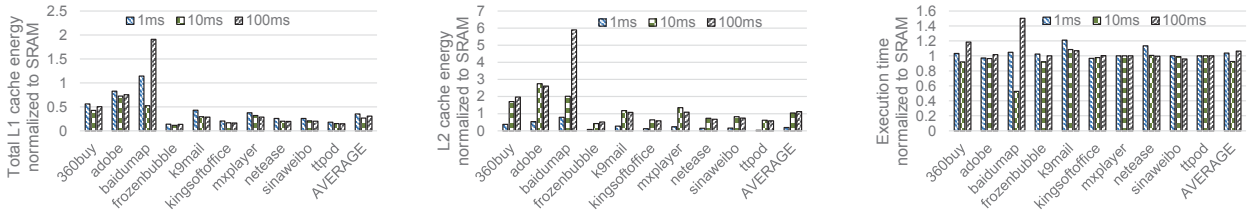
The energy savings was at the expense of some execution time overhead. Figure 9b depicts the execution time of the quad-core system with STTRAM normalized to SRAM. On average across all the benchmarks, the $1ms$ and $10ms$ retention times achieved similar execution times to SRAM. However, the $100ms$ retention time *increased* the execution time by 10.8%, owing to *baidumap*, for which the execution time was substantially higher (3x) than SRAM, due its sensitivity to the increased write access latency.

B. Two-Level Cache

The results were substantially different when an L2 cache was incorporated into the system. Figure 10 depicts the total STTRAM L1 energy (10a), L2 energy (10b), and execution time (10c) normalized to SRAM. When the L2 cache was incorporated, the energy savings in the L1 caches reduced significantly from the system without L2. As shown in Figure 10a, on average, the $1ms$, $10ms$, and $100ms$ L1 STTRAM caches' energy savings were 64.9%, 73.4%, and 69.3%, respectively, compared to SRAM. For the L2 cache, as shown in Figure 10b, on average, the $1ms$ STTRAM L2 cache reduced the energy by 80.7% compared to SRAM. However, the $10ms$ and $100ms$ caches *increased* the average energy by 4.4% and 12.1%, respectively, compared to SRAM. This was due to the increase in write latency of the $10ms$ and $100ms$ caches.

Figure 10c depicts the execution time of the quad-core system featuring L1 and L2 STTRAM caches normalized to SRAM. On average, the $1ms$ and $100ms$ STTRAM caches *increased* the execution time by 3.7% and 6.4%, respectively, while the $10ms$ reduced the execution time by 7.6%. The $10ms$ cache provided a balance between the overhead of expiration misses in the $1ms$ cache and the write latency of the $100ms$ cache. We also note that the STTRAM system with the L2 cache improved the execution time compared to the system without the L2 cache by 8.2%, 12.4%, and 12.1% for the $1ms$, $10ms$, and $100ms$ caches, respectively. These performance improvements were at the expense of increased energy consumption incurred by introducing the L2 cache.

For most applications, the presence of the L2 cache increased the energy compared to the system without the L2 cache. However, a notable difference was that the STTRAM-based systems suffered substantially less energy overhead from the presence of the L2 cache than the SRAM-based systems. Figure 11 illustrates this observation. The figure depicts the total cache energy consumption of the quad-core system with L2 cache normalized to the quad-core system without the L2 cache, for both STTRAM and SRAM. For brevity, the STTRAM results represent the averages of the energy of the $1ms$, $10ms$, and $100ms$ retention times, since the trends were similar for the different retention times. On average across all the benchmarks, for the STTRAM-based system, introducing



(a) Total L1 cache energy (b) L2 cache energy (c) Execution time

Fig. 10: Total STTRAM L1 and L2 cache energy and time normalized to SRAM. We assume homogeneous STTRAM retention times across all cache levels

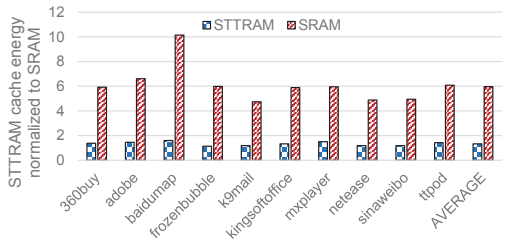


Fig. 11: Total cache energy of quad-core system with L2 normalized to system without L2.

the L2 cache increased the energy by 32.4%. For the SRAM-based system, on the other hand, introducing the L2 cache increased the energy by 6.0x. Even though the L2 cache did not introduce as much energy in the STTRAM-based system as it did in the SRAM-based system, system designers must carefully consider the tradeoffs of including an L2 cache, especially for energy-constrained systems.

VII. Asymmetric STTRAM Cache Design

The results presented so far have shown that STTRAM can provide substantial energy benefits over SRAM for mobile applications. We have also shown that while STTRAM performed well for all the mobile applications, the benefits varied for different applications. This observation is in line with prior research that has shown that different applications may have different cache requirements. Thus, to further improve the energy efficiency of STTRAM caches for mobile applications, we propose *asymmetric retention time cache* as a simple design approach for mobile device processors.

A. Proposed Retention Time Asymmetry

Figure 12 illustrates the proposed asymmetric retention STTRAM cache design. The multicore processor is designed using different retention times in each core such that threads requiring similar retention times can be scheduled to their best core during runtime by the operating system. In the private L1 instruction and data caches, different cores feature different retention times, which are carefully chosen via design-time exploration, to satisfy a variety of applications' execution requirements. Additional asymmetry can also be implemented in lower cache levels. For example, shared caches (e.g., L2, L3) can be designed with different retention times in different banks in a multi-banked design, and cache blocks are opportunistically written to the bank that most closely matches the blocks' lifetimes [26]. In the following subsection, for brevity, we limit our evaluations to the benefits of asymmetric L1 STTRAM cache design.

B. Asymmetric Retention L1 STTRAM Cache

In the asymmetric L1 STTRAM cache, during runtime, threads are scheduled to the core that most closely matches the threads' execution requirements as determined during a profiling period. Unlike the scenario described in Section V-B where different applications are run using different retention times on a single chip, in the asymmetric setup, different threads of the same application—or different applications in

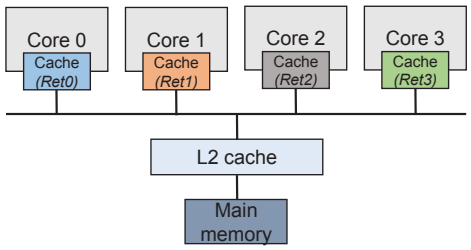


Fig. 12: Illustration of proposed asymmetric retention caches. The L1 caches feature different retention times to satisfy a variety of execution requirements.

a multi-programmed workload—are run on the cores with retention times that most closely satisfy the threads' needs.

Overall, asymmetric retention L1 STTRAM cache achieved energy savings compared to a base homogeneous L1 STTRAM cache without degrading the execution time. Figure 13 compares the asymmetric retention design to a base homogeneous retention time (baseline of one). The base homogeneous retention time is chosen as the single best retention time that achieved the lowest energy on all the cores for each benchmark. For a stringent comparison, we used application-specific homogeneous retention times; thus, the homogeneous configuration changed for different benchmarks. On average, the asymmetric design reduced the energy by 9.6% compared to the base homogeneous retention time, with energy savings as high as 16.0% for *ttpod*. In the worst case, the asymmetric design performed similarly to the homogeneous design for *baidumap*. These results suggest that that the asymmetric design is a viable approach to further improve the energy efficiency of STTRAM caches for mobile applications.

C. Profiling for Retention Time

To fully leverage the benefits of an asymmetric L1 STTRAM cache design, one important challenge is how to determine the best retention time during runtime. Solving this challenge is outside the scope of this paper; however, we suggest a few potential solutions, some of which have been explored by related work.

The first potential technique for determining the best retention time is a sampling technique as described in prior work [16]. The executing application is run on each core for a brief profiling interval (e.g., 10M instructions) after which the energy consumption is calculated. The rest of the application is then run on the core whose cache consumed the least energy. Sampling can be used without introducing substantial

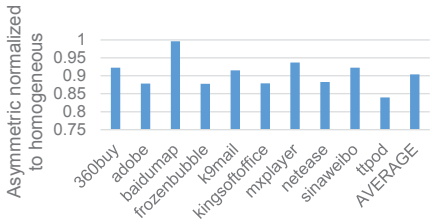


Fig. 13: Energy consumption of asymmetric retention times normalized to the best homogeneous retention time.

overheads if only a few retention times (e.g., four) must be sampled. However, in larger-scale systems, other techniques must be explored.

Unlike traditional cache configurations (cache size, line size, and associativity), the best retention time cannot be directly predicted using statistics obtained from performance counters (e.g., cache miss rates). Thus, an alternative solution is to use design-time machine learning algorithms to model the correlation of retention time requirements to easily obtainable statistics. These models can then be used during runtime to directly predict the best retention time, given the execution statistics obtained during a profiling interval. We plan to evaluate these solutions for mobile applications in future work.

VIII. Concluding Remarks

Spin-Transfer Torque RAMs (STTRAMs) have the potential to replace SRAMs in implementing on-chip caches in emerging mobile devices. Reduced retention STTRAMs, in addition to area benefits, offer further energy savings potential, if the retention times are carefully chosen to satisfy the cache block lifetimes of executing applications. In this paper, we explore and evaluate the benefits of STTRAM caches for mobile applications. We characterize mobile applications from the perspective of reduced retention STTRAM caches, and show that STTRAMs provide much energy benefits for mobile applications without introducing substantial latency overheads. We also explored the benefits of an asymmetric retention time design to provide further energy savings compared to a homogeneous retention time design. The analysis herein is performed without any orthogonal techniques for making STTRAMs more efficient (e.g., limiting the number of writes, hybrid caches), thereby minimizing runtime implementation overheads. However, our future work includes a detailed study of the synergy of prior techniques for energy efficient cache hierarchies for mobile applications and evaluating different STTRAM cell models for emerging mobile devices.

References

[1] "Cortex-a76," https://developer.arm.com/ip-products/processors/cortex-a/cortex-a76, accessed: April 2019.

[2] S. Mittal, "A survey of architectural techniques for improving cache power efficiency," *Sustainable Computing: Informatics and Systems*, vol. 4, pp. 33–43, 2014.

[3] D. Apalkov, A. Khvalkovskiy, S. Watts, V. Nikitin, X. Tang, D. Lottis, K. Moon, X. Luo, E. Chen, A. Ong *et al.*, "Spin-transfer torque magnetic random access memory (stt-mram)," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 9, no. 2, p. 13, 2013.

[4] M. Kang, S. K. Gonugondla, M.-S. Keel, and N. R. Shanbhag, "An energy-efficient memory-based high-throughput vlsi architecture for convolutional networks," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2015, pp. 1037–1041.

[5] C. W. Smullen, V. Mohan, A. Nigam, S. Gurumurthi, and M. R. Stan, "Relaxing non-volatility for fast and energy-efficient stt-ram caches," in *High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on*. IEEE, 2011, pp. 50–61.

[6] A. Jog, A. K. Mishra, C. Xu, Y. Xie, V. Narayanan, R. Iyer, and C. R. Das, "Cache revive: architecting volatile stt-ram caches for enhanced performance in cmps," in *Proceedings of the 49th Annual Design Automation Conference*. ACM, 2012, pp. 243–252.

[7] S. Kaxiras, Z. Hu, and M. Martonosi, "Cache decay: exploiting generational behavior to reduce cache leakage power," *ACM SIGARCH Computer Architecture News*, vol. 29, no. 2, pp. 240–251, 2001.

[8] 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," *IEEE Journal of Solid-State Circuits*, vol. 48, no. 2, pp. 598–610, 2013.

[9] H. Noguchi, K. Ikegami, N. Shimomura, T. Tetsufumi, J. Ito, and S. Fujita, "Highly reliable and low-power nonvolatile cache memory with advanced perpendicular stt-mram for high-performance cpu," in *2014 Symposium on VLSI Circuits Digest of Technical Papers*, June 2014, pp. 1–2.

[10] Z. Sun, X. Bi, H. H. Li, W.-F. Wong, Z.-L. Ong, X. Zhu, and W. Wu, "Multi retention level stt-ram cache designs with a dynamic refresh scheme," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2011, pp. 329–338.

[11] J. Wang, Y. Tim, W.-F. Wong, Z.-L. Ong, Z. Sun, and H. Li, "A coherent hybrid sram and stt-ram l1 cache architecture for shared memory multicores," in *Design Automation Conference (ASP-DAC), 2014 19th Asia and South Pacific*. IEEE, 2014, pp. 610–615.

[12] J. Ahn, S. Yoo, and K. Choi, "Dasca: Dead write prediction assisted stt-ram cache architecture," in *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2014, pp. 25–36.

[13] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "Energy reduction for stt-ram using early write termination," in *Proceedings of the 2009 International Conference on Computer-Aided Design*. ACM, 2009, pp. 264–268.

[14] E. Reed, A. R. Alameldeen, H. Naeimi, and P. Stolt, "Probabilistic replacement strategies for improving the lifetimes of nvm-based caches," in *Proceedings of the International Symposium on Memory Systems*. ACM, 2017, pp. 166–176.

[15] K. Qiu, Y. Zhu, Y. Xu, Q. Huo, and C. J. Xue, "Brloop: Constructing balanced retimed loop to architect stt-ram-based hybrid cache for vliw processors," *Microelectronics Journal*, vol. 83, pp. 137 – 146, 2019.

[16] K. Kuan and T. Adegbija, "Lars: Logically adaptable retention time sttram cache for embedded systems," in *Design, Automation & Test in Europe Conference & Exhibition, 2018. DATE'18.* IEEE Computer Society, 2018.

[17] "Spec cpu2006," http://www.spec.org/cpu2006, accessed: March 2019.

[18] K. Yan, L. Peng, M. Chen, and X. Fu, "Exploring energy-efficient cache design in emerging mobile platforms," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 22, no. 4, p. 58, 2017.

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

[20] A. Gutierrez, R. G. Dreslinski, T. F. Wenisch, T. Mudge, A. Saidi, C. Emmons, and N. Paver, "Full-system analysis and characterization of interactive smartphone applications," in *2011 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2011, pp. 81–90.

[21] W. Kang, L. Zhang, W. Zhao, J. Klein, Y. Zhang, D. Ravelosona, and C. Chappert, "Yield and reliability improvement techniques for emerging nonvolatile stt-mram," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 5, no. 1, pp. 28–39, March 2015.

[22] S. Senni, L. Torres, G. Sassatelli, and A. Gamatie, "Non-volatile processor based on mram for ultra-low-power iot devices," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 13, no. 2, p. 17, 2017.

[23] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti *et al.*, "The gem5 simulator," *Computer Architecture News*, vol. 40, no. 2, p. 1, 2012.

[24] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, "Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 31, no. 7, pp. 994–1007, 2012.

[25] Y. Huang, Z. Zha, M. Chen, and L. Zhang, "Moby: A mobile benchmark suite for architectural simulators," in *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2014, pp. 45–54.

[26] K. Kuan and T. Adegbija, "Halls: An energy-efficient highly adaptable last level stt-ram cache for multicore systems," *IEEE Transactions on Computers*, 2019.