STAxCache: An Approximate, Energy Efficient STT-MRAM Cache

Ashish Ranjan[1], Swagath Venkataramani[1], Zoha Pajouhi[1], Rangharajan Venkatesan[2],
Kaushik Roy[1] and Anand Raghunathan[1]
[1]School of Electrical and Computer Engineering, Purdue University
[2]NVIDIA Research, Santa Clara
[1]{aranjan,venkata0,zpajouhi,kaushik,raghunathan}@purdue.edu
[2]rangharajanv@nvidia.com

***Abstract*—STT-MRAM has attracted great interest for use as on-chip memory due to its high density, near-zero leakage and high endurance. However, its overall energy efficiency is limited by the energy requirements of spin-transfer torque switching during writes and reliable single-ended sensing during reads. Leveraging the ability of many applications to produce acceptable outputs under approximations to computations and data, we propose the use of approximate storage to improve the energy efficiency of STT-MRAM based caches. Towards this end, we explore a combination of different approximation techniques at the circuit and architecture levels that yield significant energy benefits for small probabilities of errors in reads, writes, and retention. A key challenge arises when introducing approximate storage into a cache – data that can tolerate different levels of approximation (or not at all) may be dynamically loaded into a cache line at different times. In addition, it is necessary to manage the approximations so as to obtain a desirable energy-quality tradeoff at the application level. We propose STAxCache (Spintronic Approximate Cache), an STT-MRAM based approximate L2 cache architecture that retains the full flexibility of a conventional cache, while allowing for different levels of approximation to different parts of a program's memory address space. We introduce a simple interface that allows the programmer to specify the quality requirements for different data structures, and instructions in the ISA to expose this information to STAxCache. We utilize a device-to-architecture simulation framework to evaluate STAxCache and achieve 1.44× improvement in L2 cache energy for negligible ($< 0.5\%$) loss in application-level quality across a suite of 8 benchmarks.**

***Index Terms*—Spintronics, STT-MRAM, Approximate Caches**

I. Introduction

The growth in amounts of data processed by computing platforms from mobile devices to data centers, together with the need to bridge the increasing processor-memory gap (feed increasing numbers of cores) have led to an incessant demand for more on-chip memory. Consequently, a large and growing fraction of chip area and energy consumption is expended in caches. CMOS memories face challenges with technology scaling due to increased leakage and process variations. These challenges, coupled with an increased demand for on-chip memory, have led to an active exploration of alternative on-chip memory technologies.

Spin transfer torque magnetic RAM (STT-MRAM) has gained significant interest in recent years as a potential post-CMOS memory technology [1], [2]. STT-MRAMs offer high density and near-zero leakage, making them promising candidates for on-chip memories. However, their overall energy efficiency is still limited by the energy required for spin-transfer torque (STT) switching in writes and reliable single-ended sensing during reads. Several promising research efforts have been devoted to improving the energy efficiency of STT-MRAM at the device, circuit and architecture levels [2]–[5]. In this work, we propose the use of a different approach – approximate storage – to achieve energy efficiency for STT-MRAM based caches.

Several emerging applications that have fueled the demand for larger on-chip memories (multimedia, recognition, data mining, search, and machine learning, among others) also exhibit intrinsic resilience to errors, *i.e.*, the ability to produce results of acceptable quality even with approximations to their computations or data [6]. Approximate computing exploits this characteristic of applications to derive energy or performance benefits using techniques at the software, architecture, and circuit levels [7]. Most previous work in approximate computing focuses on processing or logic circuits. Previous efforts on approximate storage [8]–[14] can be classified based on the level of the memory hierarchy that they target. Some focus on secondary storage and main memory [8], [9] using techniques that are complementary to our work. Others focus on application-specific memory designs [10], [11]. A few efforts [12]–[14] explore approximate cache architecture with CMOS memories, using techniques such as skipping cache loads on misses. Complementary to these efforts, we propose STAxCache, an approximate STT-MRAM based L2 cache that exploits the specific error-energy tradeoffs that manifest in spintronic memories to improve energy efficiency.

A key challenge in approximate computing is how to manage the approximations so as to obtain the most favorable energy *vs.* application quality trade-off. To this end, we explore a combination of various circuit- and architecture-level techniques that yield significant energy benefits for small probabilities of read, write and retention errors. We explore: (i) *Approximation through partial reads/writes*, where reads (or writes) are to selected least significant bits are ignored, (ii) *Approximation through lower read currents*, wherein a lower read current is used for sensing, thereby trading off decision failures for read energy benefits, (iii) *Approximation through skipped writes*, wherein writes to a cache block are skipped at run-time if they are similar to its current contents, (iv) *Approximations through lower write duration*, wherein writes are performed for a smaller duration, resulting in an increased probability of write failures, and (v) *Approximations through skipped refreshes*, wherein refresh operations to the low retention blocks are selectively skipped.

STAxCache utilizes a heterogeneous cache organization that is composed of low retention and high retention cache ways, each of which is in turn designed using a quality-configurable data array. The quality-configurable data array employs a

All authors contributed to the work while employed at Purdue University. Author affiliations listed are at the time of paper submission. Swagath Venkataramani is currently at IBM T. J. Watson research center. Zoha Pajouhi is currently at Intel Corporation.

This work was supported in part by STAR-net, a Semiconductor Research Corporation program sponsored by MARCO and DARPA, and in part by the National Science Foundation under grant 1423290.

978-3-9815370-8-6/17/$31.00 ©2017 IEEE

combination of the above mechanisms to dynamically perform read (or write) operations at various accuracy levels. We propose suitable enhancements in the cache replacement policy to exploit the heterogeneous cache ways for achieving lower write energy. To provide control over the errors introduced during approximations, the STAxCache architecture also consists of: (i) a quality table that captures the quality requirements for the regions of address space and also tracks the number of refreshes skipped for each region, (ii) a quality-aware cache controller that enforces the specified quality constraints. Finally, we introduce an application programming interface that enables programmers to specify quality requirements with minimal effort, and ISA extensions that allow these requirements to be conveyed to the underlying hardware.

In summary, the key contributions of this work are:

- We propose an approximate L2 STT-MRAM based cache to improve read and write energy-efficiency by exploiting the error resilient nature of applications.
- We explore a combination of circuit and architectural techniques and present a cache architecture that provides fine-grained control over the errors introduced during approximations.
- We introduce instructions in the ISA and propose mechanisms that allow programmers to specify the desired quality requirements at the data structure level.
- We develop a device-to-architecture simulation framework to evaluate STAxCache. Our experiments on a suite of 8 machine learning benchmarks demonstrate 1.44× improvement in cache energy over an iso-capacity STT-MRAM based L2 cache for $< 0.5\%$ loss in output quality.

The rest of the paper is organized as follows. Section II provides preliminaries on STT-MRAM. Section III describes the STAxCache architecture and the ISA enhancements required to expose it to software. Section IV details the experimental methodology, and the results are presented in Section V. Section VI provides an overview of related previous work, and Section VII concludes the paper.

II. STT-MRAM: Preliminaries

Fig. 1 shows the standard STT-MRAM bit-cell that comprises of an access transistor and a magnetic tunnel junction (MTJ). The MTJ stack is composed of a ferromagnetic pinned layer (which has a fixed magnetic orientation) and a free layer (whose orientation can be switched), separated by a tunneling oxide barrier. The logic state stored in the bit-cell depends on the relative orientation between the free layer and the pinned layer (we assume parallel orientation represents "0" and anti-parallel orientation represents "1"). A read operation involves activating the wordline (WL) and applying a bias voltage (V_{read}) between the bitline (BL) and the source line (SL). The resulting read current through the bit-cell (I_{read} in Fig. 1) is compared against a global reference current to determine the logic state stored. A write operation is performed by passing a current greater than the MTJ's critical switching current (I_c) for a minimum switching duration. The current direction (I_{write0},I_{write1} in Fig. 1) differs based on the logic value to be written into the bit-cell. Writes in STT-MRAM are stochastic in nature, and the magnitude and duration of the write current determines the write failure rate. Besides write failures, STT-MRAMs may also suffer from read decision failures, where the data stored in a bit-cell is incorrectly read due to process variations, and read disturb failures where a read operation accidentally writes into the cell. Another key design metric of STT-MRAM is the retention time, which is the duration for which the data stored in an idle bit-cell is retained. Lowering the retention time makes switching the MTJ easier, since it decreases I_c. However, it also makes the bit-cell more prone to retention failures due to thermal disturbances.

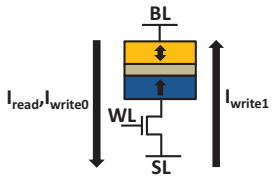

Fig. 1: STT-MRAM bit-cell

III. STAxCache Architecture

Fig. 2 provides an overview of the STAxCache architecture. STAxCache consists of a quality-configurable data array that is capable of performing read or write operations at various levels of accuracy and energy depending on the application's requirements, and a tag array that is not subject to approximations [1]. The data array is further composed of heterogeneous cache ways with varying retention levels (ways with lower retention time offer more energy-efficient writes). To allow control over which data is approximated and by how much, STAxCache also includes: (i) A *Quality table* that captures the data structure-level quality specifications of the application and (ii) A *Quality-aware cache controller* that regulates the quality of each cache access based on the entries in the quality table. The following subsections provide a detailed description of STAxCache and the different approximation techniques employed for energy efficiency.

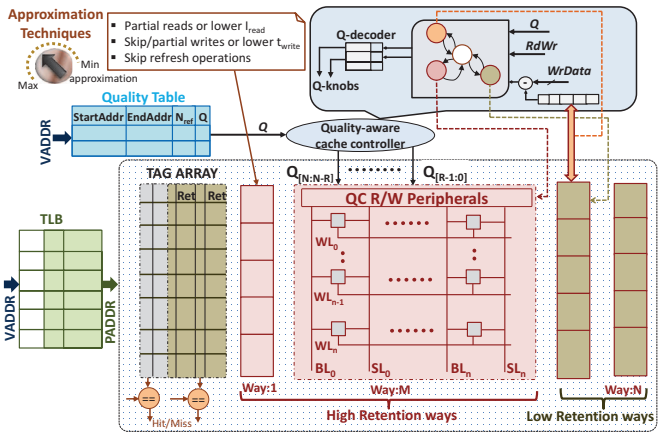

Fig. 2: STAxCache organization

A. Quality Table

STAxCache requires a mechanism to capture quality constraints in a manner that can be related to cache block-level reads (or writes). To this end, we introduce a quality table, as shown in Fig. 2. Each entry in the quality table contains a memory address range and the desired quality for accesses to addresses within the range, *e.g.*, permissible magnitude of error that may be incurred when a location in the specified range is accessed. On each cache access, we compare the cache block address with the address ranges present in the table. If there is a match, we utilize the corresponding quality for reading (or writing) the block. We modulate approximation techniques that are discussed in Section III-B to achieve the desired read/write quality. In order to avoid significant impact

[1]We focus on data array since it consumes a dominant part of cache energy. Furthermore, it is difficult to bound the impact of errors in the tag array.

on cache performance, we overlap the quality table look up and the subsequent quality decoding operation with the translation lookaside buffer (TLB) access and the following tag comparison operation. Our experiments indicate that the overheads of the additional quality table access on the cache performance is negligible ($< 0.3\%$).

B. Approximation Techniques

STAxCache employs a diverse range of approximation techniques in order to achieve fine grained control over the quality of reads and writes, as well as a favorable energy-quality tradeoff. We next describe these techniques in detail.

1) Read Approximations: To obtain energy efficient reads, we explore the following techniques:

Approximations through partial reads. In this technique, we ignore a few least significant bits (LSBs) while reading each word in a cache block. Unlike SRAM, STT-MRAM does not suffer from the half-select problem; therefore, bit lines and source lines corresponding to the LSBs may be gated to achieve energy savings. The LSBs are simply set to 0 in the value returned from the cache. We determine a worst-case error bound based on the significance of the ignored bits and ensure that it is permissible for the specified block-level constraint.

Approximations through lower read current. STT-MRAM bit-cells naturally provide energy *vs.* quality trade-offs when a current smaller than the nominal current (I_{read}) is passed through the bit-cell during reads [2]. Leveraging this attribute, we read some of the bits in a cache block with a lower I_{read}, leading to increased probability of read failures for the corresponding bits. This is performed in a bit-significance driven manner by dividing each word in the cache block into bit groups, and associating a lower read failure probability for the more significant bit groups. This approach enables a fine grained control over the errors introduced during reads.

2) Write Approximations: We next describe the approximate write mechanisms that are used in STAxCache.

Approximations through skipped or partial writes. In this scheme, we compare the difference in magnitude of the data words in the incoming write block with the previously stored values, and determine whether the write to the cache location can be skipped. If it does not violate the cache block-level quality constraint, we choose to skip the write and retain the stale value, thereby saving considerable energy. However, if the write operation cannot be skipped, we adopt a similar approach as approximate partial reads wherein few of the LSBs of a word are not written (the number of ignored bits is determined by the quality constraint).

Approximations through lower write duration. Writes in STT-MRAM bit-cells inherently lend themselves to approximations, wherein energy benefits can be obtained by lowering the write duration at the cost of write failures [2]. In doing so, we adopt a bit-significance driven approach similar to the approach for reads discussed earlier. Specifically, we introduce a higher write failure probability to the least significant bit group, and progressively associate decreasing failure probabilities for the higher significance groups within each word of the block, by suitably modulating the write duration.

3) Retention Approximations: Lowering the retention time in an STT-MRAM bit-cell reduces the write energy at the cost of increased retention failures. Since most applications contain a mix of resilient and sensitive data, simply reducing the retention time for the entire cache is not acceptable. Therefore, we design a hybrid data array that comprises of both high retention and low retention ways, as shown in Fig. 2. While we found only two levels of retention to be sufficient in practice (and desirable due to the lower design and manufacturing complexity), the concept extends to a larger degree of heterogeneity.

Refresh requirements. Cache blocks stored in the low retention ways are subject to a significant increase in the probability of errors beyond the retention time (T_{Ret}) owing to the exponential nature of retention failures [15]. Simply allowing retention errors is not always acceptable. While we preferentially allocate cache blocks with lower quality requirements to the low retention ways, data with very tight quality constraints (or data that cannot be approximated) may also be allocated to the low retention ways to ensure high cache utilization and low misses. Moreover, the lifetimes of cache blocks in low retention ways may vary considerably within and across applications. This imposes the need for periodic refresh operations, particularly when the lifetimes of the blocks are closer to (or exceed) T_{Ret}. Refreshing all the valid cache blocks in the low retention ways after each T_{ret} would ensure no retention errors, but leads to a significant number of refreshes. STAxCache addresses this issue by skipping refreshes for cache blocks that have been written due to a store instruction in the recent past. To enable this, we extend the tag array with only one retention bit per cache block to track the blocks stored in the low retention ways that have been written to or "self-refreshed" since the last refresh operation. Let us consider an example to demonstrate the proposed refresh mechanism. Fig. 3(a) shows a 2-way set associative cache that consists of a high retention way and a low retention way. For this example, we need one retention bit per block as shown in the figure. Suppose at T = 0, two cache blocks ($B0$ and $B1$) are inserted in the low retention way. At T = T_{Ret}/2, the retention bits associated with all the cache blocks are checked. In case the bit is set to logic '0', we update it to logic '1', indicating that the block is due for a refresh operation in the next update cycle, *i.e.*, T = T_{Ret}, as shown in Fig. 3(a) for $B0$ and $B1$. Next, suppose a write operation is performed on $B0$ between T = T_{Ret}/2 and T = T_{Ret}. In this case, the retention bit is reset to logic '0'. Hence, $B0$ no longer requires a refresh operation in the following update cycle. On the other hand, if the retention bit is set to '1', we perform the refresh operation for $B1$, as shown in Fig. 3(a) at T = T_{Ret}, and reset the bit to '0'.

Approximations through skipped refreshes. Despite exploiting the self-refreshes to lower the refresh overheads, the energy consumed by the refresh operations still constitute a significant fraction of the total cache energy. In order to minimize the refresh energy further, we propose to skip refreshes to the blocks that are amenable to approximations. However, it is critical to have control over the retention errors introduced in the stored blocks as a result of the skipped refreshes. Towards this end, we extend the quality table with an additional counter (N_{Ref}) for each entry that tracks the number of refreshes skipped for a given address range on each update cycle. Fig. 3(b) illustrates the proposed concept through a timeline for the cache organization discussed earlier. As shown in the figure, at T = T_{Ret} and T = $2T_{Ret}$, the addresses corresponding to $B0$ and $B1$ which are due for refresh (retention bits are set to '1'), are compared against the address ranges in the quality table. In case of a matching entry, the corresponding N_{Ref} is compared to a refresh threshold (N_{Th}) that is determined from the corresponding block-level quality constraint. If N_{Ref} exceeds (or equals) N_{Th}, we perform refreshes to each of the low retention blocks contained in the address range ($B1$ at T =

$2T_{Ret}$), else we skip those refreshes ($B1$ at T = T_{Ret} and T = $3T_{Ret}/2$) and increment N_{Ref} by 1. If a matching range does not exist in the table, the refresh operations are performed, as shown in the figure for $B0$ at T = T_{Ret} and T = $2T_{Ret}$.

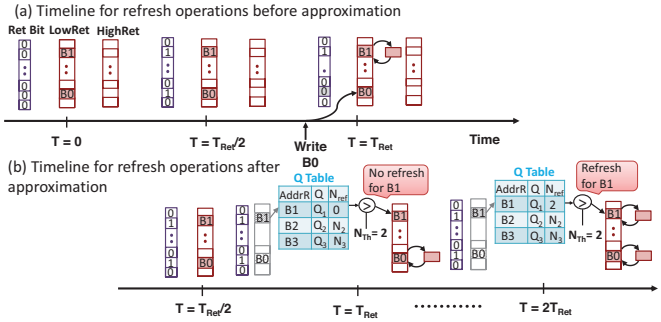

Fig. 3: Timeline showing refresh operations

C. Quality-configurable data array

We design a quality configurable data array that supports the read and write approximation mechanisms discussed above. Specifically, we enhance the standard STT-MRAM read and write peripheral circuits for each column so as to allow partial reads (or writes) of bits and modulation of read current (or write duration). However, the core data array remains unchanged. Using the enhanced peripherals, we appropriately choose a subset of the mechanisms at runtime depending on the target quality of reads (or writes).

D. Quality-aware cache controller

In this section, we describe the quality-aware cache controller that utilizes the quality-configurable array to achieve read and write energy efficiency while preserving the quality constraints obtained from the quality table. Fig. 2 provides an overview of the proposed quality-aware cache controller. The cache controller involves an additional control input (Q) that represents the desired quality for each cache access. A quality decoder takes the quality signal (Q) and the read/write control signal ($RdWr$) as inputs and generates values of quality knobs ($Q[N-1:N-R], ..., Q[R-1:0]$) for each group of R bits of the words in the block. Since the energy *vs.* quality trade-offs widely vary across the different schemes, a systematic approach is required to obtain these knobs such that the energy savings are maximized for a given quality bound. In the following paragraphs, we describe how the quality knobs are obtained.

Read quality modulation. Fig. 4 summarizes the approach used *at design time* to obtain the read quality knobs. Consider the surface plot of the block-level quality (Q_{total}) for the different quality (and energy) configurations of the two schemes – partial read and reducing the read current (Q_1/E_1, Q_2/E_2), as shown in the figure. Note that, there exist several configurations that achieve the same output quality (Q) although with significantly different energies, as shown by the Q plane. Therefore, we adopt a gradient descent approach on the Q plane to achieve the configuration that maximizes the energy benefits for a given quality. In each iteration, we rank the read schemes based on the additional energy savings obtained by approximating a group of R bits for each word and the expected error introduced in the process. We estimate the expected error using an input distribution of the resilient data elements observed during reads. Next, we choose the scheme for the bit group with the highest ratio of energy benefits and expected error, and proceed to the next iteration until the block-level quality constraint is not violated. The above steps are then repeated for each quality level supported by the cache.

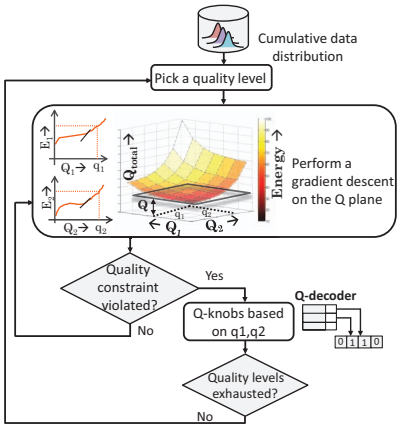

Fig. 4: Read quality modulation

Write quality modulation. Fig. 5 illustrates how the quality knob is obtained for writes in the proposed controller. In this case, we utilize a hybrid design time / runtime scheme. We first determine if the block-level quality constraint is satisfied when a write to the block is skipped altogether (*i.e.*, the stale value is retained). If the constraint is not violated, we skip the write operation; else, we utilize pre-computed knobs obtained via a similar approach as the one discussed for reads.

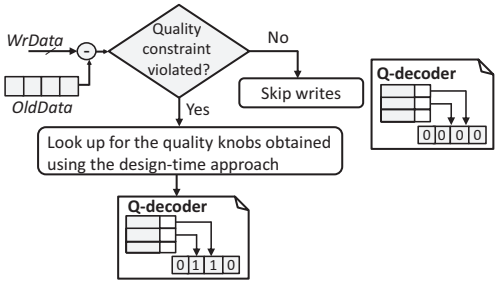

Fig. 5: Write quality modulation

E. Cache insertion and replacement policy

We introduce suitable enhancements to the cache insertion and replacement policies to exploit the heterogeneous cache ways for energy efficiency. In this work, we employ an insertion policy that is driven by the write intensity of the incoming block. In order to identify the write intensive blocks, we utilize a software-based approach similar to [16]. The write intensive blocks thus identified are inserted in the low retention ways, whereas the other blocks can be inserted either in a low retention or high retention way depending on the block being evicted. We therefore introduce a modified least recently used (LRU) policy that evicts the LRU block within the low retention ways for a write intensive incoming block and uses the regular LRU policy for the remaining blocks.

F. ISA extension

In order to expose the intrinsic resilience of applications to the STAxCache architecture, we introduce a new instruction in the ISA that is used to identify resilient regions in the application's memory space, and also specify the desired quality bounds for them. Using this instruction, we update the proposed quality table, which is in turn used to regulate the accuracy of cache accesses.

Instruction format. Equation 1 shows the instruction format of the proposed instruction. As shown in the example, the

instruction specifies the start and the end addresses associated with the resilient memory region (provided through registers R_{StAddr} and $R_{EndAddr}$) along with the maximum error bound that can be tolerated during read, write or refresh operations.

Format: **Opcode** Reg1, Reg2, Reg3
Example: **ldQTable** R_{StAddr}, $R_{EndAddr}$, $R_{Quality}$ (1)

G. Software support for STAxCache

We propose a programmer interface that can be used to specify resilient data within a program. Specifically, we introduce new functions that capture the permissible error bound that can be tolerated in various cache operations for a given address range. Fig. 6 shows an example code snippet that uses one such function to specify the data structure-level quality requirements in a k-means clustering application. In this example, an input array (*points*) is identified for approximations using the proposed function *set_axLevel*. The function *set_axLevel* takes the start and the end address corresponding to the *points* array and the tolerable error magnitude as inputs. This enables a simple interface for the programmer to specify quality targets for resilient data structures.

```
int main() {
   int* points = (int*) malloc (NUM_DATA * INT_SIZE);
   set_axLevel (points, (points + NUM_DATA), Q);
   read_points (points, NUM_DATA);
   int* means = (int*) malloc (NUM_MEANS * INT_SIZE);
    .....
    .....
   while (converges) {
      find_clusters (points, means, clusters);
      compute_means (points, means, clusters);
   }
}
void set_axLevel(int startAddr,  int endAddr, int err_mag) {
    __asm__ ( "movl  %1, %%eax ;
                movl  %2,  %%ebx ;
                movl  %3,  %%ecx;
                ldQTable %%eax, %%ebx, %%ecx"
                : /* no outputs */
                : "a" (startAddr) , "b" (endAddr), "c“ (err_mag)
}            );
```

Fig. 6: K-means clustering application for STAxCache

IV. Experimental Methodology

In this section, we discuss the device-to-architecture simulation framework and the benchmarks used in our evaluation.

MTJ Device/Array-level modeling. The STT-MRAM bit-cell was characterized in SPICE using a 32nm CMOS transistor model and a SPICE-compatible PMA MTJ device model based on self-consistent solution of Landau-Lifshitz-Gilbert (LLG) magnetization dynamics and Non-Equilibrium-Green's Function (NEGF) electron transport [17] with device parameters from [18]. To estimate the array-level latency and energy of the L2 data array, we use the bit-cell characteristics as technology parameters in a modified version of CACTI [19] that is designed for STT-MRAMs. In our analysis, we also include the circuit-level overheads for the enhanced peripherals.

System-level evaluation. We model the STAxCache architecture in the architectural simulator gem5 [20] and use it to evaluate the energy benefits and the impact on application-level quality. Table I shows the processor configuration used in our evaluation. In our experiments, we consider 4 low retention ways (with an energy barrier of 16 K_BT) for the L2 cache. We use the cache access traces from the simulator along with the array-level energies obtained from the modified CACTI tool to estimate the L2 cache energy. We also account for the overheads in energy and latency associated with the proposed quality table hardware in our experiments.

TABLE I: System configuration

Processor Core	x86, out-of-order processor, 2 GHz
L1 I/D-cache	32KB/64KB, 2 way-set associative, 64B line size
L2 unified cache	8MB shared, 8 way-set associative, 64B line size
Cache latency	L1: 2-cycle, L2 read: 10-cycle, L2 write: 15-cycle

Benchmark applications. Table II lists the applications and datasets that were used to evaluate STAxCache. The quality metric used to evaluate the output quality is also provided.

TABLE II: Application benchmarks

Application	Algorithm	Dataset	Quality metric
Digit Recognition (SVM)	Support vector machines	MNIST	Classification accuracy (fraction of inputs correctly classified)
Text Classification (TEXT)	Support vector machines	REUTERS	
Digit Classification (CNN)	Convolutional neural networks	MNIST	
Character Recognition (OCR)	K-nearest neighbors	OCR digits	
Eye Detection (GLVQ)	Generalized learning vector quantization	YUV faces	
Protein Structure Classification (MLP)	Multi-layer perceptron	Protein	
Image Segmentation (IMGSEG)	K-means clustering	Berkley dataset	Mean distance between clustered points and centroids
Optical Character Clustering (DIGITS)	K-means clustering	OCR digits	

V. Experimental Results

This section presents the results of various experiments that quantify the benefits obtained using STAxCache.

A. Energy benefits of STAxCache

Fig. 7 shows the normalized L2 cache energy obtained using the proposed architecture at different application-level output quality targets. The L2 cache energy is normalized to an accurate cache with low retention ways [2]. The energy benefits range from 1.15× to 1.84× for a negligible loss (< 0.5%) in application-level quality over all benchmarks. For a more relaxed quality requirement (< 3.5% degradation in application output quality), the benefits further extend to 1.5×-2.32×. On average, STAxCache achieves 1.44× and 1.93× improvement in energy at the two quality levels, respectively.

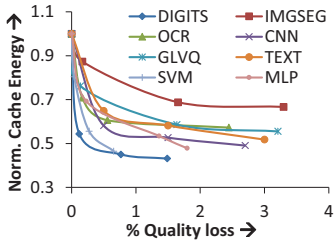

Fig. 7: Improvement in energy using STAxCache

We next present a breakdown of the different energy components, *viz.* read, write, refresh and leakage, that contribute to the overall cache energy. Fig. 8 shows the breakdown for all benchmarks at the two output quality constraints (< 0.5% and 3.5% loss in quality). For the baseline, across all benchmarks, we observe that the read, write, and refresh energies respectively constitute 26%, 61% and 10% of total cache energy, with refresh overheads being as high as 48% of the total energy in some cases. On average, we achieve 1.03× and 1.56×, improvement in read and write energy, for < 0.5% loss in output quality. Note that, for a subset of benchmarks (DIGITS, IMGSEG, OCR, and GLVQ), the read energy increases beyond the baseline. This is due to the write skipping approximation technique, that involves an additional read operation. In these

[2]We focus on an STT-MRAM cache designed with low retention ways since it consumes lower energy than a standard STT-MRAM cache.

benchmarks, such read operations dominate over the original reads, thereby increasing the total read energy, but enabling an even higher savings in write energy. For a relaxed quality bound ($< 3.5\%$), we obtain greater benefits in read and write energy, $1.29\times$ and $2.22\times$, respectively. For benchmarks with significant refresh operations (CNN, IMGSEG), we achieve $1.68\times$ and $2.02\times$ improvement in refresh energy for the tight and relaxed bounds respectively, illustrating the effectiveness of the proposed design in mitigating the refresh overheads through approximations.

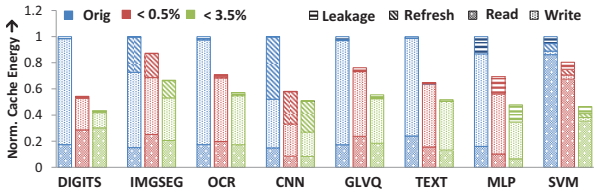

Fig. 8: Energy breakdown for STAxCache

B. Impact on system performance

Fig. 9 compares the IPC (instructions per cycle) for STAxCache at both output quality constraints to the baseline design having no approximations. On average, STAxCache degrades cache performance by 1.9% and 1.7% over the baseline design for the two quality levels. This degradation in performance is mainly due to two factors: (i) the additional latency for writes that are not skipped (since we perform a read to check whether the write can be skipped), and (ii) the overheads of accessing the quality table during each cache access and update cycle.

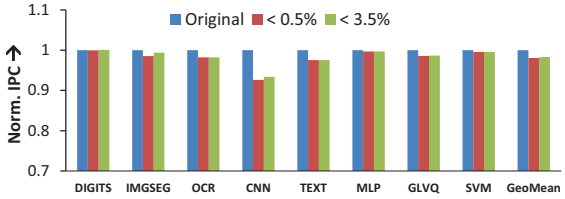

Fig. 9: Performance trend with STAxCache

VI. Related Work

In this section we briefly summarize prior efforts in two related areas – energy efficient STT-MRAMs and approximate storage and highlight the distinguishing features of our work.

Energy efficient STT-MRAMs. Many research efforts have proposed architectural optimizations to address the write-inefficiency of STT-MRAMs. [4] explored a hybrid cache that stores the write intensive blocks in the CMOS region and the remaining blocks in STT-MRAM. [3] proposed eliminating redundant writes to an STT-MRAM cache by comparing with the previously stored value. [5] explored a volatile STT-MRAM cache to achieve write energy benefits while avoiding retention errors through suitable refresh schemes. All of these efforts focused on achieving write energy efficiency while maintaining full accuracy for writes. In contrast, our work leverages application resilience to achieve energy benefits in STT-MRAM at the cost of read, write and retention errors.

Approximate storage. Prior works have explored approximate storage in the context of off-chip and on-chip memories. [8] proposed refresh rate modulation for DRAMs to lower the refresh energy at the cost of retention errors, while [9] proposed lowering the number of write pulses in PCMs, trading-off write errors for energy. In the context of CMOS on-chip memories, [12] proposed supply voltage modulation per cache way to obtain leakage benefits at the cost of failures. [10] explored scaling the supply voltage in an application-specific hybrid SRAM array to achieve energy benefits at the cost of errors. [13] explored a complementary approach of storing similar values as a single cache block to reduce the overall cache energy, whereas [14] proposed skipping cache loads on a miss to lower the miss penalty. In the context of STT-MRAM on-chip memories, [11] utilized bit-cell failure mechanisms in a scratchpad to achieve energy benefits. Our work differs from these research efforts in several aspects. First, unlike most of the efforts [8], [9], [11] that involve software directly managing approximations to these memories through load/store instructions or type qualifiers, our work explores approximate storage in an STT-MRAM cache requiring the cache hardware to dynamically regulate the quality of memory accesses. This involves fundamentally different trade-offs because the same cache region often requires different data storage accuracies at different times. Second, in contrast to [12] that reduces the cache capacity visible to both accurate and resilient data by enforcing a way-based quality, we perform quality regulation per cache line without limiting associativity. Finally, we exploit energy-error tradoffs that are unique to STT-RAMs.

VII. Conclusion

STT-MRAM offers great promise as a potential post-CMOS memory technology. However, its read and write operations are energy-inefficient due to its fundamentally different storage and switching mechanism. In this work, we proposed a cache architecture that exploits the error resilience of applications to achieve energy efficiency. We utilized a combination of different approximation techniques to achieve fine grained control over quality while maximizing the energy benefits. We also proposed ISA extensions and a programmer interface to utilize the cache architecture for energy efficiency. Our experiments yield significant benefits in cache energy across all benchmarks for a small loss in quality.

References

[1] http://www.everspin.com.
[2] X. Fong et al. Spin-Transfer Torque Memories: Devices, Circuits, and Systems. *In Proc. IEEE*, July 2016.
[3] P. Zhou et al. Energy Reduction for STT-RAM Using Early Write Termination. In *Proc. ICCAD*, November 2009.
[4] X. Wu et al. Hybrid cache architecture with disparate memory technologies. In *Proc. ISCA*, June 2009.
[5] A. Jog et al. Cache revive: Architecting volatile STT-RAM caches for enhanced performance in CMPs. In *Proc. DAC*, June 2012.
[6] V.K. Chippa et al. Analysis and characterization of inherent application resilience for approximate computing. In *Proc. DAC*, June 2013.
[7] Q. Xu et al. Approximate Computing: A Survey. *IEEE Design Test*, Feb 2016.
[8] S. Liu et al. Flikker: Saving DRAM Refresh-power Through Critical Data Partitioning. In *Proc. ASPLOS*, March 2011.
[9] A. Sampson et al. Approximate Storage in Solid-state Memories. In *Proc. MICRO*, December 2013.
[10] I. J. Chang et al. A Priority-Based 6T/8T Hybrid SRAM Architecture for Aggressive Voltage Scaling in Video Applications. *IEEE Trans. on Circuits and Systems for Video Technology*, Feb 2011.
[11] A. Ranjan et al. Approximate Storage for Energy Efficient Spintronic Memories. In *Proc. DAC*, June 2015.
[12] M. Shoushtari et al. Exploiting Partially-Forgetful Memories for Approximate Computing. *IEEE Embedded Systems Letters*, March 2015.
[13] J. S. Miguel et al. Doppelgänger: A Cache for Approximate Computing. In *Proc. MICRO*, December 2015.
[14] J. S. Miguel et al. Load Value Approximation. In *Proc. MICRO*, December 2014.
[15] H. Naemi et al. STTRAM Scaling and Retention Failure. *Intel Technology Journal*, May 2013.
[16] Y. Li et al. Combating Write Penalties Using Software Dispatch for On-Chip MRAM Integration. *IEEE Embedded Systems Letters'12*.
[17] X. Fong et al. SPICE Models for Magnetic Tunnel Junctions Based on Monodomain Approximation, Aug 2013.
[18] S Ikeda et al. A perpendicular-anisotropy CoFeB–MgO magnetic tunnel junction. *Nature materials*, April 2010.
[19] CACTI, www.hpl.hp.com/research/cacti.
[20] N. Binkert et al. The gem5 simulator. *SIGARCH Computer Archiecture News*, August 2011.