CAST: Content-Aware STT-MRAM Cache Write Management for Different Levels of Approximation

Amir Mahdi Hosseini Monazzah*†, Amir M. Rahmani§‡, *Senior Member, IEEE,* Antonio Miele¶, *Member, IEEE,* and Nikil Dutt‡, *Fellow Member, IEEE*

***Abstract*—Spin Transfer Torque Magnetic RAM (STT-MRAM) technology is one of the most promising alternative for Static RAMs (SRAMs) for implementing on-chip memories. Compared with SRAMs, STT-MRAMs benefit from higher density and near-zero leakage power, nonetheless they impose high energy consumption for reliable write operations. However in many applications, absolute data integrity is not required; thus, acting on the current applied in the write operations may represent a novel knob for disciplined approximate computing to obtain energy saving with a minimal quality loss in applications' outputs. This paper proposes CAST, a hardware/software approach to adjust the energy/quality of write operations in STT-MRAM caches in multi-core systems based on the content of requested write operations. CAST utilizes fine-grained cache-line-level actuation knobs with different levels of quality for individual write operations. This unique feature of STT-MRAMs allows to avoid inter-application actuation interference suffered by SRAMs, and makes the approach particularly suitable for systems running multiple applications with mixed accuracy sensitivity. Moreover, CAST exploits another peculiarity of STT-MRAMs represented by the asymmetry and transition-dependency of the write error rate, to further tune in a fine-grained manner the write current to achieve an additional energy saving, even in full-accurate applications. Our evaluations on workloads of full-approximate, mixed-criticality, and full-accurate applications demonstrate up to 57%, 34%, and 21% energy savings over a baseline STT-MRAM cache, respectively, with an acceptable quality of the generated outputs.**

***Index Terms*—STT-MRAM, Approximate computing, Energy consumption, Mixed-criticality.**

I. Introduction

With the technology scaling trend we assisted in recent years, deployment of Static RAMs (SRAMs) have become considerably challenging in particular in the sub-$45nm$ feature size [1]. Indeed, it has been shown that the low density of SRAM cells forces to dedicate approximately 60% of the area of today's multi-core chips to the cache memories; moreover, leakage power may contribute up to 80% of the overall energy consumption of the cache memory [2]. Nonetheless, SRAMs highly suffer from further reliability issues such as high susceptibility to soft-errors. As a result, SRAMs are becoming less appealing in the design of next-generation computing systems, and, in particular, of embedded systems facing with limited sources of energy. Considering the mentioned deficiencies of SRAMs, more recent Non Volatile Memorys (NVMs), such as Phase Change Memory (PCM), Resistive RAM (ReRAM) or Spin Transfer Torque Magnetic RAMs (STT-MRAMs), are becoming an attractive alternative to design multi-core chips. Among the various technologies, as discussed in the International Technology Roadmap for Semiconductors report [3], STT-MRAMs seem to be the most promising for implementing on-chip memories. In fact, STT-MRAMs solve the fundamental limitations of SRAM by offering a high-density, high-speed, non-volatile choice of random access memory.

However, STT-MRAMs suffer from a critical reliability issue, that is the *stochastic switching* [4]; during the write operation, the applied write signal may be unable to change the value of the STT-MRAM cell, thus leading to a *write failure*. Write failure is an asymmetric phenomenon in STT-MRAMs since $0 \rightarrow 1$ transition requires a higher current to make the STT-MRAM cell to reliably commute the value than the $0 \rightarrow 1$ transition [5].

Previous attempts that addressed write failures to keep 100% data integrity used a conservatively-high current for write operations as well as incorporating Error Correction Codes (ECCs) to recover the potential errors (e.g., [6], [7], [8]). Furthermore, these approaches neglect the asymmetry and transition-dependency of the write errors. The result is a relevant energy consumption diminishing some of STT-MRAM advantages over the SRAM.

However, in many contexts, the absolute correctness in the elaborations is not mandatory, especially for such applications that are resilient to a certain extent to errors. As an example, Recognition, Mining, and Synthesis (RMS) applications, that are widely used in embedded systems, could still process information (mainly images or other types of signals) usefully with error-prone elements or when affected by noise in the input data [9]. For this reason, *approximate computing* [10] has been investigated in the recent past as a technique to expose the applications to a controlled level of errors in order to save energy. In systems integrating STT-MRAMs, the regulation of the current (or of the voltage[1]) applied during a memory write operations may represent a novel knob to trade system's energy vs. result quality of the running application. This knob

* School of Computer Engineering, Iran University of Science and Technology (IUST), Tehran, Iran.
† School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran.
‡ Dept. of Computer Science, University of California, Irvine, CA, USA.
§ School of Nursing, University of California, Irvine, CA, USA.
¶ Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Italy.
E-mails: monazzah@iust.ac.ir, a.rahmani@uci.edu, antonio.miele@polimi.it, dutt@uci.edu
Manuscript received November 19, 2018

[1]Do consider that, as discussed later, STT-MRAMs are a resistive memory; therefore, the two types of actuation can be used interchangeably.

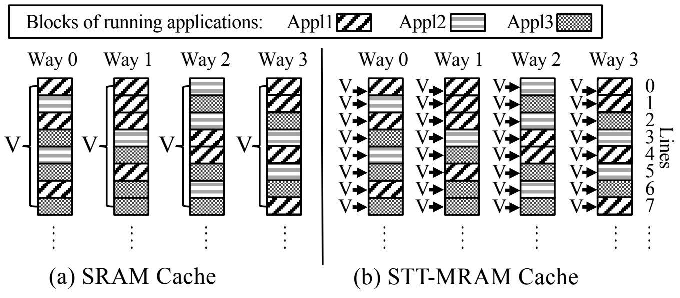


Fig. 1: Comparison between SRAM and STT-MRAM caches on voltage regulation knob (V).

is particularly well-suited in those applications, as the RMS ones, working on a large data-sets which stresses the memory hierarchy and causing a high energy consumption.

In recent years, various studies (e.g., [11], [12]) have considered knobs in on-chip memory sub-systems to define approximate computing techniques exploiting such error resilience for energy saving. Voltage scaling has been already proposed as actuation knob for defining approximate computing techniques for SRAMs (e.g. [11], [13]). However, in SRAM technology voltage scaling is applied per memory bank (i.e. per cache way) while STT-MRAMs allow a more fine-grained actuation at each cache line (i.e., at block granularity). Let refer to the example in Fig. 1, which shows the distribution of the data blocks of three running applications on a cache sub-system organized in four ways and each way in several lines; indeed STT-MRAMs open new possibilities especially when there are various concurrently running applications: i) the voltage knob actuations do not affect any other cache block unlike the SRAMs which require flushing the overall cache way, and ii) multiple applications with different degrees of quality can share the same cache way without affecting each other's guaranteed level of quality.

On the other side, when considering STT-MRAMs, very few works (e.g. [14], [15], [16], [17]) have considered hardware knobs for approximation acting either at design-time or at run-time. The design-time approach in [14] lacks of flexibility at run-time. On the other hand, dynamic approaches in [15], [16] adopted write signal durations as quality/energy knob. This strategy may impose non-uniform access challenges to the memories which may be intolerable for real-time applications with predictability constraints. Finally, the only previous work proposing a cache controller capable at exploiting write current tuning for approximate computing in STT-MRAMs has been proposed in [17]. However, none of them considered asymmetric behavior of STT-MRAMs write operations.

In this paper, which is major extension of our previously published work in [17], we propose CAST, a hardware/-software approach to adjust the quality of write operations in STT-MRAM caches in multi-core systems based on the contents of the requested write operation. CAST is organized in two steps. In the first step, CAST enables software programmers to declare approximate data objects in the source code of the application along with an acceptable write quality guarantees. This information is then passed to the hardware and used during the execution to adjust the main STT-MRAM specific quality-energy knob, i.e., write current. For approximate write operations, these knobs are set to the lowest current that is sufficient to meet the programmer-specified acceptable level of quality. In the second step, CAST adjusts the level of write current based on the type of transition ($0 \rightarrow 1$ or $1 \rightarrow 0$) according to the requested quality level.

To summarize, the main contributions of this paper against our previous proposal [17] are as follows:

- We designed an advanced cache write controller exploiting the STT-MRAM asymmetric write error rate to efficiently manage the energy hungry STT-MRAM write operations at different levels of approximation expressed at the software level.
- We enhanced our previous controller architecture to improve system performance and to reduce the modifications to the on-chip cache hierarchy.
- We performed a systematic analysis of the applicability of the proposed controller to the various levels of the cache hierarchy identifying the lower L2 and Last Level Cache (LLC) to be the best-suited memories.

The rest of this paper is organized as follows. Section II presents STT-MRAM background and its quality-energy knobs. Then, Section III presents a motivational example showing the potentialities of the proposed approach. The details on the software and hardware mechanisms of CAST are introduced in Section IV. An experimental evaluation of CAST is presented in Section V demonstrating its effectiveness, while a discussion of the related work is reported in Section VI. Finally, Section VII concludes the paper.

II. STT-MRAM Quality/Energy Trade-off Knobs

This section provides the necessary background on the STT-MRAM, discussing the structure of the memory cell and the read/write operations. Then, we discuss the STT-MRAM circuit-level knob to trade accuracy for energy.

A. Background on STT-MRAM

The standard STT-MRAM cell (1T-1J), depicted in Fig. 2, includes a Magnitude Tunnel Junction (MTJ), and an access transistor. MTJ consists of an oxide barrier layer that is sandwiched between two ferromagnetic layers. One of the ferromagnetic layers has a fixed magnetic field direction (i.e., reference layer), while the magnetic field direction of the other (i.e., free layer) can be changed. Relative magnetic field

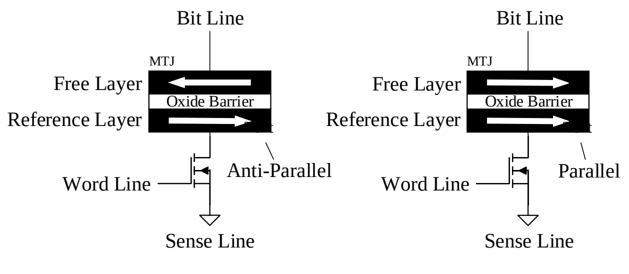


Fig. 2: STT-MRAM 1T1J structure.

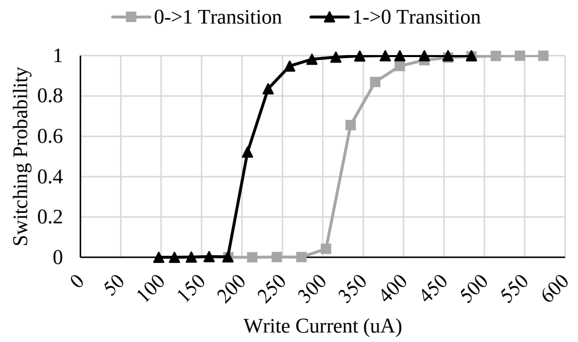


Fig. 3: $32nm$ STT-MRAM cell switching probability for different write current amplitude in 10ns write pulse.

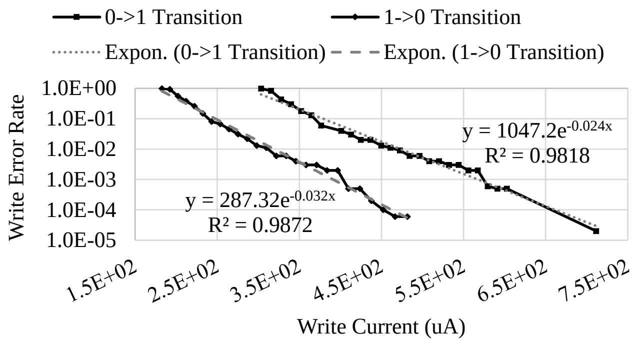


Fig. 4: $32nm$ STT-MRAM cell write error rate vs. write current in different MTJ state transitions (write pulse width of $10ns$).

TABLE I: STT-MRAM HSPICE model configurations.

Parameter	Value ($\mu \pm 3\sigma$)
MTJ length	$32nm$
MTJ width	$96nm$
MTJ thickness	$2.44nm$
Relative initial angle	$0 \pm 35^{\circ}/180 \pm 35^{\circ}$
Transistors technology size	$32nm \pm 1nm$

directions of these layers delivers different resistances used to store values. If the magnetic field directions of the two ferromagnetic layers are in a parallel state, MTJ delivers a low resistance. Otherwise, it delivers a high resistance.

Read and write operations in a STT-MRAM cell are performed by applying either a small current to read MTJ resistance via sense amplifier, or a high current to change the resistance (i.e., write a new value) in the MTJ, respectively. The initial MTJ state affects the total energy required to change its resistance [18]; for $0 \rightarrow 1$ transition, to change the MTJ state from parallel (low resistance) to anti-parallel (high resistance), more energy (power×time) needs to be spent compared with amount of energy needed for the transition in the opposite direction ($1 \rightarrow 0$) [19]. Thus, the MTJ state transition is asymmetric from energy consumption perspective. MTJ state transitions are performed with an uncertainty due to the stochastic nature of STT-MRAMs. Moreover, due to the discussed asymmetry, providing the same level of current for changing the MTJ state leads to different switching probabilities for the two types of transitions based on the initial MTJ state. Fig. 3 depicts the MTJ switching probability vs. applied current at a constant write pulse width. This simulation is executed with HSPICE and the STT-MRAM model introduced in [20] by using the characterization reported in Table I. In particular, the graph shows a difference of 120μA between the two transition directions for the same switching probability.

B. Quality-Energy Knob

Write operations can be performed at different error probabilities and consequently energy consumptions [21], [22]. As stated in [23], the write failure probability for a single-bit cell can be calculated by Equation 1:

$$P_{wf}(t_w) = exp\left(-t_w \times \frac{2\mu_B p(I_w - I_{C_0})}{(c + ln(\Pi^2 \frac{\Delta}{4})) \times (em(1+p^2))}\right) \quad (1)$$

where Δ is the thermal stability factor, I_{C_0} the critical MTJ switching current at 0°K for the specific type of transition, c the Euler constant, e the magnitude of electron charge, m the magnetic momentum of the free layer, p the tunneling spin polarization, μ_B the Bohr magneton, I_w the write current, and t_w the write pulse width. It is worth noting that, apart from the constants, Equation 1 shows that the error probability depends on three input parameters: the amount and duration of the applied write current, I_w and t_w respectively, and the critical MTJ switching current I_{C_0}. In particular, the error probability is asymmetric w.r.t. the type of transition since I_{C_0} is higher for $0 \rightarrow 1$ transition than for the opposite one.

The write error probability in Equation 1 can be considered as an expectation amount of the errors occurred in the overall number of bit-wise write operations, that is the write error rate. To retrieve the write error rates of STT-MRAM at different write current amplitudes we used the same setup of the previous experiment and we conducted several Monte Carlo HSPICE simulations. Fig. 4 depicts the write error rate of a $32nm$ STT-MRAM cell at different write currents. The result is that the set of write current vs. error rate pairs obtained for each transition type can be interpolated with an exponential curve with an inverse relationship. Moreover, it can be noted that for the same current intensity, the error probability for the $0 \rightarrow 1$ is two orders of magnitude higher than for the opposite one. In conclusion, based on Equation 1 and Fig. 4, we have characterized I_w as an effective circuit-level knob to control the quality-energy trade-off during a write operation.

It is worth mentioning that, similarly to the introduced write actuation knob, theoretically, we can defined a corresponding one for read operations (as mentioned in our first proposal [17]). However, from practical perspective, our evaluation results, which will be demonstrated in Table V, showed that write operation imposes about three orders of magnitude higher energy consumption than read operation in STT-MRAMs. Accordingly, since the read actuation knob would not open significant room for energy saving, we ignore it in this work.

III. Motivational Example

The effectiveness of our CAST approach in saving energy while delivering an acceptable output quality is here demonstrated on two image processing applications from MiBench/-Susan benchmark suite [24], namely image smoothing and edge detection, that are generally used as building blocks for more complex RMS applications.

We set up a simulation environment for a multi-core architecture with gem5[2] [25] where we enabled the LLC with the possibility to set the amount of write current. We selected the current values to be used during the memory write operations according to four defined Quality Levels (QLs) from QL0 to QL3 based on the characterization presented in Fig. 4. QL0 represents the accurate write operation, i.e., with an error probability close to 0%; while QL1 to QL3 three increasing approximation levels of write operations. We defined two different knob actuation strategies: the first one, used in QuARK [17], assigns a fixed decreasing amount of current to each QL to obtain a predefined error rate; the second strategy, here adopted by CAST, defines two different current intensities for each QL to be used for $0 \rightarrow 1$ and $1 \rightarrow 0$ memory cell transitions, respectively, thus exploiting the peculiar error rate asymmetry of write operations in STT-MRAMs. It is worth mentioning that since QuARK is more conservative than CAST, the current intensity used at a given QL is the same used in CAST for $0 \rightarrow 1$ transition; in such a way the two approaches provide the same worst-case write error rate and subsequent output QL.

Table II presents the results of the simulation by reporting for each application the produced output image, the corresponding fidelity of the output and the energy saving in percentage of each QL1-3 w.r.t. the golden reference using accurate write, QL0. It is worth mentioning that for the two approaches, the output images and corresponding fidelities are almost the same due to the aligned configurations. As expected, for both the two applications we notice that varying the QL from 1 to 3 causes at the same time an increase in the quality loss, measured in terms of Peak Signal-to-Noise Ratio (PSNR) for the image smoothing and Root Mean Square Error (RMSE) for the edge detection, and in the energy saving.

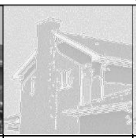
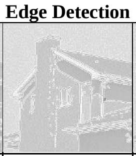
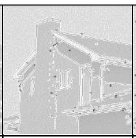
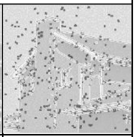
For image smoothing, results show that the PSNR of the output image is well above the threshold of 30 dB in the worst case quality level, QL3. Indeed, no perceptual differences can be visually perceived among the output images reported in the top part of the table, when using different QLs on this application. Thus, a programmer can select QL3 for the LLC write operations with a minimal quality degradation while achieving a memory's energy saving up to the 48% when using QuARK knob actuation; this saving is even more increased to 70% with CAST. On the opposite, the second part of Table II reports that there is a noticeable quality degradation in the output when QL3 is used for edge detection, with a RMSE larger than 31. Since many applications cannot tolerate this level of quality degradation, programmers can select a more precise QL2 or QL1 in LLC and still benefit from the lower energy savings spanning from 45% to 38% when using QuARK strategy, and 72% down to 41% with CAST.

In conclusion, we have shown that since the applied current

[2]The details about the configuration of the STT-MRAM LLC, CAST and gem5 are reported in Section V.

TABLE II: Output fidelities and energy savings delivered by CAST and Quark with different knob settings for Image Smoothing and Edge Detection benchmarks.

Image Smoothing					
Input		QL0 (Accurate)	QL1	QL2	QL3
PSNR		-	82.3	49.1	39.6
Energy Imp. (%)	QuARK	0%	35%	42%	48%
	CAST	5%	41%	70%	70%

Edge Detection					
					
RMSE		-	8.59	14.53	31.93
Energy Imp. (%)	QuARK	0%	38%	45%	47%
	CAST	4%	41%	72%	69%

to a STT-MRAM cell during a write operation affects the write error rate, the quality of write operations can be controlled by selecting the appropriate current level during the application execution. Hence, by considering the minimum amount of write current to satisfy the application output quality threshold, QuARK saves energy during the energy-hungry write operations in STT-MRAMs. Moreover, this result can be further improved as here shown with CAST, if we properly exploit the significant current gap between different write transitions in STT-MRAMs discussed in Section II. In particular, by properly lowering the current intensity to be applied in the $1 \rightarrow 0$ transition in order to obtain the same write error rate of the opposite direction (that represents the worst-case scenario), CAST achieves in this motivating example extra energy saving up to 28% w.r.t. QuARK.

Nevertheless, unlike the QuARK, as shown in Table II, CAST is capable of saving energy consumption even in accurate write operation, which is QL0. Indeed, fixing the same write error rate of accurate write operations at a specific threshold for both transitions in CAST leads to energy saving up to 5% in the less vulnerable transitions, i.e., $1 \rightarrow 0$.

Another relevant aspect that motivated the proposed approach is that not all the locations in the memory space of a running application can be approximated. When considering the two image processing applications, we can approximate the data structures containing the intensities of the various pixels of the processed images, since the actual output is resilient to a certain level of errors. At the opposite we have to guarantee the control variables of the program (e.g., the loop counters) and further control values (e.g., the return value stored in the memory stack during a function call) to be error-free. Indeed, an error corrupting such variables may lead to a disruptive effect in the processing data or even a crash. In our motivational experiments, we profiled the accesses to the LLC; 65% of such accesses can be approximated in the image smoothing and the 70% in the edge detection. Therefore, the actual energy saving is restricted to such percentages of memory accesses. When considering the state-of-the-art coarse-grained approximate approach acting at the granularity of the cache way (e.g., SRAM-based approaches [11], [13]), this analysis should be more conservative by considering the entire content of each single cache way. Thus, we counted that when considering the content of the LLC at each clock cycle of our simulations, only 11% and 49% of the time, for the two applications respectively, a single cache way contains uniform data w.r.t. the corresponding QL, thus considerably reducing the corresponding energy saving. Such a situation is even more exacerbated by the fact that a multi-/many-core system generally executes several applications at the same time, and, in our scenario, each of them may require a different QL. In our example, image smoothing requires QL3 and edge detection QL2. As a conclusion, CAST we here propose is able to overcome such limitations by means of fine-grained per-block knob actuations capable at managing independently heterogeneous QL accesses from the same applications or even from different ones.

IV. The CAST Approach

This section presents CAST in detail. First, we will explore the software support defined to connect applications to be run to the CAST infrastructure. Then, CAST hardware infrastructure will be discussed by presenting the mechanisms that have been integrated into the cache memory architecture.

A. Software Support

Data structures of a program can be categorized into critical (such as loop control variables, which corruption would most likely lead to catastrophic failure or significant quality degradation) or non-critical (variable storing elaborated data whose quality degradation resulting from corruption would be acceptable). This concept has been used before in [26] where a simple partitioning of data – into critical and non-critical – has been shown effective to improve DRAM energy efficiency.

To utilize CAST energy-saving opportunities, the programmer has to identify non-critical data structures and assign to each of them a QL. Typically, non-critical data structures are determined through a set of simulations using fault injection and then measuring the quality of generated output against the golden output. Frameworks such as Rely [27] and Accept [28] could also assist a programmer in identifying non-critical data structures in a program. While this paper focuses on the design of the CAST architecture, future work will be devoted to the definition of a companion methodology for the identification of non-critical data structures and corresponding selection of the proper QL for each of them.

These categorizations can be reflected in a program by annotating different data structures. One category of approaches define type qualifiers and dedicated assembly-level store and load instructions for this purpose [29], [27]. These approaches require major modifications to the compiler, Instruction Set Architecture (ISA), and processor architecture. We consider an alternative approach for CAST that uses dynamic declarations that are enforced at run-time. Our approach is ISA-independent, can be easily integrated in today's processor architectures, and requires minor modifications to the compiler tool-chain. Indeed, we implemented the CAST Application Programming Interface (API) within a special run-time library. With this approach, hardware components of CAST become memory-mapped interfaces. The run-time library uses normal read/write instructions to transfer the information provided by the API to the hardware.

CAST provides two API functions to the programmer: `add_approx` and `remove_approx`, described in Table III. The CAST API communicates with the CAST hardware support (introduced in Section IV-B) to pass the information about non-critical data structures and their acceptable QL, or to remove the previously-declared approximation level. As an example, Fig. 5 shows how CAST API can be used in the source code of a hypothetical face-detection application. According to [30], environmental lighting affects the quality of the face detection algorithm. Thus, considering a reasonable output quality in the worst-case environmental lighting (when the lighting is either very low or very high), we can trade quality for energy saving in the cases when lighting is in a normal condition (between 30 and 70 in this example).

TABLE III: CAST API.

Function	Parameters	Note
add_approx	BaseVA	Base virtual address of approx. memory region
	Size	Size of the approx. memory region
	QL	Required quality guarantee
remove_approx	BaseVA	Base virtual address of approx. memory region
	Size	Size of the approx. memory region

```
unsigned *image;
int quality_level = QL0;
int light;
...
image = (unsigned int*)malloc(WIDTH*HEIGHT*sizeof(
     unsigned));
if(approximation_enabled) {
    light = get_environment_lighting();
    if(light>= 0 && light < 30)
        quality_level = QL0;
    else if(light>= 30 && light < 70)
        quality_level = QL1;
    else /*i.e. light>= 70 && light <= 100*/
        quality_level = QL2;
}
add_approx(image,WIDTH*HEIGHT*sizeof(unsigned),
     quality_level);
...
load_image(image);
face_detection(image);
...
remove_approx(image,WIDTH*HEIGHT*sizeof(unsigned));
...
```

Fig. 5: A pseudo-code example showing how CAST APIs can be used in a face detection application.

B. Hardware Support

To implement the hardware infrastructure of CAST, we consider a multi-core system including private L1 caches and

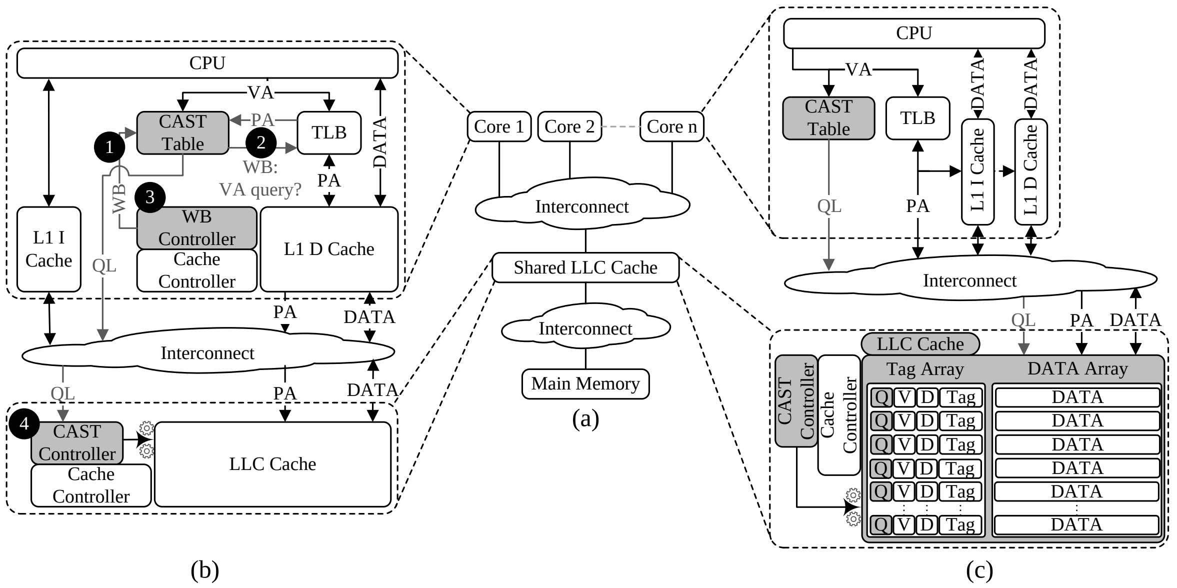


Fig. 6: Proposed CAST architectures: (a) Abstract view of a multicore system, (b) Base Architecture (BA) for LLC, and (c) Performance Efficient Architecture (PEA) for LLC (required changes are highlighted in gray).

a shared L2 LLC, as shown in Fig. 6(a). In particular, we propose to integrate CAST in STT-MRAM LLC as it uses to contains a representative set of data with different QLs from different applications run at different cores in a multi-core system, as will be demonstrated by the analysis presented in Section V. This characteristic of LLC provides a unique opportunity to trade quality for energy consumption in energy-hungry STT-MRAM write operations. It is worth mentioning that CAST can be applied to any type of architecture spanning from the single-core CPU, to homogeneous or heterogeneous multi-core systems. In this paper, a homogeneous multi-core system is considered for the sake of demonstration.

Handling the write requests in CAST is performed in two steps. At the first step, the required QL for the arriving write request is retrieved by the information provided through software API. Since write requests are actuated on LLC only when the block is replaced or requested from the L1 cache, a dedicated hardware infrastructure is used to keep track of the various requests and corresponding QLs. In the second step, when the memory block is written, a dedicated circuitry within the STT-MRAM memory tunes the voltage level for each bit to be written to minimize the energy consumption while achieving the minimum error rate required by the corresponding QL. To present the hardware support of CAST we will discuss the hardware modifications from the two perspectives: 1) architecture-level modifications and 2) circuit-level modifications. The former are applied to both the cache controller and the communication infrastructure connecting to the various cores to integrate CAST in the system. On the other end, the latter is applied to the internal structure of the cache memory to tune the STT-MRAM actuation knob during the write operations.

1) Architecture-level Modifications for CAST: The overall structure of the CAST architecture is shown in Fig. 6(b). Two modules should be added to the typical cache architecture of a multi-core system, i.e., (1) CAST table and (2) CAST controller. Furthermore, we need to change the system interconnect to pass the QL information over it. Do note that CAST does not alter the coherency protocols of the system's cache hierarchy since it only requires intra-cache modifications as it appears in the figure.

The *Base Architecture (BA)* shown in Fig. 6(b) is actually our first proposal presented in [17]. Such an architecture has been later redesigned in a more efficient *Performance Efficient Architecture (PEA)*, shown in Fig. 6(c). As discussed later in detail, the main difference between these architectures is related to the way each Write-Back (WB) request is associated with the required QL. In BA, for each WB request the required QL is retrieved after comparing the address of the requested block against the information provided in CAST table. In contrast, in PEA, the required QL for each block is saved in a reserved space for the future accesses to that block. Thus, in this way, PEA omits the potential time-consuming comparison operations in BA. In the following, first, the internal details of the CAST table and CAST controller are presented and later the different workflow in the block filling operations.

CAST table: The table, instantiated in each core of the system, stores approximation commands received from the API calls. When the `add_approx` is called during the execution of the applications, the transmitted parameters (i.e., `baseVA`, `size`, and `QL`) are stored in a row of the table. Then, calling `remove_approx` removes the corresponding data row.

Every time that processor issues a memory access, this table is searched to get the corresponding QL. Since CAST table saves Virtual Addresss (VAs) while the memory hierarchy components work with Physical Addresss (PAs) (in almost all well-known architectures, e.g., ARM), CAST table is closely coupled with Translation Look-aside Buffer (TLB) as shown in Fig. 6(b) and Fig. 6(c). Thus, each time that a VA is passed to TLB to determine its PA, the VA is also searched in CAST table in parallel. When the address is not found in the CAST table, the maximum QL is returned. Finally, the PA that is retrieved from TLB, enriched with the retrieved QL from CAST table, is fed to the memory hierarchy components.

CAST controller: This module is responsible for controlling the STT-MRAM quality-energy trade-off knob based on the QL information that is provided by the CAST table. CAST controller sits next to the cache controller and works in parallel with it. CAST controller receives the QL set by the CAST table and selects the corresponding minimum settings.

The difference between the functionality of BA and PEA manifests during the cache fillings. In the following, we show how each architecture behaves in these situations.

Cache filling in BA: Fig. 6(b) depicts the proposed CAST BA. Cache fillings in BA perform as a request is missed in the cache memory (e.g., L1 data cache) or a lower-level cache memory executes a WB request. For the first request type (missed in the lower-level cache memory) since the request comes from the processor side, it equipped with its QL. In this situation, CAST controller in LLC actuates the STT-MRAM knob based on the received QL and perform the LLC cache filling. For the later request type (WB from lower-level cache), the process of cache filling depends on the level of the lower cache. If the LLC lower cache is L1 data cache, in BA a set of steps are performed to retrieve the QL of the requests which are numbered in Fig. 6(b). A WB controller has been added to the L1 data cache memory architecture to perform some part of these steps.

1) When a WB request is generated in L1 data cache, WB controller is responsible to pass the PA of the victim block to CAST table.
2) The controller circuit in CAST table then generates VA to PA requests of its entries for TLB and check the return PA from TLB to find-out if the victim block VA is in CAST table or not. Finally, the QL of the victim block is determined.
3) The victim-block which is augmented with its QL is passed to LLC by the WB controller.
4) The arrived WB request in LLC cache is written with the specified QL by the aid of the CAST controller.

Considering WBs from lower-level memories (except L1 data cache) to LLC, all of the arrived WB requests should be satisfied with the highest QL since we cannot access the TLB farther from L1 data cache to perform the mentioned steps. Note that in CAST, all of the LLC WB operations to the upper-level memories are performed just like the typical LLCs.

Cache filling in PEA: Fig. 6(c) depicts the proposed CAST PEA. In case of L1 cache miss, just like BA, since the request is already equipped with QL, PEA only needs to actuate the STT-MRAM knob based on the corresponding QL. In PEA filling operation, the received QL for each request is saved in a specific field in LLC tag memory called *Q* (see Fig. 6(c)). The main difference between CAST BA and PEA is behind in the way that they behave for WB from lower-level cache.

Indeed, unlike the BA that perform a set of steps to retrieve the QL of the victim block and pay performance penalty, PEA uses the stored quality information in *Q* field. Furthermore, unlike the BA, the approach that PEA uses for retrieving the QLs of write requests is same for all of the lower-level caches including L1 data cache. Generally, to retrieve the QL of the WB requests, the proposed PEA depicted in Fig. 6(c) uses the provided information by the *Q* field in the cache tag array of LLC if the WB request is hit in LLC, otherwise the WB request is satisfied in LLC with the highest QL.

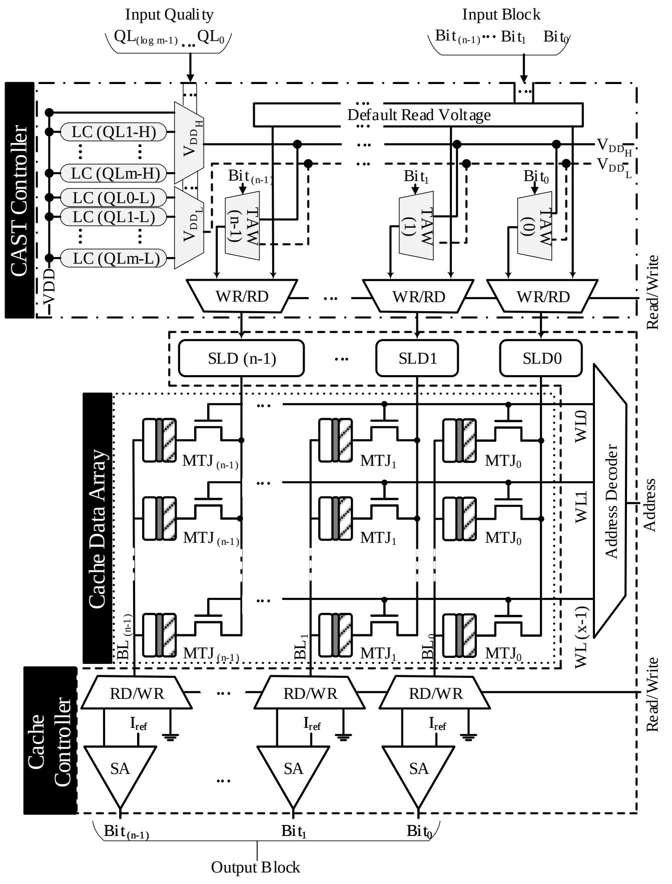


Fig. 7: CAST circuit-level modifications to the LLC.

Note that, since reading the *Q* and *Tag* fields are performed in parallel, CAST does not impose any performance overhead for WB requests, however it imposes a small area overhead to save the QLs of the stored blocks next to *Valid (V)*, and *Dirty (D)* bits in the cache tag array of LLC. Moreover, if the main memory supports QL adjustments for the LLC WB requests, CAST can broadcast the *Q* field of the requested block in the memory hierarchy of the system. Otherwise, all the WB operations to the upper-level memories are performed just like the typical LLCs

2) Circuit-level Modifications for CAST: Enabling the possibility of configuring the STT-MRAM knob, some modifications have been applied to the connections between the CAST controller and the STT-MRAM cache, as shown in gray in Fig. 7. CAST controller is responsible for providing the appropriate voltage levels for the requested read and write operations to the Source Line Driver (SLD) corresponding to each cache data array bit column. To this end, CAST controller performs the following tasks based on the type of arriving request, i.e., read or write:

■ Read request: For an arriving read request, CAST controller only passes the default read voltage level to all the SLDs in the cache controller. *Sense Amplifiers (SAs)* in the cache controller then use the current generated by voltage level to determine the logic values stored in the MTJs based on their measured resistance intensities.

■ Write request: For an arriving write request, a dedicated Transition Aware Write (TAW) manager chooses the write voltage based on the requested QL and type of transition for each cache data array MTJ, selected by the *Address Decoder* module. To determine the appropriate voltage levels for SLDs, we utilize a number of voltage Level Converters (LCs) similar to the one introduced in [31]; each LC is tuned to provide the voltage corresponding to a QL. We adopted the voltage LCs design proposed in [31]; in any case, since STT-MRAM works based on the resistance of MTJ, any other voltage or current LC can be considered here. Moreover, for the same reason, the considered voltage-based peripheral circuits can be replaced with current-based ones.

To have m QLs, the CAST controller includes $(2m-1)$ LCs. Indeed no LC is required for the $0 \rightarrow 1$ transition at highest QL since it is performed at nominal voltage. These LCs are grouped in two classes, i.e., V_{DD_H}, and V_{DD_L} connected to two multiplexers. When a request to update a block arrives, the QL information ($\log_2 m$ bits) received by the CAST controller is used by the two multiplexers to select the corresponding voltages for $0 \rightarrow 1$ transitions and $1 \rightarrow 0$ ones, and to drive the V_{DD_H} and V_{DD_L} rails. Finally, TAWs choose the suitable voltage from V_{DD_H} or V_{DD_L} rails based on the requested transition and passes it to the corresponding SLDs. According to Fig. 7, the TAW module is implemented with a multiplexer. Moreover, if the input bit is ‘1’ (V_{DD_H} is selected) it means that, based on the stored value in the targeted STT-MRAM cell, either there will be a $0 \rightarrow 1$ transition or no transition at all ($1 \rightarrow 1$). Since our SLDs are equipped with Early Write Termination (EWT) module [32], there is not any energy consumption for $1 \rightarrow 1$ case. When the input bit is ‘0’ (V_{DD_L} is selected), one of the two complementary situations will occur based on the stored value in the targeted STT-MRAM cell, i.e. $1 \rightarrow 0$ and $0 \rightarrow 0$, that will be managed in the same way discussed above.

Since CAST controls STT-MRAM knob at cache block granularity, the virtual address ranges provided by the API should be aligned to the cache block boundaries, because the cache blocks that contain a mix of critical and non-critical data should not be approximated. An address alignment module embedded in the CAST table performs the required alignment.

V. Experimental Evaluations

In the following, we show experiments with a mix of approximate and accurate applications to evaluate the CAST’s capabilities in saving energy in the on-chip caches under different levels of accuracy requested by applications.

A. Experimental Setup

1) Simulation Environment: To implement our scheme in detail, we modified the cache architecture in the gem5 simulation framework for multi-core architecture [25]. In particular, we enabled CAST for L2 LLC as an exemplar. We used gem5’s pseudo-instructions for implementing CAST’s language extensions. Moreover, we enhanced gem5 to enable random fault injection during the write operations in the L2 cache memory based on the probability described in Equation 1 per each single memory cell and with a uniform distribution over different cell positions. In the experiments, we considered a quad-core processor for embedded applications described in Table IV.

TABLE IV: gem5 settings for the used multi-core system.

Parameter	Value	Parameter	Value
ISA	ARMv7-A	L1 $ Size, Assoc.	32KB, 4
No. of Cores	4	L2 $ Size, Assoc.	1MB, 16
Cache Config.	L1 (Private) L2 (Shared, CAST-enabled)	Cache Block Size	64B

TABLE V: Quality-energy transducer map for 1MB L2 CAST-enabled STT-MRAM cache. Energy consumptions are reported for 64-byte cache line.

Quality	Read Energy	Write Error Rate	Write Energy $0 \rightarrow 1$	Write Energy $1 \rightarrow 0$
QL0 (Baseline)	$44pJ$	10^{-9}	$166nJ$	$87.9nJ$
QL1	$44pJ$	10^{-5}	$74.7nJ$	$37.8nJ$
QL2	$44pJ$	10^{-4}	$57.5nJ$	$28.5nJ$
QL3	$44pJ$	10^{-3}	$42.6nJ$	$20.5nJ$

2) Characterization of STT-MRAM Caches: To characterize the cache model in gem5, we used Bit Error Rate (BER) and access current data from HSPICE Monte Carlo simulations and through the formulas retrieved by regression methods (depicted in Fig. 4). Then, we profiled an 1MB L2 STT-MRAM cache in NVSim [33] to extract energy consumption and characterize the gem5 cache model.

We defined 4 QLs, with increasing approximation level from 0 to 3; the details of error rates and energy consumption are reported in Table V. In particular, we considered 10^{-9} as the practical error rate for write operations in full-accurate mode, that is QL0, and we consequently computed from Fig. 4 the corresponding write current amplitude, and subsequent energy consumption from the NVSim profiling. Then, we selected for the approximate QLs three points on the same plot having equidistant distance on the error probability (in logarithmic scale), and we characterized them in a similar way. It is worth noting that such points have been selected for the sake of demonstration of the effectiveness of the proposed approach. The definition of the QLs requires an accurate characterization of the specific architecture under design, also based on an extensive simulation of benchmarks to investigate the write output quality vs. energy saving; since this aspect is highly connected to the definition of the software methodology mentioned in Section IV-A, it is left as a future work. Finally, to evaluate the area and power consumption overheads of the peripheral circuits in CAST controller we used Synopsys Design Compiler®. It is worth noting that due to our availability of the only $45nm$ technology for Synopsys Design Compiler®, we rerun all characterizations in this technology node.

3) Benchmarks: In our evaluations, we used the applications from the MiBench benchmark suites [24] listed in Table VI. The first subset is composed of RMS applications

TABLE VII: List of workload combinations.

Package	Workload Mixes	Benchmarks	Package	Workload Mixes	Benchmarks	Package	Workload Mixes	Benchmarks
Full Approximate	App1	Corner Detection (QL0~3)	**Mixed Criticality**	Mix1	Blackscholes (QL3)	**Full Accurate**	Acc1	CRC (QL0)
		Sobel (QL0~3)			Basicmath (QL0)			Basicmath (QL0)
		Image Smoothing (QL0~3)			Scale (QL1)			Dijkstra (QL0)
		Blackscholes (QL0~3)			Dijkstra (QL0)			FFT (QL0)
	App2	Blackscholes (QL0~3)		Mix2	Sobel (QL2)		Acc2	Sha (QL0)
		Scale (QL0~3)			FFT (QL0)			Patricia (QL0)
		Sobel (QL0~3)			Edge Detection (QL3)			CRC (QL0)
		Image Smoothing (QL0~3)			CRC (QL0)			Basicmath (QL0)
	App3	Sobel (QL0~3)		Mix3	Sha (QL0)		Acc3	Dijkstra (QL0)
		Corner Detection (QL0~3)			Corner Detection (QL1)			Patricia (QL0)
		Scale (QL0~3)			Image Smoothing (QL2)			Sha (QL0)
		Edge Detection (QL0~3)			Patricia (QL0)			FFT (QL0)
	App4	Blackscholes (QL0~3)		Mix4	Corner Detection (QL1)		Acc4	CRC (QL0)
		Scale (QL0~3)			Blackscholes (QL3)			Sha (QL0)
		Edge Detection (QL0~3)			Basicmath (QL0)			FFT (QL0)
		Image Smoothing (QL0~3)			Sha (QL0)			Dijkstra (QL0)

TABLE VI: List of applications.

Benchmarks		Domain	Quality Metric
Approximate	Corner Detection	Image processing	Mean Pixel Difference (MPD)
	Edge Detection	Image processing	RMSE
	Image Smoothing	Image processing	PSNR
	Black Scholes	Financial Analysis	Average Relative Error (ARE)
	Image Scale	Image processing	PSNR
	Sobel Filter	Image processing	MPD
Accurate	Basicmath	Automotive	–
	CRC	Telecommunications	–
	Dijkstra	Network	–
	FFT	Telecommunications	–
	Patricia	Network	–
	SHA	Security	–

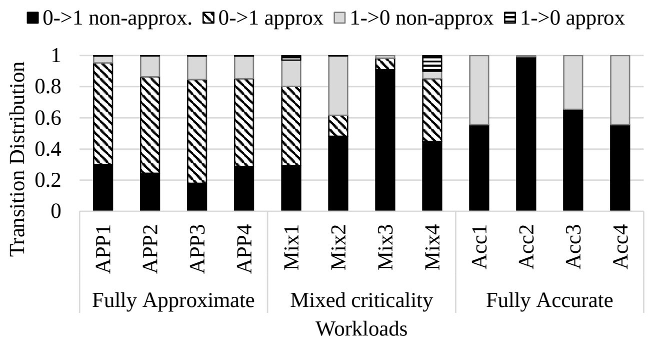


Fig. 8: L2 cache workload access patterns.

that can be approximated; for each of them, we also report the corresponding quality metrics to evaluate the produced results. We annotated RMS applications by inserting `add_approx` and `remove_approx` in the source code for non-critical data objects. Besides the approximate benchmarks, we also evaluate the efficiency of CAST in dealing with accurate benchmarks, reported in the second part of the table, to save the energy consumption in $1 \rightarrow 0$ transitions.

4) Workload Combinations: To evaluate the efficiency of CAST, we considered the effects of multi-programming in a shared L2 cache equipped with CAST. To this end, we introduced three packages of workload mixes, i.e., *Full Approximate*, *Mixed Critical*, and *Full Accurate* in Table VII. The workloads in full approximate package were run in fixed-quality mode in which we run each workload combination in four QLs: QL0 (i.e., full-accurate, QL1, QL2 and QL3 (i.e., least-quality) configurations.

In contrast to the full approximate package, in the mixed-critical package, we considered different levels of quality for each benchmark in each combination. Our empirically-chosen criteria to set the QL for each approximate application was based on the user-level Quality of Service (QoS) desirability. For instance, as shown in the motivational example, QL3 provided an acceptable QoS for Image Smoothing, while for Edge Detection QL2 was more desirable. Note that these QLs can be chosen by the user/designer w.r.t any other policies. For accurate applications in each combination, we choose the highest quality level, i.e., QL0.

According to support our discussions about the retrieved results, in Fig. 8, we explore the shared L2 cache access pattern distribution for running the workload combinations mentioned in Table VII. To this end, we can see that approximate transitions contribute to up to 67% of transitions in L2 cache when running workloads from the full approximate package (App3). Furthermore, approximate transitions also actively contribute to the transitions of mixed critical workloads. For example, over 53% of transitions in *mix1* is approximate transitions. From the full accurate package perspective, while workloads like *Acc2* does not provide a good opportunity to save energy consumption in CAST, the other workloads provide energy-saving opportunities for CAST controller because of their considerable portion of $1 \rightarrow 0$ transitions.

B. Where to Use CAST?

The first experimental issue is to analyze the most suitable place(s) to implement CAST in the cache memory hierarchy. To this end, we support our discussion by answering a fundamental question, which is *"How much room do we have to apply our approximation technique in each memory level?"*. To answer this question, we conducted a systematic analysis of the various levels of the on-chip cache hierarchy to find out the possible opportunities for approximation at each level. Besides the contribution of approximate accesses in the STT-MRAM memory, we also analyzed the contribution of each type of transitions to explore the usage efficiency of asymmetric voltage actuation.

Fig. 9 contains the L1 data and L2 caches' access information for the benchmarks listed in Tables VI. It can be seen from Fig. 9(a) that the contribution of $1 \rightarrow 0$ transitions is much higher than $0 \rightarrow 1$ transitions for both approximate and accurate applications. On average, over 96% of transitions in the approximate application, and over 86% of transitions

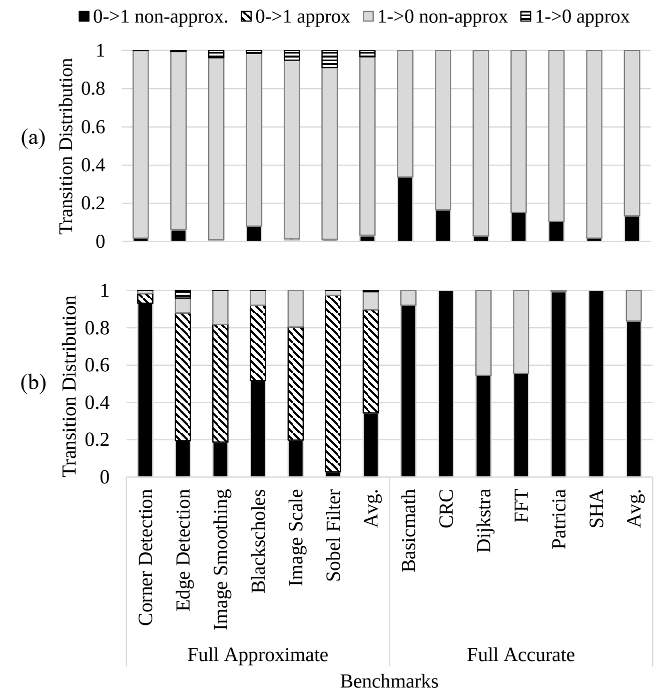


Fig. 9: Benchmarks' access patterns: (a) L1 data cache, and (b) L2 cache.

in the accurate applications belong to $1 \rightarrow 0$ transitions. Furthermore, considering approximate applications, we can see in Fig. 9(a) that only about 4% of transitions in the L1 data cache is for approximate write requests. This happens due to the high access frequency of the processor to lots of critical data (i.e., loop counters, intermediate variables, etc.) that should be resolved by L1 data cache. Thus, considering approximate techniques that proposed their approaches for L1 data caches or level one ScratchPad Memories (SPMs), they would not benefit from considerable energy saving.

Fig. 9(b) depicts the L2 access patterns over approximate and accurate applications. Unlike the L1 data cache, it can be seen in Fig. 9(b) that most of the transitions in L2 cache are $0 \rightarrow 1$ transitions. On average, over 89% of transitions in approximate applications and over 83% of transitions in the accurate applications belong to $0 \rightarrow 1$ transitions. Considering approximate applications we can see in Fig. 9(b) that over 56% of transitions is for approximate write requests, on average. As a conclusion L2 cache is the most appropriate place where to apply approximation techniques like CAST while it is discouraged to consider L1. Nonetheless, similar experimental evidence may also be obtained for layers lower than L2.

This experimental evidence is also enforced by various past studies on STT-MRAM adoption in on-chip memories, such as cache and SPM; indeed they claim that utilizing STT-MRAM is only practical for lower L2 and lower-level on-chip memories [34], [35], because STT-MRAM has fundamental issues from performance, dynamic energy consumption, and endurance. Accordingly, utilizing STT-MRAM for L1 caches that are frequently accessed by CPUs is not feasible. Furthermore, when considering L1 SPMs implemented with STT-MRAM, to address the mentioned issues we need to add too many restrictions in SPM data mapping algorithm which may highly affect the SPM utilization. To this end, we proposed to implement CAST in cache levels lower than L1.

C. Experimental Results

In the following, we evaluate how our approach helps energy saving in a system that its L2 is equipped with CAST. We explore the efficiency of CAST from these perspectives: 1) energy saving, 2) delivered quality, and 3) performance penalty. To this end, we compare CAST with QuARK [17] and a baseline architecture that only benefits from EWT mechanism [32] to save the energy. In order to have a fair comparison, we disabled in QuARK the actuation knob during read operations, even if its benefit is marginal. Finally, we provide a discussion on the performance and area overheads of proposed architectures in the CAST-enable L2.

1) Energy Saving: Fig. 10 depicts the energy consumption of L2 in different scenarios. Since there are considerable approximate accesses in L2, QuARK considerably reduces the write energy consumption in the L2 cache for full approximate and mixed-critical workloads. For these workloads, CAST outperforms QuARK since it utilizes the transition aware write voltage management in both accurate and approximate $1 \rightarrow 0$ transitions. Considering approximate benchmarks, Fig. 10 confirms that QuARK and CAST can save write energy consumption in L2 by up to 50% and 57% for the full approximate workload package, respectively. Besides, for the mixed-critical workload package, QuARK and CAST can save write energy consumption in L2 by up to 27% and 34%, respectively. Finally, for a full accurate workload package, while QuARK cannot save energy consumption, CAST saves energy consumption by up to 21%. Finally, based on the synthesis results described in Section V-C4, the energy overhead of the CAST controller is negligible (0.003%).

2) Quality: Fig. 11 depicts the CAST output quality when running full-approximate benchmarks at various quality levels for in L2. Each plot presents the results according to the quality metric considered for each specific application (as reported in Table VI. We may notice that in many cases there is a quite significant variation of the output quality depending on the selected QL that highly depends on the specific application; for instance for the *corner detection* the quality lost is up to the 7% at QL3 configuration, while for *blackscholes* it can be reached up to 56% at the same configuration. Furthermore, the quality lost acceleration from QL1 to QL3 is also differ in various applications. For example, while in *scale* we observe almost a uniform pattern in quality degradation when moving from QL1 to QL3, for *edge detection* we see an aggressive quality degeneration when moving from QL1 to QL3. Accordingly, the later application is more sensitive to CAST actuation management. As a conclusion, by analysis such results, the software engineer should select the appropriate QL to guarantee to the user the desired QoS; based on how much stringent/relaxed such requirements are the system will save less/more energy.

3) Performance: We tested the performance of the two architectures for deploying CAST in cache memories, i.e., BA,

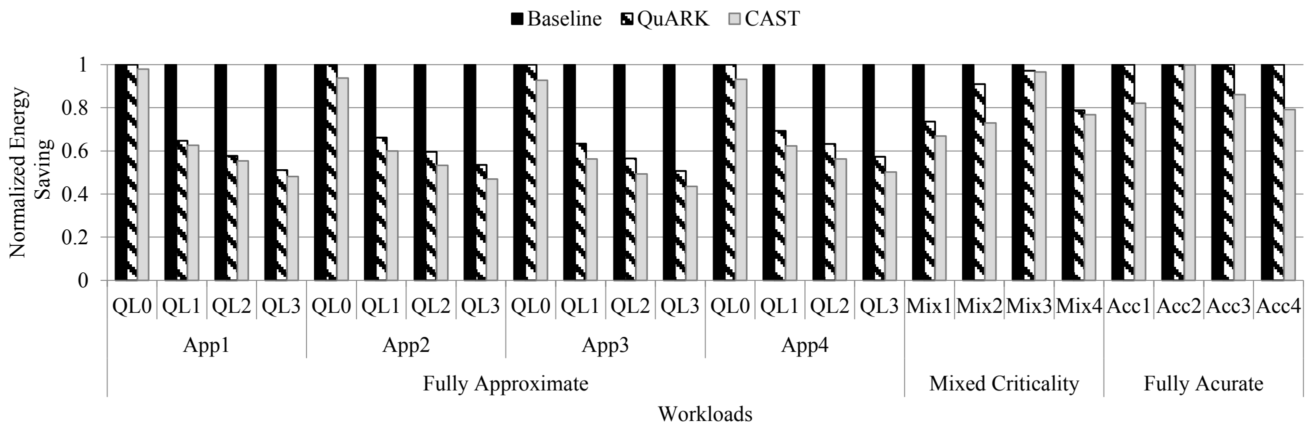


Fig. 10: Energy saving in different scenarios using quality-energy knob in L2.

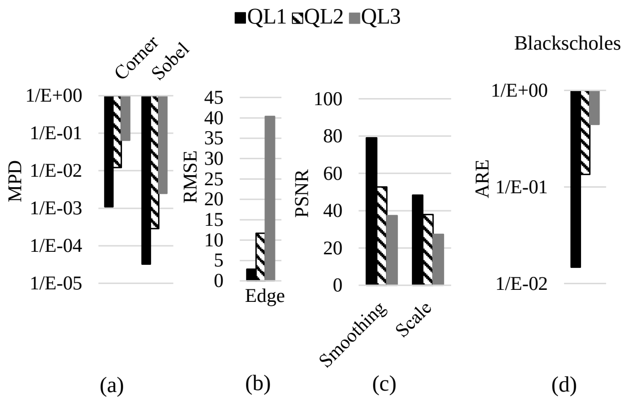


Fig. 11: CAST delivered output qualities in various scenarios.

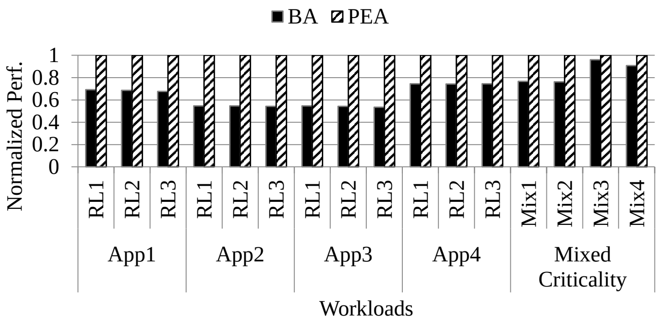


Fig. 12: CAST's performance at different configurations.

inherited from QuARK, and PEA. Considering a sequential search operation at CAST table with two clock cycle latency for each search access, Fig. 12 depicts the normalized performance of the system in the two cases. It is possible to note that BA imposes up to 47% performance overhead to the system in comparison with PEA; this is mainly because in the former we need to adjust the quality of the accesses to CAST-enabled caches. The main reason to observe this amount of performance degradation is due to CAST table search to look up the quality levels of L1 cache write-back accesses. Accordingly, the main contributor to the performance overhead of CAST in BA is the total amount of write-backs operations that should be served at different workloads. This kind of issue is solved by the newly proposed PEA, presenting no performance overhead during the quality adjustment.

Moreover, in BA, CAST does not impose any performance overhead when a memory access hits the first level cache since the CAST table and TLB lookups are performed in parallel. The only situation where CAST imposes extra cycles to execution time is when a dirty block in L1 data cache has to be written back to L2.

In this case, the L1 data cache WB controller should set the quality level of this write-back request for L2 cache write operation. To this end, the PA of the target block in L1 data cache should be searched in the VA entries of the CAST table. Accordingly, for each entry of CAST table, the corresponding PA are generated through TLB queries. Unlike BA, in PEA since we save the quality level of each access in the corresponding quality field of each block, we do not pay any performance overhead for retrieving the quality levels of write-back requests.

4) Discussion on Area Overhead: In the previous subsection, we see that BA may impose up to 47% performance overhead to the design based on the amount of write-back requests of each workload, in comparison with PEA design. On the other hand, the higher performance of PEA design achieved by imposing area overhead to save the QLs of cache blocks at the provided Q fields in the tag memory. To calculate the area overhead of PEA design comparing against BA, we modeled the considered 1MB L2 cache in NVSim.

Fig. 13 depicts the occupied area of tag array, Q fields, and data array in the BA design and the PEA one, respectively.

To calculate the area of the Q fields in the tag array of the L2 cache we considered 2 bits for saving the quality levels of each data block, 20 bits for tag address corresponding to each data block, 1-bit dirty flag, and 1-bit validity flag. Accordingly, Fig. 13(a) shows that the L2 BA design would occupy about $1.68mm^2$ of the chip. In particular, the data array of the L2 is the main contributor for occupying the area in L2. Indeed, the tag array of L2 only occupies 11% of the L2 chip area, and the remaining 89% was used by a data array. Considering the PEA design, adding the Q fields to the tag part of the L2 requires about 1% of the L2 area as depicted in Fig. 13(b).

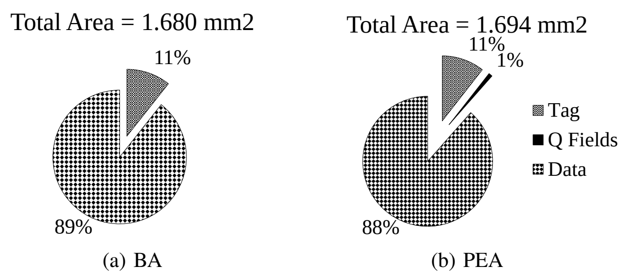


Fig. 13: CAST BA and PEA L2 chip area comparison.

Finally, we designed the CAST controller peripheral circuits and synthesized using Synopsys Design Compiler®. Indeed, we synthesized the 7 LCs, their corresponding V_{DD_L} and V_{DD_H} multiplexers, and the TAW modules in Synopsys Design Compiler® to characterize them from area and power perspectives. Then, we added the results to NVSim's area (and power) reports, obtaining an overhead of about 0.16% (and 0.003%) w.r.t. the basic cache architecture, which is negligible.

As a conclusion, PEA demonstrates to be more performance effective with a negligible area overhead.

VI. Related Work

In the past decade, several studies have explored approximate computing in the fields of computation and programming language [10]. Considering the hardware platforms that execute the approximate computing applications, an approximation can also be applied to different components of the system, including memory hierarchy.

Reviewing the previous efforts in applying approximate computing approach to the memory hierarchy, we can classify the literature studies into two groups. In the first group, there are a bunch of studies that tried to analyze the vulnerabilities of different parts of data structures across the programs [36], [37]. In [36] a dynamic solution to check the output quality of applications during the run-time was introduced. While the paper proposed how to track the output quality, it did not introduce any approach to tune the actuation knobs based on the online measured output quality. These kinds of studies can be utilized to provide a closed-loop approximate computing approach if they are attached to the approaches that control the actuation knobs across the hardware platforms. In [37], another error resilience analysis approach, was proposed especially for the convectional neural networks to find the vulnerability of different layers. Unlike the first group, the studies in the second group focused on managing the actuation knobs across the hardware platforms for approximate data considering different goals like improvement in energy consumption (e.g., [11], [15], [26], [14]), performance (e.g., [14]) or area (e.g., [12], [38]). The proposed approaches either targeted on-chip memories or off-chip memories.

On-chip memories which are typically implemented by SRAM technology face two important challenges, i.e., high static power, and low density. A promising approach to deal with these challenges is to utilize the opportunities delivered by a quality adjustment in approximate computing applications. To deal with the first challenge, the authors of [11] have shown how several cache ways of a SRAM cache can operate at a lower than the nominal voltage. These relaxed ways are used to hold approximate data, and the protected ways are used to hold critical data. On the other hand, authors of [12] exploit the approximate similarity of data in last-level caches to use the cache storage more efficiently and hence save static and dynamic energy.

With the development of non-volatile memory technologies in the recent years, several studies consider the potentials of deploying non-volatile memories at the memory hierarchy of platforms running approximate computing applications (e.g., [39], [40], [15], [38], [14], [26], [41]). Two recent works [15], [16] have considered applying the idea of approximation to STT-MRAM structures. The authors of [15] have utilized similar energy-quality knobs that are used in CAST. However, they are applied to scratchpad memories used in a vector processor running a single application. We showed that unlike the lower-level on-chip memories (like LLC) the close-to-CPU on-chip memories like L1 caches and SPMs do not provide significant opportunities for approximation since the ratio of approximate accesses in these types of memories are not considerable comparing with lower-level on-chip memories.

In [16], the possibility of approximate computing in L2 STT-MRAM cache was investigated like our previous proposal, i.e., QuARK. There are two differences between STAx-Cache proposed in [16] and CAST. Firstly, they did not consider write current actuation knob in their approach. Secondly, they did not utilize the opportunities of energy saving delivers by the asymmetric write behaviors of STT-MRAMs. There are also other efforts to propose approximate computing approaches for STT-MRAM based cache memories which are either not discuss specific actuation knob [40] or not considered dynamic voltage scaling and just provided a special STT-MRAM cell design and utilized it in their architecture [42]. Furthermore, in [38], the authors propose an approximation-aware Multi-Level Cells (MLCs) STT-MRAM cache architecture to trade quality with large capacity delivered by MLC STT-MRAM. The technique presented in [14] uses approximate computing to tolerate the increased retention failure rate caused by relaxing the thermal stability factor of STT-MRAM.

Similarly, DRAM off-chip memories can reduce the power spent on refresh cycles where bit flips are allowed [26], [43]. The work in [41] shows how approximate storage in persistent memories, where cells are at the risk of wear out, can reduce the number of flipped bits to prolong the device lifetime.

Finally, in [39] an offline Integer Linear Programming based tuning of approximate knob across the STT-MRAM based SPM and PCM based memory. The authors in [39] profiled different scenarios for tuning the approximation knob and tried to minimize the Error Per Second parameter of the memories as low as possible, but they did not discuss the actuation knob management that should be considered to conduct the best-case scenarios.

VII. Conclusion

This paper presents CAST, a hardware/software approach to adjust the quality of write operations in STT-MRAM caches in multi-core systems based on the contents of requested write operation. Because the write error rate of STT-MRAM are asymmetric and transition-dependent, controlling the amounts of applied write current based on the write transition directions in CAST not only leads to energy saving in typical applications but also introduces new opportunities for energy saving by further manipulating energy-quality knobs in many approximate computing applications. Our evaluations on sets of full approximate, mixed-criticality, and entirely accurate applications demonstrate up to 57%, 34%, and 21% energy savings over a baseline STT-MRAM cache, respectively, with an acceptable quality of the generated outputs.

Acknowledgments

This paper is dedicated to the memory of our wonderful colleague, Professor Seyed Ghassem Miremadi who passed away in April 2017.

We acknowledge financial support by the Marie Curie Actions of the European Union's H2020 Programme.

References

[1] B. Raj, A. K. Saxena, and S. Dasgupta, "Nanoscale FinFET Based SRAM Cell Design: Analysis of Performance Metric, Process Variation, Underlapped FinFET, and Temperature Effect," *IEEE Circuits and Systems Magazine*, vol. 11, no. 3, pp. 38–50, 2011.

[2] E. Cheshmikhani, H. Farbeh, S. G. Miremadi, and H. Asadi, "TA-LRW: A Replacement Policy for Error Rate Reduction in STT-MRAM Caches," *IEEE Trans. on Computers*, vol. 68, no. 3, pp. 455–470, 2019.

[3] International Technology Roadmap for Semiconductors (ITRS), "Emerging Research Devices: Focus Teams Presentations: Beyond CMOS," https://www.dropbox.com/sh/6xq737bg6pww9gq/AACQWcdHLffUeVloszVY6Bkla?dl=0&preview=2013ERD.pdf, accessed: 2019-06-13.

[4] T. Devolder, J. Hayakawa, K. Ito, H. Takahashi, S. Ikeda, P. Crozat, N. Zerounian, J.-V. Kim, C. Chappert, and H. Ohno, "Single-Shot Time-Resolved Measurements of Nanosecond-Scale Spin-Transfer Induced Switching: Stochastic Versus Deterministic Aspects," *Phys. Rev. Lett.*, vol. 100, 2008.

[5] X. Bi, Z. Sun, H. Li, and W. Wu, "Probabilistic design methodology to improve run-time stability and performance of STT-RAM caches," in *Proc. of Intl. Conf. on Computer-Aided Design (ICCAD)*, 2012, pp. 88–94.

[6] H. Sun, C. Liu, N. Zheng, T. Min, and T. Zhang, "Design Techniques to Improve the Device Write Margin for MRAM-based Cache Memory," in *Proc. of Great Lakes Symp. on VLSI (GLSVLSI)*, 2011, pp. 97–102.

[7] W. Kang, W. Zhao, J. . Klein, Y. Zhang, C. Chappert, and D. Ravelosona, "High Reliability Sensing Circuit for Deep Submicron Spin Transfer Torque Magnetic Random Access Memory," *Electronics Letters*, vol. 49, no. 20, pp. 1283–1285, 2013.

[8] Z. Azad, H. Farbeh, A. M. H. Monazzah, and S. G. Miremadi, "An Efficient Protection Technique for Last Level STT-RAM Caches in Multi-Core Processors," *IEEE Trans. on Parallel and Distributed Systems*, vol. 28, no. 6, pp. 1564–1577, 2017.

[9] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Analysis and Characterization of Inherent Application Resilience for Approximate Computing," in *Proc. of Design Automation Conf. (DAC)*, 2013, pp. 1–9.

[10] S. Mittal, "A Survey of Techniques for Approximate Computing," *ACM Comput. Surv.*, vol. 48, no. 4, pp. 62:1–62:33, 2016.

[11] M. Shoushtari, A. BanaiyanMofrad, and N. Dutt, "Exploiting Partially-Forgetful Memories for Approximate Computing," *IEEE Embedded Systems Letters*, vol. 7, no. 1, pp. 19–22, 2015.

[12] J. S. Miguel, J. Albericio, A. Moshovos, and N. E. Jerger, "Doppelgänger: A Cache for Approximate Computing," in *Proc.of Intl. Symp. on Microarchitecture (MICRO)*, 2015, pp. 50–61.

[13] F. Frustaci, D. Blaauw, D. Sylvester, and M. Alioto, "Approximate SRAMs with dynamic energy-quality management," *IEEE Trans. on VLSI Systems*, vol. 24, no. 6, pp. 2128–2141, 2016.

[14] N. Sayed, F. Oboril, A. Shirvanian, R. Bishnoi, and M. B. Tahoori, "Exploiting STT-MRAM for approximate computing," in *Proc. of European Test Symp. (ETS)*, 2017, pp. 1–6.

[15] A. Ranjan, S. Venkataramani, X. Fong, K. Roy, and A. Raghunathan, "Approximate storage for energy efficient spintronic memories," in *Proc. of Design Automation Conf. (DAC)*, 2015, pp. 1–6.

[16] A. Ranjan, S. Venkataramani, Z. Pajouhi, R. Venkatesan, K. Roy, and A. Raghunathan, "STAxCache: An approximate, energy efficient STT-MRAM cache," in *Proc. of Conf. Design, Automation & Test in Europe*, 2017, pp. 356–361.

[17] A. M. H. Monazzah, M. Shoushtari, S. G. Miremadi, A. M. Rahmani, and N. Dutt, "QuARK: Quality-configurable approximate STT-MRAM cache by fine-grained tuning of reliability-energy knobs," in *Proc. of Intl. Symp. on Low Power Electronics and Design (ISLPED)*, 2017, pp. 1–6.

[18] Y. Zhang, W. Wen, and Y. Chen, "STT-RAM cell design considering MTJ asymmetric switching," *SPIN*, vol. 2, 2012.

[19] Y. Zhang, X. Wang, Y. Li, A. K. Jones, and Y. Chen, "Asymmetry of MTJ switching and its implication to STT-RAM designs," in *Proc. of Design, Automation Test in Europe Conf. (DATE)*, 2012, pp. 1313–1318.

[20] J. Kim, A. Chen, B. Behin-Aein, S. Kumar, J. Wang, and C. Kim, "A technology-agnostic MTJ SPICE model with user-defined dimensions for STT-MRAM scalability studies," in *Proc. of Custom Integrated Circuits Conf. (CICC)*, 2015, pp. 1–4.

[21] K. Munira, W. H. Butler, and A. W. Ghosh, "A Quasi-Analytical Model for Energy-Delay-Reliability Tradeoff Studies During Write Operations in a Perpendicular STT-RAM Cell," *IEEE Trans. on Electron Devices*, vol. 59, no. 8, pp. 2221–2226, 2012.

[22] H. Li, X. Wang, Z. Ong, W. Wong, Y. Zhang, P. Wang, and Y. Chen, "Performance, Power, and Reliability Tradeoffs of STT-RAM Cell Subject to Architecture-Level Requirement," *IEEE Trans. on Magnetics*, vol. 47, no. 10, pp. 2356–2359, 2011.

[23] M. M. de Castro *et al.*, "Precessional spin-transfer switching in a magnetic tunnel junction with a synthetic antiferromagnetic perpendicular polarizer," *Journal of Applied Physics*, vol. 111, no. 7, p. 07C912, 2012.

[24] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "MiBench: A Free, Commercially Representative Embedded Benchmark Suite," in *Proc. of Intl. Workshop on Workload Characterization (WWC)*, 2001, pp. 3–14.

[25] 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," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, 2011.

[26] S. Liu, K. Pattabiraman, T. Moscibroda, and B. Zorn, "Flikker: Saving DRAM Refresh-power Through Critical Data Partitioning," in *Proc. of Intl. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2011.

[27] M. Carbin, S. Misailovic, and M. C. Rinard, "Verifying Quantitative Reliability for Programs That Execute on Unreliable Hardware," in *Proc. of Intl. Conf. Object Oriented Programming Systems Languages and Applications (OOPSLA)*, 2013, pp. 33–52.

[28] A. Sampson, A. Baixo, B. Ransford, T. Moreau, J. Yip, L. Ceze, and M. Oskin, "Accept: A programmer-guided compiler framework for practical approximate computing," *University of Washington Technical Report UW-CSE-15-01*, vol. 1, no. 2, 2015.

[29] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman, "EnerJ: Approximate Data Types for Safe and General Low-power Computation," in *Proc. of SIGPLAN Conf. on Programming Language Design and Implementation (PLDI)*, 2011, pp. 164–174.

[30] S. Ghatge and V. Dixit, "Robust face recognition under difficult lighting conditions," *Intl. Journal of Technological Exploration and Learning (IJTEL)*, vol. 1, no. 1, 2012.

[31] D. Lee, S. K. Gupta, and K. Roy, "High-performance Low-energy STT MRAM Based on Balanced Write Scheme," in *Proc. of Intl. Symp. on Low Power Electronics and Design (ISLPED)*, 2012, pp. 2356–2359.

[32] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "Energy reduction for stt-ram using early write termination," in *Proc. of Intl. Conf. on Computer-Aided Design (ICCAD)*, 2009, pp. 264–268.

[33] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, "NVSim: A Circuit-Level Performance, Energy, and Area Model for Emerging Nonvolatile Memory," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 7, pp. 994–1007, 2012.

[34] S. Wang, G. Duan, Y. Li, and Q. Dong, "Word- and partition-level write variation reduction for improving non-volatile cache lifetime," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 23, no. 1, pp. 4:1–4:18, 2017.

[35] L. Chang, Z. Wang, Y. Gao, W. Kang, Y. Zhang, and W. Zhao, "Evaluation of spin-Hall-assisted STT-MRAM for cache replacement," in *Proc. of Intl. Symp. on Nanoscale Architectures (NANOARCH)*, 2016, pp. 73–78.

[36] B. Grigorian and G. Reinman, "Dynamically adaptive and reliable approximate computing using light-weight error analysis," in *Proc. of NASA/ESA Conf. on Adaptive Hardware and Systems (AHS)*, 2014, pp. 248–255.

[37] M. A. Hanif, R. Hafiz, and M. Shafique, "Error resilience analysis for systematically employing approximate computing in convolutional neural networks," in *Proc. of Design, Automation & Test in Europe Conf. (DATE)*, 2018, pp. 913–916.

[38] F. Sampaio, M. Shafique, B. Zatt, S. Bampi, and J. Henkel, "Approximation-aware Multi-level Cells STT-RAM Cache Architecture," in *Proc. of Intl. Conf. on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*, 2015, pp. 79–88.

[39] M. T. Teimoori, M. A. Hanif, A. Ejlali, and M. Shafique, "AdAM: Adaptive approximation management for the non-volatile memory hierarchies," in *Proc. of Design, Automation Test in Europe Conf. (DATE)*, 2018, pp. 785–790.

[40] M. Shafique, F. Sampaio, B. Zatt, S. Bampi, and J. Henkel, "Resilience-driven STT-RAM cache architecture for approximate computing," in *Workshop on Approximate Computing (AC)*, 2015.

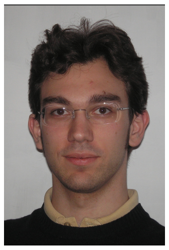
[41] A. Sampson, J. Nelson, K. Strauss, and L. Ceze, "Approximate Storage in Solid-State Memories," *ACM Trans. Comput. Syst.*, vol. 32, no. 3, pp. 9:1–9:23, 2014.

[42] M. Imani, S. Patil, and T. Rosing, "Low power data-aware STT-RAM based hybrid cache architecture," in *Proc. of Intl. Symp. on Quality Electronic Design (ISQED)*, 2016, pp. 88–94.

[43] A. Raha, S. Sutar, H. Jayakumar, and V. Raghunathan, "Quality Configurable Approximate DRAM," *IEEE Trans. on Computers*, vol. 66, no. 7, pp. 1172–1187, 2017.

Amir Mahdi Hosseini Monazzah received his Ph.D degree in computer engineering from Sharif University of Technology, Tehran, Iran, in 2017. He was a member of the Dependable Systems Laboratory from 2010 to 2017. As a Visiting Researcher, he was with the Embedded Systems Laboratory, University of California, Irvine, CA, USA from 2016 to 2017. As a postdoc fellow he was with the school of computer science, institute for research in fundamental sciences (IPM), Tehran, Iran from 2017 to 2019. He is currently a faculty member of the School of Computer Engineering, Iran University of Science and Technology (IUST), Tehran, Iran. His research interests include investigating the challenges of emerging nonvolatile memories, hybrid memory hierarchy design, and IoT applications.

Amir M. Rahmani received his Masters degree from Department of ECE, University of Tehran, Iran, in 2009 and Ph.D. degree from Department of IT, University of Turku, Finland, in 2012. He also received his MBA jointly from Turku School of Economics and European Institute of Innovation & Technology (EIT) ICT Labs, in 2014. He is currently Marie Curie Global Fellow at University of California Irvine (USA) and TU Wien (Austria). He is also an adjunct professor (Docent) in embedded parallel and distributed computing at the University of Turku, Finland. His research interests span Self-aware Computing, Energy-efficient Many-core Systems, Runtime Resource Management, Healthcare Internet of Things, and Fog/Edge Computing. He has served on a large number of technical program committees of international conferences, such as DATE, VLSID, GLSVLSI, DFT, ESTIMedia, CCNC, MobiHealth, and others, and guest editor for special issues in journals such as JPDC, FGCS, MONET, Sensors, Supercomputing, etc. He is the author of more than 170 peer-reviewed publications. He is a senior member of the IEEE.



Antonio Miele is an Assistant Professor at Politecnico di Milano since 2014. He holds a M.Sc. in Computer Engineering from Politecnico di Milano and a M.Sc. in Computer Science from the University of Illinois at Chicago. In 2010 he received a Ph.D. degree in Information Technology from Politecnico di Milano. His main research interests are related to the definition of design methodologies for embedded systems, in particular focusing fault tolerance and reliability issues, runtime resource management in heterogeneous multi-/many-core systems and FPGA-based systems design. He has served on a large number of technical program committees of international conferences, such as DATE, FPL, DFT, IOLTS, and others, and guest editor for special issues in IEEE Transactions in Emerging Topics in Computing and IET Computers and Digital Techniques journals. He is co-author of more than 70 peer-reviewed publications. He is a member of the IEEE.

Nikil Dutt received a Ph.D. in Computer Science from the University of Illinois at Urbana Champaign in 1989, and is currently a Chancellor's Professor at the University of California, Irvine, with academic appointments in the CS, EECS, and Cognitive Sciences departments. Professor Dutt's research interests are in embedded systems, electronic design automation (EDA), computer systems architecture and software, and brain-inspired architectures and computing. He received numerous best paper awards at conferences and is coauthor of 7 books. He previously served as Editor-in-Chief of ACM TODAES and as Associate Editor for ACM TECS and IEEE TVLSI. He has served on the steering, organizing, and program committees of several premier EDA and Embedded System Design conferences and workshops, and serves or has served on the advisory boards of ACM SIGBED, ACM SIGDA, ACM TECS and IEEE Embedded Systems Letters (ESL). He is a Fellow of the ACM, Fellow of the IEEE, and recipient of the IFIP Silver Core Award.