NOSTalgy: Near-Optimum Run-time STT-MRAM Quality-Energy Knob Management for Approximate Computing Applications

Arash Salahvarzi[†], Amir Mahdi Hosseini Monazzah[†‡], Mahdi Fazeli[*], and Kevin Skadron[§]

Abstract—The stochastic switching feature of Spin-Transfer Torque Magnetic RAM (STT-MRAM) provides an attractive knob to trade quality for energy consumption in approximate computing applications. Indeed, the quality of STT-MRAM functionalities (mainly write operation) is increased by consuming more energy to achieve a more stable write. On the other hand, in approximate computing applications, we do not need 100% quality for all of the data. Accordingly, in recent years, several approaches have been proposed to find a balance between output threshold quality and energy consumption in approximate computing applications employing STT-MRAM on-chip memories. While approximate computing application output qualities are highly affected by the fluctuations of the environmental-conditions or input variations, none of the previously proposed approaches have considered the effects of these fluctuations on the output quality. In this paper, we propose NOSTalgy, a closed-loop cross-layer approach to dynamically trade off the quality of STT-MRAM based cache memories for energy saving in approximate computing applications. NOSTalgy utilizes a feedback managed fine-grained cache-line-level actuation knobs with different levels of quality for individual write accesses. These knobs are adjusted with the support of the operating system and programmer at run-time. Our experimental results using a set of benchmarks show that NOSTalgy satisfies the output quality thresholds while delivering up to 52% energy savings with negligible performance and area overheads.

Index Terms—Time-variant Quality, Approximate Computing, Energy Consumption, STT-MRAM

1 Introduction

STT-MRAM is a promising memory technology that is believed to be a viable candidate for replacing Static RAM (SRAM) in on-chip memories [1], [2]. The main motivation of deploying STT-MRAMs in on-chip memories is to address the high energy consumption issue of SRAMs, which becomes more and more threatening with technology scaling trends in recent years [3]. While STT-MRAM benefits from negligible leakage power, it can impose high dynamic energy consumption if a large number of write operations are served by the on-chip memories.

In addition to high write power consumption, STT-MRAMs suffer from stochastic switching problems. A write operation in an STT-MRAM cell may fail by a probability which is a function of write current amplitude [4], [5]. The higher write current is applied, the lower the probability of write failure. However, applying high write current to STT-MRAM cells imposes noticeable energy consumption.

Several techniques have been proposed to address the dynamic energy consumption challenge of STT-MRAMs [6], [7], [8]. Some of these techniques have focused on using stochastic switching behavior of STT-MRAMs to provide a trade-off between energy consumption and output quality by lowering the write current in applications where some levels of quality loss are affordable. [9], [10], [11], [12].

There are several applications such as pattern recognition, machine learning, and image processing in which the precise results are not necessarily needed to provide useful services [13], [14]. For example, a low-resolution image encoding can be used in an application where high-quality images are not required. Approximate computing is an approach that relaxes the result accuracy to improve performance, power/energy consumption, and other design objectives in such applications. In approximate applications, based on fault consequences, data can be classified as critical and non-critical [15]. A faulty critical data item causes a mission failure, while a faulty non-critical data just leads to a quality loss in the output result. Non-critical data has an intrinsic fault tolerance attribute, as a fault occurrence does not result in a failure.

Considering such intrinsic fault tolerance in non-critical data structures of approximate applications, the stochastic switching behavior of STT-MRAM brings a unique opportunity to save energy consumption. In other words, for the write operations in non-critical data structures, we can decrease the write current (voltage) amplitude to save energy consumption with some degrees of quality loss [16], [17]. Generally, the previous studies that try to benefit from STT-MRAM energy-quality knob(s) in approximate computing applications can be categorized into two groups, i.e., *Compile-time* actuation knob(s) adjustment, and *Design-time* actuation knob(s) adjustment.

The studies in the first group introduce hardware/software approaches to trade quality of STT-MRAM caches or scratchpad memories for energy savings in the on-chip

* *Department of Computer Engineering, Boğaziçi University, Istanbul, Turkey.*
† *Department of Computer Engineering, Iran University of Science and Technology, Tehran, Iran.*
‡ *School of Computer Science, Institute for Research in Fundamental Sciences (IPM).*
§ *Department of Computer Science, University of Virginia, Charlottesville.*
E-mails: arash_salahvarzi@alumni.iust.ac.ir, monazzah@iust.ac.ir, mahdi.fazeli@boun.edu.tr, skadron@virginia.edu.

Manuscript received November 19, 2019

memory hierarchy of multi- and many-core systems running approximate applications [11], [12]. These studies enable the programmers or designers to adjust the STT-MRAM energy-quality knobs in the source codes of programs.

Unlike the studies in the first group that benefit from on-demand actuation knobs management, the studies in the second group utilize a static actuation knobs adjustment that affects the physical structure of either STT-MRAM cells or on-chip memories [18], [19], [20], [21]. While the environmental-condition fluctuations and input variations noticeably affect the output quality of approximate computing applications, *Run-time* (closed-loop) actuation knobs adjustment approaches have not been addressed carefully. For example, considering computer vision applications that operate in outdoor environments, the external environmental-condition fluctuations like ambient light variations affect the necessary output precision. Likewise, for applications that operate in indoor environments, the internal environmental-condition fluctuations such as operating temperature may affect the necessary output precision.

In this paper, first we show how environmental-condition fluctuations and input variations may affect the output quality of an approximate computing application. Then, we will introduce our run-time STT-MRAM energy-quality knob adjustment approach, so-called NOSTalgy. NOSTalgy is a closed-loop, cross-layer approach that simultaneously utilizes the information provided by *programmers*, *sensors* and *operating system supervisory programs* to adjust the actuation knob(s) at the run-time and trading quality for energy consumption in STT-MRAM caches. Please note that NOSTalgy can also be applied with any memory technology that provides cache-line-level voltage (current) actuation knobs. Accordingly, considering the maturity of current memory technologies and other issues associated with replacing SRAMs with other memory technologies for on-chip memories, STT-MRAM is the most appropriate technology. Thus, we decided to devote our analysis of NOSTalgy design to STT-MRAM technology.

Programmers are responsible for determining the critical and non-critical data structures and their corresponding output quality thresholds in the approximate computing applications. This information can be fed to the NOSTalgy hardware through provided *Application Program Interfaces (APIs)*. Then, the designed NOSTalgy supervisory program checks the output quality of approximate computing applications based on an initiated interrupt (generated as the environmental-condition fluctuation is detected by a sensor or through a countdown timer that reaches a predetermined value) and tries to provide a constant output quality level considering the quality thresholds. Accordingly, unlike the static and on-demand approaches that provide open-loop actuation knob management, the closed-loop approach proposed in NOSTalgy provides the opportunity to get feedback from the environment and respond to the environmental conditions fluctuations by tuning the STT-MRAM write current knob. To the best of our knowledge, this is the first work that has proposed a closed-loop and cross-layer approach to manage quality-energy knobs in STT-MRAM based cache memories for approximate applications.

To evaluate NOSTalgy, we employ the gem5 cycle-accurate simulator [22] using a set of benchmarks from the *computer vision*, *financial analysis*, *security*, and *automotive* domains [23], [24], [25]. Our simulation results show up to 52% energy consumption improvement.

The remainder of this paper is organized as follows: In Section 2, the background of STT-MRAM is presented. The most related previous studies are explored in Section 3. In Section 4, the motivation of this study is introduced. Afterward, in Section 5, NOSTalgy is explained, followed by a comprehensive experimental study in Section 6. Finally, we conclude this paper in Section 7.

2 Preliminaries on STT-MRAM

In this section, first we review some preliminaries on the structure of STT-MRAMs. Then, we will explain how STT-MRAMs introduce unique quality-energy knobs and the way that we can use these knobs on approximate computing applications.

2.1 STT-MRAM Fundamentals

Fig. 1 depicts the structure of an STT-MRAM bit-cell which consists of an access transistor and a *Magnetic Tunnel Junction* (MTJ). The MTJ is composed of a reference layer and a free layer separated by a tunneling oxide. The free layer can be programmed to change its magnetic field direction while the reference layer has a fixed magnetic field direction. The resistance of an MTJ is determined by the relative magnetic field directions of the reference layer and the free layer. When they are in parallel, the MTJ is in a low resistance state and represents logic "0"; when they are anti-parallel, the MTJ is in a high resistance state and denotes logic "1". ($P \rightarrow AP$ and $AP \rightarrow P$ show changing the direction of magnetic fields from Parallel to Anti-parallel and Anti-Parallel to Parallel, respectively).

During the write operation, a current greater than the critical switching current is passed through the bit-cell. Writing "0" or "1" is determined by the direction of the applied switching current between the Bit Line (BL) and Source Line (SL), while the Word Line (WL) is high (as depicted in Fig. 1). On the other hand, there are two ways to read a cell, the so-called parallel (P) direction read with applying read current at the same direction as writing "0" and the anti-parallel (AP) read with applying read current at the same direction as writing "1". In the parallel direction reading, a low voltage is applied between BL and SL. After activating WL, a current flows from BL to SL. In the anti-parallel direction reading, the voltage polarity applied to BL and SL is switched and a current flows in the reversed direction (from SL to BL). This current is compared against a reference value to determine whether the logic state is high or low (as depicted in Fig. 1). If the current is higher (lower) than the reference, the MTJ is in the low (high) resistance state, and hence the read value is a "0" ("1").

2.2 STT-MRAM Quality-Energy Knob(s)

One of the main properties of STT-MRAM is its stochastic switching. The stochastic switching in STT-MRAM may lead to failure during write operations. The probability of write failure in STT-MRAM can be calculated by Equation (1) [4]:

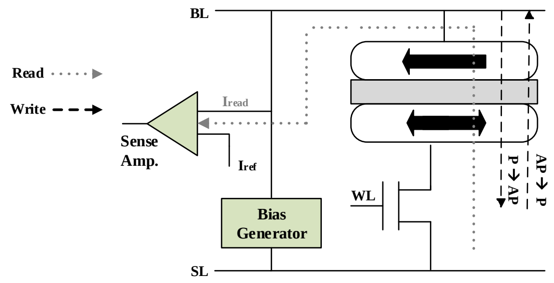


Fig. 1: Typical STT-MRAM bit-cell structure.

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

where Δ is the thermal stability factor, I_{C_0} is the critical MTJ switching current at 0°K, c is the Euler constant, e is the magnitude of electron charge, m denotes the magnetic momentum of the free layer, p is the tunneling spin polarization, μ_B is the Bohr magneton, I_w is the write current, and t_w is the write pulse width.

Whereas STT-MRAM write failure is one of the most important challenges in the deployment of STT-MRAM memory technology [1], [26], it can provide a unique opportunity to save energy consumption in STT-MRAM's energy-hungry write operations for approximate computing applications. Indeed, many approximate computing applications such as image processing can deal with erroneous data in some parts of their source codes. For these data having an intrinsic fault tolerance, we can utilize the I_w (in Equation (1)) as a quality-energy knob to pay less energy during STT-MRAM write operation. This parameter can also be changed by modifying the STT-MRAM applied voltage.

One of the important advantages of this knob over traditional knobs introduced for other memory technologies like SRAM, is its fine-granularity [11]. For example, if we consider an STT-MRAM cache utilizing I_w knob and running an approximate computing application, we can have two approximate and non-approximate data blocks next to each other simultaneously, while in the same SRAM cache utilizing traditional voltage scaling knobs for an approximate computing application, these two blocks can't be mapped next to each other (at the same memory bank) due to different quality requirements.

3 Related Work

Closed-loop quality management has been actively explored by prior studies [27], [28], [29]. Those approaches mostly focused on the software side of the design space [27], [28]. On the other hand, the efficiency of output quality managements mentioned in previous closed-loop approaches mostly depends on the type of actuation knobs in the hardware, so directly depends on the technology, which was not covered in a cross-layer manner in previous studies. A number of studies adopt approximate computing approaches to improve memory system energy efficiency. Among different memory technologies, some works focused on deploying approximate computing in SRAM [30] or DRAM memory [31], [29]. With the recent development of STT-MRAMs, several studies try to benefit from the promising characteristics of STT-MRAM in approximate computing applications. The previous studies in this area can be categorized into two groups: on-demand actuation knob adjustment, and static actuation knob adjustment. Management of the actuation knobs in the first group approaches are flexible and performed during the execution based on the provided information by the programmers [11], [12], [32], [33], [34].

Considering the first group, [11] and [12] introduced hardware/software approaches to trade quality of STT-MRAM caches and scratchpad memories (respectively) for energy savings in the on-chip memory hierarchy of multi- and many-core systems running approximate applications. The proposed approaches in [11] and [12] benefit from fine-grained block-level actuation knobs with different levels of quality for individual read and write operations. In [33], authors explored a combination of different approximation techniques at the circuit and architecture levels that yielded significant energy benefits for small possibilities of errors in reads, writes, and retention. In [32], the authors modeled a system including STT-RAM scratchpad and PCM main memory with different approximation knobs to synergistically trade data accuracy for both STT-RAM access delay and PCM lifetime by solving an integer linear programming (ILP) problem at design-time. In [34], the authors proposed utilizing formal control theory to control the quality of streaming approximate applications running on a system with quality-configurable memory by tuning memory quality knob(s). The authors in [35] designed an approach that decouples error analysis of the approximate accelerator from quality analysis of the overall application and dynamically guaranteed the worst-case error for an application, as well as gained additional coverage from leveraging slack. In [36], the authors described an architecture with a hierarchical floating-point unit (FPU) that leveraged dynamic precision reduction to enable efficient FPU sharing among multiple cores. This sharing reduced the area required by the cores, thereby allowing more cores to be packed into a given area and exploiting more parallelism. The authors in [37] proposed an approximation-aware MLC STT-RAM cache architecture that was partially-protected to restrict the reliability overhead. It leveraged the knowledge of variable resilience properties of different applications (and even different functions within an application) to adaptively reduce the protection overhead under a given error tolerance level.

Unlike the studies in the first group that benefit from on-demand actuation knob management, the studies in the second group utilize a static actuation knob adjustment during the design time. These approaches propose a hybrid architecture in their design and manage the actuation knobs statically for each part of their architecture [18], [19], [20], [21], [38].

For example, In [18] and [19], the authors proposed a hybrid cache architecture using different STT-MRAM cells with various *thermal stability factors*. They benefited from approximate computing feature to tolerate the increased retention failure rate caused by the relaxed thermal stability factor of STT-MRAM cells. In [20], the authors proposed an approximation-aware Multi-Level Cell (MLC) STT-MRAM cache architecture to trade quality with large capacity delivered by MLC STT-MRAM cells. The authors in [21] proposed a progressive scaling scheme for STT-RAM arrays, in which the power consumption was reduced at the expense of small quality degradation. In [38], the authors proposed Quora; an energy efficient, quality programmable vector processor that utilized a 3-tiered hierarchy of processing elements. It provided distinctly different energy vs. quality trade-offs and used hardware mechanisms based on precision scaling with error monitoring and compensation to facilitate quality programmable execution. The authors in [39] proposed SoftPCM to enhance both energy efficiency and lifetime of PCM. It utilized the error tolerance characteristic of video applications to relax the accuracy of write operations.

The main contribution of NOSTalgy over the previous studies is that, unlike the previous attempts that proposed open-loop approaches, it provides a closed-loop cross-layer approach. In the other words, NOSTalgy not only benefits from on-demand actuation-knob adjustment but, with the aid of the information provided by *programmers*, *sensors*, and *operating system supervisory programs*, it dynamically tracks the environmental-condition fluctuations and input variations to adjust the actuation knob at the run-time. This leads to an efficient trade of quality for energy consumption in STT-MRAM caches.

4 Motivation

The quality management in most of the previous studies that utilize the STT-MRAM quality-energy knobs for approximate computing applications is done through programmer hints [11], [12]. During compile time, these hints are hard-coded in programs based on the output quality requirement that should be provided in each application. This is done by conducting offline-profiling in which the STT-MRAM knob(s) is (are) manipulated for approximate data and the output quality is validated considering the quality threshold. Finally, the knob(s) configuration that delivers the least but still satisfactory quality is selected and hard-coded in the programs to save energy consumption.

Let's call the above-mentioned knob quality management as *static quality management*, since everything is done during compile time. We believe this kind of knob management is not efficient, since the output quality of approximate computing applications is directly affected by variations on the inputs and fluctuations in environmental conditions which are not predictable during the design/compile time.

In the following, we will see how variations on inputs and fluctuations in environmental conditions affect the output quality in a case study approximate computing application. To this end, we use `sobel` filter from Axbench [24] benchmark suite as our case study application. `Sobel` filter is utilized in image processing and computer vision, especially in edge detection algorithms, where it creates an image emphasizing edges. We consider two quality levels for write operations in the L2 cache of a system running this application. One of the quality levels used in L2 cache write operations provides full accurate (golden) outputs, whereas the other one decreases the quality to save energy

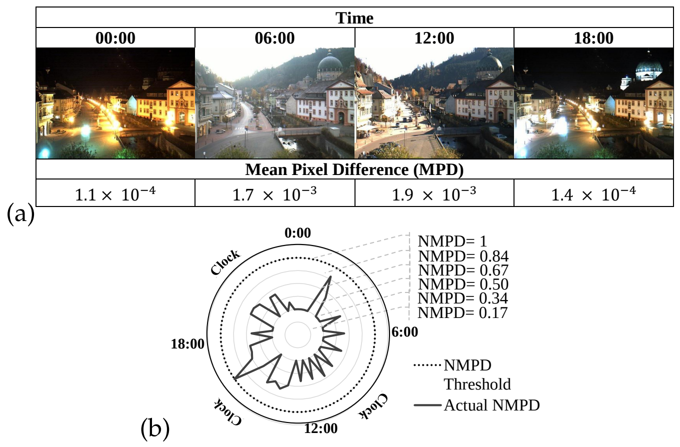


Fig. 2: Output qualities of `sobel` filter for input images taken from a city camera (with a constant view angle): (a) Absolute MPD at different times of a day, and (b) normalized output quality (NMPD) fluctuations in a day.

consumption [1]. We consider *Mean Pixel Difference (MPD)* as our quality metric for the examples. MPD is a metric that calculates the difference between two images by finding the difference between corresponding pixels.

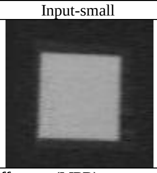
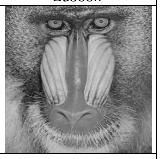
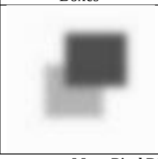
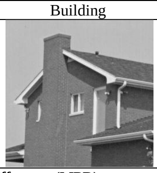
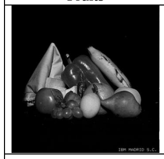
In the following, first we consider pictures from a constant point of view at different times of the day to focus on the effects of the environmental-condition fluctuations. Then, we will focus on the effects of different input images on the output qualities of the application.

4.1 Environmental-Condition Fluctuations.

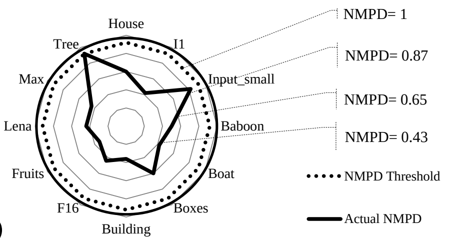
Fig. 2 depicts output qualities of `sobel` filter, for input images taken from a city camera (with a constant view angle) at different times of a day [40]. As can be seen in Fig. 2.(a) MPD of the output images varies at different times of the day, comparing the golden (fully accurate) outputs with approximate outputs at each time. Accordingly, we run the `sobel` filter at each time twice, one at full accurate configuration and one at approximate configuration. It is noteworthy that the write operation quality level (QL3) of the L2 cache in approximate runs is kept unmodified during the day. Fig. 2.(a) verifies that the vulnerability of the output quality against the occurred write failures due to decreasing the write current in the L2 cache is affected by the variations on the environmental conditions (i.e., changing sunlight during a day). Please note that most parts of the pictures that were taken at different times of the day in this experiment are constant and the main reason for observing different output qualities at different times was changing sunlight during a day.

Fig. 2.(b) shows the normalized fluctuations of the output qualities during a day for `sobel` filter when we compare the approximate outputs with golden ones at each time interval. In the previous studies that use programmer hints for adjusting the quality level of outputs, programmers set

1. The qualities reported in Fig. 2 are obtained by simulating a NOSTalgy-enabled L2 cache in gem5 [22], using simulation parameters corresponds to QL0 and QL3 quality levels that will be presented in Section 6.

House	I1	Input-small	Baboon
			
Mean Pixel Difference (MPD)			
3×10^{-3}	2.2×10^{-3}	4.1×10^{-3}	2.5×10^{-3}
Boat	Boxes	Building	F16
			
Mean Pixel Difference (MPD)			
2.1×10^{-3}	3×10^{-3}	1.8×10^{-3}	2.3×10^{-3}
Fruits	Lena	Max	Tree
			
Mean Pixel Difference (MPD)			
1.7×10^{-3}	2.2×10^{-3}	2.3×10^{-3}	4.7×10^{-3}

(a)



(b)

Fig. 3: Output qualities of `sobel` filter for different pictures.

the quality-energy knob(s) so that the quality threshold of the output image is satisfied considering the worst-case environmental situation (16:30 in this example). Accordingly, in our case study example, the programmers should consider the outer ring (*Normalized Mean Pixel Difference (NMPD)* threshold ring in Fig. 2.(b)) and configure the quality-energy knob(s) statically for all of the situations. For the sake of clarity in Fig. 2.(b), we normalize the MPD of images to the MPD of the image taken at 16:30.

Considering this static knob(s) adjustment, it can be seen in Fig. 2.(b) that we lose a lot of opportunities for saving energy consumption in the situations that the environmental conditions or image inputs lead to less vulnerability of `sobel` filter output against cache write failures. Indeed, in these conditions (inner rings in Fig 2.(b)) we could save much more energy consumption by re-configuring the quality-energy knob(s) to utilize the gap between the actual MPD and predefined MPD threshold. Accordingly, Fig. 2.(b) illustrates the important possible opportunities for energy saving by considering some dynamic quality-energy knobs adjustment methods like NOSTalgy.

4.2 Input Variations

Besides the environmental conditions fluctuations, input variations also affect the output qualities of the applications. To show the effects of the input variations, we fed 12 different images to the `sobel` filter.

Fig. 3 depicts output qualities of `sobel` filter, for the mentioned 12 images. As can be seen in Fig. 3.(a), MPD of the output images varies by varying the input images, comparing the golden (fully accurate) outputs with approximate outputs for each input. Again, it is noteworthy that the write operation quality level (QL3) of the L2 cache in approximate runs is kept unmodified between the inputs. Fig. 3.(a) verifies that while the environmental conditions fluctuations affect the output qualities of the applications more severely, still input variations noticeably affect the output qualities.

Fig. 3.(b) shows the normalized fluctuations of the output qualities of different input images for `sobel` filter when we compare the approximate outputs with golden ones. Here, again we can see that, unlike the previous studies that use only programmer hints for adjusting the quality level of outputs, closed-loop dynamic approaches like NOSTalgy can find good opportunities to save energy consumption. Indeed, considering the NMPD threshold ring depicted in Fig. 3.(b) for `sobel` output quality, the static knob-management approaches embed fixed run-time quality levels in the code. On the other hand, during run-time, those embedded running quality levels are used for all the inputs, regardless of whether those settings are appropriate for all inputs. Instead, as we can see in Fig. 3.(b) for most of the images, the running quality level can be changed so we can save the energy consumption while meeting the quality threshold.

5 NOSTALGY IN DETAIL

In this section, we will explain how run-time quality-energy knob adjustment can be achieved by NOSTalgy. To this end, NOSTalgy introduces a cross-layer approach that simultaneously utilizes the information provided by programmers and operating system supervisory programs to adjust actuation knobs at run-time and trade quality for energy consumption in STT-MRAM caches. Accordingly, NOSTalgy requires collaboration between software parts and hardware parts of the system.

Fig. 4 depicts the abstract view of the NOSTalgy approach. Considering the software part of NOSTalgy, the first challenge is to provide a facility for programmers to retrieve the quality thresholds of different parts of applications during the compile time. This challenge will be addressed in the sub-section 5.1. In NOSTalgy, the operating system takes the responsibility of tracking the environmental-condition fluctuations and input variations and manages the STT-MRAM actuation knobs during the run-time of applications. We will focus on the required NOSTalgy's operating system modifications in sub-section 5.2. Finally, NOSTalgy requires a set of hardware facilities that will be discussed in sub-section 5.3.

5.1 Program Support

The data of an application can be classified into two categories: critical and non-critical. Critical data refers to data where a corruption leads to failure of the application (such as loop control variables), while corruption in non-critical data will result in quality-reduction that will be acceptable in approximate computing applications (such as variables storing elaborated data). In general, non-critical data structures can have different granularities, from a *double*-sized

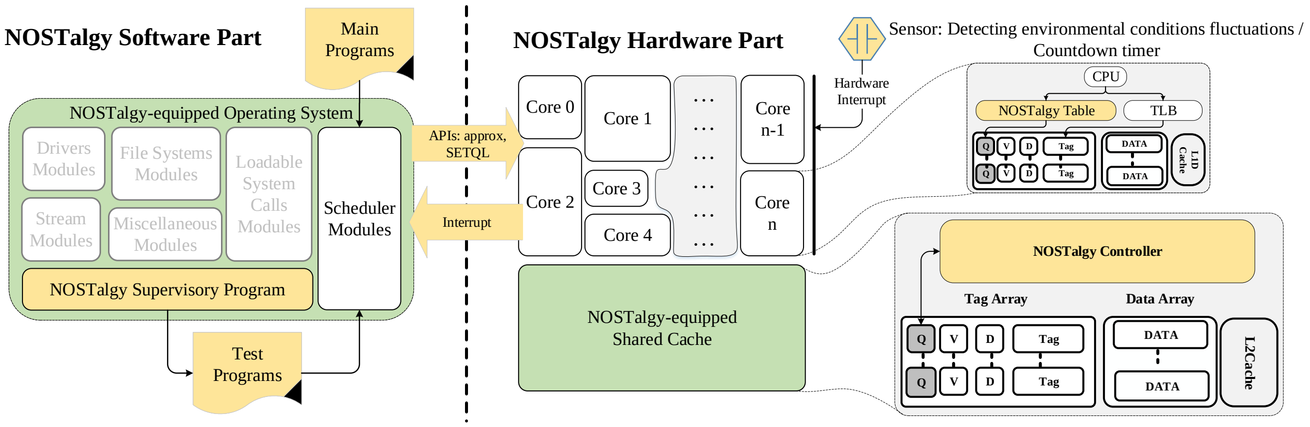


Fig. 4: An abstract view of different parts of a system equipped with NOSTalgy approach.

variable that stores the result of a calculation to a huge-sized array that keeps the information of ultra-high quality pictures. The classification of the data structures to critical and non-critical groups in the programs is typically performed by a set of fault injections at the design phase of the applications. The results of fault injections then are compared with the golden results to label the critical and non-critical data structures.

In approximate computing applications, programmers can determine whether a data block is critical or non-critical, since they are aware of the functionality of different parts of the application, as well as the importance of data. To assist the programmer in identifying the critical and non-critical data structures, some frameworks like Rely [41] and Accept [42] could also be applied. In this paper we focus more on the architecture and its dependent software modules of NOSTalgy. In the future work we will focus on developing a companion methodology for the classification of the critical and non-critical data structures. More information on the classification of program data structures to critical and non-critical segments can be found in [13].

After determining the criticality of different parts of an application, programmers should annotate these parts in the source code of the application. Accordingly, NOSTalgy introduces two APIs: `approximate` and `set_QL` (QL is quality level) Table 1 shows their formats. `begin` and `end` are the start address and the end address of approximate data in the memory, respectively. `thr.QL` is the minimum quality required to run that approximate data. We can implement the introduced APIs in NOSTalgy either by modifying the ISA of the processor or utilizing a memory-mapped IO method [11]. It is worth mentioning that the latter approach requires fewer hardware modifications, and it is more flexible because NOSTalgy's APIs can be implemented within a special run-time library. Therefore, the hardware components of NOSTalgy become memory-mapped interfaces. The run-time library uses normal read/write instructions to transfer the information provided by the API calls to the hardware.

Programmers can use `approximate (begin, end, thr.ql)` to introduce the non-critical data region to NOSTalgy's hardware. By inserting an `approximate (begin, end, thr.ql)` command at the beginning of the application source code, NOSTalgy's hardware only adjusts the quality-energy knobs for approximate parts of application. Whereas `approximate (begin, end, thr.ql)` provides the static information for quality-energy knobs adjustment, `Set_QL (begin, end, ql)` will be used by the OS to dynamically adjust the quality-energy knobs based on the environmental conditions during the run-time of application which will be discussed in the next sub-section.

TABLE 1: NOSTalgy's APIs

Method	Parameters	Note
`approximate`	`Begin`	Start address of approx. memory region
	`End`	End address of approx. memory region
	`Thr.QL`	Threshold quality level
`set_QL`	`Begin`	Start address of approx. memory region
	`End`	End address of approx. memory region
	`QL`	On-line measured quality level

5.2 Operating System Support

As we saw in Section 4, the environmental-condition fluctuations or input variations may considerably affect the output quality of the applications. In other words, for these applications, the fault-tolerance of the data structures that keep the information of the inputs tightly depend on the environmental conditions or the contents of the inputs. For example, there is a meaningful difference between the fault-tolerance of the image's data structure of Fig. 2 at day and night. Indeed, considering the same number of fault injections in an image's data structure, the MPDs of the images at day are considerably higher than the MPDs of the images at night. This verifies that some data structures in approximate computing applications are tightly dependent on input variations or environmental-condition fluctuations. While the programmer can decide about the criticality or non-criticality of a data structure at the design-time, she/he can not efficiently estimate the effects of environmental-condition fluctuations or input variations during run-time. This is the exact place that NOSTalgy can help.

In NOSTalgy, the programmer is responsible to prepare two versions of their program, i.e., the main program, and test program. During the normal situation, the scheduler runs the main program on the NOSTalgy hardware. Whenever any environmental condition fluctuation or input varia-

Algorithm 1: NOSTalgy supervisory program

```
Input: Test Programs of Application$_{(i \sim n)}$, Applications' CPU Core ID, Applications' Quality Metric Thresholds
Result: Updating the NOSTalgy Tables' QLs
for Test Program($i \sim n$) do
    $Quality\_Metric = TestProgram_i(QL1)$;
    if $Quality\_Metric >= Quality\_Metric\_threshold_i$ then
        $Quality\_Metric = TestProgram_i(QL2)$;
        if $Quality\_Metric >= Quality\_Metric\_threshold_i$ then
            $Quality\_Metric = TestProgram_i(QL3)$;
            if $Quality\_Metric >= Quality\_Metric\_threshold_i$ then
                for All the NOSTalgy Table's Application$_i$.core$_{id}$ entries do
                    set_QL(entry.begin, entry.end, QL3);
                end
                exit;
            else
                for All the NOSTalgy Table's Application$_i$.core$_{id}$ entries do
                    set_QL(entry.begin, entry.end, QL2);
                end
                exit;
            end
        else
            for All the NOSTalgy Table's Application$_i$.core$_{id}$ entries do
                set_QL(entry.begin, entry.end, QL1);
            end
            exit;
        end
    else
        for All the NOSTalgy Table's Application$_i$.core$_{id}$ entry do
            set_QL(entry.begin, entry.end, QL0);
        end
        exit;
    end
end
```

tion are detected by the OS (through hardware interrupt initiated by the sensor or countdown timer), the light-weight test program will be run by the OS (in parallel with the main program) to measure the effects of the new conditions on the output quality of the application and to update the application's run-time quality levels in NOSTalgy's table. The test programs will be fed to the OS scheduler by a NOSTalgy Supervisory Program (depicted in Fig. 4) as the hardware interrupt is detected by the OS.

Algorithm 1 shows an abstract design of a supervisory program considered to dynamically adjust the quality-energy knobs. Here we assume the hardware platform that is equipped with NOSTalgy supports four quality levels (QL0~QL3) for running the applications. To find the optimum quality levels of approximate data in each approximate computing application running in different cores of a NOSTalgy-enabled platform, Algorithm 1 is called after the aforementioned hardware interrupt is raised (due to environmental-condition fluctuations or input variations) and executes the test programs of all approximate computing applications that are currently running on the CPU(s). To avoid frequent executions of the test programs in situations that the environmental conditions or inputs change frequently, we will consider a timer and sense the environment only in a periodic manner. Based on the environmental situation that a NOSTalgy-enabled system would be used, the developer her/him self may tune this timer.

```
double main(Quality_Level){
unsigned *small_approximate_data;
unsigned *approx_result;
double output_quality;
small_approximate_data = (unsigned int*)malloc(input_size
     *sizeof(unsigned));
approx_result = (unsigned int*)malloc(output_size*sizeof(
     unsigned));
approximate(small_approximate_data,input_size*sizeof(
     unsigned),Quality_Level);
approx_result = function(small_approximate_data);
output_quality = measure_quality(approx_result,
     golden_result);
return output_quality;
}
```

Fig. 5: A typical test program with its necessary modules.

Considering Algorithm 1, each test program of an application is run with different QLs and the output quality levels that correspond with the passed QLs to the test programs are checked with the acceptable *Quality_Metric_Threshold* of the target application. In this way, Algorithm 1 finds the most energy-efficient QL that satisfies the output quality level of the approximate computing applications in the new environmental conditions. When the application's optimum QL is found in Algorithm 1, the NOSTalgy's `Set_QL()` API is used to update the QL entries in target core NOSTalgy's table. Please note that, here in Algorithm 1, we just made a simple example to show how we could design a typical supervisory program. For the applications that require different QLs for different data structures, those scenarios can be implemented in the supervisory program with the aid of changing the contents of the inner for loops to determine different QLs for different data structures using *set_QL* API. Furthermore, all of the previous software algorithms to measure the output quality of approximate computing applications can be applied as NOSTalgy supervisory programs.

The typical modules of a test program are depicted in Fig. 5. The test program of each application should include a light-weight function that could mimic the effects of the actuation knob's quality levels on the output quality of the application (*function()* in Fig. 5). This function works on a small set of data structures that will be affected by the input QL of the test program (*small_approximate_data*). Before running the function, the test program utilizes the NOSTalgy's `approximate` API to set the quality level of the approximate data by the input QL. Furthermore, the test program should measure the output quality and pass it to NOSTalgy's supervisory program for further decisions on finding the optimum QL. To this end, the test program should utilize a module that measures the output quality in the current QL (*measure_quality* in Fig. 5). This module compares the approximate result with the golden result.

5.3 Hardware Support

In the hardware part of Fig. 4, we can see the required modules in hardware to equip a computer system with the NOSTalgy approach. To this end, two modules are added to typical architecture of a multi-core system (homogeneous or heterogeneous). The added modules are: *NOSTalgy table*, and *NOSTalgy controller*. We will explain the details of each module in the following sections.

NOSTalgy table: This module provides a space for saving the quality levels of approximate data blocks. In other words, the quality level information provided by the programmers (at the start of execution) or by the NOSTalgy supervisory program (during the run-time) is saved in this space. This table has three fields for each entry, i.e., start address of approximate region, end address of approximate region, and the threshold quality level required for serving this region. Besides NOSTalgy's `approximate` API that directly fills the mentioned three fields, NOSTalgy's `Set_QL (begin, end, ql)` also can update the quality level field of this table (The `begin` and `end` operands in NOSTalgy's `Set_QL` API are used to identify the data entry in the NOSTalgy quality table that its quality level should be updated during the run-time).

NOSTalgy controller: To configure the quality-energy knobs in NOSTalgy, this controller adjusts the shared cache write voltage to serve the write request of each data block based on its quality level that was retrieved from the NOSTalgy quality table. Please note that at QL0 (the accurate mode-used in critical data structures) we just pass the standard voltage value and for the other QLs (the approximated mode-used in non-critical data structures) we pass portions of this value to the memory cells. Generally, NOSTalgy provides a closed-loop approach to manage the actuation knob of STT-MRAMs. Accordingly, while NOSTalgy's table acts as the message passing center (storing the QL information received from software side), NOSTalgy's controller is the hardware module to actuate the STT-MRAMs' knob based on NOSTalgy's table information. Fig. 6 depicts the proposed design for NOSTalgy controller. As can be seen in Fig. 6, NOSTalgy controller utilizes a set of voltage shifters (VSes) to actuate the STT-MRAM knob. Several studies used the voltage level shifter in their designs [12], [43], [44]. In particular, to have m QLs, the NOSTalgy controller includes $(m - 1)$ VSes. Indeed no VS is required for the transition at highest QL since it is performed at nominal voltage. When a request to update a block arrives, the QL information ($\log_2 m$ bits) received by the NOSTalgy controller is used to select the appropriate voltage (SV_{dd}) for an incoming write operation and to drive the SV_{dd} rail to use in bit-lines if the incoming request is write. It should be mentioned that voltage shifter switching should be fast enough to keep the performance of the system intact between two consequent write accesses that utilize different voltage levels.

To detect the environmental-condition fluctuations or input variations, NOSTalgy exploits a sensor or countdown timer as depicted in Fig. 4. Different sensors can be applied here. For example, external sensors that measuring the environmental light can be used as a NOSTalgy sensor. For input variation detection we can benefit from a countdown timer which, for every time epoch, generates an interrupt to run the supervisory programs for the current input set. We considered this mechanism in NOSTalgy with the hope of tracking the input variations during the run-time.

In the following, we will explain the different scenarios for a write operation in a NOSTalgy-enabled shared L2 cache. It is noteworthy that NOSTalgy doesn't affect the L2 cache reading operation. The write operation in L2 cache occurs due to one of the following events: 1) Filling operation from upper-side memories (like main memory or L3 cache) or 2) Write-back from L1 cache.

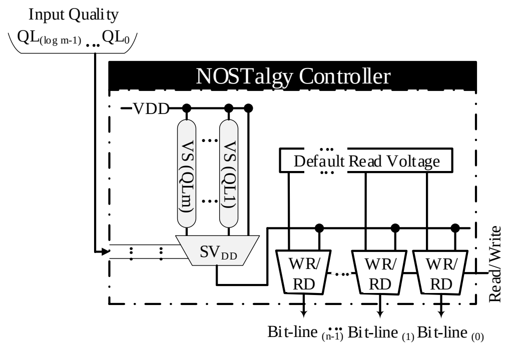


Fig. 6: NOSTalgy controller design for a cache line width of n bits (Required changes comparing with baseline cache controller are highlighted in gray).

■ **Filling Operation:** Whenever an L1 cache request is missed in the L2 cache, cache filling in L2 is required. The Quality Level (QL) of the requested missed block in the L2 cache is passed on the bus after searching the NOSTalgy quality table, which is performed in parallel with the TLB address search (depicted in Fig. 4). Besides passing the QL on the quality bus, this QL is also written in the 2-bit provided space of the requested block at the L1 tag memory (depicted in Fig. 4). The NOSTalgy controller voltage level shifter in the L2 cache will use this QL to fill the L2 cache line when the request is served by the upper-level cache(s) or memory.

■ **Write-back from L1 cache:** In two cases write-back from L1 cache is required: 1) a new block should replace a dirty one, or 2) the last modified version of the block in the L1 cache should be written to the L2 cache. Write-back form L1 data cache to L2 cache is done as follow:

1) The QL of the target block for eviction from L1 cache is passed on the QL bus by reading the corresponding QL bits from the L1 tag memory. This is done in parallel with passing the address and data of this block on the address and data buses, respectively. Indeed, when the tag search operations to find the target block finish, address, data, and QL bits are read in parallel from the corresponding fields of the target data and the values pass to the address bus, data bus, and quality bus, simultaneously.
2) NOSTalgy controller in L2 cache retrieves the QL of the requested block for write-back from the QL bus and voltage shifter adjusts the write voltage for serving this request in the L2 cache.

6 Simulation System Setup and Results

In this section, first we introduce the system setup used to explore the efficiency of NOSTalgy. Then, we will explore the experimental results of this study.

TABLE 2: gem5 settings

Parameter	Value	Parameter	Value
ISA	ARMv7-A	L1 Size,Assoc	64KB ,4
No. of Cores	8	L2 Size,Assoc	1MB,16
Cache Config	L1 (Private) L2 (Shared, NOSTalgy-enabled)	Cache Block Size	64B
L1 Write Latency	2 Cycle	L2 Write Latency	28 Cycle
L1 Read Latency	2 Cycle	L2 Read Latency	20 Cycle

TABLE 3: Quality-energy transducer map for 1MB NOSTalgy-enabled STT-MRAM cache

Quality Level	Read Energy	Write Error Rate	Write Energy
QL0 (Baseline)	0.146 nJ	Protected	10.755 nJ
QL1	0.146 nJ	5×10^{-5}	7.059 nJ
QL2	0.146 nJ	1×10^{-4}	6.386 nJ
QL3	0.146 nJ	9×10^{-4}	5.378 nJ

6.1 Simulation System Setup

We applied NOSTalgy to a shared L2 cache of an 8-core system simulated in gem5 [22]. It is worth mentioning that while there is no limitation in NOSTalgy design to be applied to any cache level in memory hierarchy, we decided to apply it to L2 cache. Indeed, various past studies on STT-MRAM adoption in on-chip memories, such as cache and SPM (Scratchpad Memories); claimed that utilizing STT-MRAM is only practical for L2 and lower-level on-chip memories [45], [46]. This is because STT-MRAM has limitations on performance, dynamic energy consumption, and endurance that make it more compatible with L2 and lower levels. Accordingly, utilizing STT-MRAM for L1 caches that are frequently accessed by CPUs is not feasible.

The details of the simulation environment are shown in Table 2. NVSim [47] is used to retrieve the latencies, energy consumption, and area of a 1MB STT-MRAM based L2 cache. Finally, to evaluate the area and power consumption overheads of the peripheral circuits in NOSTalgy controller, we used Synopsys Design Compiler® with 45nm technology library. We randomly injected faults into different cache blocks during the write operations, with uniform distribution over different bit positions. We used bit error rate (BER) and access current data from [12]. We defined 4 QLs, with increasing approximation levels from 0 to 3; the details of error rates and energy consumption are reported in Table 3. In particular, we considered no write error for write operations in full-accurate mode, that is QL0, and we consequently retrieved from [12] the corresponding write current amplitude, and subsequent energy consumption from the NVSim profiling. Energy consumption reported in Table 3 is evaluated for a 64-byte cache line. We should mention that in all simulations, the write operations included a fixed number of cycles and we only changed the current of write operations to save energy at the expense of some write accuracy. 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 5.1, it is left as a future work.

To measure the efficiency of NOSTalgy in dealing with approximate computing applications for saving energy consumption, we use a set of benchmarks from the *computer vision*, *Convolutional Neural Network* (CNN), and *financial* domains [23], [25], [24]. Furthermore, to show the generality of NOSTalgy and possibility of running the full-accurate programs next to approximate programs in a NOSTalgy-enabled system, we considered two full-accurate benchmarks. Table 4 introduces these benchmarks and the quality metrics of them (when applicable). We run these benchmarks simultaneously in our multi-core NOSTalgy-equipped platform. Furthermore, we suppose a quality threshold for approximation benchmarks so we can decide about the run-time quality levels in our NOSTalgy-enabled L2 cache. It is noteworthy that our empirically-chosen criteria to set the quality thresholds for each benchmark is based on the user-level QoS desirability.

TABLE 4: List of benchmarks

Benchmark	Domain	Quality Metric	Quality Threshold
corner detection	Approximate Computer Vision	Mean Pixel Difference (MPD)	0.12
edge detection	Approximate Computer Vision	Mean Pixel Difference (MPD)	0.004
image smoothing	Approximate Computer Vision	Peak Signal-to-Noise Ratio (PSNR)	38
sobel filter	Approximate Computer Vision	Mean Pixel Difference (MPD)	0.0003
blackscholes	Approximate Financial Analysis	Average Relative Error (ARE)	0.014
MNIST	Approximate Computer Vision	Number of Correct Decision (NCD)	0.8
SHA	Full-accurate Security	-	-
qsort	Full-accurate Automative	-	-

As mentioned in the previous section, NOSTalgy utilizes a sensor or countdown timer that can detect the environmental conditions fluctuations or input variations. During our simulations, we assume that a countdown timer is triggered every two hours during a day. We use the images taken from a city camera (with a constant view angle) at different times of the day as our application's input [40] for `corner detection`, `edge detection`, `image smoothing`, and `sobel filter`. For `MNIST` we use the test images in [48] as our inputs. Twelve different inputs feed to `blackscholes` at different times of day. Finally, for `qsort` and `SHA`, we use the standard inputs introduced in MiBench benchmark suite [23].

NOSTalgy can choose one of the quality levels mentioned in Table 3 to serve the write requests in the L2 cache. As mentioned in the previous section, NOSTalgy chooses the quality level based on the vulnerability of the requested data and the environmental conditions that affect the output quality of the applications. While every write operation in QL0 is considered to perform without any fault, we modify gem5 to inject write operation faults with the rates mentioned in Table 3 for the other quality levels (QL1, QL2, and QL3).

6.2 Simulation Results

In the following, we will evaluate NOSTalgy from two perspectives, i.e., energy saving, and output quality. Also, we discuss the performance and area overheads of NOSTalgy. For the sake of comparison, we implemented and compared the following approaches:

- **Baseline**: we consider an STT-MRAM L2 cache in full-Accurate mode (QL0), which does not benefit from any quality-energy knobs adjustment as the baseline approach.

- **On-demand Actuation Knob Adjustment (OAKA)**: Here, we emulate prior approaches that only rely on programmer hints, embedded in the programs codes, to actuate the write current of STT-MRAM L2 cache during run-time [11], [12], [33]. Accordingly, unlike NOSTalgy, these approaches are open-loop approaches that are not aware of environmental conditions fluctuations or input variations.
- **NOSTalgy**: We implemented the mentioned closed-loop, cross-layered NOSTalgy approach in the STT-MRAM L2 cache.

Please note that directly comparing NOSTalgy with non-STT-MRAM approximation techniques is not possible, since those approaches considered different actuation knob(s) specific to their targeted memory technologies and leads to incomparable results.

6.2.1 Energy Consumption

As mentioned earlier, the main superiority of NOSTalgy over other programmer driven knobs adjustment methods (like [11], [12], [33]) is its ability to change the quality-energy knobs based on environmental-condition fluctuations and input variations. Accordingly, while NOSTalgy guarantees to meet the application output quality thresholds, it dynamically changes the quality-energy knobs' configuration for approximate computing applications, whenever the environmental conditions or input variations allow the approximate computing application to function in more energy efficient quality levels. To this end, the achievement of NOSTalgy in changing the quality levels during the application execution plays an important role in the energy consumption of NOSTalgy-enabled L2 cache.

Fig. 7 depicts the normalized energy consumption of NOSTalgy-enabled L2 cache of an 8-core platform at different epochs during a day. We evaluate the L2 cache energy consumption considering three configurations, i.e., Baseline, OAKA approaches, and NOSTalgy. According to Fig. 7, NOSTalgy and OAKA both outperform the Baseline in all of the epochs during a day. However, for all epochs during a day, NOSTalgy outperforms OAKA approaches in terms of energy consumption since it considers the effects of environmental-condition fluctuations or input variations in adjusting the STT-MRAM quality-energy knob. As an example, considering the evaluated energy consumption of three scenarios at 7:15, NOSTalgy delivers about 33% and 52% energy improvement over OAKA approaches and Baseline configurations, respectively. As can be seen in the right side of Fig. 7, on average NOSTalgy provides about 27% and 42% energy improvement over OAKA and baseline configurations, respectively.

Fig. 8 depicts the results for the percent of times that our benchmarks run in each quality level during a day. According to Fig. 8, both NOSTalgy and OAKA approaches can run `corner detection` and `MNIST` at QL3 with the considered quality threshold at Table 4. Indeed, for `corner detection` benchmark the output quality threshold in Table 4 was defined at a level that environmental-condition fluctuations or input variations in our experiments did not

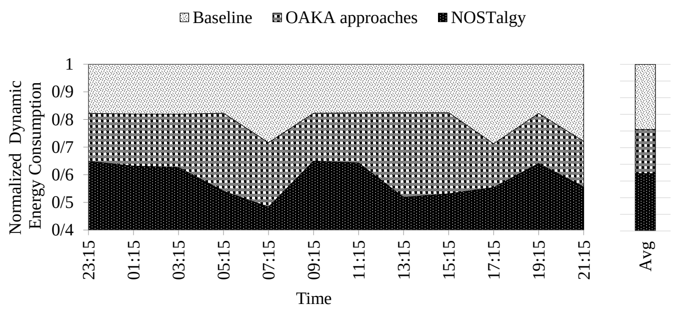


Fig. 7: Comparison of energy savings in the Baseline, OAKA approaches and NOSTalgy.

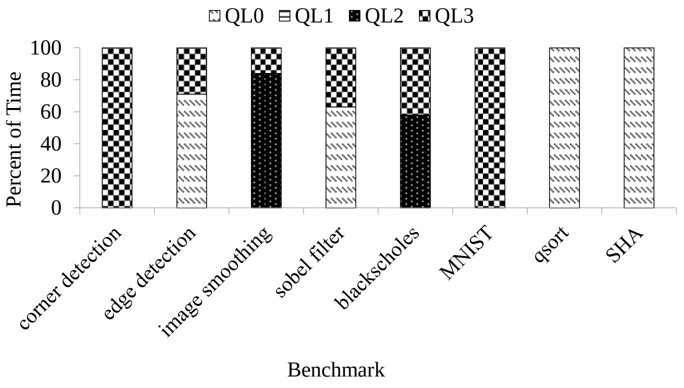


Fig. 8: Quality levels assigned by NOSTalgy to the applications.

threaten the `corner detection` output quality. In other words, with such quality threshold mentioned in Table 4 for `corner detection`, the design-time quality adjustment decisions by programmers (which was embedded in the `corner detection` code) did not change during the run-time. Thus, the benefits of NOSTalgy and OAKA approaches become the same. Furthermore, for `MNIST` we can see in Fig. 8 that elevating the write error rate at QL3 did not cause any false output in handwritten number detection process delivered by `MNIST`. This is because neural networks are intrinsically fault tolerant and the injected faults at QL3 write error rate were masked during our experiments for `MNIST`.

In this regard, like `corner detection`, the design-time quality adjustment decisions by programmers for `MNIST` (which was embedded in the `MNIST` code) did not change during the run-time. For `edge detection`, Fig. 8 confirms the possibility of running the application at QL3 for 29% of times in a day, while the considered quality threshold at Table 4 does not provide any opportunity for energy saving in OAKA approaches. As can be seen, NOSTalgy can run `image smoothing` at QL3 in 16% of times during a day. In contrast, OAKA approaches can only run `image smoothing` at QL2. According to Fig. 8, NOSTalgy could run `sobel filter` at QL3 (most energy efficient quality levels) for 37% of times in a day, while in the remaining 63%, `sobel filter` runs at QL0. It is noteworthy that in OAKA approaches, the considered quality threshold for `sobel filter` (mentioned in Table 4) does not provide any opportunity for energy saving. Considering blackscholes, we can see in Fig 8 that the online actuation knob management utilized in NOSTalgy can change the run-time QL of this benchmark from pre-defined QL2 to QL3 for 42% of times in a day, which provides a noticeable opportunity for energy saving. Finally, as can be seen, the fine-grained write current actuation knob provides this opportunity for OAKA and NOSTalgy approaches to run full-accurate qsort and SHA benchmarks alongside of our approximate applications, simultaneously. Indeed, we assigned QL0 for both qsort and SHA benchmarks.

6.2.2 Quality

Fig. 9 shows the normalized output quality evaluations for the approximate benchmarks listed in Table 4 at different times during a day. The normalized threshold quality of each application is shown by long-dashed ring and in the legends, the abbreviations of the quality metrics (defined in Table 4) start with "N" which denotes that the rings are representatives of the normalized values of quality metrics. As can be seen in Fig. 9, the environmental-condition fluctuations or input variations during a day lead to considerable output quality variation for most of the benchmarks. Unlike OAKA approaches, NOSTalgy considers these fluctuations and variations and tries to dynamically manage the STT-MRAM quality-energy knobs so that near-optimum energy efficiency is achieved while the quality threshold is also satisfied. Except for Fig. 9.(a) and Fig. 9.(f) where the predefined quality thresholds of corner detection and MNIST led to same STT-MRAM quality-energy knobs adjustment for both OAKA approaches and NOSTalgy, in other benchmarks (Fig. 9.(b), Fig. 9.(c), Fig. 9.(d), and Fig. 9.(e)) we can see that NOSTalgy tries to decrease the gap between the pre-defined threshold quality and on-line output quality to save more energy consumption compared with OAKA approaches. It is noteworthy that, according to Fig. 9, NOSTalgy meets the quality thresholds in all benchmarks during all time epochs.

6.2.3 Discussion on overheads

It should be noted that since OS utilizes the idle cores for running the test programs of approximate computing applications and in most of the times we have some idle cores in our systems, running NOSTalgy's test programs to determine the effects of environmental-condition fluctuations or input variations on the output quality of the applications does not impose any performance overhead to the system. Please note that for the situations that all the cores are busy, two cases can be considered. In the first case, since the approximate computing applications are not safety-critical, the quality threshold violations due to waiting for idle core would not lead to catastrophic failure. Thus, the application could run with the previous setting with no performance overhead. In the second case, we can use the typical priority approaches to make a core free for a critical test program. Indeed, since the test program have also a priority which is assigned to it by the operating system, all the priority-based scheduling algorithms that are currently used in the operating systems can work with NOSTalgy. On the other hand, based on the criticality level of the applications, developers are free to set different priority levels for the test

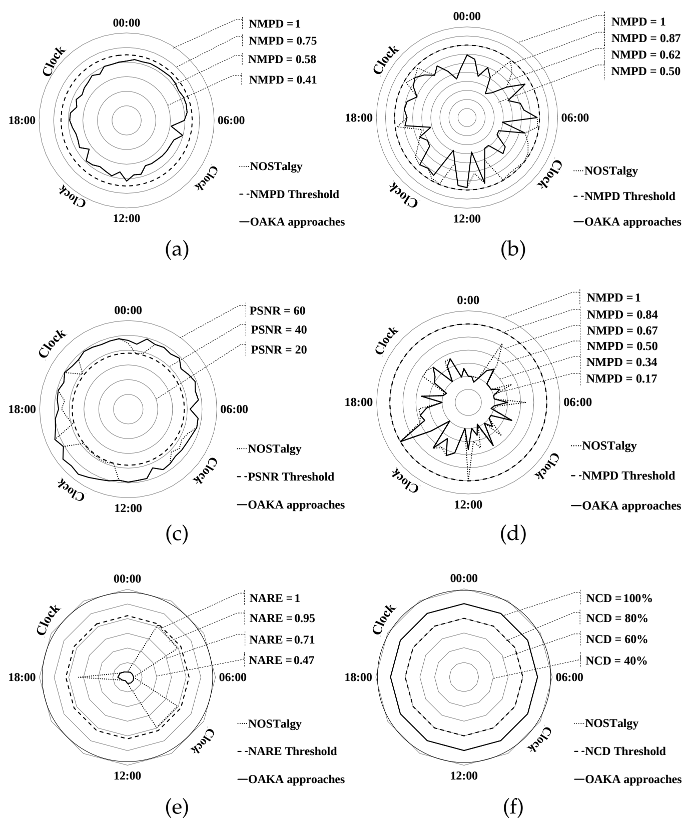


Fig. 9: Comparison of output qualities in different configurations for benchmarks: (a) corner detection, (b) edge detection, (c) image smoothing, (d) sobel filter, (e) blackscholes, and (f) MNIST.

programs and let the operating systems decide on executing the tasks and test programs in an efficient manner.

Besides, as the search operations in NOSTalgy table are done in parallel with TLB searches, these operations are performance overhead free. On the other hand, the processes of retrieving quality levels of saved blocks in L1 cache for write-back operations are also done in parallel with reading the address field of tag memory and do not impose any performance overhead to the systems. The only performance bottleneck that NOSTalgy should deal with is related to the voltage level shifter circuit. Since NOSTalgy should serve different write requests with different quality levels, the voltage level shifter circuit should be capable of switching between different voltage levels as quickly as the write requests arrive to the cache. To this end, we need a fast voltage level shifter to support this requirement. In [44], a fast voltage level shifter is presented which is able to switch the voltages and write the values in about 10ns. Considering such a fast voltage level shifter, we conduct an experiment to examine whether we have enough room between consecutive write requests for changing the voltage levels according to the requested quality levels.

Fig. 10 shows the distribution of L2 write requests based on the time between two consecutive write operations. Considering the consecutive write requests with time difference of less than 40ns, almost no write operation was observed in our simulation experiments. This confirms that we have

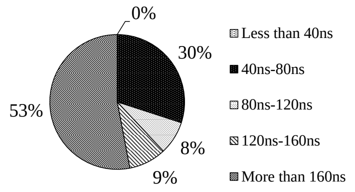


Fig. 10: Distribution of L2 write requests based on the time between two successive write operations.

enough room to change the voltage levels of the consecutive write requests with different requested qualities for L2 cache. In other words, as the delay of the used voltage level shifter [44] is much less than the time difference between consecutive writes, so this would not be a concern for the performance. Alternatively, for those voltage level shifter circuits that could not work as fast as the proposed voltage level shifter circuit in [44], we could consider either of the following solutions (with imposing negligible performance overhead):

- A temporary write buffer could be included in NOSTalgy cache to hold the next write request when it arrives and let the voltage level shifter circuit catch up to the appropriate voltage level in the background. Then, the contents of the write buffer can be written to the final destination in L2 cache memory. Here, we need to consider the Read-After-Write (RAW) hazard. Accordingly, for each request the write buffer should be checked in parallel with the L2 cache tag array, not to contain a valid data for the coming request. Then, in case of requesting the buffered value, the coming request should be answered from the write buffer.
- NOSTalgy cache does not issue the write acknowledgment until the voltage level shifter circuit provides the requested voltage for the later write request.

We designed the NOSTalgy controller peripheral circuits and synthesized them using Synopsys Design Compiler®. Indeed, we synthesized the 3 VSes and a multiplexer used in NOSTalgy controller 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 1.82×10^{-3}% (and 3.43×10^{-5}%) w.r.t. the basic cache architecture, which is negligible. Accordingly, NOSTalgy only imposes area and static energy overheads to the design by introducing the QL (Q in Fig. 4) field for the tag memory. 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. 11 shows that the L2 base design would occupy about $1.694mm^2$ 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 data array and Q fields. In the other words, considering the NOSTalgy design, adding the Q fields to the tag part of the L2 requires about 1% of the L2 area as depicted in Fig. 11, which is negligible. Considering the same discussion, it would be obvious that

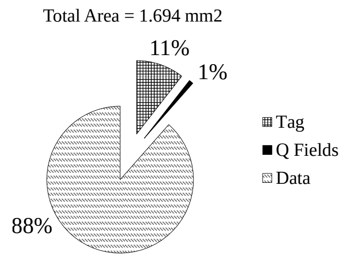


Fig. 11: L2 cache NOSTalgy-enabled Q field contribution in chip area.

since the contribution of NOSTalgy Q field in chip area is not noticeable, its contribution in static energy overhead is also negligible.

Finally, one can argue that in some scenarios, in which the OS can not immediately find the idle cores to run the test programs and determine the new QLs, applications may experience too many errors (in the situation that the QLs should be increased) or energy inefficiency (in the situation that the QLs should be decreased). While the later side-effect does not threaten the application output quality, the first one may violate the application output quality thresholds. Generally, since most of the approximate computing applications are not safety critical, these errors are tolerable until we can run the test programs and find the optimum quality levels for running the applications in new environmental conditions. However, for those applications that are sensitive to these errors, a preemptive scheduling technique assigning the highest priority to test programs could be employed.

7 Conclusions

In this paper, we introduced NOSTalgy, a hardware/software approach for dynamically trading quality of STT-MRAM caches for energy savings in the on-chip memory hierarchy of systems running approximate applications. NOSTalgy utilizes fine-grained cache-line-level actuation knobs with different levels of quality for individual write accesses that dynamically adjust based on the delivered output quality and the threshold quality of applications. Our experimental results with a set of benchmarks show that NOSTalgy satisfies output quality thresholds, while achieving up to 52% energy savings with negligible performance and area overheads.

References

[1] Z. Azad *et al.*, "An Efficient Protection Technique for Last Level STT-RAM Caches in Multi-Core Processors," *IEEE TPDS*, 2016.
[2] K. Wang *et al.*, "Low-power non-volatile spintronic memory: STT-RAM and beyond," in *Applied Physics*, 2013.
[3] T. Devolder *et al.*, "Single-Shot Time-Resolved Measurements of Nanosecond-Scale Spin-Transfer Induced Switching: Stochastic Versus Deterministic Aspects," *Phys. Rev. Lett.*, 2008.
[4] 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*, 2012.
[5] H. Farkhani *et al.*, "STTRAM Energy Reduction Using Self-Referenced Differential Write Termination Technique," in *IEEE Trans. VLSI Syst*, 2017.
[6] Y. Kim *et al.*, "Write-optimized reliable design of STT MRAM," in *Proc. of ISLPED*, 2012.

[7] S. Chatterjee *et al.*, "A Scalable Design Methodology for Energy Minimization of STTRAM: A Circuit and Architecture Perspective," in *IEEE Trans. VLSI Syst*, 2011.

[8] S.P.Park *et al.*, "Future cache design using STT MRAMs for improved energy efficiency: Devices, circuits and architecture," in *Proc. of DAC*, 2012.

[9] H. Sun *et al.*, "Design Techniques to Improve the Device Write Margin for MRAM-based Cache Memory," in *Proc. of GLSVLSI*, 2011.

[10] W. Kang *et al.*, "High Reliability Sensing Circuit for Deep Submicron Spin Transfer Torque Magnetic Random Access Memory," *Electronics Letters*, 2013.

[11] A. Monazzah *et al.*, "QuARK: Quality-configurable Approximate STT-MRAM Cache by Fine-grained Tuning of Reliability-Energy Knobs," in *Proc. of ISLPED*, 2017.

[12] A. Ranjan *et al.*, "Approximate Storage for Energy Efficient Spintronic Memories," in *proc. of DAC*, 2015.

[13] S. Liu *et al.*, "Flikker: Saving DRAM Refresh-power Through Critical Data Partitioning," in *Proc. of ASPLOS*, 2011.

[14] V. K. Chippa *et al.*, "Analysis and Characterization of Inherent Application Resilience for Approximate Computing," in *Proc. of DAC*, 2013.

[15] A. Sampson *et al.*, "EnerJ: Approximate Data Types for Safe and General Low-power Computation," in *Proc. of PLDI*, 2011.

[16] K. Munira *et al.*, "A Quasi-Analytical Model for Energy-Delay-Reliability Tradeoff Studies During Write Operations in a Perpendicular STT-RAM Cell," *IEEE TED*, 2012.

[17] H. Li *et al.*, "Performance, Power, and Reliability Tradeoffs of STT-RAM Cell Subject to Architecture-Level Requirement," *IEEE TMAG*, 2011.

[18] F. Oboril *et al.*, "Fault Tolerant Approximate Computing Using Emerging Non-volatile spintronic Memories," in *Proc. of VTS*, 2016.

[19] N. Sayed *et al.*, "Exploiting STT-MRAM for approximate computing," in *Proc. of ETS*, 2017.

[20] F. Sampaio *et al.*, "Approximation-aware Multi-level Cells STT-RAM Cache Architecture," in *Proc. of CASES*, 2015.

[21] B. Zeinali *et al.*, "Progressive Scaled STT-RAM for Approximate Computing in Multimedia Applications," in *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2017.

[22] N. Binkert *et al.*, "The Gem5 Simulator," *SIGARCH Comput. Archit. News*, 2011.

[23] M. R. Guthaus *et al.*, "MiBench: A Free, Commercially Representative Embedded Benchmark Suite," in *Proc. of WWC*, 2001.

[24] A. Yazdanbakhsh *et al.*, "AXBENCH: A Multi-Platform Benchmark Suite for Approximate Computing," in *IEEE Design and Test*, 2016.

[25] Y. LeCun *et al.*, "Gradient-based learning applied to document recognition," in *proc. of the IEEE*, 1998.

[26] A. Monazzah *et al.*, "Ler: Least-error-rate replacement algorithm for emerging stt-ram caches," 2016.

[27] D. Khudia *et al.*, "Rumba: An online quality management system for approximate computing," in *proc. of ISCA*, 2015.

[28] A. Raha *et al.*, "Quality configurable reduce-and-rank for energy efficient approximate computing," in *proc. of DATE*, 2015.

[29] A. Ranjan *et al.*, "Approximate memory compression for energy-efficiency," in *proc. of ISLPED*, 2017.

[30] M. Shoushtari *et al.*, "Exploiting Partially-Forgetful Memories for Approximate Computing," *IEEE Embedded Systems Letters*, 2015.

[31] A. Raha *et al.*, "Quality Configurable Approximate DRAM," in *IEEE Transactions on Computers*, 2017.

[32] M. Teimoori *et al.*, "AdAM: Adaptive Approximation Management for the Non-Volatile Memory Hierarchies," in *Proc. of DATE*, 2018.

[33] A. Ranjan *et al.*, "STAxCache: An approximate, energy efficient STT-MRAM cache," in *Proc. of DATE*, 2017.

[34] M. Shoushtari *et al.*, "Quality-Configurable Approximate Memory Hierarchy: A Formal Control Theory Approach," in *Proc. of WAX*, 2018.

[35] B. Grigorian *et al.*, "Dynamically adaptive and reliable approximate computing using light-weight error analysis," in *NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2014.

[36] T. Yeh *et al.*, "The art of deception: Adaptive precision reduction for area efficient physics acceleration," in *Proc. of MICRO*, 2007.

[37] M. Shafique *et al.*, "Resilience-Driven STT-RAM cache architecture for approximate computing," in *Workshop on Approximate Computing (AC)*, 2015.

[38] S. Venkataramani *et al.*, "Quality Programmable Processors for Approximate Computing," in *Proc. of MICRO*, 2013.

[39] Y. Fang *et al.*, "SoftPCM: Enhancing Energy Efficiency and Lifetime of Phase Change Memory in Video Application via Approximate Write," in *Proc. of 21th Asian Test Symposium*, 2012.

[40] N. Jacobs *et al.*, "Consistent temporal variations in many outdoor scenes," in *Proc. of CVPR*, 2007.

[41] M. Carbin *et al.*, "Verifying Quantitative Reliability for Programs That Execute on Unreliable Hardware," in *Proc. of OOPSLA*, 2013.

[42] A. Sampson *et al.*, "Mobile Systems IV," University of Washington, Tech. Rep., 2015.

[43] F. Ishihara *et al.*, "Level conversion for dual-supply systems," *IEEE TVLSI*, 2004.

[44] D. Lee *et al.*, "High-performance Low-energy STT MRAM Based on Balanced Write Scheme," in *Proc. of ISLPED*, 2012.

[45] S. Wang *et al.*, "Word- and partition-level write variation reduction for improving non-volatile cache lifetime," *ACM Transaction on Design and Automation Electronic System*, vol. 23, no. 1, pp. 4:1–4:18, 2017.

[46] L. Chang *et al.*, "Evaluation of spin-Hall-assisted STT-MRAM for cache replacement," in *proc. of NANOARCH*, 2016, pp. 73–78.

[47] X. Dong *et al.*, "NVSim: A Circuit-Level Performance, Energy, and Area Model for Emerging Nonvolatile Memory," *IEEE TCAD*, 2012.

[48] Y. LeCun. "the mnist database of handwritten digits". [Online]. Available: http://yann.lecun.com/exdb/mnist.

Arash Salahvarzi received the B.Sc. degree in computer engineering from Shahid Chamran University (SCU), Ahwaz, Iran, in 2016. He received the M.S degree in Computer engineering from Iran University of Science and Technology (IUST), Tehran, Iran, in 2018. He is a member of Dependable Systems and Architectures Laboratory (DSA) from 2016. His research interests include Low Power Design, non-volatile memory systems, and fault-tolerant embedded system design.

Amir Mahdi Hosseini Monazzah received his Ph.D degree in computer engineering from the 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.

Mahdi Fazeli received the M.Sc and Ph.D. degrees in computer engineering both from the Sharif University of Technology, Tehran, Iran, in 2005 and 2011, respectively. He has been with the department of computer engineering, Iran University of science and technology (IUST), since 2011, where he is currently an associate professor. He has established and chaired two research laboratories at IUST since 2012, namely Dependable Systems and Architectures Laboratory (DSA) and Networked and Embedded System Laboratory (NESL). His research interests include computer architectures, hardware accelerators, reliablility issues in VLSI circuit technologies, low power circuits and systems, dependable embedded systems.

Kevin Skadron is the Harry Douglas Forsyth professor and chair of the Department of Computer Science at the University of Virginia, where he has been on the faculty since 1999. His research focuses on heterogeneous architecture, design and applications of novel hardware accelerators, and design for physical constraints such as power, temperature, and reliability. Skadron and his colleagues have developed a number of open-source tools to support this research, including the HotSpot thermal model, the Rodinia GPU benchmark suite, the ANMLZoo automata benchmark suite, the MNCaRT automata processing design framework, and the RAPID programming language. Skadron is a Fellow of the IEEE and the ACM, and recipient of the 2011 ACM SIGARCH Maurice Wilkes Award.