IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 9 January 2024, accepted 4 February 2024, date of publication 19 February 2024, date of current version 23 February 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3366074

RESEARCH ARTICLE

Multi-Retention STT-MRAM Architectures for IoT: Evaluating the Impact of Retention Levels and Memory Mapping Schemes

BELAL JAHANNIA, SEYED ALI GHASEMI, AND HAMED FARBEH, (Member, IEEE)
Department of Computer Engineering, Amirkabir University of Technology (Tehran Polytechnic), Tehran 15875-4413, Iran
Corresponding author: Hamed Farbeh (farbeh@aut.ac.ir)

ABSTRACT In recent years, the energy consumption of IoT edge nodes has significantly increased due to the communication process. This necessitates the need to offload more computation to the edge nodes to minimize data transmission over the network. To achieve this, higher-performance CPUs and memory are required on the edge nodes. In this context, we propose an energy-efficient memory architecture specifically designed for edge nodes. STT-MRAM is a promising memory technology that offers potential replacements for SRAM and Flash in IoT devices. STT-MRAM exhibits notable advantages over traditional memory technologies, such as non-volatility for data retention without continuous power supply and energy efficiency, resulting in extended battery life for portable devices and IoT applications. Its potential for higher memory density and scalability through standard fabrication processes further enhances its appeal for next-generation memory solutions. However, the high write energy consumption is its main disadvantage. Previous works have explored non-volatility relaxation in CPU cache but there is a need to extend this approach to main memory in IoT devices. In this paper, we propose a multi-retention STT-MRAM architecture for IoT main memory. Additionally, we propose a memory mapping scheme for the suggested memory architecture and examine the impact of more relaxed retention levels on energy consumption. To the best of our knowledge, this is the first study to thoroughly investigate the optimal thermal stability factor value for STT-MRAM in IoT applications while also considering optimal memory mapping. The proposed architecture reduces energy consumption by an average of 70% and up to 83% compared to the currently used non-volatile STT-MRAM architecture. Furthermore, we propose two memory mappings that are easy to use and achieve an average energy savings that is just 5% away from the ideal mapping.

INDEX TERMS STT-MRAM, multi-retention STT-MRAM, IoT, memory mapping.

I. INTRODUCTION

The energy consumption of IoT nodes is experiencing rapid growth, primarily due to the significant energy consumed during data transmission over the network. Typically, IoT nodes process input data and transmit the resulting calculations over the network [1]. However, the increase in data transfer has led to a substantial rise in energy consumption for both network transmission and overall node operation. To mitigate this energy consumption, it is necessary to offload more computation to the node and minimize data transmission [2]. Achieving this requires higher-performance devices with larger non-volatile memory capacities. However, larger memory sizes also result in increased energy consumption, making memory energy efficiency a critical consideration. In general CPUs, the memory hierarchy alone accounts for approximately 40% of the total power consumption [3]. As memory sizes grow, memory becomes a significant energy-intensive component in IoT devices.

Different types of data in IoT applications have varying lifetimes. For instance, in a device used for the detection of wildfires, data lifetimes can differ significantly. these devices consist of sensors and cameras which work together to detect wildfires. The sensors collect real-time data like temperature,

The associate editor coordinating the review of this manuscript and approving it for publication was P. K. Gupta.

humidity, and smoke levels, while the cameras provide visual images of the surroundings. Since these devices are in remote and difficult-to-reach locations, we must be careful of their energy consumption due to the difficulty in replacing their batteries.

The data generated by these devices has different lifetimes depending on its purpose and importance. When the device analyzes data in real-time to identify potential wildfires, that data has a short lifetime. It only needs to be stored temporarily. Therefore, a low retention time memory is employed, proficiently storing the data for a temporary period until it is no longer required. On the other hand, the device also generates data that is recorded and used for research and prevention strategies. This data needs to be stored for a longer time [4]. To ensure it stays preserved, the device uses a memory that can keep data even when power is lost, like Flash memory. This allows the data to be accessed later or sent over the network in periodic updates.

By understanding the different lifetimes of the data, the wildfire detection device optimizes its memory usage. It does not treat all data the same, which would waste energy. Instead, it uses one type of memory for real-time analysis and another for long-term storage. This helps manage data efficiently, reduces energy usage, and effectively detects and prevents wildfires in sensitive areas.

Spin-Transfer Torque Magnetic Random Access Memory (STT-MRAM), a promising non-volatile memory with minimal leakage energy, is emerging as a potential candidate to replace existing SRAM and Flash memory in the industry [5], [6]. In the context of IoT nodes and embedded devices, SRAM and Flash memory are losing their position in the memory hierarchy due to scaling issues and high leakage energy. STT-MRAM offers benefits over SRAM, including non-volatility for data retention without constant power, energy efficiency by reducing leakage energy, and potential higher memory density and scalability due to its compatibility with standard CMOS. Additionally, it boasts durability against radiation, and better read/write endurance. Although STT-MRAM offers a viable alternative to SRAM, it suffers from high write energy requirements. Simply replacing SRAM with STT-MRAM throughout the memory hierarchy will not provide sufficient efficiency or performance. Thus, it is crucial to utilize STT-MRAM properly to minimize power consumption while maintaining performance. The main contributions of this paper are as follows:

- We determine the optimal retention time adjusted by thermal stability factor (Δ) for STT-MRAM in IoT applications. To the best of our knowledge, this is the first comprehensive study investigating the appropriate Δ for STT-MRAM in the context of IoT applications while trying to achieve the ideal memory architecture by adding more retention levels.
- We propose a memory mapping scheme for the suggested memory architecture. Our mapping approach offers a low-cost solution that is energy-efficient and comparable to the ideal mapping. Then proposed mapping simplifies memory management by considering the maximum data lifetimes of memory addresses. This eliminates the need for runtime address mapping and makes memory utilization more efficient.
- We explore the impact of relaxed retention levels on energy consumption. Our two-level memory architecture reduces energy consumption by an average of 67%, with the best case achieving a reduction of 82% compared to currently available non-volatile STT-MRAM. Similarly, the three-level architecture achieves an average energy reduction of 70%, with the best case achieving a reduction of 83%.
- For optimum energy efficiency, we also propose dynamic and static memory mapping. Our dynamic memory mapping approach performs 67% energy efficiency, while our static mapping gives 62% while significantly reducing mapping effort.

The remaining sections of the paper are organized as follows. Section II provides the necessary background on STT-MRAM memories. In Section III, we review related work and discuss their limitations. Section IV presents our findings and introduces proposed memory architecture and memory mapping. Section V presents the simulation setup and results. Finally, we conclude our findings in Section VI.

II. PRELIMINARY

A. STT-MRAM BASICS

Spin-Transfer Torque Magnetic RAM (STT-MRAM) is an emerging memory technology that holds the potential to serve as a universal memory solution. STT-MRAM offers several advantages, including near-zero leakage energy, higher density compared to SRAM, and non-volatile behavior similar to Flash memory, allowing data retention without power supply.

The structure of STT-MRAM consists of two fero magnetic layers: a free layer and a fixed layer (also known as reference layer), separated by an oxid-barrier layer (spacer layer) [7]. This three-layer structure is known as Magnetic Tunnel Junction (MTJ). The content of each memory cell is determined by the magnetic orientation of the free layer. In the parallel state, where the magnetization of the free layer aligns with the fixed layer, the cell exhibits lower resistance, representing a stored "0" (Fig. 1 (a)). Conversely, in the anti-parallel state, where the free layer has the opposite magnetic direction to the fixed layer, the cell demonstrates higher resistance, indicating a stored "1" (Fig. 1 (b)).

B. IMPACT OF RELAXED NON-VOLATILITY ON WRITE OPERATIONS

Write operation in STT-MRAM involves applying a high current to change the magnetic direction of the free layer. However, this process consumes a significant amount of energy and requires more time compared to SRAM or DRAM [8]. As a result, STT-MRAM faces performance and energy inefficiencies when compared to other memory technologies.

FIGURE 1. Schematic illustration of an STT-MRAM cell in (a) parallel state (logic "0") (b) anti-parallel state (logic "1").

To address these challenges, researchers have explored various approaches, including the relaxation of non-volatility. While STT-MRAM is typically classified as a non-volatile memory, it is possible to create semi-volatile STT-MRAM cells by relaxing the non-volatility requirements. By doing so, the retention times of the memory cells can be reduced, resulting in lower write energies.

Retention time refers to the period between the initial write to a memory cell and the time at which a bit-flip may occur, causing the data to become invalid. In the case of non-volatile memories like Flash, the retention time is typically considered to be around 10 years. The retention time of an STT-MRAM cell can be calculated using (1):

$$RetentionTime(t_{store}) = \frac{1}{f_0} e^{\Delta} \tag{1}$$

where, f_0 is the *thermal attempt frequency*, which is in the order of 1GHz for storage purposes (1ns). t_{store} is the retention time of a cell. Retention time and write energy as a following of that, can be reduced by lowering the thermal stability factor (Δ). This can be achieved by adjusting the dimensions of the free layer of MTJ cells during fabrication. Δ in STT-MRAM is influenced by the physical properties of the memory cell. A higher value of Δ indicates a higher retention time, but it also implies that the write operation requires more energy or pulse duration.

Thermal stability factor can be calculated by (2) [9]:

$$\Delta = \frac{H_K M_S A_r t}{2 k_B T} \tag{2}$$

where, H_K is anisotropy field, M_S is saturation magnetization, A_r is the area of the MTJ, t is the thickness of the free layer, k_B is Boltzmann constant and T is system temperature in kelvin.

The thermal stability factor, influenced by parameters such as H_K, M_S, or A_r, directly impacts the retention time of STT-MRAM. Reducing any of these parameters decreases the thermal stability factor and, consequently, the retention time. Similarly, reducing the planar area of STT-MRAM also decreases Δ. By lowering Δ, we can achieve STT-MRAM cells with shorter retention times. This is beneficial because it reduces the energy consumption needed for write operation. However, there is a trade-off - the downside is that data can be kept for a shorter period of time. Numerous researchers have explored the feasibility of modifying these parameters and proposed different approaches to modify the thermal stability factor and, consequently, retention time accordingly [10]. However, exploring these modifications further is beyond the scope of this study.

Table 1 demonstrates the relationship between thermal stability factor, retention time, and write energy [11], [12]. It demonstrates that reducing the thermal stability factor significantly reduces the write energy and retention time. It is noteworthy that achieving a higher retention time in STT-MRAM requires a higher value of Δ, which in turn leads to an increased energy consumption due to higher write current or longer write latency. Therefore, it is energy-efficient to utilize a lower retention time whenever possible.

Regarding the write time of STT-MRAMs, it is noteworthy that IoT devices often operate at significantly lower clock frequencies than high-end processors, in our case 100 MHz, and require around 5ns to perform a write operation, which is well within STT-MRAM's capabilities. Moreover, our main focus in this work is on maximizing energy efficiency to prolong battery life and enhance the overall device efficiency.

In summary, changes in the cell physics of STT-MRAM affect the thermal stability factor and thus the retention time and write energy of the memory cell. To meet the requirements of storage applications, a retention time of more than 10 years is necessary, which requires $\Delta > 40$. However, there is a potential situation that we can exploit to reduce the energy consumption of IoT devices. For example, by reducing Δ to 18, which results in a retention time of approximately 65 milliseconds, a vast majority of data retention requirements can be fully satisfied, and also the write energy can be reduced significantly. In this paper, we aim to find the optimal Δ value for IoT applications.

TABLE 1. Effect of thermal stability factor (delta) on STT-MRAM retention time and write energy [11], [12].

Retention time	Thermal stability(Δ)	Write energy (nJ)
10 years	40.3	0.66
1 year	37.99	0.61
1 month	35.42	0.56
1 week	34.04	0.54
1 day	32.09	0.50
1 hour	28.91	0.44
1 min	24.82	0.37
1 sec	20.72	0.29
1 ms	13.82	0.17
100 us	11.51	0.13
1 us	6.91	0.06

C. READ OPERATION

The read operation in STT-MRAM involves applying a small voltage to the memory cell and comparing the resistance of the cell to a reference cell to determine its content. Compared to SRAM and DRAM, the read energy and latency

of STT-MRAM are not significantly different. In addition, the read energy in STT-MRAM is considerably lower when compared to the write energy. Furthermore, relaxing non-volatility does not result in a significant reduction in read energy. Therefore, our main objective is to enhance energy efficiency during the write operation. As a result, our investigation primarily centers around the RAM data section of the program, where write operations take place. Other sections of a program, like code, are just read-only.

It is worth noting the difference between *retention time* and *data lifetime* to avoid any confusion in this paper. *Data lifetime* refers to the duration between the first write and the last read of application data or variables. It represents the period during which data is defined in the application and remains in use until its final access. This parameter is determined and calculated by the program. On the other hand, *memory retention time* refers to the duration for which a memory cell can retain data without any changes, as discussed in Section I. The memory retention time is a parameter dictated by the physics of the memory cell.

III. RELATED WORK

Recent studies in the field of STT-MRAM can be categorized into several topics. Firstly, there are studies focused on reducing the write energy of STT-MRAM. These include approaches that aim to minimize or avoid write operations, address the 0/1 asymmetry in writing, and propose hybrid architectures combining different memory technologies. Some researchers have also explored the use of multi-retention cells in STT-MRAM, which involves employing low-retention cells as an alternative to SRAM. These designs typically utilize different retention times for different cache levels to optimize energy consumption. Replacing main memory with STT-MRAM has also been investigated. In addition, some studies specifically explore the use of STT-MRAM in IoT applications and address memory mapping for such applications. Research in alternative domains has explored the energy efficiency aspects within other memory technologies, including PRAM and Phase Change Memory (PCM) [13], [14], [15].

A. WRITE ENERGY OF STT-MRAM CELL

As mentioned previously, STT-MRAM exhibits an energy-consuming write operation, prompting researchers to explore solutions for this issue. This section investigates STT-MRAM studies that try to lower energy consumption by avoiding energy-consuming write operations [16].

1) ADDRESSING HIGH WRITE ENERGY

Unlike SRAM, where activity does not significantly impact energy consumption due to high leakage power, STT-MRAM experiences an increase in energy consumption with increased activity. Therefore, these research efforts aim to address this challenge and minimize energy consumption in STT-MRAM. Moreover, in this section, we explore the efforts made in STT-MRAM research to reduce energy consumption by avoiding write operations.

Bouziane et al. propose a compile-time approach to reduce overall energy consumption in memory by mitigating excessive write operations and minimizing the number of writes to memory [17]. This approach involves comparing new data with the old data stored in memory, and if they are the same, the write operation is canceled.

Zhou et al. suggest utilizing the fact that during the early stage of a write operation in STT-MRAM, the old data remains valid and can be read [18]. They read the data and compare it with the new data to determine whether it needs to be written to memory or not.

Similarly, Bishnoi et al. emphasize the energy inefficiency of starting a write operation and suggests avoiding redundant writes altogether [19]. Their approach involves reading the data from memory first and then deciding whether to initiate the write operation based on the comparison results.

Yazdanshenas et al. propose a coding mechanism based on limited weight codes to reduce the number of writes on last-level STT-MRAM caches [20]. Park et al. introduce an approach where, upon a write-back request from L1 cache to L2 cache, unchanged L1 cache lines are not written to the L2 cache [21]. To achieve this, the authors divide each cache line into partial lines and assign a history bit to each partition. Based on the history bit, they decide on whether to write the partition or not. By selectively writing only the modified portions of cache lines, unnecessary write operations can be avoided, reducing energy consumption and improving efficiency.

Jung et al. utilize the observation that more than 50% of the data in the L2 cache is zero. To leverage this fact and reduce energy consumption, they introduce an *all-zero-data* flag that is added to the tag arrays [22]. This flag serves to indicate whether a specific data entry in the cache is zero or not. Throughout the write process, if the incoming data is zero, instead of writing the actual data, they simply write the zero flag and skip the operation of writing the data itself. This approach avoids unnecessary write operations and saves energy. During a read access, the system first reads the *all-zero* flag. Based on the value of the flag, they make a decision on the final data to be returned. If the all-zero flag indicates that the data is indeed zero, there is no need to read the actual data. However, if the flag indicates that the data is non-zero, they proceed to read the actual data. By employing this approach, the authors effectively reduce the number of write operations and optimize read accesses in the L2 cache, resulting in energy savings and improved efficiency.

Bishnoi et al. exploit the write asymmetry between "0" and "1" in STT-MRAM cells. The authors aim to terminate the write operation as soon as the cell content matches the data being written [19]. This approach allows them to optimize both the time and energy spent on memory write requests, avoiding unnecessary resource allocation. By leveraging the write asymmetry, the authors effectively reduce energy consumption and

improve the efficiency of write operations in STT-MRAM cells.

2) 0/1 ASYMMETRY

STT-MRAM write latency and energy differ when it comes to 0/1 writing due to the need to change the magnetic direction to the opposite of the free layer.

Nigam et al. exploit the asymmetric write energy of ''0'' and ''1'' in STT-MRAM to propose an inverted code architecture [23]. The main idea is to invert data with more ''1'' than ''0'' to reduce the energy consumption associated with writing ''1''. The authors also propose three methods to reduce the writing energy of STT-MRAM: two based on cell parameters (Ms and MTJ structures) and one based on lowering the thermal stability factor, which leads to decreased data retentions. It is important to note that while the physical alterations required to reduce cell retention time are discussed, the study does not address the need for an application or software-based approach to appropriately map data to the suitable memory cell.

Chen et al. aim to reduce the energy consumption of the last-level cache by introducing a reconfigurable cache architecture that can turn on/off the SRAM/NVM arrays [24]. However, turning the ways on and off is a wasteful strategy because there will always be some ways that are turned off. This is because the data in the memory is not evenly distributed. For write-intensive data, they use SRAM and take advantage of STT-MRAM's almost negligible leakage feature.

3) HYBRID ARCHITECTURES

In a hybrid architecture, the downsides of STT-MRAM are offset by leveraging the strengths of various technologies. Incorporating a hybrid architecture enables the compensation of STT-MRAM's inherent disadvantages by harnessing the advantages of complementary technologies. For instance, pairing STT-MRAM with SRAM brings together the non-volatility of STT-MRAM and the high-speed accessibility of SRAM, resulting in a memory subsystem that offers both data retention during power-off states and rapid data access for critical operations. The combination of these technologies effectively overcomes the drawbacks of each component, leading to an overall improvement in system efficiency and performance.

Sun et al. propose an energy-efficient L2 cache by utilizing one SRAM way in a 32-way cache (31M1S)(31 STT-MRAM line along with 1 SRAM line) [25]. The main concept is to transfer write-intensive blocks to SRAM. While recognizing that directly replacing SRAM with STT-MRAM is inefficient in terms of energy and performance, the hybrid architecture can provide competitive performance compared to SRAM while maintaining energy efficiency.

Ahn et al. suggest an architecture to enhance the energy efficiency of STT-MRAM caches [26]. Based on the observation that higher-order bits of data tend to be similar while lower-order bits change frequently, the architecture stores only the changing half of the word instead of saving the entire word.

Quan et al. utilize a prediction table for the STT-MRAM and SRAM hybrid cache to determine the appropriate cache line for data placement [27]. SRAM lines are positioned alongside non-volatile STT-MRAM in the cache to address the high write energy and long write latency of STT-MRAM. By adopting this approach, the architecture leverages the performance benefits of SRAM while also taking advantage of the density provided by STT-MRAM. To determine the proper cache line, a table is employed that utilizes cache line activity to predict the optimal placement of data. Cache lines with write-intensive activity are directed to SRAM, while lines with lower write frequency are assigned to STT-MRAM lines. However, the downside of this approach is that it does not consider the non-volatility feature of STT-MRAM.

Imani et al. determine the placement of data by counting the number of ''1''s in the data [28]. The authors propose a cache policy and architecture to write data with more ''1''s than ''0''s and based on a threshold value, decide whether to store it in SRAM or STT-MRAM. The authors' goal is to find an appropriate threshold.

Goswami et al. present an energy-efficient hybrid shared memory architecture for GPGPU applications [29]. They utilize the capabilities of SRAM alongside STT-MRAM and implement overall optimizations to improve efficiency.

B. MULTI-RETENTION STT-MRAM

Although STT-MRAM exhibits significant write energy, it can be reduced by decreasing the STT-MRAM retention time and implementing multi-retention designs, similar to hybrid architectures (SRAM/STT-MRAM). A multi-retention STT-MRAM design might surpass hybrid SRAM/STT-MRAM due to its fine-tuned power-performance control, optimized write operations, simplified integration, and enhanced efficiency by tailoring memory behavior to application requirements.

Sun et al. proposed an STT-MRAM multi-retention cache architecture that utilizes a low retention time STT-MRAM (26.5us) for the L1 cache and a high retention time STT-MRAM (3.24s) for higher levels of cache [30]. The authors also introduce a simple DRAM-style refresh mechanism to prevent retention failures for blocks that reach their retention time limit. For lower cache levels, they suggest employing a 1-way low retention and 15-way high retention cache. The authors use bit redundancy to identify read/write-intensive blocks and place them in the appropriate cache way. However, the downside is that they do not consider exploring the actual data lifetime and use a heuristic search on the delta and find a delta that seems to be good based on the simulation.

Smullen et al. propose two different types of STT-MRAM cells: Read-optimized and Write-optimized [11]. These cells are optimized for efficient read and write operations,

respectively. The authors demonstrate that completely replacing SRAM with STT-MRAM is energy-efficient due to the low leakage of STT-MRAM. However, it also results in performance degradation due to the latency. To address this, the authors propose a hybrid cache architecture where SRAM is used for the L1 cache and relaxed STT-MRAM is used for the L2 and L3 caches, mitigating the performance degradation.

Samavati et al. investigate the use of multi-retention STT-MRAM memories in the last-level cache of GPUs [31]. The study finds that a retention time of 40ms efficiently satisfies 90% of cache block writes in GPUs. After presenting a 2-part STT-MRAM cache with different retention times, the authors propose energy consumption reduction by migrating rewritten data blocks from the high-retention (HR) part to the low-retention (LR) part and vice versa.

Manohar et al. proposed a strategy to store recently written blocks, which are likely to be rewritten soon, in a low-retention section [32]. This approach is based on the prediction that these blocks will be overwritten in the near future. They also suggest writing code blocks, which are not frequently written, to a high-retention section. Additionally, they consider a mechanism for changing between the two data regions, a requirement for caches but not for main memory. Their work focuses on cache levels of memory rather than main memory.

Agarwal and Chakraborty proposed utilizing a multi-retention STT-MRAM and assigning blocks to corresponding sets based on their arrival triggers [33]. Specifically, if a read miss triggers a block's arrival, it is placed in a set with a higher retention time, referred to as the read-set. Conversely, if a write miss prompts the block's arrival, it is placed in a write-set with a lower retention time.

Jog et al. explore the trade-offs between the non-volatility of STT-MRAM and achieving better latency and power efficiency for write operations [34]. The authors first profile the lifetime behavior of the last level cache (L2 Cache) to determine a suitable retention time. By applying this retention time alongside a refreshing scheme, this approach ensures data validity, energy efficiency, and performance improvement.

Arezoomand investigates the use of STT-MRAM in 3D embedded chip-multiprocessors and utilizes three different retention times for the shared L2 cache [35]. They employ a greedy algorithm to determine the number and placement of ST1 and ST2 memory banks.

Kim et al. employ STT-MRAM with a one-month retention time for main memory [36]. The results demonstrate that this STT-MRAM can function as main memory, offering performance and energy consumption comparable to DRAM.

Gajaria and Adegbija investigated the use of multi-retention STT-MRAMs in caches but did not extend their exploration to main memory [37]. They employed a set of predefined retention times and did not explore the full range of applicable Δ values.

C. MAIN MEMORY INVESTIGATION

As observed in the related work, researchers have not extensively explored multi-retention in main memory, particularly in the context of IoT. In this section, we explore works that have been done in this domain.

Kim et al. utilize relaxed retention STT-MRAM (one-month retention time) as the main memory for mobile devices [36]. Their results demonstrate that this STT-MRAM can effectively function as main memory, offering performance and energy consumption levels similar to DRAM.

Shihab et al. present a tailored STT-MRAM architecture designed to replace DRAM as the main memory [38]. This proposed architecture achieves comparable storage density to DRAM while delivering high performance and power efficiency. However, the authors do not consider the use of low retention STT-MRAM design.

Bouziane et al. use a program C-Code to find the lifetime of data, and using a mapping method, try to write data in the proper STT-MRAM part [39].

D. MAPPING TECHNIQUES FOR EMBEDDED SYSTEMS AND IOT

Jaykumar finds the best energy-efficient memory mapping by an exhaustive search in program code binary section mapping [40]. The authors argue that each application needs to decide whether to place text, stack, and data sections in non-volatile Ferroelectric RAM (FRAM) or high-performance SRAM.

Kim et al. analyzed some of IoT applications to find energy-efficient memory mapping [41]. The authors put the IoT OS code on FRAM. Based on a thorough investigation, it is shown that with the help of profiling, the authors can utilize a static memory mapping that can lower energy and power consumption.

Kultursay evaluates Flash replacement with STT-MRAM and indicates that without any optimization, STT-MRAM is power and performance inefficient [42]. However, by altering STT-MRAM and using a little optimization, STT-MRAM can act as a proper Flash replacement. The authors do not consider STT-MRAM retention relaxation.

As observed, none of the previous studies considered STT-MRAM non-volatility relaxation in main memory, especially in the context of IoT. Furthermore, it is worth noting that the use of a cache is not common in typical IoT devices due to the constraints on energy efficiency and device size. By relaxing the STT-MRAM cell's retention time, which reduces the write energy, we can achieve significant energy savings. We can further improve energy savings by adapting the memory management with IoT applications access pattern and the memory configuration. The 2-level retention architecture was introduced in [43], but does not delve deeply into exploring additional retention levels, which we explore in this paper. We also define an ideal architecture and a baseline, explaining it in detail with energy consumption equations to make it easier to compare with other proposed architectures. Additionally, we delve into the impact of memory mapping for the first

time, defining and comparing two techniques. These mapping techniques aid in the successful application of memory architectures for various purposes. The detailed analysis of thermal stability factors and the comparison of mapping techniques presented in this study provide valuable insights for optimizing multi-retention STT-MRAM architectures, specifically tailored for IoT devices' main memory

IV. PROPOSED ARCHITECTURE AND MAPPING

In this section, we first take a look at the traditional IoT-based architecture and then define baseline and ideal architectures. After that, we present our proposed architecture and utilize our profiling method to propose static and dynamic data mapping techniques.

A. TRADITIONAL ARCHITECTURE

Fig. 2 (a) illustrates the traditional memory architecture for an embedded IoT device. Currently, this architecture consists of Flash memory for non-volatile sections (such as Program-Code and Read-Only Data) and SRAM for high-performance requirements in the working RAM (Heap and Stack). This memory architecture stems from the fact that different types of data have varying requirements. For instance, program code and read-only data, which require long-term retention, are mapped to non-volatile memory. On the other hand, data with fast and frequent access requirements, like heap and stack data, is mapped to high-performance memory, such as SRAM.

However, this architecture has become increasingly energy inefficient due to scaling issues with SRAM and Flash. While STT-MRAM can be a potential alternative to SRAM, it requires more write energy. Simply replacing SRAM with STT-MRAM can not be energy-efficient or high-performing. To achieve energy efficiency while maintaining performance, we must use STT-MRAM thoughtfully.

B. OPTIMUM ENERGY - BASELINE - FIRST APPROACH

In this section, we establish the ideal architecture, a baseline, and an initial approach. The ideal architecture and the baseline provide standards against which we assess the improvements achieved by our proposed architectures.

1) OPTIMUM ENERGY: IDEAL MEMORY AND DYNAMIC MAPPING (IDEAL-DYNAMIC) (FIG. 2(B))

To design an effective architecture, it is important to first identify the optimum conditions and then strive to achieve this optimal architecture. The desired memory architecture aims to have a range of retention time values, denoted by Δ (Fig. 2(b)). In the figure, varying shades represent different STT-MRAM cell retention times, with lighter colors indicating low retention and darker colors signifying high retention. The progression from the lightest to the darkest shade signifies the full spectrum of attainable retention times. While this hypothesis might not be immediately feasible, it serves as a conceptual foundation for exploring possibilities and defining the ideal scenario. In this case, data mapping can be based on the specific lifetime of the data, ensuring that it remains in the STT-MRAM memory for the exact duration required to minimize energy consumption (referred to as optimum energy). The energy consumption of the architecture can be calculated using (3)):

$$\text{Energy} = \sum_{x=\min}^{x=\max} E_{w_{\Delta=x}} \times p_x \tag{3}$$

In (3), $E_{w_{\Delta=x}}$ represents the energy associated with the memory retention time specified by $\Delta = x$. The variable p_x denotes the percentage of write requests assigned to the corresponding memory section.

By summing up the products of the energy values and their respective write request percentages for each memory retention level, the equation allows us to model the overall energy consumption of the memory architecture. It is worth noting that the focus of this paper is on the RAM sections of the program, where most of the write requests are occurred.

However, implementing such a memory architecture is impractical due to the high fabrication costs and the complexities associated with managing a wide and continuous range of Δ values, write times, and write energies. It is not feasible to fabricate a design with a continuous range of Δ values and retention levels (Fig. 2(d)).

Considering these challenges, we are forced to choose a limited number of retention levels. This raises the question: What is the optimal Δ value or the number of levels to use? Determining the appropriate Δ value becomes a crucial consideration for these applications. Initially, we chose to adopt a two-level architecture. However, further investigation in Section IV-F will explore the leveraging of additional levels. Nevertheless, before delving into the concept of levels, it is essential to establish our baseline.

2) BASELINE: NON-VOLATILE STT-MRAM (DELTA= 40.3)

The architecture currently prevalent in the market involves the use of non-volatile STT-MRAM that serves all purposes within the memory hierarchy. This architecture is presented in Fig. 2(c). An example of this adoption is Lucid Motors' announcement regarding the utilization of STT-MRAM in their products [44]. We consider this existing approach as our baseline and proceed to compare our proposed methods with this architecture.

In the baseline architecture, all code, read-only data, heap, and stack are mapped to a non-volatile STT-MRAM (with a Δ=40.3). However, as we mentioned in Section I, this memory architecture is inherently energy inefficient due to applying the same approach to different types of data. In this architecture, the energy consumption for write operations is determined by multiplying the number of writes with the energy required for a single write, considering Δ=40.3. The formula representing write energy is provided as (4), where all requests (100%) are directed towards the non-volatile section. The use of percentages enables the normalization of

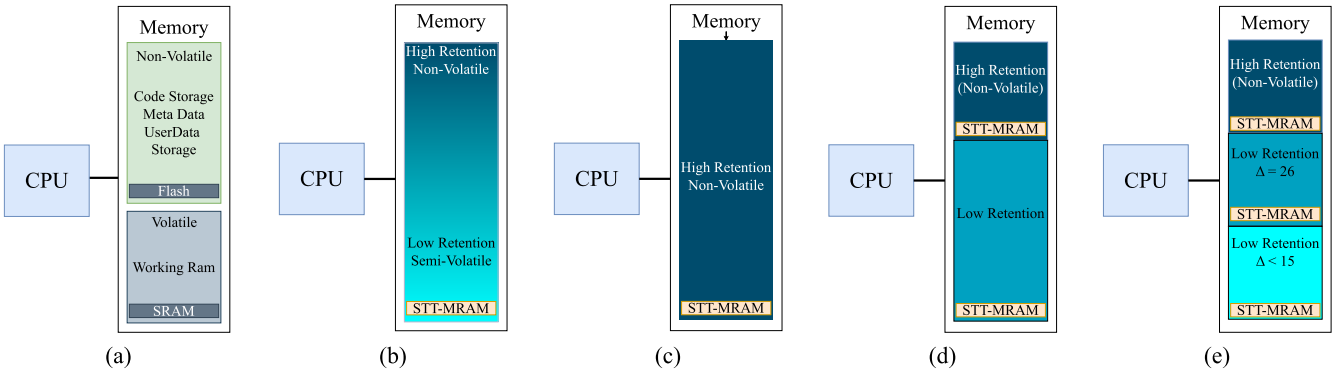

FIGURE 2. Memory Architectures. (a) traditional (b) ideal (c) non-volatile STT-MRAM (baseline) (d) two-level (e) proposed three-level.

energy consumption.

$$E_{baseline} = E_{w_{\Delta=40.3}} \times \%100 = (0.66 \times \%100) = 66 \quad (4)$$

The baseline architecture may not be a strong competitor for the traditional memory architecture due to its high energy consumption resulting from the write operations. To become a good competitor to traditional memory hierarchy, the idea is to utilize STT-MRAM with different retention times throughout the memory hierarchy. For example, incorporating semi-volatile STT-MRAM for data with shorter lifetimes.

To fulfill the requirements of the code and read-only sections, non-volatile memory is needed, which can be achieved through the use of non-volatile STT-MRAM. However, for RAM sections like heap and stack, which experience frequent write requests and are considered hot regions, a relaxed STT-MRAM is more suitable. Utilizing non-volatile STT-MRAM in these regions would be energy inefficient due to the power-consuming write operations. As a result, it is assumed that a maximum Δ value of 40.3 is assigned to the code and read-only data, but finding the best and appropriate Δ value for the low retention level in the memory architecture hierarchy is necessary.

The question arises regarding the number of levels to be used in the memory architecture and the proper Δ value for each level. This information highlights the need to determine the best Δ value for IoT applications. Previous studies (Section III) have mainly focused on the cache level within the memory hierarchy and proposed suitable Δ values for different cache levels. In contrast, our investigation aims to comprehensively explore the appropriate Δ value, not limited to cache levels. However, the Δ value for IoT applications has not been extensively studied. This research is crucial to identify the optimal Δ value for embedded applications, particularly for their main memory (RAM).

3) FIRST APPROACH: DELTA = 26

Based on our careful observations, we noticed that the RAM sections (Heap/Stack) in IoT applications typically do not require the long 10-year data retention time provided by non-volatile memories. Keep in mind that using non-volatile STT-MRAM with such an extended retention time is energy inefficient due to its high write energy consumption.

To address this issue, we consider an initial approach: we find the maximum data lifetime among all benchmarks and use it as an upper bound for all working RAM sections. This architecture, presented in Fig. 2(d), employs this approach. We have divided the memory into two distinct sections: low retention and high retention. This division essentially separates the address range into two parts, one for high retention and one for low retention. Through our analysis, we discovered that setting $\Delta = 26$ (equivalent to 200 seconds) can effectively fulfill all the data requests. Compared to $\Delta = 40.3$, this architecture results in a remarkable 42% improvement in energy consumption. We derived the energy consumption model (5) by calculating the energy required for a single write operation at $\Delta = 26$ and then multiplying it by the percentage of write requests. Notably, all write requests (100%) are directed to the $\Delta = 26$ section, similar to (4).

$$E_y = E_{w_{\Delta=26}} \times \%100 = (0.38 \times \%100) = 38 \quad (5)$$

C. OBSERVATION AND MOTIVATION

1) ADJUSTING DELTA VALUES FOR ENERGY EFFICIENCY

By analyzing the data lifetimes, we have identified that only a few data items require a Δ value of 26. Therefore, we can relocate the data that requires $\Delta = 26$ to the non-volatile section and reduce the Δ value for the low retention section of memory. This adjustment allows for significant energy savings. In Fig. 3, we provide an example using X and Y as retention levels. (5) and (6) outlines the energy consumption calculation for these points. The energy for Y is equivalent to the baseline energy (5), while the energy for X is the sum of write requests to the non-volatile section and the energy of the relaxed section (low retention) (6). By allocating the majority of write requests to the lower retention section ($\Delta = 17$), the overall energy consumption is expected to decrease.

Fig. 3 illustrates the percentage of satisfied requests for different Δ values. This result is obtained by running the entire MiBench benchmark suit, which we refer to as

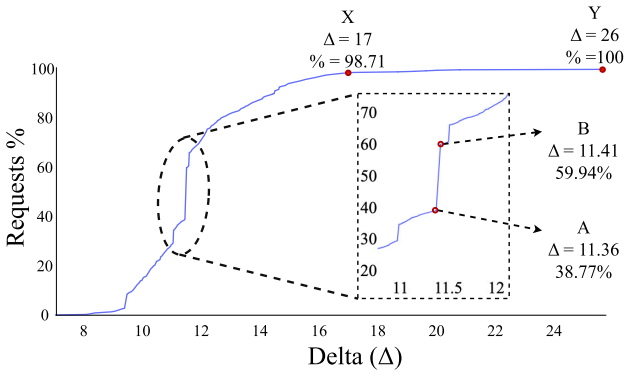

FIGURE 3. Percentage of successfully completed write requests for various delta values in an average IoT dataset [43].

the average [45]. Within the highlighted range, selecting $\Delta = 11.36$ satisfies 38.77% of requests, while $\Delta = 11.41$ satisfies 59.94% of requests. These results indicate that a slight decrease in Δ can lead to improved energy efficiency.

$$\begin{aligned} E_x &= (E_{w_{\Delta=17}} \times \%98.88) + (E_{w_{\Delta=40.3}} \times \%1.12) \\ &= (0.2278 \times \%98.71) + (0.6555 \times \%1.29) \\ &= 22.4861 + 0.8455 = 23.33108 \end{aligned} \tag{6}$$

Increasing the value of Δ results in a higher number of write requests being allocated to the low retention section, while reducing the number of write requests assigned to the non-volatile section. However, it is important to note that as Δ increases, the write energy also increases. From the perspective of satisfying the percentage of requests, we aim to increase Δ. This means that a larger value of Δ would lead to a higher percentage of requests being satisfied. However, considering the energy consumption aspect, we want to decrease Δ for every write operation in the STT-MRAM cell. This is because reducing Δ results in lower write energy consumption. Therefore, there is a trade-off between satisfying request percentages and minimizing write energy consumption, and finding the optimal value of Δ requires careful consideration of both factors.

2) MOTIVATIONAL EXAMPLE

As mentioned in Section VI, the example of wildfire detection highlights the fact that different types of data have varying lifetimes. Fig. 4 visually illustrates this concept in two parts: Fig. 4(a) demonstrates how distinct data can possess different lifetimes, while Fig. 4(b) shows that even within a single address, diverse lifetimes can exist. This presentation strongly emphasizes the crucial role of optimizing energy consumption by aligning the retention times of STT-MRAM memory cells with data lifetimes. Variables A and B, with varying lifetimes, exemplify the potential benefits of carefully mapping them to suitable STT-MRAM memory cells using corresponding Δ values (retention times) to achieve energy efficiency.

To further explore this concept, we executed a well-known mathematical program on an embedded core (Arm Cortex M0) and conducted a profiling analysis. The aim was to unravel the memory's write and read requests, thereby identifying the data lifetimes in CPU cycles associated with different memory addresses. The visualization of these data lifetimes for memory addresses is presented in Fig. 5. The x-axis in Fig. 5(a) represents the memory addresses, and the y-axis represents the corresponding data lifetimes. Each box in the graph contains four critical points: the minimum, maximum, and two quarter points (Fig. 5(b)), providing insights into the range of lifetimes required by memory cell variables. The illustration effectively depicts the diverse lifetimes, where smaller boxes denote shorter lifetimes and longer boxes symbolize extended lifetimes. This observation highlights two pivotal aspects: the variability of lifetimes among different addresses (Fig. 4(a)) and the presence of divergent lifetimes within a single memory address (Fig. 4(b)). A direct comparison between two chosen addresses, specifically (13, 117, 154) and (53, 70, 103, 721, 1497), serves to underscore the significant variations in data lifetimes both across and within memory addresses. This insightful analysis offers a comprehensive understanding of the varying lifetimes within the memory addresses, providing the foundational context for elucidating our strategies for energy-efficient memory management optimization.

Analyzing memory access patterns in IoT applications provides useful information about the data retention requirements of various addresses. We can identify which addresses require a higher retention level and which can be mapped to a lower retention level. This strategy enables us to increase energy efficiency and overall system performance by allocating resources efficiently based on the application's specific requirements.

D. TWO-LEVEL ARCHITECTURE: (2L-DYNAMIC)

Each Δ value is selected individually, and the energy consumption is determined based on that specific Δ. The process follows a specific order: if the data lifetime for a particular Δ is lower, it is assigned to the low Δ (Semi-Volatile) section, and If the data lifetime exceeds the chosen Δ, it is written in the high retention section of the memory ($\Delta = 40.3$).

During the profiling phase, we can determine the needed retention time for each data by counting the number of cycles between the first write of that data and the last cycle data is read. Following profiling, it is known the required Δ and the length of retention time required for each write request. Therefore, during compilation, data could be allocated to a low- or high-retention area of the RAM.

Fig. 6 shows the energy consumption of a two-level memory architecture as a function of the Δ value. This figure highlights that for Δ values below 14, the energy consumption is excessively high. This is due to a large number of requests being written to the high-volatile (or non-volatile) section of the memory. On the other hand, for Δ

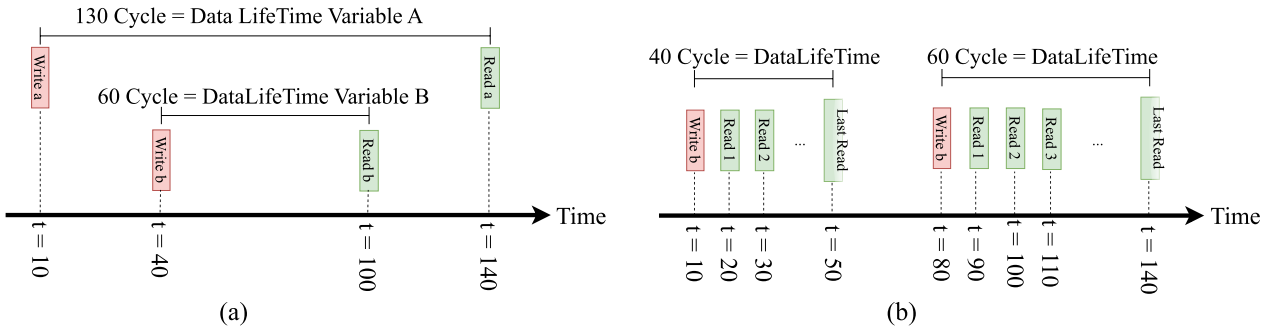

FIGURE 4. Example of different variables have different lifetimes. (a) two different variables (b) varying lifetimes within one variable.

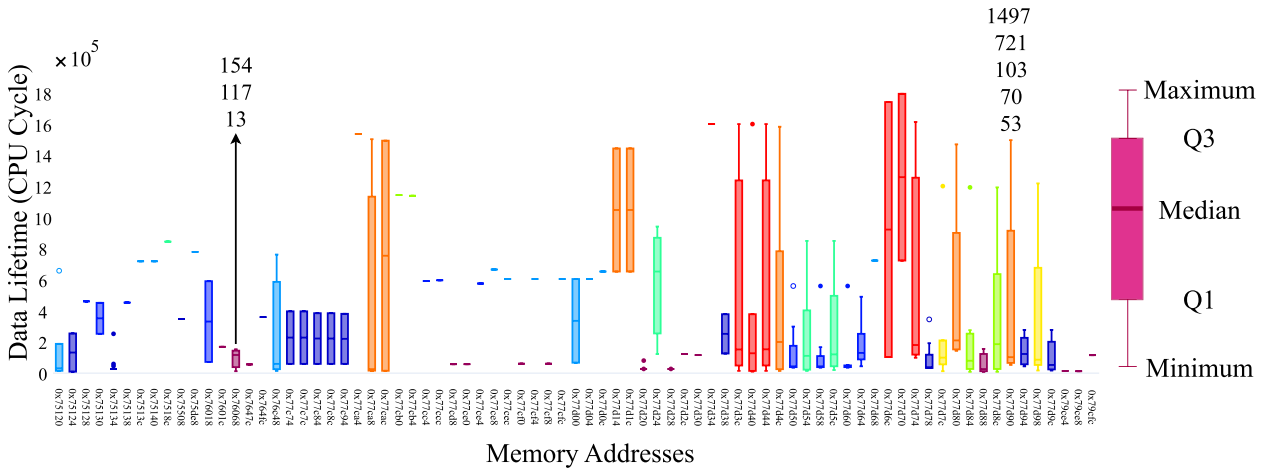

FIGURE 5. Distribution of data lifetimes across memory addresses.

values larger than 17, there is a slight increase in energy consumption as more requests are written with higher Δ values, leading to higher energy consumption per write. It is found that Δ values between 14 and 17 provide an energy-efficient solution.

Some points in Fig. 6 demonstrate a significant decrease in energy consumption, indicating that by slightly increasing the Δ value, a large number of requests can be directed to the low retention section. While this increases the energy required for writing, it is advantageous to store a substantial amount of data in the low-retention section instead of the high-retention section. This is also exemplified by points A and B in Fig. 3. ((8) and (7)).

$$E_A = (E_{w_{\Delta=11.36}} \times \%38.77) + (E_{w_{\Delta=40.3}} \times \%61.23) = (0.1319 \times \%38.77) + (0.6555 \times \%61.23) = 45.25 \quad (7)$$

$$E_B = (E_{w_{\Delta=11.41}} \times \%59.94) + (E_{w_{\Delta=40.3}} \times \%40.06) = (0.1327 \times \%59.94) + (0.6555 \times \%40.06) = 34.21 \quad (8)$$

Selecting a Δ value lower than the maximum is beneficial, but determining the appropriate Δ is crucial (as depicted in Fig. 3 and Fig. 6). Fig. 6 emphasizes the significance of finding the optimal Δ for the problem at hand. Using an incorrect retention time for applications can result in significant energy and power losses. Thus, the essence of determining the suitable retention time becomes evident in ensuring efficient energy consumption and power utilization.

E. MEMORY MAPPING TECHNIQUES

Now, we will move on from discussing the ideal memory architecture to exploring the best mapping strategy.

1) DYNAMIC MAPPING: IDEAL MEMORY ARCHITECTURE + DYNAMIC MEMORY MAPPING (FIG. 2(C))

In previous sections, we assumed that it is possible to differentiate between two distinct data lifetimes within a single memory address and determined the best Δ based on this assumption. From the software perspective, this can be achieved through application-specific mapping, albeit at the cost of programmer/compiler overhead expenses. By

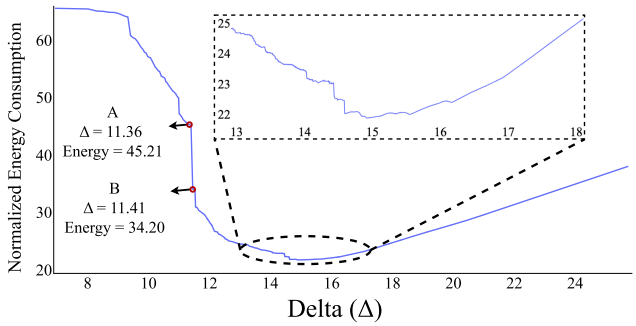

FIGURE 6. Energy consumption in a two-level architecture across selected delta values.

distinguishing between the lifetimes of an address, we mean that we can allocate these lifetimes to different memory sections during the runtime of the application.

For instance, consider Fig. 4 (b), which illustrates how a variable exhibits different lifetimes during the execution of an application. Initially, it has a lifetime of 40 cycles, followed by a lifetime of 60 cycles. Until now, we have assumed that we can assign data lifetime of 40 cycles to the low retention section and data lifetime of 60 cycles to the high retention section. This type of mapping is achievable through dynamic mapping techniques. It becomes apparent that distinguishing between these two lifetimes is challenging. Additionally, it is common for data to have varying lifetimes during the runtime of an application.

Dynamic mapping involves assigning each retention time to the appropriate memory cell. For example, assume a variable during the execution of an application, that requires different lifetimes in the following order: 53, 70, 103, 721, 1497 cycles. Assuming that the low retention section has a retention time of 500 cycles, dynamic mapping can assign the lifetimes of 53, 70, and 103 cycles to the low retention section, while assigning the lifetimes of 721 and 1497 cycles to the high retention section. The energy consumption for this memory address can be calculated by summing the energy consumption of the low retention section with that of the high retention section. The energy calculation in this scenario can be modeled as (9):

$$E_{dynamic} = (E_{w_{\Delta=LowRetention}} \times 3) + (E_{w_{\Delta=HighRetention}} \times 2) \quad (9)$$

In Fig. 7(a), we can see the dynamic mapping scenario where we can selectively assign different data lifetimes of an address to either the low-retention or high-retention sections of memory. In the case of address A, all retention times are within the limit of the low retention section, leading to all requests being written there. On the other hand, for address B, the low lifetimes are assigned to the low retention section, while the high lifetimes are written in the high retention memory.

2) STATIC MAPPING: IDEAL MEMORY ARCHITECTURE + STATIC MEMORY MAPPING (FIG. 2(C))

In dynamic mapping, the memory lookup table needs to be continuously updated as data addresses change during execution. This introduces additional complexity and overhead. On the other hand, with static mapping, we identify the maximum data lifetime for each address and establish a static mapping based on this information. This eliminates the need for runtime address updates and simplifies memory management.

While static mapping may not provide the absolute optimal energy consumption, it represents a practical compromise between the constraints of hardware limitations and the benefits of static mapping. By considering the maximum data lifetime, we can achieve a more efficient memory allocation without the overhead of dynamic mapping.

The concept of static mapping combines the notion of the maximum data lifetime with an ideal memory architecture to achieve optimal results. In this approach, we first analyze the data lifetimes of memory addresses and identify the maximum lifetime for each address by using access pattern analysis. We then perform a static mapping, where each address is assigned to a specific memory cell. Our proposed mapping is based solely on the maximum lifetime of a variable. This means that all requests for that variable are directed to memory cells with a retention time of the maximum lifetime. However, we still need to determine an appropriate Δ value for the low retention section. The energy consumption can be modeled as (10):

$$E_{Static} = (E_{w_{\Delta=HighRetention}} \times 5) \quad (10)$$

In Fig. 7(b), we can see the proposed static mapping, which is used to make decisions based on the longest lifetime of each address. This method is a simple way to solve the mapping problem. It involves assigning data to specific memory areas based on the maximum time they need to be retained. This mapping is a one-time process that can be done either during the compilation of the program or by the programmer manually. Let us take two addresses, A and B, as examples. For address A, all the data lifetimes are within the limit, so all the requests are placed in the section with a low retention time. On the other hand, for address B, some data lifetimes are within the low retention limit, but the maximum lifetime goes beyond it. In this case, all the requests for address B are treated as if they need the maximum lifetime, and they are stored in the non-volatile section. This method simplifies the mapping process and only requires one-time implementation either with the help of a compiler or manually by the programmer.

Based on our observations, it is evident that a new retention time should be considered to optimize and utilize the advantages of the static mapping approach. We need to examine whether the previously determined Δ is appropriate for this scenario or if a different Δ should be used.

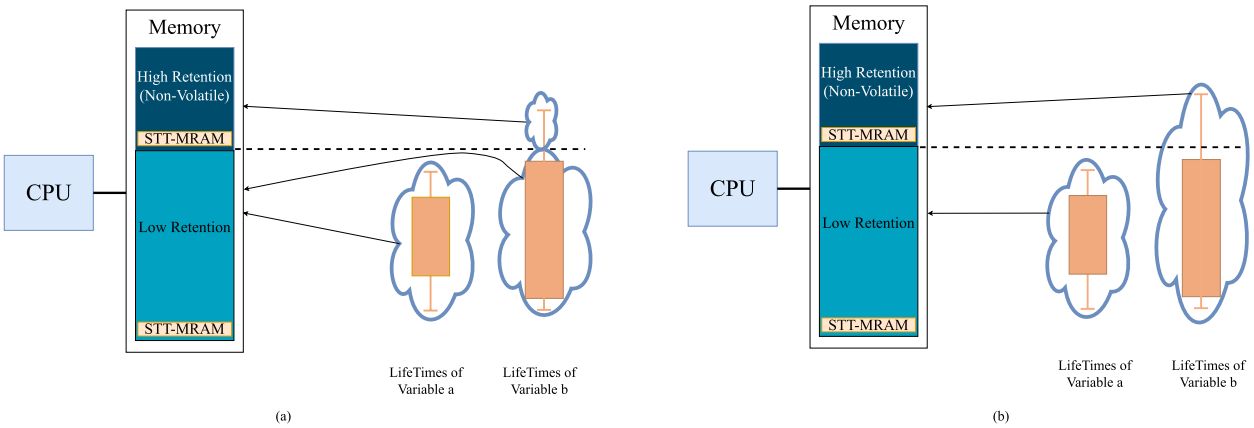

FIGURE 7. Comparing two different mapping (a) dynamic mapping (b) static mapping.

3) TWO-LEVEL STATIC MAPPING(FIG. 2(D))

A comprehensive investigation is conducted on the Δ values, similar to the previous analysis (Fig. 6), to identify the suitable Δ. Fig. 8 illustrates the trend of energy consumption for write requests in the RAM section. It can be observed that Δ values between 15-18 are deemed suitable for this mapping approach. As anticipated, the optimal Δ range is not identical to the previous findings.

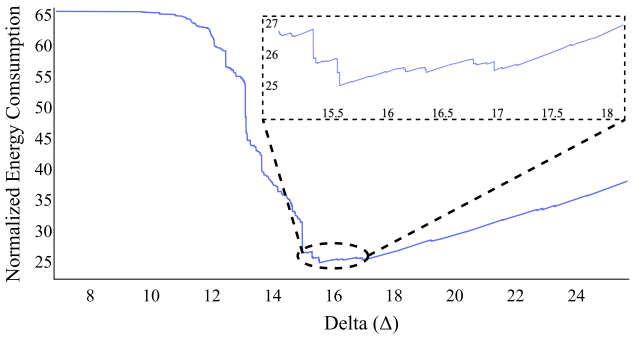

FIGURE 8. Energy consumption at different delta values for Mibench benchmark (two-level architecture, static mapping).

4) STATIC AND DYNAMIC MAPPING

Ultimately, the selection between dynamic and static mapping hinges on the specific demands of the IoT application, essentially a trade-off between high performance, energy efficiency, adaptability, and implementation cost. While dynamic mapping is well-suited for applications requiring exceptional energy efficiency, static mapping offers a practical, low-overhead solution for most scenarios. By defining both dynamic and static mapping, we empower users to choose the most appropriate approach tailored to their application's unique needs.

Dynamic memory mapping, while conceptually appealing, presents several practical challenges that hinder its widespread adoption in IoT device operating systems. These challenges include memory management overheads, increased memory size requirements, and variability in data lifetime across applications.

Implementing dynamic mapping necessitates continuous monitoring of data lifetimes and runtime remapping of memory addresses. This incurs significant overhead through the operating system or compilers, as they need to track access patterns, update mappings, and migrate data. Additionally, dynamic allocation requires provisioning adequate memory capacity in both the high and low retention sections to accommodate potential maximal usage.

Static memory mapping, which allocate data to memory sections based solely on their maximum lifetimes, presents a viable alternative to dynamic mapping. While it incurs a slight increase in energy consumption, typically around 5% on average, static mapping offers substantial reductions in mapping complexity while maintaining the energy efficiency gains offered by multi-retention memory architecture. This approach balances the trade-off of slightly higher energy consumption with the advantages of enhanced reliability and reduced memory management costs. Despite its higher energy consumption compared to dynamic mapping, static mapping remains a preferred choice for a wide range of IoT applications due to its lower complexity compared to other mapping methods. Furthermore, the energy cost associated with this approach is deemed acceptable based on the findings presented in Section V.

F. EXPLORATION OF ADDING RETENTION LEVELS

In seeking to optimize memory architecture, an avenue of exploration is the inclusion of multiple relaxed retention levels. The incorporation of more retention levels provides increased flexibility, enabling a finer mapping of data to memory sections closely aligned with the data's actual lifetime. As mentioned in Section IV-B1, the ideal architecture is the one that includes every feasible level of retention. However, adding more layers has challenges, as detailed in Section IV-B1. Here, we propose a three-layer

architecture as an initial step. In the evaluations, we compare the energy efficiency gains for each design and assess how close we get to the ideal by adding one more layer. Through a meticulous comparison of the two- and three-level architectures with the ideal standard, readers can observe the potential enhancements from introducing additional layers.

1) 3L-DYNAMIC (FIG. 2(E))

In this approach, non-volatile memory can be allocated for code or read-only data, while a relaxed semi-volatile section with $\Delta = 26$ can be utilized for the maximum requirements of working RAM. Additionally, determining the optimal Δ for the third memory section becomes crucial in achieving the desired performance. This design assumes the presence of three memory levels, as depicted in Fig. 2(e).

A comprehensive investigation, similar to the one discussed in Section IV-D, is required for this approach. Through this analysis, we determine that a Δ ranging from 12 to 15 is considered desirable and can be effectively utilized. The trend of changing energy consumption can be explained as demonstrated in Section IV-D.

2) 3L-STATIC (FIG. 2(E))

Similar to the previous section, our proposed 3-level memory architecture can be utilized with static mapping. This allows us to determine the optimal range for Δ. In addition, we address the mapping problem by identifying a suitable Δ range of 15-16.

G. SUMMARY OF PROPOSED ARCHITECTURES AND MAPPINGS

Table 2 provides an overview of the various methods proposed for optimizing the use of STT-MRAM in embedded and IoT nodes. The baseline approach refers to the conventional usage of STT-MRAM without any specific optimization. This serves as a reference point for comparing the effectiveness of the proposed methods.

TABLE 2. Overview of architectures and mappings.

	concept	proposed	proposed
Memory / Mapping	Ideal	2Levels	3levels
Dynamic	Ideal-Dynamic	2L - Dynamic	3L - Dynamic
Static	Ideal-Static	2L - Static	3L - Static

V. SIMULATION SETUP AND RESULTS

Our system is built around the Arm Cortex-M platform, running at a CPU frequency of 100MHz. We used the gem5 full-system simulator, which works with the ARM architecture [46]. We also made sure that each MiBench benchmark could work with the ARM architecture by cross-compiling it for ARM architecture. It is worth mentioning that in the context of IoT and embedded devices, where frequency changes of up to 4-5 times (400-500 MHz) are typical, the impact of frequency changes on finding the optimal delta value is minimal. Table 3 shows the STT-MRAM cell parameters used in the simulation.

TABLE 3. Critical STT-MRAM cell parameters.

Parameter	Value	
Shape	Round	
Anisotropy field (H_k)	80000 A/m	
Saturation magnetization (M_s)	880000 H/m	
Boltzmann constant (K_b)	1.38×10^{-23} J/K	
Temperature (T)	300K	
Free layer thickness	0.9nm	
Vacuum permeability (U_b)	1.25×10^{-6}	
T0	1ns	
Resistance area product (RA)	$10\Omega\mu m^2$	
Δ	Non-volatile	40.62
	High retention	27.18
	Low retention	15.38
TMR	Nominal	200%
	Under bias voltage	160%
R_P	Δ=40.62	3.591kΩ
	Δ=27.18	5.178Ω
	Δ=15.38	9.628kΩ
Volume	Δ=40.62	$\pi\times36n\times36n\times0.9n\times m^3$
	Δ=27.18	$\pi\times30n\times30n\times0.9n\times m^3$
	Δ=15.38	$\pi\times22n\times22n\times0.9n\times m^3$

During our evaluation, we employ a benchmark known as *Average* to assess the performance of all benchmarks. This benchmark serves as an average reference, ensuring that the effects of each benchmark are considered equally. To achieve this, we normalize the benchmarks based on the number of write operations per cycle. This normalization prevents a single benchmark with high computational or write demands from dominating the evaluation process.

After conducting a thorough analysis of the proposed solutions, we present a comprehensive comparison based on the architectures and mapping methods outlined in Table 2. The key takeaway from the comparison is the poor energy efficiency exhibited by the commonly used non-volatile STT-MRAM in the industry. However, by adopting the proposed architectures, significant energy savings can be achieved. In our analysis, we concentrated on minimizing the energy usage for writing data to STT-MRAMs. This decision was based on the fact that writing is the major energy consumer in these memory systems, while read operations have a relatively lower energy consumption [47].

Our investigation revolves around two primary aspects, as detailed in Table 2. Firstly, we explore the impact of incorporating additional levels of retention time into the memory hierarchy. This involves examining the effects of different memory levels on energy consumption. Secondly, we delve into the effects of various memory mapping methods, studying their influence on energy efficiency.

A. EFFECT OF MEMORY ARCHITECTURE

Fig. 9 illustrates how memory leveling impacts the energy consumption of MiBench benchmarks. All data in this figure is normalized with respect to the baseline memory architecture, which corresponds to a non-volatile memory architecture. Notably, the key takeaway is that both the

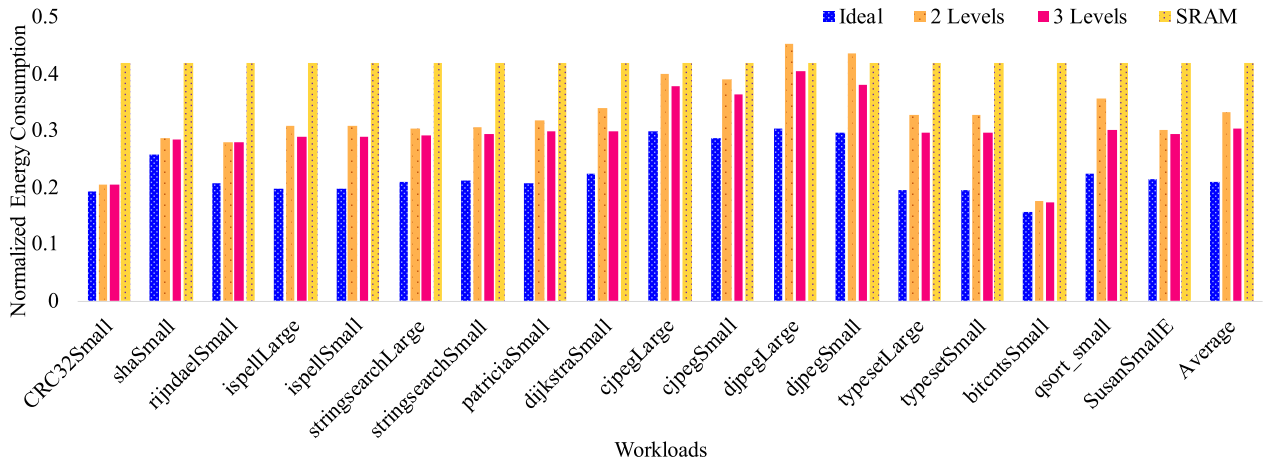

FIGURE 9. Effect of leveling on energy consumption (mapping=static) (normalized to non-volatile).

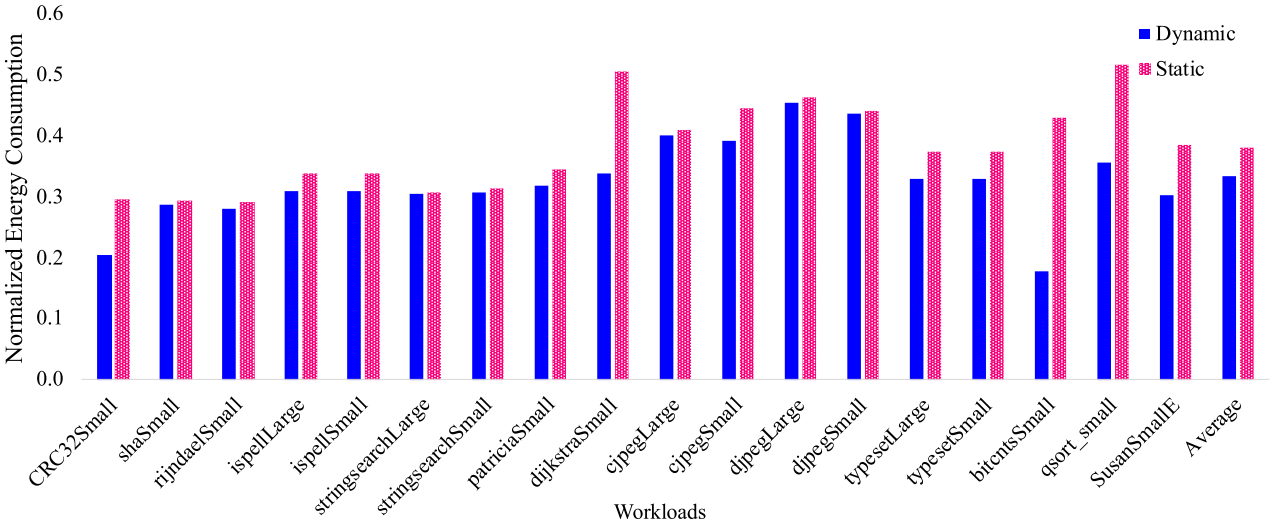

FIGURE 10. Effect of static mapping and dynamic mapping on energy consumption (memory = ideal) (normalized to non-volatile).

two-level and three-level architectures exhibit enhanced energy efficiency compared to the non-volatile memory architecture and are also in close proximity to the ideal memory architecture. Furthermore, the three-level architecture exhibits even greater proximity to the ideal, benefiting from enhanced flexibility. The proposed three-level architecture demonstrates approximately a 14% improvement in energy consumption compared to the previously suggested two-level architecture in [43].

While Flash has higher write energy per bit, its limited writability and erase-before-write requirement make it impractical for working RAM. Therefore, we exclude its energy consumption from this comparison.

When compared to conventional SRAM, the performance of the proposed architectures varies depending on the specific benchmark. For djpeglarge and djpegSmall, the two-level architecture consumes more energy than SRAM, while the three-level architecture consumes less. However, in other workloads, both the two-level and three-level architectures demonstrate improved energy efficiency compared to SRAM. This improvement is attributed to the optimized Δ values in the proposed architectures. The ideal architecture consistently outperforms SRAM due to its precisely tuned Δ values that match the requirements of each workload. On average, the proposed architectures consume less energy than SRAM, and the ideal architecture consistently outperforms all others.

An interesting observation is that *bitcntsSmall* demonstrates the highest energy efficiency across all three architectures, achieving energy efficiencies of approximately 85%, 83.4%, and 83.6% for the ideal, two-level, and three-level architectures, respectively. This indicates that this benchmark benefits the most from the leveling architecture in terms of energy efficiency, closely approximating the energy efficiency achieved by the ideal architecture. Conversely,

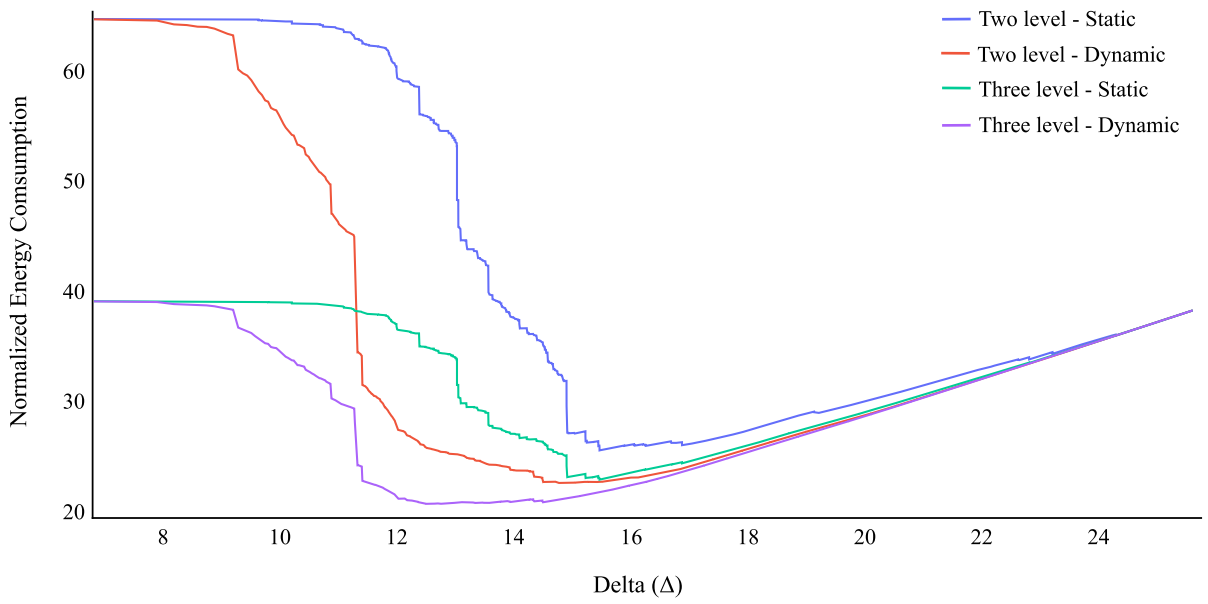

FIGURE 11. All combinations of proposed methods and architectures for different delta.

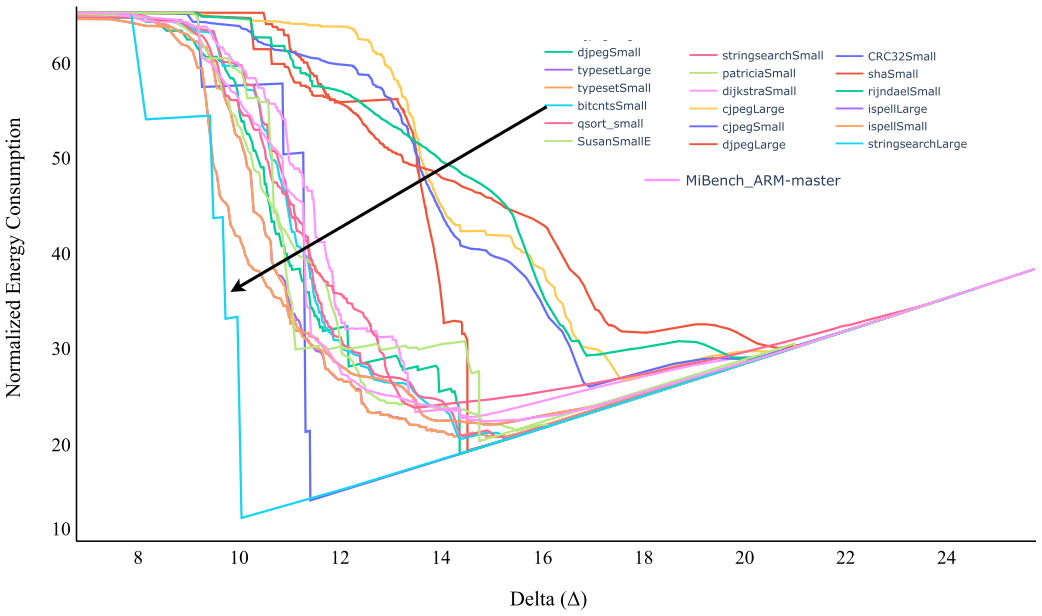

FIGURE 12. Normalized energy consumption for each benchmark in two-level architecture and dynamic memory mapping.

djpegLarge showcases the lowest energy efficiency among the benchmarks, with energy efficiencies of 70%, 55%, and 60% for the ideal, two-level, and three-level architectures, respectively.

CRC32Small and *rijndaelSmall* demonstrate similar energy efficiencies in both the two-level and three-level architectures. This indicates that leveraging the three-level architecture does not lead to significant energy efficiency gains, underscoring the inefficiency of adding an extra layer when factoring in the associated costs. Conversely, considerable efficiency improvements are evident in *djpegSmall* and *qsort_small* when comparing the three-level architecture to the two-level alternative. In these two benchmarks, incorporating an additional level significantly boosts energy efficiency, suggesting a meaningful advantage in energy savings.

Analyzing the average benchmark showcases an improved energy efficiency across all benchmarks with the introduction

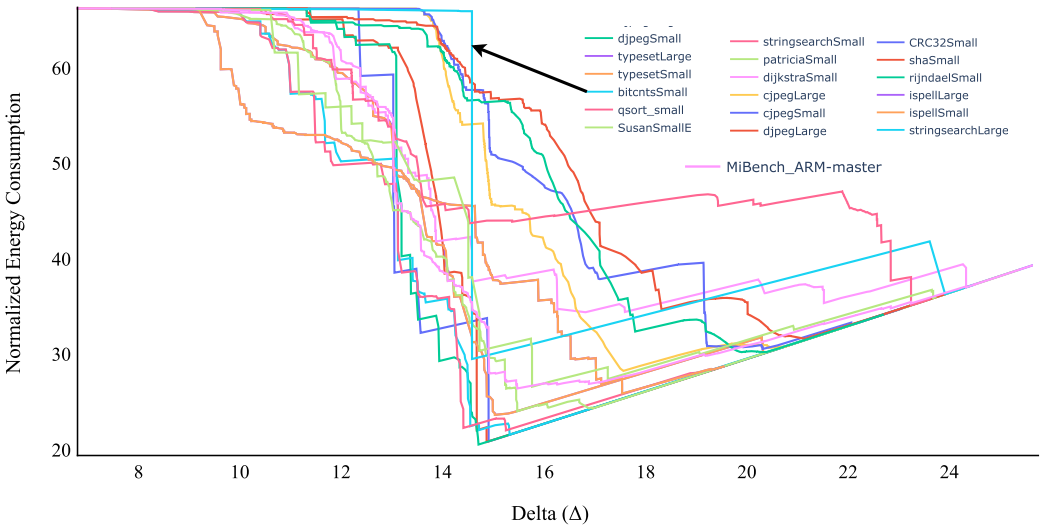

FIGURE 13. Normalized energy consumption for each benchmark in two-level architecture and static memory mapping.

of either a two-level or three-level memory architecture. While the three-level architecture outperforms the two-level, the ideal architecture outperforms both in terms of energy consumption improvements. While greater energy efficiency can be achieved by adding more retention levels, this comes at the expense of increased fabrication costs associated with these additional levels. For instance, the ideal architecture consumes only approximately 21% of the energy used by a non-volatile architecture, while a three-level architecture consumes around 30% of the energy. Beyond three levels, the benefits become less pronounced due to escalating fabrication costs. Notably, a three-level architecture is only around 9% less energy-efficient than the ideal architecture, making it a practical choice for balancing energy efficiency and cost-effectiveness.

B. EFFECT OF MEMORY MAPPING

Fig. 10 illustrates the impact of the proposed mapping methods on energy efficiency. As anticipated, dynamic mapping consistently outperforms static mapping due to its enhanced flexibility. The average benchmark in the figure demonstrates that the energy consumption of the static approach is closely aligned with dynamic mapping. Specifically, for the average benchmark, dynamic mapping achieves 67% energy efficiency, while static mapping reaches 62%, showcasing a reasonable 5% difference that encourages the use of static mapping due to its lower implementation cost compared to dynamic mapping. Static mapping, despite its lower energy efficiency compared to dynamic mapping, offers cost-effectiveness due to its one-time data allocation requirement, either by the programmer or compiler. This straightforward implementation makes it a practical choice for many applications.

Notably, in certain benchmarks such as *bitcntsmall*, there is a significant difference between the energy consumption of static mapping and ideal mapping. This observation indicates that the benchmark contains a significant number of addresses with substantially longer maximum lifetimes compared to other lifetimes for those same addresses. This underscores the high level of variation in address lifetimes within the benchmark.

Conversely, in the *stringsearchLarge* and *djpegSmall* benchmarks, there is minimal disparity between dynamic and static mapping. This indicates that the required retention times for addresses in these benchmarks are closely aligned for each address, with minimal differences between the lifetimes of individual addresses

C. ALL TOGETHER

Fig. 11 provides a visual representation of the energy consumption of the proposed architectures and mappings, with all results normalized to the baseline architecture. The comparison shows that the energy consumption of the minimal section in the 2-level and 3-level architectures is not significantly different. This suggests that there is not a substantial disparity between these two configurations. However, developers aiming to achieve even higher energy efficiency, have the flexibility to request additional retention levels. By utilizing our techniques and mapping approaches, they can further enhance energy efficiency.

Fig. 11 also provides a comparison between static mapping and ideal mapping for each value of Δ. As anticipated, static mapping consistently exhibits higher energy consumption than ideal mapping in both two-level and three-level architectures. This discrepancy arises from the fact that static mapping assigns all data lifetimes of an address to their maximum lifetime, resulting in energy inefficiency. However, both mapping methods share a range of Δ values where their minimum energy consumption levels are close to each other. This range signifies that energy inefficiency of static

mapping is negligible in exchange for the simplicity of using the static mapping. Hence, we select this range of Δ values. Interestingly, the three-level architecture maintains consistent energy consumption throughout both the early and late stages. This is due to the utilization of a low Δ value (6)-(7) in the initial stages, directing a substantial number of requests to the $\Delta = 26$ section. As we transition towards the later stages, the Δ value is progressively increased until it reaches $\Delta = 26$. As a result, all requests are directed to the memory section with $\Delta = 26$.

Table 4 presents the optimal Δ values for each benchmark and for each of proposed mappings and architecture, indicating the best retention time for achieving efficient energy consumption. However, it is worth noting that previous figures have demonstrated that a range of Δ can still yield satisfactory results. For instance, in the case of *crcsmall32*, the optimal Δ is determined as 11, but Δ values around 15 or 17 are also deemed acceptable. Therefore, there is a shared range where different Δ values can lead to satisfactory outcomes. In the case of static mapping, the best Δ is identified as 15, but a range of Δ values between 14 and 16 can be considered suitable.

While we evaluated the proposed architecture using IoT benchmarks, the core principles could be applicable to other domains such as autonomous vehicles. Determining application data lifetimes, exploring suitable STT-MRAM retention times, and static mapping policies could potentially optimize performance and efficiency in other areas. However, the specific optimal thermal stability factors should be recalculated based on the data lifetime characteristics of those applications. More information can be found in the appendix section.

TABLE 4. Calculated delta for different workloads.

Workload	2L-Dynamic	2L-Static	3L-Dynamic	3L-Static
CRC32Small	11.5	15.0	11.5	15.0
shaSmall	14.6	15.0	14.6	15.0
rijndaelSmall	14.4	14.8	14.4	14.8
ispellLarge	15.1	15.1	12.7	15.1
ispellSmall	15.1	15.1	12.7	15.1
stringsearchLarge	15.4	15.4	14.4	14.8
stringsearchSmall	15.3	15.3	14.5	14.5
patriciaSmall	15.6	15.6	13.0	15.6
dijkstraSmall	14.4	16.9	13.6	16.4
cjpegLarge	17.6	17.6	17.6	17.6
cjpegSmall	17.0	20.4	17.0	19.3
djpegLarge	20.7	21.2	17.6	18.4
djpegSmall	19.9	20.4	16.9	17.9
typesetLarge	15.1	17.6	13.9	17.2
typesetSmall	15.1	17.6	13.9	17.2
bitcntsSmall	10.2	14.7	10.2	14.7
qsort-small	13.6	23.3	13.5	14.6
SusanSmallE	14.8	15.9	14.8	15.9
Average	14.9	15.6	12.6	15.6

VI. CONCLUSION

In this paper, we proposed an energy-efficient memory architecture for IoT devices. The key idea was to minimize memory retention time as a means to mitigate the high write energy consumption associated with STT-MRAMs. To achieve optimal performance, it was crucial to implement a suitable address-mapping strategy. To minimize mapping costs, we explored the feasibility of utilizing static mapping as a cost-effective approach. In a nutshell, our proposed architecture consumed less energy than the base model and was near the optimum.

APPENDIX. WORKLOAD SPECIFIC RESULT

Figures 12 and 13 illustrate the normalized energy consumption for each workload in a two-level architecture employing dynamic and static mapping, respectively. It is important to highlight that, for each specific workload, the energy consumption of static mapping exceeds that of dynamic mapping. This observation aligns with the explanation provided in the paper, where it is clarified that, in static mapping, all write requests for a given address are done by the maximum required lifetime among the lifetimes associated with that particular address.

The energy consumption for the *bitcntsSmall* workload presents an intriguing case that merits further discussion. As depicted in dynamic mapping, it attains optimal energy consumption at lower Δ values, gradually increasing afterward. This pattern shows that a significant portion of write requests can be efficiently handled with a Δ value of $\sim$10. However, this principle does not hold in static mapping, where a larger Δ is required to achieve optimal energy consumption. This discrepancy implies that a substantial number of addresses within this workload exhibit a single large lifetime among their lifetimes. In static mapping, due to the necessity of accommodating all write operations within each address using the maximum lifetime, additional energy is expended. Further analyses akin to those conducted throughout our paper can be applied to delve deeper into these observations.

REFERENCES

[1] S. Salehi, H. Farbeh, and A. Rokhsari, "An adaptive data coding scheme for energy consumption reduction in SDN-based Internet of Things," *Comput. Netw.*, vol. 221, Feb. 2023, Art. no. 109528.
[2] H. Imani, J. Anderson, and T. El-Ghazawi, "ISample: Intelligent client sampling in federated learning," in *Proc. IEEE 6th Int. Conf. Fog Edge Comput. (ICFEC)*, May 2022, pp. 58–65.
[3] F. Sanei and H. Farbeh, "A link adaptation scheme for reliable downlink communications in narrowband IoT," *Microelectron. J.*, vol. 114, Aug. 2021, Art. no. 105154.
[4] F. Kochakkashani, V. Kayvanfar, and A. Haji, "Supply chain planning of vaccine and pharmaceutical clusters under uncertainty: The case of COVID-19," *Socio-Economic Planning Sci.*, vol. 87, Jun. 2023, Art. no. 101602.
[5] E. Cheshmikhani, H. Farbeh, and H. Asadi, "3RSeT: Read disturbance rate reduction in STT-MRAM caches by selective tag comparison," *IEEE Trans. Comput.*, vol. 71, no. 6, pp. 1305–1319, Jun. 2022.
[6] M. Rahbari and H. Farbeh, "CRP: Conditional replacement policy for reliability enhancement of STT-MRAM caches," *IEEE Trans. Magn.*, vol. 58, no. 7, pp. 1–13, Jul. 2022.
[7] M. T. Nasab, A. Amirany, M. H. Moaiyeri, and K. Jafari, "Hybrid MTJ/CNTFET-based binary synapse and neuron for process-in-memory architecture," *IEEE Magn. Lett.*, vol. 14, pp. 1–5, 2023.

[8] E. Cheshmikhani, H. Farbeh, and H. Asadi, "A system-level framework for analytical and empirical reliability exploration of STT-MRAM caches," *IEEE Trans. Rel.*, vol. 69, no. 2, pp. 594–610, Jun. 2020.

[9] M. T. Nasab, A. Amirany, M. H. Moaiyeri, and K. Jafari, "High-performance and robust spintronic/CNTFET-based binarized neural network hardware accelerator," *IEEE Trans. Emerg. Topics Comput.*, vol. 11, no. 2, pp. 527–533, Jun. 2023.

[10] D. Apalkov, A. Khvalkovskiy, S. Watts, and M. Krounbi, "Spin-transfer torque magnetic random access memory (STT-MRAM)," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 9, no. 2, pp. 1–35, 2013.

[11] C. W. Smullen, V. Mohan, A. Nigam, S. Gurumurthi, and M. R. Stan, "Relaxing non-volatility for fast and energy-efficient STT-RAM caches," in *Proc. IEEE 17th Int. Symp. High Perform. Comput. Archit.*, Feb. 2011, pp. 50–61.

[12] N. Sayed, L. Mao, R. Bishnoi, and M. B. Tahoori, "Compiler-assisted and profiling-based analysis for fast and efficient STT-MRAM on-chip cache design," *ACM Trans. Design Autom. Electron. Syst. (TODAES)*, vol. 24, no. 4, pp. 1–25, 2019.

[13] A. Shafiee, S. Pasricha, and M. Nikdast, "A survey on optical phase-change memory: The promise and challenges," *IEEE Access*, vol. 11, pp. 11781–11803, 2023.

[14] J. Anderson, E. Kayraklioglu, H. R. Imani, C. Shen, M. Miscuglio, V. J. Sorger, and T. El-Ghazawi, "Virtualizing a post-Moore's law analog mesh processor: The case of a photonic PDE accelerator," *ACM Trans. Embedded Comput. Syst.*, vol. 22, no. 2, pp. 1–26, Mar. 2023.

[15] A. Shafiee, B. Charbonnier, S. Pasricha, and M. Nikdast, "Design space exploration for PCM-based photonic memory," in *Proc. Great Lakes Symp. VLSI*, Jun. 2023, pp. 533–538.

[16] A. M. H. Monazzah, A. M. Rahmani, A. Miele, and N. Dutt, "Exploiting memory resilience for emerging technologies: An energy-aware resilience exemplar for STT-RAM memories," in *Dependable Embedded Systems*. New York, NY, USA: Springer, 2021, p. 505.

[17] R. Bouziane, E. Rohou, and A. Gamatié, "Compile-time silent-store elimination for energy efficiency: An analytic evaluation for non-volatile cache memory," in *Proc. Rapido Workshop Rapid Simulation Perform. Eval., Methods Tools*, Jan. 2018, pp. 1–8.

[18] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "Energy reduction for STT-RAM using early write termination," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design - Dig. Tech. Papers*, Nov. 2009, pp. 264–268.

[19] R. Bishnoi, F. Oboril, M. Ebrahimi, and M. B. Tahoori, "Avoiding unnecessary write operations in STT-MRAM for low power implementation," in *Proc. 15th Int. Symp. Quality Electron. Design*, 2014, pp. 548–553.

[20] S. Yazdanshenas, M. R. Pirbasti, M. Fazeli, and A. Patooghy, "Coding last level STT-RAM cache for high endurance and low power," *IEEE Comput. Archit. Lett.*, vol. 13, no. 2, pp. 73–76, Jul. 2014.

[21] S. P. Park, S. Gupta, N. Mojumder, A. Raghunathan, and K. Roy, "Future cache design using STT MRAMs for improved energy efficiency: Devices, circuits and architecture," in *Proc. 49th Annu. Design Autom. Conf.*, Jun. 2012, pp. 492–497.

[22] J. Jung, Y. Nakata, M. Yoshimoto, and H. Kawaguchi, "Energy-efficient spin-transfer torque RAM cache exploiting additional all-zero-data flags," in *Proc. Int. Symp. Quality Electron. Design (ISQED)*, Mar. 2013, pp. 216–222.

[23] A. Nigam, C. W. Smullen, V. Mohan, E. Chen, S. Gurumurthi, and M. R. Stan, "Delivering on the promise of universal memory for spin-transfer torque RAM (STT-RAM)," in *Proc. IEEE/ACM Int. Symp. Low Power Electron. Design*, Aug. 2011, pp. 121–126.

[24] Y.-T. Chen, J. Cong, H. Huang, B. Liu, C. Liu, M. Potkonjak, and G. Reinman, "Dynamically reconfigurable hybrid cache: An energy-efficient last-level cache design," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2012, pp. 45–50.

[25] G. Sun, X. Dong, Y. Xie, J. Li, and Y. Chen, "A novel architecture of the 3D stacked MRAM l2 cache for CMPs," in *Proc. IEEE 15th Int. Symp. High Perform. Comput. Archit.*, Feb. 2009, pp. 239–249.

[26] J. Ahn and K. Choi, "Lower-bits cache for low power STT-RAM caches," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2012, pp. 480–483.

[27] B. Quan, T. Zhang, T. Chen, and J. Wu, "Prediction table based management policy for STT-RAM and SRAM hybrid cache," in *Proc. 7th Int. Conf. Comput. Convergence Technol. (ICCCT)*, 2012, pp. 1092–1097.

[28] M. Imani, S. Patil, and T. Rosing, "Low power data-aware STT-RAM based hybrid cache architecture," in *Proc. 17th Int. Symp. Quality Electron. design (ISQED)*, 2016, pp. 88–94.

[29] N. Goswami, B. Cao, and T. Li, "Power-performance co-optimization of throughput core architecture using resistive memory," in *Proc. IEEE 19th Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2013, pp. 342–353.

[30] Z. Sun, X. Bi, H. Li, W.-F. Wong, Z.-L. Ong, X. Zhu, and W. Wu, "Multi retention level STT-RAM cache designs with a dynamic refresh scheme," in *Proc. 44th Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2011, pp. 329–338.

[31] M. H. Samavatian, H. Abbasitabar, M. Arjomand, and H. Sarbazi-Azad, "An efficient STT-RAM last level cache architecture for GPUs," in *Proc. 51st Annu. Design Autom. Conf.*, 2014, pp. 1–6.

[32] S. S. Manohar and H. K. Kapoor, "CAPMIG: Coherence-aware block placement and migration in multiretention STT-RAM caches," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 42, no. 2, pp. 411–422, Feb. 2023.

[33] S. Agarwal and S. Chakraborty, "ABACa: Access based allocation on set wise multi-retention in STT-RAM last level cache," in *Proc. IEEE 32nd Int. Conf. Appl.-Specific Syst., Archit. Processors (ASAP)*, Jul. 2021, pp. 171–174.

[34] A. Jog, A. K. Mishra, C. Xu, Y. Xie, V. Narayanan, R. Iyer, and C. R. Das, "Cache revive: Architecting volatile STT-RAM caches for enhanced performance in CMPs," in *Proc. DAC Design Autom. Conf.*, Jun. 2012, pp. 243–252.

[35] F. Arezoomand, A. Asad, M. Fazeli, M. Fathy, and F. Mohammadi, "Energy aware and reliable STT-RAM based cache design for 3D embedded chip-multiprocessors," in *Proc. 12th Int. Symp. Reconfigurable Commun.-Centric Syst.-Chip (ReCoSoC)*, Jul. 2017, pp. 1–8.

[36] Y. Kim, M. Imani, S. Patil, and T. S. Rosing, "CAUSE: Critical application usage-aware memory system using non-volatile memory for mobile devices," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2015, pp. 690–696.

[37] D. Gajaria and T. Adegbija, "Evaluating the performance and energy of STT-RAM caches for real-world wearable workloads," *Future Gener. Comput. Syst.*, vol. 136, pp. 231–240, Nov. 2022.

[38] M. M. Shihab, J. Zhang, S. Gao, J. Sloan, and M. Jung, "Couture: Tailoring STT-MRAM for persistent main memory," in *Proc. INFLOW@ OSDI*, 2016, pp. 1–6.

[39] R. Bouziane, E. Rohou, and A. Gamatié, "Energy-efficient memory mappings based on partial WCET analysis and multi-retention time STT-RAM," in *Proc. 26th Int. Conf. Real-Time Netw. Syst.*, Oct. 2018, pp. 148–158.

[40] H. Jayakumar, A. Raha, J. R. Stevens, and V. Raghunathan, "Energy-aware memory mapping for hybrid FRAM-SRAM MCUs in intermittently-powered IoT devices," *ACM Trans. Embedded Comput. Syst.*, vol. 16, no. 3, pp. 1–23, Aug. 2017.

[41] M. Kim, J. Lee, Y. Kim, and Y. H. Song, "An analysis of energy consumption under various memory mappings for FRAM-based IoT devices," in *Proc. IEEE 4th World Forum Internet Things (WF-IoT)*, Feb. 2018, pp. 574–579.

[42] E. Kültürsay, M. Kandemir, A. Sivasubramaniam, and O. Mutlu, "Evaluating STT-RAM as an energy-efficient main memory alternative," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, Apr. 2013, pp. 256–267.

[43] B. Jahannia, S. A. Ghasemi, and H. Farbeh, "An energy efficient multi-retention STT-MRAM memory architecture for IoT applications," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, early access, 2024, doi: 10.1109/TCSII.2023.3326303.

[44] R. Mertens. (2005). *Lucid Motors Adopted Everspin's Mram in Its Lucid Air All-electric Sedan*. [Online]. Available: https://www.mram-info.com/lucid-motors-adopted-everspins-mram-its-lucid-air-all-electric-sedan

[45] 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. 4th Annu. IEEE Int. Workshop Workload Characterizatio*, Mar. 2001, pp. 3–14.

[46] J. Lowe-Power et al., "The gem5 simulator: Version 20.0+," 2020, *arXiv:2007.03152*.

[47] A. Amirany, K. Jafari, and M. H. Moaiyeri, "High-performance spintronic nonvolatile ternary flip-flop and universal shift register," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 29, no. 5, pp. 916–924, May 2021.

BELAL JAHANNIA received the B.Sc. degree in computer engineering from Shahid Beheshti University, Tehran, Iran, in 2018, and the M.Sc. degree in computer systems architecture from the Amirkabir University of Technology, Tehran, in 2021. His current research interests include computer architecture, memory systems, and the IoT.

SEYED ALI GHASEMI received the B.S. degree in computer engineering (software) from Islamic Azad University, South Tehran Branch, Tehran, Iran, in 2015, and the M.Sc. degree in computer systems architecture from the Amirkabir University of Technology, Tehran, in 2021. His current research interests include processing-in-memory architectures, graph processing, and AI chip.

HAMED FARBEH (Member, IEEE) received the B.Sc., M.Sc., and Ph.D. degrees in computer engineering from the Sharif University of Technology (SUT), Tehran, Iran, in 2009, 2011, and 2017, respectively. He was a member of the Dependable Systems Laboratory (DSL), SUT, from 2007 to 2017. He was with the Embedded Computing Laboratory (ECL), KAIST, Daejeon, South Korea, as a Visiting Researcher, from October 2014 to May 2015, and he has collaborated with the Institute of Research for Fundamental Sciences (IPM), Tehran, as Postdoctoral Fellow, from May 2017 to January 2018. He is currently a Faculty Member of the Department of Computer Engineering, Amirkabir University of Technology (Tehran Polytechnic-AUT), Tehran, where he established the Intelligent Computing and Communication Infrastructure Laboratory (ICCI). He is also a member of the Board of Cyber-Physical Systems Society of Iran (CPSSI). His current research interests include reliable memory hierarchy, emerging memory technologies, AI processors, and cyber-physical systems.