FlowPaP and FlowReR: Improving Energy Efficiency and Performance for STT-MRAM-Based Handheld Devices under Read Disturbance

HAO YAN, University of Texas at San Antonio
LEI JIANG, Indiana University
LIDE DUAN, WEI-MING LIN, and EUGENE JOHN, University of Texas at San Antonio

Handheld devices, such as smartphones and tablets, currently dominate the semiconductor market. The memory access patterns of CPU and IP cores are dramatically different in a handheld device, making the main memory a critical bottleneck of the entire system. As a result, non-volatile memories, such as spin transfer torque magnetoresistive random-access memory (STT-MRAM), are emerging as a replacement for the existing DRAM-based main memory, achieving a wide variety of advantages. However, replacing DRAM with STT-MRAM also results in new design challenges including read disturbance. A simple read-and-restore scheme preserves data integrity under read disturbance, but incurs significant performance and energy overheads. Consequently, by utilizing unique characteristics of mobile applications, we propose FlowPaP, a flow pattern prediction scheme to dynamically predict the write-to-last-read distances for data frames running on a handheld device. FlowPaP identifies and removes unnecessary memory restores originally required for preventing read disturbance, significantly improving energy efficiency and performance for STT-MRAM-based handheld devices. In addition, we propose a flow-based data retention time reduction scheme named FlowReR to further lower energy consumption of STT-MRAM at the expense of reducing its data retention time. FlowReR imposes a second step that marginally trades off the already improved energy efficiency for performance improvements. Experimental results show that, compared to the original read-and-restore scheme, the application of FlowPaP and FlowReR together can simultaneously improve energy efficiency by 34% and performance by 17% for a set of commonly used Android applications.

CCS Concepts: • **Computer systems organization → Embedded systems**; • **Hardware → Non-volatile memory**; **Emerging architectures**;

Additional Key Words and Phrases: Handheld devices, STT-MRAM, main memory, read disturbance, flow-based applications

ACM Reference format:
Hao Yan, Lei Jiang, Lide Duan, Wei-Ming Lin, and Eugene John. 2017. FlowPaP and FlowReR: Improving Energy Efficiency and Performance for STT-MRAM-Based Handheld Devices under Read Disturbance. *ACM Trans. Embed. Comput. Syst.* 16, 5s, Article 132 (September 2017), 20 pages.
https://doi.org/10.1145/3126532

This article was presented in the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES) 2017 and appears as part of the ESWEEK-TECS special issue.
This work is supported by the National Science Foundation, under grant CCF-1566158.
Authors' addresses: H. Yan, Department of Electrical and Computer Engineering, University of Texas at San Antonio, One UTSA Circle, San Antonio, TX 78249, USA; emial: hao.yan@utsa.edu; L. Jiang, Department of Intelligent Systems Engineering, Indiana University, Bloomington, IN 47408, USA; email: jiang60@iu.edu; L. Duan, W. Lin, and E. John, Department of Electrical and Computer Engineering, University of Texas at San Antonio, One UTSA Circle, San Antonio, TX 78249, USA; emails: {lide.duan, weiming.lin, eugene.john}@utsa.edu.

https://doi.org/10.1145/3126532

1 INTRODUCTION

Currently, the semiconductor market is dominated by various handheld devices, such as smartphones and tablets, that have shown fundamental influence on people's daily lives. In 2014, the number of global mobile users reached 1.7 billion, exceeding the number of desktop computer users for the first time in history [4]. This fast growing market has attracted major chip designers from across the world to develop high-performance, low-cost handheld devices.

Current mobile applications running on handheld devices are becoming increasingly computation and data intensive, requiring fast and even real-time manipulation of large amounts of data. Therefore, handheld devices employ a variety of custom hardware components (called IP cores) to offload particular computation tasks from CPUs. These IP cores can achieve significantly improved performance and energy efficiency for the specific functionality they implement. In contrast to latency-critical memory accesses from CPUs, memory accesses from IPs are usually bandwidth-critical, transferring a large number of data frames into and out of memory. Hence, the memory access patterns of CPUs and IPs are dramatically different in a handheld device. Moreover, IPs have strict latency deadlines by which they need to process data frames; violating such deadlines results in dropping entire data frames and thus degrading user experience. Consequently, with increasingly larger numbers of CPUs and IPs incorporated in a handheld device, the shared main memory has become a critical performance and energy bottleneck in handheld systems [2, 16, 38].

Existing memory optimizations in handheld devices [2, 38] have focused on DRAM-based main memories. However, conventional DRAM memories are facing scalability challenges in small feature sizes [13]. This is primarily due to refresh, which is a necessary operation to maintain data in DRAM. As technology scaling reduces the feature size, the changes in DRAM cell capacitance and leakage current decrease the cell retention time. This will in turn result in more frequent refreshes and other complications. Therefore, it is getting increasingly difficult to scale DRAM to have larger capacities.

Consequently, non-volatile memories, particularly spin transfer torque magnetoresistive random-access memory (STT-MRAM), are emerging as a replacement for DRAM [5, 7, 14, 18, 34]. A wide variety of benefits can be obtained in STT-MRAM-based main memories. First, due to its non-volatility, STT-MRAM has almost zero idle power. For the same reason, periodic refreshes, which consume a significant portion of power in DRAM, are not required in STT-MRAM. Second, as opposed to DRAM reads being destructive, STT-MRAM enables non-destructive reads that can lead to various memory optimizations. For instance, it is not needed to write clean data back to memory arrays upon row buffer conflicts in a memory controller. Moreover, STT-MRAM has almost infinite data retention time, and can thereby reboot or wake up a computer system faster and more efficiently than DRAM.

However, employing STT-MRAM as the main memory also results in new design challenges, such as high write overheads [14] and incompatibility with LPDDR interfaces [34]. Orthogonal to these prior studies, this paper investigates a reliability challenge, namely **read disturbance** [35], for STT-MRAM-based memories in deep sub-micrometer regime. Read disturbance characterizes the accidental data corruption that occurs in STT-MRAM after a read operation. It occurs due to the write current approaching the read current in a STT-MRAM cell with increasingly small feature sizes. With technology scaling, read disturbance is inevitable in future STT-MRAM-based main memories [35]. To preserve data integrity under read disturbance, a simple read-and-restore scheme [32] has been proposed to perform a data restore to the same memory address after each read operation. However, this incurs significant performance and energy overheads.

In this paper, we first propose an application flow pattern predictor named **FlowPaP** to remove unnecessary memory restores for STT-MRAM-based handheld devices running mobile applications. It is motivated by the fact that, even in the presence of read disturbance, the restore

associated with the last read before a write can be safely removed. We identify such unnecessary data restores by analyzing and predicting the write-to-last-read distance for each data frame in a mobile application. Since such distances are mostly very short in flow-based applications [20, 21, 38], a large fraction of restores can be removed and the resulting performance and energy efficiency improvements are significant. Experimental results show that, compared to the original read-and-restore scheme, FlowPaP achieves an average energy reduction of 25% along with 8% performance improvement for a set of popular Android applications.

Second, we propose a data retention time reduction scheme named **FlowReR** to further lower energy consumption of STT-MRAM. By adjusting device-level parameters such as the cell free layer thickness, FlowReR improves STT-MRAM energy efficiency at the expense of reducing its data retention time. FlowReR is motivated by the observation that mobile applications exhibit short data write-to-last-read durations that can be easily covered by a significantly reduced data retention time. In order to mitigate the high write latency of STT-MRAM, FlowPaP further marginally trades off the already improved energy efficiency for a much better performance improvement. Collectively, the combination of FlowPaP and FlowReR can simultaneously improve energy efficiency and performance by 34% and 17%, respectively, compared to the read-and-restore baseline for commonly used Android applications.

Note that, although our design uses STT-MRAM for the whole main memory, our proposed FlowPaP and FlowReR schemes can only be applied to a part of it. This includes the frame buffer (which stores a data frame to be shown on the display) and the shared memory portion used by pairs of IP cores to store intermediate data frames. Nevertheless, this comprises the majority of the memory because processing data frames is the primary task for the IP cores in a handheld device. Furthermore, our schemes only target flow-based applications because: (1). more than 80% of the time users spend on a handheld device is in such applications [21]; and (2). the majority of the memory is occupied by these applications to store data frames. Optimizing this important class of applications will significantly improve user experience of the handheld device.

The main contributions of this paper can be summarized as follows:

- We propose using STT-MRAM as the main memory in handheld devices, identifying read disturbance as a major barrier for enabling such systems in deep sub-micrometer regime. By utilizing unique characteristics of flow-based mobile applications, we present convincing motivation studies that address the necessity of our proposed work.
- We propose a flow pattern predictor, namely FlowPaP, to accurately predict the write-to-last-read distances for data frames in handheld devices. As a result, unnecessary data restores can be identified and removed. FlowPaP dynamically switches between two simple prediction heuristics, capturing both the periodic and aperiodic behavior of the running application.
- We propose a flow-based data retention time reduction scheme, namely FlowReR, to significantly improve energy efficiency while still successfully retaining data. FlowReR initially lowers energy consumption by reducing STT-MRAM data retention time, and then trades off the improved energy efficiency for better performance improvements.
- We perform a detailed evaluation of our proposed schemes. FlowPaP turns out to be highly accurate and effective, improving energy efficiency by 25% and performance by 8% over the baseline. Applying FlowPaP and FlowReR together results in an end-to-end energy reduction of 34% and an end-to-end performance improvement of 17%. Finally, the effectiveness of FlowPaP in reducing energy and execution time achieves that of a perfect predictor.

The rest of this paper is organized as follows. Section 2 presents the background knowledge regarding handheld devices, STT-MRAM, and read disturbance. Section 3 and Section 4 provide

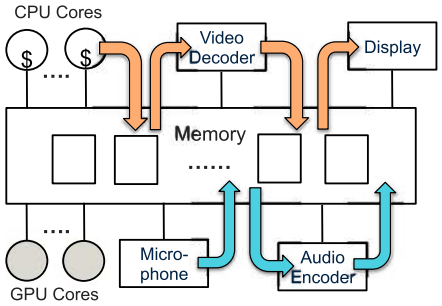

Fig. 1. A handheld platform with two flow examples.

detailed descriptions of our proposed schemes FlowPaP and FlowReR, respectively. Experiments and evaluations are presented in Section 5. Section 6 lists the related work in different aspects. Finally, Section 7 concludes our work.

2 BACKGROUND

2.1 Handheld Devices and Flow Applications

Our target platform in this work is a handheld device, as demonstrated in Figure 1, that contains CPU cores, GPU cores, IP cores, main memory, etc. Current mobile applications running on handheld devices exhibit remarkably different program characteristics than conventional desktop applications, making general-purpose CPUs a poor fit for these applications. As a result, specialized ASIC components, namely IP cores, are integrated in handheld devices to execute certain computation tasks with significantly improved performance and energy efficiency. IP cores can be broadly classified into accelerators and devices: accelerators speed up application intermediate stages, such as audio/video encoding and decoding, image processing, etc.; devices are interfaces to the outside world, and include cameras, display subsystems, etc. These IPs can achieve 10-100x better energy efficiency than general-purpose processors for their specific tasks [3]. A current handheld device typically contains tens of CPUs and hundreds of IPs, most of which need to access the shared main memory. With an increasing number of IPs and increasingly higher user demands, the main memory has become a critical bottleneck of the entire handheld system [2, 16, 38].

Applications running on handheld devices (e.g., YouTube) are usually flow-based [20, 21, 38], processing a massive amount of data frames in a pipelined manner. Figure 1 shows two flow examples in a Skype-like application: in the top flow, data frames are fetched from memory, decoded by a video decoder, and then displayed to users; in the bottom flow, users speak to a microphone that stores audio data to memory, which are then passed to an audio encoder for encoding. A large number of such flows co-exist in a handheld device, dynamically being generated and completed over time. A device component may be involved in multiple such flows, within each it constantly consumes data frames from a producer component and sends the processed data frames to a consumer component. As can be seen, the main memory is heavily used for storing data frames by each pair of components that need to communicate. The main memory performance and energy efficiency are critical to the running application.

2.2 STT-MRAM and Read Disturbance

STT-MRAM is an emerging and increasingly popular non-volatile memory technology that has been widely considered as a universal solution for the cache memory hierarchy in current processors [9]. As opposed to DRAM using electrical charge in capacitors to store data, STT-MRAM takes

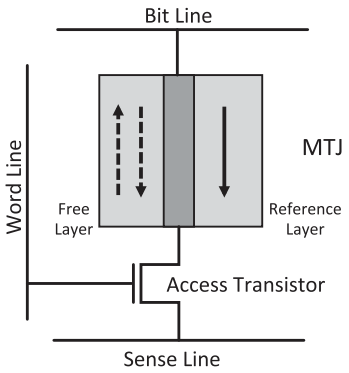

Fig. 2. A cell of STT-MRAM.

advantage of the Magnetic Tunnel Junction (MTJ) resistance for data storage. Figure 2 shows the STT-MRAM cell structure with one access transistor and one MTJ (1T-1MTJ). The MTJ contains two ferromagnetic layers separated by an oxide barrier layer. The magnetization direction is fixed in the reference layer, but can be altered in the free layer. Therefore, as depicted in the figure, the magnetic fields of the two layers can be parallel or anti-parallel, representing the two values of a bit.

STT-MRAM has first appeared as a replacement for SRAM-based low level caches, and recently been considered as a replacement for DRAM in main memory [5, 7, 14, 18, 34]. Kultursay et al. [14] include dirty bits in row buffer to eliminate unnecessary write-backs of clean data to STT-MRAM. Wang et al. [34] tackle the LPDDR pin-compatibility issue in MRAM-based mobile devices. A wide variety of advantages can be achieved in STT-MRAM-based main memories, including almost infinite data retention time, non-destructive reads, and the removal of periodic refreshes. However, adopting STT-MRAM as the main memory also results in new design challenges that need to be addressed, such as high write overheads [6] [14] and incompatible sense amplifiers [34]. Orthogonal to these existing studies, this paper investigates a critical data reliability challenge named read disturbance [35] that turns out to be inevitable and may cause significant performance and energy overheads in STT-MRAM-based memories.

The read and write operations of a STT-MRAM cell are very similar [8]. To read from a cell, a voltage is applied between the bit line and the sense line, creating a small read current through the MTJ to sense the stored bit value. To write to a cell, depending on the data to be written, a positive or negative voltage is applied. The resulting write current has a larger amplitude and a longer duration than the read current, thus changing the magnetic direction of the MTJ free layer to write the data. The write current amplitude of a STT-MRAM cell scales with its MTJ area, thus fast decreasing with the technology node scaling to small feature sizes. The read current amplitude is dependent on the sensitivity of STT-MRAM sense amplifiers, and does not scale down with the feature size and cell area. As a result, the amplitude of the write current approaches that of the read current in small feature sizes in STT-MRAM. Figure 3 [11, 35] demonstrates the read/write current scaling trends in STT-MRAM. As can be seen, these two currents have become too close to each other at 32*nm* and below. Since the read and write operations only differ in the current amplitude, a read operation may be erroneously interpreted as a write operation in a STT-MRAM cell and may accidentally change the data value stored in the cell. This is referred to as **read disturbance**.

In order to preserve data integrity in STT-MRAM under read disturbance, a simple **read-and-restore** scheme [32] has been used. It basically writes data back to the same memory location immediately after reading the data from it. Since read operations dominate memory accesses in most applications, the introduced data restores significantly degrade system performance and

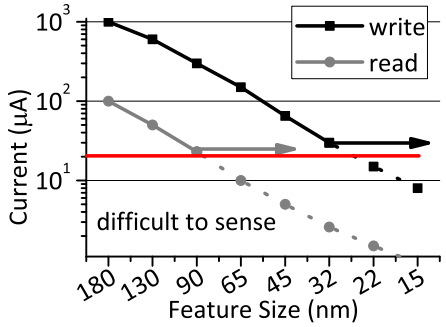

Fig. 3. The read/write current scaling in STT-MRAM. [11, 35].

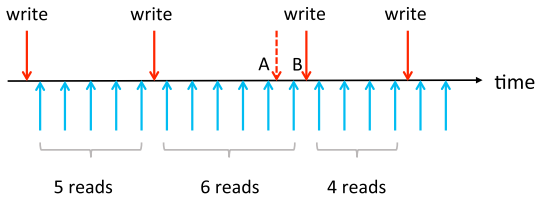

Fig. 4. An illustrative example of data frame writes and reads.

energy efficiency. In this work, we identify and remove unnecessary data restores for STT-MRAM used in handheld devices.

3 FLOWPAP: A FLOW PATTERN PREDICTOR

In this section, we propose FlowPaP, a flow pattern predictor to dynamically predict the write-to-last-read distances for data frames in flow applications running on handheld devices. Upon accurate predictions, unnecessary data restores originally required for preventing read disturbance in STT-MRAM can be identified and eliminated. The resulting performance and energy efficiency improvements are significant because the write-to-last-read distances are usually very short for most data frames.

3.1 Motivation

In a handheld device, a pair of device components that need to communicate is allocated with a shared memory region, within which the producer component periodically writes data frames that are later consumed by the consumer component. For example, a movie player app in Android typically writes to the frame buffer (part of the main memory) with 12 movie frames per second (fps), which are then read out and sent to the display subsystem with a frequency of 60 fps. As illustrated in Figure 4, this results in a write-to-read ratio of 1:5 in the Android frame buffer. In a STT-MRAM-based memory under read disturbance, a data restore (not shown) is needed after each such data frame read except for the last one before a write. Therefore, if we can successfully predict the number of reads before the next write, we can remove 1 out of 5 data restores in this example. FlowPaP seeks to identify such "last read" before a write by analyzing the **"write-to-last-read distance"** for each data frame.

The write-to-last-read distance of a data frame is defined as the number of reads of this frame before the next frame arrives. Ideally, in the above movie player example, this distance is 5 for all data frames. However, dynamically predicting the write-to-last-read distance for a data frame is

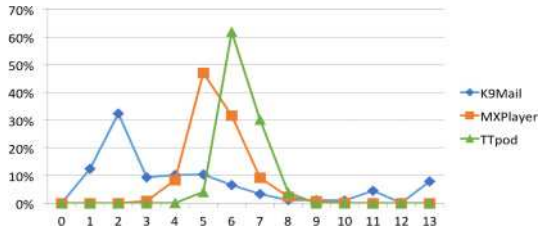

Fig. 5. Percentage distributions of write-to-last-read distances for three mobile applications.

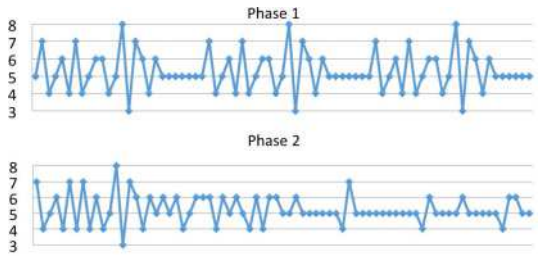

Fig. 6. The write-to-last-read distance variation over time for two program phases.

not straightforward. A data frame can be delayed due to various reasons, arriving at the frame buffer later than expected. For instance, the 3rd frame in Figure 4 arrives at time B rather than its scheduled time A, perhaps due to some emergency task occupying the writing CPU. This leads to a write-to-last-read distance of 6 for the previous data frame and 4 for the next data frame, respectively. Figure 5 shows the percentage distributions of write-to-last-read distances for data frames in three mobile applications. As can be seen, frames can arrive at very different times, resulting in a wide spectrum of write-to-last-read distances.

We make several important observations from our motivational studies. First, data frames running on handheld devices preserve regularity lower than that one may intuitively expect. The resulting write-to-last-read distances for different data frames can span a wide range of values (Figure 5). Second, the write-to-last-read distance for a data frame can vary drastically during application run time, showing periodic and aperiodic behaviors in different application phases. This is depicted in Figure 6, where phase 1 shows periodic behavior, and phase 2 is aperiodic. This motivates our FlowPaP design described in Section 3.2. Consequently, in order to identify the last read before a write and remove its associated restore, a dynamic, accurate prediction of the write-to-last-read distances for data frames is necessary but also challenging.

3.2 The FlowPaP Design

We observe that the runtime write-to-last-read distance variation of data frames exhibits a combination of periodic and aperiodic behaviors (as exemplified in Figure 6). This is because flow-based applications possess a degree of regularity in processing data frames while being influenced by various device components involved in computation. Therefore, we propose two simple heuristics to predict the write-to-last-read distance for periodic/aperiodic program phases, respectively. FlowPaP dynamically combines these two predictors for the runtime write-to-last-read distance prediction.

Slide-by-lag (SBL) for periodic phases. To predict future write-to-last-read distances in application phases showing periodic behavior, we use a simple heuristic:

$$D(n+1) = D(n+1-L) \tag{1}$$

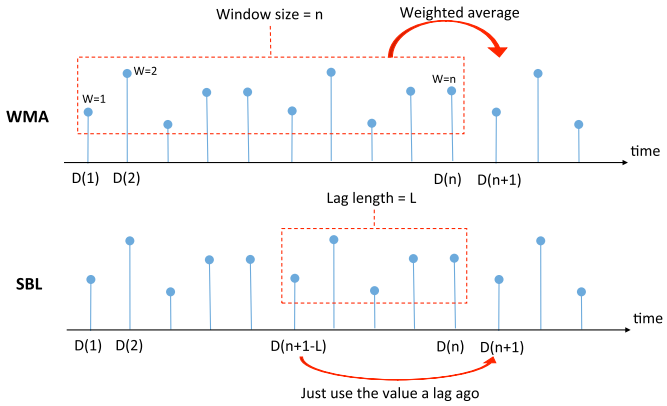

Fig. 7. Weighted-moving-average (WMA) vs. slide-by-lag (SBL).

where $D(n+1)$ denotes the first write-to-last-read distance outside a window of n past distances. Since the current window of write-to-last-read distances demonstrates a periodic repetition of L distances, we simply use the historical distance that is a lag length (L) ago to predict the next distance. The lag length L needs to be carefully chosen to capture the periodic behavior.

Weighted-moving-average (WMA) for aperiodic phases. Following a similar approach presented in [22] for calculating the dynamic data frame length, we propose using the weighted average of the n past distances as the prediction of the next distance when no periodic patterns can be found in a program phase:

$$D(n+1) = \frac{\sum_{i=1}^{n} i \cdot D(i)}{\sum_{i=1}^{n} i} \tag{2}$$

In order to predict $D(n+1)$, Equation 2 assigns a higher weight to a distance that is closer to the next one being predicted. The window size n can be tuned; and the weights are assigned in this way such that more recent historical distances are given higher weights.

These two predictors are visualized in Figure 7. If the current phase is determined to be periodic (bottom), SBL is used to adopt the write-to-last-read distance that is L distances in the past to predict the next distance. If the current phase is aperiodic (top), WMA is used, averaging the n past distances with non-uniform weights to predict the next distance. **Our proposed FlowPaP scheme dynamically switches between SBL and WMA** based on the current application phase being periodic or not. Specifically, we use *autocorrelation* [29], defined in Equation 3, as a metric to determine whether the current phase is periodic or not:

$$AutoCorrelation(n, L) = \frac{AutoCovariance(n, L)}{AutoCovariance(n, 0)} \tag{3}$$

where $AutoCovariance$ is given by:

$$AutoCovariance(n, L) = \sum_{i=1}^{n-L} (D(i+L) - \bar{D}) \cdot (D(i) - \bar{D}) \tag{4}$$

where $\bar{D}$ is the average of all distances in the chosen window. By definition, a higher autocorrelation indicates a closer match between the current window and the window that is a distance of L off; the autocorrelation achieves a value of 1 when these two windows exactly match. Therefore, autocorrelation quantitatively captures the periodic behavior of the running application phase.

Algorithm 1 describes the proposed hybrid scheme FlowPaP. Basically, FlowPaP uses SBL when the autocorrelation of the current window exceeds a predefined threshold th; otherwise, it uses WMA. This process is dynamically applied to predict future write-to-last-read distances during application run time. The window size n, the lag length L, and the autocorrelation threshold th are critical model parameters that need to be determined. When running an application, FlowPaP enters a training mode first, within which it collects a set of write-to-last-read distances, exhaustively testing different value combinations of these parameters (n, L, th) and picking the one that gives the lowest error rate for the training data. After that, FlowPaP enters the prediction mode, using the trained model parameters to predict the future write-to-last-read distances. Note that FlowPaP may choose a different set of model parameters when running a different application, thus capturing the periodicity behavior unique to the application.

ALGORITHM 1: The Flow Pattern Predictor (FlowPaP)

1: **procedure** FlowPaP
2: n, L, and th are determined from the training mode.
3: To predict the next distance $D(n+1)$:
4: **if** $AutoCorrelation(n, L) >= th$ **then**
5: Use the slide-by-lag (SBL) predictor (Equation 1).
6: **else**
7: Use the weighted-moving-average (WMA) predictor (Equation 2).
8: Advance the window by one distance.
9: **goto** line 3.

3.3 Implementation and Error Handling

The implementation of FlowPaP consists of two parts. First, the training and prediction of the algorithm are performed by a CPU since the involved calculations are lightweight tasks for the CPU. Second, the memory controller allocates an 8-bit counter for each frame slot in the memory. When a data frame arrives at this slot, the CPU predicts its write-to-last-read distance and records it in the counter, which will later be decremented by 1 when the frame is read each time. The memory controller issues a restore after each read when the counter is larger than 0, but stops issuing the restore when the counter reaches 0.

The overhead of implementing FlowPaP is negligible. In an Android system with an 800×480 display resolution and 2-byte pixels, a data frame has a size of 750K bytes. Therefore, the hardware overhead of these counters is only $1/750K = 1.3 \times e^{-4}\%$. The performance overhead of running FlowPaP is also negligible. The time and memory complexities of the prediction are both $O(n)$, where n is the window size in the algorithm. Note that n is a small constant after the model is trained. We have run FlowPaP together with other real-world applications in a CPU, and observed almost no difference in system performance and power consumption when FlowPaP is run.

If the predicted write-to-last-read distance is longer than the actual distance, no error handling is needed since we will just lose the opportunity to remove an unnecessary restore. If the predicted distance is shorter than the actual distance, a critical restore is erroneously removed. Error handling in such cases can be carried out at the system level: the CPU processing the data flow will be notified immediately when a write is expected but a read arrives at the frame slot; the CPU will then obtain the previous data frame from the frame's consumer component. Depending on the design requirement, the CPU can choose to either copy back the entire data frame or perform a pixel-by-pixel comparison to detect and correct any pixel corruption. The former is

a simple operation that causes higher data transferring overhead; and the latter involves more computation but reduces network traffic.

4 FLOWRER: A FLOW-BASED DATA RETENTION TIME REDUCTION SCHEME

Despite various advantages, STT-MRAM suffers from high write overheads in both energy and latency compared to DRAM [6]. Techniques have been proposed at both the architecture level [17, 19, 23, 33] and circuit level [37, 39] to mitigate the STT-MRAM write overheads. In particular, several attempts [12, 15, 27, 30] have been made to mitigate the STT-MRAM write overheads by relaxing its non-volatility. Since lowering STT-MRAM's data retention time can reduce its write energy and latency, this section proposes FlowReR, a flow-based retention time reduction scheme to trade off data retention for improved write performance and energy efficiency when running flow applications on handheld devices.

4.1 Motivation

FlowReR is motivated by the observation that mobile applications exhibit short data write-to-last-read durations that can be easily covered by a significantly reduced data retention time. This is because flow-based mobile applications process data frames in a pipelined manner across different device components. As shown in Figure 5, the write-to-last-read distance rarely exceeds 14 in the examined mobile benchmarks. We have conducted a detailed analysis on the memory read and write activities in the entire benchmark suite used in this work, and discovered that not a single write-to-last-read distance is greater than 50. Since Android sends data frames to the display subsystem with a typical rate of 60 frames-per-second (equivalent to 16.7 ms per frame read), the longest write-to-last-read duration for our benchmark suite is less than 1 second. Consequently, FlowReR reduces the data retention time of STT-MRAM used in handheld devices to 10 seconds, allowing more time than the observed 1 second boundary to account for unobserved corner cases. Reducing the data retention time in FlowReR significantly improves the memory system's energy efficiency and performance in handheld devices.

4.2 Reducing Data Retention Time to Improve Write Energy Efficiency

Since write energy is directly related to cell write current, reducing the cell write current (also called switching current) in STT-MRAM is an effective way to lower its write energy consumption. This can be done by reducing either the cell surface area or the cell write current density. As shrinking the MTJ planar area in a STT-MRAM cell has become increasingly difficult with lower feature sizes [30], reducing cell write current density has been a recent focus of intense research. The switching current density J_c is approximately proportional to the switching current density at zero temperature [12, 25, 30, 36], J_{c0}, where J_{c0} is determined by various device-level parameters [30]:

$$J_{c0} = \left(\frac{2e}{h}\right)\left(\frac{\alpha}{\eta}\right)(t_F M_S)\,(H_K \pm H_{ext} + 2\pi M_S) \tag{5}$$

In this equation, t_F is the free layer thickness, M_S is the saturation magnetization, and H_K is the effective anisotropy field; whereas e (the electron charge), h (the reduced Planck's constant), α (the damping constant), η (the spin-transfer efficiency), and H_{ext} (the external field) are all device-level constants. Therefore, in order to reduce J_{c0} (which will in turn reduce J_c and write energy), we can lower device-level parameters such as t_F, M_S, and H_K.

On the other hand, the data retention time of a MTJ characterizes how long the data stored in the MTJ can be retained. It is exponentially proportional to the thermal stability of the MTJ

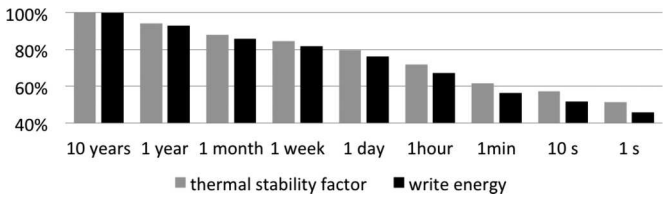

Fig. 8. The thermal stability factor and write energy of a STT-MRAM cell with different retention times. All values are normalized to the case of 10 years.

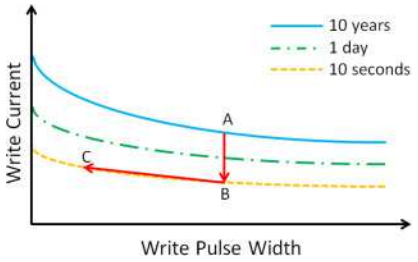

Fig. 9. Variations of cell write current on the write pulse width at different retention times.

[27, 30]:

$$t_{retention} \propto e^{\Delta} \tag{6}$$

where Δ is the thermal stability factor. STT-MRAM is deemed as a non-volatile memory because its data retention time is nearly infinite (e.g., 10 years) with a regular thermal stability factor. The thermal stability factor Δ can be further expressed as [28, 30]:

$$\Delta = \frac{H_K \cdot M_S \cdot A \cdot t_F}{k_B \cdot T} \tag{7}$$

where A is the MTJ planar area, k_B is the Boltzmann constant, and T is the working temperature. Shrinking the cell size or increasing the working temperature would lower the data stability. t_F, M_S, and H_K have been described in Equation (5). If we compare Equation (5) and Equation (7), it can be easily observed that reducing the device-level parameters t_F, M_S, and H_K reduces J_{c0} but also lowers the thermal stability factor Δ. In other words, **we can reduce STT-MRAM write energy at the expense of exponentially reducing its data retention time.**

Prior studies have used a smaller cell surface area [27] or reduced t_F, M_S, and H_K [12, 30] to lower the data retention time in STT-MRAM. In this work, we follow a similar approach that reduces these parameters, e.g., the thickness of the MTJ free layer or t_F. We illustrate the STT-MRAM thermal stability factor and cell write energy at different retention times in Figure 8. Compared to the original retention time of 10 years, FlowReR's target retention time of 10 seconds reduces thermal stability by 42.9% and cell write energy by 48.5%. In our model, the baseline 10-year retention has a thermal stability factor of 40.3 and cell write energy of 0.66 pJ.

4.3 Trading Off Write Energy Efficiency for Reduced Write Latency

The STT-MRAM data retention time reduction scheme described in Section 4.2 greatly reduces the write energy consumption, but has a neutral impact on the performance (cell write time). This is illustrated in Figure 9, which shows the variations of the cell write current on the write pulse width at three retention times: 10 years, 1 day, and 10 seconds. At each retention level, applying a lower switching current takes a longer time (write pulse width) to write to the cell. By fixing the

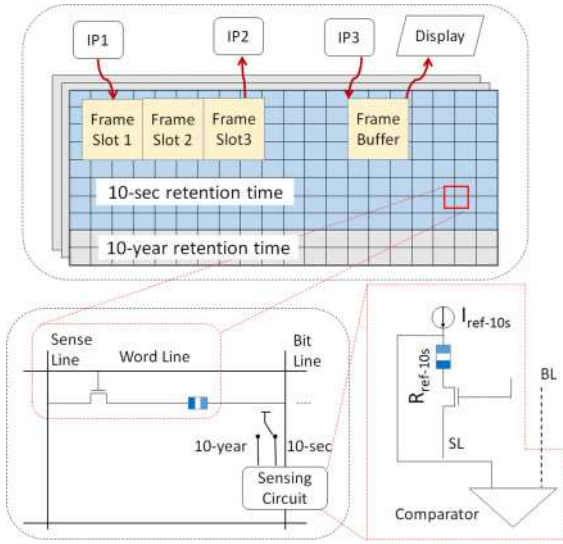

Fig. 10. An implementation of FlowReR.

write pulse width at a certain value (e.g., 10 ns), the technique presented in Section 4.2 reduces the retention time from 10 years to 10 seconds. This can be seen as moving from A to B in Figure 9. A fixed write plus width indicates a neutral impact to performance. Therefore, the performance degradation of STT-MRAM due to high write latency is not mitigated.

In this subsection, we further propose trading off the already improved write energy efficiency for a reduced write latency, by following a similar study performed in [12]. Our approach applies the tradeoff between write current and write pulse width within the same retention time, moving from B to C in Figure 9. Intuitively, while keeping the same data retention time, a higher write current shortens the write pulse width. The shortened write pulse width will in turn reduce the write latency and improve performance. An end-to-end comparison between A and C indicates that FlowReR can simultaneously improve both write energy efficiency and performance. In our experiments (Section 5.3), we will examine multiple points of C along the curve, finding the best tradeoff that can achieve the highest performance improvement with significant energy reduction.

4.4 Implementation and Comparison with Prior Work

As shown in Figure 10, our implementation of FlowReR is a hybrid memory design consisting of a region with a 10-second retention time (for storing data frames) and the rest with a 10-year retention time (for storing CPU data). All data frames are stored in the 10-second region, which covers the frame buffer (containing a data frame to be shown on the display) and the frame slots shared by pairs of IP cores. Data frames are processed by various IP cores in a pipelined manner; at each stage of the flow, a producer IP (e.g., IP1) writes frames into empty slots that are later consumed by a consumer IP (e.g., IP2). Since processing data frames is the primary task for IPs, the 10-second region occupies the majority of the memory. Furthermore, the STT-MRAM cells in this implementation can be reconfigured between the two retention times. At each retention time, the sensing circuit compares the sensed cell current with a reference current to determine the bit value in the cell. Therefore, the 10-second region can be resized by the system to adapt to different applications.

Table 1. Comparisons Between FlowReR and Prior Studies

	[27]	[30]	[12]	FlowReR
Replaced levels	multiple cache levels	L1 and L2 caches	last level cache	main memory
Technique used	reducing cell area	reducing t_F, M_S, or H_K	reducing t_F	reducing t_F, M_S, or H_K
Refreshes needed?	Yes	Yes	Yes	No
Data movement across ret. times?	No	Yes	No	No

Table 2. The Simulated Configuration

CPUs	2GHz, private L1 I/D caches with 32KB/core and 64B lines, 8MB shared L2 with 64B lines.
Memeoy Controller	on-chip, 64-entry transaction queue, 64-entry command queue, close-page row buffer policy, FR-FCFS scheduling.
STT-MRAM-Based Main Memory	1 channel with 2GB capacity, 1 rank-per-channel, 8 banks-per-rank. LPDDR STT-MRAM timing and current parameters are adopted from [34].

A few prior studies [12, 27, 30] have been conducted to mitigate the STT-MRAM write overhead by relaxing its non-volatility. Smullen et al. [27] reduce the MTJ planar area to reduce both write latency and energy. Sun et al. [30] adjust device parameters such as the MTJ free layer thickness and magnetization to generate multiple retention levels across different cache levels. Jog et al. [12] apply an application-driven approach to determine a reduced retention time for the whole last-level cache. Table 1 lists the comparison of several key aspects between FlowReR and these prior studies. In particular, FlowReR has its unique novelties:

- FlowReR utilizes the unique characteristics of flow applications to apply retention reduction in main memory of handheld devices. No refreshes are needed in FlowReR.
- FlowReR applies to the main memory, while the prior studies only targeted SRAM caches.
- FlowReR performs a second step that marginally trades off the already improved energy efficiency for performance improvement.

5 EXPERIMENTS

5.1 Experimental Setup

We use a full-system simulator gem5 [1] to simulate an Android system. A detailed memory model DRAMSim2 [26] has been incorporated into gem5 to simulate the main memory, which adopts various timing constraints and current parameters from a prior work [34] to simulate a reasonable LPDDR3 STT-MRAM-based main memory. The machine configuration used in our simulation is shown in Table 2. To take into account read disturbance, our baseline design models a simple read-and-restore scheme [32] for the entire main memory, whereas the proposed FlowPaP and FlowReR schemes are implemented to the part of memory that stores data frames.

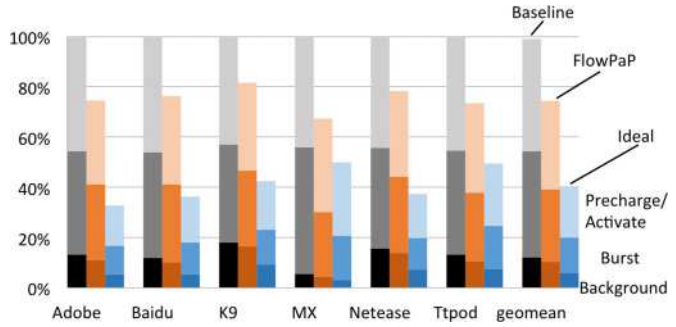

Fig. 11. Energy comparison of different schemes for evaluating FlowPaP.

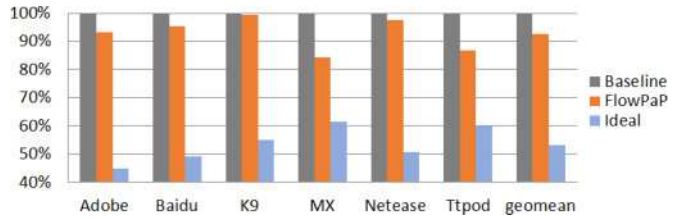

Fig. 12. Performance (execution time) comparison of different schemes for evaluating FlowPaP.

A set of commonly used mobile benchmarks from Moby [10] are evaluated in our experiments. The following is a short description of these popular Android applications:

- *Adobe*: an app for reliably viewing and interacting with PDF documents.
- *BaiduMap*: a mobile map client from China's largest search engine Baidu; similar to Google Maps.
- *K9Mail*: an open-source email client supporting POP3 and IMAP4 protocols.
- *MXPlayer*: a video player that supports various movie formats.
- *NeteaseNews*: a news reader app.
- *TTpod*: a music player for various audio formats.

5.2 Results of FlowPaP

To evaluate FlowPaP, we implement and compare the following schemes:

- Baseline: the baseline scheme that performs a data restore after each read operation to preserve data integrity under read disturbance.
- FlowPaP: our proposed hybrid and dynamic flow pattern prediction scheme that predicts the write-to-last-read distance for each data frame and removes the data restore associated with the last read before a write.
- Ideal: an ideal but unrealistic scheme in which no data restores are performed in the presence of read disturbance. This is provided to demonstrate the impact resulting from the necessary memory restores.

Figure 11 and Figure 12 depict the energy and performance (execution time) comparisons, respectively, among the three schemes. Both metrics favor lower values. In both figures, results are normalized to the corresponding workload's Baseline scheme. Figure 11 shows a breakdown of energy into three portions: precharge/activate, burst, and background. The precharge/activate

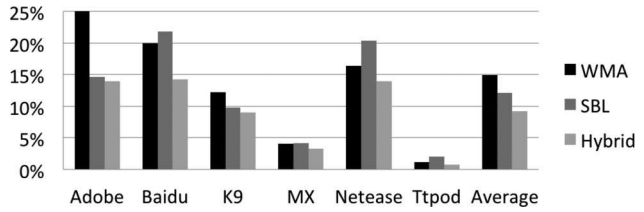

Fig. 13. Prediction error rates of different predictors used in FlowPaP.

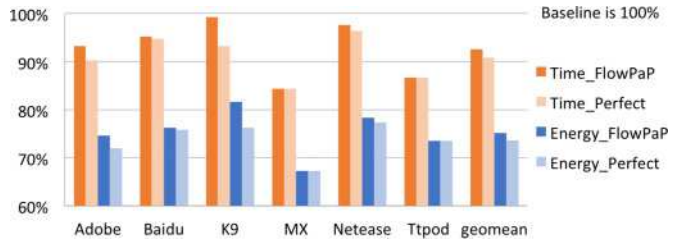

Fig. 14. The effectiveness comparison of FlowPaP and a perfect predictor. Results are normalized to the corresponding workload's Baseline scheme.

energy reflects the dynamic energy consumed by the precharge and activate operations in the memory controller; the burst energy reflects the dynamic energy consumed by the read and write operations in the memory controller; and the background energy is the static energy. Note that the refresh energy is always zero in STT-MRAM because of its non-volatility. As can be seen, FlowPaP significantly improves energy efficiency across all evaluated benchmarks, showing an average energy reduction of 25% compared to Baseline. The majority of the energy reduction is in dynamic energy (burst and precharge/activate) due to the removal of unnecessary restores. The performance improvement, as shown in Figure 12, is about 8% on average (up to 16%) from Baseline to FlowPaP. Compared with energy improvements, performance improvements are smaller because the removed data restores are not on critical timing paths.

Figure 13 compares the prediction error rates for the three prediction schemes described in Section 3.2: WMA (weighted-moving-average), SBL (slide-by-lag), and Hybrid (i.e., FlowPaP). The error rate is defined as: the absolute difference between the actual and predicted distances divided by the actual distance. The two static predictors, WMA and SBL, both show high error rates in some benchmarks. In contrast, FlowPaP consistently performs better than the static ones, achieving an average error rate of 9%. This is because FlowPaP successfully captures both periodic and aperiodic behavior of the running application. As described in Section 3.3, error handling needs to be performed for cases where the predicted write-to-last-read distance is shorter than the actual distance. On the other hand, Figure 14 compares the effectiveness of FlowPaP with that of a perfect predictor. The perfect predictor is capable of correctly predicting the write-to-last-read distance at all times. As shown in the figure, in spite of the 9% error rate, FlowPaP is highly effective and achieves within 3% in both time and energy obtained by the perfect predictor.

5.3 Results of FlowReR

As described in Section 4, FlowReR consists of two steps. First, it reduces the write energy consumption by lowering STT-MRAM's data retention time from 10 years to 10 seconds (Section 4.2). Second, it reduces the write latency by slightly trading off the already improved write energy

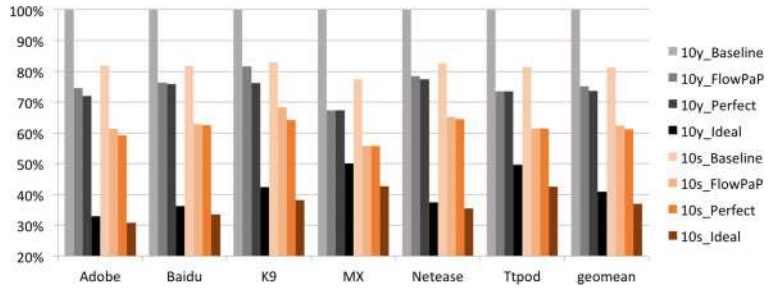

Fig. 15. Energy comparison of different schemes for two retention times: 10 years and 10 seconds. Results are normalized to the total energy of the corresponding workload's Baseline scheme at 10-year retention time.

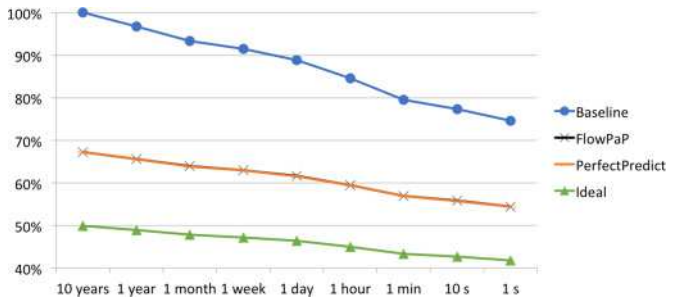

Fig. 16. The energy reduction trends across different retention times for different schemes.

efficiency (Section 4.3). Overall, FlowReR achieves a significant energy reduction with a considerable performance improvement. It is important to note that data refreshes are not needed in FlowReR even with a reduced retention time of 10 seconds. This is because the longest write-to-last-read duration in our evaluated benchmark suite is less than 1 second (Section 4.1). Real-world applications running on handheld devices are expected to have the similar write-to-last-read durations.

Figure 15 shows the energy comparison among different schemes for two retention times: 10 years and 10 seconds. At each retention time, comparison is made among four schemes: Baseline, FlowPaP, PerfectPredict, and Ideal. All these schemes have been described in Section 5.2. We can see that reducing data retention time to 10 seconds significantly reduces the energy consumption. Initially, FlowPaP reduces energy by 25% compared to Baseline; reducing data retention to 10 seconds results in another 13% (a relative 17%) energy reduction. Furthermore, the impact of FlowReR deceases from Baseline to FlowPaP to Ideal. The energy reduction due to FlowReR is a relative 19% (from 100% to 81%) in Baseline, a relative 17% (from 75% to 62%) in FlowPaP, and a relative 10% (from 41% to 37%) in Ideal. This clearly reflects the decreasing trend of number of memory writes (restores) across these schemes. In Figure 16, we further plot the energy reduction trends across different retention times for these schemes. The curves of FlowPaP and PerfectPredict largely overlap due to their similar prediction effectiveness (Figure 14). As can be seen, a continuous reduction of retention time indeed continuously reduces energy to lower levels. The curve of Baseline is steeper than the others because it has the largest amount of memory restores.

The second step of FlowReR trades off the already reduced write energy for a reduced write latency. This is accomplished by moving along the tradeoff curve of write current vs. write pulse

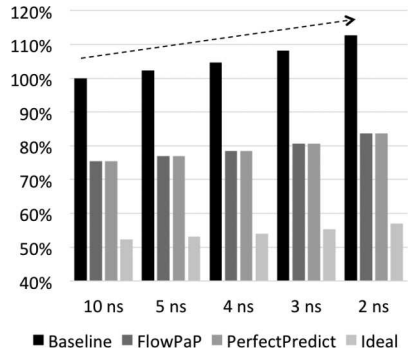

Fig. 17. Energy consumption increases with decreased write pulse widths at the 10-second retention time. Results are normalized to the total energy of *Baseline* with a 10-ns write pulse width.

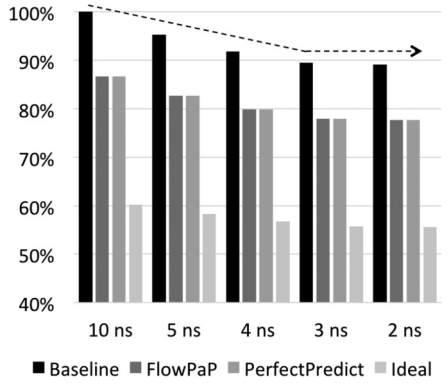

Fig. 18. Execution time decreases with decreased write pulse widths at the 10-second retention time. Results are normalized to the execution time of *Baseline* with a 10-ns write pulse width.

width when the retention time is kept the same (i.e., from B to C in Figure 9). Our experiments here examine different choices of C on the curve, decreasing the write pulse width (but also increasing the write current) to different extents. Figure 17 and Figure 18 demonstrate the variation trends of energy and execution time, respectively, with respect to decreased write pulse widths when the retention time is kept at 10 seconds. As expected, the reduced write pulse width decreases the execution time at the expense of increasing the energy consumption. As shown, while the energy consumption continuously increases approximately at a constant rate, the performance improvement encounters apparent slowdown after reaching 3 ns. Taking into account the trends shown in Figure 9, one can expect that the energy increase will continue and even speed up while the performance improvement will slow down. Therefore, the write pulse width of 3 ns turns out to be a turning point, achieving the best tradeoff between performance improvement and energy increase in our experiment. When reducing the write pulse width from 10 ns to 3 ns, FlowPaP experiences a relative 10% performance improvement together with a relative 6% energy increase.

5.4 Summary of Results

By combining FlowPaP and FlowReR, we summarize the end-to-end improvements in energy efficiency and performance of our work:

- FlowPaP by itself achieves a 25% reduction in energy consumption (Figure 11) and a 8% reduction in execution time (Figure 12), compared to Baseline.
- The first step of FlowReR achieves a relative 17% reduction in energy consumption (Figure 15) and no change in execution time.
- The second step of FlowReR suffers from a relative 6% increase in energy consumption (Figure 17) but achieves a relative 10% reduction in execution time (Figure 18).
- Overall, the end-to-end energy consumption is $(1 - 25\%) \times (1 - 17\%) \times (1 + 6\%) = 66\%$ of Baseline; the end-to-end execution time is $(1 - 8\%) \times 100\% \times (1 - 10\%) = 83\%$ of Baseline.

Consequently, our proposed FlowPaP and FlowReR schemes simultaneously improve energy efficiency and performance, by 34% and 17%, respectively.

6 RELATED WORK

There has been a number of recent studies on handheld platforms. Nachiappan et al. [2] design a heterogeneous memory controller for handheld devices. Yedlapalli et al. [38] shrink data frames to reduce IP-to-IP data reuse distances, and forward a store to the corresponding load to bypass memory. Nachiappan et al. [20] predict data frame processing slacks. Virtualized IP chains [21] are proposed to enable inter-flow scheduling and release CPUs. These studies all work on conventional DRAM-based main memory in handheld devices. In contrast, our work targets the unique challenges in STT-MRAM-based handheld devices.

Read disturbance in STT-MRAM is traditionally mitigated using either a pulsed read scheme [24] or a read-and-restore scheme [32]. Jiang et al. [11] adaptively select one of these two read schemes based on the bank busyness. Wang et al. [35] delay restores to a STT-MRAM L2 cache until the read cache line is evicted from the L1 cache. Sun et al. [31] invent a dual-mode cache architecture that can enable or disable restores based on accuracy requirements. These existing schemes are mostly cache-oriented and application-agnostic, merely focusing on removing restores in restricted cases. In contrast, our FlowPaP scheme utilizes the unique characteristics of flow applications running on handheld devices. Peters et al. [22] use a hybrid prediction scheme similar to FlowPaP to predict the workload length of data frames. In contrast, FlowPaP predicts the write-to-last-read distances for future data frames.

7 CONCLUSIONS

With memory becoming a critical bottleneck in current handheld devices, STT-MRAM is an ideal candidate to replace DRAM in the main memory for a variety of beneficial reasons. However, this also results in an inevitable reliability challenge, namely read disturbance, in future STT-MRAM-based memories. In this paper, we characterize flow-based mobile applications running on handheld devices, proposing two innovative flow-based schemes to improve system energy efficiency and performance. FlowPaP is an effective prediction scheme that identifies and removes unnecessary memory restores originally used against read disturbance; FlowReR is a data retention time reduction time for STT-MRAM to further lower energy consumption and achieve the best tradeoff between energy efficiency and performance improvements. Detailed evaluation results demonstrate the merits of our proposed schemes over prior schemes in both performance and energy efficiency for a set of commonly used mobile applications.

ACKNOWLEDGMENTS

The authors would also like to thank the anonymous reviewers for their invaluable comments and helpful suggestions. The work is supported by the National Science Foundation under Grant No. CCF-1566158.

REFERENCES

[1] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. 2011. The Gem5 Simulator. *SIGARCH Comput. Archit. News* 39, 2 (2011).

[2] Nachiappan Chidambaram Nachiappan, Praveen Yedlapalli, Niranjan Soundararajan, Mahmut Taylan Kandemir, Anand Sivasubramaniam, and Chita R. Das. 2014. GemDroid: A Framework to Evaluate Mobile Platforms. In *SIGMETRICS*.

[3] Jason Cong, Mohammad Ali Ghodrat, Michael Gill, Beayna Grigorian, Karthik Gururaj, and Glenn Reinman. 2014. Accelerator-Rich Architectures: Opportunities and Progresses. In *DAC*.

[4] Danyl Bosomworth. 2015. Mobile Marketing Statistics 2015. (2015). http://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics/

[5] Rajagopalan Desikan, Charles R. Lefurgy, Stephen W. Keckler, and Doug Burger. 2002. On-chip MRAM as a High-Bandwidth, Low-Latency Replacement for DRAM Physical Memories. In *Department of Computer Science Tech Report TR-02-47, The University of Texas at Austin*.

[6] Xiangyu Dong, Xiaoxia Wu, Guangyu Sun, Yuan Xie, H. Li, and Yiran Chen. 2008. Circuit and microarchitecture evaluation of 3D stacking magnetic RAM (MRAM) as a universal memory replacement. In *DAC*.

[7] Qing Guo, Xiaochen Guo, Ravi Patel, Engin Ipek, and Eby G. Friedman. 2013. AC-DIMM: Associative Computing with STT-MRAM. In *ISCA*.

[8] D. Halupka, S. Huda, W. Song, A. Sheikholeslami, K. Tsunoda, C. Yoshida, and M. Aoki. 2010. Negative-resistance read and write schemes for STT-MRAM in 0.13 μm CMOS. In *ISSCC*.

[9] Yiming Huai. 2008. Spin-transfer torque MRAM (STT-MRAM): Challenges and prospects. *AAPPS Bulletin* 18, 6 (2008).

[10] Yongbing Huang, Zhongbin Zha, Mingyu Chen, and Lixin Zhang. 2014. Moby: A mobile benchmark suite for architectural simulators. In *ISPASS*.

[11] Lei Jiang, Wujie Wen, Danghui Wang, and Lide Duan. 2016. Improving Read Performance of STT-MRAM based Main Memories through Smash Read and Flexible Read. In *IEEE/ACM Asia and South Pacific Design Automation Conference (ASP-DAC)*.

[12] A. Jog, A. K. Mishra, Cong Xu, Yuan Xie, V. Narayanan, R. Iyer, and C. R. Das. 2012. Cache revive: Architecting volatile STT-RAM caches for enhanced performance in CMPs. In *DAC*.

[13] Uksong Kang, Hak soo Yu, Churoo Park, Hongzhong Zheng, John Halbert, Kuljit Bains, SeongJin Jang, and Joo Sun Choi. 2014. Co-Architecting Controllers and DRAM to Enhance DRAM Process Scaling. In *The Memory Forum*.

[14] E. Kultursay, M. Kandemir, A. Sivasubramaniam, and O. Mutlu. 2013. Evaluating STT-RAM as an energy-efficient main memory alternative. In *ISPASS*.

[15] Hai Li, Xiaobin Wang, Zhong-Liang Ong, Weng-Fai Wong, Yaojun Zhang, Peiyuan Wang, and Yiran Chen. 2011. Performance, Power, and Reliability Tradeoffs of STT-RAM Cell Subject to Architecture-Level Requirement. *Magnetics, IEEE Transactions on* 47, 10 (2011).

[16] Ye-Jyun Lin, Chia-Lin Yang, Tay-Jyi Lin, Jiao-Wei Huang, and Naehyuck Chang. 2010. Hierarchical Memory Scheduling for Multimedia MPSoCs. In *ICCAD*.

[17] Mengjie Mao, Hai (Helen) Li, Alex K. Jones, and Yiran Chen. 2013. Coordinating Prefetching and STT-RAM Based Last-level Cache Management for Multicore Systems. In *GLSVLSI*.

[18] J. Meza, Jing Li, and O. Mutlu. 2012. A case for small row buffers in non-volatile main memories. In *ICCD*.

[19] Asit K. Mishra, Xiangyu Dong, Guangyu Sun, Yuan Xie, N. Vijaykrishnan, and Chita R. Das. 2011. Architecting On-chip Interconnects for Stacked 3D STT-RAM Caches in CMPs. In *ISCA*.

[20] N. C. Nachiappan, P. Yedlapalli, N. Soundararajan, A. Sivasubramaniam, M. T. Kandemir, R. Iyer, and C. R. Das. 2015. Domain Knowledge Based Energy Management in Handhelds. In *HPCA*.

[21] Nachiappan Chidambaram Nachiappan, Haibo Zhang, Jihyun Ryoo, Niranjan Soundararajan, Anand Sivasubramaniam, Mahmut T. Kandemir, Ravi Iyer, and Chita R. Das. 2015. VIP: Virtualizing IP Chains on Handheld Platforms. In *ISCA*.

[22] Nadja Peters, Sangyoung Park, Dominik F, and Samarjit Chakraborty. 2016. Frame-based and Thread-based Power Management for Mobile Games on HMP Platforms. In *ICCD*.

[23] M. Rasquinha, D. Choudhary, S. Chatterjee, S. Mukhopadhyay, and S. Yalamanchili. 2010. An energy efficient cache design using Spin Torque Transfer (STT) RAM. In *ISLPED*.

[24] A. Raychowdhury. 2013. Pulsed READ in spin transfer torque (STT) memory bitcell for lower READ disturb. In *Nanoscale Architectures (NANOARCH), 2013 IEEE/ACM International Symposium on*.

[25] A. Raychowdhury, D. Somasekhar, T. Karnik, and V. De. 2009. Design space and scalability exploration of 1T-1STT MTJ memory arrays in the presence of variability and disturbances. In *IEEE International Electron Devices Meeting (IEDM)*.

[26] P. Rosenfeld, E. Cooper-Balis, and B. Jacob. 2011. DRAMSim2: A Cycle Accurate Memory System Simulator. *Computer Architecture Letters* 10, 1 (2011).
[27] Clinton W. Smullen, Vidyabhushan Mohan, Anurag Nigam, Sudhanva Gurumurthi, and Mircea R. Stan. 2011. Relaxing non-volatility for fast and energy-efficient STT-RAM caches. In *HPCA*.
[28] Clinton W. Smullen, IV, Anurag Nigam, Sudhanva Gurumurthi, and Mircea R. Stan. 2011. The STeTSiMS STT-RAM Simulation and Modeling System. In *ICCAD*.
[29] Petre Stoica and Randolph Moses. 2005. *Spectral Analysis of Signals*. Prentice Hall.
[30] Zhenyu Sun, Xiuyuan Bi, Hai (Helen) Li, Weng-Fai Wong, Zhong-Liang Ong, Xiaochun Zhu, and Wenqing Wu. 2011. Multi Retention Level STT-RAM Cache Designs with a Dynamic Refresh Scheme. In *MICRO*.
[31] Zhenyu Sun, Hai Li, and Wenqing Wu. 2012. A Dual-mode Architecture for Fast-switching STT-RAM. In *ISLPED*.
[32] R. Takemura, T. Kawahara, K. Ono, K. Miura, H. Matsuoka, and H. Ohno. 2010. Highly-scalable disruptive reading scheme for Gb-scale SPRAM and beyond. In *IMW*.
[33] Jue Wang, Xiangyu Dong, and Yuan Xie. 2013. OAP: An obstruction-aware cache management policy for STT-RAM last-level caches. In *DATE*.
[34] Jue Wang, Xiangyu Dong, and Yuan Xie. 2014. Enabling High-performance LPDDRx-compatible MRAM. In *ISLPED*.
[35] Rujia Wang, Lei Jiang, Youtao Zhang, Linzhang Wang, and Jun Yang. 2015. Selective Restore: An Energy Efficient Read Disturbance Mitigation Scheme for Future STT-MRAM. In *DAC*.
[36] C. Xu, Y. Zheng, D. Niu, X. Zhu, S. H. Kang, and Y. Xie. 2015. Impact of Write Pulse and Process Variation on 22 nm FinFET-Based STT-RAM Design: A Device-Architecture Co-Optimization Approach. *IEEE Transactions on Multi-Scale Computing Systems* 1, 4 (Oct 2015), 195–206.
[37] Wei Xu, Hongbin Sun, Xiaobin Wang, Yiran Chen, and Tong Zhang. 2011. Design of Last-Level On-Chip Cache Using Spin-Torque Transfer RAM (STT RAM). *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* 19, 3 (2011).
[38] Praveen Yedlapalli, Nachiappan Chidambaram Nachiappan, Niranjan Soundararajan, Anand Sivasubramaniam, Mahmut T. Kandemir, and Chita R. Das. 2014. Short-Circuiting Memory Traffic in Handheld Platforms. In *MICRO*.
[39] Ping Zhou, Bo Zhao, Jun Yang, and Youtao Zhang. 2009. Energy reduction for STT-RAM using early write termination. In *ICCAD*.

Received April 2017; revised June 2017; accepted July 2017