WarmCache: Exploiting STT-RAM Cache for Low-Power Intermittent Systems

Noureldin Hassan
University of Central Florida
Orlando, FL, USA

Byounguk Min
Purdue University
West Lafayette, IN, USA

Changhee Jung
Purdue University
West Lafayette, IN, USA

Yan Solihin
University of Central Florida
Orlando, FL, USA

Jongouk Choi
University of Central Florida
Orlando, FL, USA

Abstract

This paper introduces WarmCache, an optimized STT-RAM cache design with relaxed non-volatility, for an energy harvesting system (EHS) to avoid such compulsory misses across power failure. The key insight is that if the retention time of a cache is longer than a power outage period, the cache contents can be preserved, thereby preventing compulsory misses. Based on this insight, WarmCache leverages a STT-RAM cache with reduced thermal stability to preserve non-volatility during power outages while not persisting any data. To mitigate retention failure that may occur in the relaxed STT-RAM cache, WarmCache lets its compiler partition program into a series of regions and conducts region-level error correction. At each region boundary, WarmCache verifies the execution of the region by scrubbing updated cache lines and re-executes it if any multi-bit error is detected therein. For optimization, WarmCache compiler introduces a novel region formation technique that adjusts the size of each region to match the scrubbing interval. This is achieved through region stitching for combining shorter regions and region splitting for dividing longer regions. Our experiments demonstrate that WarmCache manages to avoid compulsory cache misses and improves the performance by 1.3~1.4x on average compared to the state-of-the-art cache design for EHS.

ACM Reference Format:
Noureldin Hassan, Byounguk Min, Changhee Jung, Yan Solihin, and Jongouk Choi. 2025. WarmCache: Exploiting STT-RAM Cache for Low-Power Intermittent Systems. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture (ISCA '25), June 21–25, 2025, Tokyo, Japan.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3695053.3731049

1 Introduction

Energy harvesting system (EHS) has emerged as an alternative to the battery-equipped IoT. Without having to rely on a battery, EHS devices can sustain themselves by exploiting ambient energy from external sources, such as temperature, radio frequency (RF), Wi-Fi, and so on. Thanks to the battery-less, feather-weight, and self-sustaining nature, EHS devices enable a wide range of applications, including wearables, sensors, and implantable medical equipment [5, 12, 60, 70, 71].

However, energy harvesting sources are inherently unreliable, leading to frequent power outages. To mitigate this issue, typical EHS devices employ a non-volatile processor (NVP) that can instantly checkpoint/restore all on-chip data, i.e., volatile registers to/from neighboring non-volatile flip-flops at a power failure/recovery point [11, 87, 92]. Moreover, EHS devices utilize byte-addressable non-volatile memory (NVM), as main memory, where data can survive power failure at the cost of higher latencies. To improve the performance, prior schemes take advantage of a volatile cache with lightweight crash consistency mechanisms [3, 11, 57, 92, 99]. They keep track of cache updates and checkpoint dirty cache lines with registers, marking a recovery point along the way or at the moment of power interruption. Since cache hits prevent costly NVM accesses thanks to temporal and spatial localities, the prior schemes [11, 23, 24, 92, 99] can achieve both crash consistency and better performance compared to no-cache EHS [4, 6, 7, 9, 10, 22, 34, 51].

Nevertheless, there is ample room for improvement in the cache design of EHS devices. The key observation is that they suffer a considerable amount of compulsory cache misses in the wake of each power outage, which is referred to as *cold cache phenomenon*. In other words, since a volatile cache loses all its contents in the event of power failure, they must be brought back to the cache at every reboot time power is restored. While one might suggest using non-volatile (NV) caches to avoid the cold cache phenomenon, NV caches are significantly slower and less energy-inefficient compared to volatile caches [11, 92]. It turns out that this trade-off diminishes the performance advantages of caching.

To solve the cold cache problem, this paper proposes WarmCache, an optimized Spin-Transfer Torque Random-Access Memory (STT-RAM) cache design that relaxes the non-volatility to meet the low-power requirement of EHS. WarmCache exploits two unique characteristics of STT-RAM technology: (1) the thermal stability can be used as a knob to determine the non-volatility, and (2) as the thermal stability decreases, STT-RAM access latency, retention time, and leakage power all decrease. Taking this into account, WarmCache tunes the thermal stability to align the resulting cache retention time with the power outage duration, thereby avoiding the cold cache phenomenon while consuming less energy than NV caches.

Unfortunately, realizing such a relaxed STT-RAM cache in low-power EHS is a daunting challenge because it is prone to retention

ISCA '25, June 21–25, 2025, Tokyo, Japan

ACM ISBN 979-8-4007-1261-6/2025/06
https://doi.org/10.1145/3695053.3731049

failure, which often necessitates expensive reliability support [72, 73]. The state-of-the-art scheme proposes periodic scrubbing to detect and correct any corrupted cache lines by leveraging their associated parity lines with advanced error correction code (ECC) such as ECC-6 [73]. Although this scheme achieves a low failure-in-time (FIT) rate, it does not support crash consistency which is critical for EHS devices. More importantly, the scheme requires a RAID-4 memory configuration that is even an overkill for a power-hungry and resource-constrained EHS.

To this end, WarmCache presents a novel compiler-directed solution that achieves *region-level error correction* for relaxed STT-RAM cache in EHS devices. WarmCache compiler partitions program to a sequence of *idempotent* regions [16] that is side-effect-free and thus re-executable. While program is running, WarmCache detects errors on cache accesses using a lightweight error detection/correction scheme (ECC-1 + CRC-31 code [73]), capable of detecting multi-bit errors but correcting only single-bit errors. If a multi-bit error is detected in clean cache lines, WarmCache treats it as a cache miss and restores the correct data from the NVM. For multi-bit errors in dirty cache lines, WarmCache rolls back to the entry point of the faulty region and re-executes the region to correct the error. Upon successfully completing a region without errors, WarmCache verifies the region by scrubbing dirty cache lines updated in the region and writing them back to the NVM with *region-level persistence*, which guarantees that all cache lines modified in a region are committed before starting the next region. With the help of the region-level persistence and the error correction, WarmCache can achieve both crash consistency and data integrity—despite its relaxed non-volatility.

For optimization, WarmCache introduces a novel region formation technique that adjusts the size of each region to match with a required scrubbing interval. Basically, idempotent code regions, while inherently side-effect-free, may vary significantly in size since they must break antidependent load-store pairs to ensure idempotency. Unfortunately, if a region is too large, WarmCache may fail to scrub cache lines within the scheduled interval due to power failure, thereby increasing the probability of retention failure. This can result in repeated re-execution of the region, causing the lack of forward progress. On the other hand, if a region is excessively small (i.e., too many region boundaries), the scrubbing overhead increases a lot, reducing energy efficiency. To resolve these issues, WarmCache compiler either stitches smaller regions together or splits larger regions without compromising idempotency.

According to our experiments that explore key parameters, e.g., cache size, retention time, NVM technology, clock frequency, and error detection codes, with 17 applications and 3 different capacitors, WarmCache achieves an average speedup of 1.3~1.4x over the state-of-the-art work. Our contributions are as follows:

- First relaxed STT-RAM cache design for low-power EHS.
- Novel compiler-directed STT-RAM retention failure correction for resource-constrained EHS.
- New compiler technique that forms a series of idempotent regions, optimizing performance by stitching together short regions or splitting longer ones.
- Superior performance, i.e., WarmCache outperforms the state-of-the-art by 1.3~1.4x on average.

2 Background and Motivation

2.1 Architectural Model

EHS operates without a battery, instead collecting ambient energy from surroundings into a small capacitor as an energy buffer for computation—an approach known as *intermittent computing [60]*. Due to the battery-less design, the systems suffer frequent power outages and must ensure crash consistency for correct program execution. To address the issue, the state-of-the-arts have used NVM as main memory and introduced NVPs with a just-in-time (JIT) checkpointing protocol. NVPs monitor the voltage level of the capacitor, and checkpoint all volatile architectural states to non-volatile registers—i.e., non-volatile flip-flops (NVFFs) located next to a volatile register file (as shown in Fig. 1)—just before a power outage occurs. When power is back on, the checkpointed states are restored, allowing the program to resume execution from the exact point of recent power interrupt [7–9, 58, 61–63, 65–68, 84, 87, 98].

NVP sets two voltage thresholds in the JIT checkpointing protocol: V_{on} and V_{backup}. NVP reboots only after its capacitor is fully charged to V_{on}, ensuring that sufficient energy is available in a capacitor to run program without an immediate power outage. When the voltage reaches V_{backup}, on the other hand, NVP checkpoints all volatile states by storing register values in NVFFs before entering sleep mode. Note that the voltage gap between V_{on} and V_{backup} must be carefully set to ensure that NVP has enough energy to make forward progress in one capacitor charge cycle without compromising program correctness or system reliability [11, 92, 99].

2.2 Cache in Energy Harvesting Systems

Recent studies have explored volatile cache designs in NVPs. A key challenge lies in addressing crash consistency while achieving high energy efficiency. Since volatile caches lose their data across power failure, it is critical to back up and restore the cache contents for crash consistency in a manner that minimizes energy consumption.

Previous NVP schemes explore several approaches to handling the crash consistency issues caused by integrating with a volatile cache. A direct approach is using *volatile Write-through Cache* shown in Fig. 1(a). This exploits a conventional SRAM cache with a write-through policy, which inherently supports crash consistency by ensuring data being stored is synchronized in both the cache and the NVM. However, the write-through cache significantly degrades the performance, since each store cannot commit until the data persisted in NVM through the cache.

To address the performance issue, prior works introduced a *Non-volatile SRAM Cache (NVSRAM)* that couples write-back SRAM cache with a backup NVM counterpart, as shown in Fig. 1(b) [24, 27, 57, 58, 90, 91]. They ensure crash consistency by flushing all required contents of the SRAM to the NVM counterpart upon power failure—which can be done by exploiting the JIT checkpointing. While this approach benefits from the write-back cache policy, the NVM counterpart can be underutilized. More critically, securing a significant amount of energy for the failure-atomic SRAM checkpointing, which cannot be spent on program execution, further limits the overall energy efficiency.

To enable a volatile cache without requiring such NVM counterpart, Zeng et al. recently introduced *ReplayCache*. As shown in Fig. 1(c), ReplayCache obviates the need for the NVM counterpart

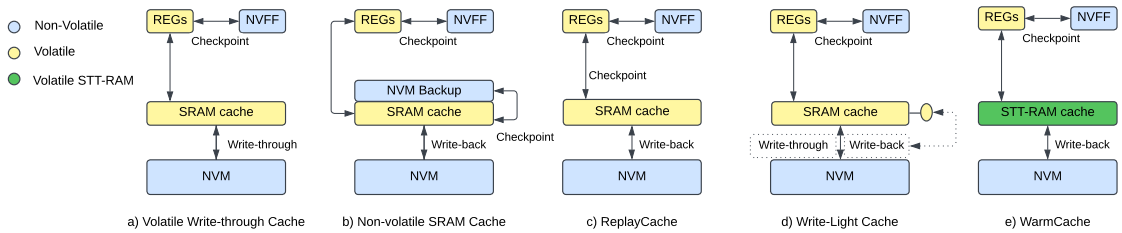


Figure 1: Design comparison of cache architectures in NVP.

of the SRAM cache owing to its novel compiler-directed crash consistency support [92]. The compiler divides program into a series of regions and generates their recovery slice in case they encounter power failure. In the wake of the failure, ReplayCache executes the recovery slice of the interrupted region and persists their potentially lost cache lines to ensure consistent memory states across the failure. This software-based solution can minimize the hardware complexity and power consumption typically associated with the NVM counterpart in the cache design.

More recently, Choi et al. proposed *Write-Light Cache (WL-Cache)*, a performance-oriented cache adaptation scheme [11], shown in Fig. 1(d). This scheme introduces a novel cache write policy that combines the benefits of both write-back and write-through approaches. WL-Cache dynamically switches between behaving as a write-through cache when the power source is weak and as a write-back cache when the power source is strong. This allows WL-Cache to adjusts its behaviors according to the energy harvesting quality, though it varies over time in one way or another.

2.3 Cold Cache Phenomenon

This paper identifies a critical challenge that affects all prior works: the "Cold Cache Phenomenon." This occurs when a memory operation is requested for the first time while all cache lines are empty, resulting in compulsory misses. They are particularly expensive because data should be retrieved from NVM main memory to fill the cache (to be "warm"). In a sense, a cache is considered "cold" when it is empty, and accessing data in this state causes "cold cache misses." We found that, in energy harvesting environments, power outages cause all volatile cache data to be lost, leading to frequent cold cache misses. In particular, if the energy source is weak, the resulting frequent outages exacerbate this issue, undermining the performance gain typically benefited from using a cache. Unfortunately, while NVSRAM—which restores data from NVM to SRAM in the wake of an outage—and non-volatile (NV) caches can prevent cold cache misses by preserving data during power outages, they incur higher energy and latency costs. Consequently, there is a compelling need to redesign the cache architecture so that EHS devices can effectively address the cold cache problem without compromising crash consistency guarantee.

2.4 Volatile STT-RAM as an Answer for the Cold Cache Phenomenon

STT-RAM is one of the most promising emerging NVM technologies [17, 30, 32, 43, 73, 76, 82]. To achieve non-volatility, STT-RAM uses data storage cells built with magnetic tunnel junctions (MTJ), comprised of two ferromagnetic layers separated by an oxide layer. Data is stored by fixing the direction of one layer (the reference layer) and altering the other (the free layer) through an electric current. STT-RAM offers not only long data retention, making it suitable as non-volatile cache, but also low access latency comparable to SRAM [79]. Despite the attractive features, it is challenging to adopt STT-RAM cache in EHS devices due to its high power leakage for cache writes.

One way to reduce write energy is relaxing the non-volatility, i.e., the data retention time [18, 82]. As the retention time decreases, the power leakage during writes also decreases, thereby reducing the overall write energy. The STT-RAM cache retention time is modeled as: $t = \frac{1}{f_0} e^{\Delta}$, where f_0 is the *thermal attempt frequency* fixed at 1 GHz [18, 35]; Δ is the *Thermal Stability Factor* given by the equation: $\Delta = \frac{M_s H_k V}{k_B T}$, where M_s is the saturation magnetization, H_k is the effective anisotropy field, V is the effective activation volume, k_B is the Boltzmann constant, and T is the working temperature. It is possible to reduce the retention time by reducing Δ. Even with the relaxed non-volatility, STT-RAM cache can still act like non-volatile cache for EHS, provided its power-off time is shorter than the retention time.

2.5 Challenges of STT-RAM

While such relaxed STT-RAM reduces write energy, it introduces a challenge, i.e., retention failure. Unlike DRAM, where retention failure results from charge loss [1], STT-RAM retention failure occurs due to random thermal noise flipping the direction of the ferromagnetic layers. This gives STT-RAM retention failure a stochastic nature, similar to soft errors caused by cosmic rays; the probability of such retention error for a single cell (P_{cell}) in a period t_s is: $P_{cell} = 1 - e^{-\frac{f_0}{e^{\Delta}} t_s}$ [72]. To address the retention failure issue, prior works have introduced DRAM-style refresh policies [82, 83]; however, they are ineffective in addressing retention failure in STT-RAM since simply reading the same value and rewriting it to a cell cannot tolerate the failure when the cell flips within the refresh interval [17, 73].

There are several solutions for retention failure as shown in Table 1. A more effective policy for STT-RAM is to conduct periodic scrubbing with a strong ECC per cache line [17, 72]. It is common to use the ECC capable of correcting 6 bit errors (ECC-6) for high reliability. At each scrub interval, every line in the STT-RAM cache is checked for errors and, if necessary, corrected using the ECC. While this method is effective, it has two critical disadvantages.

	FIT Rate	Hardware Cost	Software Support	Memory Overhead	Power Overhead	Persistency	EHS Applicability
ECC-6 [17, 72]	0.092	High	No	High	High	Strict	No
Sudoku [73]	1.05×10^{-4}	Medium	No	Medium	High	Strict	No
STAIR [30]	0.99	High	No	High	High	Strict	No
WarmCache	9.71×10^{-12}	**Low**	**Yes**	**Low**	**Low**	**Relaxed**	**Yes**

Table 1: Comparison of prior STT-RAM cache reliability solutions.

First, it requires to scrub at high frequency when the retention time is short, leading to a considerable scrubbing overhead. Second, it is not feasible for EHS, which suffers frequent power outages, making high-frequency scrubbing across outages unaffordable. For a reasonable scrubbing frequency, stronger ECC is necessary, resulting in additional latency and storage costs.

Nair et al. introduced Sudoku [73], which seeks to reduce the overhead of ECC by using a combination of lightweight ECC (i.e., ECC-1) along with a multi-bit error detection code (i.e., CRC-31). This approach relies on a RAID-4 configuration to exploit redundancy for correcting multi-bit errors when they are detected. While Sudoku offers a more efficient scrubbing mechanism, it is impractical to realize the approach for EHS since its additional power leakage and hardware cost would be an overkill. Moreover, when multiple errors occur simultaneously across redundant lines, further error correction methods are required, potentially increasing latency and complexity.

Another prior work, STAIR [30], proposed a variable ECC scheme using two codes: a lightweight Single Error Correction-Double Error Detection (SEC-DED) code for each cache line, and an extra strong Double Error Correction-Triple Error Detection (DEC-TED) code for dirty pages, dynamically allocated with the help of a detected ECC generator module. STAIR employs a Hybrid Multi-Level Cache Architecture (HCA), using underutilized pages in a lower-level cache to store the extra ECC. If no such pages are available, STAIR evicts a block from the STT-RAM to store the additional ECC. While this approach provides reasonable reliability, it has several downsides: 1) DEC-TED has a high memory overhead (79 bits) that takes space from the cache; 2) it works only for HCA systems where STT-RAM is not the first level cache; 3) STAIR also uses a table to keep track of the location of the extra ECC which adds to the overhead of error correction; and 4) it is only capable of detecting up to 3-bit errors.

Overall, prior works require expensive hardware support to achieve a low FIT rate, strictly persisting all cache data as in the NV cache. Although they work well for general-purpose computer architecture, their power consumption is too significant to satisfy the low-power requirement of energy-sensitive EHS devices.

3 WarmCache Design

WarmCache realizes a volatile STT-RAM cache with compiler-directed error correction. WarmCache is built on a volatile STT-RAM cache that holds all cache lines during power outages as shown in Fig. 1(e), preventing compulsory cache misses. Unlike a traditional volatile STT-RAM cache backed by strong ECC, WarmCache employs a lightweight ECC mechanism combined with compiler support for both retention failure correction and power failure recovery. The rest of this section describes the organization of WarmCache (Sec. 3.1), the error correction mechanism (Sec. 3.2), the power failure recovery mechanism (Sec. 3.3), and the design space exploration (Sec. 3.5).

3.1 WarmCache Organization

For power efficiency of EHS, WarmCache employs low-cost error detection support in hardware while offloading error correction to compiler support—rather than resorting to power-demanding hardware mechanisms like ECC-6 and RAID-4 [73]. Thus, WarmCache can handle errors efficiently without paying for the significant energy cost of traditional hardware-based solutions. In particular, WarmCache seeks to ensure reliability and error correction with minimal hardware modifications, so that it becomes suitable for environments with frequent power outages and limited resources.

To achieve this, WarmCache combines low-cost ECC-1 of size 10 bits for single-bit error correction and *cyclic redundancy check* [77] of size 31 bits (CRC-31) for multi-bit error detection as shown in Fig. 2. There are three reasons behind this choice of error detection codes: 1) ECC-1 is light and efficiently corrects single-bit errors; 2) after using ECC-1, the probability for multi-bit errors is relatively low, which makes the need for their correction relatively rare; 3) more importantly, CRC-31 can detect multi-bit errors (e.g., 7-bit errors [41]) in a single clock cycle; CRC-31 is useful as a lightweight error detection code for multi-bit errors, avoiding the cost of correcting the cache line. Both codes add 1 cycle latency for detection and correcting single-bit and 40 pJ for encoding and decoding the codes [73].

With this support, WarmCache fetches both ECC-1 and CRC-31 along with the data line for read and write operations. For any single-bit error, WarmCache directly corrects them by using ECC-1. For multi-bit errors, however, WarmCache first checks if the line is faulty by verifying its CRC syndrome within a single cycle during read and write operations [73]; the writes are typically performed as two sequential read-modify-write operations. Note that STT-RAM cache usually employs a read-modify-write scheme to lessen the number of bit-flips and reduce write power and latency [40, 73, 74].

Specifically, if the line is found to be faulty (i.e., multi-bit error), WarmCache treats it as a cache miss, provided that the line is not dirty, and retrieves the corresponding block to fill the line. On the other hand, if the dirty line is faulty, WarmCache cannot simply disregard it, as doing so may lead to memory inconsistency. In case multi-bit errors corrupt a dirty cache line, WarmCache incorporates compiler support for their lightweight correction as shown in the next section.

3.2 Region-level Error Correction

WarmCache offloads the responsibility of multi-bit error correction in dirty cache lines to compiler support. For this purpose, WarmCache introduces a *region-level error correction* mechanism

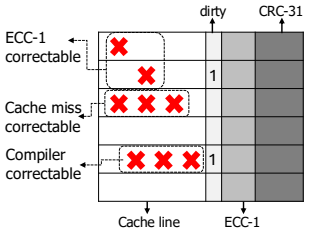


Figure 2: WarmCache's STT-RAM cache organization: WarmCache utilizes ECC-1 for single-bit error correction and CRC-31 for multi-bit error detection, treating multi-bit errors as cache misses or power failure depending on whether the cache lines are dirty.

for relaxed STT-RAM in EHS devices. The key idea is that if a region of code is idempotent, i.e., it can be re-executed multiple times without altering the outcome, then any data of the region which are corrupted by multi-bit errors in dirty cache lines can be corrected by re-executing the faulty region. Based on this insight, when multi-bit errors are detected in dirty cache lines, WarmCache re-executes the faulty region to correct the errors, as if there are power outages, without breaking the program correctness.

To implement this, WarmCache compiler constructs a series of idempotent regions by placing boundaries between anti-dependent load-store pairs and inserting checkpoint stores within each region [14, 15]; for the checkpoint stores, WarmCache compiler checkpoints live-out registers [9, 10]. This ensures that each region is self-recoverable in the event of multi-bit errors. Further details about the compiler support can be found in Sec. 4.

In a sense, WarmCache shifts the multi-bit error correction overhead to a region re-execution penalty. This strategy is effective because the cost of re-executing a region is minimal compared to the high power demands of conventional ECC hardware, which is particularly expensive for EHS, as discussed in Sec. 2.5. Given that the occurrence of multi-bit errors is extremely rare, typically below 1%, this re-execution-based error recovery mechanism is often not triggered. As a result, WarmCache eliminates the need for costly hardware solutions like ECC-6 or RAID-4, while still ensuring high reliability and efficient error correction. This approach strikes an optimal balance between minimizing energy consumption and providing robust error resilience, making it well-suited for low-power, resource-constrained EHS devices.

Verification. While WarmCache verifies cache lines for every read/write operation using ECC-1 and CRC-31, it also scrubs dirty cache lines in two key scenarios to ensure region-level error correction. First, scrubbing is performed at reboot to ensure the program resumes without encountering errors in dirty cache lines. Second, scrubbing is required at region boundaries to ensure program correctness with idempotent processing. This is mainly because all anti-dependent store operations within a region must be validated and written back to NVM before transitioning to the next region. Failing to do so could violate idempotency, leading to memory inconsistencies across regions [14, 15]. To address these challenges, WarmCache employs *region-level persistence* [21, 25, 37, 38, 92, 94, 96, 99, 100] with region-level error correction.

Discussion. STT-RAM can face three types of errors [72]: read error, write error, and retention failure. WarmCache can detect and correct all types of errors. Any errors detectable by ECC-1 and CRC-31 can be corrected by either ECC or re-execution unless a more than 7-bit error occurs. However, the probability of the case is $7.11 * 10^{-10}$, which is extremely rare [30, 72, 73, 82, 83]; addressing many-bit errors is out of this paper's scope.

3.3 Region-level Persistence

WarmCache ensures that all dirty cache lines are written back to NVM before starting the execution of a new region, thereby guaranteeing that all *in-region* data updates are correctly persisted. This mechanism ensures that data modified in the current region is safe and recoverable, preventing issues related to multi-bit errors or incomplete writes. However, one challenge with this approach is that if a new region cannot begin execution until the write-back of the previous region is complete, the resulting persistence overhead can stall program execution, reducing performance.

To address this issue, WarmCache employs asynchronous write-back [92]. This technique allows the system to overlap the process of writing back dirty cache lines to NVM with the execution of new instructions until a program control reaches a region boundary, i.e., it utilizes CLWB after a store operation; the overlap happens between write-back latency and other instructions in the same region. By decoupling persistence from region execution, WarmCache mitigates the stall caused by the persistence operation, effectively reducing the latency impact.[1] Moreover, even if region persistence latency is long and thus interrupted by power failure, the persistence operation will be resumed across power failure thanks to the long retention time (relaxed non-volatility) of WarmCache.

3.4 Power Failure Recovery

In the wake of power failure, WarmCache validates dirty cache lines through scrubbing as discussed in Sec. 3.2. If no error is detected, WarmCache resumes the interrupted program using the JIT checkpoint/recovery protocol (Sec. 2.1). However, when a multi-bit error is detected in any dirty cache lines in the wake of power failure, WarmCache takes a different recovery approach. Instead of directly resuming the program, it rolls back to the entry point of the current program region to re-execute the corrupted region for multi-bit error correction. For rollback, WarmCache restores checkpointed stores that are generated by the WarmCache compiler; WarmCache compiler can exploit recovery slices of each program region for optimization [92]. Note that WarmCache is scalable, meaning that it can be applied to various types of intermittent processors, such as rollback recovery processors [31] or QuickRecall [36]. This would be possible if a separate checkpoint storage in NVM were dedicated to region-level error correction, allowing WarmCache to integrate with such processors.

3.5 WarmCache Architectural Exploration

To enable design space exploration of WarmCache, this paper defines a total execution time cost model ($T_{\text{total}}(\Delta)$) that considers the thermal-stability of STT-RAM (Δ) as follows: $T_{\text{total}}(\Delta) =$

[1] The efficiency of asynchronous write-back was proven by a prior work, which hides about 60% NVM access latency [92].

$T_{\text{exec}}(\Delta) + C(\Delta)$, where $T_{\text{exec}}(\Delta)$ is the program execution time, which may depend on the STT-RAM thermal-stability (Δ), and $C(\Delta)$ is the cost function that indicates the overhead of scrubbing, under $0 < \Delta < \infty$. The scrubbing overhead $C(\Delta)$ can be categorized as follows: $C(\Delta) = \frac{C_{\text{Per scrub}}(\Delta)}{t_s}$, where $C_{\text{Per scrub}}$ is the cost for a single scrubbing, and t_s is the scrubbing interval. The single scrubbing overhead ($C_{\text{Per scrub}}$) is: $(R \times T_{\text{MP}}(\Delta) + W \times T_{\text{RE}}(\Delta)) \times P_{\text{Multi-bit}}$. In the equation, R and W is the percentage of cache line read and writes respectively, $T_{\text{MP}}(\Delta)$ is the cache miss penalty, T_{RE} is the average re-execution penalty, and $P_{\text{Multi-bit}}$ is the probability of multi-bit errors. In particular, the probability of multi-bit errors and the scrubbing interval are inverse proportional and directly proportional to the thermal-stability, respectively: $P_{\text{Multi-bit}} \propto \frac{1}{\Delta}$, and $t_s \propto \Delta$. Based on this model, this paper found that the cost function for the scrubbing overhead is simply inversely proportional to the thermal-stability of STT-RAM: $C(\Delta) \propto \frac{1}{\Delta}$. Also, the program execution time is directly proportional to the thermal-stability since the memory access overhead increases as the thermal-stability increases: $T_{\text{exec}}(\Delta) \propto \Delta$. From these findings, this paper concludes a performance model as follows:

$$T_{\text{total}}(\Delta) \propto \Delta + \frac{1}{\Delta}.$$

Our goal is to find an optimal Δ that minimizes $T_{\text{total}}(\Delta)$ with the help of this performance model as shown Fig. 3, i.e., a minimal value for Δ that minimizes the $T_{\text{total}}(\Delta)$.

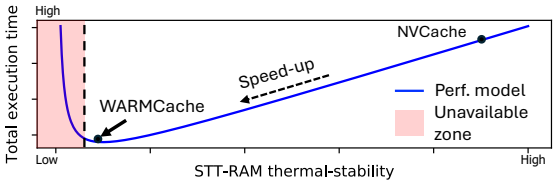


Figure 3: WarmCache performance model varying the thermal-stability. The thermal-stability cannot be reduced beyond a dashed line due to an intermittent power condition.

WarmCache Tuning. This paper found that WarmCache cannot keep reducing Δ until 0 for EHS as shown in Fig. 3. This limitation arises because WarmCache cannot perform scrubbing during power-off periods. If the scrubbing interval becomes shorter than the power-off period, scrubbing cannot occur when needed, resulting in a higher error rate. Notably, the power-off period is directly related to the capacitor recharging time. Although a larger capacitor can supply more energy during active operation, it also increases the power-off duration, making the system more vulnerable to errors during those extended periods. This creates a dilemma: larger capacitors not only increase operational energy but also increase the error rate due to the inability to scrub during recharging. Thus, finding the right balance between capacitor size and scrubbing interval is critical for maintaining EHS reliability.

With that in mind, this paper tunes the thermal stability and scrubbing interval of WarmCache, based on a 1uF capacitor as used in prior works [11, 99], to achieve the optimal performance—though we will vary the parameters for sensitivity analysis in Sec. 5.4. For a 4 KB cache with a 64-byte cache line and 2-way associativity, WarmCache sets 1 ms as the scrubbing interval. In this design, WarmCache utilizes a 1μF capacitor for energy storage when operating under RF, thermal, piezo, and solar ambient power sources ranging from 1 to 20 mW[64, 68]. This choice ensures that the capacitor recharging time matches the scrubbing interval; WarmCache scrubs in the wake of power failure (i.e., reboot time). On average, recharging a 1 μF capacitor from a power-off state (1.8 V) to the power-on voltage level (3.3 V) requires less than 1 ms with the given input power. In addition, we choose our Δ value of 28.91, with a mean retention time of 1 hour. Coupled with an ECC-1, a STT-RAM with this Δ will ensure the probability of multi-bit retention failures per line to be approx. 0.009 in the chosen time interval of 1 ms, giving a low probability of re-execution of just 0.9%. We empirically found that this value of 28.91 provides an optimal balance between reducing write energy and minimizing scrubbing overhead, resulting in the lowest execution time in our testing environment.

WarmCache has two possible scenarios of Silent Data Corruption (SDC): CRC-31 is capable of detecting up to 7-bit errors, but for 8-bit errors, it has a probability of misdetection P_{mis} of 2^{-31}[77]. Also, errors larger than 8-bit will go undetected by CRC-31, although these errors are rare with a probability $P_{\epsilon>8}$ of 7.11×10^{-10} of happening. Using these probabilities of SDC from both cases and the probability of 8-bit errors $P_{\epsilon=8}$, we get the total probability of undetected failure P_{fail} for WarmCache using the equation $P_{fail} = P_{mis}P_{\epsilon=8} + P_{\epsilon>8} - P_{mis}P_{\epsilon=8}P_{\epsilon>8}$ and from it we can get the FIT rate for WarmCache of around 9.71×10^{-12} FIT after correcting multi-bit errors using re-execution, which is lower than other STT-RAM reliability schemes as shown in Table 1.

4 WarmCache Compiler

For region formation, WarmCache compiler partitions the program into a series of idempotent code regions, enabling safe re-execution in the event of multi-bit error detection. This design allows regions to be re-executed to correct multi-bit errors with region-level granularity. Once a region's execution is confirmed as error-free, WarmCache ensures that any modified cache lines are persisted to NVM, which is resistant to retention failures. The region formation process is applied across the entire program, meaning that every point in the program belongs to a defined region [92].

At first glance, forming regions may seem as simple as generating a sequence of idempotent code regions [14–16, 88]. However, this paper identifies two key challenges as follows.

Problem 1. *If idempotent regions are either too large or too small, they will cause repeated re-execution or scrubbing, respectively.*

This paper found that if a region is too large, WarmCache compiler may fail to scrub cache lines within a scheduled scrubbing interval due to power failure, increasing the probability of retention failure, which can trigger repeated re-executions of the region and prevent forward progress, i.e., similar in effect to a denial of service (DoS). Conversely, if a region is too small, scrubbing overhead increases, reducing energy efficiency. To mitigate these issues, WarmCache introduces a *scrubbing interval aware region formation scheme* that adjusts the region size by either stitching smaller regions together or splitting larger ones. That way, WarmCache

can avoid both overly frequent and infrequent scrubbing, ensuring efficient and timely error correction.

Problem 2. *If there are too many dirty cache lines that need to be scrubbed within a region, the region may not be completed due to excessive scrubbing overhead.*

This paper also found that some regions may lead to too many writes, causing too many dirty cache lines that need to be scrubbed. Unfortunately, such regions can lead to significant performance degradation, as the scrubbing process may consume hard-earned energy, potentially preventing the regions from completing within one power cycle. In such cases, the region re-execution mechanism might be repeatedly triggered, further leading to a DoS issue. To address this challenge, WarmCache introduces a *scrubbing overhead-aware region formation* scheme. This approach controls the number of stores within a region by splitting regions that have many stores into smaller regions.

4.1 Scrubbing Interval Aware Region Formation.

WarmCache compiler starts the partitioning by traversing the Control Flow Graph (CFG) and initially putting the region boundaries at the function calls and loop headers [9, 55, 92, 93, 95]. Then, WarmCache re-forms the series of regions based on a following insight. Basically, typical idempotent regions are, on average, only 20 to 100 instructions long [14, 15, 33, 39, 50, 52–54, 97]. This results in high scrubbing overhead (i.e., less than 1μs scrubbing interval), as any dirty cache lines must be checked for errors before they are persisted to the NVM. On the other hand, there can be some outlier regions that are significantly longer than average regions [9]. This result in higher probability of retention failure, thereby causing the repeated re-execution. To address the issue, WarmCache introduces *region stitching* for short regions and *region splitting* for long regions, optimizing performance and reducing both scrubbing overhead and repeated retention failure risk.

Solution 1. *If it stitches small regions together or splits large ones, regions can neither too large nor too small.*

Region stitching. We found that if one can stitch regions together, it will effectively reduce the number of regions, thereby decreasing the scrubbing overhead as WarmCache scrubs at each region boundary. Consider the example of two consecutive regions, Rg1 and Rg2, where a memory[0] is updated in both regions. Stitching these regions together can eliminate the boundary; however, it is important to note that region stitching comes with two challenges, i.e., anti-dependent memory accesses across a region boundary and the re-execution penalty. Anti-dependent memory access across a region boundary can lead to wrong recovery, similar to data inconsistency or crash inconsistency. For instance, at the beginning of a region, if we load a value N (initially 10) from NVM to WarmCache, increment N by 1, and store it back to the same memory location, a system crash immediately after may result in a re-execution that loads N + 1 (11) instead of the original value N (10). This introduces data inconsistency. To address the issue, WarmCache employs an (undo) logging mechanism. While stitching idempotent regions, WarmCache leaves (undo) logs for anti-dependent stores in NVM. If multiple anti-dependent stores share the same NVM address, only the first one is logged. Then, at the stitched region end, WarmCache persists all dirty cache lines and clears the (undo) logs to prepare for tracking new logs in the next region. On the other hand, if retention failure occurs, WarmCache performs an undo the logged stores and restores checkpointed registers. This enables WarmCache to ensure correct recovery while preserving the idempotency of stitched regions.

Region splitting. WarmCache ensures that each region remains smaller than the scrubbing interval to prevent multi-bit errors in long regions. While aggressive region stitching—forming longer regions—can reduce scrubbing overhead, it introduces a trade-off with re-execution penalties. Restarting such long regions may be necessary multiple times across power outages [9, 14, 15, 88]. Even worse, it might have a problem called stagnation [9, 10], which means the system cannot move forward even though it can harvest and reboot, i.e., non-terminating bug [13]. To avoid such a problem, WarmCache compiler estimates the execution time of stitched regions using an instruction-level timing cost model; it traverses a region-level CFG of given program accumulating execution cycles of each instruction to measure the toal execution time of the region [9]. If the estimated execution time of a given region exceeds the configured scrubbing interval, WarmCache compiler splits the region into smaller pieces while preserving idempotency.

4.2 Scrubbing Overhead Aware Region Formation

To address the problem 2, WarmCache compiler introduces *scrubbing overhead aware region formation*. The key insight is that if a region has a limited number of stores, its execution will not lead to excessive scrubbing overhead.

Solution 2. *If a region has a limited number of stores, its execution will not cause excessive scrubbing overhead.*

WarmCache compiler determines the maximum number of stores a region can have by considering the worst-case scenario, where each store operation results in a different dirty cache line. This maximum is set as a threshold to prevent excessive scrubbing overhead that could prevent scrubbing to be never completed within a capacitor charge cycle while maximizing the region size to reduce the checkpoint overhead. To find the threshold, WarmCache compiler measures the probability of re-execution for n cache lines as $P^{n}_{\text{Multi-bit}}$. It increases the number of cache lines n and calculates the probability of re-execution, caused by multi-bit error, for each increment. From the analysis, we found that the maximum number of stores that keeps the re-execution probability below 1% is 11. With that in mind, WarmCache compiler counts the number of stores while forming a series of regions, and once the number of stores reaches the threshold, it cuts the region.

As an initial step, WarmCache compiler traverses the region-level CFG, which is generated through scrubbing interval-aware region formation (Sec. 4.1), in topological order. During this traversal, WarmCache counts the number of store operations within each region. Once this count reaches the pre-defined threshold, a region boundary is formed to start a new region thereafter. In particular, WarmCache performs alias analysis on stores within

a given region, while traversing its CFG. Through the alias analysis, WarmCache determines how many stores can be overwritten into the same cache lines, or in other words, how many cache hits occur for store operations. For instance, if two different store operations turned out to be must-aliased, the following store will apparently occur a cache hit without causing additional dirty cache lines. In this case, WarmCache does not count the following store. Finally, when the number of store operations in a region reaches the threshold, WarmCache inserts a region boundary.

Checkpoint Stores. Once WarmCache compiler completes the region formation process, it proceeds to insert checkpoint stores within each region. The region boundaries serve as recovery points, ensuring that each region can be safely recovered in the event of a power failure or the detection of multi-bit errors. The checkpoint stores allow WarmCache to restore the state of the program at recovery points, ensuring that if an error occurs, the system can resume the program execution from the last region boundary. This approach helps maintain data integrity and program correctness while avoiding the need for additional hardware support or complex error recovery mechanisms.

4.3 Forward Progress

Energy harvesting environments are inherently unpredictable and unstable, leading to long power failure again and again. These long power outages can increase the probability of retention failure in WarmCache. Furthermore, if such outages continue, a program region could repeatedly encounter retention failures, causing the program to stall and make no forward progress at all. This results in *stagnation*, where the system continuously re-executes the same region without making forward progress, wasting harvested energy [9, 11, 69, 92, 99]. To avoid stagnation, WarmCache ensures forward progress across each power outage, regardless of power source stability or outage duration. To achieve this, WarmCache leverages a unique characteristic of EHS. By design, they always start (boot) with a fully charged capacitor and run until it is depleted (Sec. 2.1). Based on this design principle, WarmCache equips its regions with *power failure immunity* [9]. That is, the compiler partitions program into a series of recoverable regions so that none of them consumes more energy than what the full capacitance offers, assuming the worst-case energy consumption. Thus, even if the regions encounter power failure once, they never fail again, i.e., any power-interrupted region is guaranteed to finish at reboot time with the help of *power failure immunity (PFI)* [9, 10, 80, 92, 99].

In detail, after forming a series of regions with previous two compiler passes (described in Sec. 4.1 and 4.2) WarmCache compiler performs the PFI pass using an instruction-level cost model. For each instruction, the compiler calculates the worst-case energy consumption and then accumulates the worst-case energy for all instructions within a region. To determine the worst-case energy consumption, this paper assumes that the EHS is under the most power-consuming condition (e.g., 0% cache hit). For memory access instructions, WarmCache conservatively assumes that they always result in cache misses. The compiler verifies whether the total energy consumption for a given region fits within the energy capacity provided by the capacitor in a single charge cycle. If the total exceeds the capacitor's capacity, WarmCache divides the region into smaller sub-regions, ensuring that no sub-region consumes more energy than the full capacity of a capacitor, i.e., for a given region R, its stores R_{stores}, and the recovery block $R_{recovery}$, the compiler ensures that the condition $FullCapacitorEnergy \geq E_{exec}(R)+E_{exec}(R_{recovery})+\sum_{i\in R_{stores}} E_{srcub}(i)$ is satisfied, where E fields are the worst-case energy consumptions. Also, to account for scrubbing overhead, WarmCache adds the scrubbing cost for each store, considering that each store may require scrubbing; this performance and energy overheads per cache line are 14.083 ns and 0.677 nJ, respectively. This paper demonstrates the soundness of WarmCache compiler in Sec. 5.4.6.

	STT-RAM			SRAM	RRAM	PCRAM
Thermal Stability Δ	28.91	34.04	40.29	//	//	//
Retention time	1 hour	1 week	10 years	//	//	//
Write Energy (nJ)	0.637	1.005	1.183	0.029	0.408	59.960
Write latency (ns)	10.083	10.076	10.76	0.016	20.072	150.039
Read Energy (nJ)	0.176	0.179	0.179	0.029	0.141	0.019
Read Latency (ns)	1.546	1.540	1.540	0.019	1.505	0.061
Leakage Power (mW)	8.837	9.250	9.310	24.958	9.943	3.870

Table 2: Comparison of different technologies and their parameters (Size: 4kB, Assoc. : 2 ways)

5 Evaluation

5.1 Experimental Setup

For our experiment, we utilized NVPsim [27], built on the gem5 [59], to evaluate WarmCache on an ARM architecture featuring a single-core processor. We re-modeled the simulator using NVSim at the 22nm technology node [19], tuning key cache parameters such as latency and power leakage for each memory component. We integrated these parameters into the NVPsim simulator as shown in Table 2. In particular, this table highlights the reduced write energy achieved by decreasing retention time and the lower leakage power compared to other technologies. For example, reducing the retention time from 10 years to 1 hour reduces the write energy by almost 0.5x. Along with the NVSim, we validated the simulator power leakage accuracy with a recent power model derived from a real microcontroller [89]. As default, we employed data and instruction caches, each with a 4 KB, 2-way set associative cache, 64-byte cache line; we also varied the size of cache for sensitivity analysis (Sec. 5.4). The simulator uses a 64 KB PCRAM main memory (as shown in Table 3), a 1 μF capacitor, and operates at a 250Mhz processor clock frequency [98][2]; we further vary the size of capacitor and the clock frequency for sensitivity analysis (Sec. 5.4).

We ran Mibench[29] and Mediabench[45] applications—as in prior work [11, 92]—atop the simulator with WarmCache as well as other cache designs including ReplayCache [92], Write-Light Cache (WL-Cache) [11], and non-volatile cache (NVCache) [27]. We implemented WarmCache compiler using LLVM compiler infrastructure [44]. The applications were all compiled with -O3 optimization level. For ReplyCache and WarmCache, we enabled their corresponding LLVM passes at compile time. We also used three different power traces to test for realistic energy harvesting environments: Thermal, RFOffice, and RFHome [11, 27, 92, 99].

[2]Apollo4 Blue commodity microcontroller, based on ARM Cortex-M processor architecture, introduced a clock frequency of 192MHz [42].

Size (kB)	Write Lat. (ns)	Write E. (nJ)	Read Lat. (ns)	Read E. (nJ)	Leak. P. (mW)
64	190.17	103.777	0.174	0.033	11.66

Table 3: NVM main memory parameter. Lat., E., Leak. and P. stand for latency, energy, leakage, and power, respectively.

	STT-RAM $\Delta = 60$	PCRAM	RRAM
Write Energy (nJ)	26.59	103.777	0.488
Write latency (ns)	250.116	190.17	500.217
Read Energy (nJ)	0.167	0.033	0.125
Read Latency (ns)	1.593	0.174	1.623
Leakage Power (mW)	34.422	11.655	18.989

Table 4: NVM parameters for different technologies (Size: 64 kB, Node: 22nm)

In our experiments, we configured cache designs as follows: NVCache employed a reliable STT-RAM cache (whose thermal-stability, Δ, is 60), while ReplayCache and WL-Cache used SRAM-based caches [11, 92][3]. For WarmCache, we utilized STT-RAM caches with an average retention time of 1 hour for both instruction and data caches. Note that we assume the input power of 1∼20mW from the energy harvester as default [57, 64]; we also conducted a sensitivity analysis by varying the constant input power levels; the details are deferred to Sec. 5.4.

5.2 Hardware Cost

We estimated the hardware cost of WarmCache. We choose 22nm technology for our STT-RAM caches using NVSim [19]. These caches will require an area of $0.058mm^2$ each and a leakage power of 8.837 mW. The thermal-stability of the STT-RAM caches is fixed at 28.91, ensuring a 1 hour average retention time. The area overhead of ECC-1 and CRC-31 for each 64-bit cache line is 41 bits: 10 bits for ECC-1 and 31 bits for CRC-31. This results in a total area overhead of 0.00127 mm^2.

5.3 Performance Analysis

5.3.1 Performance Analysis with Power Outage. We measured the total execution time of benchmark applications using power traces. We set NVCache as a baseline, and compared it to ReplayCache, WL-Cache, and WarmCache. Figures 4, 5, and 6 show the normalized speedups compared to the baseline in three different power traces, Thermal, RFOffice, and RFHome. The three power traces fluctuate with different power on/off sequences, which we take into account in our experiments.

For all applications, WarmCache shows the performance improvement over other designs in three power traces, with average speedup ratios of 5.84x (Thermal), 2.19x (RFOffice), and 5.85x (RFHome) compared to the baseline. WarmCache achieves this speedup over the baseline thanks to its lower write latency and leakage power consumption of WarmCache compared to other NVM technologies. ReplayCache and WL-Cache also show improvement over the baseline due to their use of a volatile cache, with WL-Cache having a slightly higher speedup. However, WarmCache is faster than ReplayCache by 42.8% (Thermal), 42.67% (RFOffice), and 41.95% (RFHome), also faster than WL-Cache by 31.84% (Thermal), 30.85% (RFOffice), and 30.3% (RFHome). We found this is mainly because WarmCache can avoid compulsory cache misses compared to the state-of-the-arts. ReplayCache and WL-Cache use a volatile cache, making them vulnerable to the cold cache phenomenon described in Sec. 2.3. WarmCache avoids this vulnerability by leveraging a relaxed STT-RAM that persists data across power failure and thus reduces the compulsory cache miss rate.

[3]When the thermal stability is 60, the STT-RAM cache can achieve the similar level of reliability, compared to WarmCache, without requiring scrubbing.

Compulsory cache misses. Figure 7 shows the average cache miss rate for ReplayCache, WL-cache, and WarmCache across RFHome, Thermal, and RFOffice power traces. For ReplacyCache and WL-cache, the compulsory cache miss rates are shown as diagonally hatched bars. The results indicate that the compulsory misses due to the cold cache phenomenon add significantly to the cache miss rate, with 17.56% and 15.52% compulsory cache miss rates for ReplayCache and WL-light, respectively. Because WarmCache retains the data in the cache across power failure, it can avoid these cold cache misses, decreasing its cache miss rate significantly.

5.3.2 Performance Analysis without Power Outage. We also measured the performance running NVCache, ReplayCache, WL-Cache, and WarmCache with no power failure. We set the baseline to be NVCache for performance analysis and compared it ReplayCache, WL-Cache, and WarmCache. The experiment indicated that WarmCache does not achieve a speedup over NVCache; instead, it performs slightly worse, averaging 0.95x the performance of NVCache (Figure 8). This performance gap comes from the additional overhead introduced by WarmCache's scrubbing process and the execution of idempotent regions, which offsets the benefits of its lower write energy compared to NVCache. In contrast, ReplayCache and WL-Cache demonstrate higher speedups compared to the baseline. This performance advantage is primarily thanks to their volatile nature, which avoids the scrubbing and persistence—related overheads inherent in non-volatile caches. As a result, ReplayCache and WL-Cache can achieve a speedup of 3.04x and 3.5x on average over NVCache and 1.69x and 1.84x over WarmCache, respectively.

5.4 Sensitivity Analysis

5.4.1 Cache Size. Figure 9 shows the average execution time of three cache designs with 4kB, 8kB, and 16kB cache sizes. We used RFHome as the power trace and 1 μF capacitor. All caches are 2-way and have a block size of 64 bytes. WarmCache showed consistent speedup compared to the baseline and other designs regardless of the cache size. WarmCache with 4kB achieved the highest speedup.

5.4.2 Capacitor Size. Figure 10 presents the execution time on average of three cache designs compared, evaluated using three different capacitors: 1μF, 10μF, and 100μF. RFHome was used as the power source for this experiment. WarmCache shows a lower execution time than ReplayCache and WL-Cache across all capacitors. Overall, the total execution time of all schemes increases as the capacitor size increases because larger capacitors lead to longer power-off periods.

5.4.3 NVM Technologies. Different NVM technologies, including ReRAM, PCM, and STT-RAM (with $\Delta = 60$), have different write/read

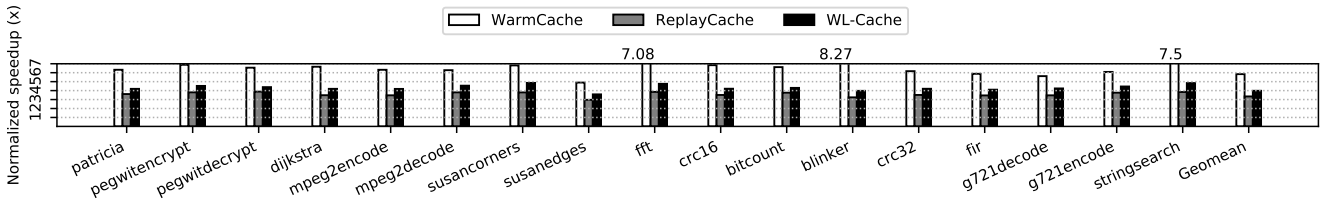


Figure 4: Normalized speedup of each cache design compared to NVCache in Thermal power trace

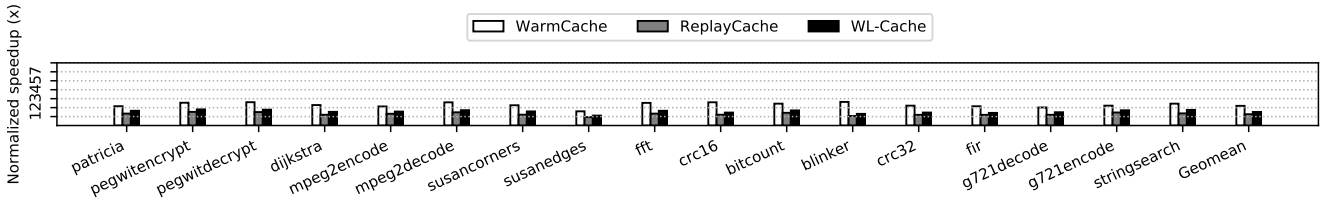


Figure 5: Normalized speedup of each cache design compared to NVCache in RFOffice power trace.

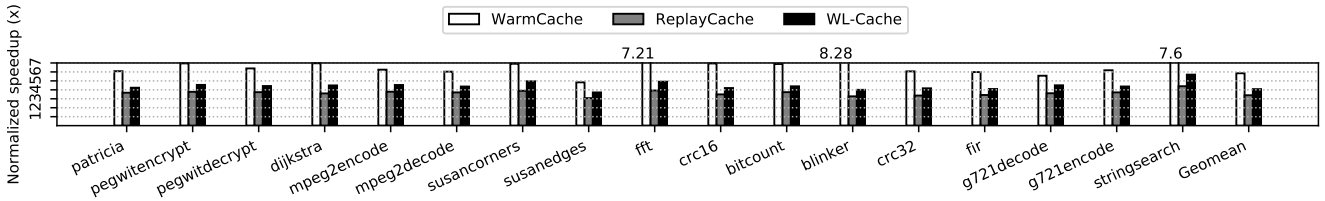


Figure 6: Normalized speedup of each cache design compared to NVCache in RFHome power trace.

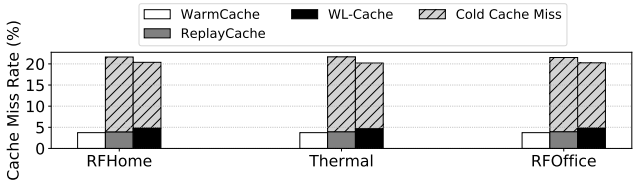


Figure 7: Cache miss rate of each cache design on average in three different power traces

power and latency characteristics. We found NVM parameters from NVSim [19] and tuned our simulator as shown in Table 4. In the context of these NVM technologies, Figure 11 presents the normalized performance of alternative cache schemes compared to three baseline systems using NVCache. The experiment demonstrates that WarmCache consistently outperforms the other schemes, achieving an average speedup of approximately 5x compared to the baseline across the different NVM technologies.

5.4.4 Clock Frequency Variation. Processors with higher clock frequencies incur higher leakage power, causing more frequent power failure. However, their higher clock speed can offset this drawback by enabling shorter program execution times, thereby enhancing overall performance. On the other hand, processors operating at lower clock frequencies lead to reduced leakage power, resulting in fewer power failure. Yet, this advantage comes at the cost of slower program execution times, which may impact performance.

In energy harvesting environments, the trade-off is influenced by the power-off period, or capacitor recharging time. When power-off periods are short, high-frequency processors outperform low-frequency processors. On the other hand, when power-off periods are long, low-frequency processors outperform due to their reduced power demands. To see the trade-off, we conducted additional experiments, varying the input power from 1mW~20mW and clock frequencies from 50Mhz~1Ghz. Figure 12 indicates the average execution time of applications with different clock frequencies, showing how input power and clock speed influence overall performance. We found that when the input power exceeds 7.5 mW, a 250 MHz processor delivers the best performance, taking advantage of shorter power-off periods and faster execution speeds. However, when the input power drops below 7.5 mW, the 50 MHz processor performs the best, with the 250 MHz processor coming in second. To this end, at input powers in between 1~20mW , which is realistic under solar, thermal, and RF power sources [57, 68], a 250Mhz processor is a viable choice in our environmental setting.

5.4.5 Forward Progress Analysis. We conducted a sensitivity analysis under unstable power conditions. We simulated long power-off

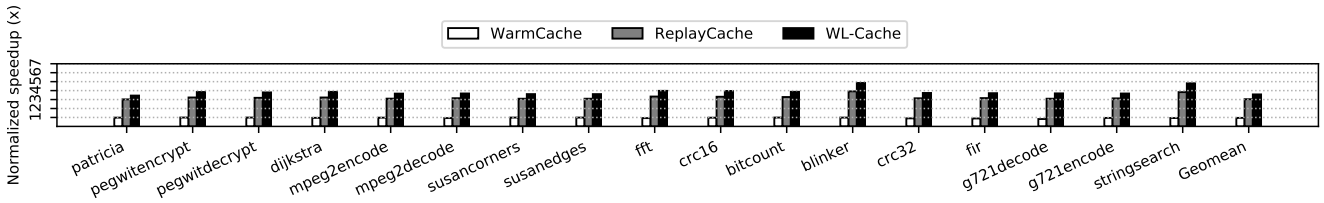


Figure 8: Normalized speedup of each cache design compared to NVCache with no power failure

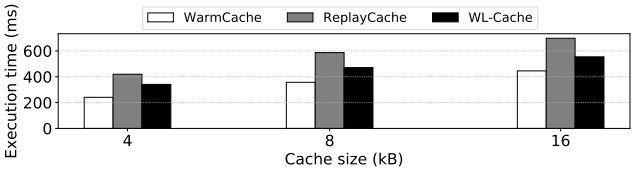


Figure 9: The execution time of each cache design on average in RFHome power trace varying each cache size.

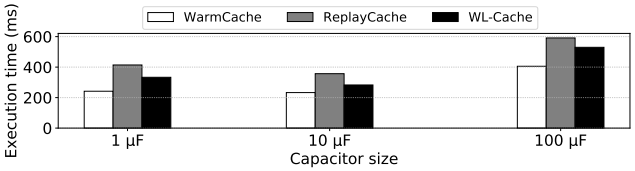


Figure 10: The execution time of each cache design on average in RFHome power trace varying capacitor size

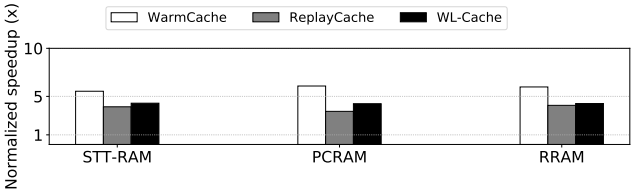


Figure 11: Normalized speedup of each cache design compared to NVCache on average in RFHome varying the NVM technology

scenarios in RFHome at random intervals using a Monte Carlo fault injection method. For each power-off event, we injected a retention failure with a probability varying from 1% to 99%, analyzing its impact on system performance. Despite the long power outages, we found that WarmCache still outperformed the state-of-the-art as shown in Fig. 13a. The reason is that, on average, 5.7 regions were executed per charge cycle, and only one region required re-execution due to retention failure across long power-off.

5.4.6 Energy Source Stability. We measured the distribution of the power-off periods in the RFHome, RFOffice, and Thermal traces while varying the capacitor size. From the experiments, we found that the min./avg./max. power-off periods for the three power traces

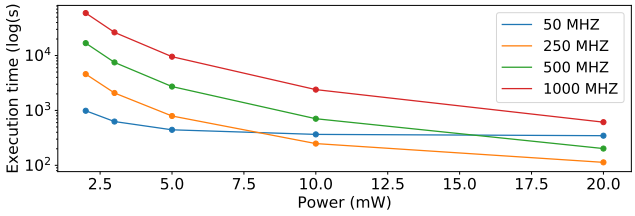
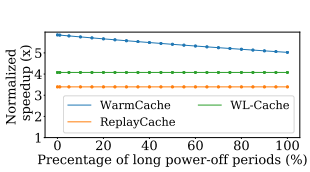
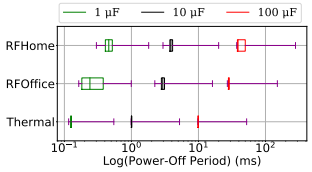


Figure 12: The execution time of WarmCache in RFHome power trace with varying clock frequency.



(a) Normalized speedup of each cache design compared to NVCache in RFHome varying the outage period.



(b) Distribution of power-off periods in RFHome, RFOffice, and Thermal with 1μF, 10μF, and 100μF capacitors.

Figure 13: Power source stability analysis

are 0.11/0.13/0.54 ms, 0.16/0.28/0.99 ms, and 0.3/0.48/1.8 ms with the 1μF capacitor, 1/1.07/5.2 ms, 2.23/3.12/16.1 ms, and 3/4.2/15 ms with the 10μF capacitor, and 9.8/15.2/52.3 ms, 27/43/150.3 ms, and 38.2/70/280.3 ms with the 100μF capacitor in Thermal, RFOffice, and RFHome, respectively as shown in Fig. 13b. Overall, the power-off period increases as the capacitor size increases. This trend highlights the need for WarmCache to carefully tune the STT-RAM retention time, considering the capacitor size, to align with the power-off periods for performance optimization.

5.5 Reliability Analysis

5.5.1 Comparison with Other Retention Times . We compared different STT-RAM caches with three average retention times, 1 minute, 1 hour, and 1 week, and calculated the probability of multi-bit errors per cache line while varying scrub intervals as shown in Figure 14. As discussed in Sec. 3.5, we aim to keep the probability of multi-bit errors per cache line under 1% to achieve the FIT rate $9.71x10^{-12}$ with low execution overhead. As shown in the figure, a 1-hour retention time STT-RAM meets that goal at around 1 ms, requiring both the region execution time and the power-off period to be under 1 ms, which fits our 1 μF capacitor and 1~20 mW input

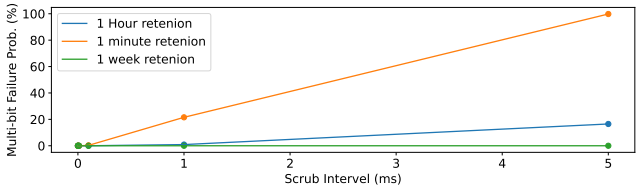


Figure 14: Probability of multi-bit retention failure in cache lines with varying scrub intervals and retention times

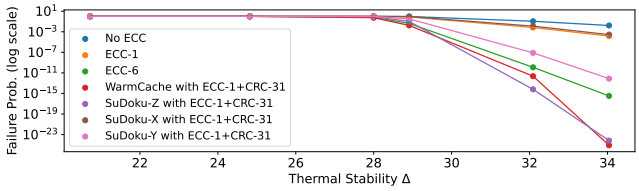


Figure 15: Cache retention failure probability under varying thermal-stability and different ECC support

power. In contrast to the 1-hour retention STT-RAM, the STT-RAM with 1-week average retention time can meet the threshold with longer scrub intervals, thus allowing longer execution cycles; however, it causes a higher write energy overhead as shown in Table 2. At the same time, one with a 1-minute average retention requires even shorter scrub intervals of 100 μs to meet the threshold of 1%, increasing significantly the scrubbing overhead.

5.5.2 Comparison with Error-Correcting Methods. We compared WarmCache to six other error correction methods: no error correction code (No ECC), scrubbing with a ECC-1 code (ECC-1), scrubbing with a ECC-6 code (ECC-6), and SuDoku error detection and correction design described in [73] at different levels X, Y, and Z (Sudoku-X, Y, and Z). For each method, we calculated the probability of retention failure over a 20 ms period while varying thermal stability, Δ, as shown in Figure 15. As shown in the figure, WarmCache shows a failure probability comparable to SuDoku-Z and outperforms all other methods assessed.

In particular, WarmCache offers advantages in latency and storage compared to these different methods. Compared to ECC-6, WarmCache requires less storage, utilizing 10 bits for ECC-1 and 31 bits for CRC-31 in one cycle instead of the 60 bits needed for ECC-6 [72]. Moreover, SuDoku relies on RAID-4 to correct multi-bit errors, which demands additional storage space. This approach can also lead to instances of detected but uncorrected errors, requiring the incorporation of SuDoku-Y and SuDoku-Z in addition to the base SuDoku-X [73], adding to the overall overhead of the system.

6 Other Related Works

Zhou et al. have recently introduced a hardware-software co-design cache, prioritizing low hardware complexity over transparency [99]. For lightweight crash consistency, SweepCache does not checkpoint cache lines but instead utilizes multiple redo buffers resident in NVM to ensure the correct recovery of any power-interrupted regions. Despite its ability to circumvent WAR dependencies using redo buffers, SweepCache ends up extending the memory access path by introducing the buffers between the cache and NVM. This results in each cache writeback involving two NVM writes—one to the buffer and another to the main memory. This is problematic because NVM writes are the most expensive in terms of both latency and power consumption, particularly in power-hungry EHS.

Ma et al. [63], have introduced a form of approximate computing for EHS, called *Incidental Computing*, that employs the relaxed STT-RAM with a short retention time. In this approach, data is computed in the relaxed STT-RAM, and if a long power outage occurs, the scheme rolls forward by computing with fresh data while marking old data as *incidental*. If the incidental data in STT-RAM turned out to be necessary for high quality of service, then the scheme rolls back and computes with the data. This approach adds such a quality knob and provides flexibility; however, it basically asks programmers to decide which datasets to apply incidental computing and determine the acceptable quality level through a new programming model, leaving critical decisions in the hands of end-users. Also, since the scheme does not address retention failure of the relaxed STT-RAM, it is orthogonal to WarmCache.

One alternative approach is to use refresh-free or reduced-refresh DRAM [2, 26, 46, 56, 75, 85, 86]. These approaches can also retain data for a short period of time, so their nature can be exploited to avoid compulsory misses. However, it would be a mistake to take this mean that they can address the cold cache issue. Since the retention time of these approaches is just a few *ms*, these approaches become susceptible when the power-off period is longer than such a short period of time; indeed, the power-off period can be more than a hundred *ms* in all traces as shown in Fig. 13. In contrast, STT-RAM retention time is configurable, allowing it to be adjusted to match the expected power-off period, thereby effectively mitigating the cold cache issue. This paper introduces a new perspective, suggesting that manufacturers could optimize performance by adjusting the retention time of STT-RAM at design time to align with the expected power failure period.

This paper also found that other NVM technologies can be adopted to WarmCache. In particular, a recent NVM technology, SOT-RAM [78, 81], can vary the retention time in a similar way of STT-RAM. Furthermore, FeFET [20, 28, 47–49] offers higher density and better energy efficiency than RRAM. Since these emerging memory technologies may further improve the overall performance of WarmCache, we leave the exploration for our future work.

7 Conclusion

This paper introduces WarmCache, a cache design for EHS to address the compulsory cold cache cache misses. By leveraging a relaxed STT-RAM cache with reduced thermal stability, WarmCache effectively preserves data across power outages. Our experiment demonstrates that WarmCache reduces compulsory misses and improves performance by 1.3~1.4x on average compared to state-of-the-art cache designs.

Acknowledgments

The authors thank anonymous reviewers for their feedback. This work is in part supported by NSF grants 2314680, 2314681, 2153749, 2001124, and 2106629 as well as ONR grant N00014-23-1-2136.

References

[1] Angelo Bacchini, Marco Rovatti, Gianluca Furano, and Marco Ottavi. 2014. Characterization of data retention faults in DRAM devices. In *2014 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*. IEEE, 9–14.

[2] Seungjae Baek, Sangyeun Cho, and Rami Melhem. 2013. Refresh now and then. *IEEE Trans. Comput.* 63, 12 (2013), 3114–3126.

[3] Mary Baker, Satoshi Asami, Etienne Deprit, John Ouseterhout, and Margo Seltzer. 1992. Non-volatile memory for fast, reliable file systems. *ACM SIGPLAN Notices* 27, 9 (1992), 10–22.

[4] Domenico Balsamo, Alex S Weddell, Geoff V Merrett, Bashir M Al-Hashimi, Davide Brunelli, and Luca Benini. 2014. Hibernus: Sustaining computation during intermittent supply for energy-harvesting systems. *IEEE Embedded Systems Letters* 7, 1 (2014), 15–18.

[5] Shihua Cao and Jianqing Li. 2017. A survey on ambient energy sources and harvesting methods for structural health monitoring applications. *Advances in Mechanical Engineering* 9, 4 (2017), 1687814017696210.

[6] Jongouk Choi, Jaeseok Choi, Hyunwoo Joe, and Changhee Jung. 2024. Caphammer: Exploiting Capacitor Vulnerability of Energy Harvesting Systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 43, 11 (2024), 3804–3815.

[7] Jongouk Choi, Hyunwoo Joe, and Changhee Jung. 2022. Capos: Capacitor error resilience for energy harvesting systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41, 11 (2022), 4539–4550.

[8] Jongouk Choi, Hyunwoo Joe, Yongjoo Kim, and Changhee Jung. 2019. Achieving stagnation-free intermittent computation with boundary-free adaptive execution. In *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 331–344.

[9] Jongouk Choi, Larry Kittinger, Qingrui Liu, and Changhee Jung. 2022. Compiler-directed high-performance intermittent computation with power failure immunity. In *2022 IEEE 28th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 40–54.

[10] Jongouk Choi, Qingrui Liu, and Changhee Jung. 2019. CoSpec: Compiler directed speculative intermittent computation. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. 399–412.

[11] Jongouk Choi, Jianping Zeng, Dongyoon Lee, Changwoo Min, and Changhee Jung. 2023. Write-light cache for energy harvesting systems. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*. 1–13.

[12] Yung-Wey Chong, Widad Ismail, Kwangman Ko, and Chen-Yi Lee. 2019. Energy harvesting for wearable devices: A review. *IEEE Sensors Journal* 19, 20 (2019), 9047–9062.

[13] Alexei Colin and Brandon Lucia. 2018. Termination checking and task decomposition for task-based intermittent programs. In *Proceedings of the 27th International Conference on Compiler Construction*. 116–127.

[14] Marc de Kruijf and Karthikeyan Sankaralingam. 2011. Idempotent processor architecture. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 140–151.

[15] Marc de Kruijf and Karthikeyan Sankaralingam. 2013. Idempotent code generation: Implementation, analysis, and evaluation. In *Code Generation and Optimization (CGO), 2013 IEEE/ACM International Symposium on*. IEEE, 1–12.

[16] Marc A. de Kruijf, Karthikeyan Sankaralingam, and Somesh Jha. 2012. Static Analysis and Compiler Design for Idempotent Processing. In *Proceedings of the 33rd ACM SIGPLAN Conference on Programming Language Design and Implementation* (Beijing, China) *(PLDI '12)*. ACM, New York, NY, USA, 475–486. https://doi.org/10.1145/2254064.2254120

[17] Brandon Del Bel, Jongyeon Kim, Chris H Kim, and Sachin S Sapatnekar. 2014. Improving STT-MRAM density through multibit error correction. In *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1–6.

[18] Zhitao Diao, Zhanjie Li, Shengyuang Wang, Yunfei Ding, Alex Panchula, Eugene Chen, Lien-Chang Wang, and Yiming Huai. 2007. Spin-transfer torque switching in magnetic tunnel junctions and spin-transfer torque random access memory. *Journal of Physics: Condensed Matter* 19, 16 (apr 2007), 165209. https://doi.org/10.1088/0953-8984/19/16/165209

[19] Xiangyu Dong, Cong Xu, Yuan Xie, and Norman P Jouppi. 2012. Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 31, 7 (2012), 994–1007.

[20] Stefan Dünkel, Martin Trentzsch, Regina Richter, Patrick Moll, Christine Fuchs, Oliver Gehring, Mateusz Majer, Stefan Wittek, B Müller, Thomas Melde, et al. 2017. A FeFET based super-low-power ultra-fast embedded NVM technology for 22nm FDSOI and beyond. In *2017 IEEE International Electron Devices Meeting (IEDM)*. IEEE, 19–7.

[21] Reem Elkhouly, Mohammad Alshboul, Akihiro Hayashi, Yan Solihin, and Keiji Kimura. 2019. Compiler-support for critical data persistence in NVM. *ACM Transactions on Architecture and Code Optimization (TACO)* 16, 4 (2019), 1–25.

[22] Gan Fang, Jongouk Choi, and Changhee Jung. 2025. Hybrid Power Failure Recovery for Intermittent Computing. In *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*.

[23] Gan Fang and Changhee Jung. 2025. Rethinking Dead Block Prediction for Intermittent Computing. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*.

[24] Gan Fang and Changhee Jung. 2025. Rethinking Prefetching for Intermittent Computing. In *Proceedings of International Symposium on Computer Architecture*.

[25] Alexander Freij, Shougang Yuan, Huiyang Zhou, and Yan Solihin. 2020. Persist level parallelism: Streamlining integrity tree updates for secure persistent memory. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 14–27.

[26] Mrinmoy Ghosh and Hsien-Hsin S. Lee. 2007. Smart Refresh: An Enhanced Memory Controller Design for Reducing Energy in Conventional and 3D Die-Stacked DRAMs. In *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*. 134–145. https://doi.org/10.1109/MICRO.2007.13

[27] Yizi Gu, Yongpan Liu, Yiqun Wang, Hehe Li, and Huazhong Yang. 2016. NVPsim: A simulator for architecture explorations of nonvolatile processors. In *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 147–152.

[28] Shreya Gupta, Mark Steiner, Ahmedullah Aziz, Vijaykrishnan Narayanan, Suman Datta, and Sumeet Kumar Gupta. 2017. Device-circuit analysis of ferroelectric FETs for low-power logic. *IEEE Transactions on Electron Devices* 64, 8 (2017), 3092–3100.

[29] Matthew R Guthaus, Jeffrey S Ringenberg, Dan Ernst, Todd M Austin, Trevor Mudge, and Richard B Brown. 2001. MiBench: A free, commercially representative embedded benchmark suite. In *Proceedings of the fourth annual IEEE international workshop on workload characterization. WWC-4 (Cat. No. 01EX538)*. IEEE, 3–14.

[30] Mostafa Hadizadeh, Elham Cheshmikhani, and Hossein Asadi. 2020. STAIR: High reliable STT-MRAM aware multi-level I/O cache architecture by adaptive ECC allocation. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1484–1489.

[31] Matthew Hicks. 2017. Clank: Architectural support for intermittent computation. *ACM SIGARCH Computer Architecture News* 45, 2 (2017), 228–240.

[32] Yiming Huai et al. 2008. Spin-transfer torque MRAM (STT-MRAM): Challenges and prospects. *AAPPS bulletin* 18, 6 (2008), 33–40.

[33] Shao-Yu Huang, Jianping Zeng, Xuanliang Deng, Sen Wang, Ashrarul Sifat, Burhanuddin Bharmal, Jia-Bin Huang, Ryan Williams, Haibo Zeng, and Changhee Jung. 2023. Rtailor: Parameterizing soft error resilience for mixed-criticality real-time systems. In *2023 IEEE Real-Time Systems Symposium (RTSS)*.

[34] Changhee Jung Jaeseok Choi, Hyunwoo Joe and Jongouk Choi. 2024. Defending Against EMI Attacks on Just-In-Time Checkpoint for Resilient Intermittent Systems. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*.

[35] Belal Jahannia, Seyed Ali Ghasemi, and Hamed Farbeh. 2024. Multi-retention stt-mram architectures for iot: Evaluating the impact of retention levels and memory mapping schemes. *IEEE Access* 12 (2024), 26562–26580.

[36] Hrishikesh Jayakumar, Arnab Raha, and Vijay Raghunathan. 2014. QuickRecall: A low overhead HW/SW approach for enabling computations across power cycles in transiently powered computers. In *2014 27th International Conference on VLSI Design and 2014 13th International Conference on Embedded Systems*. IEEE, 330–335.

[37] Jungi Jeong and Changhee Jung. 2021. PMEM-spec: persistent memory speculation (strict persistency can trump relaxed persistency). In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*.

[38] Jungi Jeong, Jianping Zeng, and Changhee Jung. 2022. Capri: Compiler and architecture support for whole-system persistence. In *Proceedings of the 31st International Symposium on High-Performance Parallel and Distributed Computing*. 71–83.

[39] Hongjune Kim, Jianping Zeng, Qingrui Liu, Mohammad Abdel-Majeed, Jaejin Lee, and Changhee Jung. 2020. Compiler-Directed Soft Error Resilience for Lightweight GPU Register File Protection. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*.

[40] Yusung Kim, Sumeet Kumar Gupta, Sang Phill Park, Georgios Panagopoulos, and Kaushik Roy. 2012. Write-optimized reliable design of STT MRAM. In *Proceedings of the 2012 ACM/IEEE international symposium on Low power electronics and design*. 3–8.

[41] P. Koopman. 2018. *Crc polynomial zoo. [online].* http://web.archive.org/web/20080207010024/http://www.808multimedia.com/winnt/kernel.htm

[42] Vito Kortbeek, Souradip Ghosh, Josiah Hester, Simone Campanoni, and Przemysław Pawełczak. 2022. Wario: efficient code generation for intermittent computing. In *Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation*. 777–791.

[43] Emre Kültürsay, Mahmut Kandemir, Anand Sivasubramaniam, and Onur Mutlu. 2013. Evaluating STT-RAM as an energy-efficient main memory alternative. In *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 256–267.

[44] Chris Lattner and Vikram Adve. 2004. LLVM: A compilation framework for lifelong program analysis & transformation. In *International symposium on code generation and optimization, 2004. CGO 2004.* IEEE, 75–86.

[45] Chunho Lee, Miodrag Potkonjak, and William H Mangione-Smith. 1997. Mediabench: A tool for evaluating and synthesizing multimedia and communications systems. In *Proceedings of 30th Annual International Symposium on Microarchitecture*. IEEE, 330–335.

[46] Sen Li, Hui Jin, Yingke Gao, Ying Wang, Shuhong Dai, Yongjun Xu, and Long Cheng. 2024. Approximate data mapping in refresh-free DRAM for energy-efficient computing in modern mobile systems. *Computer Communications* 216 (2024), 151–158. https://doi.org/10.1016/j.comcom.2023.12.037

[47] Xueqing Li, Kaisheng Ma, Sumitha George, Win-San Khwa, John Sampson, Sumeet Gupta, Yongpan Liu, Meng-Fan Chang, Suman Datta, and Vijaykrishnan Narayanan. 2017. Design of nonvolatile SRAM with ferroelectric FETs for energy-efficient backup and restore. *IEEE Transactions on Electron Devices* 64, 7 (2017), 3037–3040.

[48] Xueqing Li, John Sampson, Asif Khan, Kaisheng Ma, Sumitha George, Ahmedullah Aziz, Sumeet Kumar Gupta, Sayeef Salahuddin, Meng-Fan Chang, Suman Datta, et al. 2017. Enabling energy-efficient nonvolatile computing with negative capacitance FET. *IEEE Transactions on Electron Devices* 64, 8 (2017), 3452–3458.

[49] Xueqing Li, Juejian Wu, Kai Ni, Sumitha George, Kaisheng Ma, John Sampson, Sumeet Kumar Gupta, Yongpan Liu, Huazhong Yang, Suman Datta, et al. 2019. Design of 2T/cell and 3T/cell nonvolatile memories with emerging ferroelectric FETs. *IEEE Design & Test* 36, 3 (2019), 39–45.

[50] Qingrui Liu, Joseph Izraelevitz, Se Kwon Lee, Michael L Scott, Sam H Noh, and Changhee Jung. 2018. iDO: Compiler-directed failure atomicity for nonvolatile memory. In *International Symposium on Microarchitecture*.

[51] Qingrui Liu and Changhee Jung. 2016. Lightweight hardware support for transparent consistency-aware checkpointing in intermittent energy-harvesting systems. In *2016 5th Non-Volatile Memory Systems and Applications Symposium (NVMSA)*.

[52] Qingrui Liu, Changhee Jung, Dongyoon Lee, and Devesh Tiwari. 2015. Clover: Compiler Directed Lightweight Soft Error Resilience.

[53] Qingrui Liu, Changhee Jung, Dongyoon Lee, and Devesh Tiwari. 2016. Compiler-Directed Lightweight Checkpointing for Fine-Grained Guaranteed Soft Error Recovery. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*.

[54] Qingrui Liu, Changhee Jung, Dongyoon Lee, and Devesh Tiwari. 2016. Compiler-Directed Soft Error Detection and Recovery to Avoid DUE and SDC via Tail-DMR. *ACM Trans. Embed. Comput. Syst.* (2016).

[55] Qingrui Liu, Changhee Jung, Dongyoon Lee, and Devesh Tiwarit. 2016. Low-cost soft error resilience with unified data verification and fine-grained recovery for acoustic sensor based detection. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*.

[56] Song Liu, Karthik Pattabiraman, Thomas Moscibroda, and Benjamin G. Zorn. 2011. Flikker: saving DRAM refresh-power through critical data partitioning. In *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems* (Newport Beach, California, USA) *(ASPLOS XVI)*. Association for Computing Machinery, New York, NY, USA, 213–224. https://doi.org/10.1145/1950365.1950391

[57] Yongpan Liu, Zewei Li, Hehe Li, Yiqun Wang, Xueqing Li, Kaisheng Ma, Shuangchen Li, Meng-Fan Chang, Sampson John, Yuan Xie, et al. 2015. Ambient energy harvesting nonvolatile processors: From circuit to system. In *Proceedings of the 52nd Annual Design Automation Conference*. 1–6.

[58] Yongpan Liu, Jinshan Yue, Hehe Li, Qinghang Zhao, Mengying Zhao, Chun Jason Xue, Guangyu Sun, Meng-Fan Chang, and Huazhong Yang. 2017. Data backup optimization for nonvolatile SRAM in energy harvesting sensor nodes. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 36, 10 (2017), 1660–1673.

[59] Jason Lowe-Power, Abdul Mutaal Ahmad, Ayaz Akram, Mohammad Alian, Rico Amslinger, Matteo Andreozzi, Adrià Armejach, Nils Asmussen, Brad Beckmann, Srikant Bharadwaj, et al. 2020. The gem5 simulator: Version 20.0+. *arXiv preprint arXiv:2007.03152* (2020).

[60] Brandon Lucia, Vignesh Balaji, Alexei Colin, Kiwan Maeng, and Emily Ruppel. 2017. Intermittent computing: Challenges and opportunities. *2nd Summit on Advances in Programming Languages (SNAPL 2017)* (2017).

[61] Kaisheng Ma, Jinyang Li, Xueqing Li, Yongpan Liu, Yuan Xie, Mahmut Kandemir, Jack Sampson, and Vijaykrishnan Narayanan. 2018. IAA: Incidental approximate architectures for extremely energy-constrained energy harvesting scenarios using IoT nonvolatile processors. *IEEE Micro* 38, 4 (2018), 11–19.

[62] Kaisheng Ma, Xueqing Li, Mahmut Taylan Kandemir, Jack Sampson, Vijaykrishnan Narayanan, Jinyang Li, Tongda Wu, Zhibo Wang, Yongpan Liu, and Yuan Xie. 2018. NEOFog: Nonvolatility-exploiting optimizations for fog computing. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*. 782–796.

[63] Kaisheng Ma, Xueqing Li, Jinyang Li, Yongpwan Liu, Yuan Xie, Jack Sampson, Mahmut Taylan Kandemir, and Vijaykrishnan Narayanan. 2017. Incidental computing on IoT nonvolatile processors. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 204–218.

[64] Kaisheng Ma, Xueqing Li, Shuangchen Li, Yongpan Liu, John Jack Sampson, Yuan Xie, and Vijaykrishnan Narayanan. 2015. Nonvolatile processor architecture exploration for energy-harvesting applications. *IEEE Micro* 35, 5 (2015), 32–40.

[65] Kaisheng Ma, Xueqing Li, Huichu Liu, Xiao Sheng, Yiqun Wang, Karthik Swaminathan, Yongpan Liu, Yuan Xie, John Sampson, and Vijaykrishnan Narayanan. 2017. Dynamic power and energy management for energy harvesting nonvolatile processor systems. *ACM Transactions on Embedded Computing Systems (TECS)* 16, 4 (2017), 1–23.

[66] Kaisheng Ma, Xueqing Li, Srivatsa Rangachar Srinivasa, Yongpan Liu, John Sampson, Yuan Xie, and Vijaykrishnan Narayanan. 2017. Spendthrift: Machine learning based resource and frequency scaling for ambient energy harvesting nonvolatile processors. In *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 678–683.

[67] Kaisheng Ma, Xueqing Li, Karthik Swaminathan, Yang Zheng, Shuangchen Li, Yongpan Liu, Yuan Xie, John Jack Sampson, and Vijaykrishnan Narayanan. 2016. Nonvolatile processor architectures: Efficient, reliable progress with unstable power. *IEEE Micro* 36, 3 (2016), 72–83.

[68] Kaisheng Ma, Yang Zheng, Shuangchen Li, Karthik Swaminathan, Xueqing Li, Yongpan Liu, Jack Sampson, Yuan Xie, and Vijaykrishnan Narayanan. 2015. Architecture exploration for ambient energy harvesting nonvolatile processors. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 526–537.

[69] Kiwan Maeng and Brandon Lucia. 2019. Supporting peripherals in intermittent systems with just-in-time checkpoints. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*. 1101–1116.

[70] Michele Magno and David Boyle. 2017. Wearable energy harvesting: From body to battery. In *2017 12th International conference on design & technology of integrated systems in nanoscale era (DTIS)*. IEEE, 1–6.

[71] Michele Magno, Dario Kneubühler, Philipp Mayer, and Luca Benini. 2018. Micro kinetic energy harvesting for autonomous wearable devices. In *2018 International symposium on power electronics, electrical drives, automation and motion (SPEEDAM)*. IEEE, 105–110.

[72] Helia Naeimi, Charles Augustine, Arijit Raychowdhury, Shih-Lien Lu, and James Tschanz. 2013. STTRAM SCALING AND RETENTION FAILURE. *intel technology journal* 17, 1 (2013).

[73] Prashant J Nair, Bahar Asgari, and Moinuddin K Qureshi. 2019. SuDoku: Tolerating high-rate of transient failures for enabling scalable STTRAM. In *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 388–400.

[74] Hiroki Noguchi, Kazutaka Ikegami, Satoshi Takaya, Eishi Arima, Keiichi Kushida, Atsushi Kawasumi, Hiroyuki Hara, Keiko Abe, Naoharu Shimomura, Junichi Ito, et al. 2016. 7.2 4Mb STT-MRAM-based cache with memory-access-aware power optimization and write-verify-write/read-modify-write scheme. In *2016 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 132–133.

[75] K. Patel, L. Benini, Enrico Macii, and Massimo Poncino. 2005. Energy-Efficient value-based selective refresh for embedded DRAMs. In *Proceedings of the 15th International Conference on Integrated Circuit and System Design: Power and Timing Modeling, Optimization and Simulation* (Leuven, Belgium) *(PATMOS'05)*. Springer-Verlag, Berlin, Heidelberg, 466–476. https://doi.org/10.1007/11556930_48

[76] Minesh Patel, Geraldo Francisco de Oliveira, and Onur Mutlu. 2021. HARP: Practically and effectively identifying uncorrectable errors in memory chips that use on-die error-correcting codes. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*. 623–640.

[77] William Wesley Peterson and Daniel T Brown. 1961. Cyclic codes for error detection. *Proceedings of the IRE* 49, 1 (1961), 228–235.

[78] Guillaume Prenat, Kotb Jabeur, Gregory Di Pendina, Olivier Boulle, and Gilles Gaudin. 2015. Beyond STT-MRAM, spin orbit torque RAM SOT-MRAM for high speed and high reliability applications. *Spintronics-based Computing* (2015), 145–157.

[79] Mitchelle Rasquinha, Dhruv Choudhary, Subho Chatterjee, Saibal Mukhopadhyay, and Sudhakar Yalamanchili. 2010. An energy efficient cache design using spin torque transfer (STT) RAM. In *Proceedings of the 16th ACM/IEEE international symposium on Low power electronics and design*. 389–394.

[80] Joshua San Miguel, Karthik Ganesan, Mario Badr, Chunqiu Xia, Rose Li, Hsuan Hsiao, and Natalie Enright Jerger. 2018. The eh model: Early design space exploration of intermittent processor architectures. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 600–612.

[81] Qiming Shao, Peng Li, Luqiao Liu, Hyunsoo Yang, Shunsuke Fukami, Armin Razavi, Hao Wu, Kang Wang, Frank Freimuth, Yuriy Mokrousov, et al. 2021. Roadmap of spin–orbit torques. *IEEE transactions on magnetics* 57, 7 (2021), 1–39.

[82] 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 *2011 IEEE 17th International Symposium on High Performance Computer Architecture*. 50–61. https://doi.org/10.1109/HPCA.2011.5749716

[83] Zhenyu Sun, Xiuyuan Bi, Hai 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 *2011 44th Annual IEEE/ACM International*

Symposium on Microarchitecture (MICRO). 329–338.

[84] Sandeep Krishna Thirumala, Arnab Raha, Hrishikesh Jayakumar, Kaisheng Ma, V Narayanan, Vijay Raghunathan, and Sumeet Kumar Gupta. 2018. Dual mode ferroelectric transistor based non-volatile flip-flops for intermittently-powered systems. In *Proceedings of the International Symposium on Low Power Electronics and Design*. 1–6.

[85] R.K. Venkatesan, S. Herr, and E. Rotenberg. 2006. Retention-aware placement in DRAM (RAPID): software methods for quasi-non-volatile DRAM. In *The Twelfth International Symposium on High-Performance Computer Architecture, 2006*. 155–165. https://doi.org/10.1109/HPCA.2006.1598122

[86] Jue Wang, Xiangyu Dong, and Yuan Xie. 2014. ProactiveDRAM: A DRAM-initiated retention management scheme. In *2014 IEEE 32nd International Conference on Computer Design (ICCD)*. IEEE, 22–27.

[87] Yiqun Wang, Yongpan Liu, Shuangchen Li, Daming Zhang, Bo Zhao, Mei-Fang Chiang, Yanxin Yan, Baiko Sai, and Huazhong Yang. 2012. A 3us wake-up time nonvolatile processor based on ferroelectric flip-flops. In *ESSCIRC (ESSCIRC), 2012 Proceedings of the*. IEEE, 149–152.

[88] Joel Van Der Woude and Matthew Hicks. 2016. Intermittent Computation without Hardware Support or Programmer Intervention. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. USENIX Association, Savannah, GA, 17–32. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/vanderwoude

[89] Huanjie Wu, Chun Chen, and Kai Weng. 2021. An energy-efficient strategy for microcontrollers. *Applied Sciences* 11, 6 (2021), 2581.

[90] Mimi Xie, Chen Pan, Youtao Zhang, Jingtong Hu, Yongpan Liu, and Chun Jason Xue. 2018. A novel STT-RAM-based hybrid cache for intermittently powered processors in IoT devices. *IEEE Micro* 39, 1 (2018), 24–32.

[91] Mimi Xie, Mengying Zhao, Chen Pan, Hehe Li, Yongpan Liu, Youtao Zhang, Chun Jason Xue, and Jingtong Hu. 2016. Checkpoint aware hybrid cache architecture for NV processor in energy harvesting powered systems. In *Proceedings of the eleventh IEEE/ACM/ifip international conference on hardware/software codesign and system synthesis*. 1–10.

[92] Jianping Zeng, Jongouk Choi, Xinwei Fu, Ajay P Shreepathi, Dongyoon Lee, Changwoo Min, and Changhee Jung. 2021. ReplayCache: Enabling Volatile Caches for Energy Harvesting Systems. (2021).

[93] Jianping Zeng, Shao-Yu Huang, Jiuyang Liu, and Changhee Jung. 2024. Soft Error Resilience at Near-Zero Cost. In *Proceedings of the 38th ACM International Conference on Supercomputing*.

[94] Jianping Zeng, Jungi Jeong, and Changhee Jung. 2023. Persistent processor architecture. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*. 1075–1091.

[95] Jianping Zeng, Hongjune Kim, Jaejin Lee, and Changhee Jung. 2021. Turnpike: Lightweight Soft Error Resilience for In-Order Cores. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*.

[96] Jianping Zeng, Tong Zhang, and Changhee Jung. 2024. Compiler-directed whole-system persistence. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 961–977.

[97] Yida Zhang and Changhee Jung. 2022. Featherweight soft error resilience for GPUs. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE.

[98] Mengying Zhao, Shuo Xu, Lihao Dong, Chun Jason Xue, Dongxiao Yu, Xiaojun Cai, and Zhiping Jia. 2023. Branch Predictor Design for Energy Harvesting Powered Nonvolatile Processors. *IEEE Trans. Comput.* (2023).

[99] Yuchen Zhou, Jianping Zeng, Jungi Jeong, Jongouk Choi, and Changhee Jung. 2023. Sweepcache: Intermittence-aware cache on the cheap. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*. 1059–1074.

[100] Yuchen Zhou, Jianping Zeng, and Changhee Jung. 2024. Lightwsp: Whole-system persistence on the cheap. In *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 215–230.