Intermittence-Aware Cache Compression

Gan Fang
Purdue University
West Lafayette, USA
fang301@purdue.edu

Jianping Zeng
Arizona State University
Tempe, USA
jpzeng@asu.edu

Yuchen Zhou
Purdue University
West Lafayette, USA
zhou1166@purdue.edu

Changhee Jung
Purdue University
West Lafayette, USA
chjung@purdue.edu

***Abstract*—Cache compressions are proven effective in improving the performance of caches in conventional processors. They compress data into a smaller size, allowing caches to accommodate more blocks. This helps reduce cache misses and expensive memory accesses, ultimately improving performance. However, conventional cache compression is less effective for energy harvesting systems (EHSs) which experience frequent power failure, as many compressed blocks end up not being used before their loss upon power outage. This wastes hard-won energy, which would otherwise be used for making more program progress. To address this issue, this paper introduces Kagura, an adaptive cache compression extension with frequent power failure in mind. Specifically, Kagura disables cache compression when it finds out that many cached blocks are unlikely to be reused before the next power outage. That way, Kagura avoids the energy waste on useless compressions/decompressions, and the resulting speedup is on par with the ideal intermittence-aware cache compressor. Experimental results show that when combined with an existing cache compressor, Kagura reduces the total energy consumption by an average of 4.53% (up to 16.21%) and improves the performance by an average of 4.74% (up to 17.87%) compared to the baseline EHS without cache compression.**

I. Introduction

Energy harvesting systems (EHSs) have emerged as a compelling alternative to battery-powered embedded devices. Thanks to their battery-free design, EHSs can operate for an ultra-long time without maintenance, offering advantages like environmental friendliness and self-sustainability [133]. EHSs are applicable across a wide range of fields, including batteryless IoT [3], [27], [62], [76], [84], [173], [179], stream and river monitoring [85], [146], health and wellness tracking [29], [31], [49], [57], [128], and wearable computing [37], [45], [97], [114], [115]. Instead of using battery, EHSs harvest energy from ambient sources such as radio frequency (RF) and WiFi, storing the energy in a capacitor that serves as an energy buffer. However, these sources tend to be unstable and weak, causing EHSs to be frequently power-interrupted with all volatile data lost [1], [17], [18], [20], [28], [46], [78], [79], [88], [96], [98], [102], [108], [138], [161], [166]. This is why EHSs are regarded as *intermittent computing systems* [109], i.e., they operate only when their internal capacitors hold enough energy. To prevent the loss of critical data and ensure forward progress across frequent power outages, EHSs incorporate both nonvolatile memory (NVM) [7], [36], [67], [69], [75], [106], [157], [181] as main memory and a crash consistency mechanism [80]–[82], [105], [132], [136], [170], [176], [180], [182], [185]—e.g., just-in-time (JIT) checkpointing of volatile data when power fails and its restoration when power returns [19], [24], [78], [89], [108], [110]–[112], [144], [145], [165], [178].

Given unstable and weak input energy, maintaining high energy efficiency is crucial for achieving high-performance EHSs equipped with power-hungry NVM main memory. To lower the NVM accesses, prior proposals [24], [44], [61], [63], [118], [177], [184] leverage a volatile SRAM cache—as in NVSRAMCache [63] that JIT-checkpoints dirty cache blocks upon power failure. This strategy allows EHSs to exploit spatial and temporal locality in program, retrieving data from the cache rather than NVM whose dynamic access energy is way higher than that of SRAM. As a result, EHSs can dedicate their hard-won energy to forward progress and eventually deliver better performance [177].

Although SRAM cache reduces NVM traffic, it leads to extra energy waste caused by leakage current which becomes larger as cache size increases. Because of this, EHSs cannot afford larger SRAM caches. Figure 1 shows the performance of a typical EHS across different cache sizes, normalized to a baseline featuring 256B ICache and DCache; Section VIII details the simulation configurations. With small caches (e.g., 128B), the EHS suffers frequent misses that severely degrade performance. As the cache size increases to 256B, more data locality is exploited, reducing cache misses and improving performance. However, when the cache size exceeds 256B, performance begins to decline since the higher leakage energy accelerates capacitor depletion, causing more frequent power outages. This, in turn, forces the EHS to perform additional JIT checkpoints and restorations, resulting in performance degradation that outweighs the benefits of larger caches. ***The takeaway is that EHSs face a critical dilemma that small caches suffer from high miss rates, whereas large caches incur prohibitive leakage. This necessitates techniques to better utilize the precious cache capacity.***

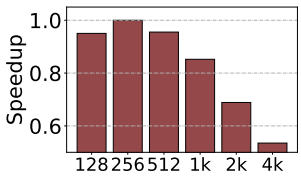


Fig. 1: Speedup over baseline (256B each for ICache and DCache) across different cache sizes in bytes (B).

To improve cache utilization, researchers propose many cache compression techniques [8]–[10], [14], [15], [21], [33], [35], [56], [65], [91], [93], [119], [127], [131], [143], [154],

[156], [160], [162], [171], [186], [187]. They allow for more data blocks to be accommodated in caches, thus increasing the effective capacity of the caches, with a little increase in hardware overhead and the associated leakage. For example, BDI [131] selects a common base for each block so that each value in the block can be represented as a delta, i.e., a data block is now represented as one base plus multiple deltas. Based on this representation, BDI requires fewer bits to store a cache block, enabling higher cache utilization and thereby improving overall performance by an average of 8.1% [131].

However, cache compressions are not free as they introduce extra delay and energy consumption on cache access, which could incur performance loss if they are blindly applied to energy-constrained EHSs. When a new block is allocated to a full cache set, compressors should compress both the incoming block and some of the existing uncompressed blocks to make room for it. Similarly, processors need to pay for a decompression cost whenever they access compressed blocks. To mitigate these overheads, researchers [10] propose Adaptive Cache Compressor (ACC) that activates compression only when it is expected to increase cache hits and disables it otherwise to avoid unnecessary latency and energy consumption.

While ACC [10] is effective for conventional processors, it cannot bring the best performance for EHSs because it does not take into account their frequent power interruptions. Specifically, even though ACC can accurately predict that many blocks will be reused and compress them in the cache accordingly, a power interruption may occur before the reuse happens, causing the compressed data to be lost and nullifying the compression effort. Given the high frequency of power outages, a significant portion of hard-won energy is wasted on compressing such useless cache blocks, ultimately degrading the performance of EHSs.

To this end, this paper introduces Kagura[1], a cache compression extension designed with frequent power failure in mind, aiming to optimize existing cache compressors for EHSs. In particular, Kagura can prevent useless cache compressions without negatively impacting cache hits. The core mechanism of Kagura is to dynamically enable or disable compression based on how likely cache blocks are to be reused before power failure. When Kagura identifies that many cache blocks are unlikely to be accessed before an outage, it disables cache compression to avoid wasting energy on useless compression. As such, even if the cache is full upon the arrival of a new block, Kagura falls back to the conventional cache replacement policy, i.e., finding a victim block for eviction, rather than performing compression.

In the implementation, Kagura takes an insight that the number of memory operations in a power cycle reflects how many cache blocks are touched therein. Thus, at run time, Kagura estimates the number of memory operations that will be executed before the next power outage to decide whether to disable cache compression. When this number is small enough, i.e., a few cached blocks will be accessed before they are lost, Kagura turns off the cache compressor to avoid energy waste. On the contrary, if the estimated number of memory operations remains large, Kagura lets the compressor operate as usual to maximize available cache space. ***In this way, Kagura strikes a balance between energy efficiency and large effective cache capacity.***

Nevertheless, two challenges must be addressed before Kagura can be put into practice. The first is determining how many memory instructions will execute from a decision point—where Kagura evaluates whether to disable compression—through the end of the current power cycle. This is challenging because power failure could happen at any time and is inherently hard to predict. To overcome this, Kagura uses the execution history of a prior power cycle as an approximation, leveraging the observation that program behavior remains consistent between two consecutive (short) power cycles as detailed in Section VIII-B. Specifically, Kagura tracks two counts: (1) memory instruction count from the prior power cycle and (2) the count accumulated so far in the current cycle up to the decision point. At that point, Kagura subtracts the latter count from the former and uses the difference to predict the number of memory instructions to be executed before the current power cycle ends.

The second challenge lies in selecting a suitable decision point within a power cycle. Basically, Kagura should disable cache compression when the number of memory instructions expected between the decision point and the end of the current power cycle falls below a threshold. However, a fixed threshold cannot consistently deliver optimal performance. If the threshold is set too high, it may result in insufficient cache space and thus extra cache misses. On the other hand, if the threshold is too low, Kagura could miss many opportunities to avert unnecessary compressions, wasting hard-won energy. To address this, Kagura adjusts the threshold whenever EHS reboots, based on the number of evicted cache blocks due to disabled compression. That is, Kagura decreases the threshold when the number of evicted cache blocks is high; otherwise, Kagura increases the threshold.

Experiments using real power traces [63], [135] show that for applications from Mibench [66] and Mediabench [97], ACC reduces total energy by only 0.47% on average and delivers a negligible performance gain of 0.0022% compared to a baseline EHS without cache compression. After applying Kagura to ACC, the total energy reduction reaches 4.53%, and the performance gain rises to 4.74% compared to the baseline. In summary, Kagura makes the following contributions:

- Kagura is the first to avert useless cache compressions for EHSs taking into account their frequent power interruptions.
- Compared to the baseline, Kagura reduces energy waste by up to 16.21% and execution time by as much as 17.87%. The average speedup of Kagura (4.74%) is close to that of the **ideal** intermittence-aware cache compressor (6.19%).
- Kagura incurs minimal hardware overhead, i.e., only a 2-bit counter and 5 registers taking up 0.14% of the core area.

[1]Kagura is a female character in *Inuyasha*, who describes herself as "the wind, the free wind," a soul unbound and carried by the currents of fate. Likewise, our proposal Kagura moves with similar freedom, dynamically shaping its behavior to the shifting power failure patterns of the EHS.

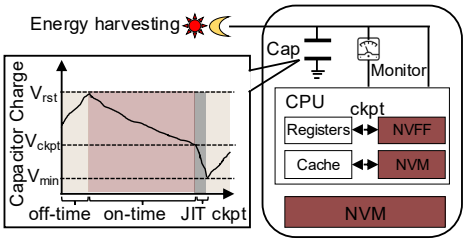


Fig. 2: Overall architecture of NVSRAMCache [63].

II. Background

A. Energy Harvesting System

A substantial body of research has explored equipping EHSs with volatile caches [24], [44], [61], [63], [118], [177], [184]. In the literature, NVSRAMCache [63] stands out as both promising and widely recognized. Figure 2 shows the basic architecture of NVSRAMCache and how it works at a high level. NVSRAMCache offers nonvolatile backups for its cache and processor states, including a register file and a store buffer. To guarantee correct program execution across power outages, it employs a just-in-time (JIT) checkpointing mechanism which proactively preserves necessary states right before a power outage. Specifically, when the voltage monitor detects that the capacitor voltage has dropped below the checkpoint threshold V_{ckpt}, NVSRAMCache immediately saves all volatiles with the core pipeline halted, e.g., checkpointing registers to nonvolatile flip-flops (NVFFs) and saving store buffer and dirty cache blocks to their nonvolatile counterparts.

After completing the checkpoint, NVSRAMCache hibernates until the capacitor recharges above the restoration threshold V_{rst}; at this point, NVSRAMCache reboots with the previously saved states restored and then resumes program execution from the interruption point. In particular, the period starting from when the EHS resumes until the next power interruption occurs is referred to as a *power cycle*.

B. Cache Compression Algorithm

Cache compression has been an active research area for decades, as it offers a compelling approach to increasing effective cache capacity and reducing cache misses by storing data in a compressed format. In this paper, we select 4 typical compression algorithms whose low hardware complexity makes them suitable for EHSs. The following provides a brief overview of the selected algorithms:

1. Dynamic Zero Compression (DZC) [160]: This simple algorithm targets common zero-value bytes. It adds a Zero Indicator Bit (ZIB) per byte so that zero bytes store only the indicator rather than eight zero bits. On a cache access, DZC checks the ZIB for the target memory location and immediately treats the byte as zero if indicated.

2. Frequent Pattern Compression (FPC) [8]: FPC splits a 64B block into 32-bit words and matches each word against common patterns (zeros, small sign-extended integers, repeating bytes). Matches are encoded with a 3-bit prefix, while unmatched words remain uncompressed. The resulting compressed words are organized into fixed-size segments for alignment. When accessing a compressed block, FPC reads its corresponding prefixes to reconstruct each 32-bit word therein.

3. C-Pack [35]: This algorithm conducts compression by examining small data chunks to see if they match common patterns or resemble recently seen data in a small dictionary. Upon access, C-pack utilizes these codes to look up and reconstruct the original data.

4. Base-Delta-Immediate (BDI) [131]: BDI compression leverages intra-block value similarity by identifying a common base value within a block. It then stores the base value plus a small delta (typically 8 to 32 bits) for each subsequent value. During a cache access, the original value is quickly reassembled by adding its respective delta back to the shared base.

C. Adaptive Cache Compression

Existing cache compressions enable caches to store more data blocks, but the associated overheads could potentially outweigh their benefits. For memory-intensive workloads, compressing blocks significantly increases effective cache capacity and reduces miss rates, thereby improving performance. However, for non-memory-intensive workloads, cache compression may provide little or no benefit because the added compression/decompression overheads can dominate.

In response, ACC [10] was proposed to enable compression only when it contributes to cache hits. ACC monitors recent cache behaviors—such as a block's position in the LRU order and its compressed size—and maintains a global saturating counter called Global Compression Predictor (GCP). When compression helps avoid cache misses, ACC increments the GCP. On the other hand, when compression causes extra delay for cache accesses due to decompression, it decrements GCP. Based on the GCP value, ACC decides whether to enable or disable cache compression. Specifically, if this counter is positive, indicating that compression is beneficial overall, ACC keeps the compressor active. Otherwise, it disables compression operations. In this manner, the cache dynamically adapts its behaviors based on the current workload characteristics, hence achieving a balance between increased effective capacity and reduced access latency.

III. Preparing Cache Compression for EHSs

Realizing cache compression in EHS requires careful consideration. The reason is that although compression can reduce misses, it also brings extra costs associated with compression and decompression. To model the effectiveness of cache compression in EHS, we introduce two metrics: energy benefit $E_{benefit}$ and waste E_{waste} resulting from cache compression. Assume that cache compression improves cache hit rate from R_{hit} to R^{+}_{hit}. For a total of N memory operations, the number of NVM accesses eliminated by cache compression is $(R^{+}_{hit} - R_{hit}) * N$. Accordingly, the total energy benefit $E_{benefit}$ from cache compression for N memory operations is given in Equation 1.

In this equation, E_{miss} denotes the energy consumed on bringing a missed block from NVM to cache—including the

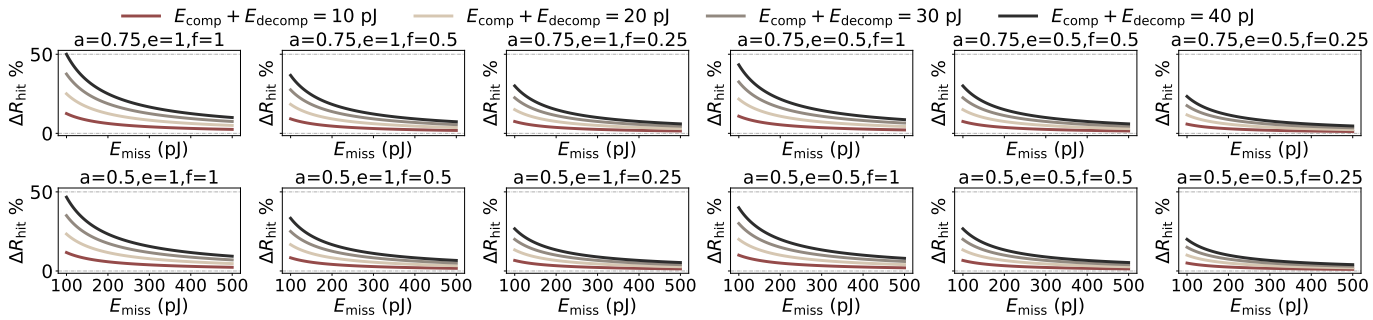


Fig. 3: Relationship among minimum ΔR_{hit}, compression and decompression cost ($E_{comp} + E_{decomp}$), cache miss penalty (E_{miss}), and different a, e and f.

energy for the miss handling itself, the NVM access energy, the cache allocation energy, and the static energy during the cache miss.

$$E_{benefit} = (R^{+}_{hit} - R_{hit}) * N * E_{miss} \quad (1)$$

Cache compression also necessitates extra energy costs for compressing and decompressing blocks. Let E_{comp} represent the energy to compress a block when it is filled into a cache, E_{decomp} stands for the energy to decompress a block when it is accessed or evicted. For N memory operations, assume that a fraction a of these operations access compressed blocks, while the remaining $1 - a$ access uncompressed blocks. Also, let L denote the number of compressed cache block evictions.

Under these assumptions, the energy spent on decompression is $(a * N + L) * E_{decomp}$. Moreover, suppose that M blocks are compressed upon insertion into the cache, i.e., the compression energy cost is $M * E_{comp}$. Therefore, the total energy waste incurred by cache compression is expressed in Equation 2.

$$E_{waste} = (a * N + L) * E_{decomp} + M * E_{comp} \quad (2)$$

By combining Equation 1 and 2, we get the total energy reduction achieved through cache compression. This reduction must be greater than zero for EHSs to realize performance improvement, as shown in Inequality 3. By applying simple algebraic transformation to Inequality 3, we derive the minimum required improvement in cache hit rate (ΔR_{hit}) necessary to achieve net energy reduction as shown in Inequality 4, where e and f stand for $\frac{L}{N}$ and $\frac{M}{N}$, respectively. ***The takeaway is that cache compression will benefit EHSs if and only if it can improve the cache hit rate by at least the calculated threshold*** ΔR_{hit}.

$$E_{total} = E_{benefit} - E_{waste} > 0 \quad (3)$$

$$\Delta R_{hit} = R^{+}_{hit} - R_{hit} > \frac{(a+e) * E_{decomp} + f * E_{comp}}{E_{miss}} \quad (4)$$

Inequality 4 demonstrates that the minimum required ΔR_{hit} varies as E_{decomp}, E_{comp}, and E_{miss} change. Figure 3 shows the relationship among the minimum ΔR_{hit}, the combined compression and decompression cost ($E_{comp} + E_{decomp}$), and the cache miss penalty (E_{miss}). Each subfigure of Figure 3 has different values of a, e and f. For example, $a = 0.75$ means 75% of memory operations access compressed cache blocks; $e = 0.5$ indicates one compressed-block eviction for every two memory operations; and $f = 0.5$ means that one block is compressed for every two memory operations.

Figure 3 depicts that when a, e or f decreases, the minimum required ΔR_{hit} also decreases, i.e., it becomes easier for compression to provide net benefits. Conversely, as a, e or f increases, cache compressor should achieve a larger improvement in hit rate to deliver energy reduction and performance gain for EHSs. In addition, each subfigure illustrates that either increasing $E_{comp} + E_{decomp}$ or decreasing E_{miss} lowers the minimum ΔR_{hit} needed for compression to be beneficial. If these factors change in the opposite direction, the minimum required ΔR_{hit} becomes higher.

IV. Problem Statement

Unfortunately, the analysis above does not consider frequent power outages in EHSs. This is a critical factor because the benefit of compression could significantly diminish if compressed blocks are not accessed before an outage occurs. If a power failure happens before the compressed data is utilized, the hard-won energy spent on fetching the blocks and compressing them is wasted. Ultimately, this wasted energy leads to suboptimal performance in EHSs.

Figure 4 and 5 show how cache compressors can cause energy waste. Figure 4 shows how a compressor can benefit performance by allowing more data to fit in the cache, while Figure 5 depicts that such a benefit can vanish in the event of power failure. Both examples assume a 50% compression ratio, meaning that each cache entry can hold up to 2 compressed blocks. The caches in both cases start empty.

As shown in Figure 4 (no power failure), at time T_1, the processor issues 4 memory instructions accessing 4 distinct cache blocks (A, B, C, and D). All four accesses miss in the cache, incurring an energy waste of $4*E_{miss}+4*E_{comp}$; these blocks are compressed as they are expected to be reused later (❶). Thanks to the ability to hold all 4 blocks, the following 4 memory instructions hit in the cache with an energy waste of only $4 * E_{decomp}$. Here in the no power failure scenario, the total energy waste is $4 * E_{miss} + 4 * E_{comp} + 4 * E_{decomp}$.

However, a power outage may occur during program execution, as shown in Figure 5. In this example, the processor

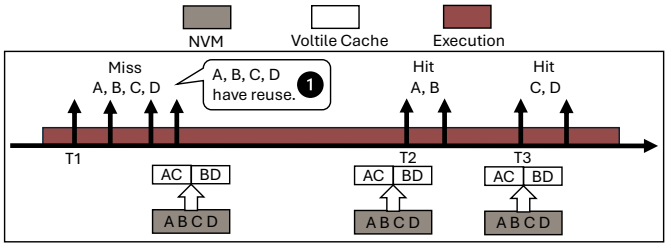


Fig. 4: Compressing cache blocks since they are going to be reused, resulting an energy waste of $4 * E_{miss} + 4 * E_{comp} + 4 * E_{decomp}$.

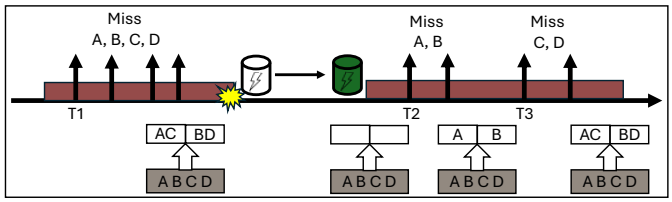


Fig. 5: Compressed cache blocks are not reused before power loss, resulting in an energy waste of $8 * E_{miss} + 8 * E_{comp}$.

still encounters 4 cache misses for blocks A, B, C, D at time T_1, resulting in an energy waste of $4 * E_{miss}$. These 4 blocks are then compressed in anticipation of future reuse, incurring an additional energy waste of $4 * E_{comp}$. Unfortunately, the power outage causes all cache blocks to be lost before they can be reused. Once power comes back, the processor finds an empty cache and has to retrieve the same cache blocks (A, B, C, and D) from NVM again, incurring another energy waste of $4 * E_{miss} + 4 * E_{comp}$. In this power failure scenario, the total energy waste becomes $8 * E_{miss} + 8 * E_{comp}$, which is significantly higher than the energy waste of Figure 4. The insight is that ***if the compression is aware of upcoming power loss, the energy of*** $4 * E_{comp}$ ***spent at time*** T_1 ***can be averted and instead used to advance program execution, thereby achieving higher performance.***

V. Overview of Kagura

What makes Kagura stand out is its ability to enable and disable compression at run time with frequent power outages in mind. This design maximizes the energy efficiency of EHSs and ultimately improves their performance. Specifically, Kagura switches between two operation modes: *Compression Mode (CM)* and *Regular Mode (RM)*. In *CM*, Kagura allows the existing cache compressor to perform compression as usual, while in *RM*, it disables compression. Kagura starts with *CM* mode after reboot, and later enters into *RM* mode—with cache compression turned off—when it determines that the benefit of compression has become insufficient. More precisely, Kagura enters into *RM* mode when the number of memory accesses to be executed before the end of the current power cycle is small, i.e., it is less than or equal to a compression-disabling threshold (N_{thres}). Once EHSs resume from power failure with their capacitors fully charged, Kagura starts with *CM* mode again and repeats the process above.

Figure 6 shows how Kagura avoids energy waste on useless compression in the presence of power failure. In this example,

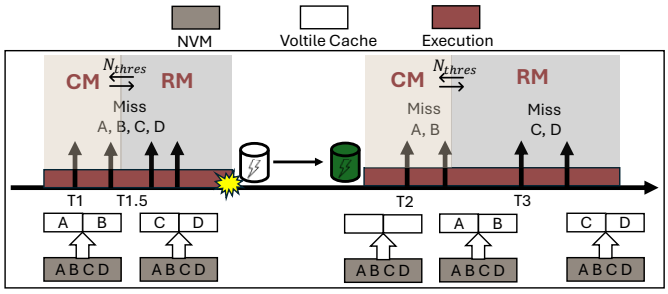


Fig. 6: High-level workflow of Kagura with mode switching; Kagura reduces energy waste to only $8 * E_{miss}$.

the cache has 2 data entries, and N_{thres} is set to 2. Initially, the processor starts with an empty cache, and Kagura operates in *CM* mode. At time T_1, the processor encounters two cache misses for blocks A and B, fetching them from NVM and filling them into the cache, which results in an energy waste of $2 * E_{miss}$. Since the cache is not yet full, Kagura does not compress blocks A and B. Shortly after, at time $T_{1.5}$, Kagura determines that compression would not be beneficial because only two memory accesses remain before the power loss—exactly equal to N_{thres}. Kagura thus switches to *RM* mode with compression disabled. As a result, when blocks C and D are accessed, they replace blocks A and B; here no compression energy is spent on any of the four blocks. After power returns, Kagura enters into *CM* again. At time T_2, the processor encounters two misses for the block A and B, leading to another energy waste of $2 * E_{miss}$. Then, the number of memory operations to be executed reaches N_{thres} again, Kagura switches back to *RM* mode. As such, when blocks C and D are accessed at time T_3, Kagura evicts blocks A and B to accommodate them, resulting in an additional energy waste of $2 * E_{miss}$ while still avoiding compression.

Compared to the total energy waste of $8 * E_{miss} + 8 * E_{comp}$ observed in Figure 5 (where compression was uselessly performed), the total energy cost here is only $8 * E_{miss}$. This highlights Kagura's effectiveness at avoiding useless compression and the associated energy overhead; the additional $8 * E_{\text{comp}}$ is successfully eliminated. One might question Kagura's decision to switch to RM, arguing that remaining in *CM* could preserve more space to accommodate future memory requests, thus turning some potential misses into hits. However, this concern is unwarranted for two primary reasons.

First, Kagura switches modes only when it predicts imminent power failure; the implication is that few memory operations remain in the current power cycle. Second, the cache blocks of the remaining memory operations are unlikely to be reused during such a short amount of time left before the imminent power failure; if reuse were expected, their values would have been allocated to registers at compile time. Under these conditions, disabling cache compression does not introduce extra capacity misses—a point supported by our cache miss rate evaluation results shown in Figure 15. As a result, Kagura's use of the uncompressed cache here is adequate to retain the working set throughout the period from the switching point until the next power failure.

VI. Implementation Details

The key to Kagura's success lies in determining (1) when Kagura should disable cache compression and (2) how Kagura should adjust the decision point for the compression disabling; fixing this decision point is suboptimal, since program behavior and energy conditions vary over time. Recall that the number of memory operations to be executed before a power outage reflects how many cache blocks are to be accessed before the outage. Therefore, Kagura takes the expected number of memory operations remaining in the current power cycle as the metric to determine the decision point (i.e., when to disable compression), while avoiding complex hardware structures. That is, Kagura disables cache compression when this number reaches the compression-disabling threshold N_{thres}. Now, the above two questions are converted into (1) how to calculate this number in an energy-efficient way and (2) how to dynamically adjust N_{thres} accordingly. Note that Kagura should address both questions with minimal hardware overhead, which would otherwise in turn offset the benefits of this adaptive cache compression. The following two sections detail how Kagura tackles each of these questions, respectively.

A. Determining When to Disable Compression

To decide whether to disable cache compression, Kagura needs to check the number of memory instructions to be executed before the next power outage. However, it is a daunting challenge to know this number at run time since power failure can occur unpredictably.

Simple Approach: Inspired by history-based predictions in computer architecture, Kagura leverages historical data to approximate the number of memory instructions to be executed through the end of the current power cycle; this number is denoted as N_{remain}. To be specific, Kagura uses the number of committed memory operations in the prior power cycle as a reference to estimate N_{remain} in the current cycle. Let N_{prev} represent the number of committed memory operations in the previous power cycle, and N_{mem} represent the number accumulated so far in the current cycle up to the decision point. Kagura computes N_{remain} simply as:

$$N_{remain} = N_{prev} - N_{mem} \tag{5}$$

Whenever N_{remain} reaches the threshold N_{thres}, Kagura disables cache compression to avoid wasting energy on useless compression. To compute N_{prev} and N_{mem}, Kagura proposes 2 registers R_{prev} and R_{mem} as shown in Figure 7. Here, both R_{prev} and R_{mem} are volatile, but R_{mem} should be checkpointed to a designated NVFF on power failure. Note that when rebooting, R_{prev} is initialized with the restored value of R_{mem}, and R_{mem} is reset to zero. Upon committing a memory instruction, Kagura performs 3 actions in a row: (1) increasing R_{mem} by 1; (2) calculating the difference $R_{prev} - R_{mem}$; (3) comparing the subtraction result with the threshold N_{thres} to determine if cache compression should be disabled. The threshold N_{thres} is kept in a volatile register R_{thres} which is also JIT checkpointed to NVFF on power failure, as shown in the figure.

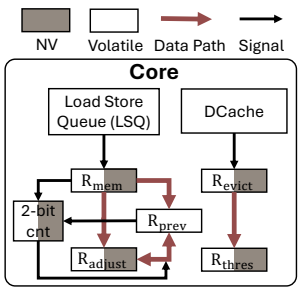


Fig. 7: Overall architecture of Kagura.

Sophisticated Approach: While the simple approach is straightforward, it is not adaptive to varying program behaviors or energy conditions. This is because it assumes that EHSs execute the same number of memory instructions in two consecutive power cycles. However, this assumption does not always hold, leading to the inaccurate computation of N_{remain} and causing performance penalties. On the one hand, if the number of memory instructions of the previous power cycle is less than that of the current cycle, N_{remain} is *underestimated*. This causes cache compression to be prematurely turned off, tragically resulting in performance loss. On the other hand, if N_{remain} is *overestimated*, cache compression is turned off too late, wasting a significant portion of hard-won energy on compressing useless data blocks in vain.

To mitigate the aforementioned issues, Kagura utilizes *Reward and Punishment Mechanism* to accurately calculate N_{remain} for each power cycle. Kagura is *rewarded* if the computed N_{remain} is close to the actual number of committed memory instructions in the current power cycle, and *punished* otherwise. To implement this idea, Kagura uses a simple per-core 2-bit saturating counter as shown in Figure 7. When the difference between the computed and the actual number is small, Kagura increases the counter by 1 as a reward. Otherwise, Kagura decreases the counter by 1 as a punishment; this saturation counter is also JIT checkpointed to NVFF right before power failure occurs. After waking up from power failure, Kagura consults the saturation counter and applies an adjustment to R_{prev} if the counter equals 00 or 01.

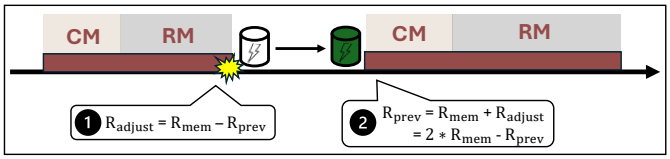


Fig. 8: R_{adjust} calculation and application.

The next step is to determine the value of the adjustment. Kagura leverages a learning-based method that tracks the difference between the estimated and actual numbers, and then applies this difference as an adjustment to future estimation. To achieve a lightweight implementation, Kagura introduces another register R_{adjust} for recording the difference between R_{mem} and R_{prev} (❶ in Figure 8). That is, Kagura computes R_{adjust} at the end of each power cycle, as below:

$$R_{adjust} = R_{mem} - R_{prev} \tag{6}$$

In the wake of the power failure, Kagura first restores R_{mem} and R_{adjust} and then updates R_{prev} as the sum of R_{mem} and R_{adjust} (❷ in Figure 8). This lightweight approach allows Kagura to ensure the estimation of N_{remain} is continually

refined based on the discrepancy between the actual and estimated memory operation counts.

B. Adaptive Threshold Tuning for Compression Disabling

To adjust the compression-disabling threshold N_{thres}, a simple approach is to select a number empirically upon reboot and leave it unchanged during program execution. However, the naive approach often leads to suboptimal performance in that setting the threshold too high causes compression to be halted prematurely, lowering effective cache capacity; this can result in more evictions when new blocks are filled in, ultimately degrading performance. Conversely, setting the threshold too low wastes hard-won energy, compressing cache blocks that are never accessed again before being lost due to power failure. Consequently, Kagura seeks to figure out a suitable compression-disabling threshold that balances energy efficiency and performance.

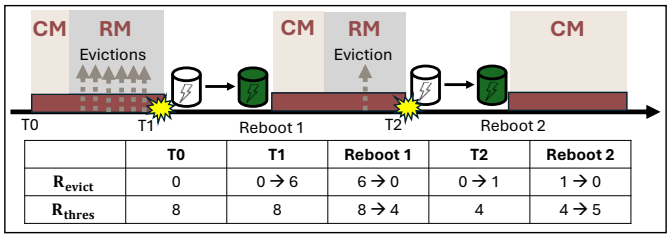
To achieve this, Kagura monitors the evicted block count to adjust N_{thres}. When a high number of evictions occurs, indicating cache capacity is insufficient in the current cycle, Kagura lowers the threshold. That way, Kagura performs cache compression more aggressively, increasing effective cache capacity in the next power cycle. On the other hand, a low eviction count indicates sufficient cache room in the current cycle, and therefore Kagura raises the threshold to allow for greater energy savings in the next power cycle.

Kagura implements the above mechanism using another register R_{evict}, as shown in Figure 7. R_{evict} is JIT checkpointed to NVFF on power loss and tracks the number of blocks evicted since the decision point. Once power comes back, i.e., a new power cycle begins, Kagura uses the restored R_{evict} value to adjust R_{thres} following a so-called *Additive Increase/Multiplicative Decrease (AIMD)* as with other adaptation proposals [2], [5], [6], [83], [99], [134]. That is, Kagura halves R_{thres} if R_{evict} is larger than half of R_{thres}. Otherwise, Kagura increases R_{thres} by 10%, which averts raising R_{thres} too quickly and thus in turn maintains low cache miss rates.

Figure 9 shows how Kagura adjusts R_{thres}. For the sake of example, Kagura initializes R_{thres} with 8 and R_{evict} with 0, respectively. At the end of the first power cycle at time T_1, R_{evict} is set to 6 because 6 blocks have been evicted. Here, R_{evict} is greater than half of 8, and upon reboot (Reboot 1 in the figure), Kagura thus lowers R_{thres} to 4 and resets R_{evict} to 0. During the second power cycle ending at time T_2, only one block is evicted, updating R_{evict} to 1. In the beginning of the third power cycle (Reboot 2 in the figure), Kagura increases R_{thres} from 4 to 5 since R_{evict} (1) is less than half of 4.

C. Putting It All Together

Figure 10 illustrates Kagura's overall operations. Initially, the registers are set as follows: $R_{mem} = 20$, $R_{adjust} = 5$, $R_{thres} = 8$, and $R_{evict} = 1$. At the start of a power cycle at time T_0, Kagura restores R_{prev} to the value of R_{mem}, capturing the committed memory operation count in the previous power cycle, and then resets it. At time T_1, Kagura applies R_{adjust} to R_{prev}, updating it from 20 to 25.



	T0	T1	Reboot 1	T2	Reboot 2
R_{evict}	0	0 → 6	6 → 0	0 → 1	1 → 0
R_{thres}	8	8	8 → 4	4	4 → 5

Fig. 9: Timeline of adaptively adjusting R_{thres}.

	T0	T1	T2	T3	Outage	T4	T5
R_{prev}	→ 20	20 → 25	25	25		→ 22	22 → 19
R_{mem}	20 → 0	0	0 → 16	16 → 22	22 (ckpt)	22 → 0	0
R_{adjust}	5	5	5	5 → -3	-3 (ckpt)	-3	-3
R_{thres}	8	8 → 9	9	9	9 (ckpt)	9	9 → 5
R_{evict}	1	1 → 0	0	0 → 6	6 (ckpt)	6	6 → 0

Fig. 10: An example of dynamic switching operation modes with registers updated; T_2 is a decision point.

Next, Kagura evaluates whether to adjust R_{thres}. Since R_{evict} is less than half of R_{thres}. Kagura increases R_{thres} to 9 and resets R_{evict} to 0. The core pipeline then runs until it reaches a decision point at time T_2 where the difference $R_{prev} - R_{mem}$ equals R_{thres}. Kagura then enters into *RM* mode with cache compression disabled, and begins counting block evictions using R_{evict}. By the end of the power cycle at time T_3, 6 blocks have been evicted, and 22 memory operations have been committed. At this point, Kagura updates R_{adjust} by subtracting $R_{prev}(25)$ from $R_{mem}(22)$—resulting in -3—as shown in Equation 6. Then, just before the impending power failure, Kagura JIT checkpoints all registers except R_{prev}. After power is restored at time T_4, Kagura begins the recovery process; it loads the saved R_{mem} value into R_{prev}, resets R_{mem} to 0, and at time T_5 completes the recovery by adjusting R_{prev} based on R_{adjust} (-3), halving R_{thres} (as $R_{evict} \leq R_{thres}$), and clearing R_{evict}.

VII. Discussion

A. Impact of Peripheral Operations

This paper only evaluates application kernel. However, in real-world systems, EHS workloads also include I/O operations. To ensure the freshness of inputs from peripherals [48], [95], [103], [137], existing EHSs typically group I/O operations and their related computations into atomic regions [23], [70], [113], [147], with an extra checkpoint, i.e., saving register states and dirty cache blocks to NVM, at the start of each region. During atomic region execution, EHSs temporarily disable JIT checkpointing to avoid taking inconsistent snapshots. As such, power-interrupted atomic regions can be restored by restoring to the beginning of the region and re-executing from there.

This region-level checkpointing incurs more energy consumption, bringing more opportunities for Kagura to avoid energy waste. The reason is that these checkpoints consume

extra energy, and therefore EHSs suffer more frequent outages, rendering many compressions useless as blocks go unused before power loss. As such, Kagura's dynamic disabling of compression thus prevents more energy waste. Moreover, by reducing the energy waste, Kagura allows EHS to commit more instructions for each power cycle (the bottom of Figure 13). These extra committed instructions make it more likely that an atomic region completes without interruption, thus reducing re-execution overhead and improving overall performance.

B. Impact on Batteryless Artificial Intelligence of Things (AIoT) Systems

Batteryless AIoT systems [47], [60], [73], [74], [140], [174] adopt machine learning for EHSs to achieve intelligent edge computation. With the help of Kagura, the AIoT devices can improve the energy efficiency and the quality of service (QoS). That is because the AIoT workloads, particularly those involving machine learning inference, exhibit high memory intensity yet require low latency for the QoS; data cache helps satisfy the low latency requirement, and compressing the cache allows the AIoT device to use a larger model while maintaining the QoS. This gives Kagura more opportunities to prevent useless compressions and reduce the energy waste under variable conditions, thereby improving the QoS of the AIoT systems.

C. Impact of Checkpoint Region Size

Unlike NVSRAMCache [63], some EHSs do not rely on JIT checkpointing [24], [61], [70], [113], [118], [147], [184]. Instead, these systems perform checkpoints periodically but not necessarily at a fixed interval (region). They vary the checkpoint region size—adapting to different energy conditions—to maintain the performance. As the checkpoint region shrinks, the EHSs perform their checkpoints more frequently during execution, consuming additional energy and triggering more power outages, in which case Kagura has more opportunities to prevent useless compressions. Conversely, a larger checkpoint region reduces the number of outages and, thus, the chances for Kagura to intervene. In our evaluation, the baseline EHS employs JIT checkpointing, i.e., it performs a single checkpoint right before each power outage, minimizing the checkpoint number. Even under this scenario, Kagura still demonstrates notable performance gains.

VIII. Evaluation and Experimental Analyses

We implement Kagura and several other cache compressors atop gem5 [26]—a cycle-level simulator—to model a single-core, in-order processor clocked at 200 MHz. All evaluated applications are compiled for ARMv7-M instruction set and statically linked. To estimate energy use, we leverage the low-power cell (LOP) provided in McPAT [104] with 45 nm technology, following prior work [112]. As our baseline, we evaluate NVSRAMCache [63]—a standard JIT-checkpoint-based EHS—without cache compression and model the voltage monitor's initialization overhead, propagation latency, and energy consumption. To assess the benefit of compression, we integrate ACC into both the ICache and DCache of NVSRAMCache. Finally, we enable Kagura for ACC to measure the extra gains Kagura brings to that design. Table I lists all of the key simulation parameters for each configuration.

TABLE I: Simulation Configuration.

	No Compressor	**ACC**	**Kagura**
Energy buffer	4.7 μF capacitor		
Core	Single-core in-order five-stage pipeline		
Clock Rate	200 MHz		
ICache/DCache	256B 2-way SRAM with 32B block size, LRU replacement, 1 cycle hit latency, write-back policy, access: 9 pJ		
Compression Algorithm	N/A	BDI, compress: 3.84 pJ, decompress: 0.65 pJ	
Main Memory	16MB ReRAM		
ReRAM config.	tCK/tBURST/tRCD/tCL/tWTR/tWR/tXAW = 0.94/7.5/18.0/15.0/7.5/150/30		

We utilize 20 applications from Mediabench [97] and MiBench [66] for evaluation [107]. To see how Kagura performs under ambient energy conditions, we conduct simulations using a real-world RF power trace (RFHome in Figure 11) [63]. To ensure consistent energy input across different configurations, we use an energy harvester to collect ambient energy and record the average power values in a text file. Each entry in the file represents the average power over a 10μs interval (i.e., $P_{avg} = E_{10\mu s}/10\mu s$, where $E_{10\mu s}$ is the energy harvested during that interval). During each simulation, the system reads these values to charge the capacitor and power the processor, guaranteeing that all experiments use the same energy budget for fair comparisons.

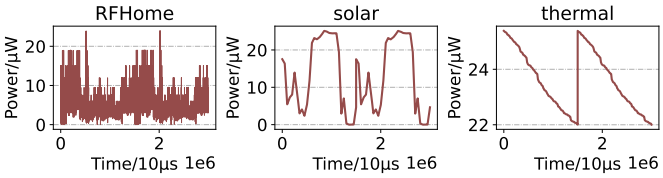


Fig. 11: Ambient power traces.

A. Hardware Overhead Analysis

Kagura only requires a little hardware because of its simple logic: only a 2-bit counter and five 32-bit registers—R_{mem}, R_{thres}, R_{prev}, R_{adjust} and R_{evict}—for a total of 162 bits. According to CACTI [151] at 45 nm, these registers take up at most 0.000796 mm^2 which is only 0.14% of the 0.538 mm^2 core (including caches) as reported by McPAT [104].

B. Program Behaviors Across Power Cycles

We observe that program behavior remains consistent between two neighboring power cycles. To show this consistency, we measure how they differ in committed load/store counts and CPI (cycles per instruction). The bar chart of Figure 12 shows that the committed load, store and CPI differ by only 5.73%, 14.11% and 5.26% on average, respectively. The observation is also supported by the line chart of the figure which depicts the percentage of the neighboring cycles that have less than 20% difference. The result shows that the majority of the neighboring cycles have less than 20% differences in load

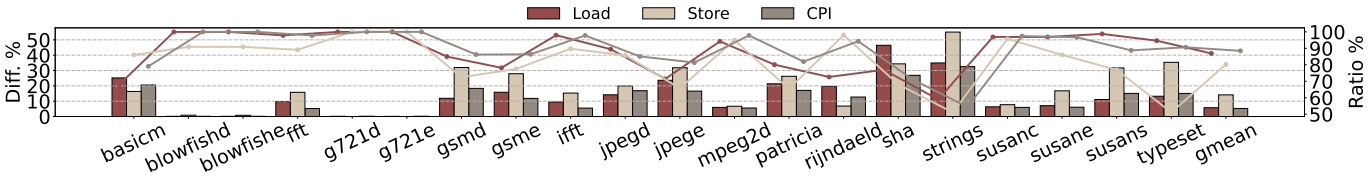


Fig. 12: Program behavior between 2 neighboring power cycles; bars (left y-axis) show the difference in load/store counts and CPI between two cycles, while lines (right y-axis) show the percentage of neighboring cycles that have $< 20\%$ difference.

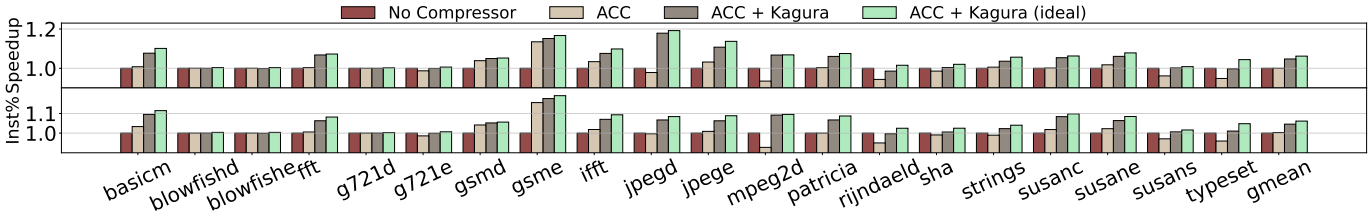


Fig. 13: Speedup (top) and average committed instruction count increase per power cycle (bottom) over NVSRAMCache.

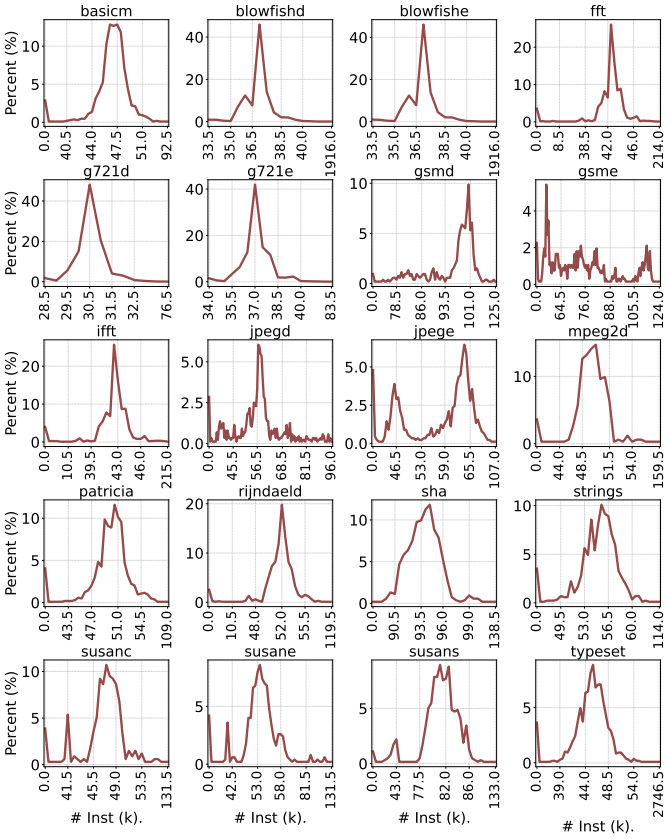


Fig. 14: Power cycle length distribution for each application.

(86.91%), store (80.27%) and CPI (88.48%), serving as a basis for Kagura to make decisions based on the previous power cycle. Furthermore, Figure 14 presents the power cycle length distribution for each application; y-axis shows the probability density of a given power cycle length shown in x-axis denoting the number of committed instructions (in thousands). The results show that most power cycles have comparable length, providing a foundation for Kagura to reference the previous cycle when estimating the new one.

C. Run-Time Performance

The top of Figure 13 depicts performance comparison of a compressor-free baseline, ACC, and ACC with Kagura across various benchmarks. On average, ACC alone delivers a 0.0022% performance improvement over the baseline. This trivial gain is due to frequent power failure which causes the loss of compressed cache data and renders the compression effort useless. Moreover, the energy overhead from compression and decompression further degrades the performance benefits. When Kagura is combined with ACC, the overall performance gain increases to 4.74% compared to the baseline. Figure 13 also plots the ideal gains of ACC and Kagura—i.e., assuming perfect knowledge of when to disable compression. To obtain this ideal case, we execute ACC+Kagura on each application using the RFHome trace in two phases. In the first run, we record a detailed trace capturing whether each compression operation actually contributes to cache hits and whether omitting it would lead to additional misses. In the second run, the simulator uses this trace as input, allowing the ideal system to adaptively decide in advance whether to perform each compression based on the recorded outcomes. Under this ideal scenario, performance improves by 6.19% over the baseline, just 1.39% above the ACC + Kagura. Also, the bottom of Figure 13 shows average committed instruction counts per power cycle. Compared to the baseline, ACC alone executes 0.28% more committed instructions for each power cycle on average. When combined with Kagura, the average committed instruction significantly increases to 4.57%.

Notably, both ACC and its combination with Kagura show limited improvements for some applications, including *blowfishd*, *blowfishe* and *g721d*. This is because these programs do not heavily rely on cache resources, so ACC naturally reduces compression. With fewer compression operations, Kagura has fewer opportunities to optimize performance and energy by averting useless compressions. Additionally, ACC performs worse than the baseline for some applications, e.g., *jpegd*, *mpeg2d*, *susans* and *typeset*, as it often unnecessarily compresses blocks that receive no hits before being lost

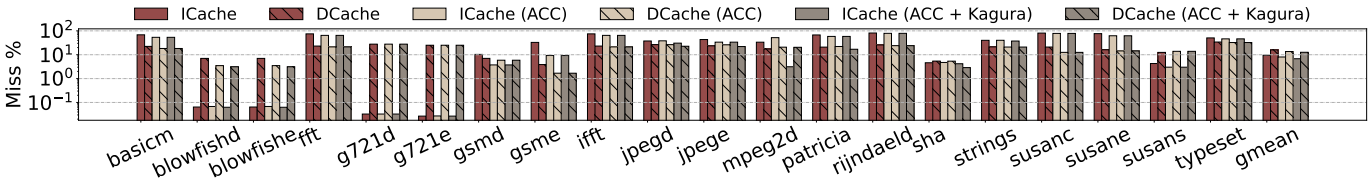


Fig. 15: Cache miss rates comparison.

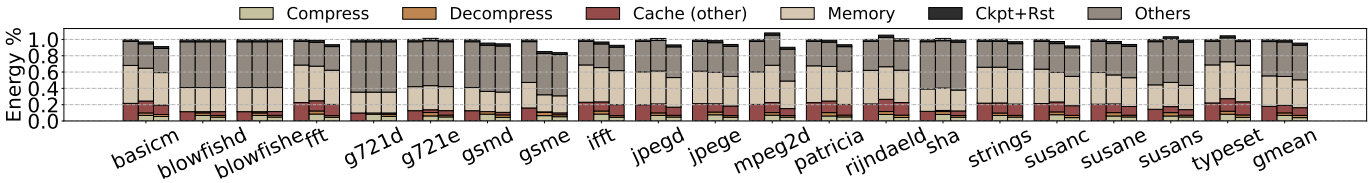


Fig. 16: Normalized energy breakdown to compressor-free NVSRAMCache (baseline). *Cache (other)* in the legend means all the cache energy consumption except for compression and decompression. *Others* includes CPU pipeline energy and energy buffer leakage. There are three bars for each application. *From left to right: NVSRAMCache, ACC, and ACC + Kagura*.

on power outages. Fortunately, Kagura identifies and avoids these useless compressions, allowing ACC to achieve better performance.

D. Impact of Arithmetic Intensity

Arithmetic intensity, defined as the ratio of arithmetic to memory operations, is a key factor influencing Kagura's effectiveness. We select six applications with varying arithmetic intensities to examine the relationship between performance and arithmetic intensity. As shown in Figure 17, Kagura's performance improvement exhibits a clear inverse relationship with arithmetic intensity, i.e., performance increases as arithmetic intensity decreases. This trend arises because applications with more memory operations (e.g., *jpegd* and *jpege*) generate frequent cache activity, allowing Kagura's adaptive compression control to prevent useless compressions and improve energy efficiency. In contrast, workloads with fewer memory operations (e.g., *patricia* and *strings*) benefit less from Kagura, as compression overhead plays a smaller role in their overall execution.

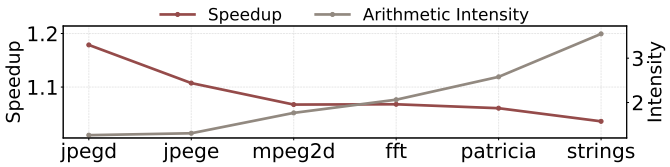


Fig. 17: Performance vs. arithmetic intensity across applications

E. Cache Miss Rate

Figure 15 shows how ACC and Kagura affect cache miss rates across various applications. Using only ACC reduces 1.45% ICache miss rates and 2.29% for DCache. When we combine Kagura with ACC, the miss rates decrease even further (2.71% for ICache and 3.24% for DCache), since most compressions averted by Kagura are useless which does not negatively impact cache hits. Furthermore, by preventing these useless compressions, Kagura saves valuable energy, allowing EHSs to make more forward progress before encountering power outages. Consequently, when running the same application, Kagura allows EHSs to experience fewer power outages; that is, Kagura reduces the number of cache data loss and thus allows cache blocks to contribute to more cache hits.

F. Compression Operation Reduction

Figure 18 shows the number of compression operations eliminated by Kagura when it is applied to ACC. On average, Kagura cuts down compression operations by $\approx$ 9.85% and over 40% for some applications like *g721d* and *g721e*. However, a high cache compression reduction ratio does not translate to significant performance improvements or energy savings. This is because (1) Kagura might occasionally avert useful compressions and thus lead to more cache misses, (2) and if compression and decompression account for a small portion of the total energy cost, then even eliminating useless compressions may only yield trivial gains.

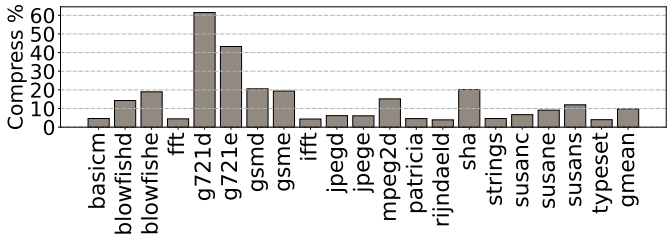


Fig. 18: Compression reduction ratio by Kagura.

G. Energy Efficiency

To clearly show energy savings provided by Kagura, we normalize its total energy consumption to that of baseline and break it down into six portions: *Compress, Decompress, Cache (other), Memory, Checkpoint/Restoration and Others* as shown in Figure 16. The results show that compression and decompression incur average energy overheads of 6.88% and 3.06% for ACC (relative to the baseline's total energy). After enabling Kagura, the compression and decompression energy overheads fall to 4.12% and 2.75%, and the total energy

consumption is reduced by 4.53% on average.

H. Sensitivity Analysis

1) EHS Designs

Figure 19 compares NVSRAMCache, NvMR [24] and SweepCache [184] with and without ACC and Kagura enabled. All speedups are normalized to each EHS. Although this figure has two bars of *+ ACC + Kagura*, we only focus on the memory-based variant (*+ ACC + Kagura (mem)*) which is the default design choice for Kagura. More details about memory-based and voltage-based variants (*+ ACC + Kagura (vol)*) are in Section VIII-H2. For a fair comparison, we calibrated NvMR and SweepCache to achieve their optimal performance under our default settings. For NvMR, we adjusted its map table, map table cache, and free list to 128, 16 and 145 entries. For SweepCache, we reduced persist buffers to 32 entries and recompiled the application to regenerate boundaries. With ACC applied, the gain of NvMR and SweepCache is slightly improved by 0.82% and 0.18%, respectively. After we further apply Kagura, the speedups increase to 5.54% and 3.15%.

2) Trigger Strategies for Kagura

By default, we leverage committed memory operations to decide the suitable time to trigger Kagura before power failure (denoted as *+ ACC + Kagura (mem)* in Figure 19). In the literature, an alternative method is voltage-based trigger (*+ ACC + Kagura (vol)*) [54], [55], which halts compression whenever capacitor voltage drops below a predefined threshold signaling an imminent power failure. A key drawback of the voltage-based trigger is its heavy dependence on a complex monitor that must track three separate thresholds—backup, restoration, and Kagura's trigger. In contrast, our memory-based trigger imposes no extra voltage-monitoring requirements and can even work in JIT-checkpoint-free EHSs that forgo complex monitors. This is especially valuable because many designs—including those in [41]–[43], [71], [118], [158], as well as recent work like SweepCache [184] and NvMR [24]—strive to avoid such complex monitors because of their extra energy cost (8.5% of total energy consumption as reported in [53]), hardware complexity, and security vulnerability [38]–[40], [64].

To assess how trigger methods affect various EHS designs, we reproduce NvMR and SweepCache (see Section VIII-H1). Figure 19 shows that on NVSRAMCache, the voltage-based trigger achieves a very similar performance gain as the memory-based one. However, for NvMR and SweepCache, the voltage-based trigger degrades the performance of ACC by 0.23% and 2.81% due to its heavy dependency on voltage monitoring. On the contrary, the memory-based trigger can still achieve good performance (5.54% for NvMR and 3.15% for SweepCache).

3) Integration with Other Cache Managements

Researchers introduce some other cache management for EHSs, including dead block prediction (EDBP [54]) and prefetching (IPEX [55]). We reproduce these two works and evaluate them atop our baseline (without any dead block predictors and prefetchers) with the same configuration in Table I; our EDBP implementation is built on top of Cache Decay [87]. Figure 20 shows the performance impact of combining cache compressors with EDBP and IPEX. As shown in Figure 20, compared to the baseline, EDBP achieves a performance gain of 5.32%, which increases to 12.14% when it is combined with ACC and Kagura. Besides, applying ACC and Kagura to IPEX improves its performance from 12.73% to 18.37%.

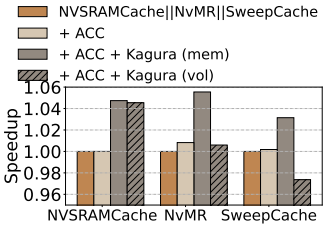


Fig. 19: Different trigger methods on different EHSs.

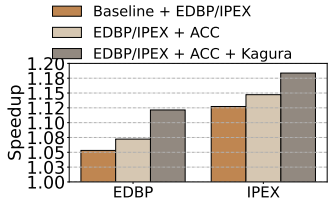


Fig. 20: Kagura with different cache managements.

4) Adaptation Schemes for R_{thres}

Kagura empirically selects AIMD policy to adaptively adjust R_{thres}. We also evaluate another 3 adaptation policies: Multiplicative Increase/Additive Decrease (MIAD), Additive Increase/Additive Decrease (AIAD), and Multiplicative Increase/Multiplicative Decrease (MIMD). Figure 21 shows that MIAD and MIMD perform poorly since aggressive threshold increase can suppress useful compressions, leading to more cache misses. Thus, using additive increase is more suitable for R_{thres} adjustment. Besides, the policies incorporating a multiplicative decrease allow immediate threshold reduction and thus avert unnecessary cache misses. Combining these analyses, we select AIMD for R_{thres} adjustment (marked by a red star in the figure).

5) Increase Step for R_{thres}

Note that the increase step of R_{thres} defaults to 10%. To show how it affects performance, we vary the increase step from 5% to 20%. Figure 22 shows that small (5%) and large (15% and 20%) steps are either too conservative or aggressive and cannot achieve the best performance. Thus, Kagura picks a 10% step to strike a balance between energy saving and compression efficiency.

6) Power Cycle Number for Memory Operation Estimation

TABLE II: Performance impact of different numbers of power cycles used for memory estimation.

# Cycle	**1** (default)	**2**	**3**	**4**
Speedup	4.74%	4.09%	3.35%	2.60%

By default, Kagura uses the memory-operation count from the previous power cycle to estimate that of the current one. In this subsection, we vary the number of past power cycles being examined from 1 to 4 to see how it impacts performance. We compute a weighted average of the memory operations in each previous power cycle. That is, more recent cycles receive a higher weight. For example, when using two cycles (with C_2 being more recent than C_1), we give C_2 twice the weight of C_1 (i.e., $N_{prev} = \frac{C_1+2*C_2}{1+2}$). Table II shows

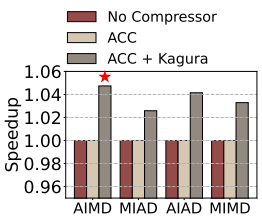


Fig. 21: Adapt Scheme.

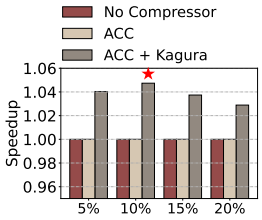


Fig. 22: R_{thes} size.

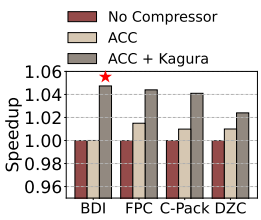


Fig. 23: Compressor.

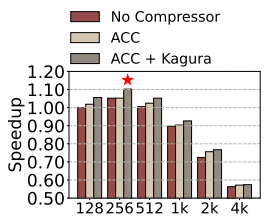


Fig. 24: Cache size(B).

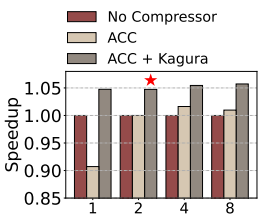


Fig. 25: Cache way.

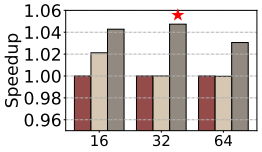


Fig. 26: Block size.

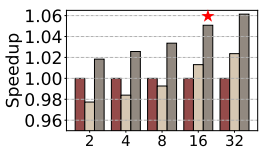


Fig. 27: Mem size(MB).

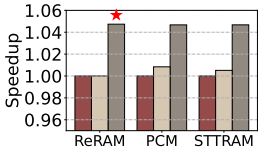


Fig. 28: Mem type.

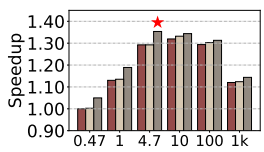


Fig. 29: Capacitor(μF).

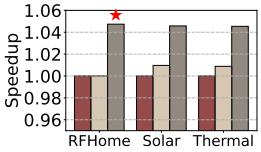


Fig. 30: Power trace.

that examining only one prior power cycle delivers the best performance because program behaviors remain consistent across the previous power cycle (see Section VIII-B).

7) Compression Algorithms

We evaluate Kagura's effectiveness across different compression algorithms, including BDI [131], FPC [8], C-Pack [35], and DZC [160]. As illustrated in Figure 23, with different compression algorithms, the performance of ACC over the baseline is 0.0022% (BDI), 1.50% (FPC), 0.99% (C-Pack), and 1.00% (DZC). When combined with Kagura, these improvements increase to 4.74% (BDI), 4.40% (FPC), 4.10% (C-Pack), and 2.41% (DZC), respectively, i.e., Kagura effectively reduces useless compressions across all tested algorithms.

8) Cache Sizes

We tested various cache sizes ranging from 128B to 4kB as shown in Figure 24. All values are normalized to the NVSRAMCache baseline with 128B caches. Across all tested cache sizes, the combination of ACC and Kagura always delivers significant performance improvements ranging from 1.97% to 5.85%. We observe that Kagura yields greater benefits with smaller caches: limited capacity forces ACC to compress more blocks, and frequent power interruptions then render many of these compressions wasted—conditions under which Kagura is most effective. Moreover, Kagura 's gains shrink with larger caches, since they hold more uncompressed data and trigger fewer compressions, leaving fewer useless operations for Kagura to eliminate.

9) Cache Ways

We evaluated how cache associativity affects Kagura by testing the same cache size with different numbers of ways, from direct-mapped to 8-way set-associative, as shown in Figure 25. Overall, the combination of ACC and Kagura consistently improves performance across all cache designs, delivering gains ranging from 4.74% to 5.73%.

10) Cache Block Sizes

Cache block size critically affects the efficiency of Kagura and other compression schemes: larger blocks reduce the number of compressions but increase the energy and latency per operation, while smaller blocks lower per-compression cost at the expense of more frequent compressions. As Figure 26 demonstrates, Kagura maintains good performance across block sizes from 16B to 64B.

11) Main Memory Sizes

We evaluate Kagura across different main memory sizes as shown in Figure 27. It shows that Kagura always helps ACC achieve good performance. However, the performance gain from Kagura becomes smaller when the memory size gets larger, decreasing from 4.22% to 3.69% as the memory size grows from 2MB to 32MB. This happens because larger NVM increases the energy cost per cache miss, potentially offsetting the savings of Kagura's elimination of useless compression.

12) Main Memory Types

Different NVM types have different cache miss penalties, which may affect the performance of Kagura. To verify the effectiveness of Kagura, we use three types of NVM as shown in Figure 28. The results reveal that Kagura can always deliver a promising speedup with all evaluated NVMs, e.g., a 4.67% speedup with PCM and a 4.68% speedup with STTRAM.

13) Capacitor Sizes

TABLE III: Capacitor leakage over total energy cost.

Cap(μF)	**0.47**	**1**	**4.7**	**10**	**100**	**1k**
Leak	0.001%	0.002%	0.01%	0.03%	0.28%	5.91%

Capacitor size affects how often power interruptions occur in EHSs. Larger capacitors store more energy and extend power cycles but at the cost of longer charging time and higher leakage. Table III shows that capacitor leakage grows significantly with size—for instance, a 1000 μF capacitor accounts for 5.91% of total energy. Smaller capacitors charge quickly and leak less, but they run out faster, leading to more frequent outages and associated JIT checkpoint and restoration costs. Capacitor size also affects how well cache compression works. With small capacitors where power cycle is short, compressor has fewer chances to make compression. The reason is that in the wake of power failure, EHS restarts with empty caches, so cache blocks are allocated uncompressed until the cache set fills up. At that point, there may be only a few memory operations before the next outage, resulting

in only limited opportunities for compression. In contrast, larger capacitor provides longer power cycles, allowing more compression opportunities once the cache set is full.

Figure 29 shows that Kagura consistently brings benefits across all capacitor sizes. All values are normalized to NVSRAMCache with a 0.47 μF capacitor. We find that as the capacitor increases from 0.47 μF to 4.7 μF, the performance benefit of Kagura over ACC grows—initially limited by the fewer compression opportunities at 0.47 μF because short power cycles provide few compressions and thus offer limited opportunities for Kagura to eliminate useless compressions. Kagura averts more useless compressions when the capacitor reaches 4.7 μF. However, as the capacitor further increases over 4.7 μF, the gap narrows: larger capacitors mean fewer outages and compressions, and thus fewer useless compressions for Kagura to eliminate, so Kagura's benefit over ACC diminishes. Overall, we find that a 4.7 μF capacitor can help EHS achieve the best performance gain across all evaluated capacitors with the help of ACC and Kagura. Thus, we select it as the default capacitor size.

14) Power Traces

We evaluate Kagura using three different power traces [63], [135] as shown in Figure 30. Solar and thermal sources have relatively higher portions of stable energy, while RFHome has less (as shown in Figure 11). The results show that Kagura consistently improves the performance of ACC across various energy conditions. When enabling ACC with Kagura, the performance over the baseline is 4.74%, 4.58% and 4.54% with RFHome, solar and thermal.

15) Counter Bits

TABLE IV: Performance impact of different counter sizes.

# Bits	**1**	**2** (default)	**3**
Speedup	3.98%	4.74%	4.21%

Kagura employs a 2-bit saturating counter by default to update R_{prev} at the beginning of each power cycle (Section VI-A). We also evaluate Kagura with 1-bit and 3-bit counters. As shown in Table IV, these counters exhibit suboptimal performance since they are either too aggressive or too conservative in adjusting R_{prev}. Consequently, Kagura adopts the 2-bit counter as its default setting, providing the best balance between responsiveness and stability.

IX. Other Related Work

Many proposals have introduced cache compression to reduce cache misses [4], [11]–[13], [16], [22], [25], [30], [32], [34], [50]–[52], [58], [59], [68], [72], [77], [86], [90], [92], [94], [100], [101], [116], [117], [120]–[126], [129], [130], [139], [141], [142], [148]–[150], [152], [152], [153], [155], [159], [163], [164], [167]–[169], [172], [175], [183]. For example, Bit-Plane Compression (BPC) [91] compresses data by first computing small differences (deltas) between neighboring values, reorganizing these deltas into bit-planes, and then using XOR between adjacent bit-planes to create long runs of zeros and other recurring patterns that can be encoded. Upon access, BPC decodes the pattern to get deltas, reverses the XOR and bit-plane transformation, and finally performs a cumulative sum from the base value to recover the original data. CC [171] replaces frequent values in a block with short codes while leaving rare values in place with pointers. On each access, CC uses a simple mask to tell compressed entries, and maps codes back to full values. DISH [127] divides a 64B block into sixteen 4B chunks and builds a shared dictionary—either storing up to eight full chunks with fixed-width pointers or using 28-bit keys plus low-bit offsets. Indirect Index Cache with Compression (IIC-C) [68] breaks blocks into variable-sized subblocks stored indirectly via multiple pointers, allowing saved space to be flexibly reallocated to other data. This ensures that frequently accessed blocks are decompressed on demand to mitigate latency penalties, while data transferred on the bus and held in main memory remains compressed to reduce bandwidth usage.

X. Conclusion

This paper introduces Kagura, an adaptive cache compression extension that improves the efficiency of EHSs by taking into account their frequent power outages. Specifically, Kagura monitors the number of remaining memory operations in each power cycle to decide whether to enable or disable cache compression. When upcoming power interruption is likely, Kagura disables compression to avoid wasting energy on data that will not be reused before the outage. Experimental results demonstrate that integrating Kagura with an existing adaptive compression scheme yields an average energy reduction of 4.53% and a performance improvement of 4.74% compared to baseline EHS that lacks cache compression.

Acknowledgment

We appreciate anonymous reviewers for their insightful and valuable comments. This work is in part supported by NSF grants 2153749 and 2314681.

References

[1] H. Aantjes, A. Y. Majid, and P. Pawełczak, "A testbed for transiently powered computers," in *arXiv preprint*, 2016.

[2] S. Ahmad, H. Guan, B. D. Friedman, T. Williams, R. K. Sitaraman, and T. Woo, "Proteus: A high-throughput inference-serving system with accuracy scaling," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, 2024.

[3] S. Ahmed, B. Islam, K. S. Yildirim, M. Zimmerling, P. Pawełczak, M. H. Alizai, B. Lucia, L. Mottola, J. Sorber, and J. Hester, "The internet of batteryless things," *Communications of the ACM*, vol. 67, no. 3, 2024.

[4] E. Ahn, S.-M. Yoo, and S.-M. S. Kang, "Effective algorithms for cache-level compression," in *Proceedings of the 11th Great Lakes symposium on VLSI*, 2001.

[5] S. Ainsworth and T. M. Jones, "Paramedic: Heterogeneous parallel error correction," in *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2019.

[6] S. Ainsworth, L. Zoubritzky, A. Mycroft, and T. M. Jones, "Paradox: Eliminating voltage margins via heterogeneous fault tolerance," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2021.

[7] K. Akhunov, K. S. Yildirim, J. Choi, and C. Jung, "Adaptive computing in memory meets conventional batteryless platforms," *ACM Transactions on Embedded Computing Systems*, vol. 24, no. 6, 2025.

[8] A. Alameldeen and D. Wood, "Frequent pattern compression: A significance-based compression scheme for l2 caches," University of Wisconsin-Madison, Tech. Rep., 2004.

[9] A. R. Alameldeen and R. Agarwal, "Opportunistic compression for direct-mapped dram caches," in *Proceedings of the International Symposium on Memory Systems*, 2018.

[10] A. Alameldeen and D. Wood, "Adaptive cache compression for high-performance processors," in *Proceedings. 31st Annual International Symposium on Computer Architecture, 2004.*, 2004.

[11] L. M. AlBarakat, P. V. Gratz, and D. A. Jiménez, "Slap-cc: Set-level adaptive prefetching for compressed caches," in *2022 IEEE 40th International Conference on Computer Design (ICCD)*, 2022.

[12] C. Aliagas, C. Molina, M. Garcia, A. Gonzalez, and J. Tubella, "Value compression to reduce power in data caches," in *Euro-Par 2003 Parallel Processing: 9th International Euro-Par Conference Klagenfurt, Austria, August 26-29, 2003 Proceedings 9*, 2003.

[13] A. Arelakis, F. Dahlgren, and P. Stenstrom, "Hycomp: A hybrid cache compression method for selection of data-type-specific compression methods," in *Proceedings of the 48th International Symposium on Microarchitecture*, 2015.

[14] A. Arelakis and P. Stenstrom, "Sc2: A statistical compression cache scheme," in *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, 2014.

[15] A. Arelakis and P. Stenström, "A case for a value-aware cache," *IEEE Computer Architecture Letters*, vol. 13, no. 1, 2014.

[16] A. Arunkumar, S.-Y. Lee, V. Soundararajan, and C.-J. Wu, "Latte-cc: Latency tolerance aware adaptive cache compression management for energy efficient gpus," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2018.

[17] A. Bakar, R. Goel, J. De Winkel, J. Huang, S. Ahmed, B. Islam, P. Pawełczak, K. S. Yıldırım, and J. Hester, "Protean: An energy-efficient and heterogeneous platform for adaptive and hardware-accelerated battery-free computing," in *Proceedings of the 20th ACM Conference on Embedded Networked Sensor Systems*, 2022.

[18] A. Bakar, R. Goel, J. de Winkel, J. Huang, S. Ahmed, B. Islam, P. Pawelczak, K. S. Yildirim, and J. Hester, "Protean: Adaptive hardware-accelerated intermittent computing," *GetMobile: Mobile Computing and Communications*, vol. 27, no. 1, 2023.

[19] D. Balsamo, A. S. Weddell, G. V. Merrett, B. M. Al-Hashimi, D. Brunelli, and L. Benini, "Hibernus: Sustaining computation during intermittent supply for energy-harvesting systems," in *IEEE Embedded Systems Letters 7, 1 (2014), 15–18*, 2014.

[20] S. Beeby and N. White, "Energy harvesting for autonomous systems," in *Artech House, Incorporated.*, 2014.

[21] L. Benini, D. Bruni, A. Macii, and E. Macii, "Hardware-assisted data compression for energy minimization in systems with embedded processors," in *Proceedings 2002 Design, Automation and Test in Europe Conference and Exhibition*, 2002.

[22] L. Benini, A. Macii, and A. Nannarelli, "Cached-code compression for energy minimization in embedded processors," in *ISLPED'01: Proceedings of the 2001 International Symposium on Low Power Electronics and Design (IEEE Cat. No.01TH8581)*, 2001.

[23] G. Berthou, T. Delizy, K. Marquet, T. Risset, and G. Salagnac, "Peripheral state persistence for transiently-powered systems," in *2017 Global Internet of Things Summit (GIoTS)*, 2017.

[24] A. Bhattacharyya, A. Somashekhar, and J. S. Miguel, "Nvmr: Non-volatile memory renaming for intermittent computing," in *Proceedings of 49th International Symposium on Computer Architecture*, 2022.

[25] E. Billo, R. Azevedo, G. Araujo, P. Centoducatte, and E. W. Netto, "Design of a decompressor engine on a sparc processor," in *2005 18th Symposium on Integrated Circuits and Systems Design*, 2005.

[26] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti *et al.*, "The gem5 simulator," *ACM SIGARCH computer architecture news*, 2011.

[27] J. Bito, R. Bahr, J. G. Hester, S. A. Nauroze, A. Georgiadis, , and M. M. Tentzeris, "A novel solar and electromagnetic energy harvesting system with a 3-d printed package for energy efficient internet-of-things wireless sensors," in *IEEE Transactions on Microwave Theory and Techniques 65, 5 (2017)*, 2017.

[28] P. Cahill, R. O'Keeffe, N. Jackson, A. Mathewson, , and V. Pakrashi, "Energy-harvesting thermoelectric sensing for unobtrusive water and appliance metering," in *In Proceedings of the 2nd International Workshop on Energy Neutral Sensing Systems, ENSsys*, 2014.

[29] P. Cahill, R. O'Keeffe, N. Jackson, A. Mathewson, , and V. Pakrashi, "Structural health monitoring of reinforced concrete beam using piezoelectric energy harvesting system," in *In EWSHM-7th European workshop on structural health monitoring*, 2014.

[30] R. Canal, A. González, and J. E. Smith, "Value compression for efficient computation," in *Euro-Par 2005 Parallel Processing: 11th International Euro-Par Conference, Lisbon, Portugal, August 30-September 2, 2005. Proceedings 11*, 2005.

[31] S. Cao and J. Li, "A survey on ambient energy sources and harvesting methods for structural health monitoring applications," in *Advances in Mechanical Engineering 9, 4 (2017)*, 2017.

[32] D. R. Carvalho and A. Seznec, "Conciliating speed and efficiency on cache compressors," in *2021 IEEE 39th International Conference on Computer Design (ICCD)*, 2021.

[33] D. R. Carvalho and A. Seznec, "Understanding cache compression," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 18, no. 3, 2021.

[34] D. Chen, E. Peserico, and L. Rudolph, "A dynamically partitionable compressed cache," 2003.

[35] X. Chen, L. Yang, R. P. Dick, L. Shang, and H. Lekatsas, "C-pack: A high-performance microprocessor cache compression algorithm," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2010.

[36] Y. Chen, "Reram: History, status, and future," *IEEE Transactions on Electron Devices*, vol. 67, no. 4, 2020.

[37] Q. Cheng, Z. Peng, J. Lin, S. Li, , and F. Wang, "Energy harvesting from human motion for wearable devices," in *10th IEEE International Conference on Nano/Micro Engineered and Molecular Systems*, 2015.

[38] J. Choi, H. Joe, C. Jung, and J. Choi, "Defending against emi attacks on just-in-time checkpoint for resilient intermittent systems," in *57th International Symposium on Microarchitecture (MICRO)*, 2024.

[39] J. Choi, J. Choi, H. Joe, and C. Jung, "Caphammer: Exploiting capacitor vulnerability of energy harvesting systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.

[40] J. Choi, H. Joe, and C. Jung, "Capos: Capacitor error resilience for energy harvesting systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 11, 2022.

[41] J. Choi, H. Joe, Y. Kim, and C. Jung, "Achieving stagnation-free intermittent computation with boundary-free adaptive execution," in *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2019.

[42] J. Choi, L. Kittinger, Q. Liu, and C. Jung, "Compiler-directed high-performance intermittent computation with power failure immunity," in *2022 IEEE 28th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2022.

[43] J. Choi, Q. Liu, and C. Jung, "Cospec: Compiler directed speculative intermittent computation," in *In Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019.

[44] J. Choi, J. Zeng, D. Lee, C. Min, and C. Jung, "Write-light cache for energy harvesting systems," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023.

[45] Y.-W. Chong, W. Ismail, K. Ko, , and C.-Y. Lee, "Energy harvesting for wearable devices: A review," in *IEEE Sensors Journal 19, 20 (2019)*, 2019.

[46] H. Cılasun, S. Resch, Z. I. Chowdhury, M. Zabihi, Y. Lv, B. Zink, J.-P. Wang, S. S. Sapatnekar, and U. R. Karpuzcu, "On error correction for nonvolatile processing-in-memory," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, 2024.

[47] L. L. Custode, P. Farina, E. Yildiz, R. B. Kilic, K. S. Yildirim, and G. Iacca, "Fast-inf: ultra-fast embedded intelligence on the batteryless edge," in *Proceedings of the 22nd ACM Conference on Embedded Networked Sensor Systems*, 2024.

[48] Y. Dong, P. Fan, and K. B. Letaief, "Energy harvesting powered sensing in iot: Timeliness versus distortion," *IEEE Internet of Things Journal*, vol. 7, no. 11, 2020.

[49] V. Dsouza, J. Pronk, C. Peppelman, V. I. Madariaga, T. Pereira-Cenci, B. Loomans, and P. Pawełczak, "Densor: An intraoral battery-free sensing platform," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 8, no. 4, 2024.

[50] J. Dusser, T. Piquet, and A. Seznec, "Zero-content augmented caches," in *Proceedings of international conference on Supercomputing*, 2009.

[51] J. Dusser and A. Seznec, "Decoupled zero-compressed memory," in *Proceedings of the 6th International Conference on High Performance and Embedded Architectures and Compilers*, 2011.

[52] M. Ekman and P. Stenstrom, "A robust main-memory compression scheme," in *International Symposium on Computer Architecture*, 2005.

[53] G. Fang, J. Choi, and C. Jung, "Hybrid power failure recovery for intermittent computing," in *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, 2025.

[54] G. Fang and C. Jung, "Rethinking dead block prediction for intermittent computing," in *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2025.

[55] G. Fang, J. Zeng, A. Gupta, and C. Jung, "Rethinking prefetching for intermittent computing," in *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, 2025.

[56] P. Franaszek, J. Robinson, and J. Thomas, "Parallel compression with cooperative dictionary construction," in *Proceedings of Data Compression Conference - DCC '96*, 1996.

[57] T. Galchev, J. McCullagh, R. Peterson, and K. Najafi, "A vibration harvesting system for bridge health monitoring applications," in *In Proc. PowerMEMS*, 2010.

[58] A. Ghasemazar, M. Ewais, P. Nair, and M. Lis, "2dcc: Cache compression in two dimensions," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2020.

[59] A. Ghasemazar, P. Nair, and M. Lis, "Thesaurus: Efficient cache compression via dynamic clustering," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020.

[60] G. Gobieski, N. Beckmann, and B. Lucia, "Intermittent deep neural network inference," in *SysML Conference*, vol. 48, 2018.

[61] H. Gong, H. He, L. Pan, B. Gao, J. Tang, S. Pan, J. Li, P. Yao, D. Wu, H. Qian *et al.*, "An error-free 64kb reram-based nvsram integrated to a microcontroller unit supporting real-time program storage and restoration," *IEEE Transactions on Circuits and Systems*, 2023.

[62] M. Gorlatova, J. Sarik, G. Grebla, M. Cong, I. Kymissis, , and G. Zussman, "Movers and shakers: Kinetic energy harvesting for the internet of things," in *ACM international conference on Measurement and modeling of computer systems*, 2014.

[63] Y. Gu, Y. Liu, Y. Wang, H. Li, and H. Yang, "Nvpsim: A simulator for architecture explorations of nonvolatile processors," in *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2016.

[64] X. Guo, H. Zhu, Y. Jin, and X. Zhang, "When capacitors attack: Formal method driven design and detection of charge-domain trojans," in *Design, Automation & Test in Europe (DATE)*, 2019.

[65] Y. Guo, Y. Hua, and P. Zuo, "Dfpc: A dynamic frequent pattern compression scheme in nvm-based main memory," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2018.

[66] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *In Proceedings of the fourth annual IEEE international workshop on workload characterization*, 2001.

[67] F. T. Hady, A. Foong, B. Veal, and D. Williams, "Platform storage performance with 3d xpoint technology," *Proceedings of the IEEE*, vol. 105, no. 9, 2017.

[68] E. G. Hallnor and S. K. Reinhardt, "A unified compressed memory hierarchy," in *11th International Symposium on High-Performance Computer Architecture*, 2005.

[69] N. Hassan, B. Min, C. Jung, Y. Solihin, and J. Choi, "Warmcache: Exploiting stt-ram cache for low-power intermittent systems," in *52nd Annual International Symposium on Computer Architecture*, 2025.

[70] J. Hester, K. Storer, and J. Sorber, "Timely execution on intermittently powered batteryless sensors," in *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*, 2017.

[71] M. Hicks, "Clank: Architectural support for intermittent computation," in *ACM SIGARCH Computer Architecture News*, 2017.

[72] S. Hong, B. Abali, A. Buyuktosunoglu, M. B. Healy, and P. J. Nair, "Touché: Towards ideal and efficient cache compression by mitigating tag area overheads," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019.

[73] X. Hou, X. Tang, J. Liu, C. Li, L. Liang, and K.-T. Cheng, "Wasp: Efficient power management enabling workload-aware, self-powered aiot devices," *IEEE Transactions on Parallel and Distributed Systems*, vol. 35, no. 8, 2024.

[74] X. Hou, T. Xu, C. Li, C. Xu, J. Liu, Y. Hu, J. Zhao, J. Leng, K.-T. Cheng, and M. Guo, "A tale of two domains: Exploring efficient architecture design for truly autonomous things," in *51st Annual International Symposium on Computer Architecture (ISCA)*, 2024.

[75] Y. Huai *et al.*, "Spin-transfer torque mram (stt-mram): Challenges and prospects," *AAPPS bulletin*, vol. 18, no. 6, 2008.

[76] S.-Y. Huang, J. Zeng, X. Deng, S. Wang, A. Sifat, B. Bharmal, J.-B. Huang, R. Williams, H. Zeng, and C. Jung, "Rtailor: Parameterizing soft error resilience for mixed-criticality real-time systems," in *2023 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2023, pp. 344–357.

[77] A. Jadidi, M. Arjomand, M. T. Kandemir, and C. R. Das, "Hybridcomp: A criticality-aware compressed last-level cache," in *2018 19th International Symposium on Quality Electronic Design (ISQED)*, 2018.

[78] H. Jayakumar, A. Raha, and V. Raghunathan, "Quickrecall: A low overhead hw/sw approach for enabling computations across power cycles in transiently powered computers," in *In 2014 27th International Conference on VLSI Design and 2014 13th International Conference on Embedded Systems*, 2014.

[79] H. Jayakumar, A. Raha, and V. Raghunathan, "Quickrecall: A low overhead hw/sw approach for enabling computations across power cycles in transiently powered computers," in *In VLSI Design. IEEE Computer Society, 330–335*, 2014.

[80] J. Jeong, J. Hong, S. Maeng, C. Jung, and Y. Kwon, "Unbounded hardware transactional memory for a hybrid dram/nvm memory system," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020.

[81] J. Jeong and C. Jung, "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*, 2021.

[82] J. Jeong, J. Zeng, and C. Jung, "Capri: Compiler and architecture support for whole-system persistence," in *International Symposium on High-Performance Parallel and Distributed Computing*, 2022.

[83] C. Jung, D. Lim, J. Lee, and S. Han, "Adaptive execution techniques for smt multiprocessor architectures," in *The 10th ACM SIGPLAN symposium on Principles and practice of parallel programming*, 2005.

[84] P. Kamalinejad, C. Mahapatra, Z. Sheng, S. Mirabbasi, V. C. Leung, , and Y. L. Guan, "Wireless energy harvesting for the internet of things," in *IEEE Communications Magazine 53, 6 (2015)*, 2015.

[85] E. Kamenar, S. Zelenika, D. Blažević, S. Maćešić, G. Gregov, K. Marković, , and V. Glažar, "Harvesting of river flow energy for wireless sensor network technology," in *Microsystem Technologies 22, 7 (2016)*, vol. 22, 2016.

[86] R. Kanakagiri, B. Panda, and M. Mutyam, "Mbzip: Multiblock data compression," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 14, no. 4, 2017.

[87] S. Kaxiras, Z. Hu, and M. Martonosi, "Cache decay: exploiting generational behavior to reduce cache leakage power," in *Proceedings 28th Annual International Symposium on Computer Architecture*, 2001.

[88] B. Kellogg, V. Talla, S. Gollakota, and J. R. Smith, "Passive wi-fi: Bringing low power to wi-fi transmissions," in *NSDI*, 2016.

[89] H. Kim, J. Zeng, Q. Liu, M. Abdel-Majeed, J. Lee, and C. Jung, "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*, 2020, pp. 989–1004.

[90] J. Kim, S. Hong, J. Hong, and S. Kim, "Cid: Co-architecting instruction cache and decompression system for embedded systems," *IEEE Transactions on Computers*, vol. 70, no. 7, 2021.

[91] J. Kim, M. Sullivan, E. Choukse, and M. Erez, "Bit-plane compression: Transforming data for better compression in many-core architectures," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016.

[92] S. Kim, J. Kim, J. Lee, and S. Hong, "Residue cache: A low-energy low-area l2 cache architecture via compression and partial hits," in *International Symposium on Microarchitecture (MICRO)*, 2011.

[93] M. Kjelso, M. Gooch, and S. Jones, "Design and performance of a main memory hardware data compressor," in *Euromicro Conference. Beyond 2000: Hardware and Software Design Strategies*, 1996.

[94] M. Kjelsø, M. Gooch, and S. Jones, "Empirical study of memory-data: Characteristics and compressibility," *IEE Proceedings-Computers and Digital Techniques*, vol. 145, no. 1, 1998.

[95] H. Ko, H. Lee, T. Kim, and S. Pack, "Information freshness-guaranteed and energy-efficient data generation control system in energy harvesting internet of things," *IEEE Access*, vol. 8, 2020.

[96] V. Kortbeek, S. Ghosh, J. Hester, S. Campanoni, and P. Pawełczak, "Wario: efficient code generation for intermittent computing," in *Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, 2022.

[97] C. Lee, M. Potkonjak, , and W. H. Mangione-Smith, "Mediabench: A tool for evaluating and synthesizing multimedia and communications systems," in *In Proceedings of 30th Annual International Symposium on Microarchitecture*, 1997.

[98] H. G. Lee and N. Chang, "Powering the iot: Storage-less and converter-less energy harvesting," in *In Design Automation Conference (ASP-DAC), 2015 20th Asia and South Pacific. IEEE, 124–129*, 2015.

[99] J. Lee, J.-H. Park, H. Kim, C. Jung, D. Lim, and S. Han, "Adaptive execution techniques of parallel programs for multiprocessors," *Journal of Parallel and Distributed Computing*, vol. 70, no. 5, 2010.

[100] J.-S. Lee, W.-K. Hong, and S.-D. Kim, "Design and evaluation of a selective compressed memory system," in *International Conference on Computer Design: VLSI in Computers and Processors*, 1999.

[101] J.-S. Lee, W.-K. Hong, and S.-D. Kim, "An on-chip cache compression technique to reduce decompression overhead and design complexity," *Journal of systems Architecture*, vol. 46, no. 15, 2000.

[102] W. S. Lee, H. Jayakumar, and V. Raghunathan, "When they are not listening: Harvesting power from idle sensors in embedded systems," in *In 5th International Green Computing Conference*, 2014.

[103] S. Leng and A. Yener, "Learning to transmit fresh information in energy harvesting networks," *IEEE Transactions on Green Communications and Networking*, vol. 6, no. 4, 2022.

[104] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *42nd International Symposium on Microarchitecture (MICRO)*, 2009.

[105] Q. Liu, J. Izraelevitz, S. K. Lee, M. L. Scott, S. H. Noh, and C. Jung, "ido: Compiler-directed failure atomicity for nonvolatile memory," in *International Symposium on Microarchitecture (MICRO)*, 2018.

[106] Q. Liu and C. Jung, "Lightweight hardware support for transparent consistency-aware checkpointing in intermittent energy-harvesting systems," in *2016 5th Non-Volatile Memory Systems and Applications Symposium (NVMSA)*, 2016.

[107] Q. Liu, X. Wu, L. Kittinger, M. Levy, and C. Jung, "Benchprime: Effective building of a hybrid benchmark suite," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 16, 2017.

[108] Y. Liu, Z. Li, H. Li, Y. Wang, X. Li, K. Ma, S. Li, M.-F. Chang, S. John, Y. Xie, J. Shu, and H. Yang, "Ambient energy harvesting nonvolatile processors: From circuit to system," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2015.

[109] B. Lucia, V. Balaji, A. Colin, K. Maeng, and E. Ruppel, "Intermittent computing: Challenges and opportunities," in *In LIPIcs-Leibniz International Proceedings in Informatics, Vol. 71. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik*, 2017.

[110] G. Lukosevicius, A. R. Arreola, and A. S. Weddell, "Using sleep states to maximize the active time of transient computing systems," in *Proceedings of the Fifth ACM International Workshop on Energy Harvesting and Energy-Neutral Sensing Systems*, 2017.

[111] K. Ma, X. Li, S. Li, Y. Liu, J. J. Sampson, Y. Xie, and V. Narayanan, "Nonvolatile processor architecture exploration for energy-harvesting applications," *IEEE Micro*, vol. 35, no. 5, 2015.

[112] K. Ma, Y. Zheng, S. Li, K. Swaminathan, X. Li, Y. Liu, J. Sampson, Y. Xie, and V. Narayanan, "Architecture exploration for ambient energy harvesting nonvolatile processors," in *In IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, 2015.

[113] K. Maeng and B. Lucia, "Supporting peripherals in intermittent systems with just-in-time checkpoints," in *ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2019.

[114] M. Magno and D. Boyle, "Wearable energy harvesting: From body to battery," in *In 2017 12th International Conference on Design and Technology of Integrated Systems In Nanoscale Era (DTIS)*, 2017.

[115] M. Magno, D. Kneubuhler, P. Mayer, and L. Benini, "Micro kinetic energy harvesting for autonomous wearable devices," in *In 2018 International symposium on power electronics, electrical drives, automation and motion (SPEEDAM)*, 2018.

[116] M. Minkin and B. Kasikci, "Zipchannel: Cache side-channel vulnerabilities in compression algorithms," in *2024 54th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2024.

[117] S. Mittal, J. S. Vetter, and L. Jiang, "Addressing read-disturbance issue in stt-ram by data compression and selective duplication," *IEEE Computer Architecture Letters*, vol. 16, no. 2, 2016.

[118] S. Mohapatra, V. Kortbeek, M. A. van Eerden, J. Broekhoff, S. Ahmed, and P. Pawełczak, "Data cache for intermittent computing systems with non-volatile main memory," in *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2025.

[119] T. M. Nguyen and D. Wentzlaff, "Morc: A manycore-oriented compressed cache," in *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2015.

[120] K. Oka, S. Kawakami, T. Tanimoto, T. Ono, and K. Inoue, "Enhancing a manycore-oriented compressed cache for gpgpu," in *Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region*, 2020.

[121] O. Ozturk, M. Kandemir, M. J. Irwin, and S. Tosun, "On-chip memory management for embedded mpsoc architectures based on data compression," in *Proceedings IEEE International SOC Conference*, 2005.

[122] O. Ozturk, G. Chen, M. Kandemir, and I. Kolcu, "Compiler-guided data compression for reducing memory consumption of embedded applications," in *Proceedings of the 2006 Asia and South Pacific Design Automation Conference*, 2006.

[123] O. Ozturk, M. Kandemir, and M. J. Irwin, "Increasing on-chip memory space utilization for embedded chip multiprocessors through data compression," in *Proceedings of the 3rd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, 2005.

[124] O. Ozturk, H. Saputra, M. Kandemir, and I. Kolcu, "Access pattern-based code compression for memory-constrained embedded systems," in *Design, Automation and Test in Europe*, 2005.

[125] Z. Pan, F. Zhang, Y. Zhou, J. Zhai, X. Shen, O. Mutlu, and X. Du, "Exploring data analytics without decompression on embedded gpu systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 7, 2021.

[126] B. Panda and A. Seznec, "Synergistic cache layout for reuse and compression," in *Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques*, 2018.

[127] B. Panda and A. Seznec, "Dictionary sharing: An efficient cache compression scheme for compressed caches," in *49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2016.

[128] G. Park, T. Rosing, M. D. Todd, C. R. Farrar, and W. Hodgkiss, "Energy harvesting for structural health monitoring sensor networks," in *Journal of Infrastructure Systems 14, 1 (2008)*, vol. 14, 2008.

[129] J. Park, S. Baek, H. G. Lee, C. Nicopoulos, V. Young, J. Lee, and J. Kim, "Hope: Hot-cacheline prediction for dynamic early decompression in compressed llcs," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 22, no. 3, 2017.

[130] G. Pekhimenko, T. Huberty, R. Cai, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, "Exploiting compressed block size as an indicator of future reuse," in *2015 IEEE 21st international symposium on high performance computer architecture (HPCA)*, 2015.

[131] G. Pekhimenko, V. Seshadri, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, "Base-delta-immediate compression: practical data compression for on-chip caches," in *International Conference on Parallel Architectures and Compilation Techniques*, 2012.

[132] S. Pelley, P. M. Chen, and T. F. Wenisch, "Memory persistency," *ACM SIGARCH Computer Architecture News*, vol. 42, no. 3, 2014.

[133] S. Priya and D. J. Inman, "Energy harvesting technologies," 2009.

[134] N. Rajesh, H. Devarajan, J. C. Garcia, K. Bateman, L. Logan, J. Ye, A. Kougkas, and X.-H. Sun, "Apollo: An ml-assisted real-time storage resource observer," in *Proceedings of the 30th International Symposium on High-Performance Parallel and Distributed Computing*, 2021.

[135] B. Ransford, J. Sorber, and K. Fu, "Mementos: System support for long-running computation on rfid-scale devices," *SIGARCH Comput. Archit. News*, vol. 39, no. 1, mar 2011.

[136] J. Ren, J. Zhao, S. Khan, J. Choi, Y. Wu, and O. Mutlu, "Thynvm: Enabling software-transparent crash consistency in persistent memory systems," in *Proceedings of the 48th International Symposium on Microarchitecture*, 2015.

[137] Q. Ren, T.-T. Chan, H. Pan, K.-H. Ho, and Z. Du, "Information freshness and energy harvesting tradeoff in network-coded broadcasting," *IEEE Wireless Communications Letters*, vol. 11, no. 10, 2022.

[138] L. Rizzon, M. Rossi, R. Passerone, and D. Brunelli, "Wireless sensor networks for environmental monitoring powered by microprocessors heat dissipation," in *In Proceedings of the 1st International Workshop on Energy Neutral Sensing Systems (ENSSys)*, 2013.

[139] D. Rodrigues Carvalho and A. Seznec, "A case for partial co-allocation constraints in compressed caches," in *International Conference on Embedded Computer Systems*, 2021.

[140] A. Sabovic, M. Aernouts, D. Subotic, J. Fontaine, E. De Poorter, and J. Famaey, "Towards energy-aware tinyml on battery-less iot devices," *Internet of Things*, vol. 22, 2023.

[141] S. Sardashti, A. Seznec, and D. A. Wood, "Skewed compressed caches," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, 2014.

[142] S. Sardashti, A. Seznec, and D. A. Wood, "Yet another compressed cache: A low-cost yet effective compressed cache," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 13, no. 3, 2016.

[143] J. A. Storer and T. G. Szymanski, "Data compression via textual substitution," *J. ACM*, vol. 29, no. 4, Oct. 1982.

[144] F. Su, Y. Liu, Y. Wang, and H. Yang, "A ferroelectric nonvolatile processor with 46mus system-level wake-up time and 14mus sleep time for energy harvesting applications," in *IEEE Transactions on Circuits and Systems*, 2016.

[145] F. Su, K. Ma, X. Li, T. Wu, Y. Liu, and V. Narayanan, "Nonvolatile processors: Why is it trending?" in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, 2017.

[146] W. Sun, T. Tan, Z. Yan, D. Zhao, X. Luo, , and W. Huang, "Energy harvesting from water flow in open channel with macro fiber composite," in *AIP Advances 8, 9 (2018)*, vol. 8, 2018.

[147] M. Surbatovich, L. Jia, and B. Lucia, "Automatically enforcing fresh and consistent inputs in intermittent systems," in *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, 2021.

[148] M. Takagi and K. Hiraki, "Compression in data caches with compressible field isolation for recursive data structures," in *Euro-Par 2003 Parallel Processing: 9th International Euro-Par Conference Klagenfurt, Austria, August 26-29, 2003 Proceedings 9*, 2003.

[149] M. Takagi and K. Hiraki, "Field array compression in data caches for dynamically allocated recursive data structures," in *International Symposium on High Performance Computing*, 2003.

[150] X. Tang, Z. Zhang, W. Xu, M. T. Kandemir, R. Melhem, and J. Yang, "Enhancing address translations in throughput processors via compression," in *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques*, 2020.

[151] D. Tarjan, S. Thoziyoor, and N. Jouppi, "Cacti 4.0," 2006.

[152] M. Thuresson, L. Spracklen, and P. Stenstrom, "Memory-link compression schemes: A value locality perspective," *IEEE Transactions on Computers*, vol. 57, no. 7, 2008.

[153] R. B. Tremaine, P. A. Franaszek, J. T. Robinson, C. O. Schulz, T. B. Smith, M. E. Wazlowski, and P. M. Bland, "Ibm memory expansion technology," *IBM Journal of Research and Development*, 2001.

[154] R. Tremaine, T. Smith, M. Wazlowski, D. Har, K.-K. Mak, and S. Arramreddy, "Pinnacle: Ibm mxt in a memory controller chip," *IEEE Micro*, vol. 21, no. 2, 2001.

[155] P.-A. Tsai, A. Sanchez, C. W. Fletcher, and D. Sanchez, "Safecracker: Leaking secrets through compressed caches," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020.

[156] P.-A. Tsai and D. Sanchez, "Compress objects, not cache lines: An object-based compressed memory hierarchy," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019.

[157] V. V. Tyagi and D. Buddhi, "Pcm thermal storage in buildings: A state of art," *Renewable and sustainable energy reviews*, vol. 11, no. 6, 2007.

[158] J. Van Der Woude and M. Hicks, "Intermittent computation without hardware support or programmer intervention," in *USENIX Conference on Operating Systems Design and Implementation*, 2016.

[159] N. Vijaykumar, G. Pekhimenko, A. Jog, A. Bhowmick, R. Ausavarungnirun, C. Das, M. Kandemir, T. C. Mowry, and O. Mutlu, "A case for core-assisted bottleneck acceleration in gpus: enabling flexible data compression with assist warps," *ACM SIGARCH Computer Architecture News*, vol. 43, no. 3S, 2015.

[160] L. Villa, M. Zhang, and K. Asanovic, "Dynamic zero compression for cache energy reduction," in *Proceedings 33rd Annual IEEE/ACM International Symposium on Microarchitecture. MICRO-33 2000*, 2000.

[161] C. Wang, N. Chang, Y. Kim, S. Park, Y. Liu, H. G. Lee, R. Luo, and H. Yang, "Storage-less and converterless maximum power point tracking of photovoltaic cells for a nonvolatile microprocessor," in *In Design Automation Conference (ASP-DAC)*, 2014.

[162] Welch, "A technique for high-performance data compression," *Computer*, vol. 17, no. 6, 1984.

[163] P. R. Wilson, S. F. Kaplan, and Y. Smaragdakis, "The case for compressed caching in virtual memory systems." in *USENIX Annual Technical Conference, General Track*, 1999.

[164] H. Wu, K. Nathella, M. Pabst, D. Sunwoo, A. Jain, and C. Lin, "Practical temporal prefetching with compressed on-chip metadata," *IEEE Transactions on Computers*, vol. 71, no. 11, 2021.

[165] T. Wu, K. Ma, J. Hu, J. Xue, J. Li, X. Shi, H. Yang, and Y. Liu, "Reliable and efficient parallel checkpointing framework for nonvolatile processor with concurrent peripherals," *IEEE Transactions on Circuits and Systems*, vol. 70, no. 1, 2023.

[166] Y. Wu, B. Min, M. Ismail, W. Xiong, C. Jung, and D. Lee, "{IntOS}: Persistent embedded operating system and language support for multi-threaded intermittent computing," in *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, 2024.

[167] Y. Xie and G. H. Loh, "Thread-aware dynamic shared cache compression in multi-core processors," in *2011 IEEE 29th International Conference on Computer Design (ICCD)*, 2011.

[168] J. Xu, D. Feng, Y. Hua, W. Tong, J. Liu, and C. Li, "Extending the lifetime of nvms with compression," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2018.

[169] J. Xu, D. Feng, Y. Hua, W. Tong, J. Liu, C. Li, and W. Zhou, "Improving performance of tlc rram with compression-ratio-aware data encoding," in *International Conference on Computer Design*, 2017.

[170] Y. Xu, J. Izraelevitz, and S. Swanson, "Clobber-nvm: log less, re-execute more," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2021.

[171] J. Yang, Y. Zhang, and R. Gupta, "Frequent value compression in data caches," in *Proceedings 33rd Annual IEEE/ACM International Symposium on Microarchitecture. MICRO-33 2000*, 2000.

[172] Y. Yang, J. S. Emer, and D. Sanchez, "Spzip: Architectural support for effective data compression in irregular applications," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021.

[173] C.-W. Yau, T. T.-O. Kwok, C.-U. Lei, , and Y.-K. Kwok, "Energy harvesting in internet of things. in internet of everything," in *IEEE Communications Magazine 53, 6 (2015)*, 2018.

[174] C.-H. Yen, H. R. Mendis, T.-W. Kuo, and P.-C. Hsiu, "Catch non-determinism if you can: Intermittent inference of dynamic neural networks," *ACM Transactions on Embedded Computing Systems*, 2025.

[175] V. Young, P. J. Nair, and M. K. Qureshi, "Dice: Compressing dram caches for bandwidth and capacity," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, 2017.

[176] J. Zeng, "Compiler and architecture co-design for reliable computing," Ph.D. dissertation, Purdue University, 2024.

[177] J. Zeng, J. Choi, X. Fu, A. P. Shreepathi, D. Lee, C. Min, and C. Jung, "Replaycache: Enabling volatile cachesfor energy harvesting systems," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021.

[178] J. Zeng, J. Jeong, and C. Jung, "Persistent processor architecture," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, 2023.

[179] J. Zeng, H. Kim, J. Lee, and C. Jung, "Turnpike: Lightweight soft error resilience for in-order cores," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 654–666.

[180] J. Zeng, S. Pei, D. Zhang, Y. Zhou, A. Beygi, X. Yao, R. Kachare, T. Zhang, Z. Li, M. Nguyen *et al.*, "Performance characterizations and usage guidelines of samsung cmm-h," *IEEE Micro*, 2025.

[181] J. Zeng, S. Pei, D. Zhang, Y. Zhou, A. Beygi, X. Yao, R. Kachare, T. Zhang, Z. Li, M. Nguyen *et al.*, "Performance characterizations and usage guidelines of samsung cxl memory module hybrid prototype," *arXiv preprint*, 2025.

[182] J. Zeng, T. Zhang, and C. Jung, "Compiler-directed whole-system persistence," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, 2024.

[183] Y. Zhang, D. Feng, Y. Hua, Y. Hu, W. Xia, M. Fu, X. Tang, Z. Wang, F. Huang, and Y. Zhou, "Reducing chunk fragmentation for in-line delta compressed and deduplicated backup systems," in *2017 International Conference on Networking, Architecture, and Storage (NAS)*, 2017.

[184] Y. Zhou, J. Zeng, J. Jeong, J. Choi, and C. Jung, "Sweepcache: Intermittence-aware cache on the cheap," in *MICRO-56: 56th Annual IEEE/ACM International Symposium on Microarchitecture*, 2023.

[185] Y. Zhou, J. Zeng, and C. Jung, "Lightwsp: Whole-system persistence on the cheap," in *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2024.

[186] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Transactions on Information Theory*, 1977.

[187] J. Ziv and A. Lempel, "Compression of individual sequences via variable-rate coding," *IEEE Transactions on Information Theory*, 1978.