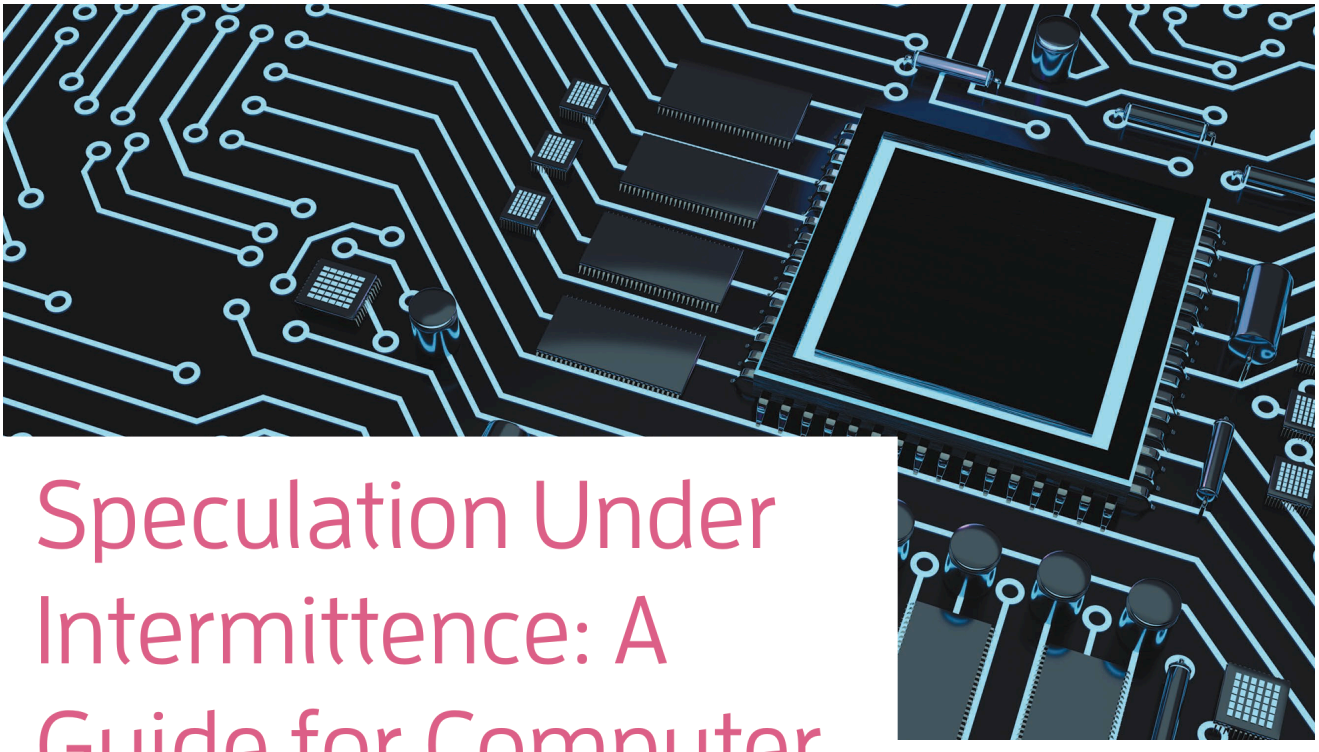
EDITORS
TIMOTHY JONES University of Cambridge, U.K., timothy.jones@cl.cam.ac.uk
ROBERT MULLINS University of Cambridge, U.K., robert.mullins@cl.cam.ac.uk



Speculation Under Intermittence: A Guide for Computer Architects

Gan Fang and **Changhee Jung**, Purdue University

This article discusses the challenges posed by frequent power interruptions in intermittently powered systems and how computer architects can tailor microarchitectural techniques for these batteryless systems to enhance their overall energy efficiency.

Intermittently powered systems (IPSs) are gaining significant traction across diverse applications—such as the Internet of Batteryless Things, infrastructure monitoring, health tracking, and wearable devices—owing to their environmentally friendly, self-sustaining, maintenance-free nature. IPSs harvest energy from ambient sources, such as radio-frequency (RF) signals, solar radiation, and thermal gradients. However, because such energy sources are weak and unstable, the harvested energy is first accumulated in a small capacitor. Consequently, IPSs (re)boot only when sufficient energy is available, shut down when the capacitor is depleted, and remain inactive while recharging. This sequence repeats throughout their lifetime, making computation inherently intermittent.[1] To ensure correct execution despite frequent power interruptions, IPSs employ nonvolatile memory (NVM) as main memory, together with checkpoint-and-recovery mechanisms that preserve system state across power failure.

Among these mechanisms, nonvolatile processor (NVP)[2] is the most prevalent, utilizing just-in-time (JIT) checkpointing and restoration. NVP preserves volatile state right before each power interruption, by saving registers (REGs) to nonvolatile flip-flops (NVFFs) and flushing store buffer (SB) entries and dirty (that is, updated) cache blocks to their nonvolatile

Digital Object Identifier 10.1109/MC.2026.3696474
Date of current version: 28 July 2026

counterparts—such as nonvolatile SRAM (NVSRAM) in Figure 1. As soon as power returns after recharging, NVP restores the saved state and then resumes program execution from the prior power interruption point. In particular, the interval from reboot to the next power interruption is referred to as a *power cycle* (that is, the on-time shown on the left of Figure 1). The duration of a power cycle is determined by multiple factors, including harvested energy quality, workload behavior, and capacitor size, for example, typically ranging from hundreds of microseconds to tens of milliseconds in RF-harvesting IPSs.

CHALLENGE

A fundamental challenge in IPSs is that volatile data are lost upon each power interruption. This behavior restricts the opportunity for data reuse and undermines the effectiveness of on-chip storage structures, such as caches. Conventional cache designs are predicated on the assumption of temporal locality, that is, the expectation that recently accessed data are likely to be reused in the near future, allowing the processor to retrieve it from the cache—rather than from slower and more energy-intensive NVM. However, this assumption is often invalidated in IPSs when power failure is impending, though caches would otherwise capture temporal reuse and reduce expensive NVM accesses. Specifically, frequent power interruptions can cause cached data to be lost before it can be reused, diminishing the effectiveness of conventional cache designs. For example, NVP[2] exhibits a four-times higher cache miss rate compared to execution with no power failure, as cached data are repeatedly lost across power cycles.

Figure 2 depicts this phenomenon. In this example, the cache is initially empty. A read request for block *A* at T_1 thus causes a cache miss (❶) and fetches the block from NVM (❷) into the cache. Similarly, requesting block *B* at T_2 results in another cache miss (❸) and NVM fetch (❹). Under normal operation, both blocks would remain in the cache and be reused there at T_3 and T_4. However, a power interruption before T_3 causes both blocks to be lost—volatile storage cannot retain data in the absence of power. As a result, the expected reuse of these blocks does not occur. When block *A* is requested again at T_3, the data are no longer available in the cache (❺), so it must be refetched from NVM (❻). The same situation repeats for block *B* at T_4. The takeaway is that power interruptions can eliminate opportunities to reuse cached data, forcing the IPS to resort to expensive NVM accesses upon subsequent cache misses and thereby reducing the overall effectiveness of the cache in IPSs.

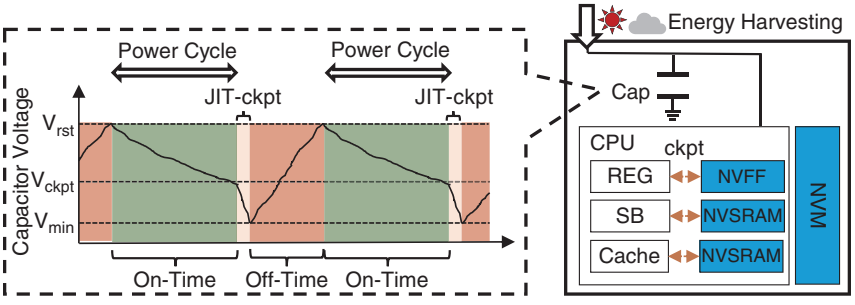


FIGURE 1. Design of NVP.

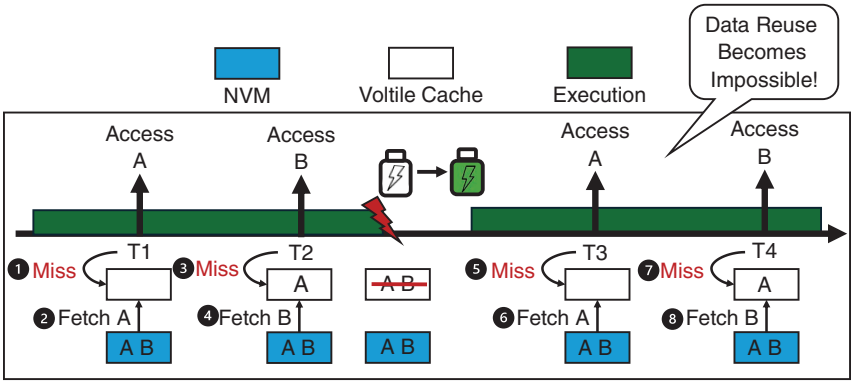


FIGURE 2. Loss of cache block reuse caused by power interruptions.

Challenge on prefetching

Power interruptions also reduce the effectiveness of cache-oriented speculation techniques. A representative example is hardware prefetching that can improve performance by proactively loading data into the cache before it is requested by the processor, overlapping memory access with the execution of other instructions. Under continuous power supply, this increases the likelihood that when the request is issued, the corresponding block is already available in the cache or will arrive sooner—effectively hiding the memory latency. In IPSs, however, prefetched blocks may be lost before they are ever used, for example, the gain of a stride prefetcher, that predicts future access patterns by identifying fixed-distance (stride) between memory requests, is reduced to only one-third of its potential compared to execution with no power failure. Specifically, if a cache block is prefetched but not accessed before a power interruption, the time and energy spent transferring the block from NVM into the cache yield no benefit to subsequent program execution and can instead degrade the overall performance and energy efficiency of IPSs.

Figure 3 illustrates this problem. At T_1, the prefetcher predicts that cache blocks *A* and *B* would soon be needed and proactively loads them into the

cache (❶) to accelerate future accesses. When the processor later requests block A at T_2, the data are already available in the cache and can be retrieved without delay (❷). However, before block B is accessed, a power interruption occurs and clears all data stored in the volatile cache, including the prefetched block. After execution resumes, when the processor requests block B, the data are no longer present in the cache, triggering a cache miss (❸). The processor must therefore fetch the block again from memory at T_3 (❹). This example highlights a key limitation of conventional prefetchers in IPSs. Because they operate without awareness of upcoming power interruptions, they cannot determine whether prefetched data would actually be used. As a result, many prefetch operations trigger memory transfers that consume scarce energy but fail to reduce future memory accesses, limiting both performance improvement and energy efficiency.

In general, IPSs thus opt for a simple prefetcher, such as a stride one—rather than sophisticated designs that are more aggressive yet power-hungry, possibly offsetting their potential benefits under intermittent execution. Moreover, since most IPS platforms rely on low-power in-order processors, the highly aggressive prefetching strategies designed for superscalar out-of-order processors are often ill-suited for intermittent execution.

Challenge on dead block prediction

Dead block prediction is another example of a technique whose behavior changes in IPSs. A dead block predictor aims to identify cache blocks that are unlikely to be reused so that they can be safely deactivated, thereby reducing leakage energy (the power consumed to maintain unused data stored in the cache).

Figure 4 shows how a dead block is identified under continuous power supply. At T_1, the processor requests block A, which is not yet present in the cache, triggering a cache miss (❶) and fetching the block from memory into the cache (❷). Then, block A is accessed again at T_2, that is, it remains useful between T_1 and T_2. However, after this access, block A is no longer used before it is eventually evicted from the cache at T_3. Therefore, the block is considered "dead" during the interval from T_2 to T_3 because keeping it in the cache does not contribute to future cache hits. Dead block predictors exploit this behavior by deactivating such dead blocks in advance, thereby reducing leakage energy without affecting cache hit rates. One representative technique is cache decay,[3] which classifies a block as dead if it remains unused for a predefined period after its last access.

However, when conventional dead block predictors are applied to IPSs, their effectiveness is diminished because power interruptions introduce an additional mechanism by which cache contents can be cleared. In IPSs, a cache block may be predicted to remain useful in the near future yet still be erased by a power interruption before it can be accessed again. We refer to such blocks as *zombie blocks*. Deactivating these blocks earlier can further reduce cache leakage energy without affecting performance.

Figure 5 illustrates this situation. At T_1, block A is requested, triggering a cache miss (❶) and the fetching of the block from NVM (❷). The dead block predictor accurately predicts that this block is likely to be reused and thus keeps it active in the cache. However, a power interruption occurs before the next access, causing the cached data to be lost. As a result, block A remains stored in the cache from T_1 until the interruption without contributing to any cache hits. During this interval, the energy spent maintaining the block in the cache provides no benefit to program execution. Such blocks are therefore identified as zombie blocks.

INTERMITTENCE AWARENESS

As previously noted, conventional speculation techniques underperform in IPSs

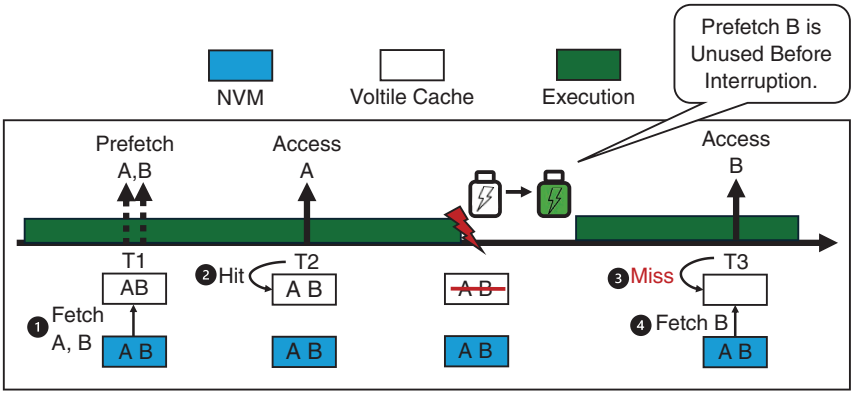


FIGURE 3. The inefficiency of existing prefetchers under power interruptions.

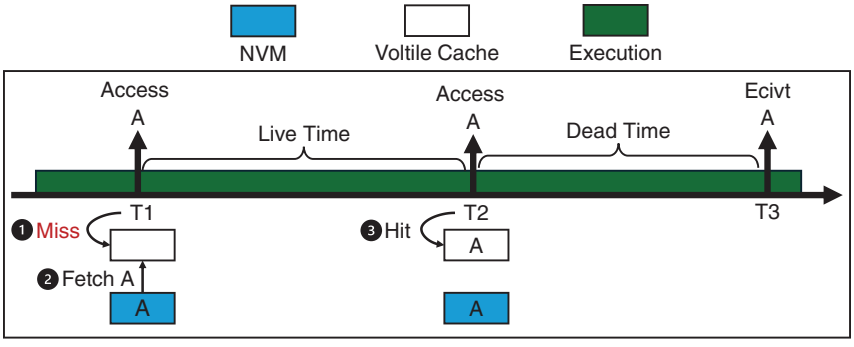


FIGURE 4. Dead block definition.

because they assume that cached data remains available until explicit eviction. In reality, frequent power interruptions often erase volatile data before it can be utilized for program execution. To overcome this, computer architects are incorporating *intermittence awareness* into existing microarchitectural techniques, enabling them to adapt their behavior based on the likelihood of upcoming interruptions—thereby improving both energy efficiency and performance.

Intermittence-aware prefetching

One example of this approach is intermittence-aware prefetching extension (IPEX),[4] which enhances conventional hardware prefetchers by reducing unnecessary data movement when a power interruption is likely. To achieve this, IPEX dynamically adjusts the aggressiveness of prefetching using a bimodal control strategy with two operating modes: *energy-saving mode* and *high-performance mode*. When the system detects that a power interruption is likely to occur soon by monitoring capacitor voltage (as shown in Figure 1), IPEX enters *energy-saving mode* and throttles the number of prefetch requests. This prevents the cache from being filled with data that is unlikely to be used before the interruption. When the risk of interruption decreases, IPEX returns to *high-performance mode* and restores the normal prefetching level to regain performance. By adapting prefetch activity to current energy conditions, IPEX helps ensure that data movement into the cache effectively contributes to useful execution.

Figure 6 shows how IPEX adjusts prefetching behavior in response to an upcoming power interruption. At T_1, IPEX detects an impending interruption and therefore switches to an *energy-saving mode*. In this mode, IPEX reduces its prefetch degree—the number of blocks prefetched in advance—from its normal value of two blocks to one. As a result, only block *A* is prefetched (❶), which is subsequently accessed from the cache at T_2 (❷); by throttling its aggressiveness, IPEX avoids prefetching additional blocks that would likely be lost before use. Once power is restored, IPEX returns to its *high-performance mode*, restoring the prefetch degree to two. It then prefetches blocks *B* and *C* (❸), allowing subsequent accesses to these blocks at T_4 (❹) and T_5 (❺) to be served directly from the cache. By adapting its prefetching activity, IPEX reduces unnecessary data movement proximate to power interruptions, while preserving the performance benefits of prefetching when energy is sufficient.

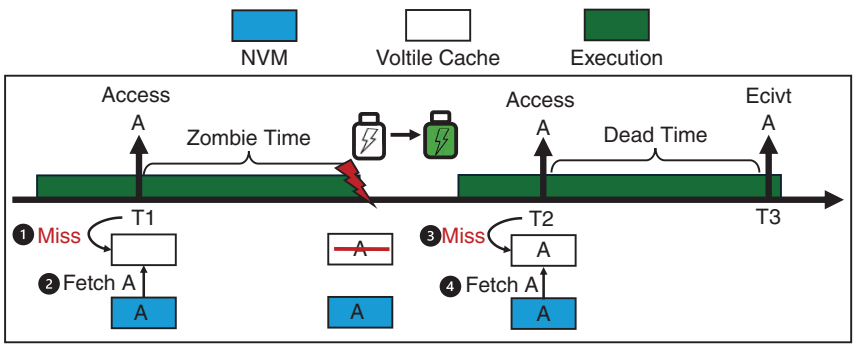


FIGURE 5. Zombie block definition in IPSs.

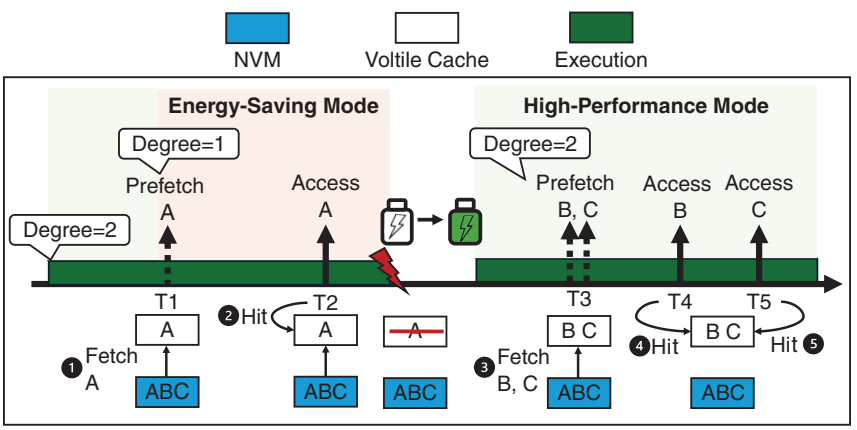


FIGURE 6. Adaptive prefetching in IPEX.

Intermittence-aware dead block prediction

A similar principle can be applied to dead block prediction. Extending dead block predictors (EDBP)[5] extends conventional dead block predictors by identifying and deactivating *zombie blocks*. Under stable energy conditions, when a power interruption is unlikely, the baseline dead block predictor continues to operate normally. As the likelihood of an interruption increases, EDBP gradually intervenes to proactively deactivate blocks improbable to contribute to future cache hits. In this way, EDBP complements existing prediction mechanisms to reduce cache leakage energy, improving the overall energy efficiency of IPSs. To operate effectively, EDBP must determine when active blocks would become zombie blocks. It relies on the observation that as a power interruption approaches, a larger portion of cached data are unlikely to be reused. This allows EDBP to anticipate which blocks are at risk of becoming useless and deactivate them earlier to save energy, thereby extending the power cycle.

As illustrated in Figure 7, EDBP leverages the cache replacement recency rank as a practical indicator of reuse likelihood, targeting blocks near the least-recently-used (LRU) position for early deactivation when an interruption is expected. More specifically, when a

potential interruption is first detected, EDBP follows a conservative strategy and deactivates only the LRU block—as the temporal distance to the interruption is still far. As the system moves closer to the interruption, EDBP progressively expands the set of blocks considered unlikely to be reused and deactivates them more aggressively, increasing the potential for energy savings without affecting performance.

One potential concern is that by reducing leakage energy and extending the power cycle, EDBP could delay the next power interruption long enough to reach the point where deactivated zombie blocks would have been accessed. In such cases, prematurely deactivating these mispredicted zombie blocks can introduce additional cache misses. To address this issue, EDBP adopts a conservative deactivation strategy that primarily targets cache blocks with a very low likelihood of reuse before the next interruption. As a result, the overall energy savings generally outweigh the occasional performance penalty caused by such mispredictions due to the extended power cycle.

LOOKING FORWARD

Reframing speculation for IPSs

IPSs challenge one of the long-standing assumptions in computer architecture: that speculation operates under continuous execution. Traditionally, speculative techniques, such as prefetching, improve performance by staging data in anticipation of future program needs. These techniques assume that once speculative data are brought into the cache, it remains available there long enough to be used by the processor, unless it is later replaced by other data during normal operation. In IPSs, however, frequent power interruptions may occur before cache block eviction and erase speculative results before they yield any performance gain. Consequently, speculative actions that provide benefits in continuously powered systems can instead waste energy in IPSs. This observation underscores a critical design principle for future IPS architectures: speculative mechanisms should be guided not only by program behavior, but also by the system's current energy conditions and the likelihood of upcoming interruptions.

Beyond speculation

Although this article has focused on speculation techniques, such as prefetching and dead block prediction, the principle of intermittence awareness applies more broadly across many other microarchitectural optimizations. For example, cache compression increases effective cache capacity by storing data in a compressed form, allowing more information to fit within the same physical space. This approach is beneficial when the compressed data remains available long enough to be reused. In IPSs, however, power interruptions may erase cached data before it can be accessed again, nullifying the benefits of compression. In such cases, the energy and latency overheads of compression and decompression provide little to no performance gain, undermining the overall efficiency of the technique. Recent work[6] addresses this limitation by adapting compression decisions to runtime energy conditions, compressing cache blocks only when they are likely to remain useful before the next interruption. This example highlights that intermittence awareness can serve as a general design principle for improving the efficiency of microarchitectural optimizations in IPSs.

Beyond single-core in-order processors

Most existing intermittence-aware architectural techniques[4,5,6,7,8,9,10] have been developed for single-core in-order processors—which represent the dominant baseline platform for today's IPSs.[11,12,13,14,15] However, future IPS deployments are expected to incorporate increasingly complex processing architectures. Out-of-order processors, for instance, could leverage intermittence awareness to dynamically modulate instruction scheduling, speculation depth, and reorder-buffer management. Beyond single-core designs, multicore systems can exploit inter-core coordination to improve energy efficiency. For example, cores may share information about energy availability and task progress to decide which threads should execute first or which data should be retained in shared caches. By aligning scheduling and data management decisions with the likelihood of interruption, the system can increase the amount of useful work completed within each power cycle. Likewise, GPUs and artificial intelligence accelerators—which rely heavily

IPSs challenge one of the long-standing assumptions in computer architecture: that speculation operates under continuous execution.

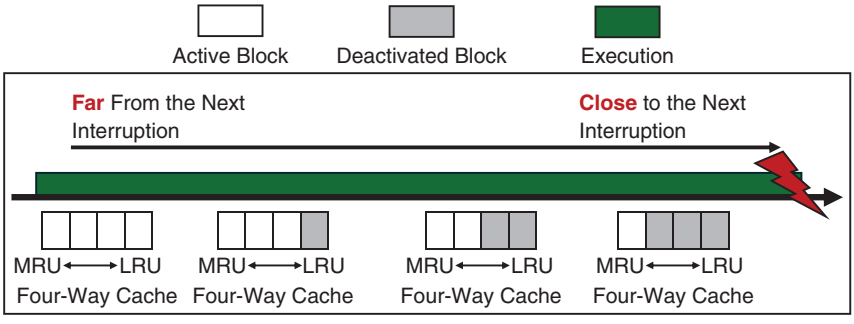


FIGURE 7. Zombie blocks detection and deactivation in EDBP.

on aggressive parallel execution and memory speculation—stand to benefit from interruption-aware scheduling and data orchestration policies. Extending intermittence awareness to these platforms represents an important direction for scaling IPS performance beyond current microcontroller-class systems.

A roadmap for practical adoption

Notably, incorporating intermittence awareness does not require redesigning processors from scratch. In many cases, lightweight monitoring of run-time energy conditions—combined with simple control logic—can adapt existing mechanisms dynamically with modest hardware overhead. Because these approaches reuse existing architectural structures, they are practical not only for research prototypes but also offer a viable path toward enhancing commercial embedded processors, microcontrollers, and emerging edge platforms that operate under strict energy constraints. This shift in perspective demands more than just incremental patches; it requires a fundamental rethinking of how architectures manage resources under power interruptions. By embedding intermittence awareness into the very fabric of the processor, we can move beyond the constraints of stable power and unlock the full potential of self-sustaining IPSs.

IPSs challenge many conventional architectural assumptions due to frequent power interruptions and the resulting loss of volatile state. Consequently, microarchitectural techniques developed for continuously powered systems often become less effective under intermittent execution. This article highlighted how intermittence affects speculative mechanisms and reviewed recent intermittence-aware solutions. Looking ahead, incorporating awareness of power interruptions into architectural design will be essential for improving the efficiency and capability of future batteryless computing systems.

REFERENCES

1. B. Lucia, V. Balaji, A. Colin, K. Maeng, and E. Ruppel, "Intermittent computing: Challenges and opportunities," in *LIPIcs-Leibniz Int. Proc. Inform.*, vol. 71, Wadern, Germany: Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.
2. K. Ma et al., "Architecture exploration for ambient energy harvesting nonvolatile processors," in *Proc. IEEE 21st Int. Symp. High Perform. Comput. Archit. (HPCA)*, 2015, pp. 526–537, doi: 10.1109/HPCA.2015.7056060.
3. S. Kaxiras, Z. Hu, and M. Martonosi, "Cache decay: Exploiting generational behavior to reduce cache leakage power," *in Proc. 28th Annu. Int. Symp. Comput. Archit. (ISCA)*, 2001, pp. 240–251, doi: 10.1109/ISCA.2001.937453.
4. G. Fang, J. Zeng, A. Gupta, and C. Jung, "Rethinking prefetching for intermittent computing," in *Proc. 52nd Annu. Int. Symp. Comput. Archit. (ISCA)*, 2025. pp, 225–240, doi: 10.1145/3695053.3731038.
5. G. Fang and C. Jung, "Rethinking dead block prediction for intermittent computing," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, 2025, pp. 732–744, doi: 10.1109/HPCA61900.2025.00061.
6. G. Fang, J. Zeng, Y. Zhou, and C. Jung, "Intermittence-Aware cache compression," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, 2026, pp. 1–17, doi: 10.1109/HPCA68181.2026.11408535.
7. G. Fang, J. Choi, and C. Jung, "Hybrid power failure recovery for intermittent computing," in *Proc. ACM/IEEE Int. Conf. Comput. Aided Des. (ICCAD)*, 2024, pp. 1–9, doi: 10.1145/3676536.3676654.
8. Y. Zhou, J. Zeng, J. Jeong, J. Choi, and C. Jung, "SweepCache: Intermittence-aware cache on the cheap," in *Proc. 56th Annu. IEEE/ACM Int. Symp. Microarchit. (MICRO)*, 2023, pp. 1059–1074, doi: 10.1145/3613424.3623781.
9. J. Choi, J. Zeng, D. Lee, C. Min, and C. Jung, "Write-light cache for energy harvesting systems," in *Proc. 50th Annu. Int. Symp. Comput. Archit. (ISCA)*, 2023, pp. 1–13, doi: 10.1145/3579371.3589098.
10. N. Hassan, B. Min, C. Jung, Y. Solihin, and J. Choi, "WarmCache: Exploiting STT-RAM cache for low-power intermittent systems," in *Proc. 52nd Annu. Int. Symp. Comput. Archit. (ISCA)*, 2025, pp. 586–600, doi: 10.1145/3695053.3731049.
11. J. Zeng et al., "ReplayCache: Enabling volatile caches for energy harvesting systems," in *Proc. 54th Annu. IEEE/ACM Int. Symp. Microarchit. (MICRO)*, 2021, pp. 170–182, doi: 10.1145/3466752.3480102.
12. J. Choi, Q. Liu, and C. Jung, "CoSpec: Compiler directed speculative intermittent computation," in *Proc. 52nd Annu. IEEE/ACM Int. Symp. Microarchit. (MICRO)*, 2019, pp. 399–412, doi: 10.1145/3352460.3358279.
13. J. Choi, H. Joe, C. Jung, and J. Choi, "Defending against EMI attacks on just-in-time checkpoint for resilient intermittent systems," in *Proc. 57th IEEE/ACM Int. Symp. Microarchit. (MICRO)*, 2024, pp. 121–135, doi: 10.1109/MICRO61859.2024.00019.
14. J. Choi, J. Choi, H. Joe, and C. Jung, "Caphammer: Exploiting capacitor vulnerability of energy harvesting systems," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 43, no. 11, pp. 3804–3815, Nov. 2024, doi: 10.1109/TCAD.2024.3446879.
15. J. Choi, J. Park, N. Lheureux, Y. Solihin, H. Joe, and C. Jung, "Intermittence-aware speculative page coloring for secure NVM," in *Proc. 53th IEEE/ACM Int. Symp. Comput. Archit. (ISCA)*, 2026.

GAN FANG is a Ph.D. candidate in the Computer Science Department at Purdue University, West Lafayette, IN 47907 USA. Contact him at fang301@purdue.edu.

CHANGHEE JUNG is a Samuel D. Conte Associate Professor in the Computer Science Department at Purdue University, West Lafayette, IN 47907 USA. Contact him at chjung@purdue.edu.