Rethinking Prefetching for Intermittent Computing

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

Jianping Zeng
Samsung Semiconductor
San Jose, CA, USA
jp.zeng@samsung.com

Aditya Gupta
Purdue University
West Lafayette, IN, USA
gupta742@purdue.edu

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

Abstract

Prefetching improves performance by reducing cache misses. However, conventional prefetchers are too aggressive to serve batteryless energy harvesting systems (EHSs) where energy efficiency is the utmost design priority due to weak input energy and the resulting frequent outages. To this end, this paper proposes IPEX, an **I**ntermittence-aware **P**refetching **EX**tension that can be integrated into existing prefetchers on EHSs. IPEX aims to avert useless prefetches by suppressing the prefetching of the cache blocks receiving no hit before their loss on power failure, which would otherwise waste harvested energy. At a proper moment before an upcoming outage, IPEX throttles the prefetch degree to target only those blocks that are likely to be used before the outage. That way IPEX saves energy and spends it on making further execution progress. Experimental results show that on average, IPEX reduces energy consumption by 7.86% (up to 21.64%) and improves performance by 8.96% (up to 23.49%) compared to a conventional prefetcher.

ACM Reference Format:
Gan Fang, Jianping Zeng, Aditya Gupta, and Changhee Jung. 2025. Rethinking Prefetching for Intermittent Computing. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture (ISCA '25), June 21–25, 2025, Tokyo, Japan.* ACM, New York, NY, USA, **??** pages. https://doi.org/10.1145/3695053.3731038

1 Introduction

Batteryless energy harvesting systems (EHSs) have been widely used in many fields, e.g., Internet of Batteryless Things [**? ? ? ? ?**], waterway monitoring [**? ?**], health and wellness tracking [**? ? ? ?**], and wearable technology [**? ? ? ? ?**], thanks to their environmental-friendly, self-sustaining, and maintenance-free [**?**] nature. EHSs harvest energy from ambient sources such as radio frequency (RF), solar, and thermal power [**? ? ? ? ? ? ? ? ? ? ? ? ?**]. Since these energy supplies are often weak and unstable, the harvested energy is first piled up in a tiny capacitor. EHSs (re)boot only when their capacitor is sufficiently charged, die with power outages upon its depletion, and hibernate to charge it again; these steps repeat during the lifetime of EHSs, making their computation *intermittent* [**?**].

The intermittent computing is characterized by frequent outages and the resulting short power cycle, i.e., a period from the reboot time to the following outage. With the intermittence in mind, researchers equip EHSs with nonvolatile memory (NVM) as main memory and some crash consistency mechanism to ensure correct program execution across power failure [**? ? ? ? ? ? ? ? ? ? ? ? ?**]. For example, nonvolatile processor (NVP) [**?**] exploits just-in-time (JIT) checkpointing to save all volatile registers—including program counter (PC)—in nonvolatile flip-flops (NVFFs) before impending power failure so that when power comes back, NVP resumes program execution from the failure point.

In the intermittent computing paradigm, the performance of EHSs is highly affected by (1) input energy quality and (2) their own energy efficiency. If the energy supply is strong, EHSs encounter fewer power outages, keeping the core pipeline busy with ample instructions. On the other hand, for a given amount of harvested energy, the length of a power cycle hinges on how efficiently EHSs utilize the energy. That is why energy efficiency is the utmost priority in the design of EHSs where saved energy can be used to make further execution progress, thus improving their performance.

To improve energy efficiency, NVP adopts a volatile cache whose dirty cache blocks are JIT checked—along with all volatile registers—for crash consistency [**? ? ?**]. Since the cache turns NVM accesses—that are the most energy-consuming in EHSs—to much cheaper SRAM accesses, it can save a considerable amount of energy and thus improve the performance of EHSs dramatically. Prior work [**? ?**] shows that on average, the EHSs backed with volatile caches can achieve roughly a 90% energy consumption reduction and a 8.5x speedup compared to a cache-free EHS.

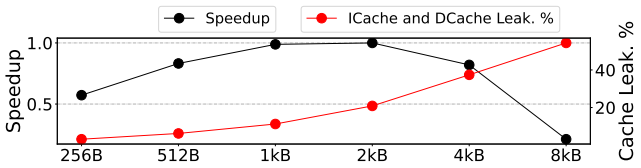


Figure 1: Speedup over baseline (2kB each for ICache/DCache) and cache leakage energy (over total energy consumption); the leakage percentage accounts for both ICache/DCache.

However, EHSs can only afford small volatile caches, which restricts potential performance gain. Figure **??** illustrates how a typical EHS performs differently with cache size variation[1]; black curve represents the trend of speedups over the 2kB baseline, i.e., 2kB each for ICache and DCache. It turns out that while each cache size increases from 256B to 8kB, the performance does not necessarily improve. Once the cache size exceeds 2kB, the performance gain starts to decline due to the increasing cache leakage energy; see red curve in the figure. When ICache and DCache are both 8kB in size, more than half of the total energy (54.38%) is wasted on their leakage, offsetting the benefits of large caches. In light of this, IPEX sets the default cache size to 2kB each for ICache and DCache

[1]Here, hardware prefetchers are disabled; Section **??** details simulation settings.

ISCA '25, Tokyo, Japan

ACM ISBN 979-8-4007-1261-6/2025/06
https://doi.org/10.1145/3695053.3731038

so that the best performance can be delivered as shown in Figure **??**.

Due to the limited cache capacity, EHSs experience frequent cache misses, suffering significant pipeline stalls with long-latency NVM accesses. That is because most EHSs rely on in-order cores [2] owing to their low-power design [**? ? ? ? ? ? ? ? ? ? ? ? ?**]. Hence, when cache misses occur, the core pipeline has no choice but to wait for data being fetched from NVM. Since it is way slower than DRAM, the pipeline stalls occupy a considerable portion of the total execution time, impacting the overall performance of EHSs. Figure **??** shows that for the default setting (2kB ICache and 2kB DCache), 23.45% and 18.64% of the total execution times are wasted due to ICache and DCache misses, respectively. For *pegwite* and *pegwitd*, the stall time caused by DCache misses goes beyond 60%, slowing down the pipeline substantially.

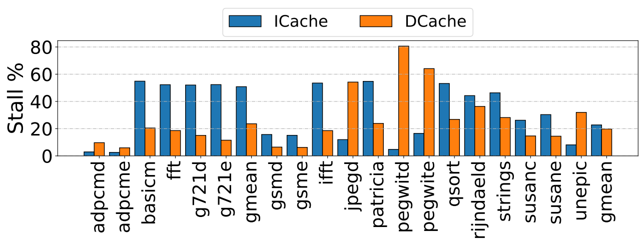


Figure 2: The ratio of the pipeline stall time caused by cache misses to the total program execution time.

To make the best use of a limited cache size, even embedded processors, e.g., ARM Cortex M7 [**?**], use hardware prefetchers that can reduce cache misses by fetching necessary data in advance to the cache [**? ?**]. Unfortunately, conventional prefetchers are not suitable for EHSs, leading to potential energy waste and performance hit. Suppose a cache block has already been prefetched to the cache but not accessed yet. If power failure then occurs immediately, it wipes out the entire cache, rendering the prefetch useless. Here, EHSs would waste the energy—prefetching the block from NVM to the cache—which could otherwise be used to progress further for performance. Without solving this problem, it is practically impossible for EHSs to take advantage of any form of prefetching.

In response, this paper introduces IPEX, an **I**ntermittence-aware **P**refetching **EX**tension that can be integrated into any existing prefetchers. The key insight of IPEX is that it is unnecessary to prefetch those cache blocks whose reuse distance [**? ?**] extends beyond an upcoming power failure, i.e., they are unlikely to be used before the failure. Taking that into account, IPEX dynamically adjusts prefetch degree, i.e., the number of cache blocks to be prefetched at a time, to suppress the useless prefetches in EHSs. If IPEX perceives that a power outage expects to happen soon, it lowers the degree such that only the data being accessed before the outage can be prefetched into caches. That way IPEX effectively reduces the energy waste caused by useless prefetches, leaving more energy for execution progress. When EHSs reboot with their capacitor charged enough, IPEX assumes sufficient input energy and optimistically resets the prefetch degree to its original value, trying to avoid NVM accesses with more aggressive prefetching.

[2]Taming out-of-order cores for EHSs is beyond the scope of this paper.

Nevertheless, there are two issues that must be addressed to bring IPEX into practice: (1) when to adjust a prefetch degree and (2) how large it should be. First, it is essential to determine the optimal timing of adjusting the prefetch degree for maximal energy saving. If the prefetch degree is adjusted late, i.e., too close to an upcoming power outage, there may be a limited opportunity for saving energy. On the contrary, adjusting the degree too early can lead to more cache misses as it may throttle useful prefetch operations. Second, IPEX should be able to accurately determine the appropriate prefetch degree for a given power outage. This is a daunting challenge due to the need for gauging how soon the outage will occur. If the prefetch degree is set too low, EHSs risk encountering more misses before power failure, i.e., data may not be in the cache when they are needed. Conversely, if the degree is set too high, some prefetched blocks may go unused before being lost on power failure, which wastes energy and degrades performance.

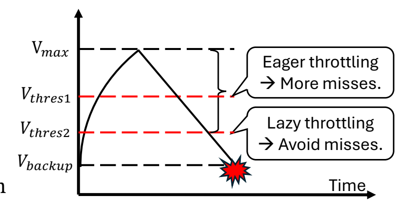


Figure 3: Varying prefetch degree upon crossing one of V thresholds.

To address the first issue of when to adjust, IPEX reformulates it as determining suitable voltage thresholds. That is, IPEX increases and decreases the prefetch degree when the capacitor voltage crosses up and down one of these thresholds. IPEX begins with empirical thresholds and dynamically adjusts them with feedback from the underlying prefetcher. For this purpose, IPEX measures a *throttling rate*, the proportion of the prefetch operations throttled in each power cycle. At reboot time, IPEX adjusts the voltage threshold based on the throttling rate of the prior power cycle. A high rate implies that *eager throttling* occurred therein with potentially more misses; this triggers IPEX to switch to *lazy throttling* by lowering the threshold, e.g., from V_{thres1} to V_{thres2} in Figure **??**, producing more prefetches for cache miss reduction.

With the voltage threshold(s) in place, IPEX addresses the second issue, i.e., prefetch degree determination, by tuning it during program execution. According to capacitor voltage changes relative to the threshold(s), IPEX determines the prefetch degree, which strikes a balance between high performance and energy efficiency. In other words, whenever the capacitor voltage drops below a specific threshold (approaching an outage), IPEX reduces the prefetch degree to reflect that it should now issue fewer prefetches to save the energy waste. In reverse, IPEX increases the prefetch degree when the capacitor voltage rises above the threshold.

The experiments show that for 20 applications, IPEX reduces energy consumption by 7.86% and improves performance by 8.96% compared to conventional prefetchers. In summary, IPEX makes the following contributions:

- It is the first to enhance prefetchers' performance by considering the unique characteristics (i.e., frequent power outages) of EHSs.
- It achieves promising energy saving (up to 21.64%) and performance gain (up to 23.49%) over the conventional prefetcher.
- It needs only 4 registers per cache (0.0018% of core area including ICache/DCache), since it reuses the existing prefetcher hardware.

2 Background and Motivation

2.1 Energy Harvesting System (EHS)

Due to the inherently unstable nature of the input energy, EHSs run from the moment their capacitor is sufficiently charged till it is depleted; EHSs then undergo a lengthy recharging period before securing enough energy in their capacitors again [? ? ? ? ? ? ? ? ? ? ? ? ?]. Therefore, given a certain quality of the input energy, EHSs' energy efficiency becomes crucial in making the most of the harvested energy with the overall recharging time and the running time both lowered. ***The takeaway is that energy efficiency must be the utmost design factor for EHSs to maximize their runtime performance.***

One way to improve energy efficiency is equipping EHSs with a volatile SRAM cache. It effectively reduces the amount of accesses to NVM by exploiting locality in user program. As a result, EHSs can retrieve program data from the cache, which is way more energy-efficient than accessing NVM. A notable instance of cache-enabled EHSs is NVSRAMCache [? ? ? ? ? ? ? ? ?] that achieves a drastic speedup (8.5x) over the cache-free baseline with a real RF power trace [?]. To ensure crash consistency for the SRAM cache, when a voltage monitor detects upcoming power failure, NVSRAMCache signals its backup/restoration controller to JIT checkpoint (1) the dirty cache blocks in the SRAM cache to its NVM counterpart [? ? ?] and (2) all volatile register values—including program counter (PC)—to nonvolatile flip-flops (NVFFs) [? ?]. In the wake of the power failure, the controller reversely restores all these checkpointed states, resuming the interrupted program from the failure point.

Nonetheless, the maximum performance gain of cache-enabled EHSs is technically capped. Figure ?? highlights that larger caches would degrade the overall performance due to their increasingly significant leakage power. Because of that, EHSs are typically equipped with small caches, e.g., 2kB, and thus experience high cache miss rates and the associated stalling costs as shown in Figure ??. Even worse, during the stalling time, EHSs continuously drain energy through leakage from various hardware components, e.g., the core pipeline resources and caches, wasting hard-harvested energy and resulting in suboptimal performance.

2.2 Exploration and Characterization for Energy-Efficient Prefetching

A conventional approach to reducing the pipeline stalls caused by cache misses is prefetching both instructions and data from main memory to caches in advance. This enables the core pipeline to quickly access instructions and data from the caches when needed. This is also possible for EHSs, though they experience frequent power failure, as their applications still exhibit some access patterns within each power cycle.

However, prefetching is a speculative technique which inherently carries the risk of misspeculation and the resulting penalty. Those blocks prefetched into caches may never receive hits, wasting hard-harvested energy which would otherwise be used to make further execution progress. With that in mind, it is important to carefully explore and characterize prefetchers so that EHS can utilize them to the best advantage. To gain a deeper understanding of the impact of prefetching in EHS, we first formulate the energy waste caused by cache misses in the absence of prefetching, as shown in Equation ??. The energy waste of a prefetch-free EHS ($E_{waste\_no\_prefetcher}$) is solely caused by the system leakage energy during the stall times of handling the cache misses (E_{leak}).

$$E_{waste\_no\_prefetcher} = E_{leak} \tag{1}$$

$$E_{waste\_prefetcher} = \begin{cases} 0 & \text{, useful prefetch} \\ E_{prefetch} + E_{leak} & \text{, useless prefetch} \end{cases} \tag{2}$$

Also, we formulate the energy waste yet with a simple sequential prefetcher enabled ($E_{waste\_prefetcher}$) as shown in Equation ??. Here, if a prefetch contributes to cache hits, no energy is wasted since it prevents misses from being occurred. However, if a prefetched block ends up being not used, energy waste comes from two parts: fetching the block from memory to caches ($E_{prefetch}$) and the system leakage of the EHS during the stall times of handling the cache miss (E_{leak}). Suppose a prefetcher has a probability P of fetching a useful cache block. To ensure that the prefetcher can enhance the performance of EHSs, $E_{waste\_prefetcher}$ must be smaller than $E_{waste\_no\_prefetcher}$. In other words, Inequality ?? has to hold, i.e., P is required to be at least $1 - E_{leak}/(E_{prefetch} + E_{leak})$ as shown in Inequality ??.

$$(1 - P) * E_{waste\_prefetcher} < E_{waste\_no\_prefetcher} \tag{3}$$

$$P > 1 - E_{leak}/(E_{prefetch} + E_{leak}) \tag{4}$$

The implication is that as $E_{prefetch}$ and E_{leak} change, the minimum probability P required to make the prefetching beneficial also changes accordingly. The takeaway is that as illustrated in Figure ??, a higher prefetch cost ($E_{prefetch}$) increases the minimum required probability P, whereas higher leakage energy consumption (E_{leak}) reduces the probability. Based on the default configuration of our EHS described in Section ??, the minimum probability P required for effective prefetching turns out to be 46.04%. In particular, the observed probabilities of the EHS' ICache and DCache prefetching useful cache blocks are 54.03% and 52.88%, respectively, and they are both greater than the minimum required probability P. This indicates that prefetching has potential to benefit EHSs.

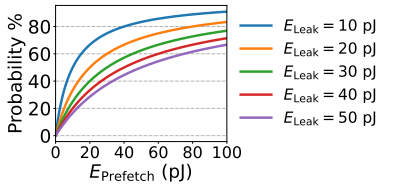


Figure 4: Relationship between minimum required P, $E_{prefetch}$, and E_{leak}.

2.3 Challenge of Prefetching across Frequent Power Outages

In addition to program behavior with recurring memory access patterns, however, whether prefetched blocks are useful also depends on when power failure occurs. If blindly applied, prefetching cannot achieve optimal performance for EHSs. This is because all volatile cache contents—including those prefetched blocks—are lost upon power failure, rendering the prefetching useless if the prefetched blocks do not receive any hits before the failure.

Figure ?? illustrates an example of useless prefetches in the presence of power interruptions. At time T_1, the prefetcher predicts that *Blocks A* and *B* will be needed soon, so it loads them into the volatile

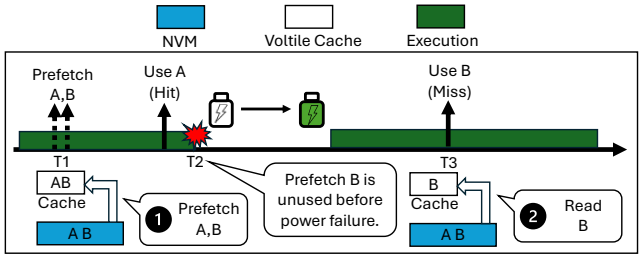


Figure 5: The inefficiency of existing prefetchers in the presence of power failure; the delay between generating the prefetch operations and receiving the fetched cache blocks is ignored for the sake of simplicity.

cache (❶) to avoid potential cache misses. When the core pipeline progresses to the point where *Block A* is desired, it retrieves the prefetched data from the cache (hit) with no pipeline stall. Then, at time T_2, power failure occurs and wipes out all volatile data including those stored in the cache, rendering the prefetching of *Block B* useless and wasting harvested energy. Indeed, resuming from the failure point, the core pipeline encounters a cache miss at time T_3, causing an expensive NVM read of *Block B* (❷), which would have been absent if the power failure had not occurred. This is a severe performance issue for EHSs as they experience frequent power interruptions, i.e., many prefetch operations consume hard-won energy in vain becoming useless. ***The challenge here is that existing prefetchers do not account for power interruptions, missing valuable opportunities to eliminate costly NVM accesses for EHSs.***

3 IPEX Approach

3.1 Reconsidering the Timeliness of Prefetching for Energy Harvesting Systems

To address the above challenge, IPEX for the first time reconsiders the timeliness of prefetching for EHSs. If the expected use of instructions or data falls within the current power cycle (i.e., a period from the reboot time to the following power failure), then it is all right for the core pipeline to proceed with prefetching as usual. However, one should not prefetch them, provided their use is anticipated beyond the current power cycle. The reason is that the prefetched blocks are going to be lost anyway on the power failure without receiving any hit. This again wastes the hard-harvested energy which would otherwise be used to make further program progress for better performance.

In response, IPEX takes into account frequent power outages when making a decision on generating prefetch operations. Unlike existing prefetchers [? ? ? ? ? ? ? ? ? ? ? ? ?] which only consider memory access patterns, IPEX adjusts a prefetch degree, i.e., the number of cache blocks to be prefetched at a time, according to not only program behavior but also power outages. As a result, IPEX allows the desired data to be available in caches at the right time while suppressing the issue of useless prefetches. The upshot is that IPEX can realize the full potential of prefetching for EHSs even in the presence of frequent power outages.

3.2 Overview of IPEX

What makes IPEX stand out is the ability to dynamically throttle existing prefetchers atop EHSs across frequent power outages while maintaining performance. To achieve this, IPEX performs bi-modal control of the underlying prefetcher, switching between (1) *energy saving mode* and (2) *high performance mode* depending on the likelihood of power failure. That is, when the capacitor voltage falls below predefined thresholds, indicating potential power failure in the near future, IPEX enters into *energy saving mode* where it throttles the prefetchers by reducing the prefetch degree. As such, IPEX avoids prefetching the cache blocks that are not likely to be accessed before power loss, thus lowering energy waste. To prevent IPEX from being stuck in *energy saving mode* unnecessarily long, IPEX raises the prefetch degree once the capacitor voltage rises above the thresholds, i.e., IPEX switches back to *high performance mode*. This allows IPEX to reissue any prefetch operations that were throttled earlier. ***Consequently, IPEX can balance between energy saving and high performance.***

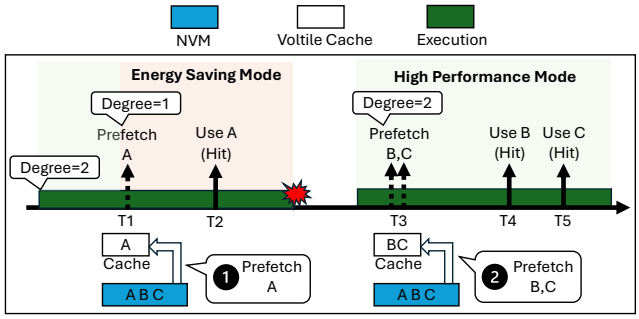


Figure 6: The high-level workflow of IPEX; the delay between generating the prefetch operations and receiving the data blocks is ignored for the sake of simplicity.

To illustrate, Figure **??** provides an overview of how IPEX adapts its prefetching to power failure. At time T_1, IPEX perceives that the power is about to be cut off and transitions to *energy saving mode*. In this mode, IPEX proactively lowers the prefetch degree from its original value (e.g., 2 in this example) to 1. This causes IPEX to prefetch only *Block A* (❶), contributing to a cache hit at time T_2 while avoiding prefetching the unnecessary *Block B* or *C*—since they are not going to be accessed before the impending power failure. Upon power resumption at time T_3, IPEX switches to *high performance mode*, resets the prefetch degree to 2, and issues two prefetch requests for *Block B* and *C* (❷). This allows their following uses at times T_4 and T_5, respectively, to hit in the cache, avoiding pipeline stalls and improving the performance.

4 Implementation Details

The goal of IPEX is tuning the prefetch degree on the fly in response to anticipated power interruptions to unleash the potential of prefetching for EHSs. For this purpose, IPEX addresses two key technical questions: when to adjust the prefetch degree, i.e., on what voltage thresholds should be (Section **??**), and how to determine the appropriate value of the prefetch degree (Section **??**). The following two sections detail how IPEX tackles each of these questions, respectively.

4.1 When to Adjust a Prefetch Degree

Recall that the first question can be translated into how to figure out suitable voltage thresholds. However, identifying an optimal voltage threshold is a challenging task not just due to (1) the difficulty of using the capacitor voltage as a proxy of the likelihood of power failure but also because of (2) the reliance on the energy harvesting quality necessitating the threshold adaptation, i.e., there is no one-size-fits-all approach.

First, setting the voltage threshold too high—called *eager throttling* as the voltage is reached soon after reboot which is far from upcoming failure—would lead to suboptimal performance resulting from the insufficient number of prefetches being issued. Conversely, setting the threshold too low—called *lazy throttling* as it is reached near upcoming failure taking some time after reboot—might waste hard-won energy on potential useless prefetch operations in the meantime.

To address the aforementioned challenge, IPEX adopts an empirical approach in the first place, by taking into account the real energy conditions of EHSs. At reboot time, IPEX sets each voltage threshold to an initial value that empirically performs best in our evaluation, e.g., a voltage threshold is set to 3.3V (another to 3.25V) as shown in Figure **??**.

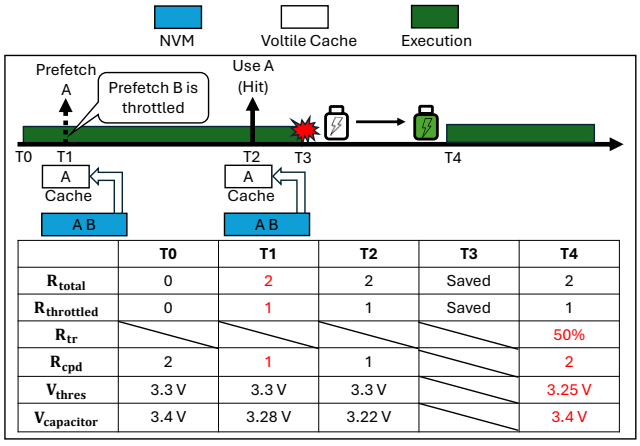
4.1.1 Adaptive Voltage Threshold Adjustment. Second, regardless of power failure, a fixed voltage threshold may not perform the best at all times due to the fluctuation of the energy harvesting quality, even if no power failure occurs for the time being. As an example, the capacitor voltage once falling below the threshold could rise above it shortly afterward, in which case permanently throttling prefetching would miss the opportunity to prefetch useful cache blocks and thus degrade performance.

In response, IPEX introduces a simple yet effective technique that can adaptively tune voltage thresholds based on a newly proposed metric called *throttling rate*. It is denoted as $P_{tr} = P_{throttled}/P_{total}$ where $P_{throttled}$ represents the portion of throttled prefetches, while P_{total} indicates the sum of the number of issued prefetches and the number of throttled prefetches. The rationale behind using P_{tr} is twofold: (1) when P_{tr} goes too high, i.e., over-throttling, IPEX should lower the voltage thresholds so that it generates more prefetching requests to achieve high performance; (2) on the contrary, if P_{tr} becomes so low, i.e., under-throttling, the voltage thresholds should be raised to reduce potential energy waste possibly caused by useless prefetches.

To compute $P_{throttled}$ and P_{total}, IPEX collects necessary statistics as the core pipeline is running. For this purpose, IPEX devises four volatile registers, i.e., $R_{throttled}$, R_{total}, R_{tr}, and R_{ipd}, for each cache (ICache/DCache) to independently manage its prefetcher based on its own statistics. The first three are 32-bit registers; $R_{throttled}$ and R_{total} track $P_{throttled}$ and P_{total}, respectively, while R_{tr} is a floating-point register and records the calculated *throttling rate*. Finally, R_{ipd} contains the initial prefetch degree (e.g., 2) and is consulted by existing ICache/DCache prefetchers for them to set up the prefetch degree at reboot time. That is, the prefetchers reset their internal register called R_{cpd}, that holds the current prefetch degree, to their R_{ipd} in the wake of power failure. In particular, R_{ipd} requires only 3 bits as IPEX allows the maximal prefetch degree of 4 for typical EHSs.

With the help of these hardware structures, it becomes straightforward to compute $R_{throttled}$ and R_{total}. In other words, when a prefetcher cannot issue a prefetch operation due to the throttling of IPEX , it simply increases $R_{throttled}$ by 1. Similarly, after issuing the requested prefetch operations, the prefetcher just increments R_{total} to keep track of the number of issued prefetches. Leveraging $R_{throttled}$ and R_{total}, it is trivial to compute *throttling rate*. Each time a power cycle starts upon the reboot after power failure, IPEX restores register $R_{throttled}$ and R_{total} from NVM—as they were JIT checkpointed right before the power failure—and writes their division result to R_{tr}, i.e., $R_{tr} = R_{throttled}/R_{total}$.

With the *throttling rate* recorded in R_{tr}, IPEX updates each voltage threshold upon reboot, i.e., at the beginning of each power cycle, and keeps the rate unchanged throughout that cycle. To be specific, IPEX decreases the voltage threshold by 0.05V if R_{tr} is not less than 5%—empirically determined through experimentation—so that IPEX can issue more prefetch operations to handle potential cache misses and thus improve the overall performance. Otherwise, IPEX increases the voltage threshold by 0.05V, which restrains IPEX from generating more prefetch operations, for the sake of energy saving.



	T0	T1	T2	T3	T4
R_{total}	0	2	2	Saved	2
$R_{throttled}$	0	1	1	Saved	1
R_{tr}					50%
R_{cpd}	2	1	1		2
V_{thres}	3.3 V	3.3 V	3.3 V		3.25 V
$V_{capacitor}$	3.4 V	3.28 V	3.22 V		3.4 V

Figure 7: Implementation details of adaptively adjusting a voltage threshold; R_{cpd} is an internal register available in the existing prefetchers and keeps their current prefetch degree.

Figure **??** shows an example of how IPEX uses those four registers to dynamically adjust a voltage threshold. Suppose R_{ipd} and initial voltage threshold are 2 and 3.3V, respectively, at the beginning of the power cycle. At time T_1, IPEX lowers the prefetch degree (R_{cpd}) from 2 to 1 as the capacitor voltage ($V_{capacitor}$) drops below 3.3V—the discussion on how IPEX adjusts the prefetch degree is deferred to Section **??**. Because of this, IPEX only prefetches *Block A* at time T_1, avoiding the prefetch for *Block B*. In addition, IPEX updates its hardware structures, e.g., $R_{throttled}$, R_{total}, R_{cpd}. That is, IPEX increases $R_{throttled}$ and R_{total} by 1 and 2 (*Block A* prefetched and *Block B* suppressed), respectively. At time T_2, reading from the address of *Block A* hits in the cache, because the block has been present therein since time T_1. Upon power failure at time T_3, IPEX JIT checkpoints $R_{throttled}$ and R_{total} so that they can be restored in the wake of the power failure (i.e., at time T_4). When the next power cycle begins at time T_4, IPEX first calculates R_{tr} as 50%, resets R_{cpd}

to R_{ipd} (i.e., 2), and decreases V_{thres} by 0.05 V since R_{tr} is greater than 5%.

4.2 How to Determine a Prefetch Degree

This section first presents a straightforward yet inefficient approach to deciding a prefetch degree along with a lesson from the approach and then introduces IPEX's efficient approach.

Naive Approach: One might try to use a single voltage threshold (V_{thres}) and reduce the prefetch degree whenever the capacitor's voltage falls below V_{thres}. Although this approach eliminates some useless prefetches, many others may still not be averted. Figure ?? illustrates such an example where the original prefetch degree is 6. At time T_2, IPEX reduces it to 3 (❶) and thus later on issues only 3 prefetch requests retrieving *Block A*, *B* and *C* (❷) at time T_3. The underlying assumption is that these 3 prefetched blocks could contribute to cache hits in the current power cycle. Unfortunately, this assumption turns out to be wrong in this example: only *Block A* receives a cache hit at time T_4, and power failure soon occurs erasing prefetched *Block B* and *C*—before their access. Consequently, the naive approach ends up wasting the harvested energy of the EHS and failing to achieve high performance.

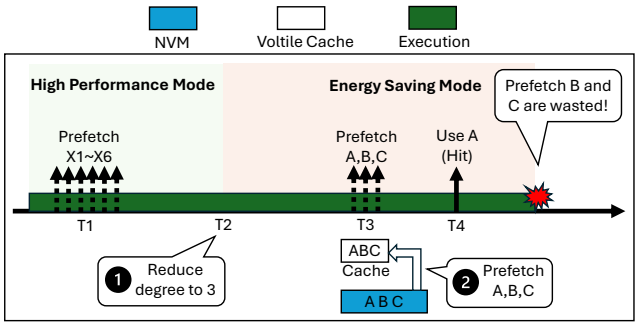
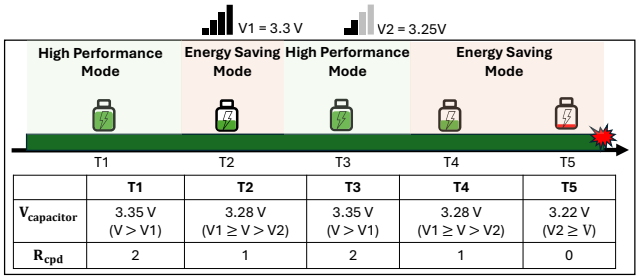


Figure 8: Stratghtforward approach that cannot minimize useless prefetches; suppose the original prefetch degree is 6.

IPEX's Efficient Approach: The crux of the problem with the naive approach is that it does not account for the growing number of useless prefetches as the EHS nears power failure. Specifically, after throttling at time T_2 (❶), there are no further adjustments to the prefetch degree, though the capacitor voltage continues to drop toward power failure.

The lesson here highlights the importance of the ability to adjust the prefetch degree as the capacitor voltage changes. In light of this, IPEX leverages multiple voltage thresholds—ranging from V_1 (highest) to V_k (lowest)[3]. Whenever each threshold is reached, IPEX triggers its prefetch degree adjustment. That is, by using multiple voltage thresholds, IPEX can control the aggressiveness of its prefetch degree throttling. A higher threshold indicates that the EHS would be far from power failure, i.e., many prefetch operations still remain useful. Hence, IPEX adopts a conservative approach, avoiding excessive reduction of the prefetch degree. In contrast, a lower voltage threshold indicates that the EHS is much closer to power failure, allowing IPEX to throttle prefetches more aggressively—preferring a smaller prefetch degree—since most prefetched blocks are unlikely to be used before the failure. Therefore, it is critical for IPEX to ensure an appropriate prefetch degree at each voltage level. To realize this, IPEX halves the prefetch degree each time the capacitor voltage falls below a threshold, and doubles the degree when the voltage rises above a threshold. This dynamic adjustment enables IPEX to respond to fluctuating input energy, thereby achieving high performance while reducing energy waste.

[3]IPEX defaults k to 2 in the simulation. Section ?? varies it for sensitivity analysis.



	T1	T2	T3	T4	T5
$V_{capacitor}$	3.35 V (V > V1)	3.28 V (V1 ≥ V > V2)	3.35 V (V > V1)	3.28 V (V1 ≥ V > V2)	3.22 V (V2 ≥ V)
R_{cpd}	2	1	2	1	0

Figure 9: Adjustment of prefetch degree with two voltage thresholds enabled; their values are determined by the techniques described in Section ??.

Figure ?? shows an example of how IPEX dynamically adjusts the prefetch degree when using two voltage thresholds; they are set to 3.3 V and 3.25 V, respectively. Suppose the initial prefetch degree is 2. When the capacitor voltage $V_{capacitor}$ is above V_1 at time T_1, implying that a power interruption is unlikely in the near future, the prefetcher thus operates as usual, staying in *high performance mode* without being throttled by IPEX. Once $V_{capacitor}$ drops below V_1 at time T_2, IPEX throttles the prefetching, entering into *energy saving mode* with the current prefetch degree R_{cpd} halved to 1. On the other hand, if $V_{capacitor}$ rises above V_1 again at time T_3, IPEX doubles the prefetch degree R_{cpd} from 1 to 2, switching back to *high performance mode*. As $V_{capacitor}$ falls once more, crossing thresholds V_1 and V_2 at time T_4 and T_5, respectively, IPEX progressively lowers R_{cpd} to 1 and eventually to 0 as shown in the figure.

5 Discussion

5.1 Handling Late Prefetches

Suppose the EHS temporarily enters *energy saving mode* and exits sometime later, in which case IPEX initially throttles prefetch operations but may reissue them upon returning to *high performance mode*. Due to the resulting delay in issuing the prefetch operations, some of them could complete later than desired, ending up with cache misses. Fortunately, IPEX can mitigate this problem in two ways. First, IPEX, in its current form, reduces the likelihood of the late prefetches through its multiple voltage thresholds. Instead of aggressively halting all prefetches when entering into *energy saving mode*, IPEX adopts a conservative approach, gradually reducing the prefetch degree. That way IPEX can issue many prefetches in a timely manner with the order prioritized by the underlying prefetcher, thereby decreasing the chances of late prefetches. Second, when switching to *high performance mode* from *energy saving mode*, IPEX can be extended to reissue all previously throttled prefetches. As a result, even if the required cache blocks are not immediately available, the core pipeline might stall only for a short period, lowering the penalty during the wait. We leave this optimization as our future work.

In addition, late prefetches can lead to another inefficiency. For example, upon a cache miss, the EHS may blindly issue a duplicate memory request even though a prior prefetch for the same block is already in progress. That is, the prefetch should have been issued much earlier to avoid wasting energy and degrading performance. To prevent this, whenever a cache miss occurs, IPEX first looks up the prefetch buffer to see whether a request for the desired block is pending; if that is the case, IPEX refrains from issuing a redundant request until the pending prefetch completes; otherwise, IPEX allows the cache miss to be handled as usual.

5.2 Application to Complex Prefetchers

Although the analysis and evaluation of this paper focuses on simple prefetchers (i.e., the sequential and stride prefetchers listed in Table **??**), IPEX can easily be applied to more complex prefetchers as well. That is because these complex prefetchers leverage either prefetch degree registers or some alternative hardware structures that determine how many cache blocks to fetch. By adding a lightweight control layer for monitoring the capacitor voltage $V_{capacitor}$ and adjusting the prefetch degree accordingly, IPEX can seamlessly integrate with any hardware prefetcher.

Rather, complex prefetchers are a great beneficiary of IPEX. In fact, they offer even more opportunities for IPEX to optimize, compared to simple prefetchers. Specifically, the complex prefetchers often involve energy-consuming prefetch address generation, e.g., through a costly table lookup [**?** **?**], which becomes as wasteful as useless prefetch operations especially when a power outage is imminent. Therefore, upon entering *energy saving mode*, IPEX can not only avert useless prefetch operations but also disable the useless address generation, thereby further reducing energy waste.

6 Evaluation and Experimental Analyses

We implement IPEX and other prefetching schemes on top of gem5 [**?**], a cycle-level architecture simulator, to model a single-core in-order nonvolatile processor clocked at 200 MHz [**?**]. This configuration has been validated against measurements from a real NVP platform [**?**]. All the evaluated applications are compiled for ARMv7-M ISA and statically linked. We model energy consumption using McPAT [**?**] and NVSim [**?**] with 45 nm technology, following prior work [**?**]. To ensure realistic simulation parameters for EHSs, we employ the low-power cache and NVM libraries from NVSim [**?**], which are tailored to ultra-low power embedded systems. In the baseline architecture, a single CPU is connected to on-chip nonvolatile main memory (NVM) via a simple bus optimized for ultra-low power consumption and efficient data access. The interconnect utilizes short and energy-efficient pathways to minimize the energy loss during communication between the CPU and the main memory. Additionally, the interconnect is designed to optimize the NVM read operation such that it consumes less energy than writes [**?**].

For the baseline, we evaluate NVSRAMCache [**?**] with both instruction prefetcher and data prefetcher enabled, and prefetched blocks are placed in prefetcher buffers to avoid polluting ICache and DCache. Table **??** provides the simulation configurations for IPEX and NVSRAMCache, with both instruction and data prefetchers enabled. In all simulations, both DCache and ICache are based on SRAM and configured as 2kB 4-way set-associative by default.

Table 1: Simulation Configuration.

	NVSRAMCache (baseline)	**IPEX**
Capacitor	0.47 μF	
ICache/DCache	2kB 4-way SRAM with 16B block size, LRU replacement, and 1 cycle hit latency, Access: 0.015 nJ; Leak: 0.205 mW	
Prefetch Buffer	4 16-byte entries for each cache	
Data Prefetcher	Stride Prefetcher (default)	
Inst. Prefetcher	Sequential Prefetcher (default)	
Main Memory	16MB ReRAM, Read: 0.039 nJ; Write: 0.160 nJ; Leak: 12.133 mW	
Prefetch Degree	2 initially and up to 4	
V_{thres} Count	2 by default	

As for benchmarks, we utilize 20 applications from Mediabench [**?**] and MiBench [**?**] for fair and accurate evaluation [**?**]. To show off how performant IPEX is under varying energy conditions, we evaluate it using multiple real-world power traces, e.g., RFHome, RFOffice, solar, and thermal [**?** **?**]. The solar and thermal sources have relatively higher portions of stable energy, while RFHome and RFOffice have less. To ensure that the simulation with different configurations can receive the same amount of input energy, we digitize the input energy and record it for repeated uses. Specifically, we use an energy harvester to collect ambient energy and log the input power values into a text file. This file contains a series of numbers, each representing the average input power over a 10 μs interval (i.e., $P_{avg} = E_{10\mu s}/10\mu s$, where $E_{10\mu s}$ is the harvested energy for every 10 μs). During the execution of the simulator, it reads these recorded values from the text file, stores the energy in the capacitor of the energy harvesting system (EHS), and powers the EHS to run a benchmark program, which is repeated across the resulting power outages along the way until the completion of the program. This approach ensures that all simulations with different configurations receive the same amount of input energy, achieving fair comparisons.

6.1 Hardware Overhead Analysis

IPEX incurs a trivial hardware overhead owing to its simple hardware design. It devises 4 volatile registers: $R_{throttled}$, R_{total}, R_{tr}, and R_{ipd} for each cache. The first three registers are 32-bit each, while the last one needs only 3 bits. In total, IPEX requires 99 bits per cache, a total of 198 bits for ICache and DCache. In general, these additional registers account for only 0.0018% of the core chip area (0.54 mm^2, including caches), as estimated using CACTI [**?** **?**] with 45 nm technology.

6.2 Run-Time Performance

Figure **??** shows performance comparison between NVSRAMCache with and without prefetchers for a variety of applications. On average, enabling the default prefetchers (Table **??**) for ICache and DCache leads to a 4.96% performance improvement. We also applied IPEX solely to the default data prefetcher and to both default prefetchers used in ICache and DCache. As shown in Figure **??**, IPEX significantly improves the performance of NVSRAMCache

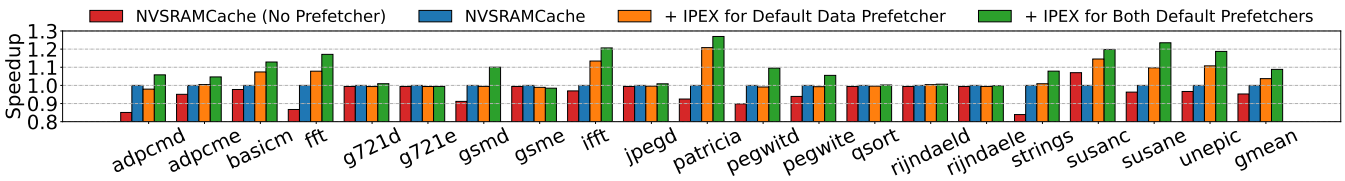


Figure 10: Normalized performance speed to NVSRAMCache (baseline) with RFHome power trace; default prefetchers are enabled for NVSRAMCache.

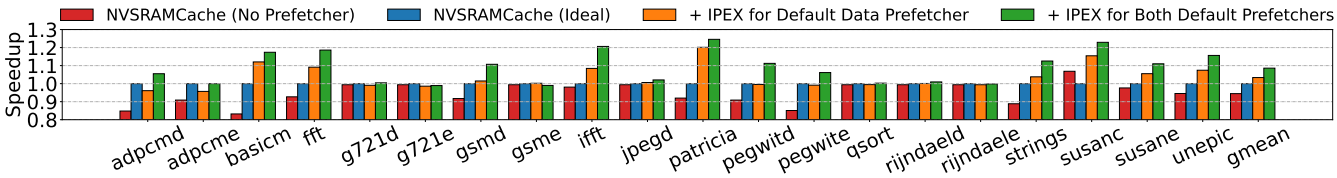


Figure 11: Normalized performance speed to NVSRAMCache (ideal) with RFHome power trace; default prefetchers are enabled for NVSRAMCache (ideal).

(baseline) thanks to its ability to save energy from useless but expensive prefetch operations. Specifically, IPEX outperforms the baseline by an average of 3.73% when only applied to the DCache prefetcher and by an average of 8.96% when applied to both ICache and DCache prefetchers.

Interestingly, IPEX boosts the performance of the instruction prefetcher more than that of the data prefetcher. This is because the evaluated applications exhibit substantially more ICache accesses—4x on average—compared to DCache accesses, which gives IPEX more opportunities to reduce useless prefetch operations in the instruction prefetcher. Notably, IPEX exhibits marginal improvements for certain applications, e.g., *g721d* and *g721e*, as the core pipeline generates fewer prefetch operations due to the inherent program characteristics of the applications. This leaves a smaller room for IPEX to throttle prefetches.

In addition, we implement an optimal version of NVSRAMCache by setting its checkpoint and restoration overheads to zero. We believe this NVSRAMCache (ideal) represents the upper bound of any EHSs that are equipped with caches. As Figure **??** shows that **IPEX still achieves a significant performance gain over the NVSRAMCache (ideal), i.e., an average of 9.06% and up to 26.02% speedup, demonstrating the potential of applying IPEX to any kind of EHSs to improve their performance**. In particular, IPEX brings quite a higher performance improvement to NVSRAMCache (ideal) for several applications, e.g., *basicm*, *fft*, *susane*, and *unepic*.

6.3 Prefetch Operation Reduction

To show how effective IPEX is in throttling prefetches when it is applied to both ICache and DCache prefetchers, we calculate the reduction in prefetch operations. As shown in Figure **??**, IPEX reduces an average of 7.11% prefetch operations, which translates to significant improvement in energy efficiency and run-time performance. In particular, IPEX brings more than 15% reduction in prefetch operations for some applications, e.g., *gsme* and *rijndaeld*. However, a high reduction ratio of prefetch operations does not necessarily lead to significant performance improvements or energy savings. For instance, IPEX might mistakenly throttle useful prefetches, failing to hide their cache misses. Furthermore, if the energy costs

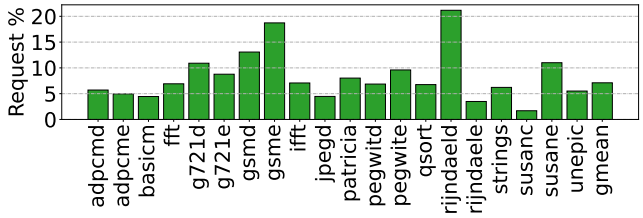


Figure 12: Reduction ratio of prefetch operations when applying IPEX to both ICache and DCache prefetchers.

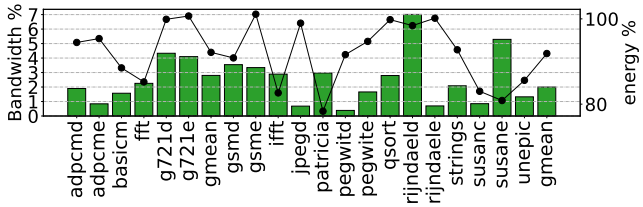


Figure 13: Bars show the reduction ratio of main memory traffic when applying IPEX to both ICache and DCache prefetchers, while the line shows normalized energy consumption to NVSRAMCache (baseline) with RFHome power trace.

associated with prefetching constitute only a small portion of the total energy consumption, eliminating useless prefetches may only yield marginal gains.

6.4 Memory Traffic Reduction

As IPEX throttles prefetch operations, it can lower the pressure of traffic to main memory. To confirm this, we investigate how much memory traffic is reduced by IPEX. Figure **??** illustrates that an average of 2.00% memory traffic is eliminated thanks to IPEX's ability to suppress useless prefetch operations (i.e., memory accesses). It is worth noting that IPEX saves more than 5% memory traffic, especially for some applications, e.g., *rijndaeld* and *susane*. Figure **??** also presents the normalized total energy consumption for each application. Notably, some applications exhibit a strong correlation between memory traffic reduction and overall energy consumption, while others do not. This occurs because IPEX may mistakenly throttle useful prefetches, causing processor pipeline

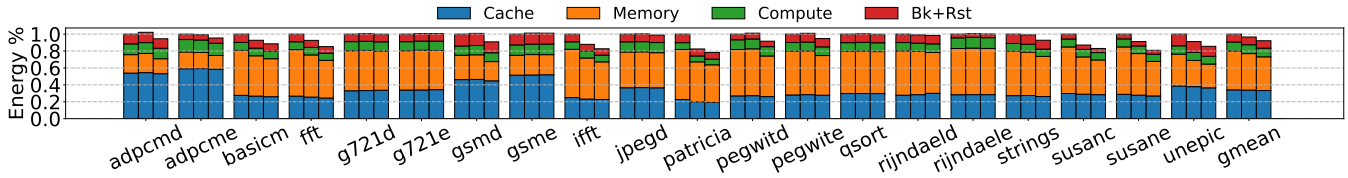


Figure 14: Normalized energy breakdown to NVSRAMCache (baseline) with RFHome power trace. There are 3 bars for each application. *From left to right: NVSRAMCache, + IPEX for DCache prefetcher, and +IPEX for Both DCache and ICache prefetchers*

stalls and increased the system leakage consumption (as discussed in Section **??**). Consequently, even when IPEX significantly reduces memory traffic, total energy consumption may remain high.

6.5 Energy Efficiency

To demonstrate how much energy IPEX can save, we normalize the energy consumption of IPEX to the NVSRAMCache (baseline) and break it down into four portions: cache, memory, computing, and checkpoint/restoration. As shown in Figure **??**, when applied to only the data prefetcher, IPEX reduces the energy consumption of the main memory by 6.03% and the overall energy consumption by 3.42%. By applying IPEX to both ICache and DCache prefetchers, the reduction in energy consumption by IPEX goes up to 13.24% for the main memory and 7.86% for the overall system. We attribute this large energy saving to two factors: (1) a significant reduction in prefetch operations as discussed in Section **??**; and (2) negligible increases in cache misses, i.e., 0.08% and 0.02% for ICache and DCache, respectively, as shown in Figure **??**.

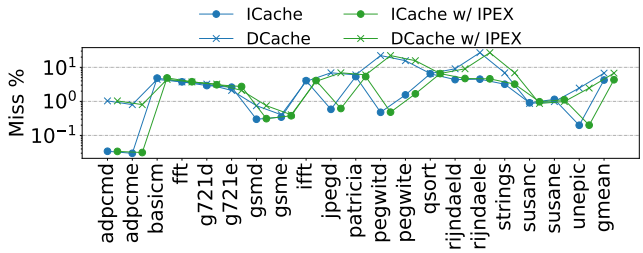


Figure 15: Cache miss rates with and without IPEX for both ICache and DCache prefetchers.

6.6 Prefetch Accuracy and Coverage

To further demonstrate why IPEX is performant, we calculate prefetching accuracy and coverage for ICache and DCache with and without IPEX. Table **??** shows that IPEX significantly enhances the accuracy, i.e., increasing it by 35% for ICache and 22.8% for DCache, while having a minor impact on the coverage, i.e., only 3% and 5% changes in the coverage for ICache and DCache, respectively.

Table 2: Prefetch accuracy (%) and coverage (%).

	Acc. (Inst.)	Acc. (Data)	Cov. (Inst.)	Cov. (Data)
NVSRAMCache	54.03	52.88	80.56	64.51
IPEX	72.88	64.93	78.24	61.44

6.7 Sensitivity Analysis

6.7.1 Voltage Threshold Counts. To show the impact of the number of voltage thresholds on IPEX, we conduct experiments with varying the voltage threshold count from 1 to 3, while keeping other parameters the same. Figure **??** shows IPEX's speedups over the NVSRAMCache baseline with the different threshold count. As shown in the figure, one threshold leads to the worst performance improvement (i.e., 6.32%) as it limits IPEX's adaptability to fluctuating energy quality. With more voltage thresholds, IPEX can control the prefetch degree in a finer-grained manner, which enables it to strike a balance between energy saving and high performance. It is worth noting that the performance improvement slightly plateaus with three voltage thresholds. This is because additional thresholds may lead to overly aggressive adjustments in the prefetch degree, ultimately offsetting the benefits. Therefore, IPEX is configured with a default of two voltage thresholds.

6.7.2 Different Prefetchers. As discussed in Section **??**, there are myriad prefetching techniques employed in real-world processors. To demonstrate if IPEX still works well for other prefetchers, we evaluate several instruction and data prefetchers on top of NVSRAMCache and apply IPEX to assess its impact on performance improvement. As shown in Table **??**, IPEX achieves significant speedups across three instruction prefetchers, demonstrating its potential to serve for various EHSs. Interestingly, IPEX delivers even higher speedup with aggressive prefetchers like TIFS [**?**], as their aggressive policy generates more useless prefetch operations for EHSs. This provides IPEX with greater opportunities to reduce energy waste, thereby improving performance.

Table 3: Speedup of IPEX with varying inst. prefetchers.

Prefetchers	Sequential (default)	Markov	TIFS
IPEX	8.96%	7.89%	9.05%

On the other hand, Table **??** presents the performance impact of applying IPEX to different data prefetchers while using the default instruction prefetcher shown in Table **??**. The results indicate that IPEX consistently delivers high speedups regardless of which data prefetcher is used. This is because all those prefetchers generate fewer prefetch operations given that power cycles are typically short in EHSs, which brings a limited opportunity for IPEX to avert useless prefetches; see Section **??** for detailed discussion.

Table 4: Speedup of IPEX with varying data prefetchers.

Prefetchers	Stride (default)	GHB	BO
IPEX	8.96%	8.83%	8.76%

6.7.3 Prefetch Buffer Sizes. To evaluate the impact of prefetch buffer size on IPEX, we analyze three buffer sizes: 32B (2-entry), 64B (4-entry), and 128B (8-entry), as illustrated in Figure **??**. Across all sizes, IPEX demonstrates stable performance improvements over the baseline. With a larger prefetch buffer (e.g., 64B and 128B), IPEX achieves higher performance gains. The reason is that larger buffers

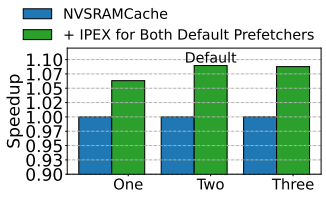


Figure 16: Threshold counts.

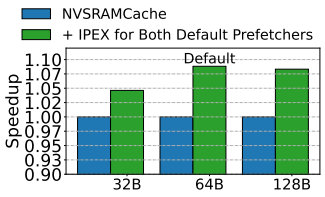


Figure 17: Prefetch buffers.

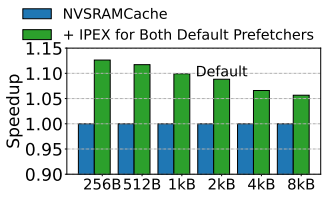


Figure 18: Cache sizes.

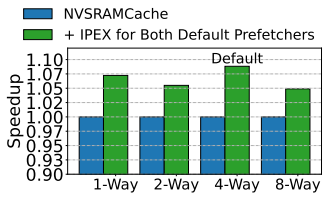


Figure 19: Cache associativity.

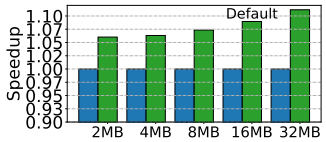


Figure 20: Main memory sizes.

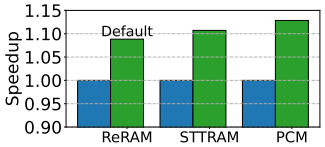


Figure 21: NVM technologies.

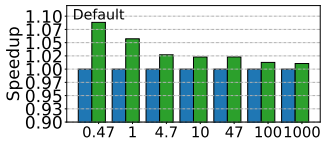


Figure 22: Capacitor sizes (μF).

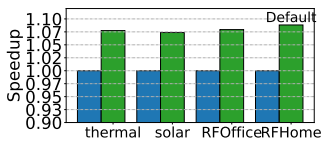


Figure 23: Power traces.

allow the prefetchers to aggressively load more blocks from main memory, increasing the likelihood of identifying and avoiding useless prefetches. This enables IPEX to effectively reduce energy waste while maintaining high performance. Note that IPEX's speedup saturates beyond 64B buffer as larger prefetch buffers are often under-utilized due to frequent power failure. Because of this, IPEX sets the default prefetch buffer size to 64B.

6.7.4 Cache Sizes. To evaluate the impact of cache size on IPEX, we vary cache size from 256B to 8kB. As shown in Figure **??**, IPEX consistently improves the performance of the baseline across all cache sizes. For smaller caches (e.g., 256B and 512B), IPEX achieves notable speedups (12.63% and 11.71%) by effectively reducing the energy waste caused by useless prefetches. Interestingly, IPEX's performance gains start to decline when the cache size is beyond 512B. This is because a larger cache can accommodate more cache blocks and thus mitigate the benefits of prefetching optimizations. Nonetheless, even with an 8kB cache, IPEX continues to deliver a measurable improvement (i.e., 5.66%).

6.7.5 Cache Associativity Sizes. We evaluate the impact of cache associativity on the performance of IPEX by varying the associativity from direct-mapped to 8-way caches. Figure **??** shows that for different cache associativities, IPEX demonstrates consistent improvement ranging from 4.89% to 8.96%. These numbers confirm IPEX's adaptability and efficiency in various cache configurations.

6.7.6 Main Memory Sizes. Memory size is another important factor impacting the effectiveness of IPEX as larger memories typically have higher access latencies and per-access energy consumption [**? ? ?**]. These characteristics amplify the benefits of IPEX in that it can save more energy waste on more expensive, useless prefetch operations (memory accesses). To demonstrate this, we evaluate IPEX with different main memory sizes. As shown in Figure **??**, IPEX achieves a higher speedup as the main memory gets enlarged, e.g., the speedup increases from 6.02% to 11.17% when the memory size is increased from 2MB to 32MB.

6.7.7 NVM Technologies. As various nonvolatile memory techniques exhibit disparate access latencies and per-access energy consumption, they might affect IPEX's performance. To show such an impact, we evaluate IPEX for three NVM technologies, e.g., ReRAM, STTRAM, and PCM. Figure **??** shows that IPEX achieves a higher speedup for a slower NVM technology, e.g., 12.84% for PCM while 8.96% for ReRAM. This confirms our conclusion made in Section **??**.

6.7.8 Capacitor Sizes. Capacitor size directly influences the frequency and duration of power interruptions in EHSs, which in turn affects the overall performance, particularly with weak energy sources like RF. Larger capacitors can store more energy, reducing the frequency of power outages and allowing for longer power cycles. However, larger capacitors also require longer charging time, i.e., slow reboot, and cause higher leakage current. Conversely, smaller capacitors have shorter charging time and lower leakage power, though they drain quickly and thus lead to more frequent power outages.

To assess the influence of capacitor sizes on IPEX's performance, we conduct a comprehensive sensitivity analysis. The results, depicted in Figure **??**, highlight that IPEX improves the performance of the baseline NVSRAMCache regardless of capacitor sizes. However, the speedup of IPEX diminishes as the capacitor gets enlarged from 0.47 μF to 1000 μF. This trend aligns with our expectations, as larger capacitors store more energy, enabling longer power cycles and consequently reducing the frequency of power interruptions. This eventually reduces opportunities for IPEX to throttle useless prefetches.

6.7.9 Power Traces. We evaluate IPEX with four power traces: thermal, solar, RFOffice, and RFHome. As Figure **??** shows, IPEX consistently improves the performance of NVSRAMCache regardless of energy conditions. Here, it is expected to see that IPEX has fewer opportunities to throttle prefetching and thus results in lower speedups with power traces with a higher portion of stable energy (e.g., solar and thermal). However, we notice that the performance gap between different traces is very small (e.g., the gap is only 1.14% between RFHome and thermal). There are two reasons for that. First, while solar and thermal traces provide a relatively higher proportion of stable energy supply compared to the other two, both solar and thermal traces still include a significant portion of poor energy, leading to frequent power outages in the EHS. Second, in our evaluation, energy availability is not the primary factor influencing IPEX's performance. Due to the small capacitor size (0.47 μF), even with a higher proportion of stable energy, the EHS still experiences frequent power outages. Combining these two reasons, the performance gap across different energy conditions remains small.

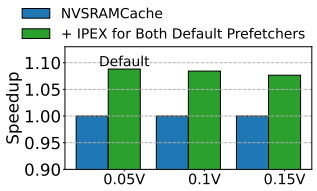


Figure 24: Voltage step size.

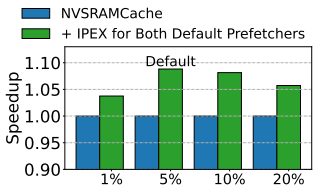


Figure 25: Throttle rates.

6.7.10 Voltage Steps. As described in Section **??**, IPEX adaptively adjusts its voltage thresholds using an empirically selected step size of 0.05 V. For sensitivity analysis on the voltage step, we test IPEX with different step sizes, ranging from 0.05 V to 0.15 V. As shown in Figure **??**, IPEX achieves the best performance with a 0.05 V step size. Larger step sizes can lead to overly aggressive threshold adjustments, resulting in either eager throttling or lazy throttling too much and thus degrading performance.

6.7.11 Throttle Rates. In Section **??**, we empirically set the throttle rate threshold for triggering the prefetch degree adjustment to 5%. To test the threshold impact on the performance of IPEX, we finally evaluate it by varying the throttle rate. The results, shown in Figure **??**, indicate that both low (1%) and high (20%) throttle rates degrade performance, as they lead to excessive eager throttling or lazy throttling, preventing IPEX from effectively balancing energy efficiency and prefetching effectiveness.

7 Limitations and Potential of IPEX

As discussed in Sections **??** and **??**, the efficiency of IPEX decreases when used with large capacitors or under consistently stable energy harvesting conditions. This is because both scenarios result in extended power cycles with fewer power interruptions, reducing IPEX's opportunities to throttle useless prefetches. However, it is important to note that these scenarios are not typical in most EHSs. For example, with typical RF-based energy harvesting [**? ? ? ? ? ? ? ? ? ? ?**], EHSs frequently experience power outages due to the unstable nature of RF sources. Additionally, the EHSs are often designed to be compact with minimal hardware costs. That is, exploiting small capacitors, which occupy less area, is a more suitable choice for the EHSs [**?**]. All these realities highlight the importance of solutions like IPEX that is devised to optimize energy efficiency in such challenging environments with scarce energy sources. The takeaway is that although the performance gains of IPEX diminish in those scenarios with long power cycles, it consistently delivers significant performance improvements in more common and challenging environments where typical EHSs are deployed, as shown in Section **??**.

Beyond the promising performance, IPEX offers broad applicability in two ways: (1) it can be easily integrated into a wide range of EHS architectures and (2) maintains high performance across diverse workloads, including those with or without peripheral communication.

Application to diverse EHSs: IPEX remains useful across a wide range of EHS architectures [**? ? ? ? ? ? ? ? ?**], no matter what crash consistency mechanism is used. The reason is that these EHSs share a common feature, i.e., all volatile cache blocks are lost upon power failure, rendering prefetched blocks useless if they do not receive any hits before the failure. For this reason, IPEX is highly promising to improve the performance of any kind of EHS by reducing its useless prefetching operations and saving energy for making forward progress.

Application to Workloads with and without Peripherals: Section **??** shows that IPEX's energy-saving capability brings good performance for workload without peripheral communication, i.e., application kernels only. As for workloads with peripherals, e.g., sensors and accelerators, IPEX can deliver even better performance because of a new challenge brought by peripherals. This challenge stems from the need to maintain fresh data out of peripherals. This means that peripheral operations and their dependent computations must be failure-atomic; the EHS groups them into atomic regions with checkpoints inserted at the entries of the regions [**? ? ? ? ?**]. These region-level checkpoints introduce additional energy overhead and increase the frequency of power outages, thereby giving IPEX more opportunities to identify and suppress useless prefetch operations before outages.

Besides, IPEX can help lower the risk of power failure within atomic regions and the associated re-execution penalty. The reason is that IPEX reduces energy waste on useless prefetches, thereby extending the durations of power cycles for the EHS. Hence, this can increase the likelihood of atomic regions being completed successfully within a single power cycle, which not only ensures the atomicity guarantee but also improves the overall performance. Furthermore, IPEX's energy-saving nature paves the way for advancements of EHSs. The significant energy efficiency improvement achieved by IPEX enables more complex and larger applications to run on EHSs, which would otherwise be unfeasible due to the high energy demand. We believe that this progress achieved by IPEX brings us closer than ever to realizing the full potential of intermittent computing.

8 Related Work

8.1 Prefetching for Non-Intermittent Systems

Instruction Prefetching: An instruction prefetcher reduces pipeline stalls by predicting and loading instructions into the ICache before they are needed, trying to make their later accesses hit in the cache. To achieve this, the prefetcher analyzes program behaviors such as control flow and instruction execution patterns. IBM introduced the first instruction prefetcher, next-line prefetching, in its System 360 Model 91 [**?**], which preloads the next sequential instruction cacheline(s) to reduce fetch delays in straight-line code execution.

Further advancements tackled prefetching beyond sequential execution, addressing branches, procedure calls, and system traps. As an example, Markov Prefetcher [**?**] employs a probabilistic model, i.e., Markov chains [**?**], to predict future memory accesses based on past access patterns. Specifically, the Markov prefetcher maintains a correlation table, the entry of which links a source address to a list of likely next addresses along with their corresponding probability of being subsequently accessed. When an instruction is fetched, the Markov prefetcher consults this table to predict and fetch the most probable subsequent instruction to the cache.

Unlike the Markov prefetcher, some prior work does not resort to probabilistic modeling. For instance, Temporal Instruction Fetch Streaming (TIFS) [**?**] improves cache hit rates by exploiting temporal repetition in instruction streams. That is, TIFS logs recurring

sequences of instruction cache misses in an Instruction Miss Log (IML) and leverages this history for prefetching. On a cache miss, TIFS first searches its prefetch buffer—that keeps prefetched blocks to avoid cache pollution—for a matching cache block. If it is found in the buffer, TIFS forwards the block to the ICache and prefetches the next block in the current temporal stream. If the search fails, TIFS consults the IML to prefetch missing instructions.

Data Prefetching: While instruction prefetchers exploit sequential or recurring patterns in instruction streams to reduce the frontend pipeline stalls, data prefetchers handle irregular and dynamic data access patterns. Among existing data prefetching schemes, stride prefetching [**?**] is the most popular technique due to its simplicity and effectiveness. The stride prefetcher predicts future memory accesses based on an observed stride between consecutive addresses (e.g., accessing addresses A, $A+4$, $A+8$, etc.) and prefetches the next block(s) accordingly with the stride in mind.

To deal with more complex access patterns, researchers propose multiple advanced prefetching schemes to further reduce cache misses. One example is Global History Buffer (GHB) prefetching [**?**] that captures recent memory access patterns and uses them as a basis for prefetching. To realize that, the GHP prefetcher tracks recent L2 cache misses using a circular buffer and groups those misses generated by the same instruction into a linked list. When a memory instruction misses in the cache, the GHP prefetcher first looks up the buffer with the instruction's PC as a key to figure out the resulting prefetch candidates on the list and then issues their prefetch requests for the corresponding cache blocks.

Rather than using such a heavyweight history table, i.e., GHB, Access Map Pattern Matching (AMPM) Prefetching [**?**] employs tiny bitmaps that can capture repetitive memory-access patterns for prefetching. To achieve this design, memory space is partitioned into a series of fixed-size regions, and each region maintains its own bitmap to record those cache blocks accessing the region. On a cache miss, AMPM consults the bitmap of the region being accessed, analyzes past access patterns, and determines which blocks are likely to be accessed in the future so that it can issue prefetches accordingly. With the help of such compact bitmaps, AMPM delivers high prefetch coverage with minimal hardware overhead.

Summary: The above instruction and data prefetchers are designed for non-intermittent systems and do not account for the frequent power outages that are a norm in EHSs. Therefore, applying them directly in the intermittent systems can lead to suboptimal performance or even performance degradation due to a large number of useless prefetch operations—generated especially when power failure is approaching. Consequently, this underscores the value of IPEX, which throttles the useless prefetches to prevent their energy waste and thus makes the existing prefetchers beneficial for EHSs.

8.2 Toward Unified Prefetching for both Non-Intermittent and Intermittent Systems

IPEX can be extended to non-intermittent systems by exploring the common challenge of prefetch timeliness in non-intermittent and intermittent systems. In a sense, these systems both contain some factors that negatively affect the timeliness obviating the prefetched cache blocks before their use, in which case the prefetching effort becomes wasted.

In non-intermittent systems, scheduling factors like thread migration between cores can make prefetched data useless. When the system reschedules a thread to a different core, the previously prefetched blocks in the original core may be left unused [**? ? ?**]. Another example is cache sharing in a Simultaneous Multithreading (SMT) processor, where one thread's prefetched data can evict another thread's prefetched block before it is used [**? ? ?**]. Also, dynamically resizing a cache to a smaller capacity can invalidate prefetched blocks before their use, making the prefetching effort in vain [**?**]. To mitigate such useless prefetches, the system must throttle its prefetching requests on the fly depending on the manifestation of the negative factors.

On the other hand, in intermittent systems, power failure acts as an external factor that erases prefetched cache blocks, rendering the prefetching worthless unless they are accessed before the failure. Unlike the aforementioned negative factors in non-intermittent systems, power failure occurs very frequently in intermittent systems, leading to a lot more useless prefetch operations. Despite the difference between these systems, their fundamental challenge of prefetch timeliness remains the same. That is, both non-intermittent and intermittent systems should avoid useless prefetches that negatively affect performance in any case.

To address this common challenge, we could extend IPEX to a unified prefetching framework so that it can evaluate prefetch timeliness and adjust the prefetch degree of the underlying prefetchers accordingly. This framework would rely on a new metric for quantifying the likelihood that a prefetched block remains available in the cache before use. In non-intermittent systems, this metric is influenced by multiple factors as mentioned before, whereas in intermittent systems, it is affected by input energy availability. This is why IPEX leverages the capacitor voltage as a proxy of the energy availability for adjusting prefetch degree dynamically to avert useless prefetches in EHSs.

9 Conclusion

This paper introduces IPEX that aims to tailor conventional prefetchers for energy harvesting systems by taking into account their frequent power failure. Depending on the likelihood of power failure, IPEX adjusts the prefetch degree of the underlying prefetchers accordingly to strike a balance between energy saving and high performance. Experimental results demonstrate that IPEX achieves an average of 7.86% (up to 21.64%) reduction in energy consumption, which translates to an average of 8.96% (up to 23.49%) performance gain compared to the baseline equipped with conventional ICache and DCache prefetchers. We believe that IPEX can lay the foundation for high-performance and energy-efficient prefetching in intermittent computing systems.

Acknowledgments

We appreciate anonymous reviewers for their invaluable comments as well as Purdue CompArch members for their constructive feedback. This work is in part supported by NSF grants 2001124, 2153749, and 2314681.