Hybrid Power Failure Recovery for Intermittent Computing

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

Jongouk Choi
University of Central Florida
Orlando, FL, USA
jongouk.choi@ucf.edu

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

Abstract

Energy harvesting systems rely on either rollback or roll-forward recovery to resume power-interrupted program correctly. However, both recovery schemes have their own inherent drawbacks. To this end, this paper presents RollSwitch, a hybrid power failure recovery scheme that can achieve low-cost yet high-performance intermittent computation for energy harvesting systems. According to the underlying energy harvesting condition, RollSwitch dynamically switches between rollback and roll-forward recovery modes to maximize the performance. In particular, RollSwitch leverages the level of available energy in the capacitor as a proxy for determining the appropriate recovery mode. For this purpose, RollSwitch devises a simple capacitor energy predictor whose outcome governs the recovery mode selection in the near future. The experimental results demonstrate that RollSwitch achieves 15.0% and 19.8% average performance gains over the state-of-the-art rollback and roll-forward recovery schemes, respectively.

ACM Reference Format:
Gan Fang, Jongouk Choi, and Changhee Jung. 2024. Hybrid Power Failure Recovery for Intermittent Computing. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD '24), October 27–31, 2024, New York, NY, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3676536.3676654

1 Introduction

Energy harvesting systems [1, 38] are becoming a promising alternative to battery-powered embedded systems. The lack of batteries eliminates the need for maintenance and aligns with eco-friendly practices, attracting many application domains—e.g., tiny IoT devices and wearables[35, 49]—to build on energy harvesting systems. Instead, they collect necessary energy from ambient sources such as radio-frequency (RF) and solar, using a tiny capacitor as energy storage. This implies that energy harvesting systems can only sustain as long as their capacitor allows; when it depletes leading to power failure, they lose all volatile data (i.e., registers) and enter hibernation to allow for recharging the capacitor. This is so-called *intermittent computation* [34]. Since the ambient sources are unstable, energy harvesting systems often suffer unpredictable power failure. Thus, commodity systems use nonvolatile memory (NVM) as main memory without cache and require crash consistency support for correct recovery across power failure [4–7, 9, 10, 14, 17, 33, 36, 37, 40, 41, 47, 50, 56].

ICCAD '24, October 27–31, 2024, New York, NY, USA

ACM ISBN 979-8-4007-1077-3/24/10
https://doi.org/10.1145/3676536.3676654

In the literature, roll-forward recovery has gained significant popularity; it persists registers before imminent power failure—thus being called just-in-time (JIT) checkpointing—and restores them in the wake of the failure [17, 33, 41, 42]. All roll-forward recovery schemes incorporate a voltage monitor to trigger the checkpoint/restore operation in a timely manner. When the voltage decreases below a checkpoint threshold, which is taken as a sign of impending power failure, they checkpoint all registers including a program counter (PC); later on, if the voltage exceeds a restore threshold, the monitor wakes up the system so that it recommences from the PC after retrieving the registers from their checkpoint. The benefit of roll-forward recovery is that it can resume power-interrupted program from the interruption point, thereby ensuring forward progress—even in the presence of frequent power failure. It is particularly advantageous when the quality of energy harvesting is good and steady; that is because while power is on, there is no need to checkpoint along the way.

However, roll-forward recovery requires that the JIT checkpointing of the entire register file should not be power-interrupted; otherwise, neither crash consistency nor forward progress is ensured. To render the JIT checkpointing failure-atomic, it is essential for the threshold voltage to be higher than what is actually required for persisting all registers [2, 17, 33, 41]. The reason is that frequent charging and discharging can diminish the capacitance gradually [3, 12, 44], jeopardizing the failure atomicity. Also, the voltage monitor may be inaccurate due to electrical noise, e.g., spontaneous current fluctuation on environmental change and prolonged circuit delay during the detection [16, 39, 41].

The crux of the problem is that since power failure can occur unexpectedly anytime, the safe voltage margin must always be secured thus preventing the full utilization of harvested energy for program execution. Moreover, to protect against unpredictable power failure, roll-forward recovery schemes incur considerable energy expenditure on voltage monitoring itself, necessitating the continuous operations of the ADC and comparator. As a result, in scenarios of poor energy harvesting, a substantial portion of the hard-won energy is squandered on the monitoring with possibly little energy left for meaningful forward execution progress.

With that in mind, researchers explore various rollback recovery mechanisms [9, 14, 26, 45, 48] as a substitute to roll-forward recovery. Instead of the JIT checkpointing, they exploit some form of periodic checkpointing so that any power failure occurred in a checkpoint interval (i.e., period in between checkpoints) can be recovered by restarting the interval with the checkpointed registers restored. Rollback recovery schemes can be more energy-efficient than their alternative roll-forward recovery, as they do not require securing the necessary energy or keeping the voltage monitor active always. Therefore, rollback recovery schemes allow more harvested

energy for program execution compared to roll-forward recovery—particularly in unstable energy.

Nevertheless, since rollback recovery schemes perform periodic checkpointing regardless of the quality of energy harvesting, they can end up wasting energy with redundant checkpoints. For example, while a sufficient amount of energy is harvested, power failure rarely occurs, making all checkpoints useless, except for the last one to be used for the recovery of some future failure if it occurs.

As such, rollback and roll-forward recovery schemes only work well in their own world. Due to the variation of the energy harvesting quality, they inevitably encounter unfriendly and pathological cases, losing the performance gain once obtained or causing the degradation significantly. Consequently, there is a compelling need for a practical solution that can maintain high performance and energy efficiency across varying energy harvesting conditions.

In response, this paper presents RollSwitch, a hybrid power failure recovery scheme that can achieve low-cost yet high-performance intermittent computation for energy harvesting systems. At each time quantum, RollSwitch dynamically switches between rollback and roll-forward recovery modes, taking into account the underlying energy harvesting condition. The key question for selecting the best recovery mode is how to precisely capture the varying conditions of energy harvesting across time quanta. One might think of predicting whether power failure will occur or not, e.g., to pick roll-forward recovery for the next quantum when the failure is unlikely therein. Unfortunately, power failure is quite unpredictable, which is a norm in energy harvesting systems due to the volatility of their ambient energy source.

RollSwitch instead leverages the level of available energy in the capacitor as a proxy of the energy source condition for determining the appropriate recovery mode. For this purpose, RollSwitch introduces a simple capacitor energy predictor whose outcome governs the recovery mode selection for each time quantum. The experiment with 23 applications from Mibench and Mediabench [13, 23, 32] shows that for various input energy conditions, RollSwitch achieves average speedups of 15.0% and 19.8% over the state-of-the-art roll-forward and rollback schemes, respectively. The contributions of this paper can be summarized as follows:

- RollSwitch is the first work to leverage both roll-forward and rollback recovery schemes for intermittent computing.
- RollSwitch devises an energy predictor to predict the external energy conditions for the first time.
- RollSwitch offers seamless and fast recovery mode switching without compromising crash consistency guarantee.
- RollSwitch adapts the size of time quantum on the fly to maximize the performance of its hybrid recovery.
- RollSwitch achieves significant speedups over the state-of-the-art rollback and roll-forward schemes.

2 Recovery Scheme Selection

To realize the full potential of the hybrid power failure recovery, it is critical to identify and integrate suitable roll-forward and rollback recovery schemes. Hence, this paper establishes two guiding principles for selecting such recovery schemes: (1) *Low run-time cost*: they should pursue high-performance design by avoiding costly checkpoints and associated expenses. (2) *Low hardware overhead*: the design should incorporate as few additional hardware components as possible to keep the energy and manufacturing costs minimal.

Adhering to these principles, this work adopts Nonvolatile Processor (NVP) [17, 33, 41, 42] as the roll-forward recovery scheme because of its fast and energy-efficient JIT checkpointing compared to other roll-forward schemes [2, 17]. The upshot is that NVP equips each register bit with a nonvolatile flip-flop (NVFF) and thus enables parallel checkpoint (restoration) of each flip-flop directly into (from) NVFF. In contrast, other roll-forward recovery schemes such as Hibernus and QuickRecall [2, 17] use NVM as the checkpoint storage, making their checkpoint and restoration demand more time and power compared to NVP's. Table 1 shows that the checkpointing time of NVP is 712500x (2045x) faster than that of Hibernus[2] (QuickRecall [17]); NVP also reduces the checkpointing energy consumption of Hibernus[2] (QuickRecall [17]) by 3810x (4x). Consequently, NVP only needs to secure significantly less energy in the capacitor for the JIT checkpointing, thereby leaving much more energy for forward progress compared to Hibernus[2] and QuickRecall [17].

Table 1: Comparison among roll-forward schemes.

	Hibernus[2]	QuickRecall[17]	NVP[42]
$T_{ckpt}(ns)$	2850000	8180	4
$T_{rst}(ns)$	2200000	4400	120
$E_{ckpt}(nJ)$	5716	6	1.5
$E_{rst}(nJ)$	4582	3	0.08

Another example of roll-forward recovery is Rockclimb [8]. It leverages compiler instrumentation to statically divide program into a series of failure-atomic regions. To be specific, Rockclimb ensures that each region can be completed with a fully charged capacitor whose capacitance is taken into account to determine the region boundary. Upon reaching each boundary, Rockclimb stalls program execution to recharge the capacitor—unless it is fully charged—so that the next region starts with the full capacitance. As a result, no regions are power-interrupted, i.e., Rockclimb achieves rollback-free intermittent computation [8] even without JIT checkpointing. Unfortunately, this is agnostic to the actual energy conditions, leading to significant slowdown. For example, even if the capacitor remains sufficiently charged at a region boundary, Rockclimb should check on the capacitance and secure it, thereby causing unnecessary delays.

Regarding rollback recovery, this work chooses TCCP [26] thanks to its simple architecture that achieves both low run-time (RT) and hardware (HW) overheads compared to other recent rollback schemes [14, 45] as shown in Table 2. For example, Ratchet [45] introduces the correct rollback recovery by identifying and eliminating the write-after-read (WAR) dependence. If WAR is not properly handled, its re-execution during rollback can lead to inconsistent state—since the original value has been overwritten. Ratchet employs its compiler to divide program into a series of idempotent regions [15, 18–20, 22, 25, 27–31, 46, 51–55, 57] so that their boundaries cut the WAR dependencies. At the beginning of such a region, its input registers (i.e., live-in registers) are checkpointed along with the PC that serves as the recovery point in case the region is power-interrupted. In this way, Ratchet can recover from power failure by

re-executing the power-interrupted idempotent region. However, program typically contains a large number of WAR dependencies [13, 23], which yields many small regions and their resulting numerous checkpoints; the WAR detection of Ratchet is imprecise due to the conservative compiler's pointer analysis, thereby generating many false WAR dependencies and further increasing the number of checkpoints. Hence, Ratchet can cause significant performance penalties in that checkpoints are essentially a nonvolatile memory store, i.e., the most time- and energy-consuming instruction in the processor.

While Ratchet uses compiler-based WAR detection at the expense of high run-time overhead, Clank [14] relies on hardware-based detection thus being free from the imprecise pointer analysis and the excessive checkpoints. Clank does not have to checkpoint volatile states, e.g., a register file (REG), every time a WAR dependence is detected. That is because Clanks keeps the data being written in a special hardware buffer until it overflows, which is helpful for decreasing the run-time overhead. Besides, Clank employs several other buffers to detect and resolve WAR dependencies so that it can correctly restore program states and resume from the most recent checkpoint across power failure. Although Clank protects NVM from the inconsistency issue of WAR dependencies during the re-execution, it comes with considerable hardware overhead. Clank's buffers and their associated logic complicates the hardware design significantly, increasing both die size and power consumption.

In particular, unlike other rollback schemes, TCCP [26] does not suffer from high run-time and hardware costs thanks to its simple and lightweight design. As a basis for crash consistency, TCCP treats all stores as speculative—though they have been retired from the processor pipeline—and quarantine them in a store buffer (SB) assuming that power failure is unlikely. Once TCCP progresses to the point where the SB becomes full, i.e., the speculation window successfully ends without power failure, TCCP persists both REG and SB—to the checkpoint storage, i.e., NVFFs and nonvolatile SB (NVSB)—with its stores made non-speculative as well as begins the next speculation window where they are to be written to NVM. If any window is power-interrupted, its stores quarantined in the SB all disappear (as SB is volatile), which eliminates the issue of potential WAR dependencies. Consequently, when power comes back, TCCP can correctly resume from the beginning of the interrupted window with the REG and the SB restored.

Table 2: Comparison among rollback schemes.

	Ratchet[45]	Clank[14]	TCCP[26]
Software	Yes	No	No
RT overhead	High	Low	Low
HW overhead	No	High	Low

3 Crash Consistency Guarantee

This paper picks NVP and TCCP for RollSwitch's roll-forward and rollback recovery modes, respectively. For seamless mode transition in between, RollSwitch allows the SB to be used by NVP, i.e., its JIT checkpoint persists both the REG and the SB as TCCP checkpoints them upon the SB overflow, and lets the two recovery modes share the same checkpoint storage, i.e., NVFFs and NVSB. Thus, no matter which recovery mode encounters power failure, RollSwitch takes

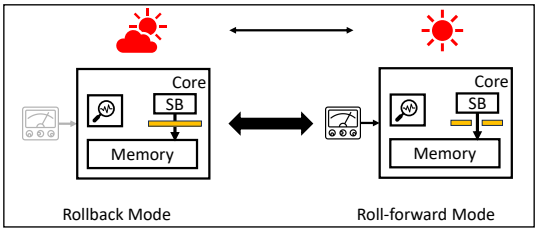

Figure 1: High-level overview of RollSwitch.

the same recovery process, i.e., the restoration of both REG and RF from the most recent checkpoint.

Nevertheless, combining NVP and TCCP does not automatically guarantee crash consistency across power failure, so RollSwitch still needs to ensure that power-interrupted program can be correctly resumed with its states restored irrespective of the time point of power failure. Given that both roll-forward and rollback recovery modes have their own mechanisms to guarantee crash consistency, the emphasis is placed on designing additional control logic for ensuring crash consistency during transitions between the two recovery modes. As shown in Figure 1, upon switching from rollback to roll-forward mode, RollSwitch activates both the ADC and the voltage monitor as well as deactivates the logic that quarantines retired stores, allowing the SB to asynchronously write them back to NVM with no quarantine. Conversely, when switching to rollback mode, RollSwitch suspends continuous voltage monitoring and reactivates the quarantine mechanism in the SB (Figure 1).

In fact, the potential crash inconsistency of RollSwitch is caused by inconsistent states of memory (not REG[1]), and they only occur when post-switch power failure occurs under the rollback recovery mode[2]. The crux of the problem is that the power failure may occur before the SB overflows, i.e., no checkpoint has been made since the switch to the rollback mode, and thus fail to recover from the failure. Figure 2 shows two possible cases of such incorrect power failure recovery, assuming that all memory states are initialized to 1 (i.e., $M1:1$ and $M2:1$). In Case 1, RollSwitch first encounters power failure in the roll-forward mode and thus makes a JIT checkpoint (*ckpt* in the figure) with the PC pointing to $R1 = M1$. After the restoration (*rst*) in the wake of the failure, $M1$ is updated to 2 persisting in NVM (i.e., $M1:2$), and then RollSwitch switches to the rollback mode. Another power failure happens before any post-switch checkpoint is made, in which case RollSwitch ends up restoring REG and SB from *ckpt*—made in the prior roll-forward mode—and thus resumes program from $R1 = M1$ pointed by the PC of restored REG. Unfortunately, this faces the inconsistency issue of WAR dependence, i.e., $M1$ has been overwritten to 2 ($M1:2$ in NVM) different from the original value ($M1:1$) when *ckpt* was made, resulting in incorrect recovery.

In Case 2, RollSwitch first makes a checkpoint (*ckpt*) with the PC pointing to $R1 = M2$ under the rollback recovery mode due to the SB overflow. Then, it switches to the roll-forward recovery mode and executes a few instructions along the way, including $M2 = R1 + 1$ that changes $M2$ to 2 in NVM. After that, it switches back to the

[1]Registers have no problem since they disappear on power failure.
[2]Under the roll-forward recovery mode, its JIT checkpointing can handle power failure as usual, regardless of switchings between the two recovery modes.

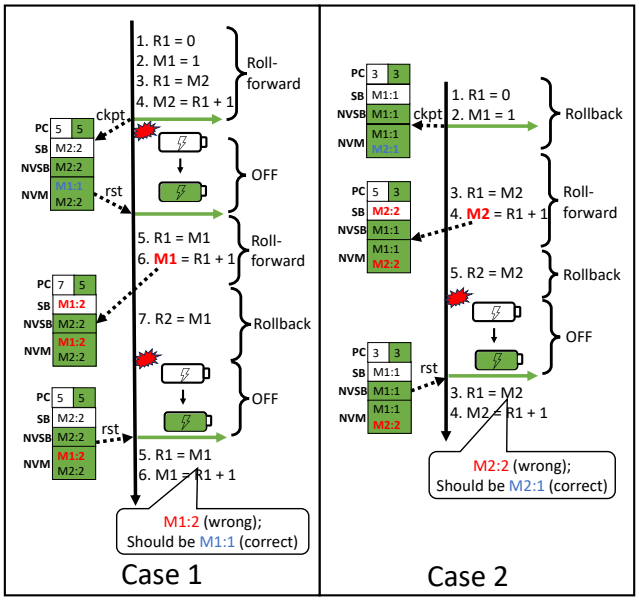

Figure 2: Possible inconsistency with mode changes; SB has 1 entry; white (green) box means volatile (nonvolatile) states.

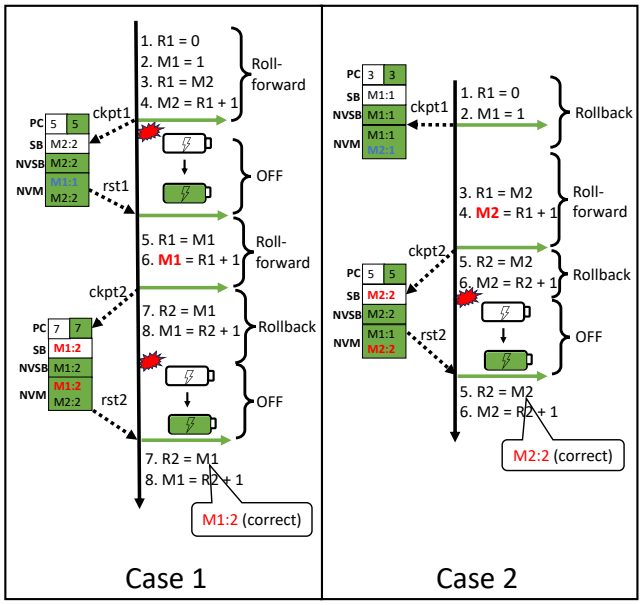

Figure 3: Crash consistency guarantee with checkpointing right after switching to the rollback recovery mode.

rollback mode. Unfortunately, power failure then happens before RollSwitch makes any checkpoint after the mode switching. Here, RollSwitch ends up restoring from *ckpt*—made in the prior rollback mode—and restarting from $R1 = M2$ pointed by the PC of restored REG. As a result, due to the WAR dependence on $M2$, $R1 = M2$ reads the overwritten value $M2 : 2$ from NVM, not $M2 : 1$ at the time of *ckpt*, leading to incorrect recovery.

To ensure crash consistency for Case 1 and 2, RollSwitch introduces an additional checkpoint as soon as it switches to the rollback recovery mode. This obviates the need for re-executing any pre-switch instructions—potentially involved in WAR dependence as with Case 1 ($M1 = R1 + 1$) and Case 2 ($M2 = R1 + 1$)—in the wake of power failure. That is because program should resume from where the additional checkpoint was made, i.e., the PC of the restored REG. Consequently, RollSwitch can correctly recover from power outages all the time regardless of the WAR issue.

Figure 3 shows how RollSwitch ensures crash consistency for Case 1 and 2 by making a checkpoint (*ckpt2*) right after switching to the rollback recovery mode. In Case 1 of the figure, for the recovery of the last power failure, RollSwitch resumes program from Instruction 7 ($R2 = M1$)—pointed by the PC restored from *ckpt2*—retrieving the correct value $M1 : 2$. In particular, $M1$ could have been overwritten in NVM by Instruction 8, leading to crash inconsistency. Fortunately, this never happens thanks to RollSwitch's SB quarantine mechanism under the rollback recovery mode, i.e., all written data held in the volatile SB are wiped out upon power failure. In Case 2 of Figure 3, similar steps follow for crash consistency guarantee. Here, program resumes from Instruction 5 ($R2 = M2$)—as it is pointed by the restored PC where the *ckpt2* (not *ckpt1*) was made—and retrieves the correct value $M2 : 2$. Note that Instruction 4 ($M2 = R1 + 1$) involved in the WAR dependence here cannot break the crash consistency, since the instruction is not to be re-executed. As a result, RollSwitch can achieve correct power failure recovery in Case 2 as well.

4 Timely Recovery Mode Switch

Besides ensuring correct power failure recovery across the mode switches, RollSwitch should figure out when to switch the recovery mode in order to maximize the performance. It would be ideal to change the recovery mode when the energy harvesting quality varies sufficiently enough to improve the performance with the other mode, e.g., switching from roll-forward to rollback mode as energy conditions worsen while doing vice versa as they become better.

In reality, achieving this is a daunting challenge, since the energy conditions are affected by various dynamic factors. For example, as energy harvesting systems are commonly utilized in portable IoT devices and wearables, the energy conditions can fluctuate with changes in location. The distance between these devices and their energy source can vary as they are moved, directly impacting the harvesting efficiency; larger distances generally result in poorer energy harvesting. Also, physical obstacles between the energy source and the device can degrade the harvesting quality by choking the energy flow. These factors make it difficult to identify the energy conditions and select the most appropriate recovery mode.

To this end, RollSwitch instead takes advantage of the level of energy available in the capacitor as a proxy for such a hard-to-assess energy condition and makes appropriate recovery mode switches around the threshold level. In particular, RollSwitch introduces a simple energy predictor taking into account the unique characteristic of energy harvesting systems which uses their capacitor to buffer harvested energy. That is, the capacitor voltage provides immediate and real-time feedback on the available energy, accurately reflecting the current energy state of the energy harvesting

system. Thanks to this characteristic, the capacitor voltage serves as a reliable proxy for predicting energy conditions, facilitating the selection of a suitable power failure recovery mode. This simple prediction approach obviates the need for additional sensors or hardware features, thus easily integrating with the existing energy harvesting systems at a lost cost.

To realize the energy predictor, a threshold voltage (V_{thres}) should be selected appropriately in that it governs the resulting recovery mode selection. That is, the predictor anticipates good energy conditions when the capacitor voltage V_c gets higher than V_{thres}, in which case RollSwitch switches to the roll-forward recovery mode; if V_c becomes lower than V_{thres}, it is likely that the energy condition would be poor, and thus RollSwitch switches to the rollback recovery mode.

Note that for the appropriate threshold voltage determination, RollSwitch should consider not just the performance but also the crash consistency. To maintain high performance, RollSwitch should stay in the roll-forward mode as long as possible if the energy condition is good. The reason is that roll-forward recovery can minimize run-time overhead by avoiding periodic checkpoints (typically required in rollback recovery). With that in mind, RollSwitch prefers a low V_{thres}.

However, it should not too low to jeopardize the crash consistency. For crash consistency guarantee, it is crucial to keep the capacitor voltage V_c above the checkpoint threshold V_{ckpt} with a margin for covering the energy consumed by the energy prediction and recovery mode switching. Otherwise, there is a risk that V_c dips below V_{ckpt}, potentially failing to complete the next checkpoint after the recovery mode switching, i.e., the post-switch checkpoint could be power-interrupted, in which case crash inconsistency occurs (Section 3). Consequently, to set up a right V_{thres}, RollSwitch leverages Equation 1 below.

$$\frac{1}{2}C_{bulk}(V_{thres}^2 - V_{ckpt}^2) > E_{predict} + E_{switch} \quad (1)$$

where C_{bulk} represents the capacitor size. The left-hand side of Equation 1 corresponds to the energy available to program execution after the post-switch checkpoint, while the right-hand side to the combined energy consumption for the prediction ($E_{predict}$) and the mode switching (E_{switch}).

5 Optimizations

While Section 4 outlines a simple method for predicting energy conditions, it faces two key limitations that may negatively affect its efficiency. First, under RollSwitch's rollback mode, the continuous voltage monitoring—though it is unnecessary—leads to substantial energy expenditure. Second, the threshold voltage determined by Equation 1 is not always accurate, e.g., due to the inability to handle real-time fluctuations in energy conditions. For instance, once RollSwitch detects that the capacitor voltage (V_c) has dropped below the threshold (V_{thres}) and thus switches to the rollback recovery mode, the voltage might happen to go above V_{thres} and quickly get back provided the energy condition varies suddenly in both directions. This can cause RollSwitch to make suboptimal recovery mode decisions, possibly degrading the performance. Therefore, this paper proposes two optimizations to address the aforementioned limitations.

5.1 Optimization 1: Time Quantum and its Adaptation

The first optimization aims to decrease the energy consumption of the voltage monitor by avoiding its continuous activation in the rollback recovery mode. That is, every time RollSwitch switches to the rollback mode, it adopts a periodic approach that activates the voltage monitor only at the end of each time quantum for predictions. This method makes the selection of the time quantum length critical for ensuring timely energy predictions. A naive approach would be empirically selecting a fixed time quantum and using it across different energy conditions. This approach is easy to implement but may not align well with the varying durations of power cycles—caused by the fluctuation of real energy conditions—which could make the predefined time quantum inappropriate. For example, if a power cycle is shorter than the time quantum, RollSwitch cannot make any prediction, rendering the recovery mode switch ineffective. Conversely, an overly long power cycle can result in an excess of predictions, most of which might be redundant and fail to trigger recovery mode changes. These unnecessary predictions would increase run-time overhead and energy consumption without providing any benefits.

Therefore, the size of the time quantum must be adaptive to accommodate the variable durations of power cycles. This paper introduces an adaptive technique [21, 24] that dynamically adjusts the time quantum size based on the number of predictions and mode switches observed within each power cycle. RollSwitch utilizes two counters: one for tracking the number of predictions and the other for the number of mode switches. In the wake of power failure, RollSwitch recalibrates the time quantum based on the two counter values recorded in the previous power cycle. If no predictions were made, implying that the time quantum was too large, then RollSwitch halves the quantum for the current cycle. On the other hand, if the previous cycle had many predictions but only a few mode switches, indicating excessive non-contributory predictions, then RollSwitch—in the wake of power failure—increases the time quantum by 50% to reduce unnecessary predictions and associated costs. In all other cases, RollSwitch maintains the current time quantum size.

5.2 Optimization 2: Energy Predictor Design

To improve the adaptability of RollSwitch to varying energy conditions, the second optimization enhances the energy predictor, addressing shortcomings not taken care of by the basic approach outlined in Section 4. For this purpose, RollSwitch employs historical voltage data to train its predictor. The intuition behind this idea is that combining the current voltage reading with the trained results, RollSwitch can be more accurate in energy predictions. This method integrates an N-bit saturation counter which is fed by detected voltage at each previous prediction point. If the voltage exceeds V_{thres}, the counter increases by one; it decreases by one otherwise. The recovery mode is then determined by the most significant bit (MSB) of the counter, i.e., '0' indicates the need for the rollback recovery mode, while '1' indicates the roll-forward mode. This approach enables RollSwitch to dynamically adjust its predictions in response to fluctuating energy conditions, thereby

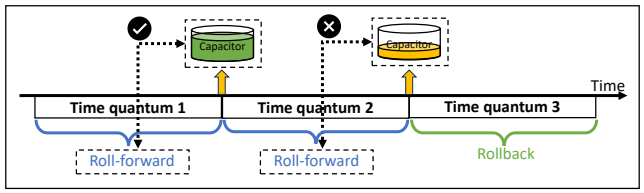

Figure 4: Energy prediction examples.

significantly improving its decision-making for appropriate recovery mode selection.

6 Energy Condition Prediction Accuracy and its Evaluation

To assess the effectiveness of the above two optimizations, this paper leverages prediction accuracy as a metric—that can tell eventually if RollSwitch's simple energy predictor can work well. The high-level idea of accuracy calculation is to check whether the voltage reading at the end of each time quantum matches the prediction result. A match indicates a correct prediction, whereas a mismatch is marked as incorrect. Figure 4 illustrates two scenarios of the prediction accuracy evaluation. In *Time quantum 1*, RollSwitch predicts energy sufficient and opts for the roll-forward recovery mode. At the end of this time quantum, the capacitor still has sufficient energy, confirming that the energy conditions are good, and thus this prediction is correct. In the next time quantum (*Time quantum 2*), the predictor still anticipates enough energy, but it turns out that the energy conditions are poor, rendering this prediction wrong. Figure 7 shows RollSwitch achieves a notably high average prediction accuracy, i.e., 93%, across different energy conditions. This confirms the effectiveness of RollSwitch's energy predictor and demonstrates that it can switch to the suitable recovery mode accordingly.

7 Discussion

This section explores two additional factors that can impact RollSwitch's overall performance and energy efficiency, i.e., non-contributory predictions and misprediction penalties.

7.1 Non-contributory Prediction

Non-contributory prediction is defined as energy predictions that do not lead to a recovery mode change. This occurs if a series of predictions are made when the detected voltage consistently remains above or below the predefined threshold (V_{thres}). The drawback of these non-contributory predictions is that they incur extra run-time overhead by stalling the core pipeline to read the current voltage level, update the saturation counter accordingly, and finally assess the need to make a recovery mode change.

While contributory predictions carry similar drawbacks—increased runtime overhead and energy usage—these are often justified by the benefits derived from the resulting recovery mode switching. In other words, if RollSwitch triggers a recovery mode change, the energy or time saved could outweigh the energy spent on making the prediction. To the contrary, non-contributory predictions, where the recovery mode remains unchanged, merely add to the system's overhead without improving its efficiency or performance. The crux of the problem is that these redundant predictions drain valuable energy resources, particularly critical in energy harvesting systems where efficiency is paramount. Recall that RollSwitch significantly mitigates the problem by leveraging an adaptive time quantum resizing strategy—mentioned in Section 5.2—which is specifically designed to decrease non-contributory predictions and thus able to enhance the performance and energy efficiency of the target energy harvesting system.

Table 3: Simulation configuration

	NVP [41]	TCCP [26]	RollSwitch
MCU Power	160 $\mu W/MHz$	160 $\mu W/MHz$	160 $\mu W/MHz$
V_{max}/V_{min}	3.3/2.8	3.3/2.8	3.3/2.8
$V_{bk}/V_{restore}$	2.9/3.1	2.9/3.1	2.9/3.1
SB size	8	8	8
Capacitance	10 μF	10 μF	10 μF
NVM	ReRAM	ReRAM	ReRAM
NVM size	16MB	16MB	16MB
NVM config.	tCK/tBURST/tRCD/tCL/tWTR/tWR/tXAW = 0.94/7.5/18.0/15.0/7.5/150/30		

7.2 Misprediction Penalty

Misprediction happens when the energy condition turns out to be opposite to the prediction result, resulting in two possible scenarios with associated penalties. First, if RollSwitch incorrectly chooses a roll-forward mode under poor energy conditions, it wastes hard-won energy by unnecessarily keeping the voltage monitor active all the time for the failure-atomic JIT checkpointing required by the roll-forward recovery In this scenario, the misprediction results in wasting energy significantly that could have otherwise be contributed to the forward program execution progress. Second, if RollSwitch happens to select a rollback mode under good energy conditions, it incurs extra latency and the energy consumption in the mean time by making unneeded checkpoints periodically. In summary both mispredction scenarios either wastes the energy—painfully harvested in general due to the volatility of energy sources—or increases unnecessary run-time overhead, which ultimately results in the degradation of performance and efficiency. Last but not least, as highlighted in Section 7.1, the prediction itself takes both time and power that could exacerbate the system's energy inefficiency, so RollSwitch should minimize the chance of mispredictions. Fortunately, as shown in Section 6, it turns out that RollSwitch can achieve quite a high prediction accuracy consistently across various energy conditions, thereby keeping the misprediction penalties low and achieving high performance.

8 Evaluation

8.1 Methodology

We conducted extensive simulations of RollSwitch along with other schemes using the gem5 simulator configured for the ARM ISA. The simulations were performed on a single-core in-order NVP, specifically the NVPsim [11], operating at a 25 MHz clock frequency. NVPsim is not equipped with caches and enables 16 volatile registers, each of which is attached with a nonvolatile counterpart for backup purposes. Additionally, we model the voltage monitor with its initialization, propagation delay, and energy consumption.

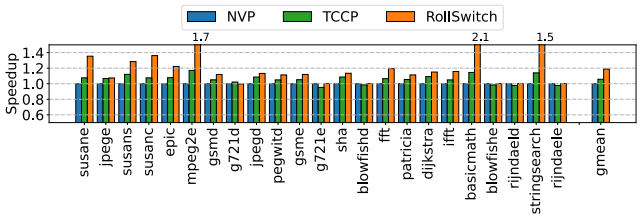

Figure 5: Performance on RFHome trace. The y-axis is the speedup normalized to NVP.

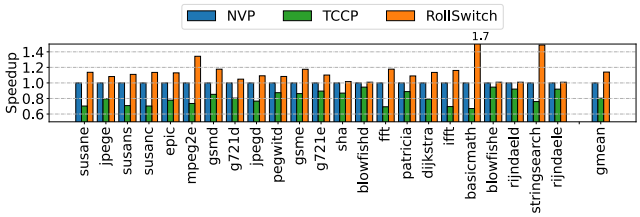

Figure 6: Performance on thermal trace. The y-axis is the speedup normalized to NVP.

The simulation configurations are outlined in Table 3. For all schemes included in the table, we adhere to the configurations detailed in their original publications, provided they are compatible with the settings of RollSwitch. Besides, we add an SB for NVP to allow simultaneous instruction execution and write-back of retired stores. This enhancement aims to improve the throughput and efficiency of NVP under varying workload conditions.

For the SB access, we employed a CAM-based associative search mechanism, evaluating its energy consumption and performance impact using CACTI [43] modeled on 90 *nm* technology [17, 33, 41]. As for benchmarks, we utilized a comprehensive suite of 23 applications from Mediabench and MiBench [13, 23, 32], covering a wide range of processing scenarios to ensure thorough testing of all schemes under diverse conditions. In our sensitivity analyses, we evaluate the performance of each scheme across multiple real-world energy harvesting traces [11], which provide a realistic context for understanding how systems perform under actual environmental energy variations. Additionally, we explored the effects of varying capacitor sizes and SB capacities. These modifications allowed us to observe how adjustments in hardware configurations influence system performance, thereby identifying the optimal setups for maximizing both efficiency and reliability in energy harvesting systems.

8.2 Hardware Cost Analysis

RollSwitch employs a simple 2-bit saturation counter as its energy predictor, coupled with minimal control logic to appropriately change the checkpointing scheme based on different recovery modes. Like TCCP, RollSwitch incorporates a gated store buffer which delays the release of retired stores. The difference is that RollSwitch employs additional control logic to enable and disable the delay logic. That is, this delay is activated only in rollback recovery mode, allowing for immediate store release in roll-forward recovery mode. Furthermore, during rollback mode, RollSwitch marks each SB entry with a flag bit to denote its non-speculative

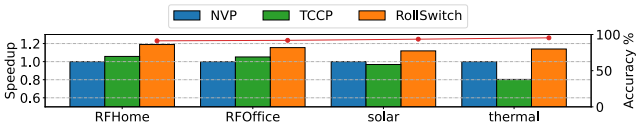

Figure 7: Performance on different energy traces.

status and also introduces an instruction counter as a threshold to ensure the system does not stagnate.

8.3 Performance

Figures 5 and 6 display performance comparisons among NVP, TCCP, and RollSwitch under two distinct energy conditions. On the RFHome trace, characterized by generally poor energy harvesting quality, TCCP outperforms NVP since it does not need to secure energy for JIT checkpointing and does not need to always maintain the voltage monitor active. Thus, TCCP has more energy used for forward progress. However, TCCP is not perfect since there are still some situations where the energy harvester collects sufficient energy supply, making its checkpointing redundant. Fortunately, RollSwitch takes into account varying energy conditions, allowing it to optimize its performance effectively. As a result, RollSwitch consistently achieves superior performance, averaging 18.9% faster than NVP and 12.4% faster than TCCP. Conversely, on the thermal trace, where energy conditions are typically good, TCCP results in high run-time overhead due to redundant checkpoints, making it significantly worse than NVP. For the same reason, RollSwitch continues to lead, outperforming NVP and TCCP by 13.9% and 42.4%, respectively.

8.4 Sensitivity Analysis of Energy Condition

Figure 7 demonstrates that RollSwitch consistently outperforms both NVP and TCCP across four different energy traces (15.0% and 19.8% faster respectively). It is not surprising to find that RollSwitch always outperforms in different energy harvesting qualities. The reason is that the varying quality of energy supply, which can fluctuate between good and poor, significantly impacts the performance of NVP and TCCP. For instance, on the RFHome and RFOffice traces, where energy conditions are generally poor, TCCP outperforms NVP due to its better efficiency. In contrast, on the solar and thermal traces, where energy conditions are generally good, TCCP is hampered by frequent redundant checkpoints, leading to worse performance compared to NVP.

8.5 Sensitivity Analysis on the Size of a Saturation Counter

The number of bits in the saturation counter directly influences the accuracy and responsiveness of energy predictions in RollSwitch. A saturation counter with an appropriate number of bits enables the system to effectively balance historical data and current trends, optimizing prediction accuracy and system performance in variable energy environments. Therefore, we present a sensitivity analysis regarding the use of different saturation counter bits in RollSwitch across various energy harvesting traces in Figure 8. The evaluation results reveal that a 2-bit saturation counter consistently provides

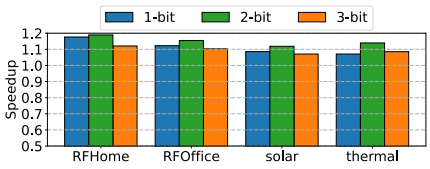

Figure 8: Performance with different counter bits.

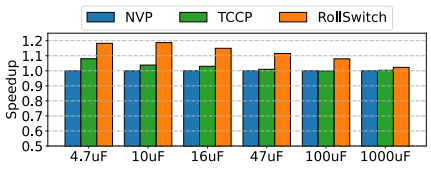

Figure 9: Capacitor size sensitivity.

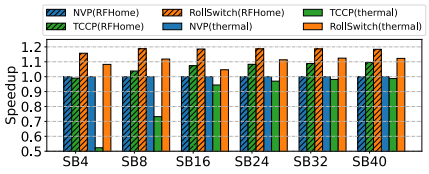

Figure 10: SB size sensitivity. We show the performance across two traces.

the best performance across four different energy conditions. Conversely, configurations with either a 1-bit or 3-bit counter experience a degradation in performance. The good performance of the 2-bit counter can be attributed to its balanced approach to integrating historical prediction data. Unlike the 1-bit counter, which lacks reliance on past prediction outcomes and may therefore easily make prediction mistakes, the 2-bit counter provides a moderate level of conservatism that aligns well with the dynamic nature of energy availability. On the other hand, the 3-bit counter's performance suffers because it is overly conservative, making it slow to adapt to changes in energy conditions. This excessive caution prevents timely adjustments to recovery modes in response to energy fluctuations, leading to suboptimal performance when rapid response is necessary.

8.6 Sensitivity Analysis on Capacitor Size

The capacitor size directly determines the booting time and the power failure number. For the small capacitors, the booting time is relatively short because only a small amount of collected energy can make the capacitor fully charged. However, at the same time, the small capacitor also makes the system susceptible to the ambient environment. When the ambient energy is scarce, the capacitor drains very fast, and then power failure occurs frequently. On the contrary, the large capacitors have longer booting time but make the system robust to the ambient energy sources (i.e., fewer power outages). We conduct the sensitivity analysis with different capacitors and show the results in Figure 9. RollSwitch always delivers the best performance over the others. It is interesting to find that with a large capacitor (e.g., 1000 μF), the performance gap among these three schemes is quite small (less than 2% difference). This outcome is not surprising, given that the booting time is quite long and constitutes the dominant portion of the total time.

8.7 Sensitivity Analysis of Store Buffer (SB) Size

The SB size is a critical factor that significantly influences the checkpoint granularity in TCCP and the rollback mode in RollSwitch. A larger SB size generally means fewer checkpoints are needed but potentially increasing the rollback penalty. Conversely, a smaller SB leads to more frequent checkpoints but reduces the latency associated with re-execution. To determine the optimal SB size, we conducted a sensitivity analysis and presented the results in Figure 10. On the RFHome trace, we observe that TCCP underperforms compared to NVP when using a very small SB (4-entry). This poor performance is primarily due to the high-frequent checkpointing. Meanwhile, under the thermal trace, TCCP shows much worse performance than NVP even with a larger SB size (up to 40 entries).

Table 4: Energy breakdown normalized to the NVP.

	Checkpoint	Voltage monitor	Total
NVP	0.069%	8.57%	100%
TCCP	7.60%	0.18%	87.31%
RollSwitch	2.56%	4.49%	70.86%

The stable energy supply in this scenario makes many of TCCP's checkpoints redundant, negatively impacting its performance. Fortunately, the performance of RollSwitch is less affected by variations in SB size due to its ability to adaptively switch between different recovery modes. This adaptability allows RollSwitch to mitigate the impact of SB size on its performance, maintaining efficiency across different energy conditions.

8.8 Energy Efficiency

Table 4 illustrates the energy consumption of NVP, TCCP, and RollSwitch across various energy traces. RollSwitch demonstrates substantially lower energy usage on average, consuming about 30% less energy than NVP and 18% less than TCCP. This efficiency is highlighted by RollSwitch's ability to save considerable amounts of energy in specific areas: it reduces energy expenditure on voltage monitoring by 58% compared to NVP. Additionally, when compared to TCCP, RollSwitch achieves a 66% reduction in energy used for checkpointing.

9 Conclusion

This paper introduces RollSwitch, a hybrid power recovery approach for energy harvesting systems. It integrates both rollback and roll-forward recovery modes and dynamically alternates between them based on the energy harvesting conditions to optimize performance and energy efficiency. To determine the appropriate recovery mode, RollSwitch leverages the available energy level in the underlying capacitor as a proxy for the energy harvesting quality. In particular, RollSwitch devises a simple capacitor energy predictor, the outcome of which directs the recovery mode selection in the near future. As a result, RollSwitch significantly outperforms both state-of-the-art rollback and roll-forward recovery schemes. We believe that RollSwitch paves the way towards more performant, adaptable, and sustainable intermittent computation.

Acknowledgments

We appreciate anonymous reviewers for their valuable comments and Purdue CompArch members for early discussion of this work with them. At Purdue, this work was supported by NSF grants 2001124 (CAREER), 2153749, and 2314681. At the University of Central Florida, this work was supported by NSF grant 2314680.

References

[1] Tosiron Adegbija, Anita Rogacs, Chandrakant Patel, and Ann Gordon-Ross. 2018. Microprocessor Optimizations for the Internet of Things: A Survey. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2018).

[2] Domenico Balsamo, Alex Weddell, Geoff Merrett, Bashir Al-Hashimi, Davide Brunelli, and Luca Benini. 2014. Hibernus: Sustaining computation during intermittent supply for energy-harvesting systems. In *Embedded Systems Letters*.

[3] Ramzi Chaari, Olivier Briat, and J-M Vinassa. 2014. Capacitance recovery analysis and modelling of supercapacitors during cycling ageing tests. *Energy conversion and management* (2014).

[4] Jongouk Choi, Jaeseok Choi, Hyunwoo Joe, and Changhee Jung. 2024. Caphammer: Exploiting Capacitor Vulnerability of Energy Harvesting Systems. In *ACM EMSOFT (2024) (EMSOFT '24)*.

[5] Jongouk Choi, Hyunwoo Joe, and Changhee Jung. 2022. CapOS: Capacitor Error Resilience for Energy Harvesting Systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2022).

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

[7] Jongouk Choi, Hyunwoo Joe, Yongjoo Kim, and Changhee Jung. 2019. Achieving Stagnation-Free Intermittent Computation with Boundary-Free Adaptive Execution. In *IEEE RTAS (2019)*.

[8] Jongouk Choi, Larry Kittinger, Qingrui Liu, and Changhee Jung. 2022. Compiler-Directed High-Performance Intermittent Computation with Power Failure Immunity. In *IEEE RTAS (2022)*.

[9] Jongouk Choi, Qingrui Liu, and Changhee Jung. 2019. CoSpec: Compiler Directed Speculative Intermittent Computation. In *IEEE/ACM MICRO (2019)*.

[10] Jongouk Choi, Jianping Zeng, Dongyoon Lee, Changwoo Min, and Changhee Jung. 2023. Write-Light Cache for Energy Harvesting Systems. In *ISCA (2023)*.

[11] Yizi Gu, Yongpan Liu, Yiqun Wang, Hehe Li, and Huazhong Yang. 2016. NVPsim: A simulator for architecture explorations of nonvolatile processors. In *ASP-DAC*.

[12] Anunay Gupta, Om Prakash Yadav, Douglas DeVoto, and Joshua Major. 2018. A review of degradation behavior and modeling of capacitors. In *International Electronic Packaging Technical Conference and Exhibition (2018)*.

[13] Matthew R Guthaus, Jeffrey S Ringenberg, Dan Ernst, Todd M Austin, Trevor Mudge, and Richard B Brown. 2001. MiBench: A free, commercially representative embedded benchmark suite. In *IEEE WWC (2001)*.

[14] Matthew Hicks. 2017. Clank: Architectural support for intermittent computation. In *ACM SIGARCH Computer Architecture News (2017)*.

[15] Shao-Yu Huang, Jianping Zeng, Xuanliang Deng, Sen Wang, Ashrarul Sifat, Burhanuddin Bharmal, Jia-Bin Huang, Ryan Williams, Haibo Zeng, and Changhee Jung. 2023. RTailor: Parameterizing Soft Error Resilience for Mixed-Criticality Real-Time Systems. In *RTSS (2023)*.

[16] Texas Instruments. 2015. TPS380x-Q1 Voltage Detectors. https://www.ti.com/product/TPS3803-Q1

[17] Hrishikesh Jayakumar, Arnab Raha, and Vijay Raghunathan. 2014. QuickRecall: A low overhead HW/SW approach for enabling computations across power cycles in transiently powered computers. In *International Conference on VLSI Design*.

[18] Jungi Jeong, Jaewan Hong, Seungryoul Maeng, Changhee Jung, and Youngjin Kwon. 2020. Unbounded hardware transactional memory for a hybrid DRAM/NVM memory system. In *IEEE/ACM MICRO (2020)*.

[19] Jungi Jeong and Changhee Jung. 2021. PMEM-spec: persistent memory speculation (strict persistency can trump relaxed persistency). In *ASPLOS (2021)*.

[20] Jungi Jeong, Jianping Zeng, and Changhee Jung. 2022. Capri: Compiler and architecture support for whole-system persistence. In *HPDC (2022)*.

[21] Changhee Jung, Daeseob Lim, Jaejin Lee, and SangYong Han. 2005. Adaptive execution techniques for SMT multiprocessor architectures. In *ACM PPoPP (2005)*.

[22] Hongjune Kim, Jianping Zeng, Qingrui Liu, Mohammad Abdel-Majeed, Jaejin Lee, and Changhee Jung. 2020. Compiler-directed soft error resilience for lightweight GPU register file protection. In *PLDI (2020)*.

[23] Chunho Lee, Miodrag Potkonjak, and William H Mangione-Smith. 1997. Mediabench: A tool for evaluating and synthesizing multimedia and communications systems. In *IEEE MICRO*.

[24] Jaejin Lee, Jung-Ho Park, Honggyu Kim, Changhee Jung, Daeseob Lim, and SangYong Han. 2010. Adaptive execution techniques of parallel programs for multiprocessors. *J. Parallel and Distrib. Comput.* (2010).

[25] Qingrui Liu, Joseph Izraelevitz, Se Kwon Lee, Michael L Scott, Sam H Noh, and Changhee Jung. 2018. iDO: Compiler-directed failure atomicity for nonvolatile memory. In *IEEE/ACM MICRO (2018)*.

[26] Qingrui Liu and Changhee Jung. 2016. Lightweight hardware support for transparent consistency-aware checkpointing in intermittent energy-harvesting systems. In *NVMSA (2016)*.

[27] Qingrui Liu, Changhee Jung, Dongyoon Lee, and Devesh Tiwari. 2015. Clover: Compiler Directed Lightweight Soft Error Resilience. In *LCTES (2015)*.

[28] Qingrui Liu, Changhee Jung, Dongyoon Lee, and Devesh Tiwari. 2015. Clover: Compiler directed lightweight soft error resilience. *ACM Sigplan Notices* (2015).

[29] Qingrui Liu, Changhee Jung, Dongyoon Lee, and Devesh Tiwari. 2016. Compiler-Directed Lightweight Checkpointing for Fine-Grained Guaranteed Soft Error Recovery. In *SC (2016)*.

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

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

[32] Qingrui Liu, Xiaolong Wu, Larry Kittinger, Markus Levy, and Changhee Jung. 2017. Benchprime: Effective building of a hybrid benchmark suite. *ACM TECS* (2017).

[33] Yongpan Liu, Zewel Li, Hehe Li, Yiqun Wang, Xueqing Li, Kaisheng Ma, Shuangchen Li, Meng-Fan Chang, Jack Sampson, Yuan Xie, Jiwu Shu, and Huazhong Yang. 2015. Ambient Energy Harvesting Nonvolatile Processors: From Circuit to System. In *DAC (2015)*.

[34] Brandon Lucia, Vignesh Balaji, Alexei Colin, Kiwan Maeng, and Emily Ruppel. 2017. Intermittent Computing: Challenges and Opportunities. In *Proceedings in Informatics, Vol. 71. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik*.

[35] Michele Magno, Dario Kneubuhler, Philipp Mayer, and Luca Benini. 2018. Micro kinetic energy harvesting for autonomous wearable devices. In *SPEEDAM (2018)*.

[36] Davide Pala, Ivan Miro-Panades, and Olivier Sentieys. 2021. Freezer: A Specialized NVM Backup Controller for Intermittently Powered Systems. *IEEE TCAD* (2021).

[37] Sebin Shaji Philip, Roberto Passerone, Kasim Sinan Yildirim, and Davide Brunelli. 2023. Intermittent Computing Emulation of Ultralow-Power Processors: Evaluation of Backup Strategies for RISC-V. *IEEE TCAD* (2023).

[38] Shashank Priya and Daniel J Inman. 2009. Energy harvesting technologies.

[39] ROHM. 2013. Datasheet of BD5xxx Free Delay Time Setting CMOS Voltage Detector IC Series. https://fscdn.rohm.com/en/products/databook/datasheet/ic/power/voltage_detector/bd52xxg-e.pdf

[40] Priyanka Singla and Smruti R. Sarangi. 2022. A Survey and Experimental Analysis of Checkpointing Techniques for Energy Harvesting Devices. *J. Syst. Archit.* (2022).

[41] Fang Su, Yongpan Liu, Yiqun Wang, and Huazhong Yang. 2016. A Ferroelectric Nonvolatile Processor with 46mus System-Level Wake-up Time and 14mus Sleep Time for Energy Harvesting Applications. In *IEEE Transactions on Circuits and Systems I (2016)*.

[42] Fang Su, Kaisheng Ma, Xueqing Li, Tongda Wu, Yongpan Liu, and Vijaykrishnan Narayanan. 2017. Nonvolatile processors: Why is it trending?. In *DATE (2017)*.

[43] David Tarjan, Shyamkumar Thoziyoor, and Norman Jouppi. 2006. CACTI 4.0. (2006).

[44] Alexander Teverovsky. 2014. Insulation Resistance and Leakage Currents in Low-Voltage Ceramic Capacitors With Cracks. *IEEE Transactions on Components, Packaging and Manufacturing Technology* (2014).

[45] Joel Van Der Woude and Matthew Hicks. 2016. Intermittent computation without hardware support or programmer intervention. In *OSDI (2016)*.

[46] Sen Wang, Dong Li, Ashrarul H Sifat, Shao-Yu Huang, Xuanliang Deng, Changhee Jung, Ryan Williams, and Haibo Zeng. 2024. Optimizing Logical Execution Time Model for Both Determinism and Low Latency. In *RTAS (2024)*.

[47] Yilun Wu, Byounguk Min, Mohannad Ismail, Wenjie Xiong, Changhee Jung, and Dongyoon Lee. 2024. {IntOS}: Persistent Embedded Operating System and Language Support for Multi-threaded Intermittent Computing. In *OSDI (2024)*.

[48] Mimi Xie, Mengying Zhao, Chen Pan, Jingtong Hu, Yongpan Liu, and Chun Jason Xue. 2015. Fixing the broken time machine: Consistency-aware checkpointing for energy harvesting powered non-volatile processor. In *DAC (2015)*.

[49] Cheuk-Wang Yau, Tyrone Tai-On Kwok, Chi-Un Lei, and Yu-Kwong Kwok. 2018. Energy harvesting in internet of things.. In *IEEE Communications Magazine (2018)*.

[50] Jianping Zeng, Jongouk Choi, Xinwei Fu, Ajay Paddayuru Shreepathi, Dongyoon Lee, Changwoo Min, and Changhee Jung. 2021. ReplayCache: Enabling Volatile Cachesfor Energy Harvesting Systems. In *IEEE/ACM MICRO (2021)*.

[51] Jianping Zeng, Shao-Yu Huang, Jiuyang Liu, and Changhee Jung. 2024. Soft Error Resilience at Near-Zero Cost. In *SC (2024)*.

[52] Jianping Zeng, Jungi Jeong, and Changhee Jung. 2023. Persistent Processor Architecture. In *IEEE/ACM MICRO (2023)*.

[53] Jianping Zeng, Hongjune Kim, Jaejin Lee, and Changhee Jung. 2021. Turnpike: Lightweight Soft Error Resilience for In-Order Cores. In *IEEE/ACM MICRO (2021)*.

[54] Jianping Zeng, Tong Zhang, and Changhee Jung. 2024. Compiler-Directed Whole-System Persistence. In *ISCA (2024)*.

[55] Yida Zhang and Changhee Jung. 2022. Featherweight Soft Error Resilience for GPUs. In *IEEE/ACM MICRO (2022)*.

[56] Yuchen Zhou, Jianping Zeng, Jungi Jeong, Jongouk Choi, and Changhee Jung. 2023. SweepCache: Intermittence-Aware Cache on the Cheap. In *IEEE/ACM MICRO (2023)*.

[57] Yuchen Zhou, Jianping Zeng, and Changhee Jung. 2024. LightWSP: Whole-System Persistence on the Cheap. In *IEEE/ACM MICRO (2024)*.