Compiler-Directed High-Performance Intermittent Computation with Power Failure Immunity

Jongouk Choi
University of Central Florida

Larry Kittinger
Block.one

Qingrui Liu
Annapurna Labs

Changhee Jung
Purdue University

I. Introduction

Energy harvesting is the logical next step in the evolution of IoT thanks to its self-sustaining, maintenance-free, and environmentally-friendly nature. However, energy harvesting systems are prone to power failure since the harvested energy is unstable and the absence of a battery. Since the systems use a tiny capacitor as an energy buffer, they intermittently compute only when it provides sufficient energy, which would otherwise die, thus being called *intermittent computation*. This implies that frequent power outages become the norm of program execution, forcing it to restart from the beginning. Hence, an intermittently-powered microcontroller (MCU) uses non-volatile memory (NVM) as main memory without caches—due to their power demand—and has some form of recovery support to backup and restore necessary data across the outage.

Existing software-based recovery schemes partition program into a series of recoverable regions (tasks) with checkpointing/logging their input register/memory data in NVM. If any region is interrupted due to power failure, the recovery schemes, in the wake of the failure, first restore the checkpointed/logged data by loading them from NVM and then resume the program at the beginning of the interrupted region [1], [2]; this is so-called rollback recovery.

Unfortunately, the existing recovery schemes are not systematic, forming their regions sometimes too conservatively or aggressively. If regions are too short (i.e., unnecessarily making frequent checkpoints at each region boundary), the schemes consume more energy for checkpointing but use less energy for computing; that is because checkpoints are the most energy-consuming in that they are essentially NVM stores. While one could take an aggressive approach by forming long regions for fewer checkpoints, expensive re-execution penalty has to be paid by restarting such a long region possibly many times across power outages. Either way, the forward execution progress is limited leading to significant performance degradation. Even worse, the existing recovery schemes could suffer from a *stagnation* problem—livelock-like situation where power failure repeatedly occurs before some long region finishes—making no forward progress despite the continuous consumption of hard-won energy.

This paper introduces power failure immunity (PFI), a novel program execution property for achieving energy-efficient intermittent computation, to address these challenges. Regardless of power failure frequency, PFI ensures that each code region can fail only once, resulting in a single in-region outage. If a region experiences a power outage, it never fails again during the re-execution—as if it had been immunized after the first failure. Because of the never-fail-again nature, intermittent computation schemes can use aggressive region formation (i.e., long regions) without incurring the costly re-execution penalty; thus, enforcing PFI technically solves the stagnation problem.

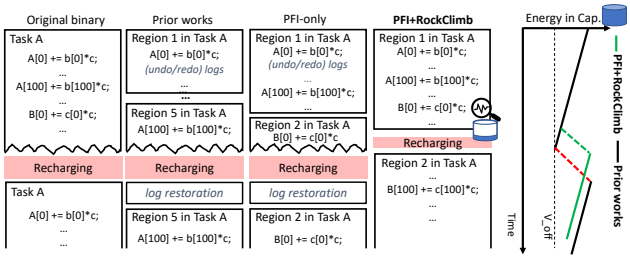

Fig. 1: Comparison of intermittent computation schemes: Each scheme runs the same program (Task A). While prior works form many regions, e.g., Region 1∼5 in Task A, PFI generates a few regions, and PFI+RockClimb further lengthens the region size and eliminates the re-execution.

The key insight is that energy harvesting systems do not boot until their energy buffer (capacitor) is fully charged. That is, in the event of a power outage, the program is guaranteed to make as much progress as the full energy buffer allows, even if no additional energy is harvested. With this in mind, this paper introduces compiler-directed PFI enforcement, which partitions the program into a series of code regions based on the energy buffer size, so that each region can be completed in a single charge cycle using the energy buffered.

Finally, this paper introduces RockClimb, a compiler optimization technique that leverages PFI to achieve *rollback-free* intermittent computation in such a way that PFI-enforced regions never fail, resulting in no in-region outage. At a high level, RockClimb determines whether a fully buffered energy is secured at each region boundary to ensure that the next region is completed without power failure. RockClimb waits until the energy buffer is fully charged before executing the next region if it is not secured. As a result, the rollback-free nature of RockClimb eliminates the need for logging for each memory write, which was required in previous work [2] to achieve rollback recovery of power failure, as shown in Figure 1.

II. Power Failure Immunity: Single In-Region Power Outage

To prevent repetitive in-region outages, this paper leverages two important observations. First, energy harvesting systems do not start to operate their microcontoller (MCU) until the energy buffer (capacitor) is fully charged as with virtually all commodity systems. That is, when the MCU is ready to resume the execution in the wake of power outage, the

capacitor is always sure to have the fully buffered (charged) energy at the starting point of the resumption. The implication is that the power-interrupted region can make as much progress as the full energy buffer allows, even if there is no additional energy is harvested. We refer to the minimum progress time, for which the MCU can be sustained under the fully buffered energy, as safe active time (SAT).

The second observation is that if the worst-case execution time (WCET) of any region is shorter than the SAT, the region is assured to finish with no power failure under the fully buffered energy.

$$\text{PFI enforcement constraint: } WCET\,(r) < SAT\,(\mu) \qquad (1)$$

With that in mind, for a given region (r) and the underlying MCU (μ), we formulate the problem of ensuring the forward execution progress as Eq.1 above. Thus, PFI can achieve stagnation freedom by partitioning the original program into *SAT-safe regions*, each of which satisfies the PFI constraint Eq.1; even if the SAT-safe regions may encounter power failure, they **never fail again** upon recovery from the failure. In other words, when power comes back, the previously interrupted region never retreats before reaching the end of the region, which ensures forward progress to the next region without exception.

III. RockClimb: Power-Failure-Free Execution

Once SAT-safe regions are formed, our compiler enables RockClimb that leverages the PFI as a basis for achieving rollback-free intermittent computation, i.e., extending the PFI to much stronger guarantee that no region ever fails. To complete each PFI-enforced region with no power failure, RockClimb checks the energy buffer at each region boundary. If the buffer is not fully charged, RockClimb waits for the buffer to secure the full energy before starting the next region; otherwise, it is immediately started with the guarantee of failure-free completion—because a PFI-enforced region can always finish with a fully buffered capacitor.

For this purpose, RockClimb leverages a voltage emergency interrupt of commodity energy harvesting systems. When a program control reaches the end of a region, RockClimb checks the energy availability (full capacitance) before starting the next region. For this purpose, the compiler inserts—at each region boundary—the voltage level checking code. That is, at each region boundary, RockClimb enables the voltage interrupt by controlling the interrupt vector and immediately starts the next region unless the interrupt is generated; otherwise, RockClimb puts the microcontroller (MCU) to a power-down mode so that it can be rebooted when the buffer is fully charged.

In particular, RockClimb's guarantee of no in-region failure simplifies achieving crash consistency obviating the memory logs in each region. Since RockClimb's regions are never power-interrupted (thus no rollback), they do not have to log memory inputs at all; the absence of rollback recovery means no need to handle the restoration of memory inputs.

The upshot is that RockClimb's rollback-freedom saves the high energy/latency of the NVM logging stores, thereby achieving an energy-efficient and high-performance energy harvesting system. Figure 1 highlights RockClimb compared to prior works that partition program into several regions with memory logs for recovery. While the prior works keep spending their energy for logging, restoring, and re-executing as shown in the figure, RockClimb here makes a further forward progress due to its log-free and re-execution-free intermittent computation.

IV. Evaluation

We implemented PFI+RockClimb in the LLVM compiler infrastructure and conducted experiments by running compute-intensive 11 benchmarks. To evaluate the effectiveness of PFI+RockClimb, we conducted experiments using TI's MSP430FR5994 evaluation board with Powercast P2110-EVB RF energy harvester as our energy harvesting system testbed; we equipped the board with a 10μF capacitor. To power the energy harvesting system, we used Powercast TX91501-3W transmitter. In the environment, we compared PFI+RockClimb to prior works such as Ratchet [1], Chinchilla [2], and RockClimb-disabled PFI-only scheme. As shown in Figure 2, PFI+RockClimb outperforms the works by about 2X on average.

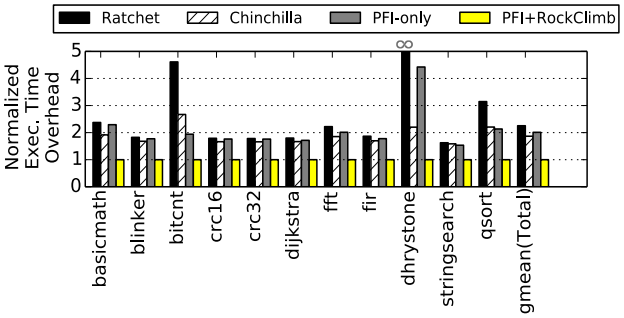

Fig. 2: Performance results in real energy harvesting situation. We compare PFI+RockClimb with Ratchet and Chinchilla. Y-axis shows the normalized execution time to PFI+RockClimb. ∞ represents the stagnation problem.

V. Summary

This paper introduces power failure immunity (PFI) that ensures each code region can fail at most once. In the virtue of PFI, this work presents RockClimb, a rollback-free intermittent computation scheme, ensuring that PFI-enforced regions never fail. Consequently, PFI+RockClimb achieves high-performance intermittent computation.

References

[1] J. V. D. Woude and M. Hicks, "Intermittent computation without hardware support or programmer intervention," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*.

[2] K. Maeng and B. Lucia, "Adaptive dynamic checkpointing for safe efficient intermittent computing," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, (Carlsbad, CA), pp. 129–144, USENIX Association, 2018.