SweepCache: Intermittence-Aware Cache on the Cheap

Yuchen Zhou
Purdue University

Jianping Zeng
Purdue University

Jungi Jeong*
Purdue University

Jongouk Choi
University of Central Florida

Changhee Jung
Purdue University

1 INTRODUCTION

Energy harvesting systems are becoming more prevalent in a wide range of applications, *e.g.,* vehicle tire pressure sensing, health and wellness monitoring, and wearable computing, just to name a few. However, due to the unstable nature of the energy sources, *e.g.,* radio frequency (RF) and Wifi, these applications experience frequent and unpredictable power failure during program execution—thus being called intermittent computation.

To resist frequent power failure, previous studies proposed a nonvolatile processor (NVP) by leveraging byte-addressable nonvolatile memory (NVM) as the main memory and voltage monitor based volatile data checkpointing. Whenever the monitor detects a voltage drop below a predefined threshold, *i.e.,* a sign of impending power failure, NVP is interrupted to checkpoint all registers before the failure; this is so-called just-in-time (JIT) checkpointing. In the wake of the failure, NVP restores the registers and continues to progress from the interruption point.

Nonetheless, the performance of NVP is limited by NVM accesses that are the most expensive in terms of both energy consumption and instruction latency. To address this problem, prior work proposes cache-enabled designs, *e.g.,* NVSRAM, checkpointing and restoring not only the registers but also the cache across power failure. However, checkpointing the entire cache consumes too much harvested energy thus limiting forward progress.

Moreover, all prior cache-enabled designs rely on JIT checkpointing which incurs nontrivial hardware complexity, *e.g.,* the voltage monitor, the backup/restoration logic, the nonvolatile flip-flops (NVFFs[1]). Furthermore, the JIT checkpointing requires that the backup should be performed in a failure-atomic way, which forces energy harvesting systems to dedicate a large amount of hard-won energy for the failure-atomic backup—leaving only a portion of harvested energy for computation.

With that in mind, this paper proposes SweepCache, a novel JIT-checkpoint-free design that achieves lightweight yet performant intermittent computation for cache-enabled energy harvesting systems by using intelligent compiler/architecture interaction. SweepCache's compiler partitions program into a series of recoverable regions—with their live-out registers checkpointed via store instructions—so that the region boundary serves as a recovery point across power failure. Then, to facilitate correct power failure recovery, SweepCache architecture runs them through *region-level persistence*, *i.e.,* all stores of a region including the checkpoint stores must be persisted to NVM before the next region starts.

Specifically, in case a region is power-interrupted, SweepCache holds all data of its stores in an NVM-resident buffer—we call persist buffer—before persisting them to the main memory (NVM), though it lengthens the critical path of memory accesses by keeping the buffer between the cache and NVM. During region execution, all cache writebacks (*i.e.,* dirty cacheline evictions) are first quarantined in the buffer; so it acts like a redo buffer for protecting the main memory against the stores of power-interrupted regions. Thus, no matter when power failure occurs, either the buffer contents or their target NVM locations always remain intact. This serves as a basis for correct recovery on top of *region-level persistence*. In other words, for *region-level persistence*, when program control reaches each region end, SweepCache [2] sweeps all dirty cachelines to the persist buffer and then moves them to the NVM. This effectively makes each region begin with a clean cache lacking dirty cachelines.

Note that such cache sweeping technically shortens the memory access path; it turns out that the loads/stores of each region usually make no dirty cacheline eviction, keeping the buffer empty since the clean cache can afford to accommodate them. The upshot is that thanks to the empty buffer, SweepCache can bypass the buffer search on a cache miss, which would otherwise end up with two NVM accesses, *i.e.,* buffer search and NVM access (in case of the buffer miss). This clean cache effect contributes to SweepCache's high performance—along with *inter-region parallelism* (Section 2.3).

2 SWEEPCACHE APPROACH

2.1 Compiler-Assisted Register Checkpointing

Rather than relying on JIT checkpointing, SweepCache leverages compiler techniques to partition program into a series of regions with live-out registers checkpointed (via stores) to NVM in a region granularity and thus can be used for region-level failure recovery. As shown in Figure 1(a), the *stores* are normal stores and *ckpt stores* are register checkpointing stores. In particular, regions are partitioned based on the buffer size, ensuring that the buffer never overflows (*i.e.,* the buffer can hold all stores of each region).

2.2 Region-Level Store Persistence

Without the luxury of JIT checkpointing, SweepCache instead offers persistence and recovery in a region granularity by deploying the persist buffer as a redo buffer.

To be more specific, all the cache writebacks of each region are first directed into the persist buffer (*s-phase1*). After that, the buffer commits its content into NVM (*s-phase2*), as shown in Figure 1(b). Temporally, the persistence process is split into three phases, which correspond to writebacks before the region end (*t-phase1*), flushing

[1] Some prior work such as QuickRecall [2] checkpoints registers to NVM (not NVFFs), but it is more time- and energy-consumingg [3].

*Now at Google.

[2] The name SweepCache is inspired by its action of having the cache swept clean between regions.

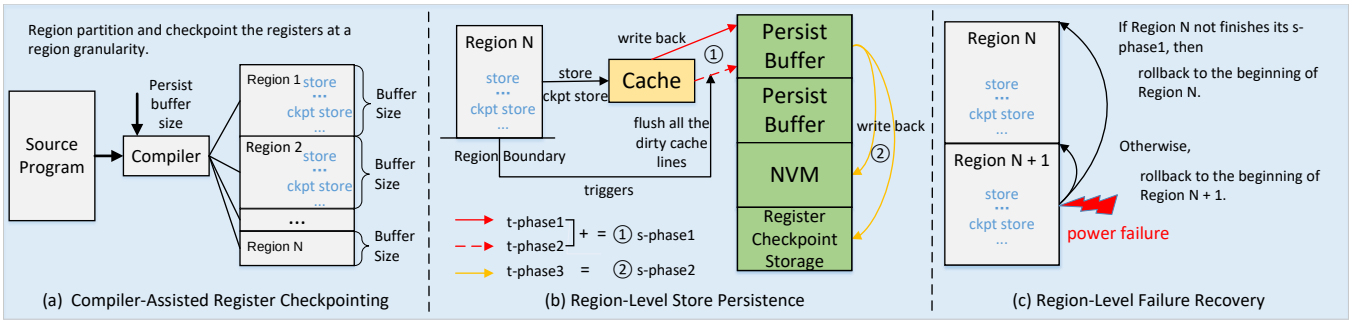

Figure 1: The high-level view of SweepCache compiler and architecture

dirty cachelines at the region boundary (*t-phase2*), and buffer commit (*t-phase3*). In this way, SweepCache ensures correct *region-level persistence* no matter when power failure happens in that either the NVM or the buffer can always remain consistently available.

2.3 Inter-Region Parallelism

To achieve *region-level persistence*, a region cannot start executing until the previous region is persisted. However, one critical issue is that the persistence latency at each region boundary can significantly degrade performance. To mitigate this issue, SweepCache introduces *inter-region parallelism*, which allows the next region to speculatively execute as if the prior region were already persisted. This helps to hide the persistence latency but ends up with structural hazards as adjacent regions compete for the persist buffer. Specifically, before the prior region finishes its persistence, the next region's speculative execution can overwrite the buffer, thereby making the *region-level persistence* fail to achieve crash consistency. Ideally, each region should be assigned a separate buffer, but incurring nontrivial hardware costs. Alternatively, the following region should wait for the prior region to complete its persistence, hurting the performance a lot. Fortunately, it turns out that two persist buffers are sufficient to achieve high parallelism, effectively hiding the persistence latency without compromising the crash consistency guarantee. That is why two persist buffers are present in Figure 1(b).

2.4 Region-Level Failure Recovery

SweepCache conducts appropriate recovery based on the point of power failure, either before *s-phase1* or after, as shown in Figure 1(c). If a power outage occurs before *s-phase1*, SweepCache discards the buffer contents and rolls back to the beginning of the power-interrupted region. In case power failure happens after *s-phase1* and during *s-phase2*, the region is considered successfully persisted, then SweepCache re-executes the *s-phase2* and restarts from the next region's beginning once the power comes back.

3 EVALUATION

We implement compiler techniques using LLVM 13.0.1. Performance evaluation is conducted with benchmarks Mibench and Mediabench on the gem5 [1] simulator with ARM ISA, utilizing a real power trace collected from an RF reader that suffers frequent power failure. As shown in Figure 2, SweepCache achieves 3.65x and 6.03x average speedups over two state-of-the-art schemes ReplayCache and NvMR, respectively.

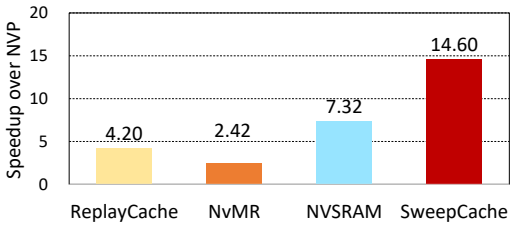

Figure 2: Speedups over NVP with RFOffice power trace.

4 CONCLUSION

This paper presents SweepCache, a novel compiler and architecture co-design approach that enables energy harvesting systems to exploit a volatile cache in a performant and lightweight way. To ensure correct power failure recovery, the compiler generates recoverable regions while the architecture runs them in a failure-atomic way. Thanks to SweepCache's *region-level persistence* that cleans up the cache across the region boundary, energy harvesting systems do not have to rely on expensive just-in-time (JIT) checkpointing, and thus they can fully utilize harvested energy for computation. As a result, SweepCache achieves 3.65x and 6.03x average speedups over two state-of-the-art work ReplayCache and NvMR under RFOffice trace, respectively.

REFERENCES

[1] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, Somayeh Sardashti, et al. The gem5 simulator. *ACM SIGARCH computer architecture news*, 39(2):1–7, 2011.

[2] Hrishikesh Jayakumar, Arnab Raha, and Vijay Raghunathan. Quickrecall: A low overhead hw/sw approach for enabling computations across power cycles in transiently powered computers. In *2014 27th International Conference on VLSI Design and 2014 13th International Conference on Embedded Systems*, pages 330–335. IEEE, 2014.

[3] Yiqun Wang, Yongpan Liu, Shuangchen Li, Daming Zhang, Bo Zhao, Mei-Fang Chiang, Yanxin Yan, Baiko Sai, and Huazhong Yang. A 3us wake-up time nonvolatile processor based on ferroelectric flip-flops. In *2012 Proceedings of the ESSCIRC (ESSCIRC)*, pages 149–152. IEEE, 2012.

[4] Yuchen Zhou, Jianping Zeng, Jungi Jeong, Jongouk Choi, and Changhee Jung. Sweepcache: Intermittence-aware cache on the cheap. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 1059–1074, 2023.