ReplayCache: Enabling Volatile Caches for Energy Harvesting Systems

Jianping Zeng
Purdue University

Jongouk Choi
Purdue University

Xinwei Fu
Virginia Tech

Ajay P. Shreepathi
Stony Brook University

Dongyoon Lee
Stony Brook University

Changwoo Min
Virginia Tech

Changhee Jung
Purdue University

1 INTRODUCTION

Energy harvesting systems have been deployed in a wide range of application domains thank to the ability to collect necessary energy from variant ambient sources—e.g., solar, thermal, and frequency radiation—and are now considered to be an enabling technology for wearables and other tiny IoT devices.

However, due to the batteryless nature, energy harvesting systems suffer unpredictable frequent power failure and thus require some form of crash consistency. Thus, existing systems have been designed with byte-addressable non-volatile memory (NVM), where data are immediately persisted and thus recoverable at the cost of long latency. While volatile write-back caches can hide the store latency and improve performance with a load hit exploiting data locality, they have been assumed to be not viable or at least challenging in energy harvesting systems.

The crux of the problem is that volatile write-back cache states are not preserved across a power outage. This may lead to an inconsistent NVM state, and therefore the power-interrupted program may fail to resume correctly. That is why existing energy harvesting systems do not use volatile data caches; prior work uses a read-only NVM-based instruction cache where crash consistency (without stores) is not an issue. Unfortunately, it is a challenging problem to ensure correct data cache persistence in a lightweight manner to maintain forward progress. For example, software logging causes serious performance degradation (100-300 % slowdown) since each regular store is preceded by the persistence barrier.

One possible hardware solution is to use a volatile write-through cache. It allows energy harvesting systems to benefit from load hits and to ensure crash consistency by enforcing that the completion of a store instruction guarantees the persistence of the data in NVM. However, write-through cache comes with a performance penalty on each store as conventional cache-free energy harvesting processors. Since they use a simple in-order core without any form of speculation, they cannot hide the data persistence latency.

Alternatively, one can design a persistent write-back data cache, e.g., non-volatile cache (NVCache) and non-volatile SRAM cache (NVSRAMCache). However, both cache designs have their own problems. Due to the NVM-based design, NVCaches incur high latency and power consumption for each access. NVSRAMCaches embed NVM to backup a SRAM-based cache, and checkpoint/restore the entire SRAM to/from the NVM backup across power failure, leading to high energy consumption. While NVSRAMCaches may be as fast as a volatile SRAM cache without power failure, it is hard to maintain the performance with frequent failure—i.e., the norm of energy harvesting—unless they use a lower-power yet fast non-volatile technology which has not been commercialized yet.

With that in mind, we propose ReplayCache, a *software-only scheme* that enables commodity energy harvesting systems to exploit a volatile write-back data cache for performance, yet ensures lightweight crash consistency of the NVM state for correctness. ReplayCache does not ensure the persistence of dirty cachelines or record their logs at run time; *i.e.,* no write amplification. Instead, ReplayCache *replays the potentially unpersisted stores* in the wake of power failure to restore the consistent NVM state from which the interrupted program can safely resume.

To realize the store replay, ReplayCache partitions program into a series of regions so that the operand registers of store instructions are intact (i.e., not overwritten by the other following instructions) in each region. We refer to this process *store-register-preserving* region formation. Then, at run time, ReplayCache checkpoints all registers just before power failure to secure the store operand registers. We note that the just-in-time register checkpointing is already available in energy harvesting systems. During recovery, these checkpointed registers are used to re-execute the stores along the same program path as the one before the power failure; for the store replay, a recovery code block is generated for each region, *i.e.,* ReplayCache directs program control to the recovery code in the wake of the power failure. After that, ReplayCache can safely resume from the interrupted program point with the checkpointed registers and the recovered consistent NVM.

2 PROGRAM REGION PARTITIONING

As shown in Figure 1(a), ReplayCache compiler partitions entire program input to a series of regions. Each region guarantees that the operand registers (*e.g.,* address, value) of stores therein are not overwritten by any other succeeding instructions in that region.

3 REGION-LEVEL PERSISTENCE

To achieve high performance, ReplayCache does not rely on any logging for each store. Instead, ReplayCache asynchronously writes back the stored values to the NVM, and overlaps the write-back operations with the executions of other following instructions, effectively exploiting instruction-level parallelism (ILP).

Unlike a traditional write-back cache, ReplayCache ensures that all the stores in a region are persisted (written back to the NVM) before the region ends; this paper calls this *region-level* persistence in which the persistence latency of in-region stores can be naturally hidden by ILP; Figure 1(b) illustrates the window of potential ILP gain, and the unpersisted state of each store. This region-level persistence assures that at the moment of a power outage, all the stores in the preceding program regions have already been persisted,

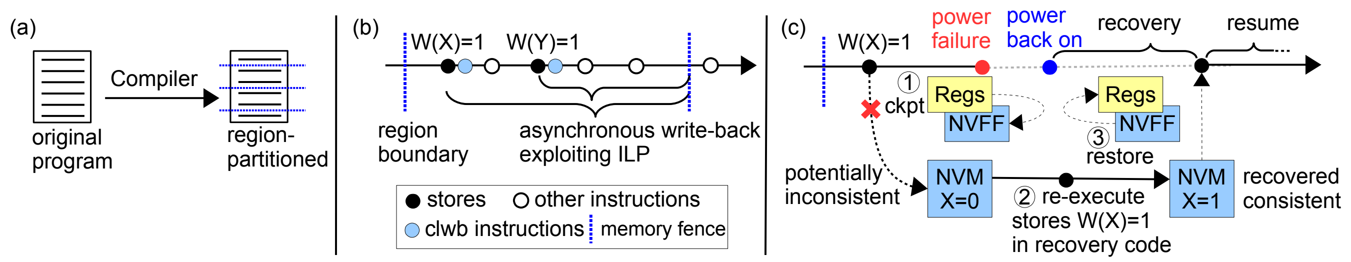

Figure 1: A high-level design of ReplayCache.

and only the stores in the interrupted region could potentially be unpersisted.

The processor stalls if there exists an outstanding unpersisted store at the end of a region, until it becomes persisted to the NVM. ReplayCache compiler dedicates a single register (*e.g.,* $r12$) to be acted as *region register* to track the most recent region boundary information for recovery code searching. That is, the register is updated with a program counter at each region boundary.

4 JIT REGISTER CHECKPOINTING

Across a power outage, ReplayCache saves register states just before the outage and restores them in the wake of the outage using the voltage monitor based JIT checkpointing mechanism in commodity energy harvesting systems. For instance, QuickRecall [2] and NVP [4] can both checkpoint register states before the power off and to restore them after the power on. In Figure 1(c), step ① illustrates that ReplayCache checkpoints the registers when power is about to be cut off.

5 RECOVERY PROTOCOL

To recover the program status from power failure, ReplayCache compiler generates necessary data structure and per-region recovery code blocks to facilitate recovery process. The recovery protocol works as follows. Upon a power outage, the interrupted region's stores before the outage may or may not be persisted, *e.g.,* $W(X) = 1$ in Figure 1(c) unpersisted till the outage—while all preceding regions' stores are guaranteed to be persisted and thus consistent (due to the region-level persistence). In the wake of the outage, ReplayCache jumps to the recovery code block of the interrupted region to replay all the stores left behind the outage. The recovery code block re-executes such unpersisted stores using the checkpointed register values in either nonvolatile flip-flop (NVP) or NVM (QuickRecall). This is shown as a step ② of Figure 1(c). Finally, the recovery code sets off a restoration signal to restore all registers (including PC) from nonvolatile flip-flop or NVM, and then resumes the program from the outage point with the restored registers and the recovered NVM states as in step ③ of Figure 1(c). In this way, ReplayCache allows energy harvesting systems to seamlessly leverage a writeback data cache without amplifying NVM stores.

6 EVALUATION

We implement all ReplayCache compiler passes using the LLVM compiler infrastructure [3] and evaluate ReplayCache on top of the gem5 simulator [1] with ARM ISA. We simulate all the applications of Mediabench and Mibench with a RF-based real power trace suffering frequent power failures. As shown in Figure 2, ReplayCache achieves an average 8.5X speedup over the baseline (original cache-free NVP). Note that the performance of ReplayCache reaches 80% of that of a hardware-expensive ideal NVSRAMCache NVP design.

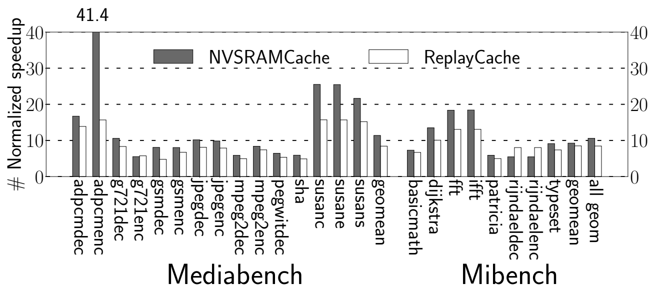

Figure 2: Performance results with RF-Office power trace normalized to the baseline without a cache.

7 SUMMARY

This paper presents ReplayCache, a software-only scheme that enables energy harvesting systems to take advantage of a volatile data cache efficiently and correctly. To achieve crash consistency with the volatile data cache, ReplayCache proposes a replay-based solution that restores the operands of potentially unpersisted stores from the register checkpoint and then re-executes them to restore consistent non-volatile memory status. Experimental results show that compared to the baseline with no cache, ReplayCache significantly improves the performance by 8.5x on average, while ensuring correct resumptions even in the presence of unpredictable and frequent power outages.

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. 2011. The gem5 simulator. *ACM SIGARCH computer architecture news* 39, 2 (2011), 1–7.
[2] 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 *2014 27th International Conference on VLSI Design and 2014 13th International Conference on Embedded Systems.* IEEE, 330–335.
[3] Chris Lattner and Vikram Adve. 2004. LLVM: A compilation framework for lifelong program analysis & transformation. In *International Symposium on Code Generation and Optimization, 2004. CGO 2004.* IEEE, 75–86.
[4] Kaisheng Ma, Yang Zheng, Shuangchen Li, Karthik Swaminathan, Xueqing Li, Yongpan Liu, Jack Sampson, Yuan Xie, and Vijaykrishnan Narayanan. 2015. Architecture exploration for ambient energy harvesting nonvolatile processors. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA).* IEEE, 526–537.
[5] 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 *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture.* 170–182.