CoSpec: Compiler Directed Speculative Intermittent Computation

Jongouk Choi
Purdue University
choi658@purdue.edu

Qingrui Liu
Annapurna Labs
qingrui@amazon.com

Changhee Jung
Purdue University
chjung@purdue.edu

1 INTRODUCTION

Energy harvesting systems continue to grow at a rapid pace due to their batteryless nature. However, since ambient energy source is unreliable, the systems suffer frequent power failure. To address the challenge, they use a small capacitor as an energy buffer and intermittently compute only when sufficient energy is secured in the capacitor; when it is depleted, the systems die. This is so-called *intermittent computation*. With the intermittent nature in mind, researchers adopt a low-power in-order processor with byte-addressable nonvolatile memory (NVM) as main memory and offer a crash consistency mechanism to checkpoint necessary data and restore them across power outages.

For the crash consistency, prior works introduce a nonvolatile processor (NVP) [5] that checkpoints volatile registers to nonvolatile flip-flops (NVFFs)—when it is about to be interrupted by power failure—and restores the checkpointed registers from NVFFs in the wake of power failure. Since the NVP restarts exactly at the power interruption point, program states in both NVM and NVFF remain the same across power failure, thereby achieving crash consistency.

Unfortunately, the NVP requires non-trivial hardware modifications. To checkpoint the entire register file right before power failure, they require not only the NVFFs but also a voltage monitor, checkpoint/controller logic, and additional capacitors for the monitor itself. Even worse, the voltage monitor has stability issues such as excessive leakage or capacitor aging effects leading to reduced capacitance and voltage detection delay with unexpected cold-start glitch. To mitigate the issues, the prior NVP works aggressively increase the voltage threshold of the system wake-up/backup, which is energy-inefficient due to the inability to make forward progress unless such a high voltage is secured to wake up the system.

With that in mind, we propose CoSpec, an architecture/compiler co-design scheme that can realize low-cost yet performant intermittent computation for commodity in-order processors used in energy harvesting systems. To realize crash consistency without the voltage monitor based checkpointing, CoSpec leverages speculation assuming that power failure would not occur and thus hold all committed stores in a store buffer (SB)—as if they were speculative—in case of mispeculation; we call this **power failure speculation**. CoSpec compiler first partitions a given program into a series of recoverable regions with the SB size in mind, so that no region overflows the SB during the region execution. When the program control reaches the end of each region, the speculation turns out to be successful; therefore, CoSpec releases all the stores of the region, which have been buffered in the SB, to NVM.

If power failure occurs during the execution of a region, all its stores buffered in the SB disappear because it is volatile. The implication is that such mispeculated stores—left behind power failure—cannot affect any program state in NVM at all. Consequently, the interrupted region can be restarted with consistent program states in the wake of power failure.

While CoSpec provides crash consistency, the region-based speculation window causes pipeline stalls at the end of each region due to the SB release. Since it consists of NVM writes that are the most time-consuming instruction, the stalls are rather long leading to a significant performance overhead. To hide the long NVM write latency of the SB release, CoSpec overlaps the SB release of the current region with the speculative execution of the next region. Such instruction level parallelism (ILP) gives an illusion of out-of-order execution on top of the in-order processor, achieving high-performance intermittent computation.

2 HARDWARE DESIGN

Store Buffer for Power Failure Speculation: Since CoSpec implements the power failure speculation in a region level, all stores of each region must be buffered, which would otherwise fail to recover from mispeculation. For this purpose, CoSpec leverages the store buffer (SB) to hold committed stores of each recoverable code region during its execution—since they are treated as speculative—until the program control reaches the end of the region (i.e., the region boundary) where the speculation turns out to be successful and thus all the buffered stores are released. This speculation approach never allows the stores of any regions being interrupted by power failure to be written to primary main memory (NVM). The takeaway is that with the help of such a gated store buffer [4], speculative stores cannot be released to NVM until they become non-speculative, i.e., their region finishes without power failure.

Failure-Atomic 2-Phase SB release: As the major challenge in achieving correct crash consistency, CoSpec should maintain failure atomicity of the *SB* draining; otherwise, when power failure occurs during the SB release, the resulting partial draining can make it impossible to recover from the failure. To overcome the challenge, CoSpec conducts a 2-phase SB release mechanism, i.e., first draining the SB entries—held by stores of each finished region—to a proxy buffer allocated in NVM and then copying them to the primary main memory area in NVM according to their store address. That way, either the buffer or the primary main memory can always be intact no matter when power is lost.

In particular, CoSpec splits the SB into two parts to enable instruction level parallelism. During the program execution, any two consecutive recoverable code regions exclusively occupy one of the two parts in the SB. That is, each code region—statically partitioned by CoSpec compiler—commits its stores to a different part of the *SB* at run time. When the program control reaches each region boundary, CoSpec drains to NVM only the stores in the part of the SB which is used by the region being finished, using the 2-phase SB release.

3 COMPILER DESIGN

To partition the program into a series of recoverable regions, CoSpec compiler first counts the number of stores while traversing the control flow graph (CFG) of the program [4]. When the number of stores

hits a threshold, i.e., a half the SB size, CoSpec compiler cuts the current basic block—where the last store is counted—by placing a region boundary. Then, the compiler analyzes the live-out registers of the resulting region—because they serve as inputs to some following regions—and inserts a checkpoint instruction, i.e., `store`, to save them into a designated register file (RF) checkpoint storage in NVM. This implies that the recovery of any power-interrupted region should first restores the checkpointed registers before restarting the region. In particular, the compiler saves a PC register at the end of each region—which serves as a recovery point—so that if the forthcoming power failure occurs in the next region, it can be recovered by restarting from the beginning.

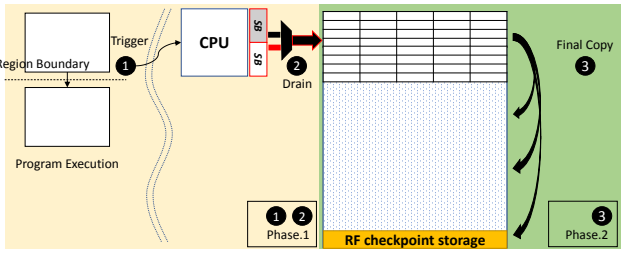

Figure 1: Failure-Atomic 2-Phase Gated Store Buffer Release

4 ARCHITECTURE/COMPILER CO-DESIGN

CoSpec hardware interacts with compiler-formed regions as follows. When the program control reaches the end of each region, CoSpec first drains the buffered stores to a proxy buffer allocated in NVM and in turn moves the drained data from the buffer to the primary main memory in NVM. Thanks to the HW-compiler interaction, CoSpec can not only leverage an existing hardware as is—which would otherwise require expensive hardware modification—but also achieve better energy efficiency compared to hardware-only solutions such as NVPs that should checkpoint the entire register file at the moment of impending power failure.

Figure 1 describes how CoSpec's hardware, i.e., the gated store buffer, interacts with the termination of a compiler-formed recoverable region using the 2-phase SB release protocol. First, the system (❶) triggers the SB release when each region boundary is reached during the program execution. Then, CoSpec (❷) drains one part of *SB*—which corresponds to the region being finished—to the proxy buffer in NVM. As soon as the draining is completed, CoSpec (❸) copies all the buffered data to primary main memory locations.

5 ILP FOR OPTIMIZING 2-PHASE RELEASE

The 2-phase SB release mechanism causes additional NVM writes. Unfortunately, this incurs significant performance degradation. To address this problem, CoSpec optimizes the 2-phase SB release by enabling instruction level parallelism (ILP). More precisely, CoSpec does not wait until its 2-level SB release is finished; rather it speculatively executes the next region's instructions while the SB release is pending. As soon as the SB draining starts, which is shown as the step ❷ in Figure 1, CoSpec speculatively executes the next instructions using the other part of SB in case for stores. By overlapping the NVM writes during entire 2-phase SB release with the speculative execution of the next code region, CoSpec is able to make further forward execution progress—committing more instructions—than a non-ILP execution does. Note that the use of ILP optimization and the region-level speculation window may increase power consumption compared to non-modified design, possibly causing more power failures. To address this issue, CoSpec adaptively turns on/off the ILP and adjusts the speculation window according to the power failure patterns in a reactive manner [1].

6 EVALUATION

We implemented the hardware support on a gem5 simulator [2] with ARM ISA, and the compiler techniques using the LLVM compiler infrastructure [3]. All the experiments were performed with a real energy harvesting trace collected from RF reader that suffers frequent power failures. We compared CoSpec to the prior work, nonvolatile processor (NVP) [5], in terms of completion time, for a mixture of MediaBench and MiBench applications. As shown in Figure 2, CoSpec outperforms the NVP by 3.0X on average.

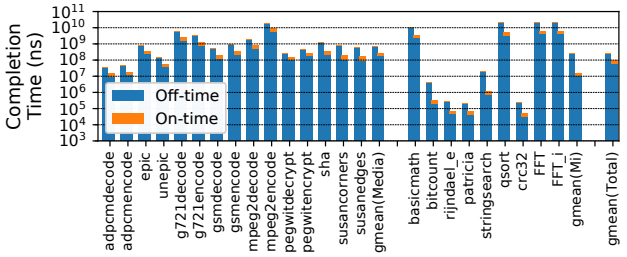

Figure 2: Completion time comparison. The 1st/2nd bars of each application represent NVP and CoSpec, respectively.

7 SUMMARY

This paper presents CoSpec, an architecture/compiler co-designed scheme, that can work for commodity in-order processors, to achieve low-cost yet performant intermittent computation. CoSpec realizes instruction level parallelism on top of the in-order processor pipeline to hide the long latency of nonvolatile memory writes, thereby improving the performance significantly. Our experiments on a real energy harvesting trace with frequent power outages demonstrate that CoSpec outperforms the state-of-the-art nonvolatile processor across a variety of benchmark applications by 3X on average.

REFERENCES

[1] Jongouk Choi, Qingrui Liu, and Changhee Jung. 2019. CoSpec: Compiler Directed Speculative Intermittent Computation. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 399–412.

[2] Nathan Binkert et al. 2011. The Gem5 Simulator. *SIGARCH Comput. Archit. News* 39, 2 (Aug. 2011).

[3] Chris Lattner and Vikram Adve. 2004. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. In *Proceedings of the International Symposium on Code Generation and Optimization (CGO '04)*. IEEE Computer Society, Washington, DC, USA, 75–.

[4] 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 *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on*. IEEE, 1–12.

[5] Fang Su, Yongpan Liu, Yiqun Wang, and Huazhong Yang. 2017. A Ferroelectric Nonvolatile Processor with 46μ s System-Level Wake-up Time and 14μ s Sleep Time for Energy Harvesting Applications. *IEEE Transactions on Circuits and Systems I: Regular Papers* 64, 3 (2017), 596–607.