CoSpec: Compiler Directed Speculative Intermittent Computation

Jongouk Choi
Purdue University
choi658@purdue.edu

Qingrui Liu
Xilinx
qingruil@xilinx.com

Changhee Jung
Purdue University
chjung@purdue.edu

ABSTRACT

Energy harvesting systems have emerged as an alternative to battery-operated embedded devices. Due to the intermittent nature of energy harvesting, researchers equip the systems with nonvolatile memory (NVM) and crash consistency mechanisms. However, prior works require non-trivial hardware modifications, e.g., a voltage monitor, nonvolatile flip-flops/scratchpad, dependence tracking modules, etc., thereby causing significant area/power/manufacturing costs.

For low-cost yet performant intermittent computation, this paper presents CoSpec, a new architecture/compiler co-design scheme that works for commodity in-order processors used in energy-harvesting systems. To achieve crash consistency without requiring unconventional architectural support, CoSpec leverages speculation assuming that power failure is not going to occur and thus holds all committed stores in a store buffer (SB)—as if they were speculative—in case of mispeculation. CoSpec compiler first partitions a given program into a series of recoverable code regions with the SB size in mind, so that no region overflows the SB. When the program control reaches the end of each region, the speculation turns out to be successful, thus releasing all the buffered stores of the region to NVM. If power failure occurs during the execution of a region, all its speculative stores disappear in the volatile SB, i.e., they never affect program states in NVM. Consequently, the interrupted region can be restarted with consistent program states in the wake of power failure.

To hide the latency of the SB release—i.e., essentially NVM writes—at each region boundary, CoSpec overlaps the NVM writes of the current region with the speculative execution of the next region. Such instruction level parallelism gives an illusion of out-of-order execution on top of the in-order processor, achieving a speedup of more than 1.2X when there is no power outage. Our experiments on a set of real energy harvesting traces with frequent outages demonstrate that CoSpec outperforms the state-of-the-art scheme by 1.8∼3X on average.

ACM Reference Format:
Jongouk Choi, Qingrui Liu, and Changhee Jung. 2019. CoSpec: Compiler Directed Speculative Intermittent Computation. In *The 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-52), October 12–16, 2019, Columbus, OH, USA.* ACM, Columbus, OH, USA, 14 pages. https://doi.org/10.1145/3352460.3358279

MICRO-52, October 12–16, 2019, Columbus, OH, USA

ACM ISBN 978-1-4503-6938-1/19/10…$15.00
https://doi.org/10.1145/3352460.3358279

1 INTRODUCTION

Energy harvesting systems continue to grow at a rapid pace. The range of their applications is becoming wider including IoT devices, vehicle tire pressure sensors, health monitoring applications, unmanned vehicles, and so on [21, 35, 56, 57, 63, 69, 77]. The biggest challenge of energy harvesting systems is the instability of energy sources (e.g., Wi-Fi, solar, thermal energy, etc.) which causes unpredictable and frequent power outages [5, 9, 11, 26, 27, 30, 33, 35, 46, 58, 69].

Energy harvesting systems use a small capacitor as an energy buffer and intermittently compute only when enough energy is secured in the capacitor; when it is depleted, the systems die. This is so-called *intermittent computation* [44]. With the intermittent nature in mind, researchers adopt low-power microarchitecture with byte-addressable nonvolatile memory (NVM) and offer a crash consistency mechanism to checkpoint (backup) necessary data and restore them across power outages [4, 7, 13, 22, 45, 50–52, 54, 64, 73].

For the crash consistency, prior works including nonvolatile processors (NVP) approaches rely on voltage monitor based checkpoint schemes [8, 28, 43, 46, 47, 59, 71]. They checkpoint volatile registers—when the voltage monitoring system detects the voltage drop below a defined threshold—by using the buffered energy in the capacitor. In addition to the voltage monitor, the schemes require non-trivial hardware modifications such as nonvolatile flip-flops, that must be laid out next to volatile flip-flops for fast backup/restoration, special hardware checkpoint/controller logic, and additional capacitors for the voltage monitor.

Even worse, the voltage monitor may cause stability issues such as excessive leakage or crack of the capacitors leading not only to reduced capacitance [61, 67] but also to voltage detection delay with unexpected cold-start glitch [65]. To mitigate the issues, existing works aggressively increase the voltage threshold of the system wake-up/backup [1]. Consequently, they waste hard-won energy with making no forward progress until such a high voltage is secured to wake up the system for sure.

With that in mind, Hicks [22] proposes a voltage monitor free crash consistency scheme called Clank by implementing idempotent processing [73] in hardware. To identify idempotence violation due to antidependence—i.e., WAR (writer-after-read) dependence, Clank uses special dependence tracking hardware modules including several memory buffers that keep track of every load/store instruction. Once such an antidependent load-store pair is detected, Clank holds the store in the write-back buffer. If any of the buffers becomes full, Clank checkpoints all registers, flushes the write-back buffer to nonvolatile scratchpad, and copies the flushed data to NVM.

[1] Without the voltage monitor, the wake-up voltage can be set between 1∼1.8V [24, 43, 71], which is about 1.5∼3X lower than that of the state-of-the-art work [65, 72].

Unfortunately, there are two major problems with Clank. First, although nonvolatile scratchpad is used for performance reason, it is not clear how to realize nonvolatile yet fast SRAM with current and future technologies. One alternative is to exploit NVSRAM [75] (i.e., 3D stacking of SRAM over nonvolatile memory) which copies SRAM data to the underlying slow nonvolatile part right before power failure. However, this also requires the voltage monitor and the checkpointing/controller logic, rendering Clank vulnerable to the same voltage monitor issues. Second, Clank can cause a significant slowdown due to frequent overflows in the memory buffers; although using much larger buffers could mitigate the problem, it would increase the dependence tracking cost in terms of the resulting power and delay. According to the Clank paper [22], the performance overhead can be more than 20% even with the unrealistic assumption of the nonvolatile scratchpad.

In summary, prior works suffer from the high hardware cost, low energy efficiency, and run-time overhead. Thus, there is a compelling need for a practical crash consistency solution that works for commodity in-order processors without a significant slowdown.

To achieve low-cost yet performant intermittent computation, this paper presents CoSpec, a novel architecture/compiler co-design scheme that works for commodity in-order processors used in energy harvesting systems. To achieve crash consistency without requiring unconventional architectural support, CoSpec leverages speculation assuming that power failure is not going to occur and thus holds all committed stores in a store buffer (SB)—as if they were speculative—in case of mispeculation. 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.

In case power failure occurs during the SB release, which may lead to incorrect recovery, CoSpec proposes a *two-phase release*, i.e., first draining the SB entries to a proxy buffer allocated in NVM and then copying them to the primary main memory area in NVM. That way, either the buffer or the primary main memory can always be intact no matter when power is lost. 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 mis-speculated stores—committed before 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 effectively provides crash consistency, the region-based speculation window causes pipeline stalls at the end of each region due to the two-phase 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 two-phase 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 a speedup of more than 1.2X.

Finally, CoSpec can take advantage of direct memory access (DMA), another existing hardware component in commodity energy harvesting systems; a DMA module has already been adopted in MSP430 series microcontrollers, and it accelerates memory-to-memory copy in the FRAM-based nonvolatile memory [1, 2]. With the help of the DMA, CoSpec can not only accelerate the 2nd phase of the SB release, i.e., memory copy from the proxy buffer entries to primary main memory locations, but also improve the ILP efficiency, delivering significantly higher performance than all prior works!

Our contributions can be summarized as follows:

- Unlike prior works, CoSpec does not require any expensive hardware modifications. CoSpec's intelligent compiler-architecture interaction provides commodity microarchitecture with crash consistency, achieving truly recoverable intermittent computation at a low cost.
- CoSpec achieves high performance intermittent computation. The proposed ILP techniques allows CoSpec to effectively hide the long latency of NVM writes in a program. It turns out that the ILP is the **main** reason for CoSpec's high performance; CoSpec achieves speedups of 1.19X and 1.23X for non-DMA (i.e., ILP alone) and DMA cases, respectively. Overall, CoSpec outperforms the state-of-the-art nonvolatile processor by 11% on average and up to 26% when there is no power outage.
- In the context of voltage-monitor-free energy harvesting systems, CoSpec can decrease the wake-up voltage by 1.5∼3X compared to the state-of-the-art work [33, 65, 72], which leads to a much higher energy efficiency. The experimentation with frequent power outages demonstrates that CoSpec consumes 2∼3X less energy than the state-of-the-art work.

2 BACKGROUND AND MOTIVATION

2.1 Architecture for Energy Harvesting

Given the power-hungry nature of energy harvesting systems, prior works have opted for extremely low-power in-order processor architecture. Despite the low performance of the in-order processor, it is more suitable than power-consuming and complex out-of-order processor architecture [2]. Since power outages become the norm of program execution in energy harvesting systems, their processors should have byte-addressable nonvolatile memory (NVM) for efficient backup/recovery across the power outages. For example, TI's MSP430FR series of microcontrollers (MCU), which are popularly used in energy harvesting systems, have already integrated FRAM as main memory for their in-order processor core.

It is important to note that CoSpec targets the commodity MCU made up of FRAM-based nonvolatile main memory and conventional microarchitecture without cache as prior works [12, 22, 46, 71, 73, 74]. In the absence of cache, only data in a processor core, i.e., registers, are transient and will be lost when power is cut off. Thus, registers need to be checkpointed—i.e., saved in nonvolatile memory (NVM)—for their safe restoration in the wake of power failure.

2.2 Crash Consistency

For crash consistency, energy harvesting systems often makes a checkpoint to save the volatile registers into the NVM during program execution and rollback to the checkpointed states in the wake

[2] Some prior works propose to use out-of-order processors or even hybrid cores equipped with both in-order and out-of-order pipelines. However, they assume strong energy harvesting source that can deliver stable power for the out-of-order execution [48, 49].

of power failure. However, NVM alone cannot ensure correct recovery due to memory inconsistency in which data in NVM is corrupted across a power outage. Suppose a simple NVM data increment after checkpoint, i.e., '$checkpoint; i++$' where i is allocated in NVM and its initial value is 0. If the code is executed with no power failure, the expected value of i must be 1. However, if power is cut off after the increment, the system rollbacks to the point where the checkpoint were made and ends up re-executing the increment again. Since the i has already been updated before the power failure, the system leads to an unexpected output, i.e., $i = 2$. Thus, the memory inconsistency leads to incorrect recovery.

2.2.1 Software Based Approaches. Prior work called Ratchet [73] identifies and eliminates the memory inconsistency by using *idempotent processing* [15, 16, 36, 38–40]. As the cause of the memory inconsistency, Ratchet recognizes the non-idempotent memory access pairs comprised of a load and a subsequent store to the same memory location, i.e., antidependence also known as Write-After-Read (WAR) dependence. In the example of the NVM data increment '$i++$', the load and the subsequent store to i is such a non-idempotent memory access pair. To get around the antidependence, the Ratchet [73] compiler automatically partitions an input program to a series of idempotent regions so that each region has no antidependence. That way, any interrupted idempotent region can be safely re-executed without memory inconsistency when power comes back. Other prior works ask users to partition a program into tasks and leave a log of data in each task—before they become inconsistent due to antidependence—along with checkpointing volatile registers that are inputs to some later tasks [13, 45, 50]. Unfortunately, all the software based crash consistency schemes often cause a significant slowdown (50∼400% run-time overhead), which leads to the advent of hardware based approaches.

2.2.2 Hardware Based Approaches. To reduce the overhead and eliminate additional burdens to end-users, prior hardware approaches propose architectural support that relies on either (1) voltage monitoring system [8, 28, 43, 46, 47, 59, 71] or (2) hardware-implemented idempotent processing [22].

Voltage Monitoring Approaches. For hardware approaches to crash consistent energy harvesting including nonvolatile processors [43, 46, 47, 71], a common requirement among many prior works [8, 28, 43, 46, 47, 59, 71] is that they use a voltage monitor to ensure the sufficient energy—left in the capacitor—enough for taking a full checkpoint of volatile registers; the voltage monitor alerts the processor to take a checkpoint when the monitor estimates that the capacitor contains the energy large enough to complete the register file checkpointing. Similarly, the voltage monitor wakes up the processor when the sufficient energy is secured for successfully restoring the checkpointed values of the volatile registers. In this way, the voltage monitor systems can always make a checkpoint at the point right before power failure and resume from exactly the same point when power comes back. Since the systems never re-execute the code finished before power failure, they are free from the memory inconsistency.

However, this comes with an expensive hardware cost. In fact, the voltage monitor based approaches require non-trivial hardware modifications such as nonvolatile flip-flops, that must be laid out right next to volatile flip-flops for the value of registers to move fast in between on their backup/restoration, special hardware checkpoint/controller logic, and the additional capacitors necessary for estimating the voltage level.

Even worse, the voltage monitor approaches have a precision issue. In other words, it is very difficult to reliably estimate whether there is only sufficient energy left in the capacitor to take a checkpoint without power failure. That is mainly because the estimation requires predicting the future rate of charge-discharge. In fact, the capacitor can suffer from unstable discharge due to in-field aging problems such as cracks [61, 67]. Increased discharge rate can make the voltage monitor underestimate the future rate of charge dissipation. Unfortunately, this causes partial (unfinished) checkpointing, leading to incorrect recovery. Even though the voltage monitor may address the unpredictable discharge rate problem by pessimistically overestimating the discharge rate, this inevitably increases the checkpointing frequency, thereby hurting the performance, energy efficiency, and wear-out rate of the NVM.

To address the capacitor issues and inaccurate timing error of the voltage monitor, more recent works have developed precise and reliable voltage detection technologies [65]. However, to ensure safe checkpoint/recovery, they increase the wake-up/backup voltage thresholds of energy harvesting systems by almost 2∼3X compared to the previous generation of voltage monitors [43, 71]; the old voltage monitor approaches can start at about 1∼1.8V [25, 43, 71], but the recent works set the threshold up high as 2.7∼3V [33, 65, 72]. Consequently, they waste hard-won energy without making progress until the sufficient voltage is provided to wake up the system. Finally, they also found out that the backup/recovery controller could make a wrong decision on a reboot time—which causes a potential checkpoint (data) corruption problem—due to the cold-start voltage spark or an unexpected glitch [65].

Voltage Monitor Free Approaches. To address the above issues, Clank [22] proposes a voltage-monitor-free processor design by implementing the idempotent processing in hardware. In detail, Clank monitors all memory accesses (load/store) at run time with several memory buffers such as write-back, read-first, write-first, and address prefix buffers. By sweeping read-first and write-first buffers, Clank keeps track of antidependent load-store pairs that make it impossible to perform idempotent processing and thus lead to memory inconsistency in the wake of power failure.

In particular, once an antidependent store is detected, Clank holds it in the write-back buffer; non-antidependent stores are directly merged into NVM. If any of the buffers is about to overflow and unable to accommodate any further memory instruction (address), Clank alerts the processor to checkpoint all its registers, flushes the write-back buffer to nonvolatile scratchpad emptying out other buffers as well, and copies the flushed data eventually to nonvolatile main memory. Note that since it holds the antidependent store in the write-back buffer, Clank requires every load to check the write-back buffer first in case of the store-to-load forwarding.

Unfortunately, Clank suffers from two significant problems that prohibit its adoption. First, although Clank takes advantage of nonvolatile scratchpad—much faster than NVM—for performance reason, there is no current technology to realize nonvolatile yet fast SRAM in reality. Clank may leverage NVSRAM, a 3D stacking

based hybrid design of SRAM and NVM [75], which copies SRAM data to the slow nonvolatile part right before power failure. However, NVSRAM also requires the voltage monitor and the necessary checkpointing/controller logics, rendering Clank vulnerable to the same voltage monitor issues. Second, Clank may involve frequent checkpoints due to overflows in its memory buffers, thus degrading the performance significantly.

While Clank proposes to increase the size of the buffers for less overflows, it presents another—potentially more serious—problem in terms of the resulting hardware and energy costs. To a large extent, enlarging the buffers puts significant pressure on the design of CAM (content addressable memory) structure for Clank's associative searches of the buffers; in fact, the size of load/store queues has scarcely increased at all in the last decade for the same reason. Apart from the additional power consumption on the larger buffers, their wire delays might lead to significant energy consumption, possibly making Clank inappropriate for energy-harvesting systems.

With the reasonable size of the buffers, the performance overhead of Clank can be more than 20% even with the unrealistic assumption of the nonvolatile scratchpad [22]. With the deficiencies of all above prior works in mind, we seek to develop a practical crash consistency solution that works for commodity processors without a significant run-time overhead.

3 COSPEC APPROACH

CoSpec is a low-cost architecture/compiler co-design scheme that enables reliable crash consistency without significant energy and performance overheads. This section first presents the basic design of CoSpec: (1) hardware design, (2) compiler support, and (3) architecture/compiler co-design. The optimization techniques of CoSpec are deferred to Section 4.

3.1 CoSpec Hardware Design

The design philosophy of CoSpec is to leave the commodity microcontroller (MCU) architecture [2, 3, 53]—used in energy harvesting systems—almost as is and enable high performance intermittent computation without expensive hardware modifications.

3.1.1 Store Buffer for Power Failure Speculation. Store buffer has been adopted for other commodity in-order MCUs [53], e.g., ARMv8-A core implementations [3], mainly to handle mispeculation such as branch misprediction [3]. To achieve lightweight crash consistency, CoSpec proposes to exploit such a store buffer (SB) for a different type of speculation.

The difference is that CoSpec uses a region-level speculation window, guessing whether each region is likely to finish without interruption due to power failure. In other words, 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.

Note that this speculation approach never allows the stores of any regions being interrupted by power failure to be written to primary main memory (NVM). If power failure occurs, all buffered stores in the SB disappear because it is volatile. It is therefore impossible for the mis-speculated stores to affect NVM. Consequently, the interrupted region can be restarted with consistent program states in the wake of power failure. The takeaway is that speculative stores cannot be released to NVM until they become non-speculative, i.e., their region finishes without power failure. As a result, CoSpec can completely eliminate memory inconsistency without adding multiple non-trivial microarchitectural components required by nonvolatile processors and Clank.

It is important to note that CoSpec splits the store buffer into two parts to enable instruction level parallelism as will be shown in Section 4.1.1. During the program execution, any two consecutive recoverable code regions exclusively occupy one of the two parts in the SB. That is, each statically partitioned code region 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.

As the major challenge in achieving correct crash consistency, CoSpec should maintain failure atomicity of the *SB* draining; otherwise, any partial draining may result in the memory inconsistency problem [22, 65, 73]. To overcome the challenge, CoSpec leverages a 2-phase SB release mechanism; CoSpec first drains the committed stores from the *SB* to a proxy buffer in NVM, and then copies the drained results from the buffer to the primary main memory area in NVM. With the help of the 2-phase SB release, either the buffer or the main memory can always remain intact no matter when power is cut off. This will be discussed in Section for more details 3.3.1.

3.2 CoSpec Compiler

To partition the program into such regions, CoSpec compiler first counts the number of stores while traversing the control flow graph (CFG) of the program. 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 and inserts a checkpoint instruction to save them into a designated register file (RF) checkpoint storage in NVM. In particular, a PC register is saved at the end of each region—which serves as a recovery point in the wake of power failure—so that the forthcoming power failure will be recovered by restarting the next region.

Note that all inserted checkpoint instructions are normal store instructions. Thus, according to CoSpec's power failure speculation, they are first committed to the SB and then drained to NVM provided the speculation turns out to be successful. Indeed, the region formation is a tricky problem due to the circular dependence during the partitioning process. That is, the live-out register checkpointing essentially adds store instructions to a region, and of course the number of (added) stores determines the region boundary, which in turn affects the live-out registers provided the region boundary changes.

3.2.1 Region Formation. To conduct the region formation, we leverage the algorithm used in our own prior work [41]. In the following, we describe the high-level idea; for more details, readers are referred to the work [41].

[3] Currently, the processors used in energy-harvesting systems have no branch predictor as in MSP430 MCUs, NVPs (nonvolatile processors), Clank, and CoSpec.

CoSpec first partitions an input program into common program structures such as calls and loops. For this purpose, CoSpec places a region boundary at all the entry and exit points of functions. Likewise, a boundary is placed at the beginning of each loop header. Next, CoSpec identifies the basic block that has region boundaries in the middle of it, and splits it into separate basic blocks. This allows the region boundaries to always start at the beginning of basic blocks, which helps the next step to compute the initial checkpoint instructions. After finishing the initial region formation, CoSpec analyzes the regions to place live-out register checkpoints (i.e., store instructions).

Then, CoSpec compiler traverses the CFG in a topological order trying to combine those initial regions into larger regions as much as possible. The region combining can eliminate many checkpoints because the live-out registers of preceding region(s) are often no longer live after being combined with following regions. During the traversal of each control flow path, CoSpec updates the sum of current and incoming basic blocks' stores from the beginning of the latest region boundary along the path. If the sum becomes greater than a half of SB size (threshold) before the next region boundary is reached, CoSpec places a boundary to cut the region. After that, CoSpec compiler analyzes the re-partitioned regions again to insert live-out checkpoints and possibly repeats the re-partitioning process as long as there is a region that has more stores than the threshold.

In this way, it is guaranteed that the each partitioned region has at most as many stores as a half of the SB size, i.e., the threshold. It would be a mistake to take this to mean that all regions have exactly the threshold number of stores; rather many regions could have less stores than the threshold due to the re-partitioning process.

I/O Operations. To the best of our knowledge, to support non-recoverable operation such as I/O operation has remained as the open problem. That being said, since CoSpec compiler places a region boundary at function calls, the function that implements I/O operations is treated as a separate region—though it cannot be recovered due to the I/O operation. We believe that CoSpec can deal with I/O operations by simply checkpointing necessary status—just before each I/O operation starts—so that the interrupted I/O operation can be restarted in the wake of power failure.

3.3 Architecture/Compiler Co-design

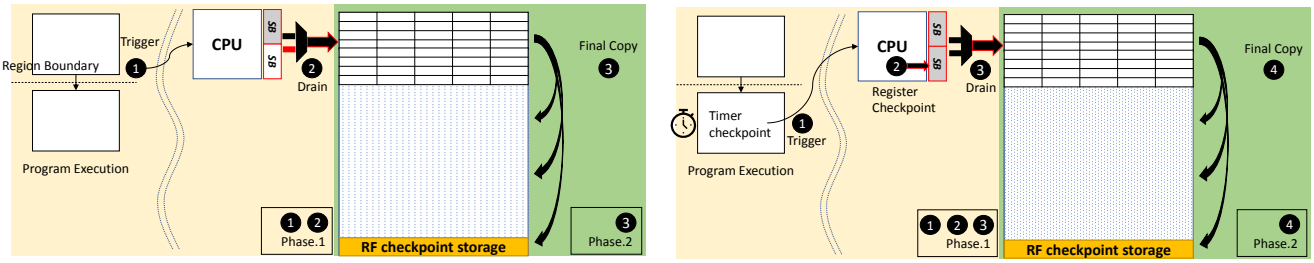
3.3.1 2-Phase Store Buffer Release Protocol. To achieve failure-atomic store buffer (SB) release, which is required for safe power failure recovery without memory inconsistency, CoSpec drains SB to NVM using a 2-phase mechanism. When each region is ended, i.e., program control reaches the end of each region boundary, CoSpec first drains the committed 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. Figure 1 describes how the 2-phase SB release protocol works for (a) a normal case (region based) and (b) an exceptional (watchdog timer based) case. First, in the normal checkpoint case, 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.

Second, CoSpec also supports an exceptional (watchdog timer based) case. In particular, if a compiler-partitioned region is excessively long, the system might be unable to make forward execution progress because of re-executing the interrupted region again and again across power outages; this paper calls this live lock like situation *stagnation*. To avoid the stagnation, CoSpec dynamically checkpoints registers to SB at the expiration of a watchdog timer—which can be adjusted at run time taking into account the dynamic power failure behaviors as will be shown in Section 4.2. Figure 1(b) shows how the dynamic checkpointing works with the 2-phase SB release. When the watchdog timer (❶) expires, CoSpec (❷) immediately checkpoints (stores) all registers and commits them to the **idle part of SB**–not used by the current region. CoSpec (❸) then drains full *SB* to the proxy buffer in NVM. When two parts of *SB* are completely drained, CoSpec (❹) makes the buffered data moved to the primary NVM locations in the same way as a normal case; note that, the watchdog timer is disabled during the 2-phase SB release process.

The 2-phase SB release mechanism protects both proxy buffer and the primary data in NVM—by managing a check bit for each—against the partial SB draining that may fail to recover from power failure. The first bit *isDrain* is devised for 'Phase 1' release, and it is set when the part of *SB*, which corresponds to the region being ended, is completely drained into the proxy buffer in NVM; in the exceptional case shown in Figure 1(b), the bit is set when both parts of SB are drained completely. The second bit *isComplete*—devised for 'Phase 2' release—is set when all the data in the proxy buffer are completely moved to primary main memory locations in NVM. These two check bits help CoSpec to restore correct data in the wake of power failure, and the next section discusses more details about the recovery protocol.

3.3.2 Recovery Protocol. CoSpec provides a safe recovery protocol to address potential memory inconsistency problem across power failures. There are three possible cases of power failure that differ in terms of their failure point in the timeline. First, a power failure can occur during *SB* draining (Phase 1 release), i.e., *isDrain* bit is not set. In this case, CoSpec simply ignores the SB data drained to the proxy buffer in NVM; the SB contents all disappear due to the volatility of the SB. To resume the interrupted region in the wake of the power failure, CoSpec first restores the saved register values including the recovery PC from the RF checkpoint storage in NVM and jumps to the PC. Note that it points to the beginning of the interrupted region at the moment. Although at the end of the region, a compiler-inserted checkpoint successfully saved a new recovery PC that points to the beginning of the next region, it was not written to neither the proxy buffer nor the RF checkpoint storage in NVM because of the power failure occurred during the 'Phase 1' release.

Second, a power failure can occur during the copy from the proxy buffer to the primary main memory in NVM (Phase 2 release). In this case, since the *isDrain* bit has been set, i.e., the recovery PC checkpoint at the end of the current region was successfully written to the proxy buffer, CoSpec does not rollback to the beginning of the current region. Instead CoSpec does redo the Phase 2 release, i.e., moving the proxy data to the primary main memory in NVM. Then, as usual, CoSpec restores the saved register values including



(a) Region-based Checkpoint (normal case)

(b) Timer-based Checkpoint (exceptional case)

Figure 1: CoSpec's checkpoint protocol for a normal case (a) and an exceptional case such as stagnation (b)

the recovery PC from the RF checkpoint storage in NVM and jumps to the PC for recovery.

Third, a power failure can occur outside of the 2-phase release, i.e., in the middle of a region, CoSpec recognizes such a case by checking the both bits, i.e., *isDrain* and *isComplete* are set. Here, the recovery process is simpler compared to the above two cases. CoSpec just restores the saved register values including the recovery PC from the RF checkpoint storage in NVM and jumps to the PC that should point to the beginning of the region interrupted by the power failure. The takeaway is that according to the status of the two check bits, CoSpec takes appropriate actions for correct recovery, thereby ensuring truly-recoverable intermittent computation no matter when power is lost and how often it occurs.

4 OPTIMIZATION

To avoid potential memory inconsistency, the 2-phase SB release mechanism requires double persistent writes for all stores. Unfortunately, this incurs significant performance overhead consuming hard-won energy—for such expensive NVM writes—that would otherwise could be used for making further forward execution progress. To address the overhead problem, CoSpec optimizes the 2-phase SB release by enabling instruction level parallelism (ILP). That is, 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. This section describes the implementation details of such an optimization: (1) how to reliably enable the ILP execution on an in-order processor without memory inconsistency and (2) how to adapt the ILP for the intermittent computation where the frequency of power outages varies.

4.1 Instruction Level Parallelism

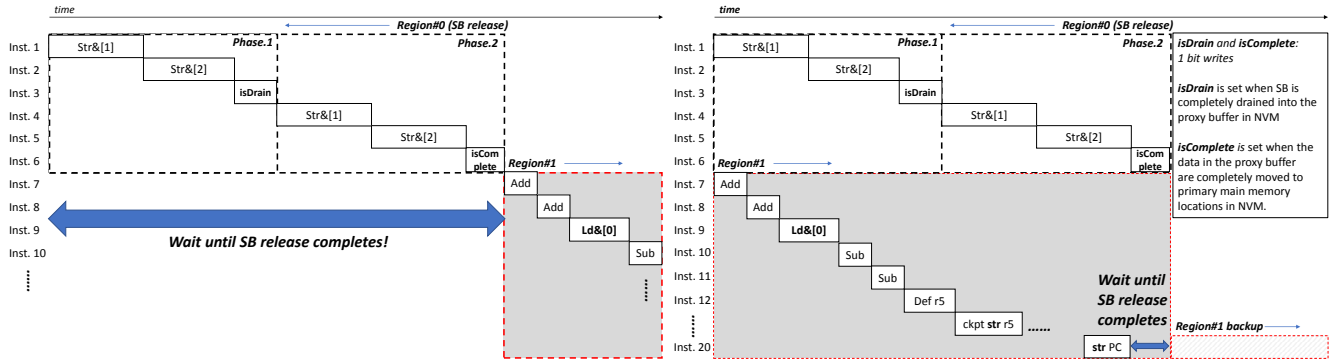
4.1.1 Enabling ILP execution. CoSpec enables instruction level parallelism to hide long NVM write latency by overlapping them with the next code region execution [4]. Figure 2 shows how the instruction level parallelism (ILP) works when two consecutive regions (i.e., Region#0 and Region#1) are executed. Figure 2(a) describes a non-ILP case; when *SB* starts its draining to NVM using the 2-phase release mechanism, the system needs to wait until the both phases finish to ensure the SB data is safely written to the primary memory. Since partial SB release can cause memory inconsistency, the power failure recovery might fail. Figure 2(b) shows how CoSpec hides such a long latency of NVM writes. By overlapping the NVM writes during entire 2-phase SB release with the speculative execution of the next code region, CoSpec is able to execute more instructions within a given time; as shown in Figure 2(b), the ILP approaches executes 13 more instructions than the non-ILP approach that encounters stalls at the instruction #6 due to the 2-phase SB release.

Note that once the speculative execution of Region#1 is completed, CoSpec should wait until the SB release of the Region#0 is finished rather than executing the next region (Region#2 not shown in the figure). This is necessary for achieving correct crash consistency. The next section shows how CoSpec solve this problem.

4.1.2 Achieving ILP without Breaking Correctness. There are a few challenges CoSpec must overcome to achieve the ILP optimization for correct recovery. First, CoSpec should avoid inserting a store to the *SB* during its draining, otherwise; it may incur the data hazard or race condition on the store buffer. To address this challenge, CoSpec lets each code region alternatively use a different part of *SB*. Recall that CoSpec splits the SB to two parts for exclusive use of any two neighboring code regions. For example, if a current region inserts its stores to one part of *SB*, then the next region inserts its stores to the other part of *SB*. That is, any two consecutive regions exclusively use a different part of *SB* all the time. However, it is still possible to insert a store to the same part of *SB*. For example, if the speculative region execution finishes too fast even before the 2-phase SB release of the previous region is completed, then executing the following region may overwrite data in the part of SB which is pending (being drained) for its 2-phase release. To avoid this problem, CoSpec conservatively waits at the end of the speculative region while the previous region's SB release is pending.

Second, load instructions should read the up-to-date data for correct execution. Suppose that a current load instruction needs to read data, but the required data is placed in the part of SB which is being drained. In this case, the load instruction should be stalled for correctness purpose. To avoid such a delay, both parts of SB must be available for correct execution. With that in mind, CoSpec does not invalidate the *SB* entries being drained until the program control reaches the end of the speculative region, i.e., the one following the prior region whose 2-phase SB release is pending. That way, the load of the speculative region can read any written data of the prior region from its part of the SB—which is being drained—without

[4] Similarly, TSO_ATOMICTY [70] leverages the overlapped region execution for atomic-region based dynamic optimizations. However, that is devised for multi-core out-of-order processors to achieve more thread interleaving. Also the store queue design and the region formation algorithm are different from those of CoSpec.



(a) CoSpec without ILP (b) CoSpec with ILP

Figure 2: Performance benefit thanks to ILP. DMA is not enabled here, though it can accelerate the 2nd phase of the SB release.

any stall. Of course, when the load in a region is to read the data written by the same region, its load can be served as usual using the conventional store-to-load forwarding through its own SB.

Discussion. One might argue that adding a SB in a simple in-order pipeline could reduce the core clock frequency as with modern processors where their SB must provide a dependent load with data within L1 hit time to avoid complicating their scheduling logic. However, we believe that CoSpec is free from this concern thanks to its architecture characteristics. Apart from the use of in-order pipeline and low clock frequency (∼25MHz) in energy harvesting systems, CoSpec does not have a cache (Section 2.1). The implication is that the SB search has only to finish within NVM (i.e., FRAM) access time. Note that this is always doable because each SB entry access is orders-of-magnitude faster than FRAM access latency. Consequently, CoSpec causes neither clock frequency reduction nor scheduling logic complication [5]. In addition, CoSpec can bypass SB searches for the majority of following loads; Section 4.3 details the SB bypassing and necessary compiler analysis.

4.2 Stagnation-Free Intermittent Computation

CoSpec should address the **stagnation** problem (Section 3.3.1), which would otherwise waste the harvesting energy in vain without making forward execution progress. To ensure the forward progress in the presence of frequent power outages, CoSpec proposes adaptive execution techniques [12, 29, 34] that take into account dynamic power failure behaviors.

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 during intermittent computation. In light this, CoSpec adaptively turns on/off the ILP and adjusts the speculation window according to the power failure patterns in a reactive manner.

When the system suffers from power failures, CoSpec first turns off the ILP execution. Then, if the power failure happens in the same region more than twice, which might be a sign of stagnation, CoSpec turns on the watchdog timer checkpoint. Once the timer is expired, CoSpec checkpoints registers to the store buffer (SB) and performs the 2-phase SB release as shown in Figure 1(b). Since the timer is set for it to be expired in the middle of the stagnating region, CoSpec can resume from the timer expiration point in the wake of power failure—rather than jumping back to the beginning of such a long region. If the region still encounter another power failure, CoSpec decreases the watchdog timer to a half of the previous value. This in effect doubles the frequency of the register checkpointing (and the 2-phase SB release) and can be repeated to get out of any long stagnating region across power outages.

On the other hand, if the system continues to make progress without a power outage in which case CoSpec assumes the system is under a good energy harvesting condition, then it enables ILP and disables the watchdog timer approach. With this simple adaptive execution heuristic, CoSpec can address the stagnation problem and improve the performance by spending more harvested energy for forward execution progress rather than wasting it for the re-executions of stagnating regions.

4.3 Energy-Efficient Store Buffer Search

In case of store-to-load forwarding, every load should consult the store buffer (SB). However, this involves expensive CAM (content addressable memory) based associative search in the SB. To address this issue, CoSpec (1) bypasses unnecessary SB searches and (2) designs a cost effective SB search logic.

First, CoSpec compiler statically checks if each load can be may- or must-aliased to stores in the current and previous regions by leveraging alias analysis [6, 31, 66, 68]. When no alias is found, CoSpec compiler marks the load instruction so that it can bypass the SB. During the program execution, if the processor detects such a special load instruction, it avoids the SB search and directly accesses to primary main memory.

To see the impact of this compiler-directed SB bypass scheme, we conducted measured how many load instructions could avoid SB searches at both compile time and run time. The experimental result demonstrates that a significant number of loads is able to bypass the SB search. As shown in Figure 3, at compile time, more than 80% of total load instructions can be marked to bypass the SB search on average by using both basic alias analysis (BasicAA) [31] and

[5] Technically, accessing 40 SB entries takes less than 1 cycle [18, 19]

advanced alias analysis called SVF (static value-flow analysis [66] [6]). At run time, 98∼99% of dynamic loads turn out to be from the SB search. That is mainly because many non-aliased loads are found in hot loops whereas aliased loads are not.

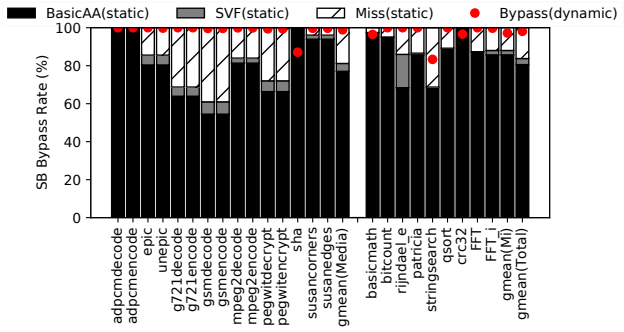


Figure 3: Store buffer bypass rates at compile time and run time. Both BasicAA and SVF are static alias analysis.

In particular, the promising results of high SB bypass rates motivate the different design of the SB search mechanism. In other words, CoSpec can afford a sequential search logic rather than the expensive CAM-based associative search. This gives a freedom to use the SB for energy harvesting system without worrying about the high power consumption required for the CAM search.

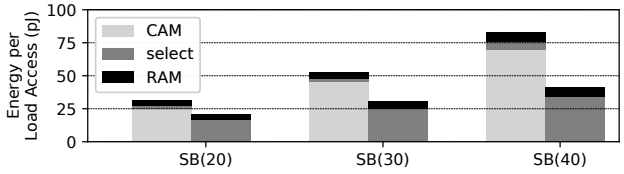


Figure 4: Energy consumption breakdown of different SB search schemes. For each SB configuration on x-axis, the first and second bars represent conventional CAM search and CoSpec's sequential search, respectively.

To this end, we estimated the energy consumption and performance of both conventional CAM-based associative search and sequential search by using CACTI [62] with 90nm technology [25] in the same way as prior work [19]. While the conventional associative SB search is comprised of three components, i.e., CAM, select logic, and RAM (buffer), CoSpec can remove the CAM part thanks to the sequential search. Figure 4 describes the energy consumption breakdowns of the CAM-based associative search and the sequential search. When the SB is 40, the energy consumption of the associative SB search is about 2X greater than the sequential search.

We also analyzed the latency overhead of CoSpec's sequential search compared to the CAM search. Figure 5 shows both the normalized access latency and the energy consumption overhead. The

[6]CoSpec compiler could run SVF—which is field- and flow-sensitive—successfully on top of program's region-based control flow sub-graph; while such an advanced analysis is very expensive for whole program analysis, our region-based (per-region) analysis makes it possible to run the SVF for all the benchmarks we tested.

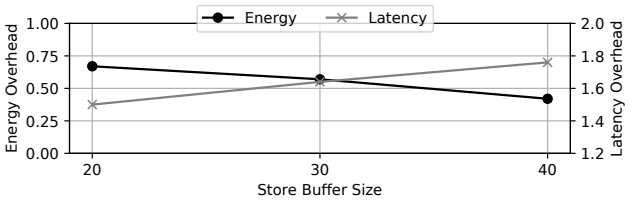


Figure 5: Normalized energy/latency overheads of the sequential SB search compared to the CAM based associative search

latency overhead is about 1.5∼1.8X when the store buffer size is 20∼40, while the energy consumption reduction is 40∼70%. Section 5 evaluates the impact of the both SB search schemes for various benchmark applications.

4.4 Direct Memory Access (DMA)

Although ILP execution can hide the long latency of the 2-phase SB release, it does not reduce the latency. To accelerate the SB release, CoSpec can opt for DMA processing available in commodity energy harvesting microcontrollers (MCUs), e.g., MSP430 series. In fact, the DMA engine of MSP430 MCUs [2] can speed up NVM data transfer, i.e., memory-to-memory copy, by ≈4X faster then normal read-write based copy [1]. In light of this, CoSpec can use the DMA to accelerate the second phase (i.e., Phase 2 data copy shown in Figure 1) of the 2-phase SB release.

However, care must be taken to perform the DMA processing because every data in the proxy NVM buffer needs to be copied to the corresponding primary main memory locations in a precise manner. Currently, CoSpec uses a single DMA channel multiple times in a row. That is, the number of DMA operations is the same as the number of the proxy buffer entries to be copied. Although a series of DMA copies seem to be not optimized, the DMA processing is still helpful thanks to its 4X faster NVM copy. It is important to note that due to the DMA processing can improve the ILP efficiency as will be shown in Section 5.2.1. That is because the prevention of the SB race condition lets the ILP mechanism conservatively wait at the end of the speculative region for the previous region to complete its 2-phase release (See Section 4.1.2).

5 EVALUATION

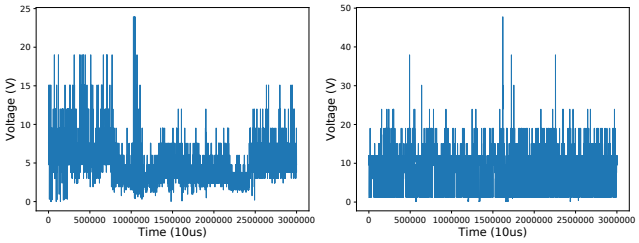
We implemented CoSpec compiler techniques described in Section 3.2 using the LLVM compiler infrastructure [31]. All the experiments were performed on the gem5 simulator [10] with ARM ISA, modeling a single core 3-stage in-order pipeline as in NVP simulator [18]. We compared CoSpec to nonvolatile processor (NVP) [65], i.e., the state-of-the-art NVFF based checkpoint scheme, using the mixture of Mediabench and MiBench applications [20, 32, 42]. They were all compiled with standard -O3 optimization. As a default configuration, CoSpec uses the *SB* size of 40 entries with the sequential search logic (Section 4.3) [7]. Table 1 describes the hardware specifications of the baseline NVP and CoSpec.

[7]Since the target microcontroller [4] has 16 registers, the SB size must be at least two times bigger than the register file size (16) to safely enable the watchdog timer based checkpoint scheme shown in Figure 1(b).

	NVP	CoSpec
Capacitor	100nF	100nF/No
Computing Power	100uW/MHz	100uW/MHz
Voltage Monitor(VM)	18uA	No
Store Buffer	No	Yes (Section 4.3)
DMA	No	Optional
Von/Voff	3.3/2.8	1.8/1.8
Ckpt/Restore V	3.1/2.9	No/1.8
Write/Read (latency)[8]	120ns/20ns	120ns/20ns
Write/Read (power)	2mW	2mW
Sleep/Wakeup T	46/14us	212/310us[71]
Recovery Point	VM hit	Boundary
ILP	No	Yes

Table 1: Simulation configuration

To evaluate CoSpec for harsh environment with frequent power outages, we used two power traces of the NVP simulator which were collected from real RF energy-harvesting systems [18]. Figure 6 describes the shape of the two power traces: (a) home and (b) office. In the following, we provide the detailed analyses of CoSpec on (1) hardware cost, (2) execution time with and without power failure, and (3) energy consumption breakdown.



(a) Power trace#1 (Home) **(b) Power trace#2 (Office)**

Figure 6: Energy harvesting voltage traces. Trace#1 and#2 incur ≈20 and ≈400 power outages in every 30 seconds, respectively.

5.1 Hardware Cost Analysis

Schemes	NVP [65]	Clank [22]	TCCP [37]	CoSpec
Core Type	InO	InO	OoO	InO
Buffers	No	4 buffers	*SB*	*SB*
DMA	No	No	No	Optional
ISA Change	No	Yes	Yes	Optional
Double Backup	No	Yes	No	Yes
Counter(s)	No	No	**Yes (2)**	No
NV Scratchpad	No	**Yes**	No	No
NVFF	**Yes**	No	**Yes (+NVSB)**	No
Extra Energy Buffer	**Yes**	No	**Yes**	No
Voltage Monitor	**Yes**	No	**Yes**	No
Total Cost	High	High	High	Low

Table 2: Hardware cost comparison: In the first column, the entries in bold are non-commodity hardware components, i.e., the bold marks represent expensive hardware modifications. Others have already been adopted to commodity hardware designs.

This section analyzes the hardware cost of prior works [22, 37, 65] and highlights the low cost of CoSpec. Table 2 provides the major hardware cost comparison. First, NVP [65] requires the voltage monitor, NVFF, and extra energy buffer. The voltage monitor consumes a significant amount of energy and occupies a nontrivial portion of die size [22][9]. Also, integrating the NVFF (nonvolatile flip-flops), that must be laid out in close proximity to the volatile flip-flops, in the core microarchitecture is complex and expensive due to the manufacturing cost. Overall, the hardware cost of NVP is high. Second, Clank [22] introduces new hardware components such as nonvolatile scratchpad and idempotence violation (i.e., antidependence) detector with several memory buffers. Since the dependence tracking has to monitor every single load/store and sweep the buffers for CAM based associative searches, it is fair to say that the total cost of Clank is high. Third, TCCP [37], a variant of NVP, builds up an out-of-order processor. As with NVP, TCCP requires the voltage monitor, extra energy buffer, and NVFF. In addition, TCCP introduces a nonvolatile store buffer (NVSB) as well as two threshold counters and their controller logic for varying the checkpoint interval. Given all this, TCCP is another high cost approach.

Finally, CoSpec re-purposes the existing *SB* and introduces its 2-phase release logic. Other than that, CoSpec does not modify core microarchitecture unlike above prior works. Although CoSpec currently assumes a special load instruction for bypassing the SB, this can be done without ISA change. The idea is to (1) set the most significant bit of the aliased load address operand—which must be zero due to the word granularity—and (2) let the pipeline architecture check the bit to reset it and enable the SB bypassing. Although the bit setting instruction must be inserted at compile time, the overhead will not be significant thanks to the small portion of aliased loads as shown in Figure 3. Although CoSpec can opt for a DMA engine, it has already been adopted by commodity in-order processors such as MSP430 series MCUs. Overall, the hardware cost of CoSpec is significantly lower than that of the prior works.

5.2 Execution Time Analysis with No Outage

To analyze the execution time of CoSpec, we first set the baseline to the state-of-the-art NVP [65] with uninstrumented binaries. We measured the execution time of CoSpec for 24 benchmark applications with 6 configurations.

First, we analyzed the performance impact of the alias analysis based SB bypass by turning it off (NoAA) and on (AA). As shown in Figure 7, without the *SB* bypass (NoAA), i.e., the first bar in the figure, CoSpec incurs about 24% execution time overhead due to the region-based power failure speculation overheads such as the 2-phase SB release and the inserted register checkpoints. When the SB bypass is enabled (AA), i.e., the second bar in the figure, the resulting execution time reduction is only marginal. This implies that the SB search is not the main source of the execution time overhead.

Second, we also analyzed the impact of DMA and ILP on the execution time of applications. Recall that DMA is used for fast memory-to-memory copy, and therefore it can only speed up the second phase of the SB release. When both SB bypass and DMA are enabled (AA+DMA), i.e., the third bar in Figure 7, CoSpec causes about 10% execution time overhead; as with MSP430 microcontrollers, we set the DMA speed to 4X faster then normal memory copy as default. When both SB bypass and ILP are enabled

[8]We configured the NVM write/read latency based on the commodity design [4] and the state-of-the-art works [18, 76]

[9] The die area occupied by the commodity voltage monitor is about $0.3mm^2$ [65].

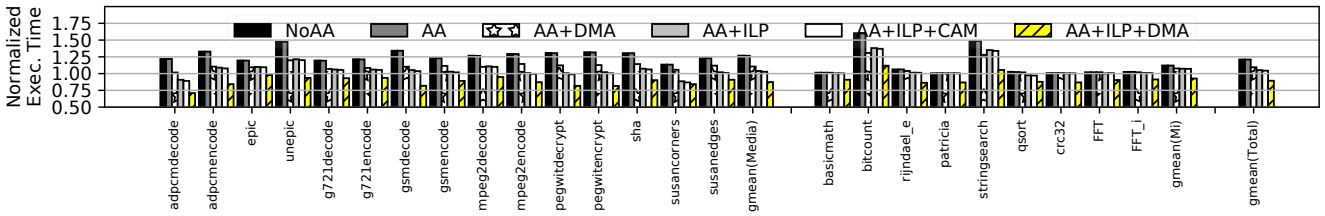


Figure 7: Normalized execution time of CoSpec compared to NVP [65]. As a default, CoSpec enables SB bypass, ILP, and DMA support for all other experiments

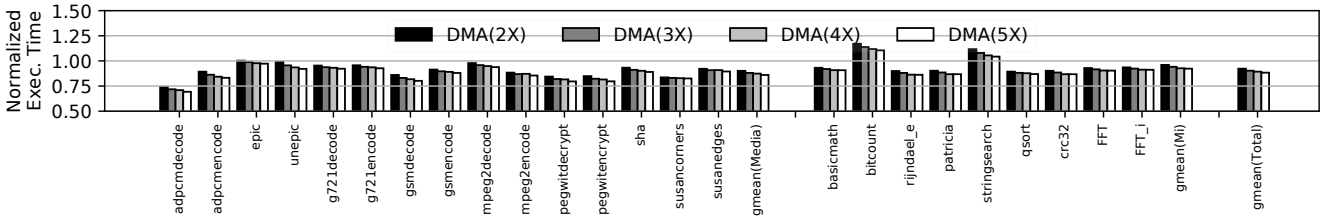


Figure 8: Normalized execution time of CoSpec compared to NVP [65] varying DMA speed. DMA(4X) is the default configuration for all other experiments.

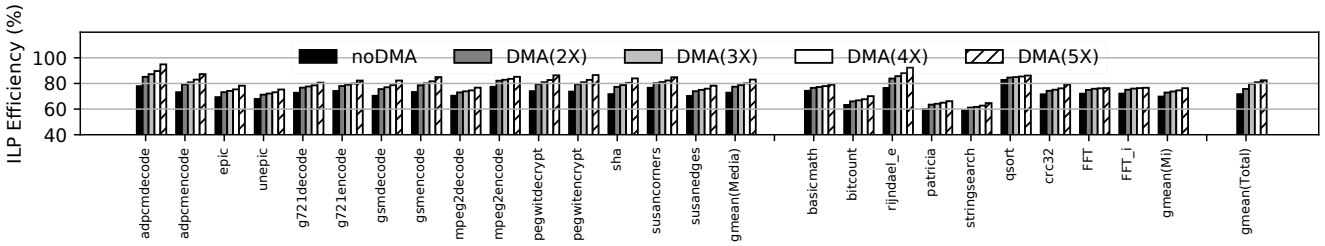


Figure 9: ILP Efficiency comparison varying DMA speed. DMA(4X) is the default configuration for all other experiments

(AA+ILP), i.e., the fourth bar in the figure, the resulting execution time overhead is only 4∼5% though DMA is not enabled. This confirms that ILP is the main reason for CoSpec's high performance.

Finally, we enabled all the optimizations to see the performance bound of CoSpec. When the best configuration is set (AA+ILP+DMA), i.e., the sixth bar in the figure, CoSpec rather outperforms the state-of-the-art NVP by 11% on average. As the next section shows, the use of DMA is able to improve the ILP efficiency. In this way, CoSpec can effectively hide the long latency of NVM writes involved in the 2-phase SB release.

Interestingly, Figure 7 shows that CAM search does not make a huge impact on the execution time on average. When the CAM search is enabled with both SB bypass and ILP (AA+ILP+CAM), i.e., the fifth bar in the figure, there is only marginal difference compared to the sequential search with SB bypass and ILP (AA+ILP). That is because 1∼2% of total loads access to the store buffer—as shown in Figure 3—thanks to the precise alias analysis of CoSpec's compiler. That is, only a few loads could get the CAM search benefit. Note that all the other bars except for AA+ILP+CAM in Figure 7 use the sequential SB search logic.

5.2.1 Sensitivity Analysis. We explored the performance impact of the DMA and NVM technology with the highly optimized CoSpec (AA+ILP+DMA). First, we varied the DMA speed of data transfer in NVM, i.e., 2X, 3X, and 5X faster than a normal NVM copy—and then measured the resulting execution times of NVP and CoSpec for the same set of benchmark applications.

Figure 8 shows the normalized execution time of CoSpec compared to NVP which is the same baseline used in the prior experiment. To a large extent, CoSpec becomes faster as the DMA speed is increased. When the DMA speed is 5X, CoSpec can achieve ∼6% speedup than the default speed of 4X.

To further analyze the correlation between the DMA speed and ILP execution, we measured the ILP efficiency varying the DMA speed. The ILP efficiency is defined as how much the time taken for the 2-phase SB release of a code region is overlapped with the execution time of the next region. For example, if the SB release time is completely overlapped with the next region execution, the ILP efficiency is 100%. Note that the perfect efficiency is achieved when the region execution time is greater than or equal to the SB release time; either way, the SB release time is fully hidden, the ILP efficiency is 100%. On the other hand, if the next region finishes while the SB release is still pending, the ILP efficiency is decreased.

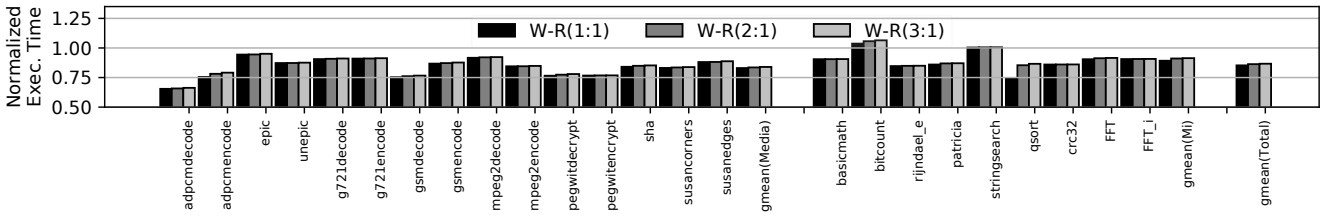
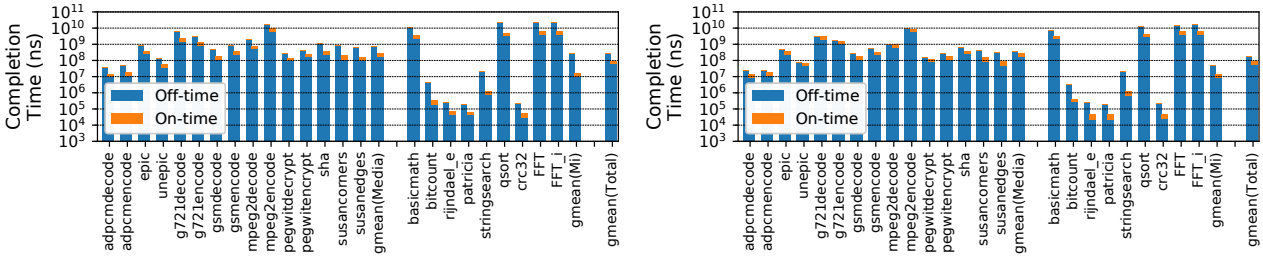


Figure 10: Normalized execution time of CoSpec compared to NVP [65] varying the write-to-read ratio of NVM. The ratio, 6:1, is the default configuration for all other experiments.



(a) Power trace#1 (Home) **(b) Power trace#2 (Office)**

Figure 11: Completion time comparison. The 1st/2nd bars of each application represent the times of NVP and CoSpec, respectively.

That is because CoSpec must wait—at the end of the next region—for the SB release to finish. As shown in Figure 9, 70∼82% of the 2-phase SB release can be overlapped with the next region execution when the DMA speed is 2X∼5X.

Memory	FRAM [2]	NVsim [17]	PCM [23, 55]	Re-RAM [76]
Ratio	1:1	2:1	3:1	6:1

Table 3: Write-to-read ratios of different NVM technologies.

Second, we varied the NVM write/read latency ratio, i.e., 1:1, 2:1, and 3:1, assuming different NVM technologies[17, 23, 55, 76] shown in Table 3. Figure 10 shows the normalized execution times of CoSpec compared to the same baseline NVP again. On average, CoSpec outperforms the NVP by about 13∼16% when the NVM write/read ratio becomes 3∼1:1.

5.3 Execution Time Analysis with Outages

To test the ability to make forward execution progress in the presence of a myriad of power outages, we measured the completion time of benchmark applications using two voltage traces shown in Figure 6; they are collected from a real RF-based energy harvesting system when it is deployed in home (a) and office (a). Figure 11 shows the completion time of the baseline NVP (the first bar) and CoSpec (the second bar) with breaking down the time to 2 parts, i.e., power-off-time and power-on-time. As shown in the figure, the system off-time dominates the completion time of both NVP and CoSpec. However, NVP is designed to wake up at 1.5∼3X higher voltage level than the minimum supply voltage of MCUs, due to voltage monitor issues (see Section 2.2.2). This implies that NVP should stay in a sleep mode for a substantial amount of time without making forward progress. Unlike the NVP, CoSpec can start to operate once the minimum supply voltage is secured, thus achieving further forward progress. Figure 11 (a) and (b) highlights that CoSpec outperforms the NVP by 3.0X and 1.8X in the trace#1 (a) and trace#2 (a), respectively.

Interestingly, the NVP makes further forward execution progress in trace#1 than trace#2. As shown in Figure 11 (b), NVP's completion time using trace#1 is only 60% of that of using trace#1. Given that trace#2 has relatively less power outages than trace#1, NVP tends to prefer more reliable voltage trace. With that in mind, we expect that CoSpec can outperform the NVP more significantly when the energy source is more unreliable.

5.4 Energy Breakdown with Outages

Finally, we analyzed the average energy consumption breakdown across power outages [10]. The total energy consumption can be divided into two parts: one under ILP execution and the other under non-ILP execution. The ILP part is further broken down to successful- and mis-speculation, each of which is comprised of 3 parts: the Phase1/Phase2 of the SB release and the computation. On the other hand, the non-ILP part is two-fold: NoILP and re-execution. NoILP is simply the energy consumption of CoSpec when it executes without ILP excluding that of re-executing any interrupted regions.

Figure 12 indicates that the overhead of CoSpec mostly comes from the re-execution cost—i.e., Re-exec in the figure. Although CoSpec enables the ILP and the watchdog-timer-based checkpoint in an adaptive manner according to a dynamic power failure pattern, the adaptation may not help for the first time to make progress (see Section 4.2). For example, to avoid stagnation, CoSpec might need

[10]CoSpec shows similar energy consumption trends in both power traces. On average, the total energy consumption of CoSpec in the presence of power failures using two traces is 2∼3X less than the NVP's.

to perform multiple times of the adaptation in a reactive manner (involving the sequence of ILP off -> watchdog timer on -> timer halving). Thus, the re-execution consumes the harvested energy without making actual progress until CoSpec finally gets out of the stagnating region after the multiple adaptations. As shown in Figure 12, the re-execution consumes 40% of the total energy on average.

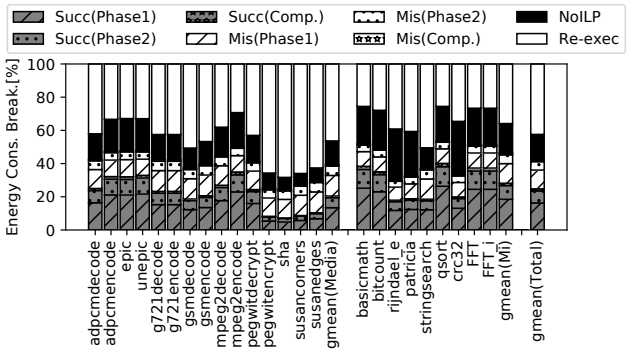


Figure 12: Energy consumption breakdown of CoSpec

On average, CoSpec consumes about 40% of its total harvested energy for ILP executions while it does the rest of the energy for non-ILP executions. NoILP consumes 20% of total energy on average due to the adaptation of CoSpec which throttles down its execution to escape (potentially) stagnating regions. In particular, extra NVM writes (i.e., the Phase2 of the SB release during successful- and mis-speculation) account for 18% of the total energy consumption on average. Also, it turns out that the wasted energy of mis-speculated execution (i.e., computation during mis-speculation) is negligible thanks to CoSpec's adaptive execution.

6 OTHER RELATED WORK

The problem of ensuring data consistency and improving the forward progress of an intermittently powered system is at the heart of energy harvesting computing. A variety of different techniques have been proposed, but they end up introducing non-traditional microarchitecture modifications or incurring significant performance overheads. In contrast, CoSpec can achieve high performance speculative intermittent computation for commodity energy-harvesting microcontrollers without expensive modifications.

Ma *et al* propose so-called incidental computing [47]. This scheme attempts to trade off the output quality of a program to improve forward execution progress. The key observation is that input data for many signal/image processing applications are mostly free from data dependencies, and the applications often contain memory independent loop iterations that could be skipped over in their entirety. With that in mind, the scheme ignores some computations interrupted due to power failure. When power comes back, it starts to process the most recent data, producing an earlier output at the expense of the quality degradation. Unfortunately, the scheme requires user-intervention, e.g., programmers must mark the skip points, recovery points, and so on. Unlike the scheme, CoSpec does not require user intervention at all but can still achieve the same goal, i.e., improving forward progress, without compromising the output quality.

Ma *et al* also suggest a hybrid processor equipped with both in-order and out-of-order pipelines to adapt the processor execution to underlying energy harvesting condition [48]. This scheme chooses one of the pipelines according to run-time power failure behaviors. Unfortunately, the scheme does not only require expensive hardware modifications due to the hybrid processor design but also cause additional switching delay and energy consumption when it turns on/off the hardware resources. In contrast, CoSpec builds on top of the current commodity in-order microarchitecture without expensive hardware modifications. Nevertheless, CoSpec can achieve further forward execution progress with the help of speculative intermittent computation.

Colins *et al* propose a reconfigurable energy buffer with multiple capacitor banks for energy efficiency [14]. Since each application may require a different amount of energy, the authors attempt to reconfigure the energy capacity to match the application's demand by switching on/off some part of the banks. However, the energy buffer can suffer from unstable discharge due to in-field aging problems or environmental changes. Thus, the scheme may be unreliable especially when energy harvesting systems are deployed in harsh environment. While the capacitor reconfiguration techniques are orthogonal to those of CoSpec, it can achieve a truly reliable crash consistency without the custom capacitor logic and the additional hardware support.

Ruppel *et al* devise an event-driven transactional concurrency control scheme. To provide a transaction, that can include a set of tasks, with failure-atomicity, the authors leverage interrupt service routines [60]. For this purpose, program tasks must be delineated in the first place and encapsulated in a transaction and an asynchronous event. Apart from this burden placed on programmers, the scheme causes a significant performance overhead. In contrast, CoSpec is an automated compiler/architecture co-design scheme that enables high-performance intermittent computation.

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 takes advantage of power failure speculation to enable crash consistency without significant hardware and performance overheads. In particular, 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.

ACKNOWLEDGMENTS

We appreciate anonymous reviewers for their constructive comments. This work was supported by NSF grants 1750503 (CAREER Award) and 1814430.

REFERENCES

[1] 2015. Maximizing Write Speed on the MSP430 FRAM. http://www.ti.com/mcu/docs/litabsmultiplefilelist.tsp?sectionId=96&tabId=1502&literatureNumber=slaa498b&docCategoryId=1&familyId=5012. Accessed: 2018-10-14.

[2] 2016. MSP430FR5994LaunchPad Development Kit (MSPEXP430FR5994). http://www.ti.com/lit/ug/slau678a/slau678a.pdf. Accessed: 2017-11-08.

[3] 2017. ARM Research Starter Kit: System Modeling Using gem5. https://raw.githubusercontent.com/arm-university/arm-gem5-rsk/master/gem5_rsk.pdf. Accessed: 2018-11-18.

[4] 2017. MSP430FR59xx Mixed-Signal Microcontrollers (Rev. F). http://www.ti.com/lit/ds/symlink/msp430fr5969.pdf. Accessed: 2017-11-08.

[5] Henko Aantjes, Amjad Y Majid, and Przemysław Pawełczak. 2016. A Testbed for Transiently Powered Computers. *arXiv preprint arXiv:1606.07623* (2016).

[6] Lars Ole Andersen. 1994. *Program analysis and specialization for the C programming language*. Ph.D. Dissertation. University of Cophenhagen.

[7] Sara S Baghsorkhi and Christos Margiolas. 2018. Automating efficient variable-grained resiliency for low-power IoT systems. In *Proceedings of the 2018 International Symposium on Code Generation and Optimization*. ACM, 38–49.

[8] Domenico Balsamo, Alex S Weddell, Anup Das, Alberto Rodriguez Arreola, Davide Brunelli, Bashir M Al-Hashimi, Geoff V Merrett, and Luca Benini. 2016. Hibernus++: a self-calibrating and adaptive system for transiently-powered embedded devices. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 35, 12 (2016), 1968–1980.

[9] S. Beeby and N. White. 2014. *Energy Harvesting for Autonomous Systems*. Artech House, Incorporated. https://books.google.fr/books?id=7H9xdFd4sikC

[10] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. 2011. The Gem5 Simulator. *SIGARCH Comput. Archit. News* 39, 2 (Aug. 2011).

[11] Bradford Campbell, Branden Ghena, and Prabal Dutta. 2014. Energy-harvesting thermoelectric sensing for unobtrusive water and appliance metering. In *Proceedings of the 2nd International Workshop on Energy Neutral Sensing Systems, ENSsys '14, Memphis, Tennessee, USA, November 6, 2014*. 7–12. https://doi.org/10.1145/2675683.2675692

[12] Jongouk Choi, Hyunwoo Joe, Yongjoo Kim, and Changhee Jung. 2019. Achieving Stagnation-Free Intermittent Computation with Boundary-Free Adaptive Execution. In *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 331–344.

[13] Alexei Colin and Brandon Lucia. 2015. Chain: Tasks and Channels for Reliable Intermittent Programs.. In *In Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*. ACM, 514–530.

[14] Alexei Colin, Emily Ruppel, and Brandon Lucia. 2018. A Reconfigurable Energy Storage Architecture for Energy-harvesting Devices. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 767–781.

[15] Marc de Kruijf and Karthikeyan Sankaralingam. 2013. Idempotent code generation: Implementation, analysis, and evaluation. In *Code Generation and Optimization (CGO), 2013 IEEE/ACM International Symposium on*. IEEE, 1–12.

[16] Marc A. de Kruijf, Karthikeyan Sankaralingam, and Somesh Jha. 2012. Static Analysis and Compiler Design for Idempotent Processing. In *Proceedings of the 33rd ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '12)*. ACM, New York, NY, USA, 475–486. https://doi.org/10.1145/2254064.2254120

[17] Hussein Elnawawy, Mohammad Alshboul, James Tuck, and Yan Solihin. 2017. Efficient Checkpointing of Loop-Based Codes for Non-Volatile Main Memory. In *2017 26th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. IEEE, 318–329.

[18] Yizi Gu, Yongpan Liu, Yiqun Wang, Hehe Li, and Huazhong Yang. 2016. NVPsim: A simulator for architecture explorations of nonvolatile processors. In *Design Automation Conference (ASP-DAC), 2016 21st Asia and South Pacific*. IEEE, 147–152.

[19] Erika Gunadi and Mikko H Lipasti. 2007. A position-insensitive finished store buffer. In *Computer Design, 2007. ICCD 2007. 25th International Conference on*. IEEE, 105–112.

[20] 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 *Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop on*. IEEE, 3–14.

[21] Josiah Hester, Kevin Storer, Lanny Sitanayah, and Jacob Sorber. 2016. Towards a Language and Runtime for Intermittently-Powered Devices. *sleep* 9 (2016), 10.

[22] Matthew Hicks. 2017. Clank: Architectural Support for Intermittent Computation. In *In Proceedings of ISCA âĂŹ17*. ACM.

[23] Xing Hu, Matheus Ogleari, Jishen Zhao, Shuangchen Li, Abanti Basak, and Yuan Xie. 2018. Persistence Parallelism Optimization: A Holistic Approach from Memory Bus to RDMA Network. In *Proceedings of the 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*.

[24] Texas Instruments. 2015. MSP430FR family of Ultra Low-Power Microcontrollers. http://www.ti.com/lsds/ti/microcontrollers_16-bit_32-bit/msp/ultra-low_power/msp430frxx_fram/what_is_fram.page.

[25] Texas Instruments. 2017. MSP430FR59xx Mixed-Signal Microcontrollers. http://www.ti.com/lit/ds/symlink/msp430fr5969.pdf.

[26] Hrishikesh Jayakumar, Kangwoo Lee, Woo Suk Lee, Arnab Raha, Younghyun Kim, and Vijay Raghunathan. 2014. Powering the Internet of Things. In *Proceedings of the 2014 International Symposium on Low Power Electronics and Design (ISLPED '14)*. ACM, New York, NY, USA, 375–380. https://doi.org/10.1145/2627369.2631644

[27] 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 *VLSI Design*. IEEE Computer Society, 330–335. http://dblp.uni-trier.de/db/conf/vlsid/vlsid2014.html#JayakumarRR14

[28] Harishankar 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 *VLSI Design and 2014 13th International Conference on Embedded Systems, 2014 27th International Conference on*. IEEE, 330–335.

[29] Changhee Jung, Daeseob Lim, Jaejin Lee, and SangYong Han. 2005. Adaptive execution techniques for SMT multiprocessor architectures. In *Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming*. ACM, 236–246.

[30] Bryce Kellogg, Vamsi Talla, Shyamnath Gollakota, and Joshua R Smith. 2016. Passive Wi-Fi: Bringing Low Power to Wi-Fi Transmissions.. In *NSDI*, Vol. 16. 151–164.

[31] 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–.

[32] Chunho Lee, Miodrag Potkonjak, and William H. Mangione-Smith. 1997. MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communicatons Systems. In *Proceedings of the 30th Annual ACM/IEEE International Symposium on Microarchitecture (MICRO 30)*. IEEE Computer Society, Washington, DC, USA, 330–335. http://dl.acm.org/citation.cfm?id=266800.266832

[33] Hyung Gyu Lee and Naehyuck Chang. 2015. Powering the IoT: Storage-less and converter-less energy harvesting. In *Design Automation Conference (ASP-DAC), 2015 20th Asia and South Pacific*. IEEE, 124–129.

[34] 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 Distrib. Comput.* 70, 5 (May 2010), 467–480.

[35] Woo Suk Lee, Hrishikesh Jayakumar, and Vijay Raghunathan. 2014. When they are not listening: Harvesting power from idle sensors in embedded systems. In *Proceeding of the 5th International Green Computing Conference (IGCC),*.

[36] 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 *51st Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2018, Fukuoka, Japan, October 20-24, 2018*. 258–270. https://doi.org/10.1109/MICRO.2018.00029

[37] Qingrui Liu and Changhee Jung. 2016. Lightweight hardware support for transparent consistency-aware checkpointing in intermittent energy-harvesting systems. In *Non-Volatile Memory Systems and Applications Symposium (NVMSA), 2016 5th*. IEEE, 1–6.

[38] Qingrui Liu, Changhee Jung, Dongyoon Lee, and Devesh Tiwari. 2015. Clover: Compiler Directed Lightweight Soft Error Resilience. In *Proceedings of the 16th ACM SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems 2015 CD-ROM (LCTES'15)*. ACM, New York, NY, USA, Article 2, 10 pages. https://doi.org/10.1145/2670529.2754959

[39] Qingrui Liu, Changhee Jung, Dongyoon Lee, and Devesh Tiwari. 2016. Compiler-directed lightweight checkpointing for fine-grained guaranteed soft error recovery. In *High Performance Computing, Networking, Storage and Analysis, SC16: International Conference for*. IEEE, 228–239.

[40] 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 Transactions on Embedded Computing Systems (TECS)* 16, 2, 32.

[41] 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.

[42] Qingrui Liu, Xiaolong Wu, Larry Kittinger, Markus Levy, and Changhee Jung. 2017. Benchprime: Effective building of a hybrid benchmark suite. *ACM Transactions on Embedded Computing Systems (TECS)* 16, 5s (2017), 179.

[43] Yongpan Liu, Zewei Li, Hehe Li, Yiqun Wang, Xueqing Li, Kaisheng Ma, Shuangchen Li, Meng-Fan Chang, Sampson John, Yuan Xie, et al. 2015. Ambient energy harvesting nonvolatile processors: from circuit to system. In *Proceedings of the 52nd Annual Design Automation Conference*. ACM, 150.

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

[45] Brandon Lucia and Benjamin Ransford. 2015. A simpler, safer programming and execution model for intermittent systems. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM, 575–585.

[46] Yongpan Lui, 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 *Proceedings of the 52nd Annual Design Automation Conference (DAC '15)*. ACM, New York, NY, USA, 6.

[47] Kaisheng Ma, Xueqing Li, Jinyang Li, Yongpwan Liu, Yuan Xie, Jack Sampson, Mahmut Taylan Kandemir, and Vijaykrishnan Narayanan. 2017. Incidental computing on IoT nonvolatile processors. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 204–218.

[48] Kaisheng Ma, Xueqing Li, Srivatsa Rangachar Srinivasa, Yongpan Liu, John Sampson, Yuan Xie, and Vijaykrishnan Narayanan. 2017. Spendthrift: Machine learning based resource and frequency scaling for ambient energy harvesting nonvolatile processors. In *Design Automation Conference (ASP-DAC), 2017 22nd Asia and South Pacific*. IEEE, 678–683.

[49] 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 *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*. IEEE, 526–537.

[50] KIWAN MAENG, ALEXEI COLIN, and BRANDON LUCIA. 2017. Alpaca: Intermittent Execution without Checkpoints. In *Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*. ACM.

[51] Kiwan Maeng and Brandon Lucia. 2018. Adaptive Dynamic Checkpointing for Safe Efficient Intermittent Computing. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. USENIX Association, Carlsbad, CA, 129–144. https://www.usenix.org/conference/osdi18/presentation/maeng

[52] Kiwan Maeng and Brandon Lucia. 2019. Supporting peripherals in intermittent systems with just-in-time checkpoints. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM, 1101–1116.

[53] Sanyam Mehta and Josep Torrellas. 2016. WearCore: A core for wearable workloads?. In *Parallel Architecture and Compilation Techniques (PACT), 2016 International Conference on*. IEEE, 153–164.

[54] Saman Naderiparizi, Aaron N Parks, Zerina Kapetanovic, Benjamin Ransford, and Joshua R Smith. 2015. Wispcam: A battery-free rfid camera. In *RFID (RFID), 2015 IEEE International Conference on*. IEEE, 166–173.

[55] Prashant J Nair, Chiachen Chou, Bipin Rajendran, and Moinuddin K Qureshi. 2015. Reducing read latency of phase change memory via early read and turbo read. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 309–319.

[56] Shahriar Nirjon. 2018. Lifelong Learning on Harvested Energy. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 500–501.

[57] Ebelechukwu Nwafor, Andre Campbell, David Hill, and Gedare Bloom. 2017. Towards a provenance collection framework for Internet of Things devices. In *2017 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*. IEEE, 1–6.

[58] Luca Rizzon, Maurizio Rossi, Roberto Passerone, and Davide Brunelli. 2013. Wireless Sensor Networks for Environmental Monitoring Powered by Microprocessors Heat Dissipation. In *Proceedings of the 1st International Workshop on Energy Neutral Sensing Systems (ENSSys '13)*. ACM, New York, NY, USA, Article 8, 6 pages. https://doi.org/10.1145/2534208.2534216

[59] Alberto Rodriguez Arreola, Domenico Balsamo, Anup K. Das, Alex S. Weddell, Davide Brunelli, Bashir M. Al-Hashimi, and Geoff V. Merrett. 2015. Approaches to Transient Computing for Energy Harvesting Systems: A Quantitative Evaluation. In *Proceedings of the 3rd International Workshop on Energy Harvesting & Energy Neutral Sensing Systems (ENSsys '15)*. ACM, New York, NY, USA, 3–8. https://doi.org/10.1145/2820645.2820652

[60] Emily Ruppel and Brandon Lucia. 2019. Transactional concurrency control for intermittent, energy-harvesting computing systems. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM, 1085–1100.

[61] Ryo Shigeta, Tatsuya Sasaki, Duong Minh Quan, Yoshihiro Kawahara, Rushi J Vyas, Manos M Tentzeris, and Tohru Asami. 2013. Ambient RF energy harvesting sensor device with capacitor-leakage-aware duty cycle control. *IEEE Sensors Journal* 13, 8, 2973–2983.

[62] Premkishore Shivakumar and Norman P Jouppi. 2001. Cacti 3.0: An integrated cache timing, power, and area model. (2001).

[63] Emiliano Sisinni, Abusayeed Saifullah, Song Han, Ulf Jennehag, and Mikael Gidlund. 2018. Industrial Internet of Things: Challenges, Opportunities, and Directions. *IEEE Transactions on Industrial Informatics* (2018).

[64] Joshua R. Smith. 2013. *Wirelessly Powered Sensor Networks and Computational RFID*. Springer, New York, NY, USA.

[65] 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.

[66] Yulei Sui and Jingling Xue. 2016. SVF: interprocedural static value-flow analysis in LLVM. In *Proceedings of the 25th International Conference on Compiler Construction*. ACM, 265–266.

[67] Alexander Teverovsky. 2014. Insulation resistance and leakage currents in low-voltage ceramic capacitors with cracks. *IEEE Transactions on Components, Packaging and Manufacturing Technology* 4, 7 (2014), 1169–1176.

[68] Naveen Vedula, Arrvindh Shriraman, Snehasish Kumar, and William N Sumner. 2018. NACHOS: Software-Driven Hardware-Assisted Memory Disambiguation for Accelerators. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 710–723.

[69] Cong Wang, Naehyuck Chang, Younghyun Kim, Sangyoung Park, Yongpan Liu, Hyung Gyu Lee, Rong Luo, and Huazhong Yang. 2014. Storage-less and converter-less maximum power point tracking of photovoltaic cells for a nonvolatile microprocessor. In *Design Automation Conference (ASP-DAC), 2014 19th Asia and South Pacific*. 379–384. https://doi.org/10.1109/ASPDAC.2014.6742919

[70] Cheng Wang and Youfeng Wu. 2013. TSO_ATOMICITY: efficient hardware primitive for TSO-preserving region optimizations. *ACM SIGPLAN Notices* 48, 4 (2013), 509–520.

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

[72] Yiqun Wang, Yongpan Liu, Cong Wang, Zewei Li, Xiao Sheng, Hyung Gyu Lee, Naehyuck Chang, and Huazhong Yang. 2016. Storage-less and converter-less photovoltaic energy harvesting with maximum power point tracking for internet of things. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 35, 2 (2016), 173–186.

[73] Joel Van Der Woude and Matthew Hicks. 2016. Intermittent Computation without Hardware Support or Programmer Intervention. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. USENIX Association, Savannah, GA, 17–32. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/vanderwoude

[74] Mimi Xie, Mengying Zhao, Chao Pan, Jingtong Hu, Yongpan Liu, and Chun Xue. 2015. Fixing the Broken Time Machine: Consistency-Aware Checkpointing for Energy Harvesting Powered Non-Volatile Processor. In *Proceedings of The 52nd IEEE/ACM Design Automation Conference (DAC 2015) (DAC '15)*. ACM, New York, NY, USA, 6.

[75] Yuan Xie. 2016. *EMERGING MEMORY TECHNOLOGIES.* Springer.

[76] Cong Xu, Dimin Niu, Naveen Muralimanohar, Rajeev Balasubramonian, Tao Zhang, Shimeng Yu, and Yuan Xie. 2015. Overcoming the challenges of crossbar resistive memory architectures. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 476–488.

[77] Kasım Sinan Yıldırım, Henko Aantjes, Amjad Yousef Majid, and Przemysław Pawełczak. 2016. On the synchronization of intermittently powered wireless embedded systems. *arXiv preprint arXiv:1606.01719* (2016).