COACH: Consistency Aware Check-pointing for Nonvolatile Processor in Energy Harvesting Systems

Ali Hoseinghorban*, Amir Mahdi Hosseini Monazzah†‡, Mostafa Bazzaz*, Bardia Safaei*, and Alireza Ejlali*

Abstract—Recently, energy harvesting systems that utilize hybrid NVM-SRAM memory in their designs are introduced as a promising alternative for battery-operated systems. Since the ambient input power of an energy harvesting system fluctuates as the environmental conditions change, the system may stop the execution of programs until it receives enough energy to continue the execution. Resuming the execution of a program after the suspension may lead to data inconsistency in an energy harvesting system and threatens the correct functionality of the programs. In this paper, we propose COACH, an energy-efficient consistency-aware memory scheme which guarantees the correct functionality and consistency of the program in an energy harvesting system. The experimental results show that COACH improves forward-progress of the programs in the system by up to 60% compared with the state of the art consistency-aware approaches without imposing considerable energy overhead to the system.

Index Terms—Data consistency, Energy-efficiency, Nonvolatile memory, Energy harvesting system.

1 Introduction

Energy harvesting systems are considered as a replacement for battery-operated systems in situations that using battery-operated systems might be restricted (e.g., some wearable application) or the charging process of batteries is not feasible or is difficult. Energy harvesting systems are one of the main contributors to the marketing of IoT devices which have been anticipated to reach more than 100 billion objects in the near future [1], [2], [3].

While these systems are considered as a feasible solution for the situations that battery-operated systems can not be applied, the unpredictability of the ambient power leads to frequent failures in these systems. Therefore, in these systems, we need to back up the state of the processor and memories into nonvolatile storage to enable the processor and memories to return to their nearest possible state after restarting from a failure [4].

In the capacitor-less energy harvesting systems, the output of harvester directly powers up the system. The characteristics of the energy sources (like solar, radio frequency radiation, and piezoelectricity) have a significant impact on these systems. Furthermore, capacitor-less energy harvesting systems are only operational when the input power is higher than the energy consumption of the system; so, these systems are inactive most of the time [5]. For this purpose, the state of the art studies [6], [7], [8], [9], exploit a bulk capacitor to accumulate energy and power up the system; so, the system is operational even when the input power is weak or unstable. In energy harvesting systems, the processor is nonfunctional when the energy in the bulk capacitor (the input capacitor in the system which stores the absorbed energy) drops below a specific threshold. On the other hand, in such systems, the rate of harvesting energy and charging the bulk capacitor is uncontrollable and might be low on many occasions. Accordingly, inefficient usage of harvested energy in the programs running on these systems increases the rate of energy consumption and consequently discharges the bulk capacitor faster. As a result, the overall duration of nonfunctional state(s) in the processor will be increased, and the rate of the program forward-progress (number of validly executed instructions) will be reduced.

Emerging nonvolatile memories (NVMs) like STT-MRAM, PCM, and FRAM have promising features such as high density, low leakage power consumption, byte-writable granularity and non-volatility (the ability to keep data in case of power failure). These features motivate system designers to exploit NVMs in energy harvesting systems with the hope that they can save the states of the processor when the input power to the system fails. Despite the interesting features of NVMs, accessing to them is slower and consumes more energy compared with volatile memories (such as SRAM). Therefore, the hybrid approaches that benefit from positive attributes of both volatile and nonvolatile memories in their architecture are more promising. In this regard, previous studies showed that hybrid designs have better energy consumption compared to fully NVM designs [6], [10], [11]. Accordingly, there are some efforts to develop such architectures by chip manufacturers. For example, MSP430FR5964 by Texas Instruments® [12] exploits a nonvolatile processor (NVP) with 256 KB FRAM and 8 KB SRAM memory.

However, a processor with hybrid NVM-SRAM memory

* *Department of Computer Engineering, Sharif University of Technology, Tehran, Iran.*
† *School of Computer Engineering, Iran University of Science and Technology (IUST), Tehran, Iran.*
‡ *School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran.*
EMail: {hoseinghorban, bazzaz, bsafaei}@ce.sharif.edu, monazzah@iust.ac.ir, ejlali@sharif.edu.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2019.2961007, IEEE Transactions on Emerging Topics in Computing

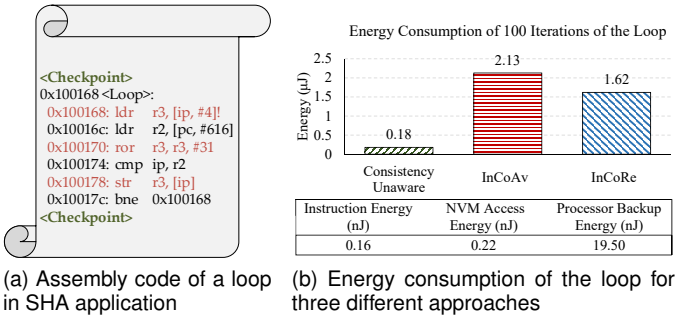
in an energy harvesting system will be faced with a challenge called *data inconsistency*. In fact, in case of failure in such systems, while NVM keeps the modified data, the contents of SRAM will be lost. After the failure, the system rolls back to the last successful check-point and resumes the program from there. However, merely resuming the program from the last successful check-point may cause data inconsistency problem since some data has been modified in the NVM after the check-point (we will discuss data inconsistency problem with more detail in section 2.1).

To deal with data inconsistency challenge, there are many studies like [9], [13], [14] which put a check-point between each load-store pair targeting the same address to avoid data inconsistency. These approaches are slow and energy inefficient because they impose too many check-points to the program [15], [16]. DINO [17], ALPACA [15], and CHAIN [7] proposed programming models in which the programmer should partition the program to a set of atomic tasks. The system either completes an atomic task and commits the changes to the memory without power failure or discards all the changes. To this end, in the entry of each atomic task, the system copies the consistency sensitive data of the task to a buffer, and if the task fails to execute successfully, the system discards the buffer (double buffering technique). These studies assume that the programmers have precise knowledge about the energy consumption and the bulk capacitor of the platform to decide the size of atomic tasks which is a non-trivial assumption. Furthermore, these approaches need to commit (discard) the changes in the buffer in case of backup (recovery), which increases the backup and recovery overheads. To alleviate the interactions of the programmers, proposed approaches in [2], [16] dynamically track the modified data during execution and instead of whole memory, they only backup (discard) the modified data since the last check-point. However, tracking the modified data in these approaches increases the energy consumption of normal read and write operations of the system significantly (see section 2.2).

In this paper, we propose COACH, an energy-efficient and consistency-aware memory management scheme for energy harvesting systems. Similar to double buffering techniques, COACH consists of two memories to keep the value of data in the last successful check-point in addition to the last modified value of data after that check-point. Accordingly, in case of power failure, COACH could resolve the inconsistency problem. COACH introduces a new mechanism for backup and rollback operations. The main superiority of COACH over double buffering techniques like [2], [7], [15], [16], [17], is that COACH's backup and rollback operations are very fast and energy efficient which improves forward-progress compared to state of the art approaches. Furthermore, COACH is a pure hardware approach which guarantees the consistency without any programmer effort [7], [15], [17], compiler modification [9], [13], or operating system support [16]. COACH works with any check-pointing mechanism and does not force any additional check-points to the system.

The main contributions of this paper are as follow:

- We introduce a simple and effective memory management scheme to efficiently handle the data inconsistency problem of the NVM memories in energy harvesting systems which does not impose any extra check-point to the system.
- We introduce new backup and rollback operations compatible with the proposed hardware, which are fast and energy efficient because COACH eliminates the need for discarding (committing) modified data in the rollback (backup) operation.
- We explore the area and energy overhead of COACH and other states of the art consistency-aware approaches.
- We investigate the effects of the bulk capacitor size, operational frequency of the processor, and the ambient power fluctuations on the efficiency of COACH.



(a) Assembly code of a loop in SHA application

(b) Energy consumption of the loop for three different approaches

Fig. 1. Energy overhead of the state of the art consistency-aware approaches (InCoAv [9] and InCoRe [2]) compared to Consistency-Unaware approach for a small loop in SHA application.

Light is considered as the source of the ambient power in this study since it is widely used in energy harvesting systems, and it is available in in-door and out-door locations. COACH is evaluated considering weak, strong, stable, and unstable solar traces [8], [18]. The results show that in comparison with the state of the art consistency-aware energy harvesting systems, proposed by Xie et al. [9] and Senni et al. [2], COACH improves the forward-progress by 60% and 48%, respectively.

The rest of this paper is organized as follows: A motivational example is presented in Section 2. Section 3 includes the details of COACH. We will evaluate the efficiency of COACH in Section 4. Related works are presented in Section 5, and finally, we will conclude the paper in Section 6.

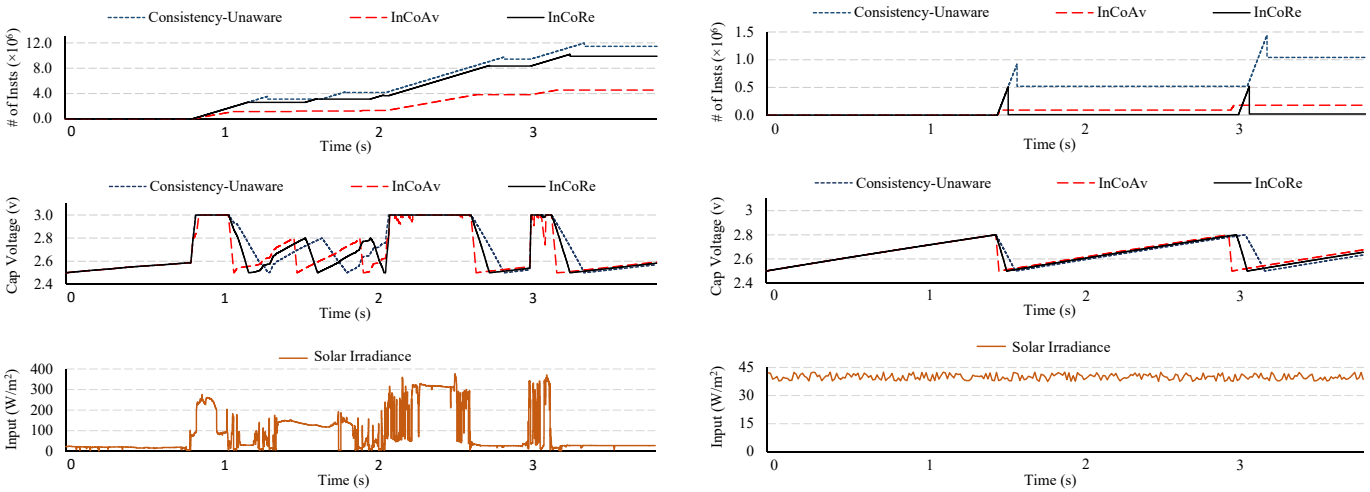
2 Observations and Motivations

With the advent of energy harvesting NVPs, inconsistency problem becomes a significant challenge. In this section, first, we explain the inconsistency problem of energy harvesting NVPs (2.1) as our main consideration and then we will discuss challenges in the state of the art approaches (2.2) as our primary motivation in this study.

2.1 Inconsistency Problem

Here we will show how the inconsistency problem threatens the system focusing on an example. Assume that we have an energy harvesting NVP which is responsible for running

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2019.2961007, IEEE Transactions on Emerging Topics in Computing



(a) Mobile device, At 01:00 PM (b) Stationary device, At 08:00 PM

Fig. 2. Forward-progress and input capacitor voltage trace of executing SHA application at two different hours of a day (a) 01:00 PM and (b) 08:00 PM for three different approaches (Consistency Unaware, InCoAv [9] and InCoRe [2]). In each column, the top, middle, and bottom charts show the forward progress, bulk capacitor voltage, and the input solar irradiance, respectively.

SHA application. An example of a loop part in SHA code is shown in Fig. 1a. In each iteration of this loop, an element is read from the array (0x100168), rotated (0x100170) and stored back to the array (0x100178). In case of power failure (voltage of the input capacitor drops below a specific threshold), the system rolls back to the last valid check-point (which in this example is the one placed before the loop), and re-executes the program from there.

During the execution of the loop, some elements of the array are in the SRAM-based cache, some of them are written back to the nonvolatile memory (evicted from cache), and the others have not been modified yet. In case of power failure, cache loses the data while nonvolatile memory keeps them. Therefore re-executing the loop from the last check-point causes inconsistency problem because some elements in the array have been modified (rotated) in the nonvolatile memory.

2.2 Challenges in the State of the Art Approaches

To tackle the inconsistency problem in the previous subsection, NVPs need a proper rollback mechanism to restore the modified entries of memory (after system failure) to their values in the last valid check-point. Literature studies that have considered failed execution in their proposed methods, either avoid inconsistency in nonvolatile memory by imposing a large number of check-pointing operations to the systems (inconsistency avoidance, abbreviated InCoAv), like approaches proposed by Van Der Woude et al. [13], Xie et al. [9], [14], or keep track of modified data and discard (commit) them in case of failed (successful) execution (inconsistency recovery, abbreviated InCoRe), like the approaches proposed by Ma et al. [19], Senni et al. [2] and Maeng et al. [16]. To show the energy inefficiency of both approaches, we conducted a set of simulations on the provided SHA code shown in Fig. 1.

Considering [9] as a representative of InCoAv check-pointing approaches, in this study, a check-point is placed between each pair of load and store instructions that target the same address. This strategy imposes a large number of check-points to the system. For example, in the small loop code presented in Fig. 1a, this approach forces a check-point in each iteration of the loop between load (0x100168) and store (0x100178) instructions.

In the InCoRe check-pointing approaches, Senni et al. [2] proposed a new memory scheme which exploits a backup memory, main memory, and an address buffer (all nonvolatile). At the start of the program, the contents of backup and main memory are the same. In the normal execution, data is read and written to the main memory and the address buffer keeps the addresses of the modified data. In case of a successful backup, the modified data will be copied from the main memory to the backup memory or vice versa (in case of rollback). Among the studies in the second group approaches, the last mentioned approach is the most promising one for ultra-low-power energy harvesting systems. Therefore, we assume the study by Senni et al. [2] as a representative of the InCoRe check-pointing approaches.

Considering the approach in [2], additional check-points might be added to the program if the address buffer is full. Backup and rollback overhead is relatively high since this approach needs to copy all the modified data to the backup memory. Furthermore, in each write operation, the controller searches the address buffer to check whether the corresponding address exists in it or not (if the corresponding address is already in the address buffer, the controller will not add that address to the buffer). Thus, the architecture presented by Senni et al. [2] will significantly increase the consumed energy by write operations. In the following, for the sake of simplicity, we will call the Xie et al. [9] approach, as InCoAv check-pointing and the proposed method by Senni et al. [2], as InCoRe check-pointing.

Fig. 1b shows the energy consumption corresponding to the execution of the mentioned loop in Fig. 1a for 100 iterations. The left bar shows the energy consumption of a Consistency-Unaware approach. The Consistency-Unaware

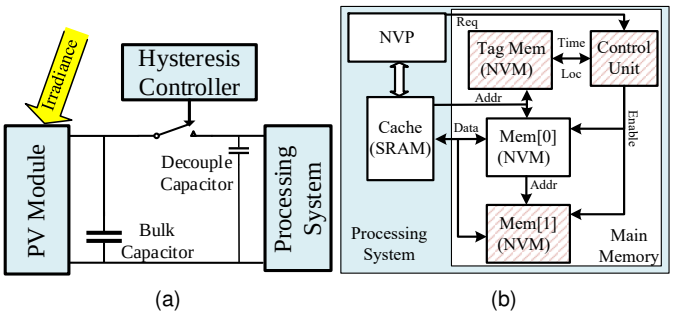


Fig. 3. An overview of an energy harvesting system (a) and a COACH-enabled processor (b).

approach performs normal read and write operations without considering the consistency of the application. Thus, in case of power failure, Consistency-Unaware can not restore the modified data in the nonvolatile memory. While Consistency-Unaware approach suffers from wrong functionality, it is the ideal case in terms of energy consumption. The middle and right bars show the energy consumption of InCoAv and InCoRe (with a 64 words address buffer), respectively. As the state of the art approaches, both InCoAv and InCoRe approaches guarantee the consistency and correct functionality of the application at the cost of imposing high energy overhead to the system (depicted in Fig. 1b).

To compare the three approaches from the energy harvesting perspective, we conduct several experiments with two different solar irradiance intensities that represent the solar radiation of the sun at 01:00 PM (a mobile device with solar irradiance between 0 $\mathrm{W/m^2}$ to 380 $\mathrm{W/m^2}$) and 08:00 PM (a stationary device with 40 $\mathrm{W/m^2}$ solar irradiance). For each scenario, the forward-progress, the voltage of the bulk capacitor and the input irradiance is depicted in Fig. 2. It should be noted that the processor input voltage in our experiments must be in the range of 2.5v to 3v. If the voltage of the bulk capacitor drops below 2.5v, hysteresis controller disconnects the processor from the capacitor, and when the capacitor voltage reaches to 2.8v, it connects the capacitor to the processor again (see Fig. 3).

Fig. 2a illustrates that InCoAv has very slow forward-progress since most of the time the energy is spent on unnecessary check-points. On the other hand, the inefficiency of write operations (searching relatively large address buffer during each write access) and high backup cost in InCoRe reduce forward-progress significantly in case of low input irradiance (see Fig. 2b). The capacitor voltage trace in Fig. 2a, and Fig. 2b shows that InCoAv discharges the bulk capacitor faster than InCoRe. It is noteworthy to mention that, although Consistency-Unaware approach uses energy of the capacitor more efficiently (the capacitor is discharged slower and the processor executes more instructions), it cannot guarantee the correct functionality of the application, hence the total executed instructions are useless in many applications.

3 COACH Approach

Section 2 clearly shows the importance of developing a novel energy efficient scheme, i.e., COACH which can guarantee the consistency and the correct functionality of the tasks during different energy absorbing states of NVPs, while providing a considerable forward-progress for the applications. In this section, we first describe the proposed memory scheme and its corresponding algorithms in subsection 3.1. Then, we will explore the functionality of the COACH with a case study example in subsection 3.2. Finally, we will discuss the possible overheads of COACH in subsection 3.3.

3.1 COACH's Proposed Hardware Scheme

Fig. 3 depicts an overview of a typical energy harvesting system and the proposed COACH architecture in this study. COACH benefits from a nonvolatile processor connected to an SRAM-based cache. From the memory hierarchy perspective, besides the mentioned cache, the main memory in COACH utilizes two NVM memories to keep the current value of the application data as well as the last successful check-point. Furthermore, to improve backup and recovery overheads and resolve the inconsistency problem, COACH is equipped with a control unit which contains two registers to keep track of the total backups and rollbacks in the system. Furthermore, COACH exploits an NVM memory as *Tag Mem*, which has an entry for each data to keep the state of the data. The modified parts in Fig. 3b are highlighted with hatching. The required peripheral modules for energy absorption are depicted in Fig. 3a.

While COACH is capable of working with any check-pointing approaches, for the sake of generality and simplicity we consider check-pointing approach proposed by Liu et al. [20] which exploits a watchdog timer that raises a backup signal periodically. During a normal execution of the program, the evicted dirty cache blocks are written back to the NVM memory. According to the discussion in the previous sections, in case of system failure, the system rolls back to the nearest successful check-point. However, the system needs a proper recovery mechanism to revert the modified data in the NVM to their corresponding values in the last successful check-point to avoid inconsistency problem [9].

Accordingly, when a data entry is modified in the NVM memory (we refer to it as *TempValue*), the valid value of the data in the last successful check-point (we refer to it as *LastChkptValue*) should be kept, hence, in case of failure, the system could discard changes in the NVM memory. To this end, COACH considers a duplicated memory (*Mem*[*0*] and *Mem*[*1*] in Fig. 3b) to keep both *TempValue* and *LastChkptValue* versions for each data. To mitigate the overhead of committing (or discarding) *TempValue* of the modified data in case of successful backup (or rollback), COACH exploits an NVM tag memory (*Tag Mem*). Each row of this memory includes two fields, called *Loc* and *Time*. For each data block written to memory during the normal execution, the information about the location (*Mem*[*0*] or *Mem*[*1*]) and the time (count of backups and rollbacks) of this update is saved in *Loc* and *Time* fields, respectively.

To control the backup and rollback operations, COACH utilizes a control unit depicted in Fig. 3b. This control unit plays an important role in redirecting the memory requests to the proper memory modules (*Mem*[*0*] or *Mem*[*1*])

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2019.2961007, IEEE Transactions on Emerging Topics in Computing

Algorithm 1 Control Unit Procedure

Input: Request (*Req*), Request Address (*Addr*), Request Data (*Data*), TBR_Cntr Register (*TBR_Cntr*), SR_Cntr Register (*SR_Cntr*), *Tag Mem* (*Tag*), Memory (*Mem*).

1: **procedure** MEMORY ACCESS
2: $Loc_{Tag} = Tag[Addr]_{Loc}$
3: $Time_{Tag} = Tag[Addr]_{Time}$
4: **if** *Req* is a Read request **then**
5: **if** $Time_{Tag} < (TBR\_Cntr + SR\_Cntr + 1)$ **and** $Time_{Tag} > TBR\_Cntr$ **then**
6: $Data = Mem[\textbf{not}\ Loc_{Tag}][Addr]$
7: **else**
8: $Data = Mem[Loc_{Tag}][Addr]$
9: **end if**
10: **else if** *Req* is a Write request **then** // Write request
11: **if** $Time_{Tag} < TBR\_Cntr$ **then**
12: $Mem[\textbf{not}\ Loc_{Tag}][Addr] = Data$
13: $Tag[Addr] = \{\textbf{not}\ Loc_{Tag}, TBR\_Cntr + SR\_Cntr + 1\}$
14: **else**
15: $Mem[Loc_{Tag}][Addr] = Data$
16: $Tag[Addr] = \{Loc_{Tag}, TBR\_Cntr + SR\_Cntr + 1\}$
17: **end if**
18: **else if** *Req* is a Backup request **then**
19: $TBR\_Cntr = TBR\_Cntr + SR\_Cntr + 1$
20: $SR\_Cntr = 0$
21: **else if** *Req* is a Rollback request **then**
22: $SR\_Cntr = SR\_Cntr + 1$
23: **else** // Flush request
24: $Tag[*]_{Time} = 0$
25: $TBR\_Cntr = 0$
26: $SR\_Cntr = 0$
27: **end if**
28: **end procedure**

based on the validity of data in the memories. To this end, this module is equipped with two nonvolatile registers: 1) *Total Backup and Rollback Counter* (we refer to it as *TBR_Cntr*), and 2) *Successive Rollback Counter* (we refer to it as *SR_Cntr*). While *TBR_Cntr* keeps track of the total backups and rollbacks since the start of the program up to the last successful check-point, *SR_Cntr* counts the number of successive rollbacks since the last successful backup. The *TBR_Cntr* and the *SR_Cntr* are nonvolatile registers because their data must be sustained in case of power failure. In the state of the art nonvolatile processors [21], [22], [23], nonvolatile registers are made with NVFF (NVFF is a CMOS flip-flop attached to a nonvolatile cell), to improve energy and performance of the processor. Therefore, during execution, the data are accessed through CMOS flip-flops, and in case of backup (restore), the CMOS flip-flops are written to (read from) nonvolatile cells.

To handle the inconsistency problem COACH should consider new mechanisms for read, write, backup, and rollback operations. In the following, we will introduce the above modifications made to the system equipped with COACH. To handle read and write operations in COACH, we have defined three states for each data residing in *Mem*[*0*] or *Mem*[*1*].

- *safe*: Data has not been modified since the last successful check-point.
- *modified*: Data has been modified during the current execution.
- *failed*: Data has been modified in failed execution.

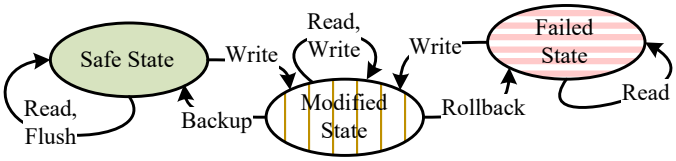


Fig. 4. The state transition diagram for each entry of the memory.

Upon each memory access, the control unit specifies the state of accessed data by using the *TBR_Cntr* register, the *SR_Cntr* register in the control unit, and *Time* field of the data in the *Tag Mem*. If *Time* of a data entry is less than *TBR_Cntr*, it means that the last modification of data was before the last successful check-point; thus, data is in a *safe* state. On the other hand, if the *Time* of a data entry is greater than *TBR_Cntr* + *SR_Cntr*, it is in a *modified* state. Finally, if *Time* of a data entry is less than *TBR_Cntr* + *SR_Cntr* and greater than *TBR_Cntr*, the data is in a *failed* state. The control unit specifies *LastChkptValue* of each data is in *Mem*[*0*] or *Mem*[*1*] by using the *Loc* field in the *Tag Mem*. For a data entry in a *safe* state, the *Loc* field points to the memory module which contains *LastChkptValue*, and in *modified* and *failed* states, it points to the *TempValue*. Therefore with this information, COACH can backup and recover NVM memory fast and with low energy overhead. Algorithm 1 represents the procedure that has been used in the control unit of COACH for the read, write, backup, rollback, and flush operations.

3.1.1 COACH's Read/Write Operation

Considering read operation, based on the requested address, COACH checks the location and the time of the last update in the requested address to retrieve the state of the data (line 2-3 in Algorithm 1). Accordingly, if the requested data is in a *failed* state, the *LastChkptValue* of the data will be read from $Mem[\textbf{not}\ Loc_{Tag}]$ (line 5-6 in Algorithm 1); otherwise the last modified value of the data is read from $Mem[Loc_{Tag}]$ (line 7-8 in Algorithm 1).

For each write request, if the requested data is in *safe* state, the new data is written to $Mem[\textbf{not}\ Loc_{Tag}]$ (line 11-12 in Algorithm 1) which does not contain *LastChkptValue* of the data, otherwise it will be written to $Mem[Loc_{Tag}]$ (line 14-15 in Algorithm 1) which contains the modified version of data in the last system failure (*FailedValue*) or current execution (*TempValue*). Thus, *LastChkptValue* of the data will not be overwritten, and in case of system failure, COACH could easily rollback to the last successful check-point without causing any inconsistency problem. After each write, the control unit updates the *Tag Mem* to keep track of the last modifications (line 13 and 16 in Algorithm 1).

3.1.2 COACH's Backup/Rollback Operation

Backup and rollback operations in COACH are simple and fast, without consuming a considerable amount of energy. During the backup operation, COACH simply increments the *TBR_Cntr* by the value of *SR_Cntr* plus one and resets the *SR_Cntr* (line 18-20 in Algorithm 1). As a result of this operation, the attributes of all data in the *modified* state is

changed to a *safe* state (*Time* of the *modified* data becomes less than *TBR_Cntr*).

When the processor wakes up after a failure, the rollback procedure in Algorithm 1 will be executed. To this end, COACH only increments the *SR_Cntr* register by one (line 21-22 in Algorithm 1). Accordingly, the attributes of all data in the *modified* state are changed to *failed* state (*Time* of *modified* data becomes less than *TBR_Cntr* plus *SR_Cntr* and greater than *TBR_Cntr*).

3.1.3 COACH's Flush Operation

We have considered a flush operation in COACH to prevent overflow of *TBR_Cntr* or *SR_Cntr* registers. The procedure used by COACH during the flush operation is shown in Algorithm 1 (line 23-26). As it can be seen, the flush operation is intrinsically an energy-consuming operation since it should set all of the values of *Time* field in *Tag Mem* entries to zero (line 24 in Algorithm 1). However, since the flush operation rarely performed during the execution (considering 2 Bytes *TBR_Cntr* register, the system performs a flush operation after each 65535 backup and rollback operations), the energy overhead imposed to the system by flush operations is negligible. We will discuss the energy penalty of flush operations further in section 3.3. Finally, Fig. 4 shows the state diagram for each entry of the memory considering read, write, backup, rollback, and flush operations.

3.2 Case Study Example

To explore the efficiency of COACH, we consider a case study scenario that includes a sequence of write, read, successful check-points, and system failure in a COACH-enabled system. Fig. 5 depicts the sequence of the instructions (Fig. 5(a)) and the contents of different entries of COACH's storage elements (Fig. 5(b)-(l)) during the execution of these instructions. Initially, all data are stored in *Mem*[*0*] and *TBR_Cntr* register, *SR_Cntr* register, *Loc* and *Time* for each entry of the memory are set to zero which means all the data are in a *safe* state (Fig. 5(b)).

The first instruction performs a write operation in *A2* entry which is in the *safe* state. Thus, new data(*D22*) is written to *Mem*[*1*] because $Tag[A2]_{Loc}$ points to *Mem*[*0*] (*LastChkptValue* of data is in *Mem*[*0*]). Then, $Tag[A2]_{Loc}$ and $Tag[A2]_{Time}$ are updated, and the state of *A2* is changed to *modified* (Fig. 5(c)). The next instruction is a write operation to *A1*, which will be executed in the same way as the first instruction (Fig. 5(d)). After executing the previous instructions, now another write operation should be executed on *A2* entry. Since *A2* is in a *modified* state, new data(*D222*) will be written to *Mem*[*1*] because $Tag[A2]_{Loc}$ is pointing to it (Fig. 5(e)).

Assuming that the backup instruction depicted in Fig. 5(a) has been executed successfully, *TBR_Cntr* is incremented by the value of *SR_Cntr* (which is zero) plus one, and as a result, the state of all entries are changed to *safe* (Fig. 5(f)). After executing the backup instruction, a request to read *A1* should be satisfied. Since *A1* is on a *safe* state, $Tag[A1]_{Loc}$ points to *Mem*[*1*], so the data in *Mem*[*1*] will be returned (Fig. 5(g)). Considering a successful backup instruction, the sixth instruction performs a write on a data entry in a *safe* state like the first two instructions (Fig. 5(h)).

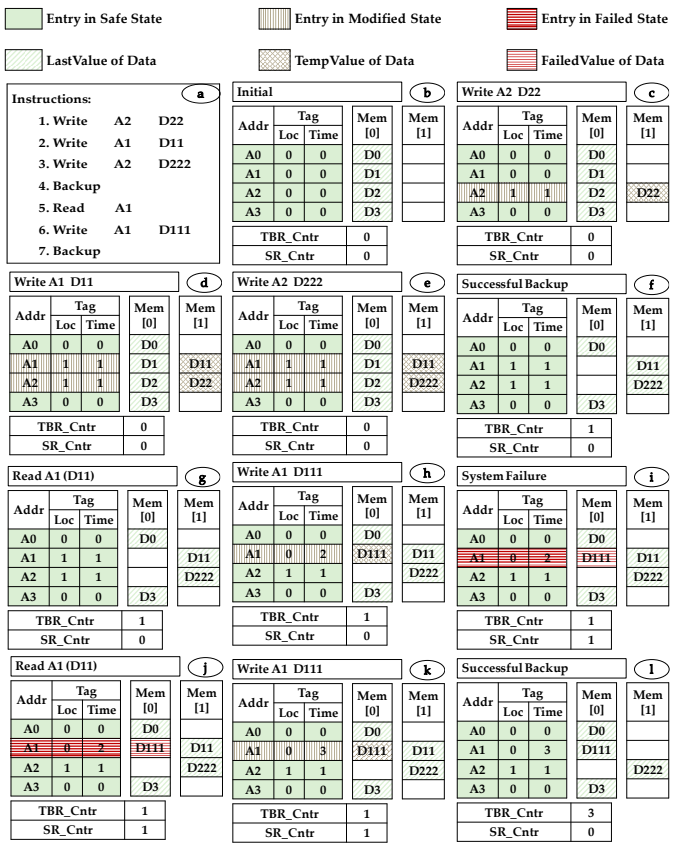


Fig. 5. An example of COACH functionality and how it can avoid inconsistency in case of system failure.

Now, assume that a system failure occurs after executing sixth instruction, COACH restores memory content to previous valid check-point only by incrementing *SR_Cntr* register by one. As a result, the state of *A1* is changed to *failed* state (Fig. 5(i)). This time, executing the fifth instruction after recovery leads to a read from a *failed* entry, so *LastChkptValue* of data in *Mem*[*1*] where $Tag[A1]_{Loc}$ does not point to it will be returned (Fig. 5(j)). Now, the sixth instruction is a write on a *failed* entry, and new data(*D111*) is written to *Mem*[*0*] where $Tag[A1]_{Loc}$ points to it (Fig. 5(k)). Finally, by a successful backup, the attribute of all data is changed to the *safe* state (Fig. 5(l)).

3.3 Discussion on the COACH Overheads

In this subsection, we discuss the possible overheads of COACH. To this end, we compare COACH with other state of the art approaches, i.e., the approach presented by Xie et al. [9] (as a deputy of the InCoAv approaches) and the approach presented by Senni et al. [2] (as a deputy of the InCoRe approaches) from the energy consumption and area perspectives.

In the following, for the sake of brevity we only discuss energy consumption of the main memory (NVM memory), hence energy consumption of non-memory instructions or the effects of cache memories on the energy consumption of the system are not considered in the provided equations because they are exactly the same in all of the approaches (note that we have considered them in our experiments).

The energy consumption of read, write, and check-point operations in [9] are calculated as follow :

$$Read\_Eng_{InCoAv} = Reads \times NVM_{Rd}\_Eng \quad (1)$$
$$Write\_Eng_{InCoAv} = Writes \times NVM_{Wr}\_Eng \quad (2)$$
$$Chkp\_Eng_{InCoAv} = (Chkps + LdSt\_Chkps) \times Chkps\_Eng \quad (3)$$

where *Reads*, NVM_{Rd}_Eng, NVM_{Wr}_Eng, *Writes*, *Chkps*, *Chkps_Eng*, and *LdSt_Chkps* represent the number of reads from NVM, energy per read access to NVM, energy per write access to NVM, number of writes to NVM, number of watchdog timer check-points, processor backup energy consumption and number of load store pairs to one address, respectively. While [9] does not affect read and write energy consumptions of the system, it imposes a large number of extra and unnecessary check-points (*LdSt_Chkps*) which increases the energy consumption of the system.

On the other hand, the energy consumption of read, write, and check-point operations in [2] are calculated as follow :

$$Read\_Eng_{InCoRe} = Reads \times NVM_{Rd}\_Eng \quad (4)$$
$$Write\_Eng_{InCoRe} = Buf\_Hit \times (NVM_{Wr}\_Eng + Srch\_Buf) + Buf\_Miss \times (2 \times NVM_{Wr}\_Eng + Srch\_Buf) \quad (5)$$
$$Chkp\_Eng_{InCoRe} = Chkps \times Chkps\_Eng + Buf\_Miss \times (2 \times NVM_{Rd}\_Eng + NVM_{Wr}\_Eng) + Buf\_Overflow\_Chkps \times Chkps\_Eng \quad (6)$$

where *Reads*, NVM_{Rd}_Eng, NVM_{Wr}_Eng, *Buf_Miss*, *Buf_Hit*, *Srch_Buf*, *Chkps*, *Chkps_Eng* and *Buf_Overflow_Chkps* represent number of reads from NVM, energy per read access to NVM, energy per write access to NVM, number of write access miss events in the address buffer, number of write access hit events in the address buffer, energy consumption of searching the address buffer, number of watchdog timer check-points, processor backup energy consumption and number of times the system forces a check-point because the address buffer is full, respectively.

The InCoRe method [2] does not increase the energy consumption of read accesses because it returns data from the main memory. On the other hand, in each write access, it needs to search the address buffer and if it could not find the data (*Buf_Miss*), in addition to updating the data, it needs to write address of the data to the address buffer (one more NVM access is required in case of *Buf_Miss*). The proposed approach by Senni et al. [2] introduces additional check-points in case that the address buffer becomes full (Buf_Overflow_Chkps). Therefore, the size of the address buffer is an important parameter which introduces a trade-off between the number of additional check-points and the cost of searching the address buffer. Furthermore, in each check-point, the modified data (for each *Buf_Miss* an entry is added to the address buffer which needs to be backed up in the next check-point) must be written back to the backup memory which leads to high backup cost in this approach (for each modified data, three NVM accesses is required because the controller needs to read the address from the address buffer, read corresponding data from main memory and write it to backup memory).

TABLE 1
Overhead of FRAM area on total price of a micro controller.

Part	FRAM (KB)	SRAM (KB)	Price (US$)	Price Overhead (%)
MSP430FR5731	4	1	1.20	-
MSP430FR5735	8	1	1.26	5.0
MSP430FR5739	16	1	1.38	9.5
MSP430FR6977	64	2	3.20	-
MSP430FR6979	128	2	3.35	4.7
MSP430FR5962	128	8	2.85	-
MSP430FR5964	256	8	3.25	14.1

Finally, the energy consumption of read, write, and check-point operations in COACH are calculated as follows:

$$Read\_Eng_{COACH} = Reads \times (3 \times NVM_{Rd}\_Eng) \quad (7)$$
$$Write\_Eng_{COACH} = Writes \times (2 \times NVM_{Wr}\_Eng + NVM_{Rd}\_Eng) \quad (8)$$
$$Chkp\_Eng_{COACH} = Chkps \times Chkps\_Eng + Flush\_Eng \quad (9)$$

where *Reads*, NVM_{Rd}_Eng, NVM_{Wr}_Eng, *Writes*, *Chkps*, *Chkps_Eng* and *Flush_Eng* represent number of reads from NVM, energy per read access to NVM, energy per write access to NVM, number of writes to NVM, number of watchdog timer check-points, processor backup energy consumption and energy consumption of flushing *Tag Mem*, respectively. For a read operation, a data entry is fetched from each memory (*Mem*[*0*] and *Mem*[*1*]) and the *Tag Mem*, hence for each read operations we need three accesses to NVM memory. In each write operation, COACH reads *Tag Mem*, writes new data to the target memory and updates *Tag Mem*, as a result, the write operation requires three accesses to NVM memory.

According to the above discussion and the formulation, we can conclude that InCoAv [9] does not increase the energy consumption of read and write accesses to the NVM memory, however, the experiments show that most of the benchmarks, the number of check-points due to load-store pairs are high which decrease the energy efficiency and forward progress of the system. InCoRe [2] needs to search address buffer in each write access; therefore, it has poor performance in the write intensive benchmarks. The results show that COACH improves forward progress in almost all the benchmarks, except the benchmarks which the number of loads is significantly higher than stores. This happens because in COACH, a data needs to be fetched from *Mem[0], Mem[1],* and *Tag Mem* which increases the energy consumption of read operation compared to two other approaches (see section 4).

It is noteworthy to mention that COACH does not increase the memory access latency during the normal execution. In other words, during a read operation, data of *Mem[0], Mem[1], Tag Mem*, *TBR_Cntr* register, and *SR_Cntr* register are fetched concurrently and based on the state of the *Tag Mem*, data in *Mem*[*0*] or *Mem*[*1*] will be returned to the processor. Therefore, the latency of a read operation is almost equal to the latency of a single NVM read access. The peripheral circuits for selecting the suitable memory for reading data have negligible latency compared to NVM

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2019.2961007, IEEE Transactions on Emerging Topics in Computing

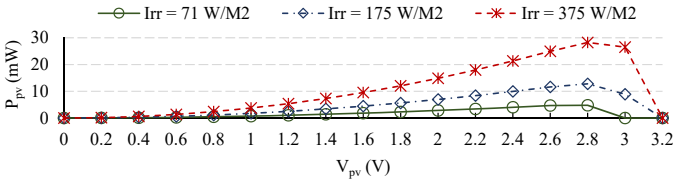


Fig. 6. Power-Voltage curve of energy harvester model proposed by Wang et al. [8]. The curve shows that the system achieves the maximum power when the bulk capacitor voltage is between 2.5v and 3v.

latency (see Table 2). On the other hand, during a write operation, the write controller should first read the *Tag Mem*, *TBR_Cntr* register, and *SR_Cntr* register concurrently and based on the state of the data, it should select a suitable memory for the write operation. Accordingly, write operation takes two clock cycles and imposes one clock cycle performance overhead to the system. It should be mentioned that this overhead could be resolved by exploiting a couple of buffers that keep the address and data of the write request and lets the processor resumes the execution, while the memory controller updates the corresponding memory fields.

From the check-point operation perspective, COACH does not impose any additional check-point to the system, and the energy consumption for each backup or rollback is equal to updating *TBR_Cntr*, and *SR_Cntr* registers in addition to the flush cost. Although flush is an energy consuming operation (since it resets all *Time* entries in *Tag Mem*), it occurs very rarely (especially when the size of the time field in *Tag Mem*, *TBR_Cntr* and *SR_Cntr* registers are relatively large). Assuming a system containing a 64 KB (addressable) of NVM memory in which, each word is four Bytes, the *Tag Mem* would contain 16 K (16384) entries. Thus, if we consider that each entry of time field occupies 2 Bytes, then each flush operation requires to reset 32 KB of memory (size of *Tag Mem*). On the other hand, since after 65536 backup or rollback operations, the system needs to be flushed, amortized cost (energy and latency) of flush is 16 Bytes per each backup or rollback. The amortized cost (energy and latency) of flush operation could be reduced by increasing the size of the *Time* field. For example, if we consider three Bytes (four Bytes) space for each *Time* entry, the amortized flush cost (energy and latency) for every 1000 (200000) check-points would be only three Bytes.

COACH exploits two memories (*Mem*[*0*], *Mem*[*1*]), a *Tag Mem* and a control unit which requires more area than the InCoRe [2] (two memories and one address buffer) and the InCoAv [9] (no area overhead). However, in the energy harvesting systems, energy plays an important role in the system, and it directly influences the forward progress of the system. Furthermore, it is important to mention that the two memories and the *Tag Mem* are all NVM with almost zero leakage power consumption [23], [24]. The control unit is a simple circuit with low energy and latency overhead (latency, leakage power, and dynamic energy of the control unit are presented in Table 2). Therefore the additional circuits of COACH do not increase the energy consumption of the system noticeably. A brief comparison among the existing MSP430 microcontrollers, which benefit from FRAM technology, has been presented in Table 1. According to Table 1, by duplicating the size of FRAM memory, there will be only 5%-15% of increment in the overall costs of manufacturing these microcontrollers, which is tolerable.

TABLE 2
Setup configurations used in simulations.

Simulation configurations*	
Processor Energy Per Instruction (No Memory Access)	0.16 nJ
FRAM Read Access Energy	0.22 nJ
FRAM Write Access Energy	0.34 nJ
SRAM (Cache) Access Energy	0.05 nJ
Backup Energy (Processor)	19.50 nJ
Restore Energy (Processor)	16.90 nJ
Processor Leakage Power	675 µW
Platform Power Consumption (without processor)	450 µW
FRAM Access Latency	125 ns
SRAM (Cache) Access Latency	1 Cycle
Backup/Restore Latency (Processor)	400 ns
Control Unit Access Latency	0.98 ns
Control Unit Leakage Power	10.09 µW
Control Unit Dynamic Energy	0.63 pJ
Control Unit Area	3628 μm^2
Processor Frequency	8, 16, 24 MHz
Bulk Capacitor Size	10, 50, 100, 150 µF
Stable Solar Irradiance (see Fig. 2b)	71, 175, 375 W/m^2
Unstable Solar Irradiance (see Fig. 2a)	0-380 W/m^2
Simulation Time (for each case)	3.5 s

* Energy and latency of backup, restore, access to FRAM and SRAM were adopted from MSP430FR5969 microcontroller by Texas Instrument® [12] and proposed microcontroller by Khanna et al. [23]. The power consumption of the energy harvesting platform was adopted from Wang et al. [8]. The energy and latency characteristics of the control unit were estimated using Synopsys Design Compiler [26] using 90 nm technology. The stable and unstable solar irradiance traces were taken from Gorlatova et al. [18].

4 Experiments

In this section, we will explore the effectiveness of COACH in handling the consistency challenge for nonvolatile processors used in energy harvesting systems. To this end, first we will introduce the system setup which has been used during our experiments in the following subsection. Then, we will discuss the experiment results in the second subsection.

4.1 Experimental Setup

Since COACH is introduced for the energy harvesting systems, the first step is selecting the input energy source for the system. Accordingly, we consider light as the input energy source as it is popular and available in in-door and out-door locations [18]. Furthermore, the light power trace could be stable (for stationary devices), unstable (for mobile devices), strong (in the morning, and afternoon) or weak (in the evening). Since the light irradiance (the power of input source) is a function of time and location, for the sake of diversity in our explorations, we consider three stable power traces with low (71 W/m^2), medium(175 W/m^2) and high (375 W/m^2) irradiances for stationary devices, in addition to an unstable power trace (0-380 W/m^2) for mobile devices (unstable input trace is depicted in Fig. 2a, and a sample of stable power trace is depicted in Fig. 2b). To mimic a proper behavior of the energy absorption in

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2019.2961007, IEEE Transactions on Emerging Topics in Computing

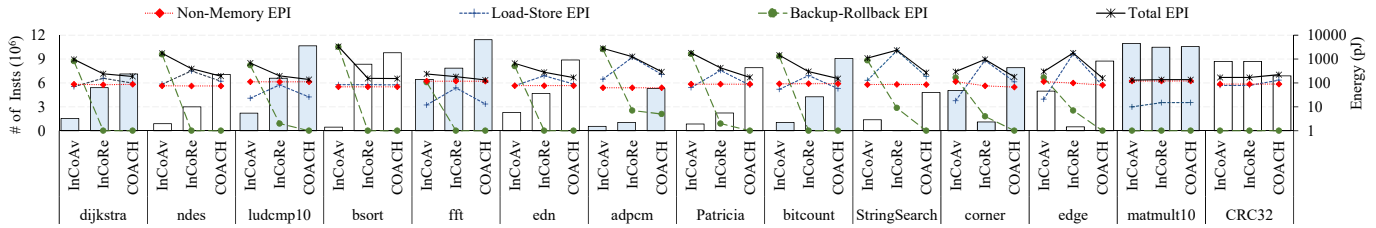


Fig. 7. The results for comparing energy consumption and forward-progress of COACH, InCoAv [9] and InCoRe [2] with the unstable power trace (see Fig. 2a), 24 MHz processor frequency, and 50 µF bulk capacitor.

energy harvesting systems, we consider the converter-less photovoltaic power system proposed by Wang et al. [8] with two photovoltaic cells (4.5×5.5 cm^2) to absorb energy from ambient sources and store it to a relatively small bulk capacitor (10 µF to 150 µF).

The inconsistency is an important challenge for all nonvolatile memories that have been used in check-pointing methods, however, in the experiments we consider nonvolatile processors with Ferroelectric-CMOS technology as a case study since there are a number of state of the art studies that fabricate nonvolatile processors using Ferroelectric-CMOS technology [22], [23], [25]. Furthermore, MSP430FR microcontrollers by Texas Instruments [12] are commercial off the shelf NVPs with on-chip FRAM which enabled us to adapt energy consumption of instructions, FRAM access, and SRAM access from a real energy harvesting platform for the experiments (Table 2). We considered a nonvolatile possessor equipped with 64 KB of FRAM and 0.5 KB of SRAM cache running at 8, 16, and 24 MHz. The considered configurations for COACH-enabled nonvolatile processors is in line with the MSP430FR microcontroller by Texas Instruments® [12] and the microcontroller proposed by Khanna et al. [23]. We estimated the area, latency, dynamic energy, and leakage power of the control unit using Synopsys Design Compiler [26] using 90 nm technology.

We have exploited the harvester model proposed by Wang et al. [8]. The P-V curve of this model is depicted in Fig. 6. Based on this chart, the output power of the harvester is determined by the input irradiance, and voltage of the bulk capacitor. Furthermore, we integrated the power consumption of the energy harvesting platform, the latency and energy consumption of instruction executions, FRAM accesses, SRAM accesses, and leakage dissipation of MSP430FR5969 microcontroller to the SimpleScalar simulator [27] (see Table 2). Therefore, for each instruction, the simulator calculates the harvested energy, and consumed energy to update the energy (voltage) of the bulk capacitor.

We have compared the results of COACH with two states of the art approaches introduced in Section 2 (InCoAv [9] and InCoRe [2]). Fourteen different benchmarks (*Patricia, StringSearch, BitCount, Dijkstra, MatMult, NDES, LUDCMP, Bubble Sort, FFT, EDN, ADPCM, Corner(Susan), Edge(Susan)* and *CRC*) from MiBench and Malardalen benchmark suites are exploited for the evaluations. In line with the previous studies [16], [20], we have considered a watchdog counter to raise a backup signal after executing 50000 instructions as the check-pointing policy. The details of the simulation setup are presented in Table 2.

4.2 Results

The energy consumption and the forward-progress of an application running on a typical energy harvesting system have a close relationship with each other. Indeed, because the input energy in these systems is limited and unstable, the system will be inactive most of the time, and it will be waiting for the bulk capacitor to be charged. Therefore, the inefficient use of the stored energy in the bulk capacitor will slow down the forward-progress of the system.

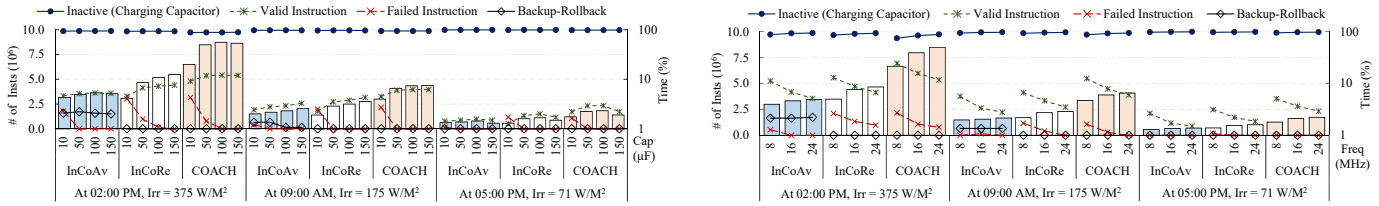
Fig. 7 shows the energy consumption and forward-progress of different benchmarks for InCoAv [9], InCoRe [2] and COACH, under unstable power trace (see Fig. 2a) with 24 MHz processor frequency and 50 µF bulk capacitor. Since accessing memories during the execution of applications plays the main role in the energy consumption of the energy harvesting system, we classify the instructions based on their characteristics in accessing memories. To this end, we will explore the contributions of instructions in three classes, i.e., non-memory instructions, load-store instructions, and backups-rollback instructions. We will use the *Energy Per Instructions* (EPI) metric for this purpose.

In Fig. 7, the bars show the average of validly executed instructions, and the black line with star marks shows *Total EPI*. Besides evaluating *Total EPI*, we further evaluate and illustrate the contribution of *Non-Memory EPI* (red line with diamond marks), *Load-Store EPI* (blue line with plus marks), and *Backup-Rollback EPI* (green line with circle marks), separately to provide more understanding on the contributions of different classes of instructions (introduced above) on *Total EPI* of an energy harvesting system.

1) ***Backup-Rollback EPI***: As it has been depicted in Fig. 7, *Backup-Rollback EPI* in InCoAv [9] is responsible for a large portion of *Total EPI* while the impact of Backup-Rollback on *Total EPI* of InCoRe is much lower. Furthermore, the effect of *Backup-Rollback EPI* in COACH is ideal since COACH benefits from fast and low cost backup and rollback operations. It is noteworthy to mention that for the applications such as *StringSearch, Corner*, and *Edge* with a high number of write accesses to different locations, InCoRe imposes relatively high backup overhead to the system, since the address buffer (mentioned in section 2) fills up frequently and imposes many additional backup operations to the system.

2) ***Load-Store EPI***: From the Load-Store instructions perspective, InCoAv [9] has ideal energy consumption since it does not affect the normal read and write operations of memory. On the other hand, in InCoRe, a significant

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2019.2961007, IEEE Transactions on Emerging Topics in Computing



(a) Capacitor capacity effects (24 MHz frequency)

(b) Processor frequency effets (50 μF capacitor)

Fig. 8. The effects of irradiance strength, processor frequency and capacity of the bulk capacitor in forward-progress of energy harvesting systems.

portion of the energy is consumed due to store operations since it requires to search the entire address buffer (64 words in the experiment) for each write access. Therefore, InCoRe consumes a considerable amount of energy in the applications with a high number of write operations, e.g., *StringSearch*, *Corner*, and *Edge*, which leads to slower forward-progress.

3) ***Non-Memory EPI***: Considering Non-Memory instructions, Fig. 7 illustrates that none of the three approaches affect the non-memory instructions energy consumption. Therefore, *Non-Memory EPI*s are relatively the same for all approaches.

According to the above discussion and the results of Fig. 7, we can conclude that the COACH has lower EPI than InCoAv [9] and InCoRe [2] for almost all the benchmarks in our experiments, while it resolves the inconsistency problem. According to our observations, *CRC* and *Matmult* were the only benchmarks where COACH may impose more EPI or forward-progress penalties (compare to InCoAv and InCoRe) to the system. This is mainly due to increment of energy consumption of read operations (3 FRAM accesses) in COACH while other approaches do not affect the energy of read accesses. Thus, in applications like *CRC* and *Matmult*, which the number of loads are significantly higher than stores, COACH may impose EPI or forward-progress overheads to the system. The results show that on average, COACH, improves forward progress by 48% and 60% compared to InCoRe and InCoAv approaches respectively.

The forward-progress in each energy harvesting system is not only affected by the efficiency of the EPI metric in that system, but it is also affected by the strength of the input power, the capacity of the system's bulk capacitor (as the system energy provider), and the operational frequency of the system. Fig. 8 depicts the effects of capacitor's size and frequency on the forward-progress of a system with low (at 5 PM, 71 $\mathrm{W/m^2}$), medium (at 9 AM, 175 $\mathrm{W/m^2}$) and high (at 2 PM, 375 $\mathrm{W/m^2}$) irradiances for a stationary device with a stable input power (see Fig. 2b). For the sake of brevity, only the average results of the benchmarks are presented. Here, the bars show the total validly executed instructions and the lines show the portion of time that the system spent on: 1) charging the bulk capacitor (blue line with circle marks), 2) executing valid instructions (green line with star marks), 3) executing failed instructions (red line with cross marks), and 4) executing backup-rollback instructions (black line with diamond marks).

Considering Fig. 8a, exploiting large capacitor could reduce the number of failed instructions, but on the other hand, it has some disadvantage like occupying a large area and imposing safety concerns (for wearable devices). In addition, charging a large capacitor requires longer time, especially when the input power is weak (i.e., after 5 PM in out-door situations). The results in Fig. 8a shows that by increasing the capacitor size from 10 μF to 50 μF, number of failed instructions will be reduced significantly and increasing the size of the capacitor beyond 50 μF would have a negligible impact on the forward-progress. Hence, for COACH, exploiting 50 μF, 100 μF, and 150 μF will improve the forward-progress by 36%, 43%, and 31% compared to 10 μF capacitor, respectively.

Fig. 8b illustrates the effects of frequency changes in an energy harvesting system equipped with 50 μF bulk capacitor. As it was mentioned earlier, in an energy harvesting system, the input energy is limited, and the system is in an inactive state (waiting for the bulk capacitor to be charged) most of the time. Therefore, increasing the frequency of an energy harvesting system would not improve the forward-progress of the system as it does for typical battery powered embedded systems. Indeed, by increasing the frequency of the processor, while instructions will be executed faster and the rate of the energy consumption in the system will be increased, charging rate of the bulk capacitor remains the same.

Furthermore, considering the FRAM technology, the memory operates at 8 MHz frequency. Therefore, increasing the frequency of the processor will not increase the speed of memory access or backup (rollback) instructions. In this regard, according to Fig. 8b in InCoRe and COACH, increasing the frequency to 24 MHz improves the forward-progress by up to 44%, but in InCoAv the forward-progress would be improved up to 21% since InCoAv frequently stops the execution for taking check-points.

5 Related Work

The previous studies which have targeted the correct functionality of the systems with unstable ambient power could be classified into three groups. The studies in the first group recover the state of the system from previously backed up information at FLASH memory in case of system failure [4], [28]. Although these studies exploit different techniques to reduce the number of system failures, they suffer from high failure penalties since backup and recovery operations from FLASH memory impose noticeable overheads in terms of latency and power consumption [1].

The second group of studies exploits hardware-based or software-based techniques to prevent system failure in case

of power failure. Therefore, these studies do not consider any recovery mechanisms in their solutions. Considering the software-based approaches [6], [7], [15], [17], a task is partitioned to one or more atomic sections. Then, in the entry of each atomic section, the system checks the energy of the bulk capacitor and only executes the next atomic section if the capacitor has enough energy, otherwise the system stalls until the capacitor receives enough charge. Unlike the software-based approaches, in hardware-based approaches [3], [29], [30] system starts executing the tasks until it detects a failure in the input power. Then, the system begins to back up the dirty blocks of SRAM memories (i.e., cache or scratchpad memory) in the nonvolatile memory.

There are some challenges that both of the hardware and software approaches should consider them, e.g., self-discharging of the bulk capacitor [8], [31], measuring the state of the charge of the bulk capacitor [32], [33], input power fluctuations. In the hardware-based approaches, since the state of the processor is unknown when the input power fails, the number of blocks that need to be backed up may be too small or too large (in the worst case situation, the system needs to backup all of the SRAM memory [34]). Accordingly, before a system failure, we cannot accurately estimate the amount of the required charge in the bulk capacitor for the backup operations. Likewise, for the software-based approaches estimating a safe worst case energy consumption of a task (or section) requires conservative approaches [35], [36] and precise knowledge about the energy consumption of the platform which could be complicated. To this end, it seems that an efficient recovery mechanism should be considered alongside the proposed approaches in the second group.

The studies in the third group propose a rollback operation to the last successful check-point when the system fails without accessing FLASH memory or starting the application from the beginning. In this regard, Hibernus++ [5] adaptively set threshold voltage to minimize the number of check-pointing for capacitor-less energy harvesting systems. These studies rollback the system to the last successful check-point in case of system failure, but they did not consider the data inconsistency problem. DINO [17], ALPACA [15], and CHAIN [7] proposed programming models and suggested breaking the program to set of atomic tasks with the aid of programmers. To deal with the data inconsistency, the system copies the task's data to a buffer and runs the task. The system backs up the buffer's data to the main memory if it completes the task without power interrupts; otherwise, the system discards the buffer. The main drawback of these approaches is that the programmers need to have a precise knowledge of the energy consumption and the capacitor size of the platform which is a non-trivial assumption and cause poor re-usability of the software. Unlikely, in the Chinchilla [16], the programmers divide the program to code blocks which none of them exceeds the capacitor energy. Chinchilla exploits an operating system level approach to insert check-points adaptively between blocks to minimize the number of check-points while providing forward-progress. To ensure data consistency, Chinchilla uses a log file called *undo Log*, which logs the address and value of the modified data after the last successful check-point. However, running operating system is not feasible for many ultra-low-power systems.

Senni et al. [2] approach exploits a memory architecture which has duplicated nonvolatile memory as the main memory and backup memory along with an address buffer. During the normal execution of the program, backup memory is off, and the buffer keeps track of the data which their value has been changed. When the backup signal is raised, dirty data in the main memory will be copied to the backup memory. Thus, in case of soft error or power failure, the main memory data will be recovered from backup memory. Van Der Woude et al. [13], Xie et al. [9], [14], propose a consistency-aware check-pointing scheme which finds the same address referenced load and store pairs and put a check-point before the store instruction in each founded pair. While their approach does not have any area overhead, it considerably increases the number of backups in the program. Although the proposed approaches in the third group have better energy consumption than the proposed approaches in the first group, they waste a considerable amount of energy to guarantee the correct functionality (as discussed in section 2).

Comparing with previous studies mentioned above, in this study, we proposed COACH, an energy-efficient and consistency-aware memory scheme for energy harvesting nonvolatile processor. COACH benefits from fast and low energy backup and rollback operations. COACH is a full hardware approach; thus, it does not require any programmer or operating system interactions. COACH works with all check-pointing mechanism, and it does not force any additional check-point to the system.

6 Conclusion

In this paper, we proposed COACH, an energy-efficient and consistency-aware memory scheme which guarantees correct functionality and consistency of the program in an energy harvesting system. COACH benefits from fast backup and rollback operations which impose low energy overhead to the design. Unlike the previous consistency-aware approaches, COACH does not charge any additional check-points to the program. The results show that COACH can guarantee consistency of the program and improves the forward-progress of the system compared to the state of the art approaches proposed by Xie et al. [9] and Senni et al. [2] by 60% and 48%, respectively.

References

[1] Y. Liu, Z. Li, H. Li, Y. Wang, X. Li, K. Ma, S. Li, M.-F. Chang, S. John, Y. Xie *et al.*, "Ambient energy harvesting nonvolatile processors: from circuit to system," in *Proceedings of the 52nd Annual Design Automation Conference (DAC)*. ACM, 2015, p. 150.

[2] S. Senni, L. Torres, G. Sassatelli, and A. Gamatie, "Non-volatile processor based on mram for ultra-low-power iot devices," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 13, no. 2, p. 17, 2017.

[3] M. Xie, M. Zhao, C. Pan, H. Li, Y. Liu, Y. Zhang, C. J. Xue, and J. Hu, "Checkpoint aware hybrid cache architecture for nv processor in energy harvesting powered systems," in *Proceedings of the Eleventh IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES/ISSS)*. ACM, 2016, p. 22.

[4] B. Ransford, J. Sorber, and K. Fu, "Mementos: System support for long-running computation on rfid-scale devices," *Acm Sigplan Notices*, vol. 47, no. 4, pp. 159–170, 2012.

[5] D. Balsamo, A. S. Weddell, A. Das, A. R. Arreola, D. Brunelli, B. M. Al-Hashimi, G. V. Merrett, and L. Benini, "Hibernus++: a self-calibrating and adaptive system for transiently-powered embedded devices," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 35, no. 12, pp. 1968–1980, 2016.

[6] H. Jayakumar, A. Raha, J. R. Stevens, and V. Raghunathan, "Energy-aware memory mapping for hybrid fram-sram mcus in intermittently-powered iot devices," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 16, no. 3, p. 65, 2017.

[7] A. Colin and B. Lucia, "Chain: tasks and channels for reliable intermittent programs," in *ACM SIGPLAN Notices*, vol. 51, no. 10. ACM, 2016, pp. 514–530.

[8] Y. Wang, Y. Liu, C. Wang, Z. Li, X. Sheng, H. G. Lee, N. Chang, and H. Yang, "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 (TCAD)*, vol. 35, no. 2, pp. 173–186, 2016.

[9] M. Xie, M. Zhao, C. Pan, J. Hu, Y. Liu, and C. J. Xue, "Fixing the broken time machine: Consistency-aware check-pointing for energy harvesting powered non-volatile processor," in *Proceedings of the 52nd Annual Design Automation Conference (DAC)*. ACM, 2015, p. 184.

[10] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable high performance main memory dystem using phase-change memory technology," *ACM SIGARCH Computer Architecture News*, vol. 37, no. 3, pp. 24–33, 2009.

[11] J. Hu, C. J. Xue, Q. Zhuge, W.-C. Tseng, and E. H.-M. Sha, "Data allocation optimization for hybrid scratch pad memory with SRAM and nonvolatile memory," *IEEE Transactions on Very Large Scale Integration (TVLSI) Systems*, vol. 21, no. 6, pp. 1094–1102, 2013.

[12] Texas Instruments, "MSP430FRxx Microcontrollers," *http://www.ti.com*, 2019.

[13] J. Van Der Woude and M. Hicks, "Intermittent computation without hardware support or programmer intervention," in *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI})*, 2016, pp. 17–32.

[14] M. Xie, C. Pan, M. Zhao, Y. Liu, C. J. Xue, and J. Hu, "Avoiding data inconsistency in energy harvesting powered embedded systems," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 23, no. 3, p. 38, 2018.

[15] K. Maeng, A. Colin, and B. Lucia, "Alpaca: intermittent execution without checkpoints," *Proceedings of the ACM on Programming Languages*, vol. 1, no. OOPSLA, p. 96, 2017.

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

[17] B. Lucia and B. Ransford, "A simpler, safer programming and execution model for intermittent systems," *ACM SIGPLAN Notices*, vol. 50, no. 6, pp. 575–585, 2015.

[18] M. Gorlatova, A. Wallwater, and G. Zussman, "Networking low-power energy harvesting devices: Measurements and algorithms," *IEEE Transactions on Mobile Computing (TMC)*, vol. 12, no. 9, pp. 1853–1865, 2013.

[19] K. Ma, Y. Zheng, S. Li, K. Swaminathan, X. Li, Y. Liu, J. Sampson, Y. Xie, and V. Narayanan, "Architecture exploration for ambient energy harvesting nonvolatile processors," in *21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2015, pp. 526–537.

[20] Q. Liu and C. Jung, "Lightweight hardware support for transparent consistency-aware check-pointting in intermittent energy harvesting systems," in *5th Non-Volatile Memory Systems and Applications Symposium (NVMSA)*. IEEE, 2016, pp. 1–6.

[21] W. Song, Y. Zhou, M. Zhao, L. Ju, C. J. Xue, and Z. Jia, "Emc: Energy-aware morphable cache design for non-volatile processors," *IEEE Transactions on Computers*, vol. 68, no. 4, pp. 498–509, 2018.

[22] Y. Liu, F. Su, Y. Yang, Z. Wang, Y. Wang, Z. Li, X. Li, R. Yoshimura, T. Naiki, T. Tsuwa *et al.*, "A 130-nm ferroelectric nonvolatile system-on-chip with direct peripheral restore architecture for transient computing system," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 3, pp. 885–895, 2019.

[23] S. Khanna, S. C. Bartling, M. Clinton, S. Summerfelt, J. A. Rodriguez, and H. P. McAdams, "An fram-based nonvolatile logic mcu soc exhibiting 100% digital state retention at vdd=0v achieving zero leakage with $<$ 400-ns wakeup time for ulp applications," *IEEE Journal of Solid-State Circuits*, vol. 49, no. 1, pp. 95–106, 2014.

[24] S. C. Bartling, S. Khanna, M. P. Clinton, S. R. Summerfelt, J. A. Rodriguez, and H. P. McAdams, "An 8mhz 75μa/mhz zero-leakage non-volatile logic-based cortex-m0 mcu soc exhibiting 100% digital state retention at vdd=0v with $<$ 400ns wakeup and sleep transitions," in *IEEE International Solid-State Circuits Conference Digest of Technical Papers*, Feb 2013, pp. 432–433.

[25] F. Su, Y. Liu, Y. Wang, and H. Yang, "A ferroelectric nonvolatile processor with 46 μs system-level wake-up time and 14 μs sleep time for energy harvesting applications," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 64, no. 3, pp. 596–607, 2016.

[26] Synopsys Inc., "Design Compiler: RTL Modeling User Guide," *http://www.synopsys.com*, 2019.

[27] D. Burger and T. M. Austin, "The simplescalar tool set, version 2.0," *ACM SIGARCH computer architecture news*, vol. 25, no. 3, pp. 13–25, 1997.

[28] H. Li, Y. Liu, Q. Zhao, Y. Gu, X. Sheng, G. Sun, C. Zhang, M.-F. Chang, R. Luo, and H. Yang, "An energy efficient backup scheme with low inrush current for nonvolatile sram in energy harvesting sensor nodes," in *Proceedings of Design, Automation & Test in Europe Conference & Exhibition (DATE)*. EDA Consortium, 2015, pp. 7–12.

[29] Y. Liu, J. Yue, H. Li, Q. Zhao, M. Zhao, C. J. Xue, G. Sun, M.-F. Chang, and H. Yang, "Data backup optimization for nonvolatile sram in energy harvesting sensor nodes," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 36, no. 10, pp. 1660–1673, 2017.

[30] H. Jayakumar, A. Raha, W. S. Lee, and V. Raghunathan, "Quickrecall: A hw/sw approach for computing across power cycles in transiently powered computers," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 12, no. 1, p. 8, 2015.

[31] K. Ma, X. Li, H. Liu, X. Sheng, Y. Wang, K. Swaminathan, Y. Liu, Y. Xie, J. Sampson, and V. Narayanan, "Dynamic power and energy management for energy harvesting nonvolatile processor systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 16, no. 4, p. 107, 2017.

[32] S. Ulukus, A. Yener, E. Erkip, O. Simeone, M. Zorzi, P. Grover, and K. Huang, "Energy harvesting wireless communications: A review of recent advances," *IEEE Journal on Selected Areas in Communications*, vol. 33, no. 3, pp. 360–381, 2015.

[33] N. Michelusi, L. Badia, and M. Zorzi, "Optimal transmission policies for energy harvesting devices with limited state-of-charge knowledge," *IEEE Transactions on Communications*, vol. 62, no. 11, pp. 3969–3982, 2014.

[34] K. Ma, M. J. Liao, X. Li, Z. Huan, and J. Sampson, "Evaluating tradeoffs in granularity and overheads in supporting nonvolatile execution semantics," in *18th International Symposium on Quality Electronic Design (ISQED)*. IEEE, 2017, pp. 39–44.

[35] P. Wägemann, T. Distler, T. Hönig, H. Janker, R. Kapitza, and W. Schröder-Preikschat, "Worst-case energy consumption analysis for energy-constrained embedded systems," in *27th Euromicro Conference on Real-Time Systems (ECRTS)*. IEEE, 2015, pp. 105–114.

[36] J. Morse, S. Kerrison, and K. Eder, "On the limitations of analyzing worst-case dynamic energy of processing," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 17, no. 3, p. 59, 2018.

Ali Hoseinghorban received his B.Sc. in computer engineering from Shahid Beheshti University and his M.Sc. from Sharif University of Technology in 2015 and 2017, respectively. He is currently a Ph.D. student in the Computer Engineering department of Sharif University of Technology. His research interests include memory management in energy harvesting system, investigating the challenges of emerging nonvolatile memories, and low-power real-time embedded systems.

Amir Mahdi Hosseini Monazzah received his Ph.D degree in computer engineering from the Sharif University of Technology, Tehran, Iran, in 2017. He was a member of the Dependable Systems Laboratory from 2010 to 2017. As a Visiting Researcher, he was with the Embedded Systems Laboratory, University of California, Irvine, CA, USA from 2016 to 2017. As a postdoc fellow he was with the school of computer science, institute for research in fundamental sciences (IPM), Tehran, Iran from 2017 to 2019. He is currently a faculty member of the School of Computer Engineering, Iran University of Science and Technology (IUST), Tehran, Iran. His research interests include investigating the challenges of emerging nonvolatile memories, hybrid memory hierarchy design, and IoT applications.

Mostafa Bazzaz received his B.Sc. in computer engineering from Amirkabir University of Technology and his M.Sc. from Sharif University of Technology in 2009 and 2011, respectively. He is currently a Ph.D. candidate in the Computer Engineering department of Sharif University of Technology. His research interests include low-power multi-core embedded systems, nonvolatile memories, and real-time embedded systems.

Bardia Safaei (IEEE Student Member) received his B.Sc. degree in computer engineering (CE) from KNTU, Tehran, Iran, in 2014 and the M.Sc degree from Sharif University of technology (SUT), Tehran, Iran, in 2016 respectively. Currently he is pursuing the Ph.D. degree in CE at SUT. He is also conducting research as a visiting PhD student at Chair for Embedded Systems at Karlsruhe Institute of Technology (KIT), Germany. He is honored to be selected as a member of national elites foundation since 2016. His research interests include power efficiency and dependability challenges in Internet of Things (IoT).

Alireza Ejlali received the Ph.D. degree in computer engineering from the Sharif University of Technology, Tehran, Iran, in 2006. He is an Associate Professor of computer engineering with the Sharif University of Technology. He is currently the Director of the Embedded Systems Research Laboratory, Department of Computer Engineering, Sharif University of Technology. His current research interests include low power design, real-time embedded systems, and fault-tolerant embedded systems.