Checkpoint Aware Hybrid Cache Architecture for NV Processor in Energy Harvesting Powered Systems*

Mimi Xie¹, Mengying Zhao², Chen Pan¹, Hehe Li³

Yongpan Liu³, Youtao Zhang⁴, Chun Jason Xue⁵, Jingtong Hu¹

¹School of Electrical and Computer Engineering, Oklahoma State University, OK, USA

²School of Computer Science and Technology, Shandong University, China

³Department of Electronic Engineering, Tsinghua University, China

⁴Computer Science Department, University of Pittsburgh, PA, USA

⁵Department of Computer Science, City University of Hong Kong, Hong Kong

ABSTRACT

Energy harvesting is one of the most promising battery alternatives to power future generation embedded systems in Internet of Things (IoT). However, energy harvesting powered embedded systems suffer from frequent execution interruption due to unstable energy supply. To bridge intermittent program execution across different power cycles, non-volatile processor (NVP) was proposed to checkpoint register contents during power failure. Together with register contents, the cache contents also need to be preserved during power failure. While pure non-volatile memory (NVM) based cache is an intuitive option, it suffers from inferior performance due to high write latency and energy overhead. In this paper, we will propose replacement and checkpoint policies for SRAM and NVM based hybrid cache in NVPs whose execution is interrupted frequently. Checkpointing aware cache replacement policies and smart checkpointing policies are proposed to achieve satisfactory performance and efficient checkpointing upon a power failure and fast resumption when power returns. The experimental results show that the proposed architectures and policies outperform existing cache architectures for NVPs.

Keywords

Non-volatile processor; Hybrid cache; Checkpointing; IoT; Energy harvesting

1. INTRODUCTION

The applications of Internet of Things (IoT), such as smart manufacturing, smart city and transportation, and smart energy, have been and will continue to transform the way we live in a positive way. In IoT applications, small sensors and systems are used to

collect information of interest to support optimal decision making. It is predicted that IoT will consist of 50 billion objects by 2020 [1]. While the vision is promising and exciting, there are several challenges in achieving this goal. One of the most important challenges is how to power these 50 billion embedded devices. While battery power has been the energy source for most embedded systems these days, it is not a favorable solution in the long run due to size, longevity, safety, and recharging concerns. Therefore, researchers are actively pursuing power alternatives. Among all solutions, energy harvesting is one of the most promising techniques to meet the requirements of large scale embedded devices.

Energy harvesters generate electric energy from their ambient environment using direct energy conversion techniques. Examples of power sources include kinetic, light, RF, and thermal energy. The obtained energy can be used to charge a capacitor to power the electronics. However, there is an intrinsic drawback with harvested energy. They are all **unstable**. With an unstable power supply, the whole computer system will be interrupted frequently, which will cause severe performance degradation. What is worse, large tasks may never finish since the intermediate results cannot be saved.

In order to bridge the intermittent execution under unstable energy supply, non-volatile memories (NVMs) [18, 30, 20] based non-volatile processors (NVPs) [17, 30, 39] were proposed. Upon a power failure, the program status, including registers and caches are saved in NVMs. When the power comes back on, the saved contents are loaded back to the registers and caches so that the execution can continue from where was interrupted. Since NVMs can retain the data even when the power is off, it can successfully preserve computation status across different power cycles.

In existing NVPs, ferroelectric memory (FRAM) based NV register was adopted where a FRAM cell is attached to each standard flip-flop. The standard flip-flops are accessed during normal execution and the FRAM cell is used to save the state during a power failure. The same design strategy could be adopted for caches in NVPs. Li et al. [12] integrates a SRAM cell and a non-volatile element in cell level, forming a direct bit-to-bit connection. In this design, the NVM part is underutilized because the NV elements are idle most of time, and the area size is unnecessarily large. Therefore, it is not always desirable. Another possible design is adopting pure NVMs-based cache [16]. However, since NVMs such as STT-RAM [7] and PCM [19, 21] typically have high write latency and energy overhead, pure NVM based cache will become the major performance bottleneck.

Different from two designs mentioned above, hybrid cache architecture, which consists of both SRAM and NVM (e.g. STT-RAM), was proposed to achieve energy efficiency and high perfor-

*This work was supported in part by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. CityU 11214115).

†This work was partially supported by NSF CNS-1464429, CCF-1527506, CCF-1535755, and CCF-1617071.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

CODES/ISSN '16, October 01-07, 2016, Pittsburgh, PA, USA

© 2016 ACM. ISBN 978-1-4503-4483-8/16/10...\$15.00

DOI: <http://dx.doi.org/10.1145/2968456.2968477>

mance [31, 28, 33, 37]. It also serves as a promising cache architecture for energy harvesting systems since it promises both high efficiency and non-volatility. However, all existing hybrid cache architecture and policies are designed for energy efficiency and performance purposes. None of them considered checkpointing efficiency. Therefore, in this paper, we aim to develop checkpointing aware hybrid cache architecture and policies to achieve the following goals: 1) high performance; 2) full utilization; 3) reliable and efficient checkpoint during a power failure.

The complexity lies in the following aspects. First, during a power failure, the available energy for checkpointing is limited by the capacitance of the capacitor. Second, NVM is both used for normal cache access and for preservation of the volatile blocks at power failure. The usage of its space between these two purposes should be balanced to achieve the best performance. Third, it is also challenging to identify data that is unnecessary to checkpoint. Consequently, the cache architecture and cache management policies must be carefully investigated for energy harvesting powered systems. In order to answer these challenges, this work makes the following contributions:

- We present a hybrid cache architecture built with SRAM and STT-RAM, and STT-RAM is fully utilized not only for normal cache access but also for volatile SRAM checkpoint.
- We design replacement and migration policies to balance the usage of STT-RAM space between normal access and checkpoint. Proactive write back policy is also designed to guarantee successful and efficient checkpointing.
- We propose an efficient checkpointing policy to save all necessary volatile blocks to STT-RAM before capacitor energy is depleted upon a power failure.

The remainder of this paper is organized as follows. Section 2 presents the related works. Section 3 describes the system architecture. Section 4 describes the motivation of this paper. Section 5 presents the new cache architecture checkpointing policy. Detailed experimental evaluation is provided in Section 6. Finally, Section 7 concludes this paper.

2. RELATED WORKS

In this section, we will describe related works on energy harvesting systems, NVPs, and cache architectures.

Energy harvesting system: Energy harvesting technologies are promising long-lasting replacements for batteries in embedded system. Ambient sources such as solar [26, 6], motion [25, 10], radio-frequency (RF) electromagnetic radiation, thermal gradients [9, 11] can provide enough energy for embedded system to be completely self-sustainable. For ultra low power devices, the sources of low power densities, such as micro-solar [26] and body heat (2.4~4.8W) should be able to provide sufficient power to drive the devices at low duty cycles [9]. For systems that require high reliability, multi-source energy sources can be employed [32]. For example, [26] designs a micro-solar power sensor network, [27] proposes a non-volatile microprocessor powered with a solar energy harvesting system even under low solar irradiance, and [22] designs a self-powered pushbutton controller which functions with a piezoelectric conversion mechanism. Even though the power harvested is lower than the power required by the complete system, it is still possible to operate the system with proper energy management. [9] presents a self powered wearable health monitoring system where each module works under different number of recharge cycles.

However, instability of energy harvesting sources will lead to power failure and state loss as a consequence, which impedes wide adoption of large software in energy harvesting powered embedded

system. This problem have attracted a great deal of interest from both academic institution and industry. Researchers have deployed NV memory into energy-harvesting devices to store the execution states [39, 23, 24, 14] because of their features of non-volatility and low leakage. Among all these works, non-volatile processor appears as a promising solution to bridge intermittent program execution under unstable power.

Non-volatile processor: NV processor attaches a nonvolatile memory cell to each volatile element and therefore allows fast local backup of intermediate results and fast recovery. FRAM based processors [39, 17, 36], present great potential to be deployed in energy-harvesting devices. They show many desirable characteristics of energy-harvesting systems, such as no battery, zero stand-by power, and fast access. FRAM also has a superior endurance as long as 10^{14} write cycles. For example, Yu et al. [8] propose a non-volatile processor architecture which integrates non-volatile elements into volatile memory at bit granularity. Wang et al. [30] design a FRAM based processor, which attaches a NV FRAM cell to each volatile standard flip-flop. The flip-flops are accessed for normal execution while the FRAM cells are used to checkpoint the states in flip-flops at power failure. To reduce the backup overhead and energy, different technologies have been proposed including instruction scheduling [35], register reduction [38], compare-and-write [29], and instruction selection [34].

Cache architecture: In addition to register contents, cache contents in NVPs should also be saved to ensure correctness and fast resuming. Among all existed cache designs, two options can be adopted in energy harvesting systems. One is pure NVM based cache which replaces SRAM with NVMs totally [16]. However, this design incurs large write overhead and degrades the system performance, which is especially true when the cache is part of a pipeline stage. Another option is adopting NVSRAM [15] which integrates a SRAM cell and a non-volatile element in cell level, forming a direct bit-to-bit connection. In this design, the NVM part is underutilized because the NV elements are idle most of time and the area size is unnecessarily large. There are some other designs based on both SRAM and NVM to achieve energy efficiency and high performance [31, 33]. However, all existing hybrid cache architectures and policies are designed for energy efficiency and performance purposes. None of them consider the scenario of intermittent power supply and therefore are not resistant to power failure. In this paper, we will design a checkpointing aware cache architecture for the energy harvesting systems.

3. BACKGROUND

In this section, the system architecture will be presented and the design of the energy harvesting embedded systems will be explored.

3.1 System Architecture

The targeting energy harvesting based embedded system is shown in Figure 1. The system is powered with energy harvested from ambient sources, such as solar energy, thermal energy, piezoelectric, or radio frequency (RF). Besides the power of harvested energy, a small-size storage capacitor is used to supply energy for checkpointing during power failure. The NVP consists of three elements: NV registers, NV hybrid cache, and NV main memory. Since all three parts of the processor have non-volatile components, the execution state can be preserved when there is a power interrupt under unstable power supply. After power returns, the NVP will restore the execution state and resume execution from the interrupted point.

Figure 1: Energy harvesting system architecture

The small-size storage capacitor enables accumulating execution states by supporting checkpointing. It is notable that, the energy stored in this capacitor enables a successful checkpointing whenever there is a power failure in this paper supported with the proposed cache architecture. Besides, checkpointing is always necessary. This is because if we do not do checkpointing, although the states in SRAM can retain for a while powered with the capacitor, everything in SRAM will be lost if the power does not come back right after the energy in the capacitor depletes.

3.2 Non-volatile Registers

Figure 2: NV Processor and Ferroelectric Flip-Flop [30]

Figure 2(a) shows a NV processor that is based on the Intel 8051 micro-controller. Figure 2(b) shows the architecture of a NV register based on ferroelectric memory. In this NV register, the left side is a standard two-stage flip-flop. A ferroelectric memory based non-volatile storage is attached to the standard flip-flop. The content of the standard flip-flop can be copied to the ferroelectric memory to save the state. This work enables the processor's registers to be non-volatile and the execution can be quickly recovered from power failure.

3.3 Checkpointing Aware Cache

As we mentioned in the introduction, currently, there are two designs for cache in NVP. Li et al. [12] integrates a SRAM cell and a non-volatile element in cell level, forming a direct bit-to-bit connection. In this design, the NVM part is underutilized because the NV elements are idle most of time. Another possible design is adopting pure NVMs-based cache [16]. However, since NVMs typically have high write latency, pure NVM based cache will become the performance bottleneck. In this work, we are proposing a checkpointing aware hybrid cache which consists of both S-

RAM and STT-RAM. Hybrid cache architecture is not new. However, most existing designs are aiming for high performance and low power consumption. [31, 33] are two representative works on hybrid cache which build last level cache with a small region of SRAM for fast access and a large region of NVM for large capacity. However, these hybrid caches are not specifically designed for energy harvesting embedded systems where only one level L1 cache is often implemented. Therefore, if this kind of hybrid cache is directly used as first level cache, the performance will be largely degraded and important states in SRAM will be lost when there is a power failure. Instead, the cache proposed in this work aims for reasonable high performance, reliable checkpointing, instant resumption, and low energy consumption. To have a better understanding of this targeted architecture, we compare it with many existed cache designs as shown in Table 1.

4. MOTIVATION

In this section, we will show the motivation of this work. First, the performance overhead of building pure STT-RAM cache will be presented. Then, several observations will be provided to show that traditional hybrid cache architecture is not suitable for energy harvesting system.

Figure 3: Performance of SRAM cache and NVM cache

Energy harvesting powered processor built with non-volatile cache will be able to overcome data loss problem caused by frequent interrupts. However, purely non-volatile cache will degrade the performance due to its expensive access time. Among all NVMs, STT-RAM is considered as the most promising candidate for building non-volatile cache because of its fast access time and high density. However, its write latency is 10 times large as its read latency [7]. In a pipelined processor, the slowest pipeline stage determines the clock cycle. We configured two processors with pure SRAM cache and pure STT-RAM cache in gem5 [2] and compared their performances. The SRAM cache and the STT-RAM cache are of the same size and their features are shown in Table 4. Figure 3 shows the performance comparison of these two systems. This figure shows that STT-RAM based cache increases system execution time by 50% on average when compared with SRAM based cache.

Compared with pure STT-RAM solution for energy harvesting NVP, hybrid cache architecture based on SRAM and NVM can be a better solution. During power failure, all dirty blocks in SRAM need to be saved since they are modified. Figure 4 shows the percentage of dirty blocks for four different benchmarks through their execution time. From the figure, we can see that the percentage of dirty blocks is high for most benchmarks except *sperand*. In existing hybrid caches, only the last level consists of NVM and NVM is also used for storing dirty data. Therefore, in the presence of power

Table 1: Comparison of different cache architectures

Properties	SRAM cache	Hybrid cache [31]	STT-RAM cache	NVSRAM cache [15]	Proposed hybrid cache
Performance	High	High	Low	High	High
Energy	High	Low	Medium	Low	Medium
Size	Medium	Small	Small	Large	Small
Resistant to power loss	No	No	Yes	Yes	Yes
Resumption speed	—	—	Instant	Fast	Instant
Checkpointing energy	—	—	—	High	Low

Figure 4: Percentage of dirty blocks during lifetime

failure, it is possible that most NVM blocks are dirty and should not be overwritten. Thus, dirty data in SRAM still needs to be written back to non-volatile main memory. Without any optimization, the checkpoint process is both time and energy consuming due to large amount of data to write. Even worse, checkpoint may fail because of the limited energy provided by capacitor. In this work, we are trying to adjust the usage of NVM blocks dynamically such that certain blocks are reserved for possible checkpointing.

Another observation from Figure 4 is that, if we reserve NVM blocks for possible checkpointing, there might be extra clean NVM blocks available after saving all dirty SRAM blocks to NVM blocks for benchmarks with low dirty data such as *sperand*. In such cases, additional clean blocks can be checkpointed during a power failure to improve system performance after resuming. The proposed cache architecture will take the utilization of NVM, checkpoint efficiency, and system performance after resuming into consideration.

5. CHECKPOINTING AWARE CACHE ARCHITECTURE

In this section, we will present a checkpointing aware cache architecture which will both have high performance and checkpointing efficiency. The hybrid cache architecture will be presented in Section 5.1. The replacement and migration policy is presented in Section 5.2. Finally, the checkpointing policy is presented in Section 5.3.

5.1 Hybrid Cache Architecture

Figure 5: Hybrid cache architecture

In a typical processor for embedded systems, there is often only one level cache and the size of the cache is not large. Besides, it takes longer time to access a second level cache than the first level. Therefore, in this paper, hybrid one level cache architecture is designed. The proposed hybrid cache architecture is shown in Figure 5. In each cache set, there are both SRAM cache blocks and STT-RAM cache blocks. For the SRAM portion, there is a counter, *DBCounter*, to record the current number of dirty SRAM cache blocks. For each cache set, there is a SRAM based migration counter, *MCounter*, for checkpointing purposes. Besides, each cache block has three state bits: valid bit (VB), dirty bit (DB), and live bit (LB). These three bits are used for directing the cache placement. This figure shows a four way set-associative cache with two SRAM blocks and two STT-RAM in each set for illustration purposes. However, this cache can have other associativity settings with flexible ratios of SRAM and STT-RAM. When power failure is detected, the necessary SRAM cache blocks will be backed up to STT-RAM. When power returns, no restoration process is needed to restart execution.

The novelties of the proposed hybrid cache architecture lie in the following aspects:

- This is the first time that STT-RAM is implemented as the first level cache in embedded systems. The high write latency of STT-RAM is neutralized by only searching all volatile blocks in the same set, when there is a write request.
- *DBCounter* and *Mcounter* directed migration policy and write-back policy can fully utilize the fast access speed of SRAM and the space of STT-RAM, while reserving a portion of the STT-RAM for checkpointing upon power failures.
- The cache architecture supports reliable checkpointing. When there is power outage, the most important states in SRAM are

always successfully backed up to STT-RAM. When power returns, the execution can resume instantly without restoration.

5.2 Replacement and Migration Policies

This section will discuss the checkpointing aware placement policy and migration policy. These two policies will be able to maintain the overall efficiency of the cache architecture, as well as guaranteeing a reliable and fast checkpointing at each power interrupt.

5.2.1 Replacement Policy

The **basic replacement policy** is the dead block based LRU replacement policy and it will be assisted with proactive writing back. The techniques proposed in [13] is used to predict dead blocks. Before we explain the details of the policy, the following notations will be defined first.

Table 2: Notations used in this paper

Notation	Description
$Vvictim$	LRU dead block or live block if there is no dead block in SRAM.
$NVvictim$	LRU dead block or live block if there is no dead block in STT-RAM.
$DVvictim$	Dirty $Vvictim$.
$DNVvictim$	Dirty $NVvictim$.
$CVvictim$	Clean $Vvictim$.
$CNVvictim$	Clean $NVvictim$.

Suppose this cache is N -way set associative, and there are totally M sets. In each set of the hybrid L1 cache, N_v cache blocks are volatile and N_{nv} cache blocks are non-volatile, which are distributed among multiple banks. Therefore, we have:

$$N_v + N_{nv} = N \quad (1)$$

Upon a power failure, there should be enough space for checkpointing the dirty volatile cache blocks in the same set. Therefore, the following constraint should be satisfied:

$$DB_v^i \leq CB_{nv}^i \quad (2)$$

In this inequality, DB_v^i is the number of dirty volatile cache blocks in set i , and CB_{nv}^i is the number of clean non-volatile cache blocks in set i .

Suppose the total available energy in storage capacitor can only support checkpointing T cache blocks, as a result, if we want to guarantee a successful checkpointing, the total number of dirty cache blocks in SRAM should be less than this threshold. That is

$$\sum_{i=1}^M DB_v^i \leq T \quad (3)$$

The replacement policy will be directed by the inequality (2) and inequality (3). To satisfy these two conditions, we implemented two counters in the cache structure: $DBCOUNTER$ for SRAM portion and $MCounter$ for each cache set.

The $DBCOUNTER$ records the current total number of dirty cache blocks in the SRAM cache part. When the $DBCOUNTER$ exceeds the preset threshold, a $DVvictim$ block in the same cache set will be written back to the main memory. By writing back a dirty block, the total number of dirty volatile cache blocks keeps below or equal to the preset threshold such that checkpointing can always be successful with energy in capacitor.

The $MCounter$ is used to identify whether there is enough space in the non-volatile portion for checkpointing the dirty volatile cache blocks in the same set. The size of the counter depends on the number of non-volatile cache blocks in each set N_{nv} . The number of bits for $MCounter$ in each cache set is set as follows:

$$size = \lceil \log_2 N_{nv} \rceil \quad (4)$$

This counter keeps tracking the state of each cache block in the same set. Its value is initially set to N_{nv} . If one cache block becomes dirty in this set, the value of $MCounter$ decrements by one. If one dirty cache block turns to be clean, this value increments by one. Once this value reaches zero, it means that there is no space in STT-RAM for checkpointing more SRAM blocks. Therefore, at this time, a $DNVvictim$ (or $DVvictim$ if no $DNVvictim$) will be proactively written back if another cache block becomes dirty in the same set.

When there is a read, all cache blocks in the same set will be searched. Since STT-RAM's read latency is close to SRAM's read latency, this will not slow down the pipeline stage. In this way, STT-RAM blocks are not only used for checkpointing as in [12] but also for data accesses. If it is a read hit, it will be served right away. If it is a read miss, data will be loaded to the blocks in this set. Due to the maintenance of $MCounter$, there are always clean blocks. The priority of the destination blocks will be clean SRAM > clean STT-RAM. LRU dead block will be chosen first as victim. If there is no LRU dead block, then LRU live block will be chosen.

When there is a write, instead of all blocks, only SRAM blocks in the cache set will be searched. The rationale is that, since writing to STT-RAM takes much longer than reading, if writing to STT-RAM is in the pipeline stage critical path, the whole stage will be slowed down. If it is a hit in SRAM blocks, the write will be served. At the same time, $DBCOUNTER$ and $MCounter$ will be updated. If it is a write miss in SRAM blocks but the data is in STT-RAM blocks, then the pipeline will be stalled and these two data blocks will be switched. In this way, STT-RAM blocks behave as a second level cache for SRAM blocks during cache write. Since STT-RAM is on-chip, the stall cycles are relatively small compared with fetching data from the main memory. However, if the requested block is absent in both SRAM and STT-RAM blocks, the migration policy described in the following subsection will be used to manage the cache replacement process and fetch data from the main memory.

5.2.2 Migration Policy

The goal of the migration policy is to conduct data migrations such that the space of the STT-RAM should be fully utilized while most read or write hits still land on SRAM blocks. The proposed policy answers this challenge by using SRAM as a sifter, and placing the blocks sifted out in the STT-RAM.

Once there is a miss and the requested block is absent in both SRAM and STT-RAM blocks, a $Vvictim$, which is either a LRU dead block or a LRU live block if all blocks are live, will be selected in SRAM for replacement. If the $Vvictim$ is a block that is predicted live, we will migrate it to the STT-RAM. This is because, although it is selected in the SRAM as a victim to place new block, there is still a high probability that it will be used in the future. If it is written back to the main memory, it will take a long time to reload it from the main memory for the future request. Therefore, rather than writing it back to the main memory, we migrate it to the non-volatile cache part, since this takes less time and energy. The new destination for $Vvictim$ will be the $NVvictim$. If $Vvictim$ block is predicted dead, there is no need to migrate it. If it is clean, it will be simply overwritten, since there is very little chance that it will be accessed again. If it is dirty, it will be written back to the memory,

Figure 6: Flow of checkpointing friendly block placement policy

in case that this prediction was a mistake. Besides, a small write buffer is implemented to mitigate the latency of migration.

Figure 6 illustrates the detailed cache policies with write miss example. In this example, there are 4 blocks in the cache set including volatile blocks 1 and 2, and non-volatile blocks 3 and 4. Initially, *DBCCounter* is equal to 32. Because there are 1 dirty volatile block and 1 dirty non-volatile block, the *MCounter* is 0. When there is a request for block tag5 and it is a write miss, volatile block tag2 is then selected as the *Vvictim* and it is written to the write buffer (operation ①). After that, the new block tag5 is loaded to volatile block tag1 from the main memory and it becomes the MRU. At the same time, the block in the write buffer will be written to non-volatile block tag4 (operation ③), which is selected as a *NVvictim*. In this way, operation ② does not need to wait for a long non-volatile write time before it starts and it only waits for a volatile write operation. At this time, *MCounter* is less than 0, which means there is no space for checkpointing block tag5. Therefore, we will choose *DNVvictim* to write back (operation ④) and *MCounter* increases by 1.

In all, the main novelty of the proposed cache replacement and management mechanism lies in two aspects: 1) Six kinds of victim blocks are defined in Table 2, and the new replacement policy can determine an appropriate kind of victim block for replacement or for writing back, given the system requirements. 2) The new mechanism not only considers the performance of cache access but also considers the checkpointing ability when there is power outage via replacement policy and migration policy.

5.3 Checkpointing Policy

In this section, we will present the checkpointing policy based on the proposed cache architecture. This checkpointing policy will specify what to be checkpointed, as well as where to store these volatile cache blocks.

5.3.1 Selecting Volatile Blocks for Checkpointing

Checkpointing is performed within each cache set. Therefore, upon a power failure, the necessary volatile cache blocks will be backed up to non-volatile blocks in each set. The first problem we need to answer is what to be checkpointed. From analysis, we know that all dirty blocks need to be backed up. However, if energy supply and STT-RAM spaces allow, clean blocks that will be used in the near future can also be backed up to NVM to improve the performance after power resumes. This means if the number of dirty blocks in SRAM is less than T , then $T - \sum_{i=1}^M DB_v^i$ blocks will be checkpointed to improve the cache performance. In this case, these most recently used live clean volatile cache blocks should be chosen first so that they can be accessed right away after power recovers.

The selection of clean volatile blocks depends on two considerations: first, there should be available non-volatile space in the same set for placing the selected clean volatile block; second, the remaining energy in capacitor should be sufficient for checkpointing these selected clean blocks.

Figure 7: Clean block selecting policy

For the first consideration, during the selection of clean volatile cache blocks, only the *MCounter* most recently used clean blocks will be selected for checkpointing. The other blocks will be dropped. ICache does not have this consideration because it does not have dirty blocks and there will always be enough space for checkpointing. For the second consideration, age bits are maintained for ICache and DCache blocks in order to differentiate LRU and MRU blocks. As shown in Figure 7, during the checkpointing process, the most recently accessed cache set will be scanned first for DCache and ICache sequentially. Before a clean block is checkpointed, its LB bit will be checked first. If it is already predicted dead, it will be given up because dead block has high probability not to be accessed and another live block will be selected instead. In this work, we apply dead data prediction policy proposed in [13] to guide the content selection, which predict dead blocks based on bursts of accesses to a cache block.

5.3.2 Selecting Non-volatile Blocks

After we have determined the SRAM blocks that need to be checkpointed, we need to decide which blocks in STT-RAM should be used to store them. Not all non-volatile cache blocks can be used to place these clean volatile cache blocks. Since non-volatile cache portion also contains some dirty cache blocks. Therefore, only these non-volatile clean cache blocks can be overwritten to place the volatile dirty cache blocks. Among all these non-volatile clean cache blocks, dead clean blocks or LRU live clean blocks in STT-RAM will be the first choice. As a result, for each dirty volatile cache block, the *CNVvictim* will be selected for checkpointing each time. In addition, a *MCounter* is updated to ensure there is always enough space for checkpointing in STT-RAM.

Figure 8 illustrates the process of checkpointing with two cache sets. Before checkpointing, there is only 1 volatile dirty block whose tag is tag2 in set 1, and there are two dirty volatile blocks in set 2. Therefore, current *DBCCounter* is 3. If a power failure happens, we will first find the *CNVvictim* in set 1, which is block tag3, and then replace block tag3 with block tag2. After that, we will find the *CNVvictim* in set 2, which is block tag8, and replace the content of block tag8 with the first dirty block, tag5. Then block tag6 will be selected as the new *CNVvictim*, and it is used to checkpoint another volatile dirty block tag7. After that, if the threshold for *DBCCounter* is larger than 3, then the clean block tag1 in set 1 can be further checkpointed to block tag4 for better performance, because it is the MRU block while tag4 is the LRU.

After power failure happens, not only volatile cache blocks, the values in *MCounter* and *DBCCounter* will also disappear because they are based on SRAM for fast access speed. Therefore, they also need to be saved to non-volatile memory.

Figure 8: Checkpointing policy

After power returns, we do not need the process of restoration, since STT-RAM based cache part is also used for normal access. Therefore, the execution can start quickly without copying anything back to STT-RAM.

5.4 Hardware overhead

This section will analyze the area overheads introduced by extra state bits, dead block prediction, and checkpointing logics. We will take a 8-way associative L1 cache (16K hybrid I-cache and 16K hybrid D-cache) with block size of 64B as an example to show the hardware overhead. The overhead of the proposed hybrid cache architecture mainly comes from three respects.

1. One dirty bit (VB) and one live bit (LB) for each cache block; one migration counter (MCounter) for each cache set; one dead block counter (DBCounter) for the whole SRAM part. 16K DCache has $16 \times 1024 / 64 = 256$ blocks. Since it is 8-way associative, there are totally $256 / 8 = 32$ sets. Since the volatile part of DCache has $256 / 2 = 128$ blocks, DBCounter needs 7 bits. Besides, each MCounter needs 3 bits. Therefore, the overhead for DCache is $2 \times 256 + 3 \times 32 + 7 = 615$ bits. For ICache, only a LB is needed for each block, thus the area overhead for ICache is 256 bits. Therefore, the total overhead is $256 + 615 = 871$ bits.
2. An age counting module is maintained for checkpointing the clean volatile part of both ICache and DCache. Since there are totally 32 cache sets, 6 bits can be attached for each set to store the age. Therefore the memory area for counting the priority is calculated as $6 \times 64 \times 2 = 768$ bits.
3. The overhead of the dead block prediction we employed in this paper needs less than 1024 bits.

Totally, the area overhead is $871 + 768 + 1024 = 2663$ bits = 2.6K. The percentage of hardware overhead is about $2.6K / 32K = 8.1\%$. Compared to the efficiency of the proposed architecture, the hardware overhead is trivial.

6. EXPERIMENTAL EVALUATION

In this section, we will first present the experiment setup in sub-section 6.1. Then, the evaluation results will be presented in sub-section 6.2.

6.1 Experiment Setup

The experiments are carried out on gem5 simulator [2]. The proposed SRAM and STT-RAM based hybrid cache architecture and policies are implemented in gem5 simulator. Table 3 details the experimental system configuration. The hybrid cache architectures to be evaluated include three types of SRAM/STT-RAM set configuration including 4/4, 5/3, and 6/2. We obtain STT-RAM and SRAM parameters using NVSim [3].

Table 3: System configuration

Component	Description
Processor	1.0GHZ NV processor, 1 core
Cache	Private L1 cache (16K hybrid I-cache and 16K hybrid D-cache), private, 64-byte block size, 8-way associative, 2 cycles access time (write to STT-RAM: 20 cycles), LRU, write-back
Main Memory	30 cycles access time

Table 4: Characteristics of SRAM and STT-RAM Caches (22nm, temperature=350K)

Memory Type	16K SRAM	16K STT-RAM
Area(mm^2)	0.076	0.028
Read Latency (ns)	1.230	1.956
Write Latency (ns)	1.210	10.500
Read Energy (nJ/access)	0.006	0.124
Write Energy (nJ/access)	0.002	0.515
leakage power (mW)	18.972	3.014

The storage capacitor is set to be able to support checkpointing a quarter of the whole cache. The baselines for comparison are two existing cache architectures that are proposed for NVP: pure STT-RAM based cache architecture as proposed in [16] and SRAM based cache architecture as proposed in [12].

Table 5: Characteristics of benchmarks

Bench.	Instr. #	Mem reads #	Mem writes #
<i>qsort</i>	469467835	73791088	60602059
<i>susan</i>	301988532	79999803	1045712
<i>dijkstra</i>	49870857	13422737	5004930
<i>patricia</i>	609908196	118364722	94550973
<i>sha</i>	111653876	24586223	10127992
<i>rijndael</i>	485044372	160386369	97659918
<i>basicmath</i>	277951743	24217872	23164606
<i>fft</i>	1405211413	146740685	139188555
<i>soplex</i>	72255874	14663565	5234462
<i>specrand</i>	93348662	21802840	11392014
<i>libquantum</i>	290069996	31977967	16944750
<i>gobmk</i>	20495669	1318011	3334960

Eight benchmarks from Mibench [4] and four benchmarks from the SPEC CPU2006 suite [5] are selected for evaluation and their characteristics are shown in Table 5. In this table, the first eight rows show benchmarks from Mibench and the last four rows show benchmarks from SPEC CPU2006. The benchmarks are chosen because Mibench is a very representative benchmark suite for embedded system applications, while SPEC CPU2006 suite is for CPU-intensive general purpose microprocessors.

6.2 Results

The performance of the proposed hybrid cache architecture will be evaluated from two aspects: first, the programs are executed in environment of stable power supply; second, the program is interrupted frequently by power failures.

6.2.1 Execution Under Relatively Stable Power

In this section, we will evaluate the proposed hybrid cache under relatively stable power. The cache performance, energy consumption, and miss rate will be analyzed to evaluate the proposed cache policies.

Figure 9: Comparison of execution time under different cache architectures

Performance Evaluation Figure 9 shows the execution time of 12 benchmarks under 5 cache settings when the power supply is relatively stable. Here, all caches are 32 KB. In this figure, the performance of the other four cache architectures are normalized based on the non-volatile cache architecture. From this figure, we can see that non-volatile cache architecture has the worst performance of the five because of its high write latency. The performance of the proposed hybrid cache architectures is 18% better than the non-volatile cache architecture on average. The middle three architectures are based on different ratios of SRAM and STT-RAM, which are 4/4, 5/3, and 6/2. Normally, the 6/2 architecture works better than the other two, since the larger the size of SRAM is, the more blocks will be placed and accessed in SRAM. However, it does not mean 6/2 architecture is the best, because SRAM has higher leakage energy than STT-RAM while STT-RAM has three or more times larger density than SRAM. What is more, in order to perform successful checkpointing, dirty blocks need to be written back to the main memory more frequently in 6/2 configuration. Therefore, the selection of hybrid architectures also need to take into account the requirement of size, energy, and main memory stress.

Energy Consumption Evaluation Figure 10 shows the energy consumption comparison between pure non-volatile cache, hybrid cache, and pure SRAM cache. For each benchmark, there are three bars. In each bar, the solid upper part shows the dynamic energy consumption and the textured bottom part shows the leakage energy consumption for each cache architecture. From the figure, we can see that pure non-volatile cache incurs the largest dynamic energy consumption and lowest leakage energy consumption. On the other hand, pure volatile cache incurs largest leakage energy and lowest dynamic energy. Hybrid cache's energy consumption is in the middle of the other two configurations. This is because the leakage power of SRAM is more than six times of STT-RAM as shown in Table 4.

From the energy comparison results, we can see that the proposed hybrid cache is more energy efficient than the pure non-volatile cache. However, it does not mean the proposed hybrid cache is less energy efficient than the volatile cache. These results are achieved when the processor is running; when it is idle,

Figure 10: Comparison of energy consumption

the leakage energy is also consumed for volatile cache which is not negligible.

Table 6: Overall Miss Rate Comparison with Pure Nonvolatile Cache

Benchmarks	Nonvolatile	Proposed	Increased
<i>qsort</i>	0.496%	0.505%	0.008%
<i>susan</i>	0.014%	0.023%	0.009%
<i>dijkstra</i>	0.526%	1.370%	0.844%
<i>patricia</i>	0.363%	0.519%	0.155%
<i>sha</i>	0.001%	0.010%	0.009%
<i>rijndael</i>	0.041%	0.043%	0.001%
<i>basicmath</i>	0.093%	0.291%	0.198%
<i>fft</i>	0.053%	0.086%	0.033%
<i>soplex</i>	6.399%	8.579%	2.180%
<i>sperand</i>	0.001%	0.244%	0.243%
<i>libquantum</i>	14.423%	12.075%	-2.347%
<i>gobmk</i>	16.140%	16.194%	0.054%
Average			0.116%

Miss Rate Evaluation Table 6 shows the cache miss rate comparison between the proposed hybrid cache and the pure non-volatile cache. We want to figure out if the implementation of checkpointing aware cache replacement policy increases cache miss rate. From the table, we can see that the increased miss rate is negligible. The largest miss rate increase is 2.180%. One of the benchmark, *libquantum*, actually shows decreased miss rate. On average, the miss rate only increases by 0.116%.

Table 7 shows the cache miss rate comparison between the proposed hybrid cache and the SRAM [12] based cache of the same size. Table 7, we can see that the cache miss rate of proposed hybrid cache architecture is lower than the SRAM based cache, and the proposed cache architecture can decrease the cache miss rate by 0.394% on average. This improvement over the baseline cache is because the nonvolatile memory in the SARM based cache is not for cache access and is only used to store volatile states, and the actual working SRAM part is half the size of the proposed architecture. On the contrary, the proposed hybrid cache architecture fully takes advantage of the nonvolatile memory.

Table 7: Overall Miss Rate Comparison with SRAM cache

Benchmarks	Volatile	Proposed	Increased
<i>qsort</i>	0.57%	0.505%	-0.063%
<i>susan</i>	0.02%	0.023%	0.007%
<i>dijkstra</i>	2.13%	1.370%	-0.759%
<i>patricia</i>	1.38%	0.519%	-0.859%
<i>sha</i>	0.12%	0.010%	-0.108%
<i>rijndael</i>	0.73%	0.043%	-0.690%
<i>basicmath</i>	0.21%	0.291%	0.085%
<i>fft</i>	0.06%	0.086%	0.027%
<i>soplex</i>	8.21%	8.579%	0.370%
<i>sperand</i>	0.58%	0.244%	-0.332%
<i>libquantum</i>	14.45%	12.075%	-2.377%
<i>gobmk</i>	16.22%	16.194%	-0.031%
Average			-0.394%

6.2.2 Execution Frequently Interrupted

In this section, the checkpointing aware ability of the 4/4 cache architecture will be evaluated in the scenario where there are frequent power failures.

The power failures are simulated by imputing two different power traces where power failures happen at different frequencies. In the first power trace, a power failure happens about every 500ms; in the second power trace, a power failure happens every 200ms. The frequencies of both two power traces are set quite large to evaluate the performance. In reality, power failures do not happen so frequently as the two power traces used here. Therefore, the results are quite conservative. It takes much less time for a benchmark to run on the hybrid cache architecture in normal energy harvesting systems.

Figure 11: Execution time under frequent power failures

Figure 11 shows the performance of the proposed hybrid cache architecture when there are frequent power failures. In this figure, the first column shows the execution time of non-volatile cache architecture, the second and third columns show the execution time of the hybrid cache architecture when power failures happen every 200ms and 500ms, separately. From this figure, we can see that even facing with radical frequent power failures, the proposed cache architecture outperforms the pure non-volatile cache architecture. In the environment of less frequent power failures, the checkpointing aware hybrid architecture works better than when there are more frequent power failures.

Figure 12: Comparison of energy consumption under frequent power failures

Figure 12 shows the energy consumption of the proposed hybrid cache architecture when there are frequent power failures. In this figure, the first column shows the execution time of non-volatile cache architecture, the second and third columns show the energy consumption of the hybrid cache architecture when power failures happen every 200ms and 500ms, separately. For each column, the solid upper part shows the dynamic energy consumption and the textured bottom part shows the leakage energy consumption for each cache architecture. From this figure, we can see that even facing with radical frequent power failures, the proposed cache architecture outperforms the pure non-volatile cache architecture for most benchmarks. When power failures happen every 200ms, more energy is consumed for hybrid cache than when power failures happen every 500ms. This is because, leakage energy increases with execution time while dynamic energy increases with more memory accesses generated by checkpoints.

From the experimental results, we can see that the proposed hybrid cache works efficiently for energy harvesting powered systems. Compared with existed cache architecture, it has properties of high performance, relatively low energy consumption, and instant resumption as shown in Table 1. Besides, this hybrid cache supports reliable checkpointing because we can control the volatile states in the SRAM to be checkpointed.

7. CONCLUSION

Hybrid SRAM/STT-RAM cache is a promising candidate to be employed in future generation energy harvesting embedded systems in Internet of Things (IoT), due to its fast access, high-density, and low leakage. To guarantee successful checkpointing with given limited capacitor storage, we propose proactively write back policy directed by monitoring the state of the whole SRAM cache part and the migration state of each associative set, so that the percentage of dirty blocks always remains within the checkpointing ability. To fully utilize the size of STT-RAM, we propose to migrate less recently used live victim blocks to it while retaining enough space in STT-RAM for checkpointing of SRAM states. When power is down, the proposed efficient checkpointing policy is utilized to first checkpoint dirty blocks and then important clean blocks in SRAM. After power returns, the system resumes fast without special restoration process.

8. REFERENCES

- [1] <http://www.brookings.edu/blogs/techtank/posts/2015/06/8-future-of-iot-part-1>.
- [2] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, et al. The gem5 simulator. *ACM SIGARCH Computer Architecture News*, 39(2):1–7, 2011.
- [3] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi. Nvsm: A circuit-level performance, energy, and area model for emerging nonvolatile memory. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 31(7):994–1007, 2012.
- [4] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. Mibench: A free, commercially representative embedded benchmark suite. In *Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop on*, pages 3–14. IEEE, 2001.
- [5] J. L. Henning. Spec cpu2006 benchmark descriptions. *ACM SIGARCH Computer Architecture News*, 34(4):1–17, 2006.
- [6] X. Jiang, J. Polastre, and D. Culler. Perpetual environmentally powered sensor networks. In *Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on*, pages 463–468, 2005.
- [7] A. Jog, A. K. Mishra, et al. Cache revive: architecting volatile stt-ram caches for enhanced performance in cmps. In *Proceedings of the 49th Annual Design Automation Conference*, pages 243–252. ACM, 2012.
- [8] W. kei Yu, S. Rajwade, S.-E. Wang, B. Lian, G. Suh, and E. Kan. A non-volatile microcontroller with integrated floating-gate transistors. In *2011 IEEE/IFIP 41st International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pages 75–80, 2011.
- [9] Q. Khan and S. Bang. Energy Harvesting for Self Powered Wearable Health Monitoring System. *Health, San Francisco*, pages 1–5, 2009.
- [10] J. Kymissis, C. Kendall, J. Paradiso, and N. Gershenfeld. Parasitic power harvesting in shoes. *Digest of Papers. Second International Symposium on Wearable Computers (Cat. No. 98EX215)*, pages 132–139.
- [11] V. Leonov. Thermoelectric Energy Harvesting of Human Body Heat for Wearable Sensors. *IEEE Sensors Journal*, 13(6):2284–2291, June 2013.
- [12] H. Li, Y. Liu, Q. Zhao, Y. Gu, X. Sheng, G. Sun, C. Zhang, M. 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 the 2015 Design, Automation & Test in Europe Conference & Exhibition, DATE*, pages 7–12, 2015.
- [13] H. Liu, M. Ferdman, J. Huh, and D. Burger. Cache bursts: A new approach for eliminating dead blocks and increasing cache efficiency. In *Proceedings of the 41st annual IEEE/ACM International Symposium on Microarchitecture*, pages 222–233. IEEE Computer Society, 2008.
- [14] 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*, page 150, 2015.
- [15] Y. Liu, F. Suy, Z. Wang, and H. Yang. Design exploration of inrush current aware controller for nonvolatile processor. In *Non-Volatile Memory System and Applications Symposium (NVMSA), 2015 IEEE*, pages 1–6. IEEE, 2015.
- [16] 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 IEEE International Symposium on High Performance Computer Architecture, HPCA 2015, February 7-11, 2015*, pages 526–537, 2015.
- [17] H. Nakamoto, D. Yamazaki, T. Yamamoto, H. Kurata, S. Yamada, K. Mukaida, T. Ninomiya, T. Ohkawa, S. Masui, and K. Gotoh. A passive uhf rf identification cmos tag ic using ferroelectric ram in 0.35-um technology. *IEEE Journal of Solid-State Circuits*, 42(1):101–110, 2007.
- [18] C. Pan, S. Gu, M. Xie, C. Xue, and J. Hu. Wear-leveling aware page management for non-volatile main memory on embedded systems. *IEEE Transactions on Multi-Scale Computing Systems*, 2016.
- [19] C. Pan, M. Xie, J. Hu, Y. Chen, and C. Yang. 3m-pcm: Exploiting multiple write modes mlc phase change main memory in embedded systems. In *Proceedings of the 2014 International Conference on Hardware/Software Codesign and System Synthesis, CODES '14*, 2014.
- [20] C. Pan, M. Xie, J. Hu, M. Qiu, and Q. Zhuge. Wear-leveling for pcm main memory on embedded system via page management and process scheduling. In *2014 IEEE 20th International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 1–9, 2014.
- [21] C. Pan, M. Xie, C. Yang, Z. Shao, and J. Hu. Nonvolatile main memory aware garbage collection in high-level language virtual machine. In *2015 International Conference on Embedded Software (EMSOFT)*, pages 197–206, 2015.
- [22] J. Paradiso and M. Feldmeier. A compact, wireless, self-powered pushbutton controller. *Ubicomp 2001: Ubiquitous Computing*, 2001.
- [23] B. Ransford, S. S. Clark, M. Salajegheh, and K. Fu. Getting things done on computational rfids with energy-aware checkpointing and voltage-aware scheduling. In *HotPower*, 2008.
- [24] B. Ransford, J. Sorber, and K. Fu. Mementos: system support for long-running computation on rfid-scale devices. *ACM SIGPLAN Notices*, 47(4):159–170, 2012.
- [25] N. S. Shenck and J. A. Paradiso. Energy scavenging with shoe-mounted piezoelectrics. *Ieee Micro*, 21(3):30–42, 2001.
- [26] J. Taneja, J. Jeong, and D. Culler. Design, modeling, and capacity planning for micro-solar power sensor networks. In *Proceedings of the 7th international conference on Information processing in sensor networks*, pages 407–418. IEEE Computer Society, 2008.
- [27] C. Wang, N. Chang, Y. Kim, S. Park, Y. Liu, and H. Lee. Storage-less and converter-less maximum power point tracking of photovoltaic cells for a nonvolatile microprocessor. *ASP-DAC*, (2013035079):379–384, 2014.
- [28] J. Wang, X. Dong, and Y. Xie. Point and discard: A hard-error-tolerant architecture for non-volatile last level caches. In *2012 49th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 253–258, June 2012.
- [29] J. Wang, Y. Liu, H. Yang, and H. Wang. A compare-and-write ferroelectric nonvolatile flip-flop for energy-harvesting applications. In *2010 International Conference on Green Circuits and Systems (ICGCS)*, pages 646–650, June 2010.
- [30] Y. Wang, Y. Liu, S. Li, D. Zhang, B. Zhao, M.-F. Chiang, Y. Yan, B. Sai, and H. Yang. A 3us wake-up time nonvolatile processor based on ferroelectric flip-flops. In *ESSCIRC (ESSCIRC), 2012 Proceedings of the*, pages 149–152, 2012.
- [31] Z. Wang, D. Jimenez, C. Xu, G. Sun, and Y. Xie. Adaptive placement and migration policy for an stt-ram-based hybrid cache. In *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, pages 13–24, Feb 2014.
- [32] A. S. Weddell, M. Magno, et al. A survey of multi-source energy harvesting systems. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 905–908, 2013.
- [33] X. Wu, J. Li, L. Zhang, E. Speight, R. Rajamony, and Y. Xie. Hybrid cache architecture with disparate memory technologies. In *Proceedings of the 36th Annual International Symposium on Computer Architecture, ISCA '09*, pages 34–45, 2009.
- [34] M. Xie, C. Pan, J. Hu, C. J. Xue, and Q. Zhuge. Non-volatile registers aware instruction selection for embedded systems. In *2014 IEEE 20th International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 1–9, 2014.
- [35] M. Xie, C. Pan, J. Hu, C. Yang, and Y. Chen. Checkpoint-aware instruction scheduling for nonvolatile processor with multiple functional units. In *The 20th Asia and South Pacific Design Automation Conference*, pages 316–321, 2015.
- [36] M. Xie, M. Zhao, C. Pan, J. Hu, Y. Liu, and C. J. Xue. Fixing the broken time machine: Consistency-aware checkpointing for energy harvesting powered non-volatile processor. In *Proceedings of the 52nd Annual Design Automation Conference, DAC '15*, pages 184:1–184:6, 2015.
- [37] J. Zhao, C. Xu, and Y. Xie. Bandwidth-aware reconfigurable cache design with hybrid memory technologies. In *2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 48–55, Nov 2011.
- [38] M. Zhao, Q. Li, M. Xie, Y. Liu, J. Hu, and C. J. Xue. Software assisted non-volatile register reduction for energy harvesting based cyber-physical system. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition, DATE '15*, pages 567–572, 2015.
- [39] M. Zwerg, A. Baumann, R. Kuhn, M. Arnold, R. Nerlich, M. Herzog, R. Ledwa, C. Sichert, V. Rzehak, P. Thanigai, and B. Eversmann. An 82ua/mhz microcontroller with embedded feram for energy-harvesting applications. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2011 IEEE International*, pages 334–336, Feb 2011.