Enabling Reliable, Efficient, and Secure Computing for Energy Harvesting Powered IoT Devices

by

Mimi Xie

B.E., Chongqing University, 2010

M.S., Chongqing University, 2013

Submitted to the Graduate Faculty of

the Swanson School of Engineering in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2019

UNIVERSITY OF PITTSBURGH

SWANSON SCHOOL OF ENGINEERING

This dissertation was presented

by

Mimi Xie

It was defended on

May 30, 2019

and approved by

Jingtong Hu, Ph.D., Assistant Professor
Department of Electrical and Computer Engineering

Samuel Dickerson, Ph.D., Assistant Professor
Department of Electrical and Computer Engineering

Feng Xiong, Ph.D., Assistant Professor
Department of Electrical and Computer Engineering

Jun Yang, Ph.D., Professor
Department of Electrical and Computer Engineering

Youtao Zhang, Ph.D., Associate Professor
Department of Computer Science

Dissertation Director: Jingtong Hu, Ph.D., Assistant Professor
Department of Electrical and Computer Engineering

Enabling Reliable, Efficient, and Secure Computing for Energy Harvesting Powered IoT Devices

Mimi Xie, PhD

University of Pittsburgh, 2019

Energy harvesting is one of the most promising techniques to power devices for future generation IoT. While energy harvesting does not have longevity, safety, and recharging concerns like traditional batteries, its instability brings a new challenge to the embedded systems: the energy harvested from environment is usually weak and intermittent. With traditional CMOS based technology, whenever the power is off, the computation has to start from the very beginning. Compared with existing CMOS based memory devices, emerging non-volatile memory devices such as PCM and STT-RAM, have the benefits of sustaining the data even when there is no power. By checkpointing the processor's volatile state to non-volatile memory, a program can resume its execution immediately after power comes back on again instead of restarting from the very beginning with checkpointing techniques.

However, checkpointing is not sufficient for energy harvesting systems. First, the program execution resumed from the last checkpoint might not execute correctly and causes inconsistency problem to the system. This problem is due to the inconsistency between volatile system state and non-volatile system state during checkpointing. Second, the process of checkpointing consumes a considerable amount of energy and time due to the slow and energy-consuming write operation of non-volatile memory. Finally, connecting to the internet poses many security issues to energy harvesting IoT devices. Traditional data encryption methods are both energy and time consuming which do not fit the resource constrained IoT devices. Therefore, a light-weight encryption method is in urgent need for securing IoT devices.

Targeting those three challenges, this dissertation proposes three techniques to enable reliable, efficient, and secure computing in energy harvesting IoT devices. First, a consistency-aware checkpointing technique is proposed to avoid inconsistency errors generated from the inconsistency between volatile state and non-volatile state. Second, checkpoint aware hybrid

cache architecture is proposed to guarantee reliable checkpointing while maintaining a low checkpointing overhead from cache. Finally, to ensure the security of energy harvesting IoT devices, an energy-efficient in-memory encryption implementation for protecting the IoT device is proposed which can quickly encrypts the data in non-volatile memory and protect the embedded system physical and on-line attacks.

Table of Contents

List of Tables

List of Figures

Acknowledgement

First of all, I want to thank my Ph.D. advisor Prof. Jingtong Hu for the continuous support of my Ph.D. study and research. I appreciate all his ideas, patience, enthusiasm, and perseverance in academic pursuits. He provided me immense hands-on help when I was in my early years of PhD study. He helped revise my first paper word by word. I appreciate his constructive advice on my career plan and job hunting and his encouragement when I was not confident of my research ideas.

I am especially grateful to Prof. Youtao Zhang, for being my dissertation committee member, and for his recommendation and help in my job hunting. I also reserve my sincere gratitude to Prof. Jun Yang, Prof. Samuel Dickerson, and Prof. Feng Xiong, for serving on my PhD committee and providing insightful advice and instructions to my research and dissertation.

Moreover, I thank all my coauthors Prof. Chengmo Yang, Prof. Yiran Chen, Prof. Mengying Zhao, Prof. Chun Jason Xue, Prof. Yongpan Liu, Dr. Shuangchen Li, and all the others around the world for their generous contributions to the works in this dissertation.

I have been fortunate to work with my group comrades, Chen Pan, Xinyi Zhang, Zhenge Jia, Yawen Wu, and Zhenpeng Wang, who are always generous with their time and knowledge. They have made my life vivid and beautiful during the past years.

Last but not least, I would also like to thank my parents for their continuous support and unconditional love to me. I dedicate this dissertation to them!

1.0 Introduction

The vision of Internet of things is to connect everything with modern technologies to improve the wellbeing of whole society. Thanks to the development of integrated circuits, the size of computer systems has shrinked from a mainframe to a coin size little small sensor. While the vision is promising and exciting, there are several challenges in achieving this goal. One of the imminent challenges is how to power all these small devices. For all these devices, a battery is no longer a favorable solution. First of all, it is difficult to shrink the size of battery while maintaining the required power supply. Second, closely wearing many batteries will pose safety and health concerns for users. What is more, charging all these batteries every one or two days will give users bad experiences. Therefore, researchers are actively pursuing power alternatives and trying to replace battery completely. Out of all possible solutions, energy harvesting is one of the most promising techniques to meet both the size and power requirements of IoT devices.

Energy harvesting technologies generate electric energy by harvesting energy from ambient environment using direct energy conversion techniques. Examples of power sources include kinetic, electromagnetic radiation (including light and RF), and thermal energy. The obtained energy can be used to recharge a capacitor or, in some cases, to directly power the electronics. However, there is an intrinsic drawback with harvested energy. They are all *unstable*. Its instability brings a new challenge to the embedded systems: the energy harvested from environment is usually weak and intermittent. With traditional CMOS based technology, whenever the power is off, all computation state is lost and the computation has to start from the very beginning. With frequent power outages, the processor execution will be interrupted frequently. Frequent turning-off and booting-up will place an extra burden on a limited power budget.

To address this challenge, researchers have employed non-volatile memory in energy harvesting embedded systems. Compared with existing CMOS based memory devices, emerging non-volatile memory devices such as FRAM, PCM, and STT-RAM, have the benefits of sustaining the data even when there is no power. Their non-volatility allows fast recovery from

power failures, thus enabling long running computations even with unreliable power sources. When there is a power failure, the processor's volatile state is backed up into non-volatile memory, which is called checkpointing. When power comes back again, the state will be copied back from the closest checkpoint next time. In this way, a program can resume its execution from the last checkpoint instead of restarting from the very beginning with checkpointing techniques.

These distinctive advantages, along with their great market potential, are driving the use of NVMs in embedded systems across various layers on the memory hierarchy. With non-volatile memory, we can turn off the processor and resume from where was left. In this way, we can either turn off processor on purpose to save energy or passively survive unstable power. For example, FRAM has been widely used to deploy scratchpad memory taking advantage of its non-volatility characteristic for intermittently powered embedded systems. In this kind of system, the on-chip non-volatile scratchpad memory is used for backing up the volatile system state upon power failure, which is critical for running long computation tasks. STT-RAM has been widely used to deploy energy-efficient on-chip cache with small size for embedded system due to its high density, low leak power dissipation, and high read speed. PCM has been widely researched as a replacement for DRAM as off-chip main memory.

Despite these great benefits over SRAM and DRAM, deploying NVMs in different levels of memory hierarchy is faced with different challenges originating from the more expensive writes compared to read operation and the feature of non-volatility itself.

1.1 Challenges in Energy Harvesting Powered IoT devices

Energy harvesting powered IoT devices are faced with reliability, efficiency, and security issues. This dissertation considers three main challenges in enabling the efficient working of energy harvesting powered embedded systems.

First, the correctness of the program execution must be preserved across power interruptions. Our initial investigation shows that existing programs could cause data inconsistency and therefore a wrong result due to checkpointing and resuming. While the employment

of non-volatile memory as on-chip memory enables continuous computation by backing up (checkpointing) the volatile system state in energy harvesting embedded systems, it creates inconsistency between volatile system state and non-volatile system state can bring serious system errors in battery-less embedded system. This problem results from partly finished checkpoint and rollback recovery from last successful checkpoint if the current checkpoint fails. Besides, process of checkpointing consumes a considerable amount of energy and time due to the slow and energy-consuming write operation of non-volatile memory. Therefore, how to efficiently save the program execution state in terms of both time and energy is vital for practical application of energy harvesting embedded systems.

Second, although non-volatile cache is preferable as a solution for energy harvesting embedded systems to achieve low energy, high density, and instant resumption when power recovers. Its slow write severely degrades the system performance because of the slow and energy consuming write to non-volatile memory. The solution of volatile cache, however, requires frequent checkpointing a large amount of volatile system data which consumes a massive amount of energy and time. Therefore, how to efficiently save the program execution state in terms of both time and energy is vital for practical application of non-volatile processors. Existing approach that simply saves all the volatile system state is not a desired solution. This will place severe burden on already constrained energy supply.

Third, there is no light-weight encryption method for securing the energy harvesting powered IoT devices. Although the integration technology allows processor vendors to integrate the encryption engine into the embedded processor/microcontroller to make the embedded system more secure, existing engines are energy and time consuming for resource and energy limited energy havesting IoT devices. An light-weight encryption method is in urgent need.

In this dissertation, these challenges are addressed through software and hardware optimization and designs. The goal is to enable reliable, efficient, and secure computing for energy harvesting powered IoT devices.

1.2 Research Contributions

Research contributions for this dissertation can be concluded as:

- To eliminate inconsistency errors in non-volatile memory based intermittently powered embedded system, a systematic consistency-aware checkpointing mechanism is proposed for energy harvesting powered non-volatile processor with low checkpointing overhead. Specifically, the proposed checkpointing mechanism makes the following contributions:
  - A consistency-related error locating mechanism is proposed to find all the potential error pairs and all the program paths between each error pair;
  - A consistency-aware checkpointing algorithm is designed to eliminate all error pairs and it generates the minimal number of checkpoints;
  - Detailed experimental evaluation is implemented to demonstrate the efficiency of the proposed consistency-aware checkpointing mechanism.
- A checkpoint aware hybrid cache architecture to guarantee reliable checkpointing while maintaining a low checkpointing overhead from cache.
  - A hybrid cache architecture built with SRAM and STT-RAM is proposed , and STT-RAM is fully utilized not only for normal cache access but also for volatile SRAM checkpoint.
  - A write intensive and dead block predictor is proposed for directing the cache line placement policies.
  - Replacement and migration policies are designed 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.
  - An efficient checkpointing policy is proposed to save all necessary volatile blocks to STT-RAM before capacitor energy is depleted upon a power failure.
- AIM, a novel **A**ES **I**n-**M**emory encryption architecture is proposed for fast and energy efficient data encryption. Embracing the benefit of the processing-in-memory (PIM) architecture, the proposed encryption architecture takes advantage of large internal memory bandwidth, vast bitline-level parallelism, and low in-situ computing latency. Besides, by eliminating data movement between memory and host, higher energy efficiency is

achieved. To this end, the non-destructive read in NVMs is leveraged for performing efficient XOR operations, which dominate AES. The sense amplifier circuit is modified so that the vector XOR operations can be calculated by performing a memory-read-like operation. After adding lightweight logic gates to the memory peripherals circuitry, we can perform the entire AES procedure in-place.

The major contribution is summarized as follows:

- Different levels (chip, bank, and subarray) of parallelism are explored to provide different design choices in order to satisfy different performance and energy efficiency requirement.
- A novel combined cipher modes is proposed for AIM in order to maintain high parallelism with best performance and reduced area overhead.
- The proposed encryption architecture is evaluated and compared with the state-of-the-art encryption engine. The experimental results show that, compared with state-of-the-art AES encryption engine, AIM can speed up the encryption process by 80× while reducing the energy by 10×. .

1.3 Dissertation Organization

This dissertation proposal is organized as follows:

Chapter 2.0 introduces the architecture of energy harvesting embedded system for this dissertation and presents the related work of non-volatile memory-based energy harvesting embedded systems.

Chapter 3.0 explores the consistency problem in both on-demand and periodical checkpointing schemes and proposed inconsistency aware checkpointing technique to eliminate inconsistency errors. Besides, live variable reduction and adaptive on-line checkpointing techniques are proposed to reduce checkpointing overhead.

Chapter 4.0 proposes checkpoint aware hybrid cache architecture to guarantee reliable checkpointing while maintaining a low checkpointing overhead from cache.

Chapter 5.0 proposes a light-weight in-memory encryption implementation technique that encrypts the data on non-volatile main memory to protect the embedded system from physical attack or on-line attack.

Chapter 6.0 summarizes this dissertation.

2.0 Energy Harvesting Embedded System

In this chapter, the system architecture of energy harvesting embedded system will be presented and the design of the energy harvesting embedded systems will be explored followed with the related work.

2.1 System Architecture

This dissertation targets on embedded systems that are powered directly by energy harvesting [72] technologies. Since the power supply is intermittent, to defend against frequent power interruptions, the states in volatile memory need to be checkpointed periodically or on-demand when detecting a low voltage. The targeting energy harvesting based embedded

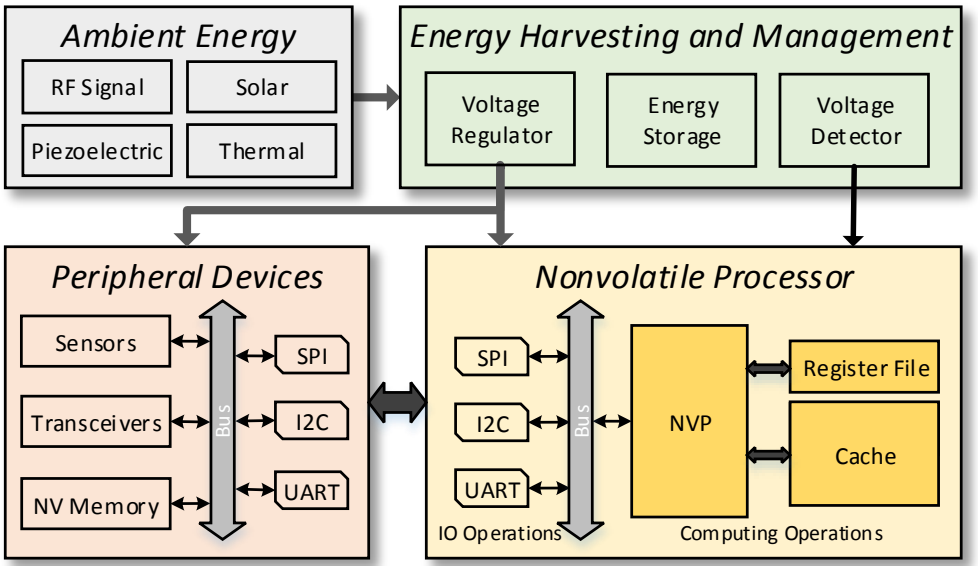

Figure 1: Energy harvesting system architecture

system is shown in Figure 1. This is a typical system architecture with non-volatile memory for energy harvesting powered embedded systems. In this architecture, the system is powered with energy harvested from ambient sources, such as solar energy, thermal energy,

piezoelectric, or radio frequency (RF). Besides the energy harvesting module, there is also a energy management module where there is an energy storage, a voltage regulator, and a voltage detector. The energy storage is powering the whole system including the processor and the peripheral devices. Whenever the voltage detector detects the voltage bellow a certain threshold, it will send a interrupt to the non-volatile processor. Then the non-volatile processor will save the content to the non-volatile memory.

The non-volatile processor (NVP) consists of the processing unit, non-volatile register file, and non-volatile cache as on-chip memory. Besides the on-chip memory, there is also a non-volatile off-chip memory as main memory. 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.

The small 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 supported with the proposed cache architecture. Besides, checkpointing is always necessary. This is because without checkpointing, although the states in SRAM can retain for a while powered with the capacitor, everything in SRAM will be lost if power does not come back right after the energy in the capacitor depletes.

2.2 Non-volatile Processor

2.2.1 Non-volatile Register File

Due to frequent usage, the register file goes through massive accesses, hence a volatile register file cannot be made with pure non-volatile memory considering energy efficiency. To achieve high efficiency, researchers designed FRAM based nonvolatile flip-flops (NVFFs) [78]. The NVFF structure is shown in Figure 2. 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 register file enables the processor's registers to

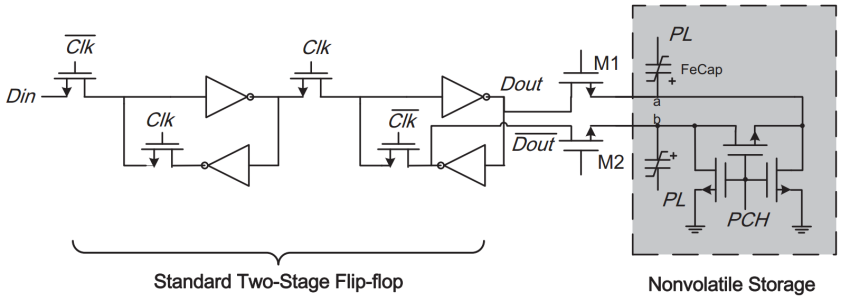

Figure 2: Ferroelectric Flip-Flop [77]

be non-volatile and the execution can be quickly recovered from power failure. When the content of the volatile register is lost due to power failure, the content in the NV storage can be copied back into the volatile register. Besides this hardware solution, the values in the volatile register file can be moved to a non-volatile memory upon power outages to save volatile state.

2.2.2 Non-volatile On-chip Memory

There are different ways to integrate non-volatile memory into the processor. For example, Texas instrument MSP 430 microcontroller already have FRAM integrated into their on chip memory as a scratch pad memory. TI's MSP430FRxx series microcontrollers [27] is an existing microprocessors with similar architecture. Or we can integrate non-volatile memory as a cache to checkpoint the content [47]. The on-chip cache can be deployed with STT-RAM, which has fast access speed and long lifetime. Since the on-chip memory is non-volatile, the content does not need to be checkpointed. This dissertation targets on the system that integrateds cache as on-chip memory.

2.3 Related work

2.3.1 Energy Harvesting

Energy harvesting technologies are promising long-lasting replacements for batteries in embedded systems. Ambient sources such as solar [69], motion [64], radio-frequency (RF) electromagnetic radiation, and thermal gradients [35, 37] 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 [69] and body heat ($2.4 \sim 4.8W$), should be able to provide sufficient power to drive the devices at low duty cycles [35]. For example, [69] designs a micro-solar power sensor network, [73] proposes a non-volatile microprocessor powered with a solar energy harvesting system even under low solar irradiance, and [59] designs a self-powered pushbutton controller which functions with a piezoelectric conversion mechanism. Even though the harvested power is lower than the power required by the complete system, it is still possible to operate the system with proper energy management.

2.3.2 Non-volatile memory

Energy harvesting power sources can lead to frequent power failure and state loss, 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 non-volatile (NV) memory into energy-harvesting devices to store the execution states [101, 60, 62, 43] because of their features of non-volatility. Non-volatile memory based embedded systems bring promising opportunities to the computing paradigm since they have extremely low leakage power and better scaling than CMOS technology. A lot of promising candidates such as Phase Change Memory (PCM) [31, 30, 54], Spin-Transfer Torque RAM (STT-RAM) [25, 100, 39] and Ferroelectric RAM (FRAM) [66, 92, 5] are currently under active research and development. These NVMs have several significant advantages over traditional static and dynamic RAMs such as high density, lower leakage power, low-cost, high-scalability, and non-volatility properties. The potential high density of NVMs allows a further reduction in chip size and cost for embedded

systems. NVMs consume less standby power than SRAM or DRAM by orders of magnitude, significantly prolonging the battery usage.

2.3.3 Non-volatile Processor

Non-volatile memory based on-volatile processor appears as a promising solution to bridge intermittent program execution under unstable power. There are two kinds of N-V processor. The first kind of 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 [101, 53, 89], 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. [34] propose a non-volatile processor architecture which integrates non-volatile elements into volatile memory at bit granularity. Wang et al. [77] 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 [87], register reduction [98], compare-and-write [75], and instruction selection [86].

The second kind of NV processor employs non-volatile memory as a piece of independent scratchpad memory. The non-volatile memory is used to reserve the system state by backing up all volatile system state upon power failure. After the system comes back from a power failure, the execution states are copied back and thus the program can be resumed efficiently. Zwerg et al. [101] present an ultra-low-power micro controller unit which embedded FRAM as on-chip memory for fast write capability. When power runs out, a charge reserve in a 2nF capacitor is used to complete memory access to FRAM. In this work, a special circuit was also designed to detect power-drop. Checkpointing has been shown to be an efficient methodology for saving the runtime state [60]. The work of Ransford et al. [62] presented a software system for transiently powered RFID-scale devices in which energy-aware state checkpointing is utilized against frequent power losses. This system deployed flash to back

up the regions in use at the checkpoint, while flash has a limited write endurance in the order of 10^5 [33] and the access to flash is quite slow. Mirhoseini et al. [52] also proposed a framework for small-scale battery-less devices with discontinuous energy-harvesting supplies. This work partitions an application into smaller computational blocks and inserts checkpoints at the end of functions and loops to reduce the cost and overhead. The checkpointed data are also stored on on-chip memory in this work.

There have been work on efficiency optimizations for NVP [86] aiming at improving the lifetime and efficiency of NV registers by generating NV register friendly code to reduce writes to NVM. Wang et al. explores hybrid register architecture with both volatile and non-volatile registers to improve checkpoint efficiency through register allocations [76]. In order to reduce the necessary NVM size for checkpointing, data compression based hardware design is proposed to reduce the content to back up [65]. Even though the previous works can improve the checkpointing efficiency, they have potential risk of errors, which severely degrades the functionality and service quality of the energy harvesting powered devices.

2.3.4 Non-volatile Cache

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 [47]. 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 [44] 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 [80, 83]. 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 thus are not resistant to power failure. In this project, a checkpointing aware cache architecture is designed for energy harvesting systems.

2.3.5 Secure Non-volatile Main Memory

Memory Encryption: Encryption has been widely suggested as a solution to secure both DRAM [24] and NVM-based main memory [12, 93]. These implementations perform encryption/decryption when writing/reading a cache line to/from main memory. Though encryption techniques base on Pad-based or Stream cipher encryption where memory access could be overlapped with the Pad or Keystream generation reduces the decryption overhead, the system still suffers, since that overhead is on the critical path (memory read access). Different from them, [14] proposes to perform one-time encryption for smartphones and tablets only when the device is screen-locked. AIM performs a one-time encryption to the main memory system before there is a possible attack (e.g., before power-off). Other than that, it runs as normal main memory without any latency overhead. Furthermore, existing encryption methods rely on a dedicated encryption engine on the processor or in the main memory. AIM takes advantage of in-memory computing, hence achieves better throughput with less energy consumption.

NVM Encryption: There are several work that are particularly optimized for encryption on NVM [12, 68, 93, 41, 26]. I-NVMM [12] proposes to encrypt main memory incrementally. However, our method taking advantage of the PIM architecture outperforms i-NVMM, because i-NVMM relies on the dedicated AES engine on the processor side and limited by its small bandwidth and parallelism. DEUCE [93] and SECRET [68] propose techniques to reduce the bit flips during data encryption, which helps NVM reliability since encryption involves significant amount of expensive writes. Silent Shredder[6] proposes techniques to obviate the writing of zeros to memory pages. Their techniques are orthogonal to AIM, and their method can be applied to AIM to further reduce the encryption energy.

In-Memory Encryption: In-memory encryption is a promising solution for non-volatile memory encryption which has limited research. [17] explores different spintronic devices based memory that could be leveraged to implement logic functions with AES algorithm as a case study. [4] demonstrates the efficiency of AES algorithm on a proposed in-memory processing platform with novel spin Hall effect-driven domain wall motion devices that support both nonvolatile memory cell and in-memory logic design. A recent work, Recryptor [96],

proposes a reconfigurable cryptographic processor using in-memory computing. By replacing a standard SRAM bank with a custom bank with in-memory and near-memory computing, Recryptor provides an IoT platform that accelerates primitive cryptographic operations. Domain wall memory (DWM) is also utilized to perform in-memory encryption [79, 46, 94], where inherent DWM device functions were used to perform the operations for encryption.

3.0 Inconsistency-aware Checkpointing for Energy Harvesting Embedded System

This chapter presents a project that eliminates inconsistency errors in non-volatile memory based energy harvesting embedded system [90]. It is organized as follows. First, the background of this project is introduced, and then the motivation is presented. Next, the details of the proposed techniques are presented including potential error locating and consistency-aware checkpoints insertion. Finally, the experimental results are presented before this project is summarized.

3.1 Background

3.1.1 Energy harvesting

In energy harvesting system, ambient energy is harvested to provide energy for embedded systems through energy harvesters. The harvested power is then regulated to maintain a constant voltage level for the microcontroller. In energy harvesting systems, there is often a small storage capacitor to deal with power failures caused by intermittent power supply. At run time, the energy remaining in this energy buffer can be estimated by measuring its voltage. All components of an energy harvesting system are shown with a block diagram in Figure 3.

The storage capacitor is connected in parallel to the MCU. Therefore, when the storage voltage V_{STORE} is above the input voltage V_{CC} of MCU, the capacitor will begin to provide energy for the MCU. When V_{STORE} drops below a threshold, there is a high probability that there will be a power outage soon. Inspecting the voltage of this capacitor provides important information of transient condition of the harvested energy. In order to prevent against state loss, the states of all volatile registers will be checkpointed to non-volatile memory with the remaining energy in the capacitor. Therefore, the storage should be large

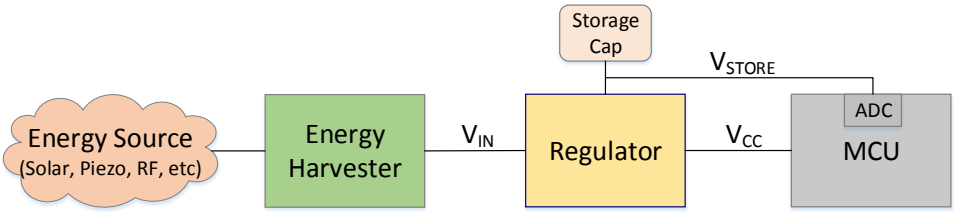

Figure 3: Block diagram of energy harvesting systems

enough to support a complete checkpointing operation that stores all volatile register states to non-volatile memory. For example, if RC is the time constant of the capacitor, V_r is the smallest operational input voltage for the regulator, and V_c is the current voltage across the capacitor, this capacitor is able to provide E_Δ energy for checkpointing.

$$E_\Delta = \frac{1}{2}CV_c^2 - \frac{1}{2}CV_r^2 \tag{3.1}$$

The unstable nature of energy sources leads to frequent power failures and interrupts the program execution in MCU almost every tens to hundreds of milliseconds. Figure 4 gives two different power trace examples of two different energy sources that have different magnitudes and intermittencies. Radio frequency can generate power of several microwatts which varies frequently as shown in Figure 4 (a), while in-door lights can generate power of up to several milliwatts which is more stable than radio frequency as shown in Figure 4 (b). The unstable power can result in power failures almost every tens to hundreds of milliseconds. Therefore, existed energy harvesting systems enable continuous computation across power failures by saving all volatile intermediate results into non-volatile memory and restore them when power recovers. The backing up and recovering processes of the system state have a large influence on the overall performance. However, it is difficult to predict the energy harvesting behavior at run time which makes it difficult to determine when to initiate a checkpoint. Existed checkpointing schemes estimate the harvested energy by measuring

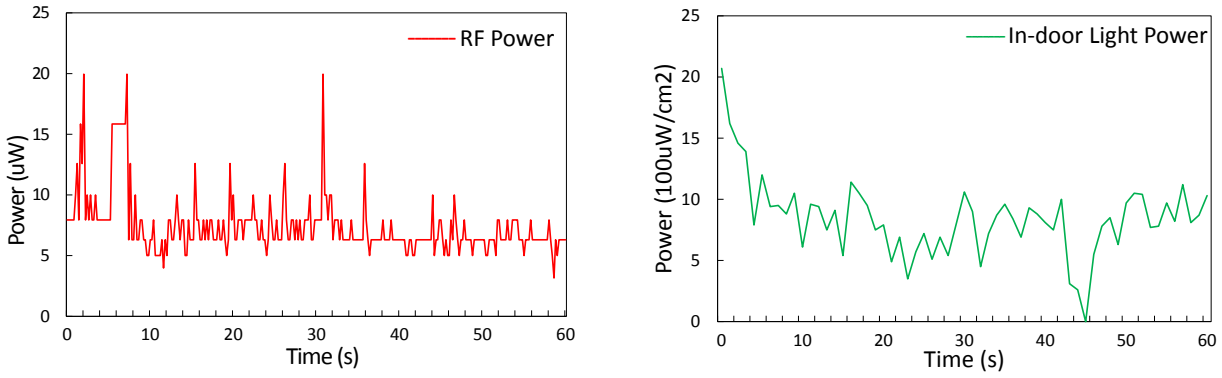

Figure 4: Power traces: a) Wifi RF, b) Light

the voltage level in the storage capacitor, which can be classified into two groups including on-demand checkpointing and dynamic checkpointing.

With existed checkpointing strategies, it is highly possible that the current energy supply is not able to complete one checkpoint leaving the system state partly checkpointed. Besides, power loss can also happen between two adjacent checkpoints. In both two occasions, roll back is needed while inconsistency issues reside in execution without further improvement to the checkpointing strategy. Figure 5 shows these two occasions in the computation progress. In this figure, the supply voltage level is represented with the blue line and the threshold is represented with dashed red line. At each trigger point (green dot), the system will check the voltage level in energy buffer to predict an imminent power failure. If the current voltage is below the threshold, a checkpoint will be initiated.

As shown in this figure, the voltage level is below the threshold at four checkpoints (23s, 32s, 46s, 63s). Therefore, four checkpoints are initiated at these four checkpoints. All four checkpoints succeed except for the checkpoint at 32s because of insufficient energy. During the computation process, there are two rollbacks. One rollback happens because the checkpoint fails at 32s. The other rollback happens because power fails before the trigger point at 83s. Both of the two rollbacks can cause inconsistency error in the computation program and the detailed reason will be discussed in the next section. Besides, setting a

fixed threshold for all checkpoints is of low efficiency, since the amount of modified volatile states keeps changing as the computation continues. Actually, smartly setting trigger points provides an opportunity to analyze the program state and reduce the overhead for each checkpoint [87, 98].

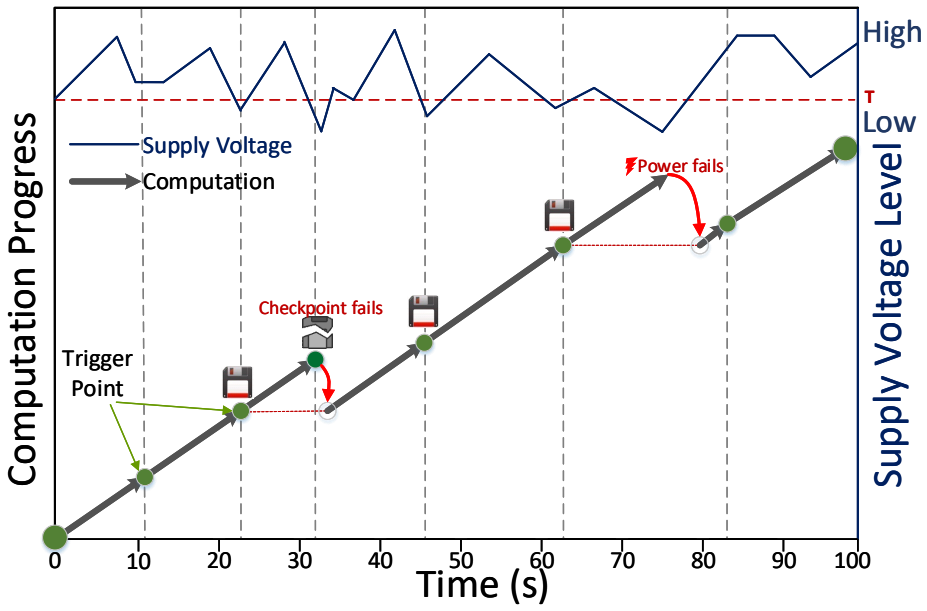

Figure 5: Computation progress.

3.1.2 Checkpointing

The nature of intermittence of harvested energy impedes wide adoption of large software in energy harvesting powered embedded systems. To enable continuous execution of programs across different power cycles, researchers have deployed non-volatile memory into energy-harvesting devices to save the volatile work state [28, 43, 51, 60, 62]. For example, Mementos [62] is a software system for transiently powered RFID-scale devices. This system deploys flash memory to save an snapshot of the volatile state. However, flash memory

has a limited write endurance in the order of 10^5 [33] and access to flash memory is quite slow. Quickrecall [28] deploys pure FRAM to increase the checkpointing efficiency. The access efficiency of FRAM can even catch up with that of SRAM [66], and it has a superior endurance, as long as 10^{14} write cycles. There are mainly two categories of checkpointing schemes including dynamic checkpointing [28, 62] and on-demand checkpointing [7, 8].

3.1.3 Inconsistency

The problem of inconsistency in intermittently powered systems was first discussed in [61]. This work explains that checkpointing is not sufficient in intermittently powered systems and new programming models and system support are needed to address correctness and programmability. DINO [45] was proposed to maintain memory consistency across continuous checkpoints. DINO ensures that non-volatile data remain consistent across reboots with *data versioning* mechanism. Before checkpointing, DINO makes a volatile copy of each non-volatile variable that is potentially inconsistent at task boundary. [45] decomposes programs down to a series of re-executable sections and glues them together with checkpoints of volatile state. The proposed technique in this dissertation decomposes the program on the instruction level and initiates a checkpoint with more fine-grain analysis which allows optimizations to reduce the runtime overhead. Another potential solution to solve inconsistency problem is to increase the threshold that triggers a checkpoint for architectures like [28] and then put the system to hibernation state. If the threshold is large enough to always guarantee a successful checkpoint and then the system hibernates, there will be no rollback and thus no inconsistency problem. However, this also means that the computation is triggered only with high voltage supply. As a result, the normal system working time is largely reduced and every time the system has to wait for a long time to build up the voltage. If these architectures continue execution instead of hibernating after checkpointing, the inconsistency problem also happens.

3.2 Motivation

For illustration purpose, part of the code from dijkstra's algorithm is used. The assembly code generated by gcc-arm cross-compiler is shown in the left column of Figure 6. In this code, there are three basic blocks of code whose names are *enqueue*, *.L11* and *.L13*, respectively. The relation of these three basic blocks of code is *enqueue*→*.L11*→*.L13*. Under stable power supply, these three basic blocks of code will be executed sequentially without being interrupted. However, intermittent power supply could interrupt the execution frequently. The right column of Figure 6 shows the execution status with checkpointing to enable computation across different power cycles. In this figure, checkpointing is triggered online when low voltage below threshold is detected, which means there is a high chance of power failure and the system state needs to be saved. However, incorrect checkpointing positions may result in vital memory inconsistency errors both inside a basic block and across multiple basic blocks.

First, let's take basic block *.L13* as an example to show that an error may occur inside a basic block if there is a power failure. After executing the first instruction of *.L13*, a checkpoint is initiated and all volatile system state in volatile registers is successfully backed up to non-volatile memory. Then instruction 2 of *.L13* loads the value in non-volatile memory address Mem[r3+0] to register r2. Instruction 3 increases the value in r2 by 1. After that, instruction 4 stores the new value in r2 to the same address Mem[r3+0]. Suddenly, power fails before a checkpoint. Therefore, after power returns, the execution rolls back to the previous successful checkpoint after instruction 1 in basic block *.L3* and resumes the execution from instruction 2. The same three instructions are executed again which will first load the value in non-volatile memory address Mem[r3+0] to register r2, then increases r2 by 1, and finally store value of r2 to Mem[r3+0]. Before the power failure, the value at Mem[r3+0] is already increased by 1. However, rollback results in twice the increase. What's worse, all subsequent computations that depend on this value will no longer be correct. Consequently, inconsistency errors occur inside basic block *.L13*.

This inconsistency error may also occur across several basic blocks. In Figure 6, this error occurs across *enqueue* and *.L11* as well. In this code, *.L11* is executed sequentially

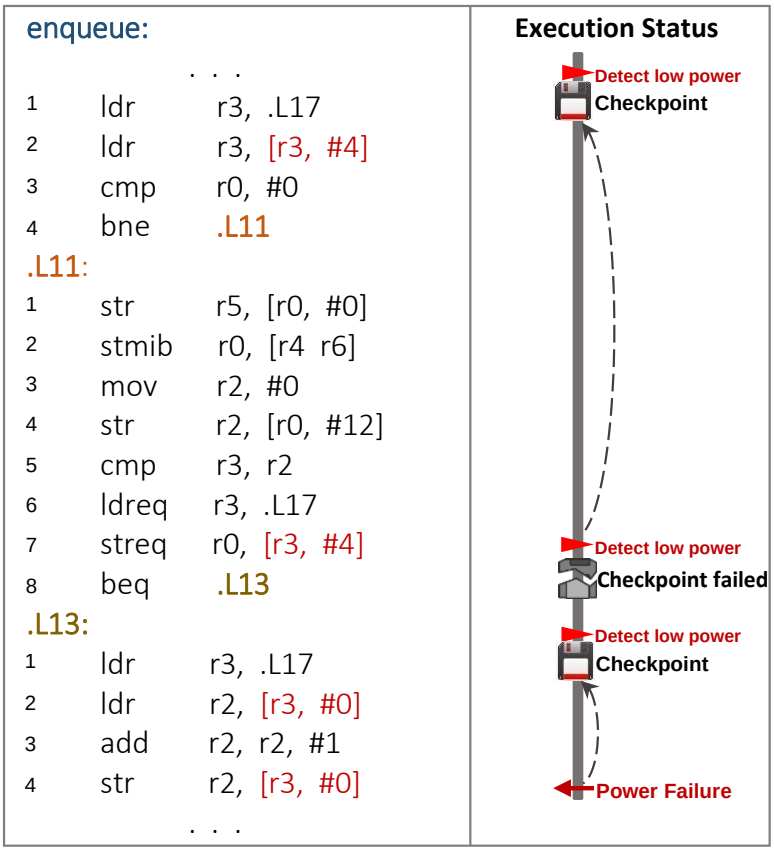

Figure 6: Example of inconsistency problem.

after *enqueue*. Instruction 2 of *enqueue* loads the value stored in Mem[r3+4] into register r3, and instruction 7 of *.L11* stores another value in register r0 into the same non-volatile memory address Mem[r3+4]. This pair of load and store operations is able to change the value stored at the same address in non-volatile memory. When the processor is executing the first instruction of basic block *enqueue*, the system detects a low power status. Therefore, after finishing this instruction, all system state is checkpointed to non-volatile memory. After completing this checkpoint, the processor continues execution and begins to execute the second instruction which loads the value in Mem[r3+4] to register r3. After all instructions in this basic block are finished, the processor begins to execute instructions in basic block *.L11*. When the processor is executing instruction 7, which stores value in register r0 to Mem[r3+4], the system detects another low power status and triggers a checkpointing operation after this instruction is finished. However, checkpointing is only partly finished because of too low power supply. Therefore, after power returns, the execution rolls back to the checkpoint after instruction 1 in basic block *enqueue* and resumes the execution from instruction 2 which loads the value in Mem[r3+4] to register r3. Unfortunately, the value in Mem[r3+4] has been modified by last execution of instruction 7 in basic block *.L11* and an inconsistency error occurs across two basic blocks *enqueue* and *.L11*.

3.3 Methodology

Inconsistency aware checkpointing technique is proposed that eliminates errors from inconsistency between volatile and non-volatile system state. I explore the underlying reason for inconsistency error. In Fig. 7(a), there is a checkpoint before the first ldr instruction. Assume the initial value in Mem[r3+0] is 17. After executing the three instructions, the content in non-volatile memory address Mem[r3+0] is updated from 17 to 18. When power failure happens after the *str* instruction, the execution has to roll back to the previous checkpoint and resume execution from the *ldr* instruction again which loads the updated value in address Mem[r3+0]. In this case, the value in address Mem[r3+0] is updated again which is 19 now. Crucially, this deviates from normal execution.

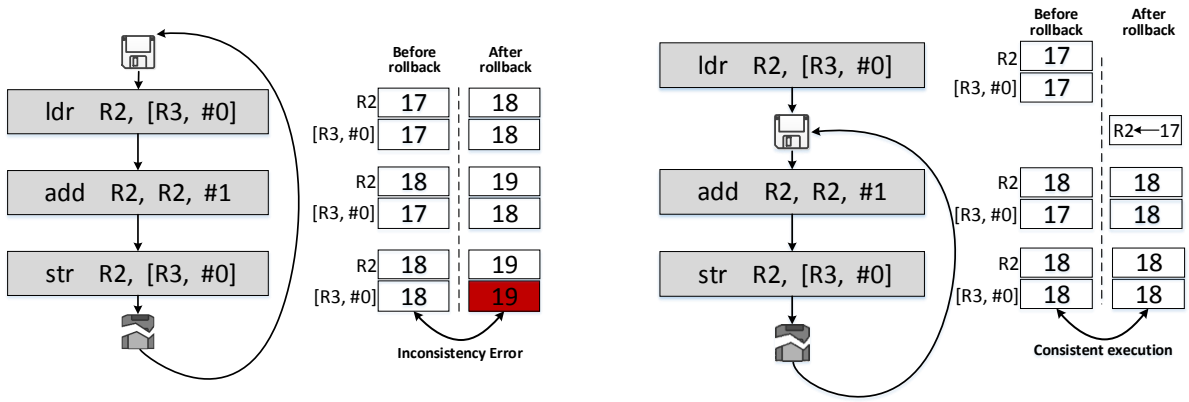

Figure 7: a) Inconsistency error. b) Inserting a checkpoint properly can eliminate inconsistency.

To eliminate this error, a checkpoint is set after the *ldr* instruction which initiates a checkpoint as shown in Fig. 7(b). Therefore, when power fails after the *str* instruction, the execution rolls back to the last checkpoint and all states of registers are restored to volatile registers. Then the value in register r2 is added by 1 and stored to Mem[r3+0]. Although the value in Mem[r3+0] is already updated before power fails, this updated value is not loaded again after rollback like in Figure 7(a), because the checkpoint is after *ldr* instruction instead of before *ldr* instruction.

Carefully rearranging the positions of the checkpoints can eliminate inconsistency errors. This type of errors are due to inconsistency between volatile registers and non-volatile memory. In order to ensure correct execution, these code sections where inconsistency may happen are searched. Next, the checkpoint positions to eliminate these potential inconsistency are found. Based on previous analysis, the "load" and "store" pairs to the same address are potential error regions when rollback recovery is needed. Such an instruction pair can reside both in a single basic block or across multiple basic blocks. If this pair of instructions exists inside a block, it is relatively easy to be found. However, if a pair of instructions exists across multiple blocks, all possible execution paths of the program should be traversed.

In this chapter, I propose an error locating algorithm to find out all potential error pairs considering all possible execution paths. These error pairs are load and store operations on the same nonvolatile memory address. After this, an algorithm is designed to insert checkpointing instructions in the code during off-line for eliminating inconsistency errors. During run time, the system will initiate checkpointing by running the instructions.

3.3.1 Potential Error Locating

In this subsection, the Potential Error Locating (PEL) algorithm is presented, which can identify all potential error-prone load-store pairs as well as corresponding paths between each pair.

The first challenge in eliminating inconsistency errors lies in how to locate error pairs. Since a program consists of many branches and loops, inconsistency errors both inside the same basic block and across different basic blocks need to be located. An algorithm is designed to locate the potential load-store error pairs and search for all the acyclic paths between the load instruction and the store instruction. This error locating algorithm will find out all the load-store pairs that could lead to inconsistency errors.

Based on previous analysis, the "load" and "store" pairs to the same address in nonvolatile memory are potential error regions when rollback recovery is needed. Such an instruction pair can reside both in a single basic block or across multiple basic blocks. If this pair of instructions exists inside a basic block, it is relatively easy to be found. However, if a pair of instructions exists across multiple basic blocks, it is a nontrivial task since we need to traverse all possible execution paths of the program. In this subsection, the Potential Error Locating (PEL) algorithm is presented to identify all potential error-prone load-store pairs as well as corresponding paths from "load" to "store" instruction.

Algorithm 1 Potential Error Locating (PEL)

Require: Basic block based *CFG*, the instruction sequence of each basic block;
Ensure: All pairs of memory modification instructions and all acyclic paths for each pair;

```
Derive basic block list S based on the depth-first search on CFG;
n ← 0;
for each basic block BB_i in S do
    for each instruction k from 1 to Len(BB_i) do
        if k loads a value from the memory address Mem then
            for each basic block BB_j in S do
                for each instruction l from 1 to Len(BB_j) do
                    if l stores a different value to the same memory position Mem then
                        Record all the acyclic paths from BB_i and BB_j to the set PATH_n;
                        Record instruction index k and l to the set INDEX_n;
                        n ← n + 1;
                    end if
                end for
            end for
        end if
    end for
end for
```

Algorithm 1 shows the procedure of the PEL algorithm. The input is a control flow graph (CFG) and the instructions in each basic block. To identify load-store pairs, all basic blocks in CFG are examined in the depth-first order to locate "load" instructions (Line 5). All pairs of "load" and"store" instructions targeting to store a different value at the same memory address are searched, and all possible acyclic paths from the "load" to "store" in a pair are stored (Line 6-14).

An example is shown in Figure 8(a) to illustrate the algorithms. There are five basic blocks in this example, containing 6, 8, 8, 5, and 2 instructions, respectively. Based on Algorithms 1, 4 "load" instructions will be found: *ldr a* at position (BB_0, ins1), *ldr c* at position (BB_0, ins4), *ldr b* at position (BB_1, ins2) and *ldr a* at position (BB_4, ins0). Figure 9 shows paths for each pair.

3.3.2 Consistency-aware checkpoints inserting

In this section, an algorithms for inserting consistency-aware checkpoints is proposed to eliminate inconsistency errors. These checkpoints are positions where the volatile system state will be saved to the non-volatile memory.

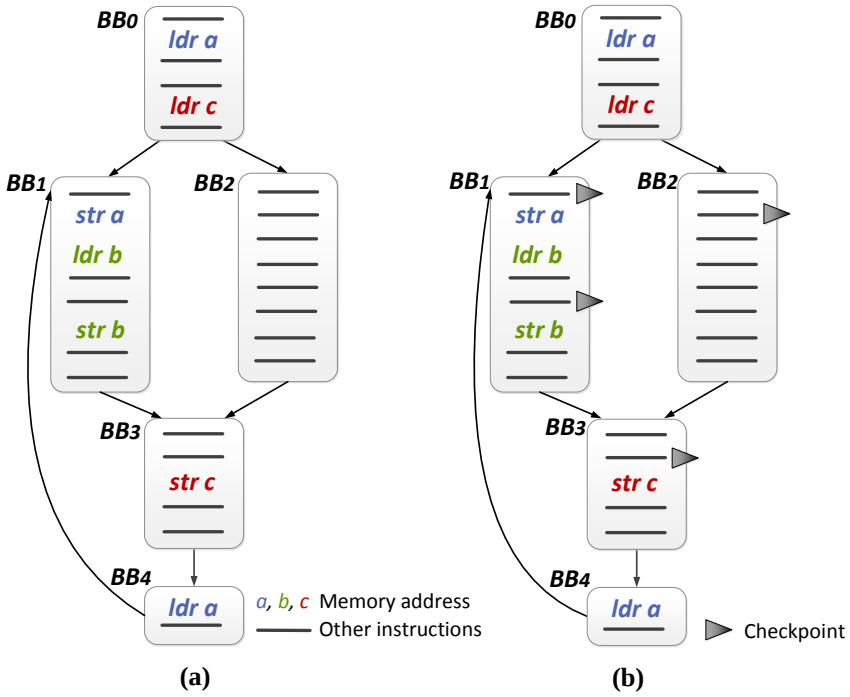

Figure 8: An example of error locating and checkpoint insertion: (a) Potential error pairs located with PEL algorithm, and (b) Checkpoints inserted with CATI algorithm

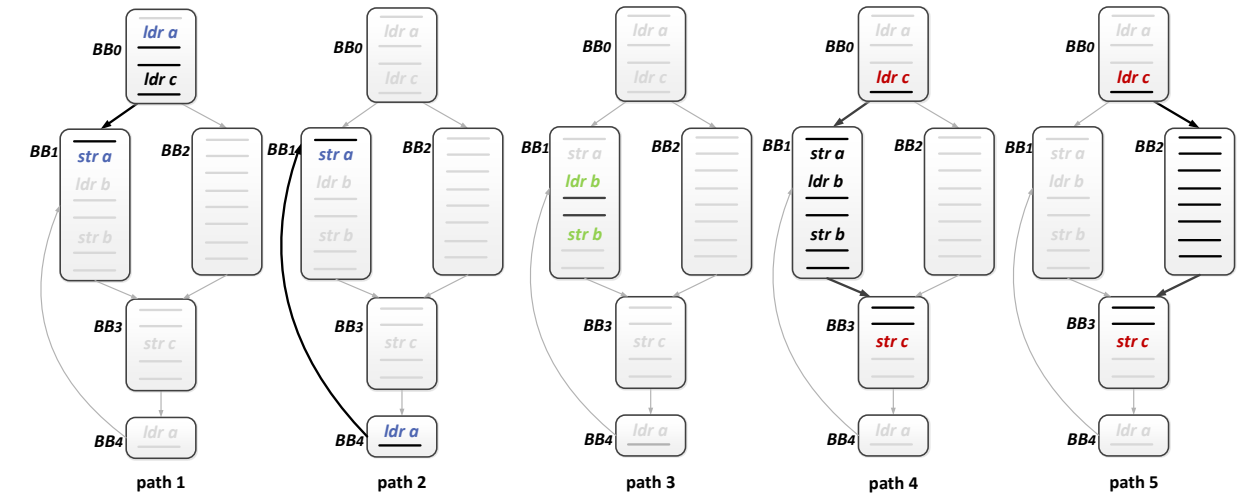

Figure 9: Paths between all load-store pairs.

After deriving all the potential error locations, we are ready to insert consistency-aware checkpoints to avoid the occurrence of this kind of errors. Besides these checkpoints to ensure correctness, extra checkpoints are set at proper positions to maintain a small rollback distance. In this way, when power fails before a checkpoint, the execution does not need a long rollback. There are two goals by inserting checkpoints: 1) all the inconsistency errors can be eliminated; 2) rollback recovery distance is within requirement. The former guarantees the correctness of the program while the latter maintains a low recovery overhead.

The distance between adjacent checkpoints to be up-bounded by a constant L_{max}. This constant is named as checkpointing distance and its unit is number of clock cycles to ensure that the processor can resume execution as fast as required without wasting too much energy and time for rollback. Therefore, a checkpoint should be inserted at every L_{max} checkpointing distance. Meanwhile, the inserted checkpoints should eliminate the errors. This problem is NP-Complete, which can be proved by reducing from the set cover problem [32]. Therefore, an efficient two-step heuristic algorithm is proposed to solve this problem.

The main challenges lie in the following aspects. First, to reduce all inconsistency errors, there should be at least one checkpoint between each load-store pair. Simply inserting a checkpoint between each pair is not feasible due to the high overhead. Therefore, effective approaches are necessary to reduce the number of checkpoints. Second, since the runtime path is not available during the offline analysis, all possible paths need to be taken into consideration to reduce the checkpointing overhead. Third, the checkpoint locations in various basic blocks may affect each other. Therefore, a global optimal solution is needed.

I propose a polynomial time heuristic approach, consistency-aware checkpoints inserting (CATI) algorithm shown in Algorithm 2. The input is a CFG and all the paths between load-store pairs. Positions for checkpoints inside a basic block are highly dependent on those in its predecessors. At the same time, they also affect those in its successors. To record this kind of relation, two arrays are employed: $E1$ to represent the distance between the first instruction and the first checkpoint in the current basic block, if any; $E2$ to represent the distance between the last checkpoint in this basic block and the last instruction of this basic block (Line 1-2).

Algorithm 2 Consistency-aware checkpoints inserting algorithm (CATI)

Require: Basic block based CFG, $INDEX$, $PATH$;
Ensure: Consistency-aware checkpoints;
$E1_i \leftarrow 0, i = 1, 2, ..., N$; //$N$ is the number of all basic blocks.
$E2_i \leftarrow 0, i = 1, 2, ..., N$;
for each BB_i, $BB_i \in$ breadth-first traversal array S **do**
Search $INDEX$ and $PATH$ to find all the load-store pairs (l_i, s_i), single load indices ll_i and single store indices ss_i in BB_i, and store them separately into sets LS, L and S;
$Energy \leftarrow 0$; $start \leftarrow 1$; $K_i \leftarrow Len(BB_i)$;
Find all the predecessors of BB_i, which are BB_{p1}, BB_{p2},..., BB_{pn};
$Energy \leftarrow Energy + max(E2_{p1}, ..., E2_{pn})$;
for $k = start$ to K_i **do**
$Energy \leftarrow Energy + en_k$; //$en_k$ is the energy consumption of instruction k.
if $(Energy > L_{max})$ **or** $(k == K_i)$ **then**
$ckp \leftarrow min(k, s_j, ss_j) - 1$, $s_j \in LS, ss_j \in S$;
Insert a checkpoint after instruction ckp;
ReduceErrors(ckp, LS, L, S);
end if
$start \leftarrow ckp + 1$;
$Energy \leftarrow 0$;
end for
Find all the descendants of BB_i, which are $BB_{d1}, ..., BB_{dn}$;
if $sum(en_{ckp_{last}}, ..., en_{K_i}) + max(E1_{d1}, ..., E1_{dn}) > L_{max}$ **then**
Insert a checkpoint after instruction K_i;
end if
if BB_i has no checkpoint **then**
$E1_i \leftarrow sum(en_1, ..., en_{K_i}) + max(E1_{d1}, ..., E1_{dn})$;
$E2_i \leftarrow sum(en_1, ..., en_{K_i}) + max(E2_{p1}, ..., E2_{pn})$;
else
$E1_i \leftarrow sum(en_1, ..., en_{ckp_{first}})$;
$E2_i \leftarrow sum(en_{ckp_{last}}, ..., en_{K_i})$;
end if
end for

Each basic block is processed in the breadth-first order in CFG. For each basic block, first the *Energy*, which represents the distance between this basic block and previous checkpoint, is initialized as the maximum value of its predecessors (Line 7). This initialization guarantees the L_{max} principle whichever path is taken at runtime. Inside the process of a basic block, *Energy* is updated by going through each instruction.

There are two cases to insert a checkpoint (Line 10). First, once *Energy* reaches L_{max}, a checkpoint will be inserted. Second, a checkpoint will be inserted to eliminate errors. Two sets, *(l_i, s_i)* and ss_i, are constructed to guide the error detection. *(l_i, s_i)* includes all the

Algorithm 3 $ReduceErrors(ckp, LS, L, S)$

1: **for** each load-store pair (l_i, s_i), $(l_i, s_i) \in LS$ **do**
2: **if** $ckp \geq l_i$ **and** $ckp < s_i$ **then**
3: Remove (l_i, s_i) from LS and renew LS; Remove (l_i, s_i) from $INDEX$;
4: **end if**
5: **end for**
6: **for** each single load ll_i, $ll_i \in L$ **do**
7: **if** $ckp \geq ll_i$ **then**
8: Remove ll_i from L and renew L; Remove ll_i from $INDEX$ and its corresponding paths from $PATH$;
9: **end if**
10: **end for**
11: **for** each single store ss_i, $ss_i \in S$ **do**
12: **if** $ckp < ss_i$ **then**
13: Remove ss_i from S and renew S; Remove ss_i from $INDEX$ and its corresponding paths from $PATH$;
14: **end if**
15: **end for**

load-store pairs inside BB_i and ss_i records all the single stores inside BB_i. If both sets are not empty, one checkpoint will be inserted before the first store instruction (Line 11-12).

Take Figure 8(a) as an example. Assume *ldr*/*str* instructions consume 2-unit amount of energy while others consume 1 unit. Assume $L_{max} = 10$. The five basic blocks will be visited in the order: $BB0, BB1, BB2, BB3, BB4$. Starting with BB_0, since there is no load-store pair, it is processed with no checkpoint inside and $E2_0$=8. Then for BB_1, (l_1, s_1)={(*ldr b, str b*)} and ss_1={*str a*}. *Energy* is initialized to be max{$E2_0$, $E2_4$}=8. Although *Energy* is 9 before *str a* which is less than L_{max}, a checkpoint is inserted here to avoid consistency error. After this, Algorithm 3 is called to update the potential error location and all the paths passing the error location after each checkpoint insertion (Line 13). Note that the inserted checkpoint could be able to avoid errors for one or more paths between load-store pair(s). In this example, this checkpoint can eliminate the error for both path 1 and path 2 shown in Figure 9. Then the process of this basic block starts from the instruction after the checkpoint (Line 15-16). In this example, the processing will start from *str a* in BB_1, with current ss_1 being empty and (l_1, s_1)={(*ldr b, str b*)}. The next checkpoint will be inserted before *str b*, followed by the update of $E1_1$=1 and $E2_1$=4. As a result, path 1 and 2 in Figure 9 are removed from the error set.

Different from BB_1, *(l_2, s_2)* and ss_2 are empty. After going through the first two instructions, *Energy* is increased to L_{max} because $E2_0$+2=10. Thus, a checkpoint is inserted after the second instruction in BB_2. The left instructions cost 6 energy units. Therefore, $E1_2$=2 and $E2_2$=6. After processing BB_3, one checkpoint is inserted before *str c*. Thus, $E1_3$=2 and $E2_3$=2. This checkpoint eliminates the error for both path 4 and path 5 shown in Figure 9. There will be no checkpoint in BB_4 and $E1_4$=5 and $E2_4$=5. At the end of processing BB_4, the loop to BB_1 is examined to guarantee the distance between two checkpoints is less than L_{max} (Line 18-21). The final set of checkpoints is shown in 8(b). There are totally 4 checkpoints. For better understanding of the details of CATI algorithm, a block diagram is employed to illustrate the major idea of processing different basic blocks in input CFG as shown in Figure 10.

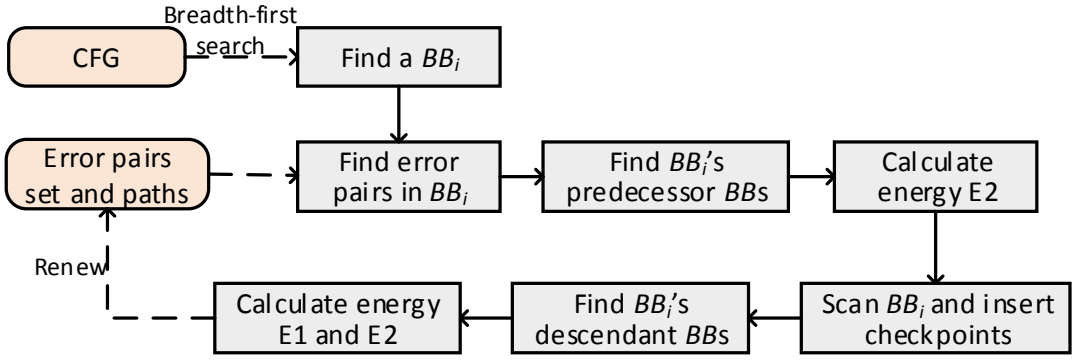

Figure 10: Block diagram for Algorithm 2.

After Algorithm 2, all loops in the CFG will be checked to see whether it contains a checkpoint. If not, a checkpoint is inserted at the end of the loop. This insertion only happens when the energy consumption of a loop iteration is less than L_{max}. In a nutshell, the proposed scheme achieves an error-free checkpoint set by assigning a checkpoint either before *store* or at the distance of L_{max} from the previous checkpoint.

3.4 Experiment

3.4.1 Setup

The experiments are conducted with a custom simulator. The input of the simulator is 1million instruction traces generated with gem5 simulator [10] . An example of the instruction traces is shown in Figure 11. The collected instruction has the function names, the detailed instruction, and the memory addresses if the instruction is a ldr or a str instruction. Assume that ldr and str takes two cycles and other instructions take one cycle. Based on this trace, the CFG of basic blocks can be generated. With the memory access traces, the ldr and str instruction pairs are found that access the same memory addresses given different checkpointing frequencies including 50, 100, 500, 1000, 2000 cycles. After this, the corresponding instructions are labeled.

```
@dijkstra+436    mov   r2, r12
@dijkstra+440    ldr   r12, [r12, #12]   ReadReq 82b6c
@dijkstra+444    cmps  r12, #0
@dijkstra+448    bne
@dijkstra+452    str   r0, [r2, #12]     WriteReq 82b6c
@dijkstra+456    ldr   r2, [r5, #0]      ReadReq 7276c
@dijkstra+460    ldr   r8, [r10, #0]     ReadReq 7ddbc
@dijkstra+464    add   r2, r2, #1
@dijkstra+468    str   r2, [r5, #0]      WriteReq 7276c
```

Figure 11: Instruction traces example.

The experiments are conducted on a set of benchmarks including basicmath, dijkstra, and fft, etc. These benchmarks are from the Mibench [18] benchmark suite. The proposed algorithms are run on the implemented simulator first. The simulator analyzes potential error regions and outputs assembly code labeled with checkpoints. Therefore, they can be

easily incorporated into any existing compiler. Besides, the instruction trace is generated with gem5 [10] for each benchmark and used these traces to evaluate the on-line running techniques by applying real power traces.

3.4.2 Experimental Results and Analysis

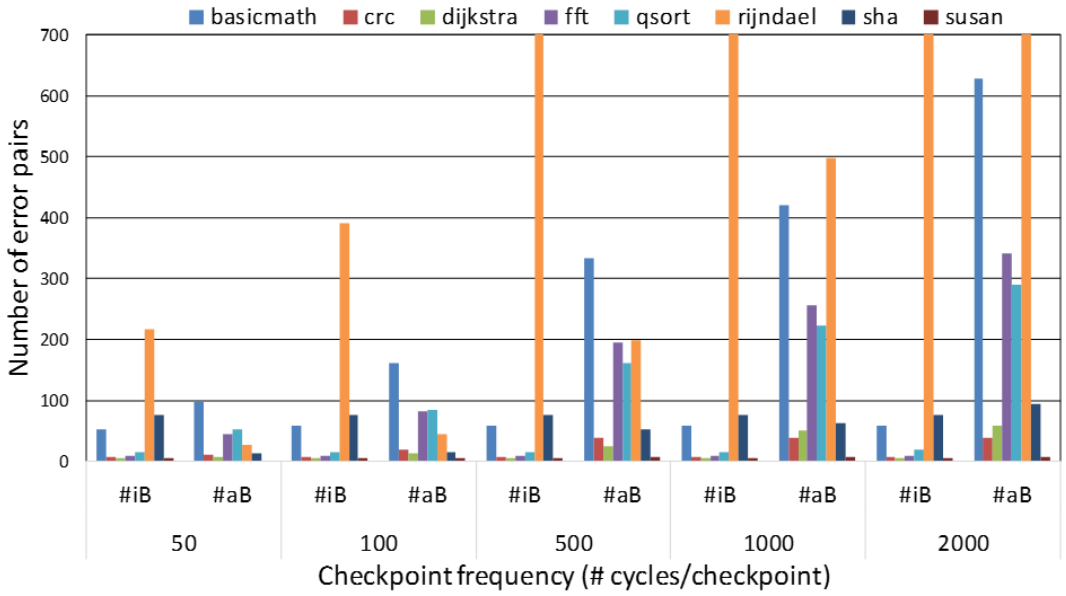

Figure 12: Number of potential error pairs given different checkpointing frequencies.

3.4.2.1 Error Locating In this section, the programs are analyzed and the proposed algorithms are applied to locate the potential error regions. The results of collected errors are shown in Figure 12. This figure shows the number of potential errors when the checkpointing energy distance is set to be 50, 100, 500, 1000, and 2000 cycles, separately. From this figure, we can see that as the checkpointing distance increases, the number of error pairs increases. This is because as checkpointing distance increases, those error pairs that have longer paths will be located. The proposed PEL algorithm can find all the potential error pairs and eliminate them. From the table, we can see that the number of errors varies in

different benchmarks. However, even a single error can cause the program to have a fatal result. Therefore, the proposed error aware algorithm is of significant importance. The results of error locating will vary depending on different compiling methods.

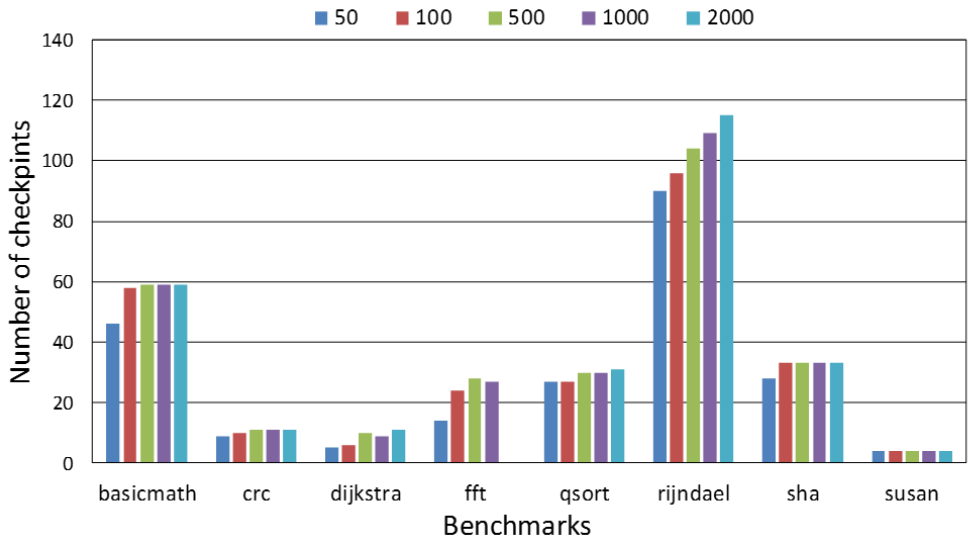

Figure 13: Number of inserted checkpoints given different checkpointing frequencies.

3.4.2.2 Inserting Checkpoints Selecting checkpoint positions is important for the whole embedded system, since it can help eliminate consistency errors. Meanwhile, unnecessary checkpoint should be avoided to reduce analyzing effort and checkpointing overhead. In the experiments, a checkpoint is inserted at a fixed energy distance and also inside each error pair. Figure 13 shows the comparison of checkpoints for the proposed error aware algorithm when the checkpointing distance is 50, 100, 500, 1000, and 2000 cycles. From this figure, we can see that as the checkpointing distance increases, the number of checkpoints also increases. The energy distance is set to avoid a long rollback distance. Therefore, it should be determined considering both the energy traces and the rollback overhead. In conclusion,

the proposed CATI algorithm can dynamically modify the number of checkpoints, and thus the checkpointing overhead, by setting different energy distances.

3.5 Summary

In this chapter, an inconsistency-aware checkpointing scheme is proposed for energy harvesting powered non-volatile processors. To guarantee the correct execution and reduce checkpoint overhead, an approach to determine the most appropriate checkpointing positions for each basic block by considering all execution paths that have an influence on the correctness and the overhead. The evaluation results show that all the inconsistency errors can be located, and the overhead of the extra checkpoints for eliminating the errors is acceptable given the importance of execution correctness.

4.0 Checkpointing-aware Hybrid Cache for Intermittently Powered IoT Devices

This chapter presents a hybrid cache architecture for intermittently powered IoT devices [88, 91]. The remainder of this chapter is organized as follows. Section 4.1 presents the background of this project. Section 4.2 describes the motivation of this project. Section 4.3 and Section 4.4 presents the basic placement and migration policies and the checkpointing aware cache policies under the new restrictions imposed by intermittent computing. Section 4.5 presents the new cache architecture checkpointing policy. Experimental evaluation is provided in Section 4.6. Finally, Section 4.7 concludes this project.

4.1 Background

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) [55, 77, 57] based non-volatile processors (NVPs) [53, 77, 101] 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. [38] 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 [47]. However, since NVMs such as STT-RAM [29] and PCM [56, 58] 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 performance [80, 74, 83, 97]. 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, this chapter aims 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 proposed cache is specifically tailored for intermittently powered embedded systems.

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.

Currently there are two designs for cache in NVP. Li et al. [38] 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 [47]. However, since NVMs typically have high write latency, pure NVM based cache will become the performance bottleneck. To solve this problem, a checkpointing aware hybrid cache will be proposed which consists of both SRAM and STT-RAM. Hybrid cache architecture is not new. However, most existing designs are aiming for high performance and low power consumption. [80, 83] 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 bybrid 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 project aims for high performance, reliable checkpointing, instant resumption, and low energy consumption.

4.2 Motivation and Overview

In this section, the motivation of this project will be presented followed with an overview of the self-checkpointing aware cache architecture which will have both high performance and checkpointing efficiency.

Table 1: Comparison of different cache architectures

Properties	STT-RAM cache	NVSRAM cache [44]	Proposed hybrid cache
Performance	Low	High	High
Energy	Medium	Low	Medium
Size	Small	Large	Small
Resistant to power loss	Yes	Yes	Yes
Resumption speed	Instant	Fast	Instant
Checkpointing energy	—	High	Low

4.2.1 Motivation

The existing cache design for intermittently powered embedded system is either based on pure non-volatile memory or NVSRAM. The first design suffers from long write latency and energy overhead, while the latter suffers from high inrush current and large memory size.

Energy harvesting powered processor built with non-volatile cache is 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 one of 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 [29]. Two processors with pure SRAM cache and pure STT-RAM cache are configured in gem5 [10] 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 14 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, NVSRAM based cache architecture is a better solution in terms of performance. However, during power

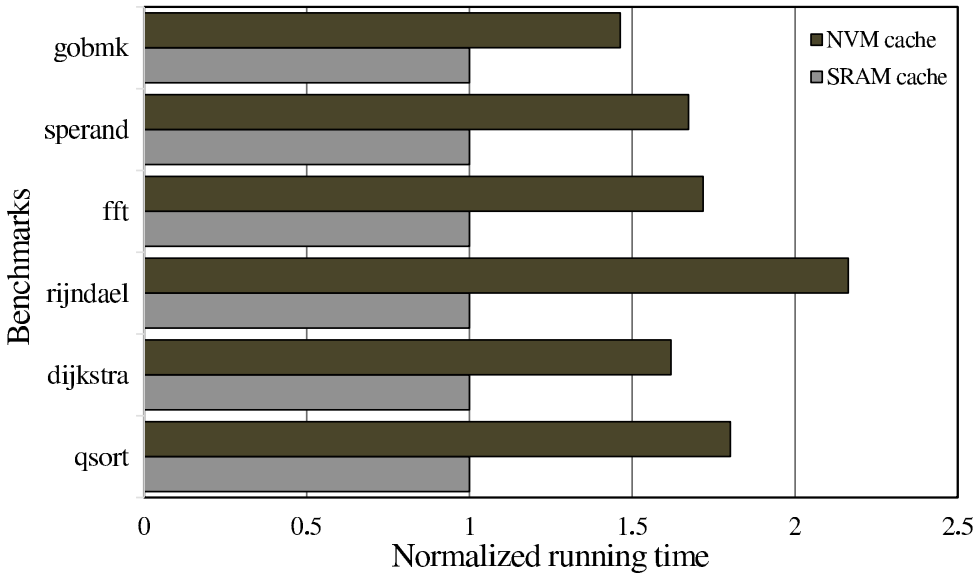

Figure 14: Performance of SRAM cache and NVM cache

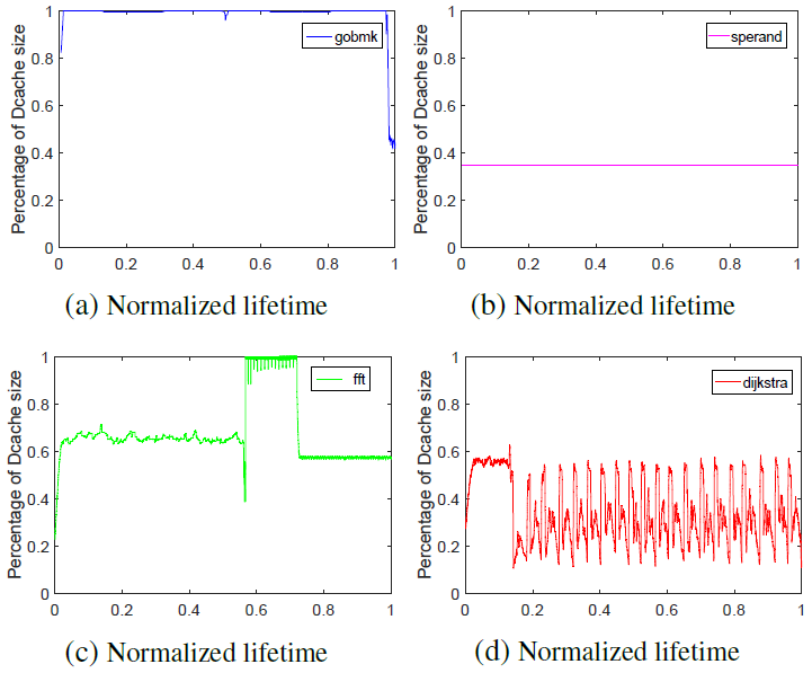

Figure 15: Percentage of dirty blocks during lifetime

failure, all dirty blocks in SRAM need to be checkpointed to the NVM cells since they are modified which leads to high inrush current and checkpointing overhead. Figure 15 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*. 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 chapter, a one-level hybrid cache is designed which is specifically tailored for low intermittently-powered embedded system with small size and high energy efficiency. Table 1 compares the hybrid cache with the existed cache designs. This cache architecture is inspired by [81] which proposes a STT-RAM based hybrid L2 cache, taking advantage of the fast access speed of SRAM for write operations and high-density of STT-RAM for the read operations. This one-level cache further enables checkpointing capability with non-volatile STT-RAM, so that STT-RAM is not only used for normal cache access but also for checkpointing important cache blocks upon power outages. From Figure 15, if NVM blocks are reserved 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.

4.2.2 Overview

4.2.2.1 Self-checkpointing Cache 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 a longer time to access a second level cache than the first level. Therefore, in this project, a hybrid one level cache architecture is designed. The proposed hybrid cache architecture is shown in Figure 16. 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

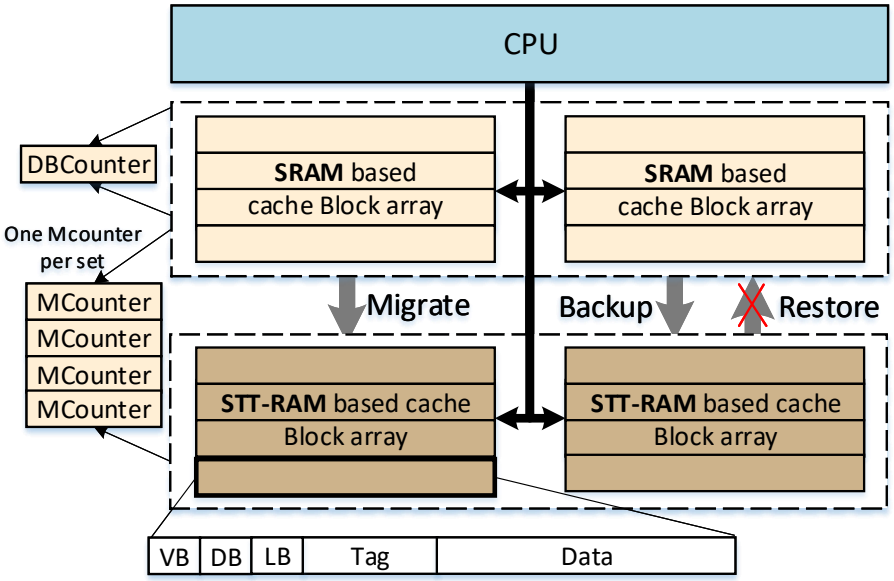

Figure 16: Hybrid cache architecture

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 most important SRAM cache blocks will be backed up to STT-RAM replacing the less important clean cache blocks in STT-RAM while the important STT-RAM blocks still stay there. When power returns, no restoration process is needed to restart execution.

4.2.2.2 Challenges There are many challenges in designing the self-checkpointing energy-efficient cache. Only checkpoiting the most important cache blocks means a part of cache blocks in each cache set will be lost upon power outages. If those lost cache blocks will not be accessed in the near future after power recovers, the miss rate will not be influenced by storing all the left cache blocks in STT-RAM; Otherwise the miss rate will increase,

thus lowering the overall performance. On the other hand, the ratio of STT-RAM influences the miss rate by determining how many cache blocks in a set can be checkpointed: larger ratio of STT-RAM means that more important cache blocks can be stored for future access. However, if the ratio of STT-RAM is too large, there might not be enough space for placing write-intensive cache blocks in SRAM. Therefore, we need intelligent prediction method to identify the blocks that will not be accessed in the near future. Besides, we also should be careful about choosing the ratio between SRAM based blocks and STT-RAM based cache blocks. Besides, for dirty cache blocks in SRAM, even they will not be accessed in the future of high possibility, they still need to be checkpointed into the STT-RAM. Therefore, another challenge lies in how to guarantee there is always enough space for checkpointing dirty volatile cache blocks.

4.3 Basic Placement and Migration policies for one-level hybrid Cache

In this section, an access pattern based predictor will be proposed for directing the cache policies. The access pattern predictor is able to predict dead blocks and write-intensive cache blocks. After this, cache policies will be proposed including the placement, migration, and write policies for the one-level cache. The goal of these polices is to reduce the long write latency and energy overhead of non-volatile memory by placing the write-intensive cache blocks in SRAM.

4.3.1 Access Pattern Predictor

Three patterns of memory access are the most important for designing high-performance and energy-efficient hybrid L1 cache for the energy harvesting system, which are: write intensive, dead hit, dead-on-write fill, and dead-on-read fill. Write intensive blocks are the cache blocks that will be written may times before their eviction. Predicting the write intensive pattern helps designing policies for placing the write intensive cache blocks in SRAM to reduce the large latency and energy overhead of write in STT-RAM. Dead hit blocks are

the cache blocks that will not be accessed again after being moved out of the MRU postion. Dead-on-write fill blocks are the cache blocks that are filled into the cache for write miss which will not be accessed again before they are evicted. Dead-on-read fill blocks are the cache blocks that are filled into the cache for read miss which will not be accessed again before they are evicted.

The baseline is the access pattern predictor for the last level STT-RAM-based hybrid cache [81]. This predictor targets on predicting write burst and dead blocks in LLC considering all types of LLC accesses. It is based on PC and correlates different access patterns in the LLC with instruction addresses. The intuition is that if a given memory access instruction PC leads to an access pattern, then the same instruction PC will lead to the similar access pattern. Similar to its baseline [42], it samples only a small group of sets from the LLC and only updates the pattern sampler when there is an access request to these sets. To predict both write burst and dead block, it separates read and write access with a read PC field and a write PC field as well as the corresponding LRU bits for the read and write, respectively. For making prediction, the pattern sampler updates the prediction table indexed by the signatures generated with PC. Each entry in the prediction table has two counters which generates the prediction results if they reach the thresholds.

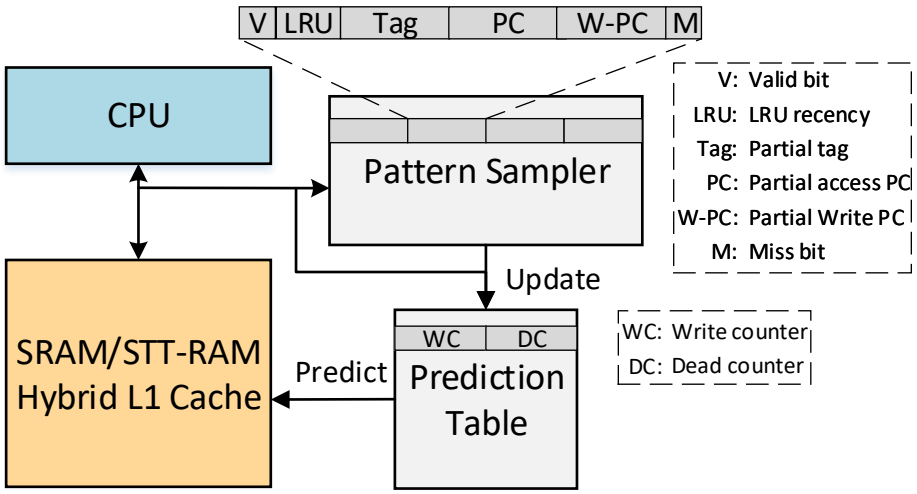

Figure 17: System structure of predictor

To predict the access pattern for the one-level cache while maintaining a small storage and energy overhead, the baseline predictor is tailored specifically for the one-level cache. The baseline learns the dead behavior only with the read PC. However, a write PC can also leads to the dead behaviour of a block. Therefore, different from baseline, our pattern sampler learns the dead behavior with both read PC and write PC. A miss indication field is added at the end for identifying miss or hit for the last access. Besides, the update policy for the prediction table is also different. The baseline is dedicated for the last level cache and therefore updates the dead counter upon almost every access to the cache. Unlike the baseline, we update the dead counter based on bursts of accesses to a cache block, since predicting dead block based on bursts of accesses to a cache block is proved to have a higher accuracy for its hiding the irregularity of individual references for L1 cache [42].

The pattern access predictor for one-level cache is shown in Figure 17. The predictor is composed of a pattern sampler and a combined prediction table making predictions for both write intensive blocks and dead blocks. Each entry in the general access sampler corresponds to a sampled set which has six fields for each block in the set, including valid bit (V), LRU recency (LRU), lower 16 bits of the tag (Tag), lower 16 bits of the most recent PC that accesses the block (PC), lower 16 bits of the most recent PC that writes to the block (W-PC), and miss bit (M). The PC field is for learning dead behavior while the W-PC field is for learning the write intensive behavior. The M field has one bit for indicating a miss (0) or hit (1) of the last access. Each entry of the prediction table has two 2-bit saturating counters which are the write counter for predicting write intensive behavior and the dead block counter for predicting dead behavior.

For each memory access request with a PC to the sampled sets, the pattern sampler simulates the block placement for the corresponding set by modifying the six fields for the requested cache block and LRU fields of the other cache blocks in the same set. After this, the related entry indexed by the hashed PC is updated according to the results of the pattern sampler. Meanwhile, the counter values of the prediction table predict the behavior of the requested cache block which is further used to direct the cache placement policy. The detailed policies for updating the predictor and predicting the cache block behavior will be described in the following sections.

4.3.1.1 Updating the pattern access predictor

The write counter in the prediction table is updated upon every write access or eviction of a cache block in the sampled sets. The read access does not influence the write behavior and therefore will not update the write counter value. The dead counter in the prediction table is updated upon every cache burst or eviction of a cache block. The set sampling process of the pattern sampler and the prediction table update are illustrated in Figure 18.

On each write hit request, the pattern simulator updates the write prediction towards "write intensive" by increasing the counter value indexed with the related write PC. On each write miss request, the write counter value will not be updated. This is because there is no relating write PC, since there is also no matching block in the pattern sampler. When a block is evicted, the simulator updates the write prediction towards "not write intensive" by decreasing the counter value indexed with the related write PC.

The dead counter update is based on cache burst history of each block. The continuous accesses of a cache block starting from its being moved to the MRU position and ending after its being moved out of the MRU position are called a cache burst. On each read hit or write hit request from CPU to a sampled set, the LRU recency of the requested cache block will be checked. If this block is not at the LRU position, the dead counter in the prediction table indexed with the previous related PC of this block will be decreased towards "Live". When a block is evicted from the pattern simulator, the dead counter in the prediction table indexed with the previous related PC of this block will be increased towards "Dead".

One each read miss or write miss request, if the M field for this block indicates a miss, then the dead counter in the prediction table indexed with the previous related PC of this block will be decreased. If the last access is also a miss, this means this block is not filled to the cache because of a wrong dead prediction. Note that although the the dead read and dead write requests will not place cache blocks in the corresponding set, it will still be recorded in the pattern sampler. This is necessary for updating the prediction table if the dead prediction is later proved to be wrong.

4.3.1.2 Making Prediction

The prediction table is capable of predicting dead access and write intensive access by thresholding the counter values in the table entry indexed by

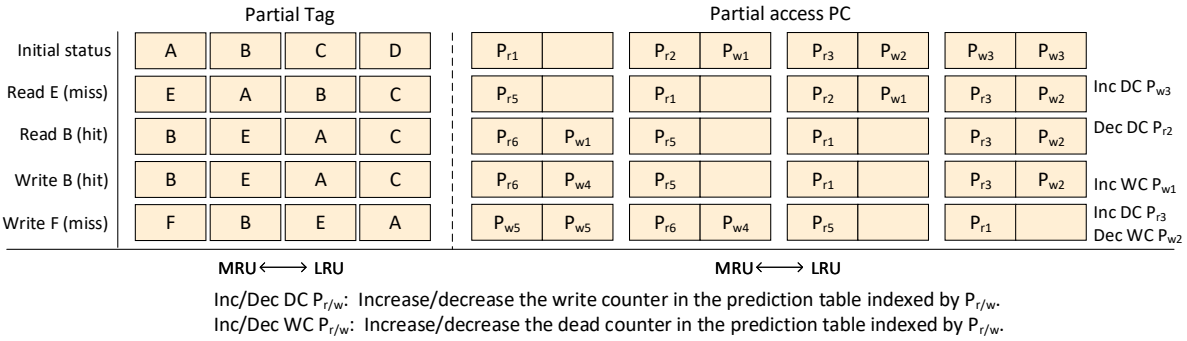

Figure 18: An example of sampling the cache set with pattern sampler and updating the prediction table.

the hashed instruction PC. Upon a read request from the CPU, the dead counter in the prediction table will be accessed to predict whether it is a dead block. Upon a write request from the CPU, both the dead counter and the write-intensive counter in the prediction table will be accessed to predict write-intensive and dead behavior. Note that a predicted dead block becomes dead after being moved out of the MRU position. On write/read hit, the requested cache block will be marked dead by setting its live bit (LB) to 0. This block later becomes a predicted dead block after being moved out of the MRU position.

4.3.2 Cache Placement and Migration Policy

The cache placement and migration policy is guided by the prediction results. The dead prediction is used to guide the bypassing of dead-on-write fill and dead-on-read fill cache blocks and victim block placement. The write intensive prediction is used to guide the migration of write intensive blocks from STT-RAM to SRAM to reduce the write overhead of cache blocks in STT-RAM.

On a read hit request, the requested data will be sent to the CPU directly. On a read miss, the entry of the prediction table indexed with the hashed requesting PC will be checked to predict dead behavior. If it is predicted live, the missing cache block will be filled to cache

and then the data is served; if it is predicted live, the requested data will be returned to the CPU directly without filling in the missing cache block. On a write hit request, if the requested data is in SRAM, the data will be written to its original memory block. If the requested cache block is in STT-RAM, the write intensive behavior will be predicted by checking the write counter entry of the prediction table indexed with the hashed requesting PC. If the requested block is predicted write intensive, then this block will be migrated to SRAM; otherwise, the new data will be written to the original cache block.

The best destination for migration is $Vvictim$, which is the LRU dead block or a live LRU block in SRAM if there is no dead block. Meanwhile, the victim block in the SRAM should be checked for its live bit. If *Vvictim* is predicted dead, this block will be evicted; If *Vvictim* is predicted live, it will be migrated 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 evicting it, it is migrated to the STT-RAM, since this takes less time and energy. The new destination for *Vvictim* will be the original position of the write intensive block. On a write miss request, the dead behavior will be predicted by checking the dead counter entry of the prediction table indexed with the hashed requesting PC. If the requested block is predicted dead, this data will be written to the main memory without filling the missing block into cache; otherwise, the new data is written to the cache block after it is filled into cache.

In conclusion, the basic dead block prediction and write intensive prediction based cache replacement and migration policies maximize the performance of the hybrid first level cache by placing the most write-intensive cache blocks in the SRAM, migrating the evicted live SRAM block into the STT-RAM, and bypassing the dead-read-fill and dead-write-fill cache blocks. However, the cache replacement and migration policies do not work under intermittently powered scenarios. The next section will propose the problems and checkpointing aware policies for intermittently powered scenarios.

4.4 Checkpointing Aware Cache Policies

The checkpointing aware cache will fully take advantage of the fast access speed and the large density of STT-RAM, while it is checkpointing aware. The STT-RAM portion will not only be used for normal cache access but also for backing up the necessary cache blocks in SRAM upon a power failure. Therefore, checkpointing aware cache policies will be proposed to adapt to this cache architecture.

4.4.1 New restrictions Imposed by Intermittent Computing

For self-powered embedded systems, when there is a power failure, all the dirty cache blocks in the volatile SRAM should be checkpointed to the non-volatile STT-RAM. Besides these dirty blocks, it's better to keep as many as live clean blocks in the STT-RAM for accelerating future access. Moreover, there are a large percentage of dead blocks in the cache. Among these dead blocks, the blocks that are dirty will be written back when they are evicted. However, from the research, those dirty dead blocks stay in the cache for a long time after they become dead. Therefore, if these blocks are written back early when the harvested is sufficient, they will not need to be checkpointed to the STT-RAM upon power failure. For these live dirty blocks, we need to guarantee that there is always enough space for checkpointing them. Otherwise, the most recent data modification will be lost resulting computation errors. Furthermore, we should guarantee that the limited energy in the capacitor is capable of checkpointing all important cache blocks according to our analysis.

The **basic replacement and migration policy** assisted by the proposed pattern access predictor will be further restricted by the dirty block control policies. Besides dirty block control policies, proactive write policy is proposed to reduce checkpointing overhead. To better explain the details of the policy, the notations for five types of cache blocks are defined in Table 2.

Table 2: Notations of cache blocks

Notation	Description
$victim$	LRU dead block or live block if there is no dead block.
$Vvictim$	$Victim$ in SRAM.
$DVvictim$	Dirty $Victim$ in SRAM.
$DDvictim$	Dirty dead block in SRAM or STT-RAM.
$DLvictim$	Dirty live block in SRAM or STT-RAM.
$CNVictim$	Clean $Victim$ in STT-RAM.

4.4.2 Dirty Block Control

Suppose the 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:

$$N_v + N_{nv} = N \tag{4.1}$$

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:

$$DB_v^i \leq CB_{nv}^i \tag{4.2}$$

where 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.

The replacement policy will be directed by the inequality (4.2). To satisfy this condition, a counter $MCounter$ is implemented in the cache structure for each cache set. 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:

$$size = \lceil \log_2 N_{nv} \rceil \tag{4.3}$$

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 volatile blocks in STT-RAM. Therefore, at this time, a *DDvictim* or *DLvictim* if there is no *DDvictim* will be proactively written back if the number of dirty blocks increases again in the same set.

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

$$\sum_{i=1}^{M} DB_v^i \leq T \tag{4.4}$$

The replacement policy will be directed by the inequality (4.4). To satisfy this condition, one counter is implemented in the cache structure: $DBCounter$ for SRAM portion. The $DBCounter$ records the current total number of dirty cache blocks in the SRAM cache part. When the $DBCounter$ exceeds the preset threshold, a $DVvictim$ will be proactively written back if the number of dirty blocks increases again in the same set. 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.

4.4.3 Proactive Early Write Back

Upon a power failure, all dirty cache blocks should be saved to the STT-RAM including the dirty cache blocks that are checkpointed to the STT-RAM and the dirty blocks that are placed in STT-RAM before power failure. Among those dirty blocks, many of them are already dead a long time ago and do not need to be checkpointed upon power failure. However, because they cannot be guaranteed to be dead for the one hundred percent with the dead prediction technique, all of them should be checkpointed to avoid computation errors. Nevertheless, the dead dirty blocks will be evicted sooner or later after the power comes back on again. Checkpointing them not only wastes time and energy but also takes away the checkpointing opportunity of many live cache blocks in SRAM which will be accessed soon in the near future. As a result, many live cache blocks have to be loaded from the main memory again which increases the miss rate.

Proactive write method is proposed to write back those dead dirty blocks back to main memory earlier before they are finally evicted. In this way, these dead dirty blocks will become clean and will not need to be checkpointed upon power failure. Proactive writes are realized incrementally instead of all at once each time. The proactive write is conducted each time after the LRU recency of a set is updated. This is because that a cache block may become dead only after being moved out of the MRU position. Therefore, updating the LRU recency means that a new cache block may becomes dead. If a $DDvictim$ or a $DLvictim$ if there is no $DDvictim$ is found in the same accessed set, this cache block will be marked clean and be written back to the main memory. In this way, if the dirty cache block is predicted to be dead correctly, it will not affect the miss rate, since it will be evicted and written back to main memory anyway; if the prediction is wrong, the penalty happens only if this cache block receives a write request in the future. In this case, the penalty is writing back this cache block again to update the main memory in the future. If this cache block will only be read again, there is no penalty. In the proposed hybrid, a queue buffer is employed between the cache and main memory to overlap the latency for writing back following the state-of-the-art cache architecture.

4.4.4 Reliable and Energy-efficient Checkpointing Aware Cache Policies

The Checkpointing Aware Cache Policy Implementation is described in Algorithm 4. The basic cache replacement and migration policy are assisted with the dirty cache block control and early write back functions described at the bottom of this algorithm. The actions upon an access request of four scenarios including read hit, read miss, write hit, and write miss are described at the top part of this algorithm. $REDUCE\_CTR$ is the control function of actions when the number of dirty block is increased. $INCREASE\_CTR$ is the control function of actions when the number of dirty block is decreased. $EARLY\_WB$ is the function for writing back the dirty dead blocks earlier before they are evicted. $REDUCE\_CTR$ and $INCREASE\_CTR$ ensure that there is always enough space for checkpointing in STT-RAM and the storage energy is capable of checkpointing all important cache blocks. $EARLY\_WB$ further reduces the checkpointing overhead without degrading the overall performance. These three functions together makes the hybrid cache checkpointing friendly.

The $REDUCE\_CTR$ function is called to modify the $MCounter$ and $DBCounter$ when the number of dirty blocks is reduced by one. For example, loading a missing cache block (clean) by evicting a dirty victim cache block will reduce the number of dirty blocks by one (line 9-11). Besides, migrating a dirty write intensive cache block in STT-RAM to the position of a dirty $Vvictim$ will also reduce the number of dirty blocks (line19-24). The $INCREASE\_CTR$ function is called to modify the $MCounter$ and $DBCounter$ and control the number of dirty blocks when the number of dirty blocks is increased by one. The number of dirty blocks is increased in four scenarios: 1) there is a write hit in a clean cache block (line 16-17 and 32-34); 2) a clean write intensive cache block is migrated to replace a clean $Vvictim$ upon a write hit (line 25-26); 3) the live $Vvictim$ is not evicted and migrated to the original clean write hit cache block in STT-RAM (line 29-30); 4) loading a new cache block upon write miss while the evicted cache block is clean. The $EARLY\_WB$ function is called only when an MRU cache block moves out of the MRU position. This is because that a block becomes dead only after it's moved out of the MRU position. Therefore, $EARLY\_WB$ is called in Line 4 and line 36 to write a dirty dead block back to main memory when there is a read hit or write hit if this block just moves to the MRU position.

Algorithm 4 Checkpointing Aware Cache Policy Implementation

Require: An access request to set s, $s.MCounter$, and $DBCounter$.
Ensure: Serve the request while meeting the restricts.

```
1: if read hit then
2:     Return requested data;
3:     if hit block was not MRU then
4:         EARLY_WB(s);
5:     end if
6: end if
7: if read miss then
8:     if it is a dead fill then
9:         Return data without loading the cache block;
10:    else
11:        Replace vctim and return data;
12:        if the evicted cache block Vctim is dirty then
13:            REDUCE_CTR(s, s.MCounter, DBCounter);
14:        end if
15:        EARLY_WB(s);
16:    end if
17: end if
18: if write hit then
19:    if hit in SRAM then
20:        Write to original cache block;
21:        if original cache block is clean then
22:            INCREASE_CTR(s, s.MCounter, DBCounter);
23:        end if
24:    else if hit in STTRAM then
25:        if write intensive then
26:            Migrate to Vvictim in SRAM;
27:            if Vvictim is dead then
28:                if Vvictim is dirty and hit block is dirty then
29:                    REDUCE_CTR(s, s.MCounter, DBCounter);
30:                else if Vvictim is clean and hit block is clean then
31:                    INCREASE_CTR(s, s.MCounter, DBCounter);
32:                end if
33:                Evict Vvictim;
34:            else
35:                Migrate Vvictim to STT-RAM;
36:                if hit cache block is clean then
37:                    INCREASE_CTR(s, s.MCounter, DBCounter);
38:                end if
39:            end if
40:        else
41:            Write to original cache block;
42:            if original cache block is clean then
43:                INCREASE_CTR(s, s.MCounter, DBCounter);
44:            end if
45:        end if
46:    end if
```

```
47:     if hit block was not MRU then
48:         EARLY_WB(s);
49:     end if
50: end if
51: if write miss then
52:     if it is a dead fill then
53:         Write data to memory without loading the cache block;
54:     else
55:         Replace a Victim and write data;
56:         if the evicted cache block Victim is clean then
57:             INCREASE_CTR(s, s.MCounter, DBCounter);
58:         end if
59:         EARLY_WB(s);
60:     end if
61: end if
62: function REDUCE_CTR(s, s.MCounter, DBCounter)
63:     s.MCounter ← s.MCounter + 1;
64:     DBCounter ← DBCounter − 1;
65: end function
66: function INCREASE_CTR(s, s.MCounter, DBCounter)
67:     if s.MCounter > 0 then
68:         s.MCounter ← s.MCounter − 1;
69:         if DBCounter > T then
70:             Write back a DVvictim;
71:         else
72:             DBCounter ← DBCounter + 1;
73:         end if
74:     else
75:         Write back a DDvictim or a DLvictim;
76:     end if
77: end function
78: function EARLY_WB(s)
79:     if s has a DDvictim then
80:         Write DDvictim back;
81:     end if
82: end function
```

Besides a hit, $EARLY\_WB$ is also called when a missing cache block is loaded to cache as shown in line 12 and 44.

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.

4.5 Checkpointing Policy

In this section, the checkpointing policy will be presented 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.

When detecting a power-loss, we are facing two problems: 1) In SRAM, what need to be backed up? Dirty blocks and part of clean blocks. Dirty blocks definitely need to be backed up. Clean blocks that will be used most recently also can be moved to NVM to reduce miss rate after resumption. 2) Which block in NVM should be used to store these data blocks? Dead blocks or those clean blocks which will be used in the furthest future.

4.5.1 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.

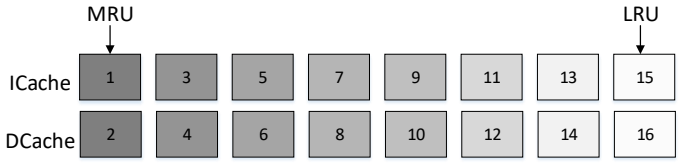

Figure 19: Clean block selecting policy

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 19, 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 project, a dead data prediction policy proposed in [42] is employed to guide the content selection, which predict dead blocks based on bursts of accesses to a cache block.

4.5.2 Selecting Non-volatile Blocks

After deciding 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 20 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 *DBCounter* 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 *DBCounter* 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 *DBCounter* will also disappear because they are based on SRAM for fast access speed. Therefore, they also need to be saved to non-volatile memory.

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.

4.6 Experimental Evaluation

This section will first present the experiment setup in subsection 4.6.1. Then, the evaluation results will be presented in subsection 4.6.2.

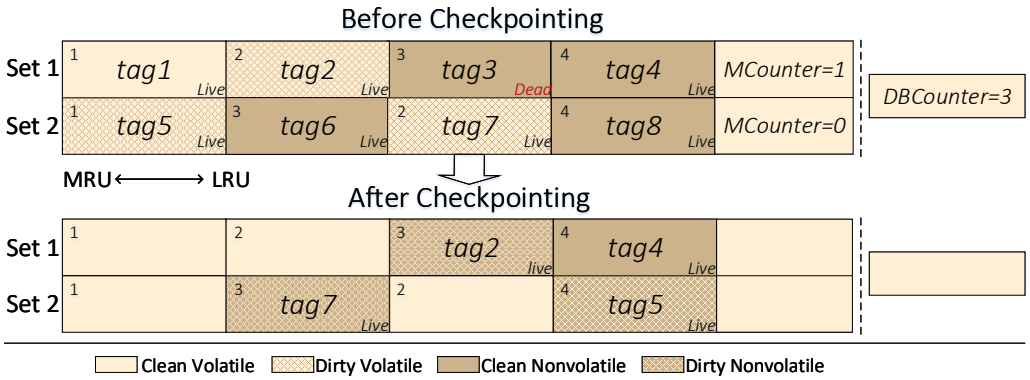

Figure 20: Checkpointing policy

4.6.1 Experiment Setup

The experiments are carried out on gem5 simulator [10]. 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 architecture of SRAM/STT-RAM set configuration consists of 2 SRAM cache blocks and 2 STT-RAM cache blocks in each set. The STT-RAM and SRAM parameters are obtained with NVSim [15].

Table 3: System configuration

Component	**Description**
Processor	480MHZ NV processor, 1 core
Cache	Private L1 cache (16K hybrid I-cache and 16K hybrid D-cache), private, 64-byte block size, 4-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

Table 5: Characteristics of benchmarks

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

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 [47] and SRAM based cache architecture as proposed in [38].

Eight benchmarks from Mibench [19] and four benchmarks from the SPEC CPU2006 suite [23] 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.

4.6.2 Results

The performance of the proposed hybrid cache architecture is evaluated in two aspects: execution progress and energy consumption.

Execution Progress Evaluation Figure 21 shows the execution progress of 12 benchmarks under 2 cache settings when the system is powered with energy harvesting technology. In this figure, the performance of the proposed hybrid cache architecture is normalized based on the non-volatile cache architecture. From this figure, we can see that the performance of the proposed hybrid cache architectures is 31% better than the non-volatile cache architecture on average.

Energy Consumption Evaluation Figure 22 shows the energy consumption comparison between pure non-volatile cache and the proposed hybrid cache. For each benchmark, there are two 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 larger dynamic energy consumption and lower leakage energy consumption. This is because the leakage power of SRAM is more than six times of STT-RAM. We can also observe that the proposed hybrid cache is more energy efficient than the pure nonvolatile cache combining noth leakage ener-

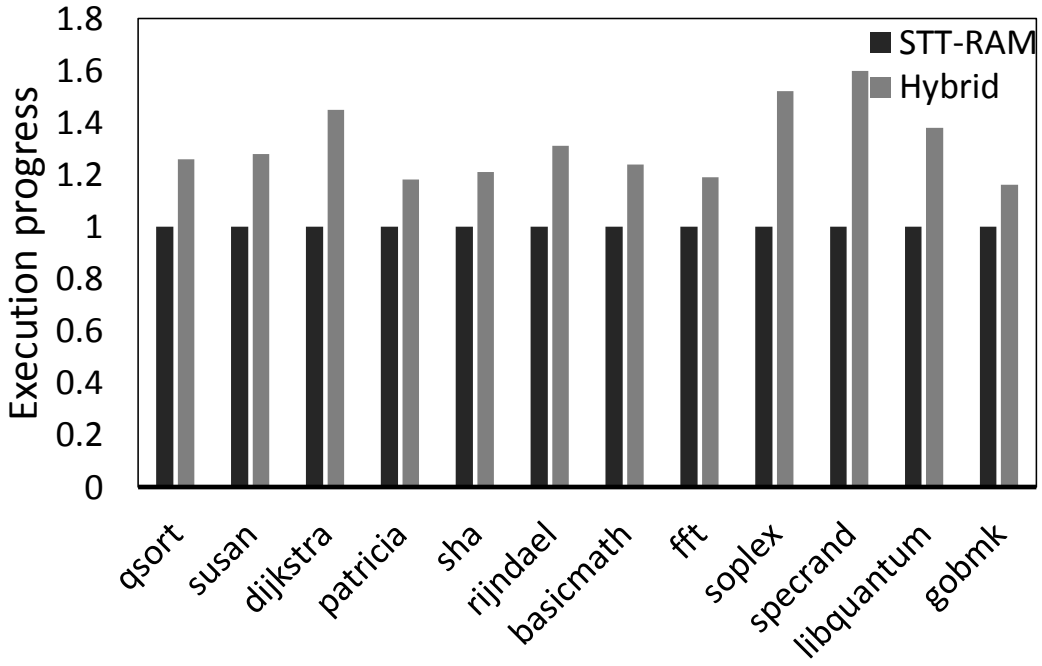

Figure 21: Comparison of execution progress under different cache architectures

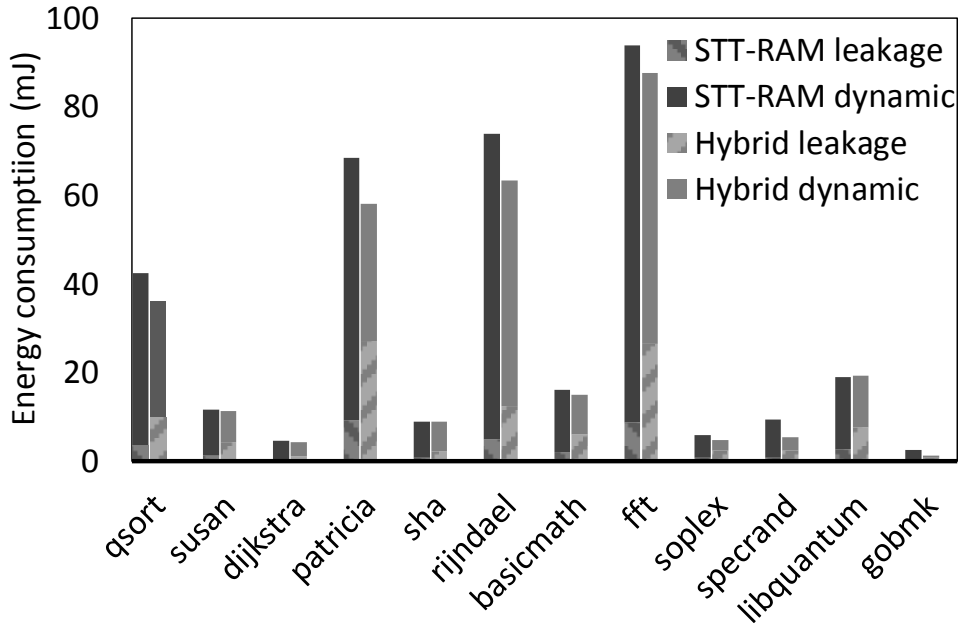

Figure 22: Comparison of energy consumption

gy and dynamic energy. the proposed hybrid cache with the pattern-predictor-based cache placement and migration policies achieves 15% energy reduction on average compared with the STT-RAM-based cache.

4.6.3 Execution Frequently Interrupted Under Harvested Power

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.

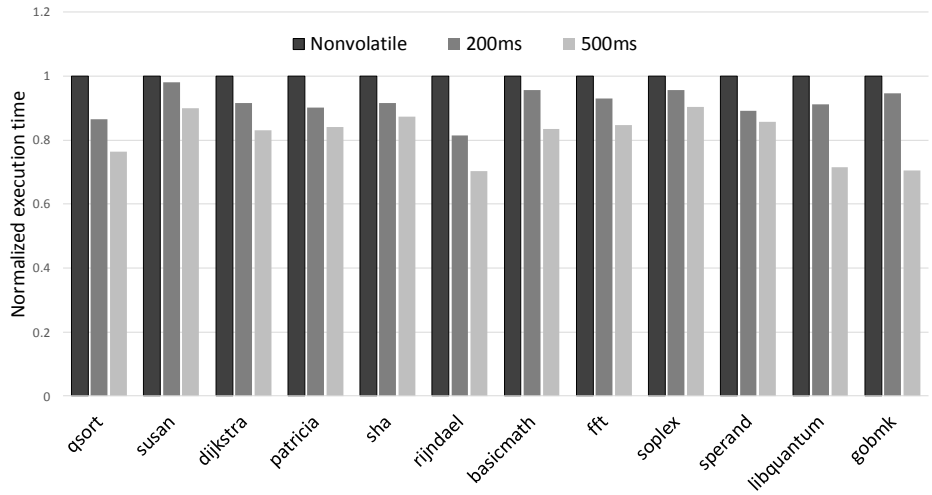

Figure 23: Execution time under frequent power failures

Figure 23 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.

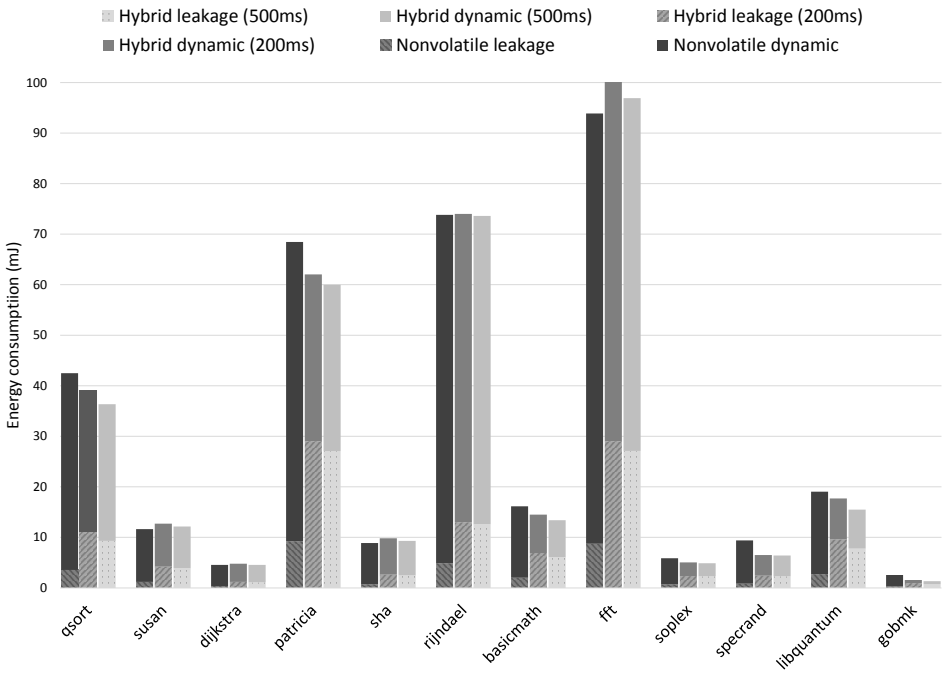

Figure 24: Comparison of energy consumption under frequent power failures

Figure 24 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.

Discussion: In this project, the hybrid one-level cache is proposed instead of simply using SRAM as L1 and STT-RAM as L2 with the write-through policy because this setup has even worse performance than the STT-RAM based one-level cache. Although write-through policy that keeps updating the inclusive cache block in L2 avoids the necessity of checkpointing upon power outage, it is energy and time consuming. Write-through is like checkpointing all the time. The overhead is much higher than write-back policy and checkpoint when there is a power failure. Furthermore, compared with the hybrid cache, simply using the STT-RAM as the L2 cache generates massive slow and energy-consuming write operations in STT-RAM while the hybrid cache can alleviate this problem signifi-cantly by migrating the write intensive cache blocks to the SRAM.

4.7 Summary

The proposed self-checkpointing hybrid SRAM/STT-RAM cache is a promising candidate to be employed in the ever-increasing self-powered embedded edge devices of IoT, due to its fast access, high-density, and low leakage. The proactive write back policy, directed by monitoring the state of the whole SRAM cache part and the migration state of each as-

sociative set, further promotes the checkpointing speed with low energy overhead. We look forward to exploring the benefits of the proposed cache in real edge devices powered with various energy harvesting technologies.

5.0 Securing Non-volatile IoT Devices With Fast and Energy-Efficient AES In-Memory Implementation

This chapter presents an AES In-Memory Implementation method for securing IoT devices embedded with non-volatile memory [84, 85]. The remainder of this chapter is organized as follows. Section 5.2 describes the background on non-volatile main memory organization and AES encryption. Section 5.3 describes the overview of this project. Section 5.4 presents the complete in-memory encryption architecture. Section 5.5 and 5.6 discuss the proposed encryption mode and key storage, respectively. Detailed experimental evaluation is provided in Section 5.7. Finally, Section 5.8 concludes this project.

5.1 Introduction

The vision of IoT system is to connect everything that's embedded with modern technologies. With the advancement of embedded system, IoT devices are becoming smaller, smarter, and more powerful which keep increasing the employment of IoT devices a variety of industries. Furthermore, combined with the benefit of closing to data, many industry leaders are transforming the attention from cloud computing to local computing with IoT edge devices. Instead of wasting a lot of time and energy transmitting the collected data to the far-away cloud, processing the data in the IoT devices is much faster and more energy efficient. Despite the fast development and increasing employment of IoT devices, the security remains as a issue to be solved.

Securing IoT devices is restricted to the limited resources in IoT devices. Because of its nature of small size, it has limited energy storage. However, security mechanisms usually requires more computing power than IoT devices can provide. On the other hand, the complexity of the backbone of IoT devices that consists of the communication platform, networks, gateways, etc. on the other hand further makes IoT device security more challenging, because it creates multiple potential vulnerabilities and bring many attack opportunities

for malicious users. To protect the data in the IoT devices against eavesdropping of communication message by unauthorized users, sensitive data must be encrypted before being transmitted to a different part of the internet. A security solution for IoT devices must secure the data stored in the device, guarantee secure communication, and protect the device from physical attacks. Although general-purpose embedded processors can do the encryption and decryption of the data, the compute-intensive requirements of encryption algorithms can heavily slow down the overall transaction. The level of integration nowadays makes it possible to integrate the encryption engine into those small IoT devices, allowing the system to handle more transactions. However, encrypting data with encryption engine is not fast enough to match the throughput of transmitter, since encryption engine between processor and memory takes a long time on the path to fetch and write back the data.

Non-volatile IoT devices have been undergoing extensive research. DRAM has been employed as the main memory for computers for decades. However, as technology scales down, DRAM will suffer from prohibitively high leakage power. Consequently, researchers are actively developing promising candidates such as phase change memory (PCM) [99], resistive random access memory (ReRAM) [82], and spin-transfer torque magnetic random access memory (STT-MRAM) [70] to be deployed as next-generation non-volatile main memory (NVMM). These non-volatile memories have several significant advantages over traditional DRAM main memory. They provide promising features such as non-volatility, high density, low leakage power, and high scalability. The nature of non-volatility avoids the need of frequent refresh for DRAM and allows the data in NVM to be retained a long time after power is off. Intel's recent announcement of 3D Xpoint [3] and the JEDEC's NVDIMM-P specification [2] are latest efforts towards the goal of next-generation NVMM.

In spite of these advantages, NVMM suffers from a new security vulnerability. Since the information in NVMM will not lose data after power is turned off, an attacker with physical access to the system can readily scan the main memory content and extract all valuable information from the main memory [20, 12]. In contrast, the security of DRAM memory relies on its short retention time which varies from 500 ms to 50 seconds [71]. To protect the data of the NVMM, the whole memory should be provided with a security mechanism with comparable security level to DRAM.

Real-time memory encryption with pad-based or stream cipher is an effective solution for this vulnerability, in which every cache line is encrypted or decrypted before being written to or read from the main memory [24]. The real-time memory encryption is a strong protection, and it can also prevent other attacks such as memory bus snooping [14]. Unfortunately, the strong protection is at the expense of runtime performance loss, since the decryption latency (as an overhead of read access) is on the critical path. Besides, encrypting and decrypting every memory access also result in severe energy overhead.

Actually, such a strong real-time protection is not always necessary. For example, when a mobile device (e.g. smart phone or laptop) is being used, the attack that requires physical access to the NVMM can rarely happen. Only when the device is shut down or put into sleep/screenlock mode, the memory encryption is required. i-NVMM [12] further proposes to encrypt main memory incrementally while maintaining an unencrypted working set which needs fast bulk encryption when necessary. Instead of the real-time encryption with performance loss and energy cost for every cache line, encrypting the working set in bulk or the whole memory when necessary is preferred in such mobile scenarios where strong protection for this part of memory is not required all the time.

Even though the bulk memory encryption approach results in no performance loss at runtime and reduces the encryption tasks hence the energy consumption, two challenges still remains: First, it should be fast in order to lower the vulnerability window when locked and provide instant response when unlocked. This is even more critical under the development of multi-core processor and increasing demand of much larger main memory. Second, it requires energy-efficient encryption considering the limited battery life.

In order to address these challenges, in this project, a novel **A**ES **I**n-**M**emory encryption architecture, AIM, is proposed for fast and energy efficient non-volatile main memory encryption. Embracing the benefit of the processing-in-memory (PIM) architecture, the proposed encryption architecture takes advantage of large internal memory bandwidth, vast bitline-level parallelism, and low in-situ computing latency. Besides, by eliminating data movement between memory and host, higher energy efficiency is achieved. Furthermore, instead of a straightforward solution that integrates a dedicated AES on the memory side, we solve the security problem that originally raised by NVM's non-volatility, with the non-volatility

itself. To this end, we leverage the non-destructive read in NVMs for performing efficient XOR operations, which dominate AES. We modify the sense amplifier circuit so that we can calculate vector XOR operations by performing a memory-read-like operation. After adding lightweight logic gates to the memory peripherals circuitry, we can perform the entire AES procedure in-place.

5.2 Background

In this section, we introduce the basics of the NVMM, the PIM architecture in NVMM, and the AES algorithm.

5.2.1 Non-volatile Main Memory

Main memory is logically organized as a hierarchy of channels, ranks, and banks. Channels work in parallel and share the same physical link to the processor. Each channel contains several ranks and each rank has several physical chips. A physical chip has several banks which contend for the same I/O in the same channel. Banks from different channels can be accessed completely independently of each other. A memory bank has several subarrays which share the global data line and global row buffer. Each subarray is a two-dimensional array of memory cells which has it local row buffer. A subarray can be further divided into different mats which has its private row buffers and write driver. Row and column addresses are often decoded at mat level. Peripheral circuitry, such as sense amplifiers and write drivers are shared among several columns.

5.2.2 Pinatubo: PIM in NVM

Besides recent research that leverages 3D-stacking DRAM like Hybrid Memory Cube (HMC) to support PIM architecture, Pinatubo [40] paves another way that leverages the emerging NVM to support PIM while incurring negligible area overheads. As shown in Figure 25, Pinatubo modifies the sense amplifier (SA) circuit of the normal emerging NVMM,

so that the SA not only can serve for reading, but also carry out bitwise operations such as AND, OR, and XOR. Instead of activating one row and reading the data out, Pinatubo activates two rows at once which correspond to the two operand vectors. The output of the SA is then the result of the bitwise operation of these two rows (vectors). To perform an XOR operation, Pinatubo first opens the one operand row, and stores the data in the capacitor inside the modified SA. Then, it opens the second operand row, and the data from this row and the previous data in the capacitor go through the simple XOR circuit inside the modified SA, after which, the readout result of this SA is the XOR result of these two rows. the AIM design takes advantage of this fast in-memory XOR operation offered by Pinatubo [40].

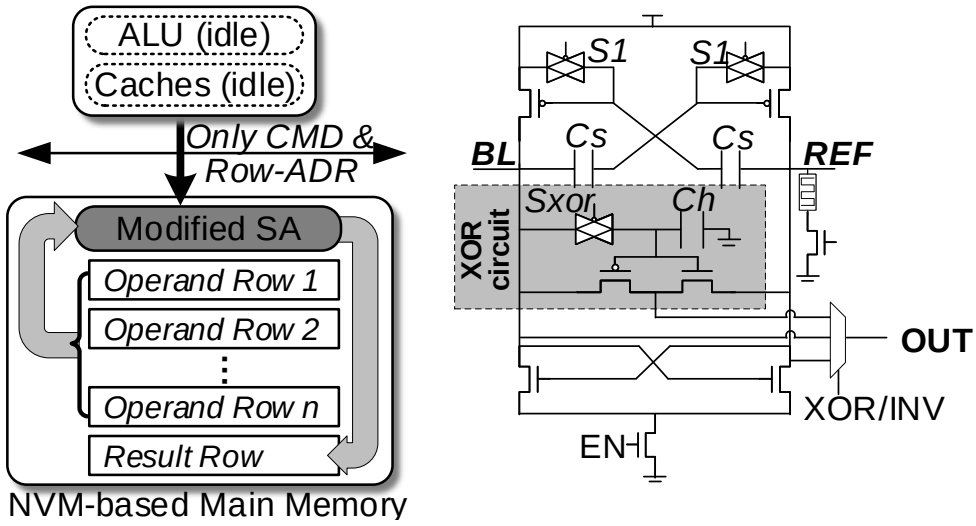

Figure 25: Left: Pinatubo's architecture computes vector bitwise operations inside NVMs. Right: SA modification in Pinatubo to perform in-memory XOR operations [40].

5.2.3 Advanced Encryption Standard

The Advanced Encryption Standard (AES) [13] is a symmetric block cipher. The AES algorithm consists of four transformations as shown in Figure 26. The intermediate results

after each transformation are maintained as a state matrix of bytes. At the start of the algorithm, a round key is added to the input by a bitwise XOR operation. After that, the state array is transformed by implementing four basic transformations 10 times when the key has 128 bits, while the last round does not include MixColumns. In summary, AES is comprised of XOR, shift, and LUT operations.

SubBytes is a non-linear invertible byte substitution that replaces each byte of the state matrix using a substitution table (S-box). As shown in Figure 26, each byte $D_{i,j}$ in the state matrix is replaced with a new byte $S_{i,j}$ in the step of *SubBytes*.

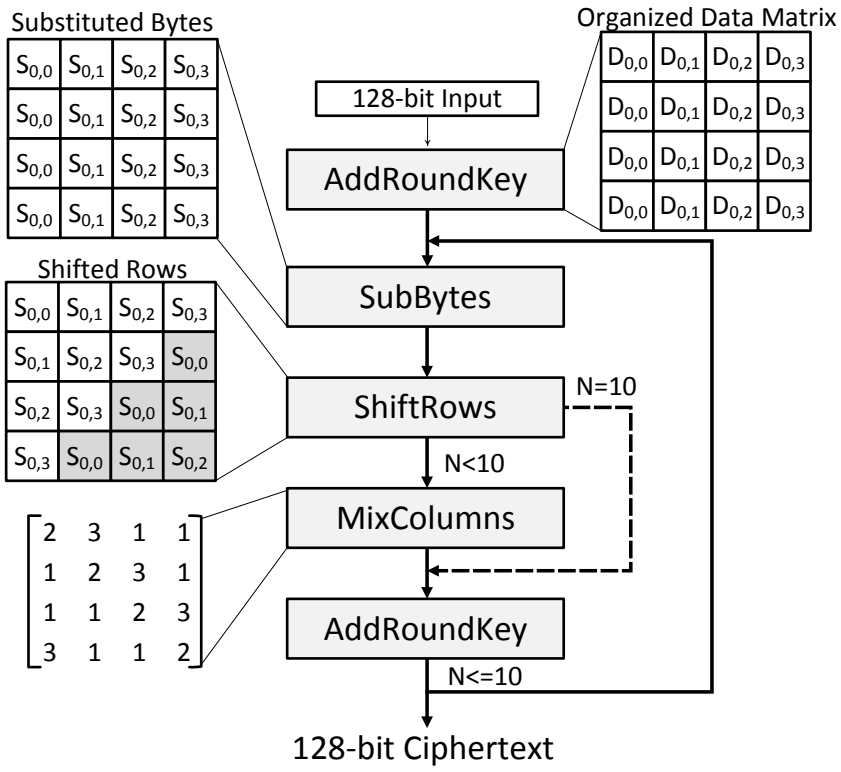

Figure 26: AES Flow Chart.

ShiftRows cyclically shifts all bytes in each row by different offsets. As shown in Figure 26, the first row is unchanged while each byte in the i^{th} row is cyclically shifted left by i bytes respectively.

MixColumns combines the four bytes of each column of the state matrix using an invertible linear transformation. This transformation can be written as a matrix multiplication in the finite field of GF(2^8) where the state matrix is multiplied by a constant matrix composed of 1, 2, and 3, as shown in Figure 26.

AddRoundKey combines the state matrix with the round key by bitwise XOR operations. Each byte in the state matrix is XORed with a byte in the same row and column of the key matrix. These round keys are generated from the key with a key schedule which expands a short key into a number of separate round keys.

5.3 Overview

5.3.1 NVMM's Vulnerability Challenge

We take the case of smartphones as a motivating example. We assume NVMM have been adapted as the replacement of DRAM, due to its advantages of low leakage and high density. The vulnerability challenge emerges that the content in the memory is under risk if the attacker stoles the device. Even though the device is locked, the attacker can remove the memory, plug it in another machine, and read it. The threat is more severe in the case of NVMM, since the retention time of NVM cells is typically much longer (a few years [63]) compared with 500 ms to 50 seconds [71] in the case of DRAM. An effective solution is real-time memory encryption with Pad-based or Stream cipher encryption, however, at an expense of performance degradation and also energy overhead (4% reported by previous work [95]). Instead of the real-time encryption, a smarter approach is to encrypt the memory only when necessary. For example, when the device is being used (unlocked), the attacker can rarely take it away. Only when the device is turned off or put into sleep/screenlock mode, we should do bulk encryption for the memory at one time. However, even though the one-time memory encryption approach result in no performance loss at runtime and reduces the encryption tasks hence the energy consumption, two challenges still remains: First, it should be fast in order to lower the vulnerability window when locked and provide instant response when unlocked. Second, it requires energy-efficient encryption considering the limited battery life.

5.3.2 PIM: A Potential Solution

To address those challenges, we propose a PIM architecture for memory encryption. The PIM offers the benefit of high internal memory bandwidth, massive parallelism (chip, bank, and subarray-level), and most importantly, it eliminates the data movement between the memory and processors. Meanwhile, we observe that in the one-time memory encryption application, the memory bandwidth is a bottleneck since a dedicated AES encryption engine provides a much larger throughput (53Gbps [49]) than the DDR throughput. Moreover, the energy for fetching data from memory with the DDR bus is also dominant. It is shown that 91.6% energy is spent on fetching and writing this data from the experimental results. Considering both advantages offered by PIM and the workload characteristics of the target one-time memory encryption application, we believe the PIM can effectively address NVMM's vulnerability challenge.

5.3.3 Design Overview

Based on the above observations, we propose AIM, an in-memory encryption mechanism for NVMM, as shown in Figure 27. Different from the co-processor AES engine (Figure 27(a)), the proposed AIM avoids the narrow DDR bus and embraces the large intra-memory bandwidth. It also benefits from multiple memory blocks parallel encryption by leveraging the flexible parallelism inside the memory, marked as chip-level, bank-level, and subarray-level parallelism in the figure. To perform the AES algorithm, we build all its required arithmetics (i.e., XOR, Shift, and LUT) inside each memory subarray. Instead of implement all those operations with logic gates, we take the advantage of NVM's unique feature and implement the most time consuming operation, XOR, within the SAs themselves, as described in Section 5.2.2 [40]. Data buffer is added to store intermediate results, reducing expansive write operations to NVM cells. In addition, an encryption controller is implemented in each chip to provide control signals to direct the encryption process. In the following sections, the details of the hardware implementation and how the AES algorithm is mapped to the proposed AIM are described.

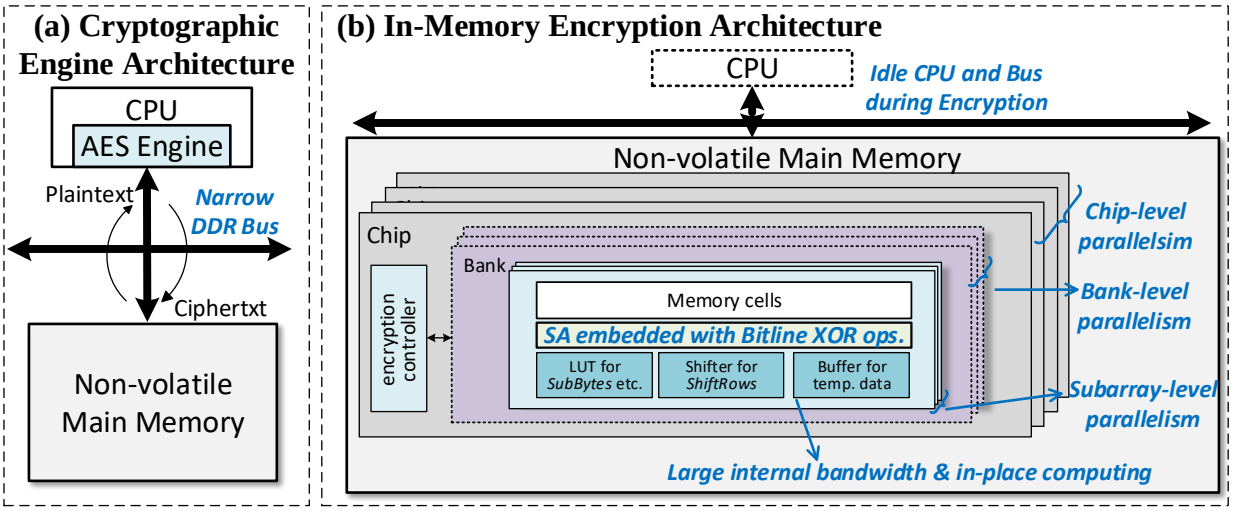

Figure 27: Memory encryption architecture: a) Traditional encryption approach implemented an cryptographic engine outside main memory, b) The proposed AIM design: in-memory computing with NVM's intrinsic features.

5.4 AES In-Memory Implementation

In this section, the implementation of AES in-memory encryption will be presented.

5.4.1 Data Organization

AES in-memory implementation takes advantage of different levels of parallelism in the NVMM. In this project, in-memory encryption is performed directly on the data in the memory cells. These data are read out with sense amplifiers (SAs), each of which is shared by several adjacent columns with a MUX as shown in Figure 28. Since the unit data matrix to be encrypted needs to be organized in a certain fashion to facilitate the encryption process, we distribute the 8 bits of each element in the data matrix into different mats and different columns in the same mat so that they can be used concurrently. In this way, the plaintext data block does not have to be pre-transformed into matrix form before encryption starts. For illustration purposes, we assume that there are M mats $Mat_i (i = 0, 1, ..., M)$, the size of each mat is $N * N$, and K columns share one SA. In total, there are N/K SAs in each mat.

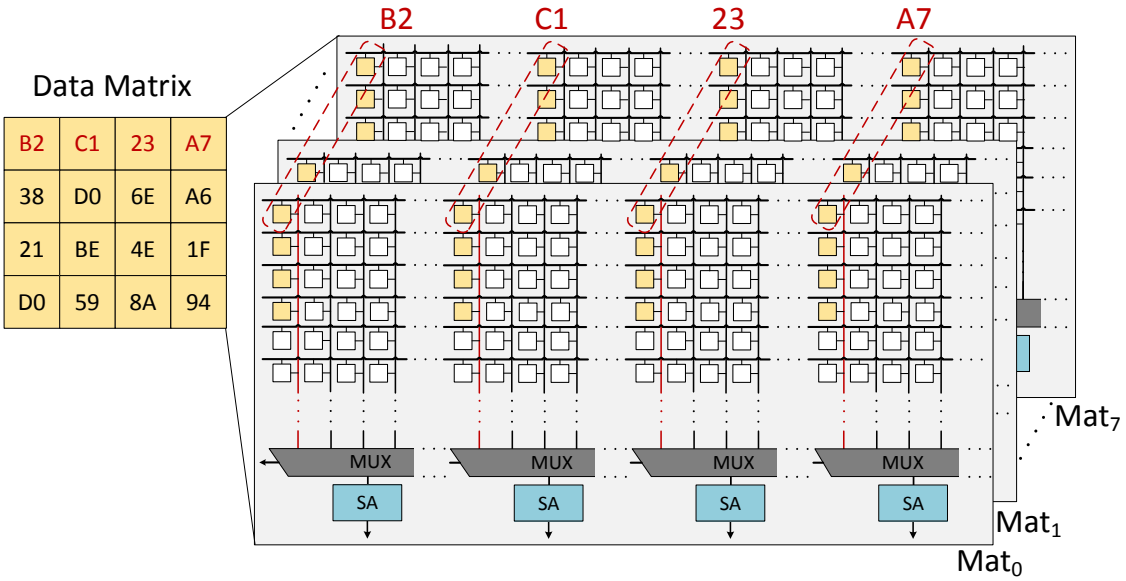

Figure 28: Distributed data organization for AES encryption.

Figure 28 illustrates the memory distribution for one data matrix. In AES algorithm, the basic processing unit is one byte of the data matrix, therefore the data matrix is distributed into eight mats so that each mat has 1-bit level of the data matrix. In order to encrypt each row of the data matrix in parallel, four columns of the data matrix are distributed to different columns of memory array connecting four adjacent SAs separately. These four columns of memory array are of the same local column address. In this way, when one row of the subarray is activated and a local column address is selected for each MUX, every four adjacent SAs will sense out a row of the data matrix. In total, every four rows of the subarray contains $MN/4K$ data blocks of 128 bits.

To enable further processing of the data matrix in different encryption stages, the intermediate results, which are the state matrices, need to be buffered. In this project, to avoid extra hardware overhead and simplify the circuity, we write the intermediate results back to the data matrix. In the proposed encryption mechanism, AES encryption generates less than 60 writes during 10 rounds of encryption to each cell in encrypted memory block. We will show that this number of writes has negligible impact on the endurance of memory. NVMM encryption is performed before the system is powered down. We assume that we need to encrypt the main memory 20 times every day and conservatively assume that the deployed non-volatile memory has an endurance of 10^9 cycles. In five years, AES encryption will generate $60 * 20 * 356 * 5 = 1.1 * 10^5$ writes which is less than 0.2% of its total life cycles.

5.4.2 AddRoundKey

In this stage, the data matrix is combined with the key matrix. Each byte of the data matrix is combined with the corresponding subkey of the key matrix using bitwise XOR operation. *AddRoundKey* is implemented with the modified SA design of Pinatubo [40], which realizes bitwise XOR operation with two micro-steps inside SA.

Figure 29 shows the process of *AddRoundKey* transformation for one row of data. First, the first row of data in data matrix is read into the added capacitor in each SA by activating the first wordline in red color and selecting a column with MUX. Second, the first row of data in key matrix is read into the latch in each SA by activating the second wordline in red color

and selecting a column with MUX. After these two steps, the bitwise XOR result of the first row is latched in each SA. Suppose it takes t_{xor} to complete XOR operation for one row of data matrix, it takes $4t_{xor}$ to complete *AddRoundKey* transformation for a data matrix since there are four rows in each data matrix. This *AddRoundKey* transformation is parallelized because of multiple SAs. Since there are M mats and N/K SAs in each mat, $(N/4K)(M/8)$ data matrices are transformed simultaneously. In our design, after *AddRoundKey* transformation for one row of data, SubBytes is performed immediately for this row of data instead of continuing performing *AddRoundKey* for all four rows.

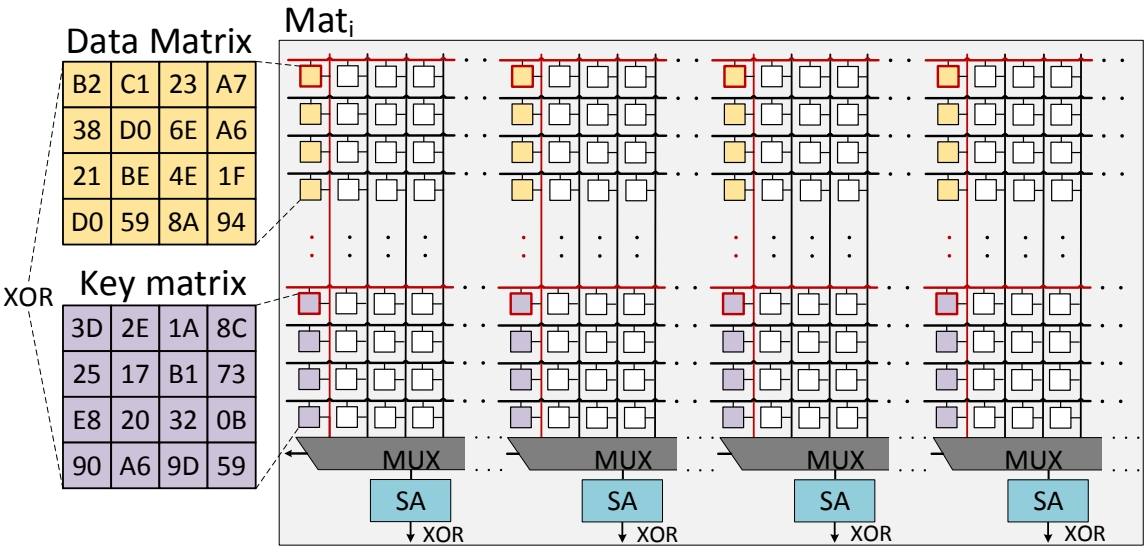

Figure 29: Addroundkey stage with xor operation.

The initial *AddRoundKey* stage is performed with the initial key. The other 10 rounds of *AddRoundKey* are performed with the corresponding round key. As shown in Figure 29, the encryption key is maintained in the non-volatile memory array and round keys overwrite the encryption key after finishing each round of encryption.

5.4.3 SubBytes

In this step, each byte of the data matrix is replaced with a new byte by doing nonlinear transformation. This transformation is realized with S-box which is used to obscure the relationship between the key and the ciphertext. The S-box can be realized with LUT by implementing combinational logic which has 8-bit input and 8-bit output or ROM which has 16 rows and 16 columns while each entry is a byte. In this project, S-box is realized with combinational logic since it incurs lower overhead.

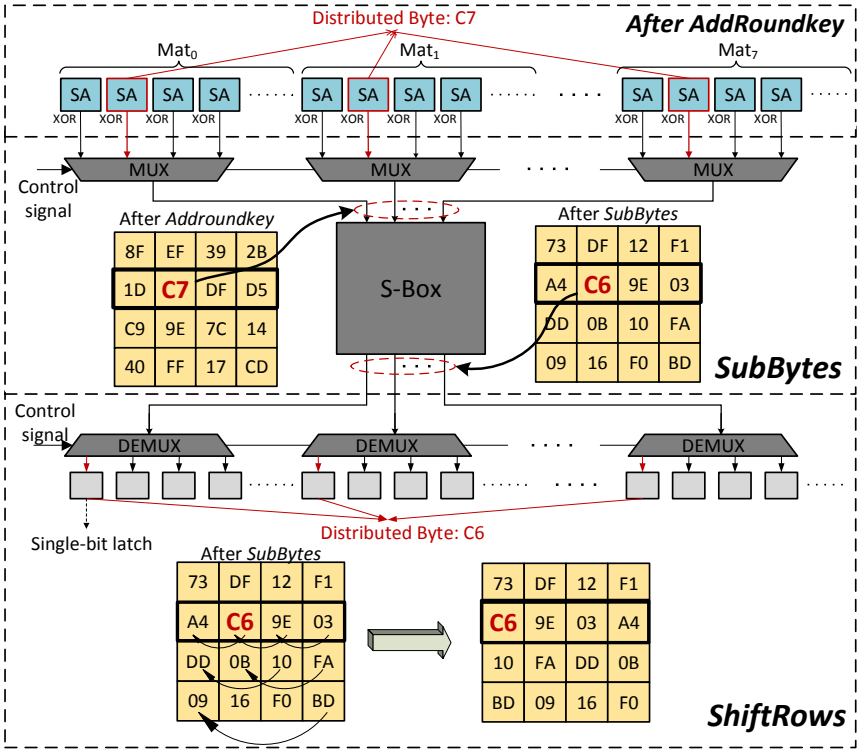

Figure 30: SubBytes transformation with LUT and ShiftRows transformation with addressing logic.

After the *AddRoundKey* stage of one row of state matrix, the intermediate results are latched in the SAs. For *SubBytes* transformation, each byte of the data matrix is decoded from eight mats and input to the S-box as shown in Figure 30. In this figure, the

AddRoundkey results of the second row of data matrix are latched in the SAs. *SubBytes* is performing on the second byte $C7$. The output of S-box is the substituted byte $C6$. In this figure, there is one S-box combinational logic which has 8-bit input and 8-bit output. Since we can only input one byte each time to the S-box, the *SubBytes* transformation can only be done sequentially which takes a long time. To accelerate the *SubBytes* transformation, we can add more S-box combinational logics to enable parallel *SubBytes* performing. At the same time, we need to consider the hardware overhead introduced by multiple S-box. We have different designs in terms of S-box considering both encryption speed and overhead. The experimental section will show the performance comparison of different designs.

After we obtain the 8-bit output of S-box, it will not be immediately written back. Instead, the next stage, *ShifRow*, will be performed on the output.

5.4.4 ShiftRows

In this step, the bytes in each row of the data matrix are cyclically shifted by a certain offset. Specifically, while the top row remains unchanged, each bit in the second row of the bit-level data matrix is cyclically shifted left by 1 bit, each bit in the third row of the bit-level data matrix is cyclically shifted left by 2 bits, and each bit in the third row of the bit-level data matrix is cyclically shifted left by 3 bits (right by 1 bit).

The *ShiftRows* transformation is realized with control signal and address decoding, as shown in the lower part of Figure 30. Originally, the 8-bit output of S-box needs to be written back where each input bit is located. This process needs address decoding to write to the right position. The *ShiftRows* transformation can leverage this address decoding process to do shifting by address decoding. By combining an offset with the column address, the output of S-box is shifted to another address according to the *ShiftRows* algorithm.

In Figure 30, the second byte $C6$ in the second row of state matrix after *SubBytes* needs to be shifted to the left by one byte. This means each bit needs to shifted left by one bit according to the data matrix distribution in the memory. This shifting process is down by selecting the first column with the control signal.

After *ShiftRows* transformation, each bit will be buffered in the single-bit latch until *SubBytes* and *ShiftRows* are performed on all data in the SAs. Then, the values in the row

buffer are transmitted to the write driver and written back to memory array. This row buffer gathers the intermediate results of one row and avoids writing to the non-volatile memory row multiple times.

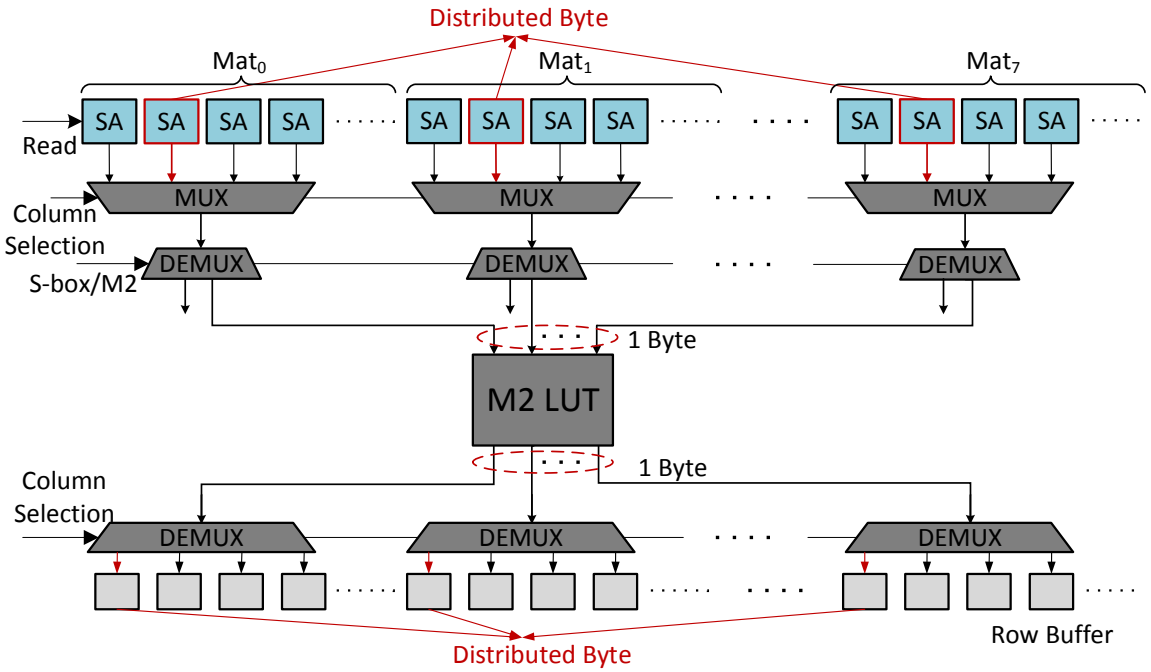

Figure 31: *MixColumn* substep: M-2 LUT.

5.4.5 MixColumns

In *MixColumns* stage, the four bytes of each column of the data matrix are combined together using an invertible linear transformation to provide diffusion in the cipher. The *MixColumns* transformation multiplies the data matrix by a known matrix as shown in Figure 26.

This matrix multiplication is done in the finite field GF(2^8), which can be decomposed to modular multiplication and XOR operations. We use $S_{i,j}$ and $S'_{i,j}$ to indicate the byte in row i, column j of the state matrix and the transformed state matrix respectively. The

MixColumns transformation is performed as follows:

$$\begin{aligned} S'_{0,j} &= 2 \cdot S_{0,j} \oplus 3 \cdot S_{1,j} \oplus S_{2,j} \oplus S_{3,j}; \\ S'_{1,j} &= S_{0,j} \oplus 2 \cdot S_{1,j} \oplus 3 \cdot S_{2,j} \oplus S_{3,j}; \\ S'_{2,j} &= S_{0,j} \oplus S_{1,j} \oplus 2 \cdot S_{2,j} \oplus 3 \cdot S_{3,j}; \\ S'_{3,j} &= 3 \cdot S_{0,j} \oplus S_{1,j} \oplus S_{2,j} \oplus 2 \cdot S_{3,j}; \end{aligned} \tag{5.1}$$

Multiplication-by-2 (M-2) in the finite field can be realized by shifting each bit of the operand left by 1 bit, followed by a XOR operation with 0x1B if the most significant bit is 1. A more efficient way is leveraging LUT. M-3 in the finite field GF(2^8) of *MixColumn* can be realized with M-2 and XOR logic. This is because

$$3 \cdot S_{i,j} = 2 \cdot S_{i,j} \oplus S_{i,j} \tag{5.2}$$

Therefore, *MixColumns* stage is decomposed into M-2 LUT and XOR operations. *MixColumns* needs several sub steps and generates several intermediate values. To both accelerate this transformation and maintain a low hardware overhead, we leverage the vacant non-volatile memory rows as buffer rows for intermediate results. The *MixColumns* stage is realized with LUT and XORs as follows:

$$\begin{aligned} S'_{0,j} &= T_j \oplus \underline{2 \cdot S_{0,j}} \oplus \underline{2 \cdot S_{1,j}} \oplus S_{0,j} \\ S'_{1,j} &= T_j \oplus \underline{2 \cdot S_{1,j}} \oplus \underline{2 \cdot S_{2,j}} \oplus S_{1,j} \\ S'_{2,j} &= T_j \oplus \underline{2 \cdot S_{2,j}} \oplus \underline{2 \cdot S_{3,j}} \oplus S_{2,j} \\ S'_{3,j} &= T_j \oplus \underline{2 \cdot S_{0,j}} \oplus \underline{2 \cdot S_{3,j}} \oplus S_{3,j} \end{aligned} \tag{5.3}$$

where

$$T_j = S_{0,j} \oplus S_{1,j} \oplus S_{2,j} \oplus S_{3,j}; \tag{5.4}$$

This first step of *MixColumns* is M-2 transformation with LUT. This process shares the same address decoding logic of S-box with a MUX as shown in 31. Since we can only input one byte each time to the LUT, this transformation can only be done sequentially which takes a long time. To accelerate this transformation, we add multiple M-2 LUT combinational logics to enable parallel performing. Like S-box design, we have different multiplication-by-2 LUT designs considering both encryption speed and overhead. After M-2 transformation,

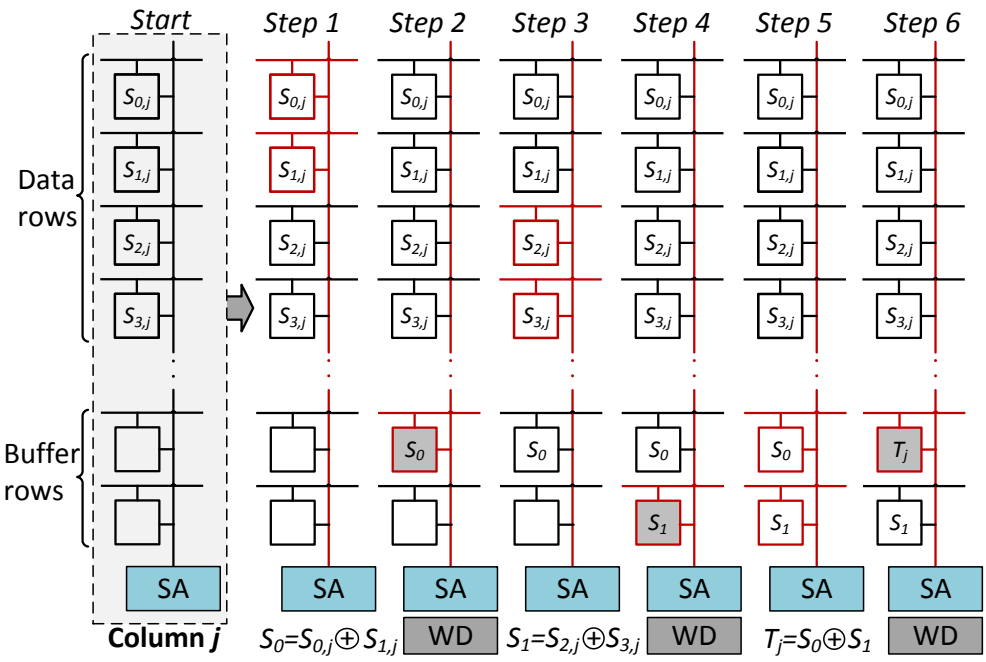

Figure 32: Example of *MixColumns* substep: Calculate T_j for each column (Eq. (4)).

outputs are latched in a row buffer until all bytes of the activated row finishes M-2 transformation. Then data in this row buffer is written to a vacant memory row. As shown in Figure 33, four empty non-volatile memory rows are used for storing LUT results.

The next step of *MixColumns* is calculating T_j following Eq. (4). Figure 32 shows the detailed process of calculating T_j for a specific column. Every time two rows are activated to get the XOR result of two memory cells, then next step this result is written to an empty buffer row. From this figure, this step costs three XOR operations and three writes. In this step, since all SAs are working simultaneously, T_j for each column is calculated in parallel.

The final step of *MixColumns* is calculating the result of *MixColumns* transformation following Eq. (3). In this step, with M-2 LUT results stored in four rows and T_j values, we can finish the *MixColumns* transformation for one row of selected columns in six steps as shown in Figure 33. This figure shows an example of how to calculate $S'_{0,j}$ in row 0. After three times of activating two rows, four operands are XORed together to get the final result of $S'_{0,j}$ and then this result is written back replacing $S_{0,j}$.

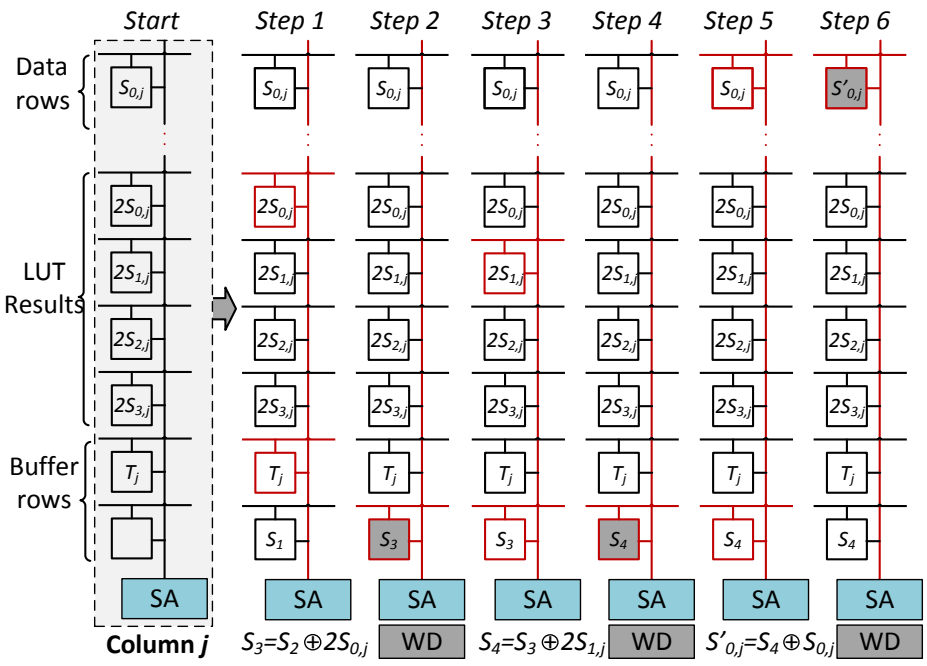

Figure 33: Example of *MixColumns* substep: Calculate $S'_{0.j}$.

During *MixColumns*, several writes are generated. In M-2 LUT step, M-2 results are written to four rows, therefore a column of 4 memory cells takes 4 writes. In the second step of calculating T_j, 3 writes are generated as shown in the colored memory cells of Figure 32. In the final step, transforming one value takes 3 writes. Thus 12 writes are generated for transforming 4 values. In total, 19 writes are generated for each column of four memory cells which means 5 writes on average are generated for each cell in *MixColumns*.

5.4.6 Discussion

The AES encryption process leverages innate parallelism of main memory to accelerate encryption. To support in-memory AES encryption, multiple S-box LUT, M-2 LUT, MUX, and DEMUX are implemented inside the memory system. For decryption, inverse S-box LUT needs to be added except for the available resources for encryption. When the memory system receives an encryption signal, the original key shared among all memory chips is

transferred to different memory chips. When each round of encryption finishes, the initial key is expended to get the round key. The encryption controller in each chip takes care of the detailed encryption and decryption process.

5.5 Cipher Modes

In this section, we will first discuss different block cipher modes that can be employed in AIM implementation for enhancing the security of encrypted NVMM. After this, we will propose a combined cipher mode that will achieve both high parallelism and security level for the proposed AIM implementation.

5.5.1 Cipher Modes

Encrypting two identical plaintext blocks with the same key will generate two identical ciphertext blocks. An attacker would be able to achieve useful information and discover the original plaintext by analyzing the identical blocks of the ciphertext. To allow block ciphers to work with a large number of data blocks, different block cipher modes of operations are devised to blur the ciphertext so that the ciphertext blocks of two identical plaintext are different. Common modes of block cipher include ECB, CBC, CFB, OFB, and CTR [9].

5.5.1.1 Electronic Codebook (ECB) ECB is the simplest mode that encrypts each data block of the input plaintext separately. Since there is no dependency in encrypting different data blocks, this cipher mode allows different blocks to be encrypted simultaneously and supports high parallelism. However, if there are identical plaintext blocks in the NVMM, encrypting bulk NVMM with the same key is vulnerable.

5.5.1.2 Cipher Block Chaining (CBC) In the CBC mode, the next plaintext is always XORed with previously produced ciphertext block before it is encrypted. Since there is no previous ciphertext block, the first plaintext block is XORed with a random initialization

vector (IV) which has the same size as a plaintext block. As a result, every subsequent ciphertext block depends on the previous one. Since the CBC mode encrypts the plaintext sequentially, it will lead to high latency in the AIM encryption process. Different from encryption, decrypting different cipher blocks can be done simultaneously.

5.5.1.3 Cipher Feedback (CFB) In the CFB mode, the previous ciphertext block is encrypted and then XORed with the next plaintext block to generate the next ciphertext block. Since there is no previous ciphertext block before the first plaintext block, a random IV is encrypted and then XORed with the first plaintext. Similar to the CBC mode, encryption in CFB mode is performed sequentially while decryption can be performed simultaneously. Therefore, it suffers similar drawback to the CBC mode. Compared with CBC mode, the CFB mode only uses the encryption of the block cipher. Therefore, the CFB mode gets rid of the required resource for implementing decryption.

5.5.1.4 Output Feedback (OFB) The OFB mode creates keystream blocks with the original key and a random IV, which are then XORed with the plaintext blocks to get the ciphertext blocks. Because of the continuous creation of keystream bits, both encryption and decryption is done sequentially. Therefore, this mode has poor parallelism. Besides, the usage of only the encryption of the block cipher gets rid of the required resource for implementing decryption.

5.5.1.5 Counter (CTR) The CTR mode creates keystream blocks by encrypting a nonce value added by an increasing counter. The plaintext blocks can be encrypted simultaneously with different counters allowing high-level parallelism. However, CTR mode becomes vulnerable if counters repeat. This mode also gets rid of the required resource for implementing decryption.

Besides the five cipher modes, GCM [50] is also a very interesting and powerful cipher mode. However, the implementation of GCM requires adding more circuitry to the memory architecture than other cipher modes because of its authenticity and confidentiality ability. Meanwhile, GCM requires more steps to generate a tag for authenticity and thus has much

longer encryption time and energy consumption compared with the other cipher modes. Because of the much larger area overhead and the lower performance and energy efficiency, GCM mode has inferior performance to the other discussed cipher modes. Therefore, GCM mode is not considered in this project.

5.5.2 CTR-CFB Encryption

The cipher algorithm requirement and parallelism of encryption direction and decryption direction are summarized in Table 6. Among them, CTR mode has the best parallelism based on counters. CFB, OFB and CTR only need encryption direction implementation which saves hardware resource for implementing decryption. Compared with OFB, CFB has better parallelism in decryption.

A direct solution to enhancing the security is deploying CTR mode which allows high parallelism. However, CTR mode fails catastrophically when a counter value is reused, because it's a pure xor stream cipher: XORing two ciphertext blocks that were generated with the same key and counter values cancels out the encryption. Therefore, in this project, we propose to combine CTR and CFB modes to enhance the AES security while maintaining the parallelism level.

Table 6: Comparison of different cipher modes

Modes	Cipher requirement		Parallelism	
	Encryption	Decryption	Encryption	Decryption
ECB	✓	✓		✓
CBC	✓	✓		✓
CFB	✓			✓
OFB	✓			
CTR	✓		✓	✓

The implementation is shown in Figure 34. In the vertical direction, the CFB mode can be implemented, since each word row needs to be activated for encryption one by one in

sequential. In the horizontal direction, CTR mode can be implemented to allow parallelism since several columns can now be encrypted simultaneously. The introduction of the CFB mode to CTR mode avoids the counters for the vertical encryption direction.

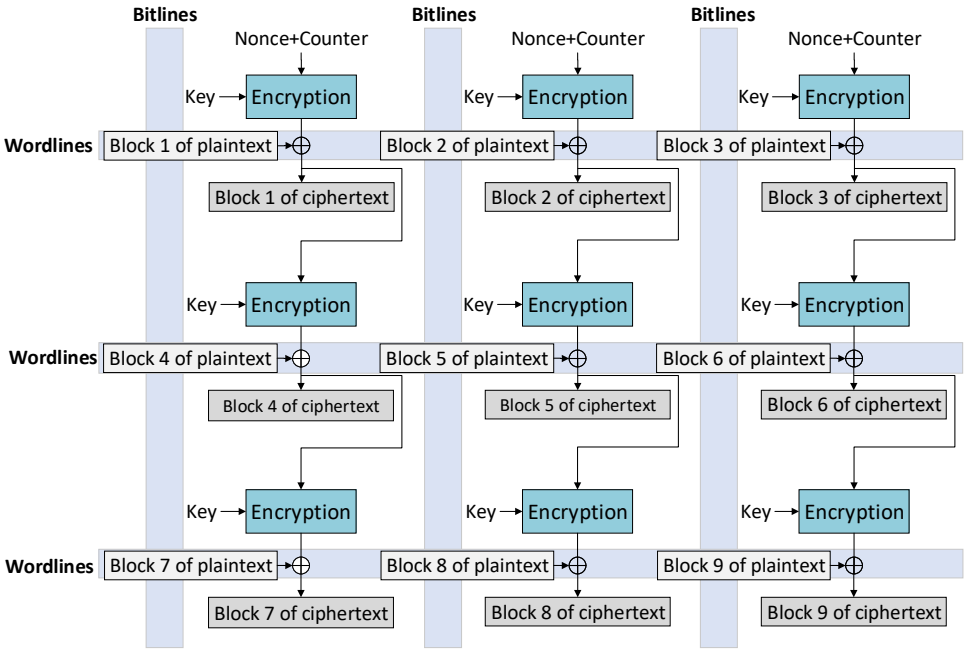

Figure 34: Encryption of CTR+CFB cipher mode.

The challenges in the implementation lie in generating nonce which is a random number, and the design of counters (different value for different blocks). For generating a nonce, there are two ways: first, the key generator can generate a second key as the nonce; second, the original key can be used to generate the nonce by hashing the original key. To have different counters for different blocks, the bank id, subarray id, mat id, and column id are concatenated together to generate different counters for the first row of plaintext blocks in NVMM as follows:

$$\begin{aligned} Counter = & Bank_{id}||Sub_{id}||Mat_{id}||Column_{id}||Mux_{id} \\ & ||Counter_{+1} \end{aligned} \tag{5.5}$$

where $Counter_{+1}$ is the increment-by-one counter, which works by incrementing by one after finishing one time of bulk encryption. Therefore, the encryption function for the first row of data blocks is as follows:

$$C = P \oplus En_{key}(Nonce + Counter) \tag{5.6}$$

This design of *Counter* guarantees a unique sequence for each plaintext so that different plaintext blocks are encrypted with different key blocks. Meanwhile, the same counter sequence will not repeat for a long time so that the same plaintext block will not be encrypted with the same keystream twice for a long time, thus ensuing the security of the proposed cipher mode.

5.5.3 CTR-CFB Decryption

The decryption process of CTR+CFB cipher mode is shown in Figure 35. From this figure, the decryption of different columns of data blocks are decrypted simultaneously while the data blocks for the same columns are decrypted sequentially. This cipher mode only uses the encryption algorithm of the block cipher as shown in the blue box, thus avoiding the required resource for implementing decryption algorithm especially the inverse S-box.

5.6 Key generation and storage

The method of key generation, key storage, and key handling significantly influences the security of the crypto-systems. In this section, we will first describe three possible master key generation schemes. Then, we will describe the round key generation in AIM.

5.6.1 Master Key Generation and Storage

For the AIM encryption mechanism, there are three possible ways of generating the master key: user input, randomizer, and physical unclonable function (PUF). The user input

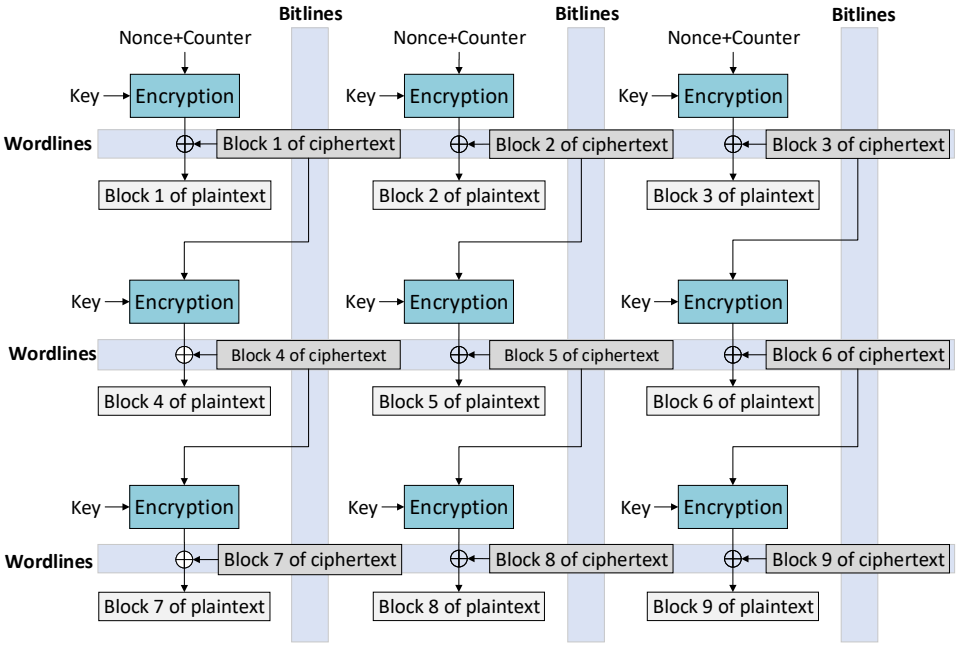

Figure 35: Decryption of CTR+CFB cipher mode.

method allows the user of a device to input a key that the user can remember or a biometric-based key. After inputting a key, This key is then transferred to the NVMM to start the AES in-memory encryption. After finishing bulk encryption, this key is cleared. When the user wants to use the device again, the same key is input to decrypt the non-volatile main memory. In this way, there is no key storage overhead or key leakage risk. The second way of creating the original key, randomizer, is to use a pseudo-random number generator to create a one-time random number. Since the device might be powered down, this key should be stored in a protected non-volatile memory for decryption. Therefore, this key should be placed far away from the non-volatile main memory, such as in the processor, to keep the generated key away from the attackers.

Compared with a randomizer, PUF avoids the need of key storage in non-volatile memory. The process of extracting a key from the physical intrinsic properties due to different materials and physical variations from fabrication process of hardware is described in [48]. PUF-based key generator avoids the need of a pseudo-random number generator by harvest-

ing the hardware-unique randomness and processes it into a cryptographic key. Since the randomness is already intrinsically present in the device, there is no need for a protected non-volatile memory. Since the randomness is static throughout the lifetime of the device, it can be harvested again to regenerate the same key for decryption. This PUF-based key cannot be found by an attacker who opens up the device because the key is not permanently stored and not present when the device is not active. This way of deriving keys has security advantages compared to randomizer which needs key storage in non-volatile memory.

5.6.2 Round Key Generation - Rijndael Key Schedule

AES requires a separate round key for each round of encryption to achieve a high level of confusion. Expanding the original key into several rounds of keys in AES is known as the rijndael key schedule. AES key expansion consists of RotWord, SubWord, XOR operations, all of which can be realized with the previously introduced implementations of AES encryption. Therefore, the round keys can also be generated within NVMM instead of using a dedicated key generator. In the proposed CTR+CFB cipher mode implementation, the key for each round is generated only once and stored in the NVMM for each time of memory bulk encryption. After completing the encryption, these round keys are cleared to avoid key information leakage.

5.7 Experimental Evaluation

In this section, we evaluate the proposed method and compare it with state-of-the-art solutions.

5.7.1 Experiment Setup

AIM is evaluated on both MRAM-based and PCM-based main memory with a DDR3 interface and 65nm technology. The MRAM-based main memory has a 512-bit page size. We conservatively assume the MRAM has 256Mb per chip with a $34F^2$ cell size. The PCM-

based main memory's page size is 1024 bit, and the capacity is 1Gb per chip with the cell size of $9F^2$. We modified NVSim [16] and Cacti-3DD [11] to achieve the parameters for the NVM-based main memory. Table 7 lists the parameters of PCM and MRAM at bit level for main memory implementation [67]. To evaluate the circuitry we added to support AIM, we synthesize these circuits with Design Compiler with FreePDK.

We compare AIM with three different dedicated memory encryption engines (EEs) as follows:

EE-1 [21] designs an AES encryption hardware core suited for devices with low power consumption. It has a maximum frequency of 290MHz, and takes 9.9nJ and 160 cycles to encrypt a data block.

EE-2 [49] implements an AES CMOS ASIC encryption core which has a frequency of 2.1GHz with a total power consumption at 125mW, and takes 5 cycles to encrypt 4 data blocks with an area of $4400um^2$.

DW-AES [79] implements an AES encryption core with domain-wall nanowires which has a frequency of 30MHz and takes 1022 cycles and 2.4nJ to encrypt 1 data block.

For the proposed design, we evaluate different configurations described as follows.

AIM: is the basic configuration, where only one bank works on encryption at one time.

AIM-B: has the encryption add-on circuit for each bank. To perform a whole memory encryption, all banks in a chip can work in parallel.

AIM-S: has the add-on circuit for each subarray. By leveraging the subarray-level parallelism [36], multiple subarrays in the same bank work on the encryption/decryption task simultaneously.

Table 7: PCM and MRAM parameters at bit level

Features	**PCM**	**MRAM**
Read Latency (ns)	27.17	31.97
Write Latency (ns)	146.39	41.52
Read Energy (pJ)	0.04	0.03
Write Energy (pJ)	0.12	0.06

5.7.2 Performance and Energy Evaluation

In this section, we evaluate AES from four aspects including encryption latency, power, energy efficiency, and area overhead.

5.7.2.1 Latency Figure 36 shows the encryption latency of 1GB memory. We have three observations. First, the encryption latency becomes quite large when the size of non-volatile memory is large. For a low-frequency encryption engine DW-AES, encrypting the whole memory can take as long as many hours or days. Second, for an encryption engine of very high frequency, the encryption latency is very small. If the writing latency is larger than the encryption latency, the encryption latency will be countervailed by the writing latency. EE-2 has very high encryption speed, the time it takes to encrypt the whole memory turns to the time of reading and writing to all memory blocks sequentially. Therefore, the encryption time of EE-2 is different for PCM and MRAM as shown in the corresponding two columns of Figure 12. On the contrary, EE-1 has very low encryption speed and its encryption latency is much larger than the data movement which is thus overlapped by the encryption time that dominates the overall latency. Therefore, the encryption time of EE-1 is the same for both PCM and MRAM as shown in the corresponding two columns of EE-1. Fourth, multiple levels of parallelism in non-volatile memory accelerate the encryption process of AIM mechanism. When only one bank works in a chip, AIM can reach the encryption speed of 21 seconds and 1.2 seconds if the memory is implemented with PCM and MRAM, respectively. When we enable bank-level parallelism and let banks encrypt independently, AIM-B is able to encrypt the whole memory in 2.66 seconds and 0.15 seconds correspondingly. When the subarray level parallelism is enabled, AIM-S is able to encrypt the whole memory in 0.33 seconds and 0.018 seconds for PCM and MRAM, respectively.

From Figure 36 we can see that EE-2 has the fastest encryption speed. When the main memory is implemented with PCM, the AIM-B design has similar encryption performance for 1GB main memory and AIM-S can encrypt 1GB much faster than EE-2. When the main memory is implemented with MRAM, all three designs AIM, AIM-B, and AIM-S work faster than EE-2. Besides, when the size of non-volatile memory scales up, the latency of EE-2 will

scale up accordingly. However, for AIM-B, as long as main memory power budget allows, it can continue to leverage the parallelism and maintain a short encryption latency.

5.7.2.2 Power All three designs AIM, AIM-B, and AIM-S work within the power budget [22] of main memory. Among the three designs, AIM has the smallest power which is around 1mW and 13mW for encrypting one chip of PCM-based main memory and MRAM-based main memory, respectively. The power of AIM-B is around 8mW and 108mW for encrypting one chip of PCM-based main memory and MRAM-based main memory respectively. AIM-S has the largest power consumption since it has the best performance among the three designs and the power is 70mW for encrypting each chip of PCM-based main memory. When the main memory is implemented with MRAM, the power of AIM-S exceeds the budget since the parallelism of AIM-S is the highest. Therefore, this design is not recommended if the power budget is small. However, since AIM-S has the best performance, if the system has the need for fast encryption and a large power budget, this design can still be employed. In conclusion, when we implement the AIM encryption schematic inside the NVMM, both power budget and encryption latency should be considered together to choose the most suitable design.

5.7.2.3 Energy Efficiency Figure 37 compares the energy efficiency for encrypting a 128-bit block and for encrypting 1GB main memory sequentially. From Figure 37(a), EE-2 incurs smallest energy, 0.265nJ to encrypt 128-bit block while AIM ranks the third and costs 2.78nJ and 3.17nJ for 128-bit PCM and MRAM blocks, respectively. Figure 37(b) shows the energy consumption for encrypting 1GB non-volatile PCM and MRAM. In this figure, the lower parts of the first 6 columns show the energy spent on accessing main memory and the upper parts show the energy spent on encrypting process with the encryption engines. From this figure, we have two observations. First, EE-1, EE2, and DW-AES cost significant amount of energy on memory access. This is because, for an encryption operation outside of main memory like those of EE-1 and EE-2, the encryption processor needs to read a memory block from the main memory and then write this memory block back to its original position after encryption is completed. During the reading and writing periods, complex address

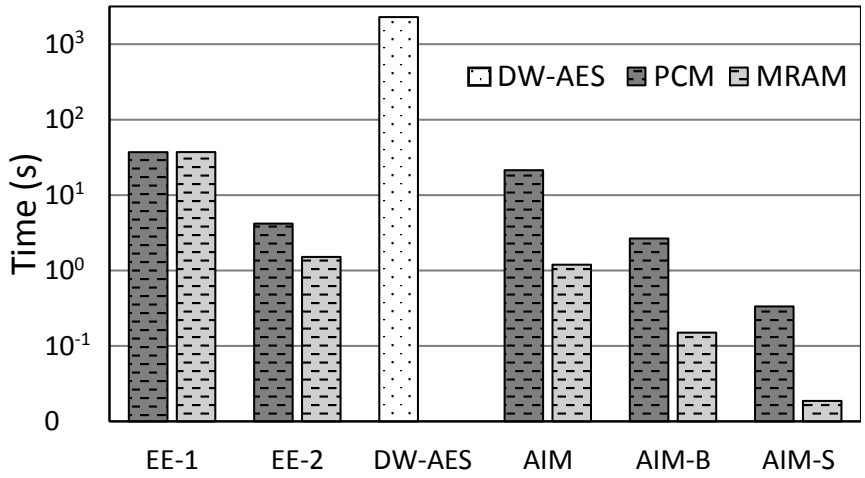

Figure 36: Comparison of latency among different baselines and different AIM designs.

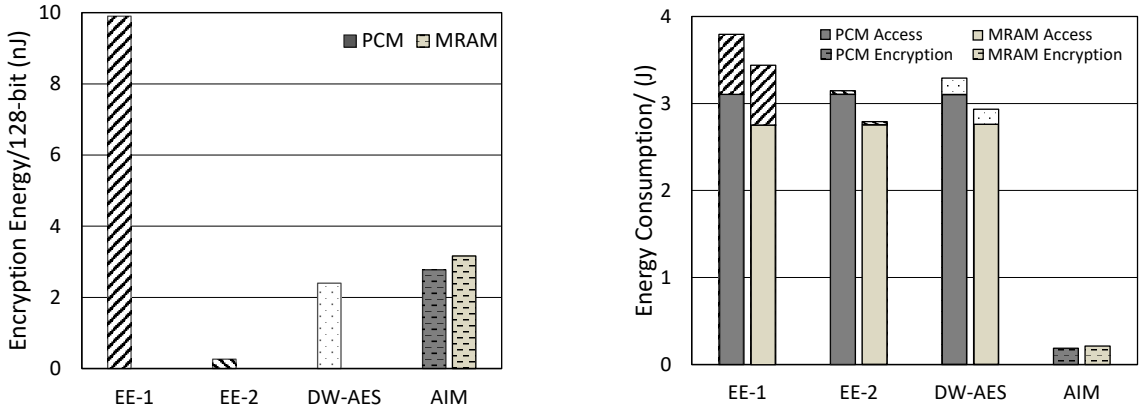

Figure 37: Left: Energy for encrypting 128-bit block. Right: Energy for accessing and encrypting 1GB main memory.

decoding and bus transfer costs a lot of energy which is much more than the energy spent for encrypting this block. For DW-AES, the large energy comes from the large number of shifting operations required for write to perform the AES with DWM. Second, AIM costs the lowest energy compared with the three specific encryption engines. Since AIM encrypts each memory block inside the main memory, it avoids a large part of reading and writing energy consumption from outside the main memory.

5.7.2.4 Overhead Evaluation Figure 38 shows the area overhead results. As shown in this figure, AIM and AIM-B both incur insignificant area overhead of only 0.06% and 0.45% area overhead for PCM-based main memory, and 0.08% and 0.63% for MRAM based main memory. Compared with AIM and AIM-B, AIM-S incurs a relatively larger area overhead of 3.59% and 5.05%.

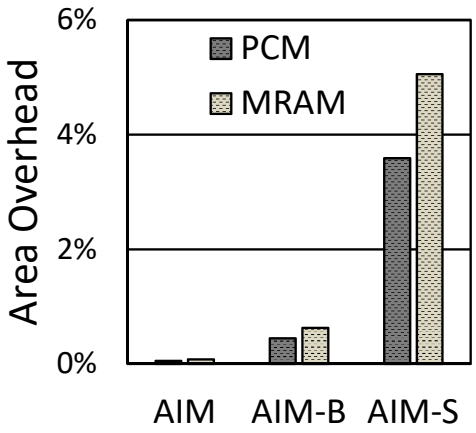

Figure 38: Different AIM designs area overhead.

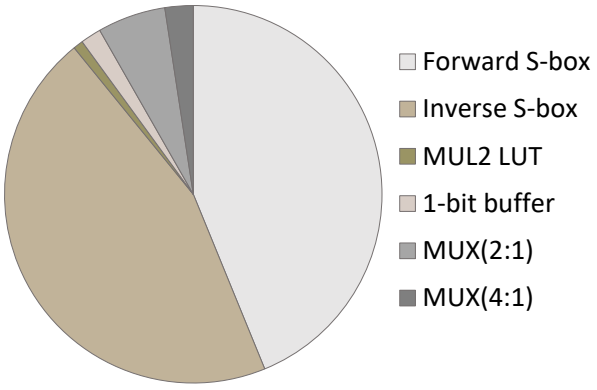

Figure 39: Breakdown of encryption overhead.

Figure 39 shows the distribution of hardware overhead. Among all added circuitry, forward S-box and inverse S-box have the largest area overhead. Since we can only look up byte by byte each time, more S-box LUTs mean more parallelism. Therefore, this overhead is unavoidable.

Besides the area overhead for added circuitry, buffer rows are required for storing intermediate results generated in the encryption process. As shown in Figure 32, 6 buffer rows are required at most in the MixColumns step. For a normal memory bank that has 512 rows, 6 buffer rows is only 1.2% of the whole memory size. During the normal working time of the main memory, these 6 buffer rows can also be used for storing working data.

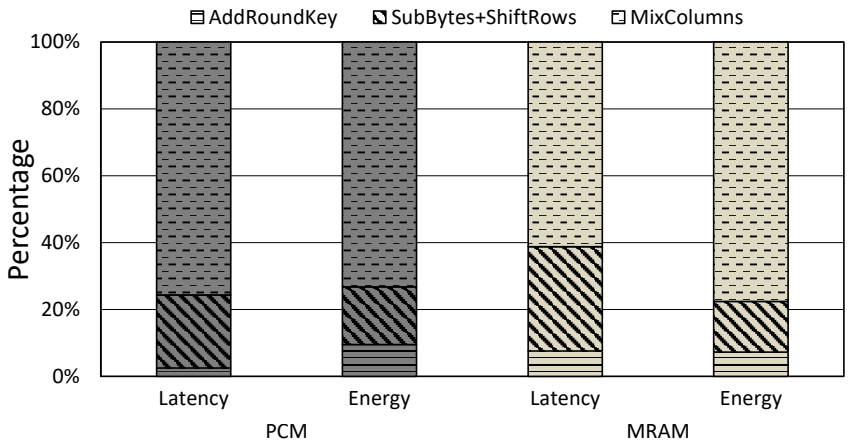

Figure 40: Breakdown of latency and energy consumption.

5.7.2.5 Further Improvement The breakdown of latency and energy consumption of AIM implementations for different encryption stages is shown in Figure 40. Since *SubBytes* and *ShiftRows* are combined together in the AIM design, we analyze the two stages together. From Figure 40, *AddRoundKey* consumes the minority of both total encryption latency and energy. This is because *AddRoundKey* stage only consists of parallel XOR operations based on Pinatubo design which is fast and costs a little energy. *MixColumns* consumes the medium latency and energy. This stage involves LUT operations of S-box. The latency of this stage varies with the number of S-box. *MixColumns* consumes the majority of both total latency and energy. This is because *MixColumns* generates several intermediate encryption state matrices from sub-steps. These intermediate encryption state matrices are buffered

in the non-volatile memory cells in AIM design. This buffering process costs considerable energy and latency, since write operations in NVMM are usually expensive in terms of both energy and latency.

Writing pulse width to NVM determines the retention time of the written states. In AIM, the encryption latency and energy can be further reduced by supporting short-latency light writes, since the intermediate encryption states only need to stay for a short while. As a consequence, buffering intermediate encryption states with light writes will cost less energy and latency.

5.7.3 Evaluation of Different Cipher Modes

In this section, we evaluate the proposed CTR+CFB cipher modes from three aspects including encryption latency, energy efficiency, and area overhead. Among the evaluated cipher modes, CBC mode and CTR mode are used as baselines. The other kinds of cipher modes are not evaluated since they do not support parallel encryption.

5.7.3.1 Latency Figure 41 shows the encryption latency of 1GB memory with three different cipher modes under AIM design. Among the three cipher modes, the CBC mode has the shortest encryption latency. Compared with the CBC mode, the CTR mode increase the encryption latency by 2.17% and 2.97% when the main memory is implemented with PCM and MRAM, respectively. Compared with the CTR mode, the proposed CTR+CFB cipher only slightly increases the encryption latency by 0.25% and 1.00% when the main memory is implemented with PCM and MRAM, respectively.

5.7.3.2 Energy Efficiency Figure 42 shows the encryption latency of encrypting one memory block with three different cipher modes under AIM design. Among the three cipher modes, the CBC mode has the highest energy efficiency because it generates the lowest energy overhead. Compared with the CBC mode, the CTR mode increase the energy consumption by 2.96% and 2.68% when the main memory is implemented with PCM and MRAM, respectively. Compared with the CTR mode, the proposed CTR+CFB cipher only

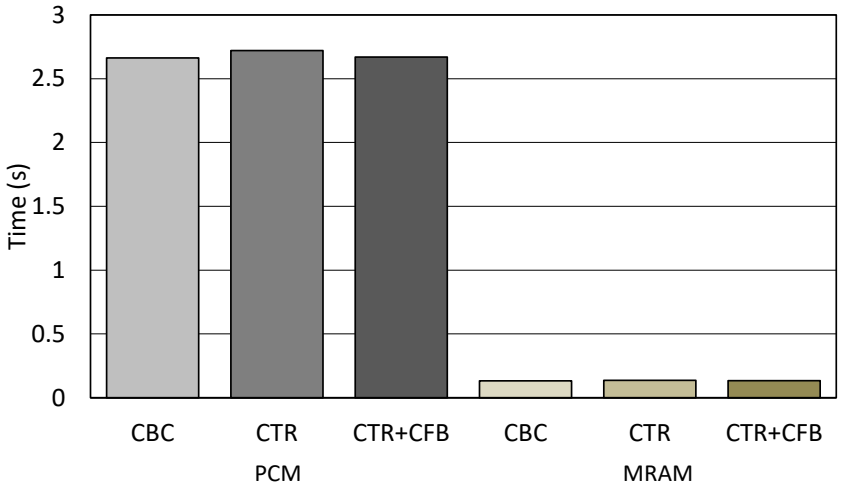

Figure 41: Comparison of latency among different cipher modes.

slightly increases the energy consumption by 0.88% and 0.68% when the main memory is implemented with PCM and MRAM, respectively.

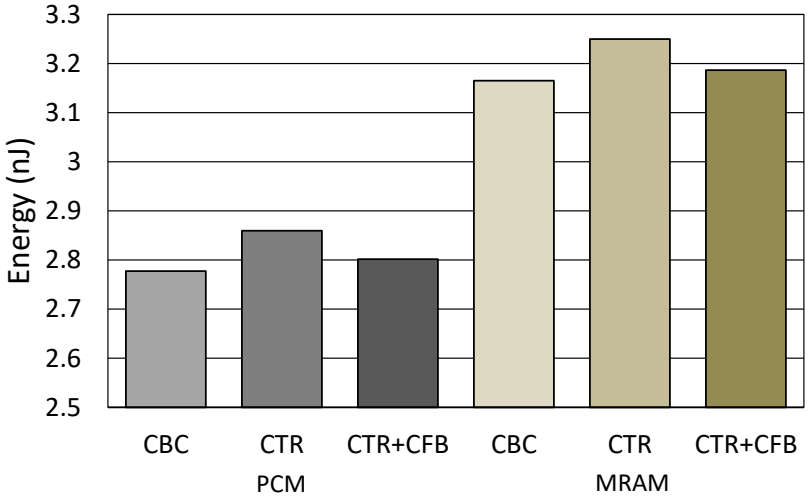

Figure 42: Comparison of energy for encrypting 128-bit block among different cipher modes.

5.7.3.3 Overhead Figure 43 shows the area overhead results of implementing the proposed CTR+CFB cipher mode. As shown in this figure, AIM and AIM-B both incur in-

significant area overhead of only 0.03% and 0.26% for PCM-based main memory, and 0.05% and 0.37% area overhead for MRAM-based main memory. Compared with AIM and AIM-B, AIM-S incurs a relatively larger area overhead of 2.10% and 2.95% for PCM-based main memory and MRAM-based main memory. Compared with the CBC mode as shown in Figure 38, the proposed CTR+CFB mode further reduces the area overhead by 40% on average by removing the inverse S-box.

Figure 44 shows the distribution of area overhead. Compared with Figure 39, the overhead from inverse S-box is removed. Among all added circuitry, forward S-box has the largest area overhead.

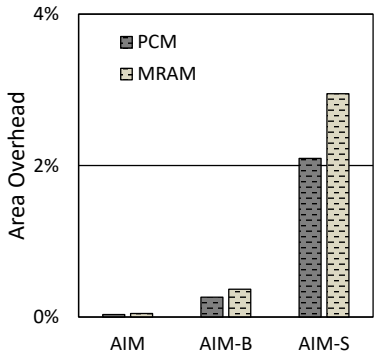

Figure 43: Different AIM design area overhead with the CTR+CFB mode.

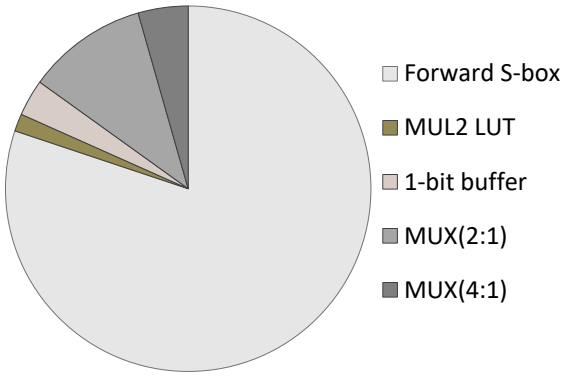

Figure 44: Breakdown of encryption overhead with the CTR+CFB mode.

5.7.3.4 Discussion The proposed CTR+CFB cipher avoids the complex management of large amount of counter values and the security problem of repeated counter in a short time. Meanwhile, the whole decryption process is based on AES encryption algorithm, saving the hardware resource for implementing AES decryption algorithm. The above experimental results show that the proposed CTR+CFB cipher mode achieves comparable or even lower latency and energy consumption with small hardware overhead.

The high encryption performance of AIM is owing to the high parallelism inside main memory architecture instead of relying on a specific kind of memory technology. As long as the memory cells are resistance-based, they can be used with the proposed architecture. Furthermore, from our experimental evaluation, the activation latency of a memory row is much larger than the read/write latency of a memory cell and thus dominates the overall memory access latency. This activation latency mainly depends on the length of a memory row and the circuit design instead of the memory type. From our research, most NVMs have

the properties varying between MRAM and PCM. Therefore, the proposed techniques are good for other NVMS as well.

Although AIM requires memory architecture modification for encryption, the modification is small, simple, and easy to implement. The three levels of encryption parallelism AIM supports and no requirement of data movement with in-memory design bring benefits of significantly improved encryption throughput and lowered energy overhead. Therefore, the advantages of AIM are much more than the drawbacks AIM brings.

5.8 Summary

In this chapter, a fast and energy-efficient AES in-memory implementation is proposed for non-volatile IoT devices, by taking advantage of the resistive nature of non-volatile memory and utilizing existing memory peripheral circuits. With AIM, the memory blocks are encrypted simultaneously within each memory bank and the entire encryption process can be completed within the main memory without exposing the results to the memory bus. Compared with state-of-the-art AES engine running at 2.1 GHz, AIM can speed up the encryption process by 80X while reducing 10X energy overhead.

6.0 Conclusion

In this dissertation, we have considered employing emerging non-volatile memory to enable reliable, efficient, and secure computing of energy harvesting powered IoT devices. With the ever-increasing power needs of IoT devices, researchers have been ambitiously developing substitutes of batteries due to their limited longevity, safety, and inconvenience of replacing. Energy harvesting (EH) is one of the most promising techniques to power devices for future generation IoT. While EH has significant potential for powering tremendous IoT devices, the instability of harvested power brings a new challenge to the embedded systems: intermittency. In order to address this challenge, researchers have been proposed to checkpoint the intermittent processor state to non-volatile memories (NVMs) before power outage and restore the processor state after power recovers. However, NVMs and backup are not sufficient for EH systems. Without extra hardware and software support, the program execution resumed from the last checkpoint might not execute correctly under some circumstances and causes inconsistency problem to the system. Besides, frequent checkpointing incurs significant overhead both in time and energy. Furthermore, there is no light-weight security mechanism for protecting the data on energy harvesting embedded system.

Towards these challenges, this thesis propose three techniques. First, consistency-aware checkpointing mechanism is proposed for intermittently powered embedded system with FRAM based scratchpad memory to avoid inconsistency errors generated from the inconsistency between volatile memory and non-volatile memory state. Second, hybrid cache architecture is proposed to take advantage of the high density of STT-RAM while mitigating the high latency of write in STT-RAM. Towards intermittently powered embedded system, the proposed hybrid cache is further improved to be checkpoint-aware to guarantee reliable checkpointing while maintaining a low checkpointing overhead from cache. Finally, to ensure the IoT device and protect the data, an energy efficient in-memory encryption implementation for AES algorithm is designed to quickly encrypt the data in non-volatile memory and protect the embedded system from physical attacks and on-line attacks.

Bibliography

[1] http://www.brookings.edu/blogs/techtank/posts/2015/06/8-future-of-iot-part-1.

[2] JEDEC DDR5 & NVDIMM-P Standards Under Development. `https://www.jedec.org/news/pressreleases/jedec-ddr5-nvdimm-p-standards-under-development`.

[3] Intel: First 3D XPoint SSDs will feature up to 6GB/s of bandwidth. `http://www.kitguru.net/components/memory/anton-shilov/intel-first-3d-xpoint-ssds-will-feature-up-to-6gbs-of-bandwidth`, 2015.

[4] S. Angizi, Z. He, N. Bagherzadeh, and D. Fan. Design and evaluation of a spintronic in-memory processing platform for non-volatile data encryption. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2017.

[5] K. Asari, Y. Mitsuyama, T. Onoye, I. Shirakawa, H. Hirano, T. Honda, T. Otsuki, T. Baba, and T. Meng. Feram circuit technology for system on a chip. In *Evolvable Hardware, 1999. Proceedings of the First NASA/DoD Workshop on*, pages 193–197, 1999.

[6] A. Awad, P. Manadhata, S. Haber, Y. Solihin, and W. Horne. Silent shredder: Zero-cost shredding for secure non-volatile main memory controllers. In *ACM SIGPLAN Notices*, pages 263–276, 2016.

[7] 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*, 35(12):1968–1980, 2016.

[8] D. Balsamo, A. S. Weddell, G. V. Merrett, B. M. Al-Hashimi, D. Brunelli, and L. Benini. Hibernus: Sustaining computation during intermittent supply for energy-harvesting systems. *IEEE Embedded Systems Letters*, 7(1):15–18, 2015.

[9] E. B. Barker and W. C. Barker. Guideline for using cryptographic standards in the federal government: Directives, mandates and policies. Technical report, 2016.

[10] N. Binkert, B. Beckmann, et al. The gem5 simulator. *ACM SIGARCH Computer Architecture News*, 39(2):1–7, 2011.

[11] K. Chen, S. Li, N. Muralimanohar, J. H. Ahn, J. B. Brockman, and N. P. Jouppi. CACTI-3DD: Architecture-level modeling for 3D die-stacked DRAM main memory. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 33–38, 2012.

[12] S. Chhabra and Y. Solihin. i-NVMM: a secure non-volatile main memory system with incremental encryption. In *Annual International Symposium on Computer Architecture (ISCA)*, pages 177–188, 2011.

[13] P. Chown. Advanced encryption standard ciphersuites for transport layer security. Technical report, 2002.

[14] P. Colp, J. Zhang, J. Gleeson, S. Suneja, E. De Lara, H. Raj, S. Saroiu, and A. Wolman. Protecting Data on Smartphones and Tablets from Memory Attacks. *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 177–189, 2015.

[15] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi. Nvsim: 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.

[16] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi. Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 7(31):994–1007, 2012.

[17] D. Fan, S. Angizi, and Z. He. In-memory computing with spintronic devices. In *VLSI (ISVLSI), 2017 IEEE Computer Society Annual Symposium on*, pages 683–688. IEEE, 2017.

[18] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R. Brown. Mibench: A free, commercially representative embedded benchmark suite. In *Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop on*, pages 3–14, Dec 2001.

[19] 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.

[20] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten. Lest We Remember : Cold Boot Attacks on Encryption Keys. In *USENIX Security Symposium*, pages 45–60, 2008.

[21] P. Hamalainen, T. Alho, M. Hannikainen, and T. D. Hamalainen. Design and implementation of low-area and low-power aes encryption hardware core. In *Proceedings of the 9th EUROMICRO Conference on Digital System Design*, pages 577–583, 2006.

[22] A. Hay, K. Strauss, T. Sherwood, G. H. Loh, and D. Burger. Preventing pcm banks from seizing too much power. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 186–195, 2011.

[23] J. L. Henning. Spec cpu2006 benchmark descriptions. *ACM SIGARCH Computer Architecture News*, 34(4):1–17, 2006.

[24] M. Henson and S. Taylor. Memory encryption: A survey of existing techniques. *ACM Computing Surveys*, 46(4):1–26, mar 2014.

[25] M. Hosomi, H. Yamagishi, T. Yamamoto, K. Bessho, Y. Higo, K. Yamane, H. Yamada, M. Shoji, H. Hachino, C. Fukumoto, H. Nagao, and H. Kano. A novel nonvolatile memory with spin torque transfer magnetization switching: spin-ram. In *IEEE International Electron Devices Meeting, 2005. IEDM Technical Digest.*, pages 459–462, Dec. 2005.

[26] F. Huang, D. Feng, Y. Hua, and W. Zhou. A wear-leveling-aware counter mode for data encryption in non-volatile memories. In *Proceedings of the Conference on Design, Automation & Test in Europe*, pages 910–913. European Design and Automation Association, 2017.

[27] T. Instruments. Msp430frxx microcontrollers. `http://www.ti.com/lsds/ti/microcontrollers_16-bit_32-bit/msp/ultra-low_power/msp430frxx_fram/overview.page`.

[28] H. Jayakumar, A. Raha, and V. Raghunathan. Quickrecall: A low overhead hw/sw approach for enabling computations across power cycles in transiently powered computers. In *VLSI Design and 2014 13th International Conference on Embedded Systems, 2014 27th International Conference on*, pages 330–335. IEEE, 2014.

[29] 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.

[30] e. a. K.-J. Lee. A 90nm 1.8v 512mb diode-switch pram with 266mb/s read throughput. *IEEE Journal of Solid-State Circuits*, 43(1):150–162, Jan. 2008.

[31] D.-H. Kang, J.-H. Lee, J. Kong, D. Ha, J. Yu, C. Um, J. Park, F. Yeung, J. Kim, W. Park, Y. Jeon, M. Lee, Y. Song, J. Oh, G. Jeong, and H. Jeong. Two-bit cell operation in diode-switch phase change memory cells with 90nm technology. In *2008 Symposium on VLSI Technology*, pages 98–99, June 2008.

[32] R. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. 1972.

[33] S. Kawai, A. Hosogane, S. Kuge, T. Abe, K. Hashimoto, T. Oishi, N. Tsuji, K. Sakakibara, and K. Noguchi. An 8kb eeprom-emulation dataflash module for automotive mcu. In *Digest of Technical Papers. IEEE International Solid-State Circuits Conference, 2008*, pages 508–632, 2008.

[34] 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.

[35] Q. A. Khan and S. J. Bang. Energy harvesting for self powered wearable health monitoring system. *Health*, pages 1–5, 2009.

[36] Y. Kim, V. Seshadri, D. Lee, J. Liu, and O. Mutlu. A case for exploiting subarray-level parallelism in dram. *Annual International Symposium on Computer Architecture (ISCA)*, pages 368–379, 2012.

[37] V. Leonov. Thermoelectric Energy Harvesting of Human Body Heat for Wearable Sensors. *IEEE Sensors Journal*, 13(6):2284–2291, 2013.

[38] 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 non-volatile SRAM in energy harvesting sensor nodes. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition, DATE*, pages 7–12, 2015.

[39] J. Li, P. Ndai, A. Goel, H. Liu, and K. Roy. An alternate design paradigm for robust spin-torque transfer magnetic ram (stt mram) from circuit/architecture perspective. In *ASP-DAC '09: Proceedings of the 2009 Asia and South Pacific Design Automation Conference*, pages 841–846, Yokohama, Japan, 2009.

[40] S. Li, C. Xu, Q. Zou, J. Zhao, Y. Lu, and Y. Xie. Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories. In *ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, 2016.

[41] D. Liu, X. Luo, Y. Li, Z. Shao, and Y. Guan. An energy-efficient encryption mechanism for nvm-based main memory in mobile systems. *Journal of Systems Architecture*, 76:47–57, 2017.

[42] H. Liu, M. Ferdman, J. Huh, and D. Burger. Cache bursts: A new approach for eliminating dead blocks and increasing cache efficiency. In *2008 41st IEEE/ACM International Symposium on Microarchitecture*, pages 222–233. IEEE, 2008.

[43] Y. Liu, Z. Li, H. Li, Y. Wang, X. Li, K. Ma, S. Li, M.-F. Chang, S. John, Y. Xie, J. Shu, and H. Yang. Ambient energy harvesting nonvolatile processors: from circuit to system. In *Proceedings of the 52nd Annual Design Automation Conference*, page 150, 2015.

[44] Y. Liu, F. Suy, Z. Wangy, 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.

[45] B. Lucia and B. Ransford. A simpler, safer programming and execution model for intermittent systems. In *ACM SIGPLAN Notices*, volume 50, pages 575–585. ACM, 2015.

[46] T. Luo, W. Zhang, B. He, and D. Maskell. A racetrack memory based in-memory booth multiplier for cryptography application. In *ASP-DAC*, pages 286–291. IEEE, jan 2016.

[47] 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.

[48] R. Maes, A. Van Herrewege, and I. Verbauwhede. Pufky: A fully functional puf-based cryptographic key generator. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 302–319, 2012.

[49] S. Mathew, F. Sheikh, A. Agarwal, M. Kounavis, S. Hsu, H. Kaul, M. Anders, and R. Krishnamurthy. 53Gbps native GF$(2^4)^2$ composite-field AES-encrypt/decrypt accelerator for content-protection in 45nm high-performance microprocessors. In *IEEE Symposium on VLSI Circuits (VLSIC)*, pages 169–170, 2010.

[50] D. McGrew and J. Viega. The galois/counter mode of operation (gcm). *submission to NIST Modes of Operation Process*, 20, 2004.

[51] A. Mirhoseini, E. Songhori, and F. Koushanfar. Automated checkpointing for enabling intensive applications on energy harvesting devices. In *2013 IEEE International Symposium on Low Power Electronics and Design (ISLPED)*, pages 27–32, 2013.

[52] A. Mirhoseini, E. M. Songhori, and F. Koushanfar. Automated checkpointing for enabling intensive applications on energy harvesting devices. In *Low Power Electronics and Design (ISLPED), 2013 IEEE International Symposium on*, pages 27–32. IEEE, 2013.

[53] 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.

[54] J. H. Oh. Full integration of highly manufacturable 512mb pram based on 90nm technology. In *International Electron Devices Meeting 2006*, pages 49–52, 2006.

[55] 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.

[56] 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.

[57] 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.

[58] 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.

[59] J. Paradiso and M. Feldmeier. A compact, wireless, self-powered pushbutton controller. *Ubicomp 2001: Ubiquitous Computing*, 2001.

[60] 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 *Proceedings of the 2008 Conference on Power Aware Computing and Systems*, pages 5–5, 2008.

[61] B. Ransford and B. Lucia. Nonvolatile memory is a broken time machine. In *Proceedings of the Workshop on Memory Systems Performance and Correctness*, MSPC, 2014.

[62] 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.

[63] U. Russo, D. Ielmini, and A. L. Lacaita. Analytical modeling of chalcogenide crystallization for PCM data-retention extrapolation. *IEEE transactions on electron devices*, 54(10):2769–2777, 2007.

[64] N. S. Shenck and J. A. Paradiso. Energy scavenging with shoe-mounted piezoelectrics. *Ieee Micro*, 21(3):30–42, 2001.

[65] X. Sheng, Y. Wang, Y. Liu, and H. Yang. Spac: A segment-based parallel compression for backup acceleration in nonvolatile processors. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, pages 865–868, 2013.

[66] H. Shiga, D. Takashima, S. Shiratake, K. Hoya, T. Miyakawa, R. Ogiwara, R. Fukuda, R. Takizawa, K. Hatsuda, F. Matsuoka, Y. Nagadomi, D. Hashimoto, H. Nishimura, T. Hioka, S. Doumae, S. Shimizu, M. Kawano, T. Taguchi, Y. Watanabe, S. Fujii, T. Ozaki, H. Kanaya, Y. Kumura, Y. Shimojo, Y. Yamada, Y. Minami, S. Shuto, K. Yamakawa, S. Yamazaki, I. Kunishima, T. Hamamoto, A. Nitayama, and T. Furuyama. A 1.6 gb/s ddr2 128 mb chain feram with scalable octal bitline and sensing schemes. *IEEE Journal of Solid-State Circuits*, 45(1):142–152, 2010.

[67] K. Suzuki and S. Swanson. The non-volatile memory technology database (nvmdb). *Department of Computer Science & Engineering*, 2015.

[68] S. Swami, J. Rakshit, and K. Mohanram. SECRET: smartly EnCRypted energy efficient non-volatile memories. In *ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, 2016.

[69] 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, 2008.

[70] K. Tsuchida, T. Inaba, K. Fujita, Y. Ueda, T. Shimizu, Y. Asao, T. Kajiyama, M. Iwayama, K. Sugiura, S. Ikegawa, et al. A 64Mb MRAM with clamped-reference and adequate-reference schemes. In *ISSCC*, pages 258–259, 2010.

[71] R. K. Venkatesan, S. Herr, and E. Rotenberg. Retention-aware placement in DRAM (RAPID): Software methods for quasi-non-volatile DRAM. In *The Twelfth International Symposium on High-Performance Computer Architecture (HPCA)*, pages 155–165, 2006.

[72] C. Wang, N. Chang, et al. Storage-less and converter-less maximum power point tracking of photovoltaic cells for a nonvolatile microprocessor. In *ASP-DAC*, pages 379–384, 2014.

[73] C. Wang, N. Chang, Y. Kim, S. Park, Y. Liu, H. G. Lee, R. Luo, and H. Yang. Storage-less and converter-less maximum power point tracking of photovoltaic cells for a nonvolatile microprocessor. In *Design Automation Conference (ASP-DAC), 2014 19th Asia and South Pacific*, pages 379–384, 2014.

[74] 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.

[75] J. Wang, Y. Liu, et al. 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.

[76] Y. Wang, H. Jia, et al. Register allocation for hybrid register architecture in non-volatile processors. In *ISCAS*, pages 1050–1053, 2014.

[77] Y. Wang, Y. Liu, et al. A 3us wake-up time nonvolatile processor based on ferroelectric flip-flops. In *ESSCIRC (ESSCIRC), 2012 Proceedings of the*, pages 149–152, 2012.

[78] Y. Wang, Y. Liu, et al. A 3us wake-up time nonvolatile processor based on ferroelectric flip-flops. In *ESSCIRC*, pages 149–152, Sept 2012.

[79] Y. Wang, L. Ni, C.-H. Chang, and H. Yu. DW-AES: A Domain-Wall Nanowire-Based AES for High Throughput and Energy-Efficient Data Encryption in Non-Volatile Memory. *IEEE Transactions on Information Forensics and Security*, 11(11):2426–2440, 2016.

[80] 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.

[81] Z. Wang, D. A. Jiménez, C. Xu, G. Sun, and Y. Xie. Adaptive placement and migration policy for an stt-ram-based hybrid cache. In *High Performance Computer Architecture (HPCA), 2014 IEEE 20th International Symposium on*, pages 13–24. IEEE, 2014.

[82] H.-S. P. Wong, H.-Y. Lee, S. Yu, Y.-S. Chen, Y. Wu, P.-S. Chen, B. Lee, F. T. Chen, and M.-J. Tsai. Metal–oxide rram. *Proceedings of the IEEE*, 100(6):1951–1970, 2012.

[83] 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.

[84] M. Xie, S. Li, A. O. Glova, J. Hu, Y. Wang, and Y. Xie. Aim: Fast and energy-efficient aes in-memory implementation for emerging non-volatile main memory. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 625–628. IEEE, 2018.

[85] M. Xie, S. Li, A. O. Glova, J. Hu, and Y. Xie. Securing emerging nonvolatile main memory with fast and energy-efficient aes in-memory implementation. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 26(11):2443–2455, 2018.

[86] M. Xie, C. Pan, et al. Non-volatile registers aware instruction selection for embedded systems. In *RTCSA*, pages 1–9, 2014.

[87] 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.

[88] M. Xie, C. Pan, Y. Zhang, J. Hu, Y. Liu, and C. J. Xue. A novel stt-ram-based hybrid cache for intermittently powered processors in iot devices. *IEEE Micro*, 39(1):24–32, 2018.

[89] 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.

[90] 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*, page 184. ACM, 2015.

[91] 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*, page 22. ACM, 2016.

[92] K. Yamaoka, S. Iwanari, Y. Murakuki, H. Hirano, M. Sakagami, T. Nakakuma, T. Miki, and Y. Gohou. A 0.9-v 1t1c sbt-based embedded nonvolatile feram with a reference voltage scheme and multilayer shielded bit-line structure. *Solid-State Circuits, IEEE Journal of*, 40(1):286–292, 2005.

[93] V. Young, P. J. Nair, and M. K. Qureshi. Deuce: Write-efficient encryption for non-volatile memories. In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 33–44, 2015.

[94] H. Zhang, C. Zhang, X. Zhang, G. Sun, and J. Shu. Pin tumbler lock: A shift based encryption mechanism for racetrack memory. In *Design Automation Conference (ASP-DAC), 2016 21st Asia and South Pacific*, pages 354–359. IEEE, 2016.

[95] X. Zhang, C. Zhang, G. Sun, J. Di, and T. Zhang. An efficient run-time encryption scheme for non-volatile main memory. In *Proceedings of the 2013 International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, page 24, 2013.

[96] Y. Zhang, L. Xu, K. Yang, Q. Dong, S. Jeloka, D. Blaauw, and D. Sylvester. Recryptor: A reconfigurable in-memory cryptographic cortex-m0 processor for iot. In *Symposium on VLSI Circuits*, pages C264–C265. IEEE, 2017.

[97] 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.

[98] 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.

[99] P. Zhou, B. Zhao, J. Yang, and Y. Zhang. A durable and energy efficient main memory using phase change memory technology. In *Annual International Symposium on Computer Architecture (ISCA)*, pages 14–23, 2009.

[100] P. Zhou, B. Zhao, J. Yang, and Y. Zhang. Energy reduction for stt-ram using early write termination. In *Computer-Aided Design - Digest of Technical Papers, 2009. ICCAD 2009. IEEE/ACM International Conference on*, pages 264–268, 2009.

[101] 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 *2011 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC),*, pages 334–336, Feb 2011.