Design of Non-Volatile Processors for Intermittent Computing

A Thesis Submitted

in Partial Fulfilment of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

by

Satya Jaswanth Badri

(2018CSZ0002)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY ROPAR

September 2023

Dedicated to my beloved parents, my brother, and those who, somewhere along the way, believed in me

Declaration of Originality

I hereby declare that the work that is being presented in the thesis entitled **Design of Non-Volatile Processors for Intermittent Computing** has been solely authored by me. It presents the result of my own independent investigation/research conducted during the time period from August 2018 to July 2023 under the supervision of Dr. Neeraj Goel, assistant professor, IIT Ropar, and Dr. Mukesh Saini, assistant professor, IIT Ropar. To the best of my knowledge, it is an original work, both in terms of research content and narrative, and has not been submitted or accepted elsewhere, in part or in full, for the award of any degree, diploma, fellowship, associateship, or similar title of any university or institution. Furthermore, due credit has been attributed to the relevant state-of-the-art collaborations (if any) with appropriate citations and acknowledgments in line with established ethical norms and practices. I also declare that any idea/ data/ fact/ source stated in my thesis has not been fabricated/ falsified/ misrepresented. All the principles of academic honesty and integrity have been followed. I fully understand that if the thesis is found to be unoriginal, fabricated, or plagiarized, the institute reserves the right to withdraw the thesis from its archive and revoke the associated Degree conferred. Additionally, the Institute also reserves the right to assess all sections of the society concerned with the matter for their information and necessary action (if any). If accepted, I hereby consent for my thesis to be available online in the institute's open access repository, inter-library loan, and the title & abstract to be made available to outside organizations.

B.S. Jaswanth

Signature

Name: Satya Jaswanth Badri
Entry Number: 2018CSZ0002
Program: Ph.D.
Department: Computer Science and Engineering
Indian Institute of Technology Ropar
Rupnagar, Punjab, 140001

Date: September 08, 2023

Acknowledgement

First, I express my sincere thanks to the Almighty God for blessing me with courage, patience, and sincerity of purpose throughout this research work.

I express my gratitude to Prof. Rajeev Ahuja, the honourable director Indian Institute of Technology Ropar, and the administration for providing all necessary administrative support and technical facilities required for carrying out the present research work. I also express my deep gratitude to the head and entire faculty and staff members of the Department of Computer Science and Engineering, Indian Institute of Technology Ropar, for their continuous support and encouragement. I wish to acknowledge the members of my Doctoral Committee and external experts, Dr. Hemangee K. Kapoor, Professor, Indian Institute of Technology Guwahati, and Dr. Georgios Karakonstantis, Professor, Queen's University Belfast, for their valuable comments and suggestions during my research work.

I express my deepest sense of gratitude to my esteemed supervisors, Dr. Neeraj Goel, assistant professor, Indian Institute of Technology Ropar, and Dr. Mukesh Saini, assistant professor, Indian Institute of Technology Ropar, for giving me the opportunity to work under their supervision and for their constructive and enthusiastic instructions during the fulfillment of my Ph.D. thesis. I am very lucky to have them as my supervisors/guides on this beautiful journey. Their academic excellence continues to be a source of inspiration, but beyond that, I am especially grateful for their limitless patience and fatherly support during tough times. Without their wise counsel and able guidance, it would have been impossible to complete the thesis work in this manner.

I would also like to thank Dr. T.V. Kalyan for motivating me during my Ph.D. work. I would also like to thank the entire faculty and staff members of the Department of Computer Science and Engineering for their motivation and support during my Ph.D. work. I strongly acknowledge the facilities and other technical help provided by MRIG Laboratory, Department of Computer Science and Engineering, and AWaDH, Indian Institute of Technology Ropar. My sincere thanks to all the editors and reviewers of our published work for their useful and valuable comments.

I express my sincere thanks to my senior lab members, Pratibha and Priyankar, for their support and to my friends, especially the MRIG lab and the CA-SIG research group in the CSE department of IIT Ropar. I express my sincere thanks to Bhaskar, Rajesh, Pranav, and Pankaj for their constant support during this wonderful journey.

Finally, I thank all those who believed in me somewhere along the way. I would like to thank my parents, my brother, and my friends for their sacrifices and for providing me with love and support when needed.

"You have a right to perform your prescribed duty, but you are not entitled to the results of your duty. Never consider yourself to be the cause of the results of your activities, and never be attached to not doing your duty."

–Bhagavad Gita

Certificate

This is to certify that the thesis entitled **Design of Non-Volatile Processors for Intermittent Computing**, submitted by **SatyaJaswanth Badri (2018csz0002)** for the award of the degree of **Doctor of Philosophy** of Indian Institute of Technology Ropar, is a record of bonafide research work carried out under my (our) guidance and supervision. To the best of my knowledge and belief, the work presented in this thesis is original and has not been submitted, either in part or fully, for the award of any other degree, diploma, fellowship, associate, or similar title from any university or institution.

In my (our) opinion, the thesis has reached the standard of meeting the requirements of the regulations related to the degree.

Neeraj Goel [illegible]

Signature of the Supervisors
Dr. Neeraj Goel and Dr. Mukesh Saini
Department of Computer Science and Engineering
Indian Institute of Technology Ropar
Rupnagar, Punjab 140001
Date: September 08, 2023

Lay Summary

Battery-less technology evolved to replace battery usage in space, deep mines, and other environments to reduce cost and pollution. The alternative and promising solution to replace battery-operated devices is energy harvesters, which help to collect energy from the environment to power up IoT devices. The collected energy is stored in a capacitor, and this energy is used for computations, so power failures may often occur in these IoT systems. We refer to this computation as an intermittent computation. Data loss is the major challenge in these intermittently powered IoT devices.

In the literature, non-volatile memory (NVM) based processors, i.e., Non-Volatile Processor (NVP), have been explored to save the system state during frequent power failures. NVPs are required for these intermittently powered IoT devices. We proposed three different architectures that could be combined to create an appropriate and efficient NVP for intermittent computing. We proposed three works that address these challenges while remaining efficient for intermittently powered embedded devices. We support intermittent computing in all three works.

Our first work proposes an efficient architecture that deploys NVM at the L1 and main memory. In the first work, we proposed efficient prediction and migration policies that perform better and consume less energy than existing and baseline architectures during frequent power failures. Our second work uses NVM at both the LLC and main memory levels. In the second work, we propose efficient cache management policies that perform better and consume less energy than the baseline architectures during frequent power failures. In the second work, we use fixed energy to backup all volatile contents to NVM. Our third work proposes an ILP-based memory mapping technique for modern MCUs such as MSP430FR6989 and MSP430F5529. In the third work, the proposed memory mapping technique achieves a lower EDP than existing and baseline architectures during stable and unstable power scenarios.

Abstract

Internet of Things (IoT) devices are rapidly expanding in many areas, including deep mines, space, industrial environments, and health monitoring systems. Most sensors and actuators are battery-powered, and these batteries have a finite lifespan. Maintenance and replacement of these many batteries will increase the maintenance cost of IoT systems and cause massive environmental damage. Energy Harvesting Devices (EHDs) are an alternative and promising solution to these battery-operated IoT devices.

The energy harvester stores enough energy in a capacitor to power the embedded device and compute the task. This type of computation is known as intermittent computing. Energy harvesters cannot provide continuous power to embedded devices, resulting in power failures in the IoT system. On conventional processors, all registers and caches are volatile. We require a processor that consists of Non-Volatile Memory (NVM) at either the cache or main memory level to store volatile contents during a power failure.

We must use NVM at either the cache or main memory levels to design an NVM-based processor. NVM caches degrade system performance and use more energy than volatile caches. Using a pure NVM at L1 reduces system performance by 45.93%, inspiring the idea of an efficient hybrid cache architecture. We propose efficient placement and migration policies for a hybrid cache architecture at L1 that uses volatile memory and NVM. The proposed architecture includes cache block placement and migration policies to reduce the number of writes to NVM. During a power failure, the backup strategy identifies and migrates critical blocks from the volatile memory region to NVM.

The energy stored in a capacitor is used as a backup during a power failure. Because the size of a capacitor is fixed and limited, the available energy in a capacitor is also limited and fixed. Thus, the capacitor energy cannot store the entire program state during frequent power failures. We propose an NVM-based architecture at the last-level cache (LLC) that ensures safe backup of volatile contents during a power failure under energy constraints. Using a proposed dirty block table (DBT) and a writeback queue (WBQ), the proposed architecture limits the number of dirty blocks in the L1 cache at any given time. We conducted experiments by varying the parameter sizes to help users make appropriate design decisions regarding their energy requirements.

Recent NVM-based microcontrollers, such as MSP430FR6989 and MSP430F5529, comprise hybrid main memory. Such devices have small volatile memory and large non-volatile memory. To make the system energy efficient, we need to use volatile memory efficiently. Therefore, we must select some portions of the application and map them to volatile memory or NVM. We propose an Integer Linear Programming (ILP) based memory mapping technique for intermittently powered IoT devices. Our proposed technique gives an optimal mapping choice that reduces the system's Energy-Delay Product (EDP). We validated our system using TI-based MSP430FR6989 and MSP430F5529 development boards.

Keywords: Non-Volatile Memory; Hybrid cache; Intermittent power; Limited Energy; MSP430FR6989; ILP; Memory mapping.

List of Publications

Journal

SJ Badri, M Saini, N Goel. Efficient Placement and Migration Policies for an STT-RAM based Hybrid L1 Cache for Intermittently Powered Systems. Springer Design Automation for Embedded Systems (DAEM)), 2023, doi: 10.1007/s10617-023-09272-w.

SJ Badri, M Saini, N Goel. An Efficient NVM based Architecture for Intermittent Computing under Energy Constraints. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2023, doi: 10.1109/TVLSI.2023.3266555.

SJ Badri, M Saini, N Goel. Mapi-Pro: An Energy Efficient Memory Mapping Technique for Intermittent Computing. arXiv preprint arXiv:2112.05410. 2023 Dec 10. **(under review in ACM Transactions on Architecture and Code Optimization)**

Book Chapter

SJ Badri, M Saini, N Goel. Design of Energy Harvesting based Hardware for IoT Applications. **Accepted to appear in** Taylor & Francis, Network Optimization in Intelligent IoT Applications – Principles and Challenges.

Posters

SJ Badri, M Saini, N Goel. Unforgettable: Designing a Non-Volatile Processor for Intermittently Powered Devices. Design, Automation, and Test in Europe Conference (DATE)" at Antwerp, Belgium, 2023.

List of Abbreviations and Notations

Abbreviations

BR	backup region
BR	backup region
BV	binary variable
CPU	central processing unit
CTF	charge trap flash
CCS	code-composer studio
CMOS	complementary metal-oxide semiconductor
ctpl	compute through power loss
CONF	confidence bit
DBT	dirty block table
DRAM	dynamic random access memory
EHD	energy harvesting device
EDP	energy-delay product
FRAM	ferroelectric random access memory
FLY-DRAM	flexible latency dynamic random access memory
FG	floating gate
GPU	graphics processing unit
HAW	heat-aware write
HCA	hybrid cache architecture
ILP	integer-linear programming
IDE	integrated development environment
IoT	internet of things
KB	kilo bytes
LLC	last level cache
LFW	least frequently written
LRU	least recently used
LRW	least recently written
L1	level 1
L2	level 2
L3	level 3
MTJ	magnetic tunnel junction
MRAM	magneto-resistive random access memory
MCU	microcontroller unit
ms	milli-second
MLC	multi-level cell
NVFF	non-volatile flip flop
NVM	non-volatile memory

NVP	non-volatile processor
PCM	phase-change memory
PF	power failure
PR	previous region
RAM	random access memory
RDE	read-disturbance error
RI	read-intensive
RIC	read-intensive counter
ROI	region of interest
Re-RAM	resistive random access memory
STT-RAM	spin-transfer torque random access memory
SPM	scratch-pad memory
SSD	solid state drive
SRAM	static random access memory
TI	texas instruments
TL-DRAM	tiered-latency dynamic random access memory
TM	transactional memory
V-NAND	vertical-NAND
VMRAM	vertical transport magneto-resistive random access memory
WL	wordline
WC	write counter
WBQ	writeback queue
WI	write-intensive
WIC	write-intensive counter

Notations

L	number of entries in the prediction table
rd_i	read to i^{th} cache block
wr_i	write to i^{th} cache block
$E_{overall}$	overall energy consumed to execute an application
E_{backup}	energy required to for backup procedure
$E_{restore}$	energy required to for restore procedure
N_{w_L1}	number of writes to L1 cache
e_{w_sttram}	energy per write for the STT-RAM
N_{w_main}	number of writes to main memory
e_{w_pcm}	energy per write for PCM RAM
e_{r_sttram}	energy per read for the STT-RAM
N_{r_main}	number of reads to main memory
e_{r_pcm}	energy per read for PCM RAM

N_{r_L1}	number of reads to L1 cache
η	backup efficiency
E_{normal}	energy required in normal execution
θ	energy efficiency
B_t	backup time
$N_{B/L1}$	number of blocks at L1
$E_{capacitor}$	fixed capacitor energy
K	fixed number of cache blocks
C	capacitance
E_{reg_file}	energy required to backup the register file to STT-RAM
$N_{dirty/L1}$	number of dirty blocks at L1
M	number of entries in DBT
N	number of entries in WBQ
V	valid bit
E_{system}	system's energy consumption
Num_cycles	number of CPU cycles
$NC_{Execute}$	number of cycles required for executing an application
$NC_{Intermittent}$	number of cycles required for executing an application during intermittent power supply
NC_{Backup}	number of cycles required for the backup operation
$NC_{Restore}$	number of cycles required for the restore operation
G	global variables in a program
r_i	number of reads to variable 'i'
w_i	number of writes to variable 'i'
E_{r_j}	energy required for each read to memory region 'j'
E_{w_j}	energy required for each write to memory region 'j'
NC_{v_i}	number of CPU cycles required for executing a variable v_i
$I_J(v_i)$	binary variable that refers a variable v_i is allocated to memory region 'j'
E_{global}	energy required to allocate global variables
N_f	number of functions in a program
r_{F_i}	number of reads to i^{th} function
w_{F_i}	number of writes to i^{th} function
$Sec_k(i)$	section information related to i^{th} function
$I_j(Sec_k(i))$	binary variable that refers of allocating sections of i^{th} function to memory region 'j'
EDP_{system}	EDP required for overall system
EDP_{global}	EDP required to allocate global variables
EDP_{func}	EDP required to allocate functions
$FRAM_B$	FRAM-based backup region
F	lowest lithographic dimensions for any memory technology

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter introduces intermittent computing, NVM-based architectures, and various related problems. We analyze various research questions, as well as the contribution that we have made to answer them. A review of the most recent research literature is used to identify research gaps. The objectives of this study are based on these research gaps.

1.1 Intermittent Computing

The Internet of Things (IoT) has created several fascinating applications consisting of intelligent sensors and systems. IoT may consist of billions of sensors and systems by the end of 2050 [2]. This prediction is exciting and promising, but deciding how to power these IoT devices is the main challenge [3]. Most of the IoT devices are battery-powered. In some areas, such as deep mines, space, and industrial environments, replacing batteries after installation is difficult and expensive.

Furthermore, the battery has a specific problem with its lifetime [3]. As a result, an alternative and promising solution is to replace the battery with energy harvesters. Energy-harvesting devices extract energy from their surroundings, such as light, vibration, radio, and many others [4]. The accumulated energy is used to power these IoT devices.

The unpredictable nature of energy harvesters causes voltage fluctuations or power failure. A voltage stabilizer or capacitor is a standard solution to the issue of voltage fluctuation [5]. However, in a conventional processor, power failures result in data loss. The data is lost because registers, caches, and main memories are designed using volatile memories, such as SRAM and DRAM [6]. Data lost include the application's program state and progress. The contents of the registers, cache, and main memory are all part of the program state. As a result, when the power failure occurs, some parts of the application have to re-execute, causing the execution progress to be slow and consuming extra energy. This type of computing is known as intermittent computing [7]. We discuss intermittent computing in detail in Section 1.2.

The solution is to store the application's program state at a precise restart point before a power failure. The question is, where should the program state be stored? Using Non-volatile memory (NVM), we can save the application's program state during a power failure. Recently, several new NVMs have been proposed, such as spin-transfer-torque RAM (STT-RAM) [8], phase-change memory (PCM) [9], resistive RAM (Re-RAM) [10], and ferroelectric RAM (FRAM) [11]. We need to change the memory model in conventional architectures to support intermittent computing, discussed in Section 1.2.1.

1.2 Application Scenario

When enough energy is harvested in a capacitor, and the energy harvester directly provides enough energy, the IoT application runs as usual, using the energy directly from the harvester source. When the harvester does not provide energy directly from the source, the IoT device must rely on the energy from the capacitor to perform essential tasks. Fig. 1.1 shows enough solar energy during the day for the IoT device to run the application without intervention around 01:00 PM. The IoT device must use the energy of the capacitor to complete important tasks before turning off in the evening, around 05:00 PM.

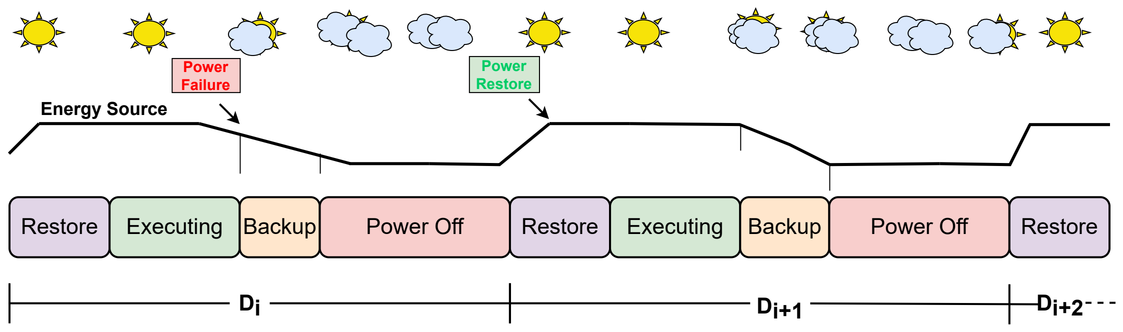


Figure 1.1: An Overview of Intermittent Computing for Solar-based Harvesting Systems

Because energy is not always available for harvesting, execution in such devices is intermittent, resulting in power failures in the IoT environment [7, 12, 13, 14]. Even when energy is readily available, it takes time to accumulate enough energy to perform valuable work. To design an intermittently aware architecture, new memory technologies and additional procedures must be incorporated into the execution and memory model of a conventional processor.

1.2.1 Memory Model for Intermittently Powered Devices

The hardware of an intermittently functioning device may include general-purpose computing units such as a CPU or a microcontroller unit (MCU), a group of sensors, and one or more radios for communication. Almost all of these devices use a volatile memory model. Fig. 1.2 (a) shows the conventional memory hierarchy, which includes the register file, caches, main memory, and secondary storage. As illustrated in Fig. 1.1, when no energy is available from the harvester source, such as at night time, the IoT device will shut down; during this time, the data stored in registers, caches, and main memory is lost. This thesis uses the term CPU to mean MCU, CPU, or any other processing unit.

There are two alternatives to keep volatile contents safe [15, 16]. One approach is to save all computed results and decisions made during the execution phase to secondary storage before the CPU enters the power-off state. The second approach is to restart the IoT application whenever the energy harvester provides enough energy to the CPU. Both alternatives are inefficient because backup/recovery to/from secondary storage and re-executing the same application take more time and energy. As a result, it is necessary

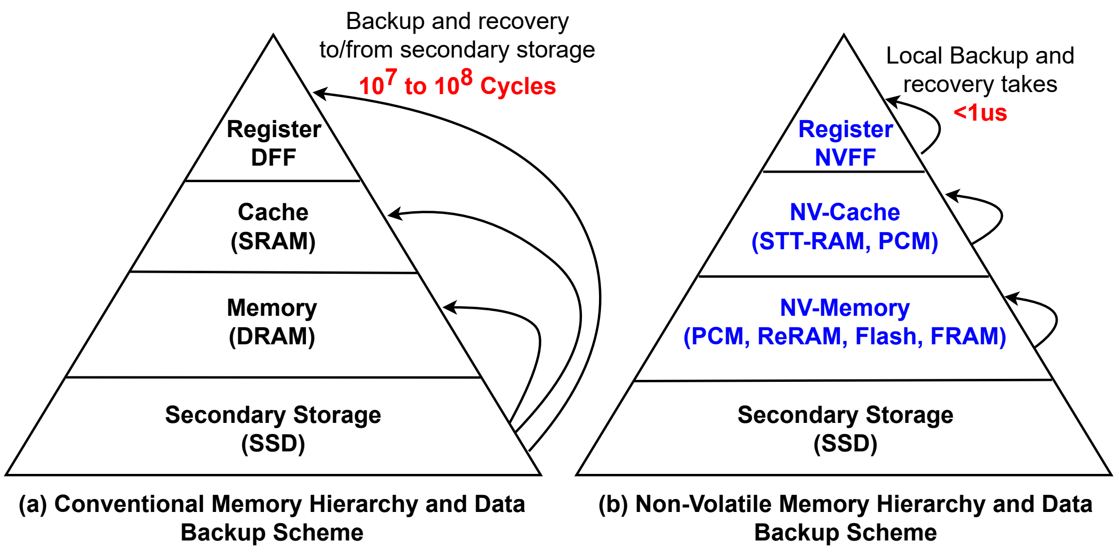

Backup and recovery to/from secondary storage
10^7 to 10^8 Cycles
Register DFF
Cache (SRAM)
Memory (DRAM)
Secondary Storage (SSD)
(a) Conventional Memory Hierarchy and Data Backup Scheme
Local Backup and recovery takes
<1us
Register NVFF
NV-Cache (STT-RAM, PCM)
NV-Memory (PCM, ReRAM, Flash, FRAM)
Secondary Storage (SSD)
(b) Non-Volatile Memory Hierarchy and Data Backup Scheme

Figure 1.2: Differences between Conventional and Non-Volatile Memory Models

to have a memory model that can store the memory contents when the energy source is unavailable. Thus, we require a new memory technology that can maintain a system state without consuming power, i.e., NVM based model.

NVM technology is being developed to overcome the disadvantages of volatile memory technologies. Flash, STT-RAM, PCM, ReRAM, and FRAM are examples of emerging NVM technologies. Due to their physical properties, NVMs have the potential to consume very little power while providing a significantly higher density than conventional memory technologies. A standard SRAM cell, for example, has a size of $125 - 200F^2$, and a PCM and Flash cell have sizes of $4-12F^2$ and $4-6F^2$, accordingly, where F refers to the lowest lithographic dimensions that range in a specific technology node. Due to their advantages, NVMs are becoming more common in real-time devices. Flash memory is, for example, used as a cache in Intel TurboMemory.

Table 1.1: Comparisons between different Traditional Memory Technologies for different Features

Property	**SRAM**	**DRAM**	**HDD**
Cell Size (F^2)	$120 - 200$	$6 - 12$	NA
Read Latency	~1 ns	~110 ns	5 ms
Write Latency	~1 ns	~110 ns	5 ms
Write Energy (J/bit)	$\sim 10^{-15}$	$\sim 10^{-14}$	$\sim 10^{-14}$
Leakage Power	High	Medium	Medium
Erase Latency	NA	NA	NA
Access Granularity (B)	64	64	512
Endurance	$> 10^{16}$	$> 10^{16}$	$> 10^{15}$
Standby Power	$0.6 - 1.2W$	Refresh Power	$1 - 2W$

However, these NVMs have some limitations. NVMs, for example, have higher latency and consume more energy than volatile memory technology. Write endurance is the property that determines how many writes a memory block can withstand before it becomes ineffective. NVMs have significantly lower write endurance than traditional memory technologies. Tables 1.1 and 1.2 provides detailed comparisons of various properties with various memory technologies. Access granularity is defined as the minimum size of data read/written in each access. Furthermore, they can store data for many years without requiring standby power under regular circumstances. As shown in Table 1.1, the common understanding of all NVM technologies is that the write latency/energy is greater than the read latency/energy [17, 18, 19, 20, 21].

Table 1.2: Comparisons between different Non-Volatile Memory Technologies for different Features

Property	**SLC Flash**	**PCM**	**STT-RAM**	**ReRAM**	**FRAM**
Cell Size (F^2)	4 – 6	4 – 12	6 – 50	4 – 10	12 – 15
Read Latency	$25\mu s$	50 ns	<10 ns	<10 ns	50 ns
Write Latency	$500\mu s$	500 ns	10 ns	<10 ns	50 ns
Write Energy (J/bit)	$\sim 10^{-9}$	$\sim 10^{-11}$	$\sim 10^{-13}$	$\sim 10^{-13}$	$\sim 10^{-12}$
Leakage Power	Low	Low	Low	Low	Low
Erase Latency	2 ms	NA	NA	NA	NA
Access Granularity (B)	4K	64	64	64	64
Endurance	$10^4 – 10^5$	$10^8 – 10^{15}$	$>10^{15}$	$10^8 – 10^{12}$	$10^{10} – 10^{12}$
Standby Power	0	0	0	0	0

In terms of characteristic features, STT-RAM outperforms all other NVM technologies. Table 1.1 shows that STT-RAM outperforms other NVMs in terms of write endurance, latency, and energy consumption. As a result, STT-RAM [22, 23, 8, 24, 25, 26] is a promising candidate for cache, main memory, and scratch-pad memory. However, because STT-RAM is more expensive than other NVMs, it is not suitable for use at the main memory level. PCM is the next better NVM technology after STT-RAM because its size and write endurance are better than other NVMs [9, 27, 28, 29, 30, 31]. As a result, PCM is a promising candidate for use in the cache and main memory. However, because STT-RAM has a longer lifespan than PCM, it is not suitable for use at the cache level. PCM is the better candidate for main memory. ReRAM and FRAM share some characteristics; both were promising candidates for main memory. ReRAM, on the other hand, has lower latency and energy consumption than FRAM. For example, the MSP430FR6989 is a recent TI-based microcontroller with 2KB SRAM and 128KB FRAM at the main memory level.

Incorporating NVMs at each level preserves the data in these IoT devices from frequent power failures, switching the traditional processor into a non-volatile processor (NVP). Fig. 1.2 (b) shows the memory hierarchy of non-volatile flip-flops, non-volatile caches (STT-RAM / PCM), and non-volatile main memory (PCM, ReRAM, and FRAM). Thus, using the non-volatile memory model instead of volatile memory helps in intermittent computing and reduces the time and energy required to backup/retrieve volatile contents.

1.2.1.1 Designing NVM-based Processor for Intermittently Powered IoT Devices

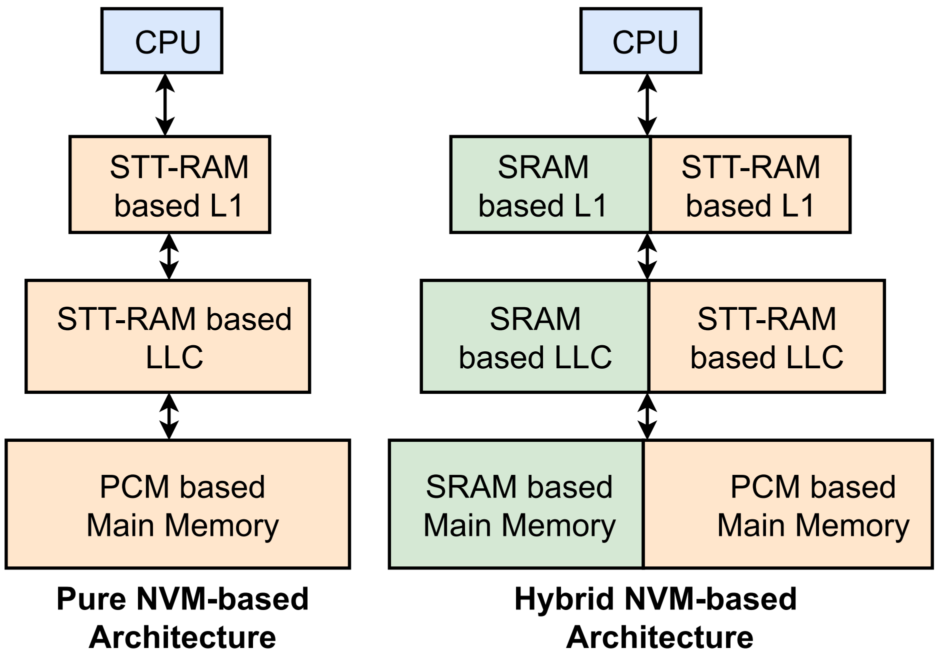


Figure 1.3: (a) Pure-NVM based architecture, STT-RAM at cache levels & PCM at main memory level and (b) Hybrid-NVM based architecture, SRAM+STT-RAM at cache levels & SRAM+PCM at main memory level

NVP is designed by replacing volatile memory with NVM at each level. Fig. 1.3 shows two distinct architectures that demonstrate the differences between the pure NVM architecture and the hybrid NVM architecture. There is pure NVM technology at each level in architecture-1, particularly STT-RAM at the L1 and last-level cache (LLC) levels and PCM at the main memory level. In architecture-2, hybrid NVM technology is used at each level, with SRAM + STT-RAM at the L1 and LLC levels and SRAM + PCM at the main memory level [32, 11, 33, 1, 34, 35].

The main advantage of hybrid architecture over pure NVM architecture is that it takes advantage of both SRAM and NVM, i.e., performance benefits from SRAM and non-volatility; and density benefits from NVM. Fig. 1.3 (b) shows a design that allows you to experiment with various combinations. As needed, the designer must select hybrid NVM and pure NVM architectures. The main disadvantage of using hybrid NVM over pure NVM architecture is that volatile contents in the hybrid architecture must be stored in NVM during frequent power failures, which increases backup time and energy.

There is a good possibility of making new NVM-based architectures, as shown in Fig. 1.4. Instead of pure NVM-based and hybrid-based architectures, we can selectively use NVM at the cache or main memory level. For instance, we can only use NVM at

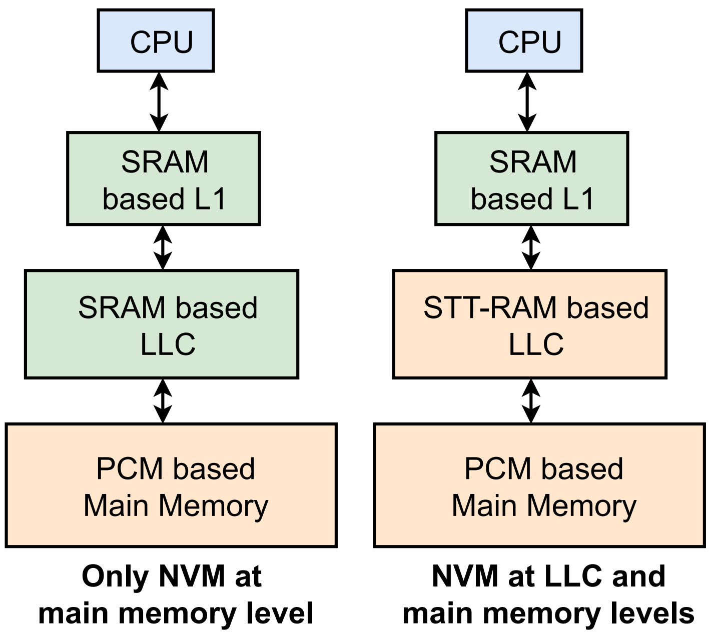


Figure 1.4: (a) NVM-based architecture only PCM at main memory level and (b) NVM-based architecture, STT-RAM at LLC & PCM at main memory levels

the main memory level, as shown in Fig. 1.4 (a), and in Fig. 1.4 (b), we use NVM at LLC and main memory levels. Thus, using NVM at different levels and utilizing NVM characteristics is user-defined and application-specific. Research has been going on all of these scenarios and possibilities [36, 37, 38, 32, 39, 40] mentioned in Fig. 1.3 and Fig. 1.4.

1.2.2 Execution Model for Intermittently Powered Devices

Despite several significant differences between the intermittent execution model and the conventional execution model, designers of today's intermittently powered devices use standard C-like embedded computing abstractions. Applications on intermittently powered devices run until the energy of the device has been drained. Once the application's energy is restored, it resumes execution from a certain point in its execution history, such as the start of the main() function or a safe point. The main difference between traditional and intermittent execution models is that a program that normally runs should run until completed. In contrast, a model of intermittent execution must complete the execution of the program despite multiple power interruptions. Various system components, such as languages, runtime behavior, and program semantics, need to be modified to create an intermittence-aware design.

Among all these changes, we have identified three significant changes that are required in the application execution flow for the intermittence-aware design.

Checkpoint(): When a checkpoint procedure is detected, all volatile content is copied to NVMs to maintain the system state. Existing literature in this area focuses on when and where checkpoints should be placed. Recent research suggests two ways to determine when to perform the checkpoint procedure.

1. Specially designed hardware is required to monitor the power source and capacitor. When it falls below a certain threshold, the system sends an interrupt that stops the application and starts the backup() procedure. Thus, checkpoints can occur at any time.

2. Instead of maintaining hardware to monitor energy requirements, existing solutions track changes in the application state. There are solutions that determine the variation at checkpoint time, either through hash comparisons or by comparing main memory word-by-word with the most recent checkpoint data, which are already in NVM. When there are many changes to initiate a backup() procedure, the system issues an interrupt.

Backup(): Whenever a backup() procedure is initiated, it copies the volatile contents to NVM, which means that it reads the contents from volatile memory technology and writes them to NVM technology.

Restore(): Whenever a restore() procedure is initiated, it copies the backed-up contents from NVM to volatile memory, which means that it reads the contents from NVM technology and writes them to volatile memory technology.

Adding additional procedures such as checkpoint(), backup(), and restore() to the conventional execution supports intermittent computing, which also completes Fig. 1.1. However, these additional procedures may incur additional costs. To reduce additional overhead in the intermittent execution model, efficient checkpoint, backup, and rollback policies are required.

1.3 Motivation

This section discusses observations that motivate us to propose new architectures and techniques. We also investigated research questions that motivated us to explore new techniques and architectures.

1.3.1 Challenges associated with NVM-based architectures to support intermittent computing

Let's explore some of the challenges associated with NVM architectures that support intermittent computing, which gives a way to identify some of the research questions in the next section.

- Introducing NVM at the cache or main memory level can degrade system performance and consume more energy due to their higher latency and energy compared to traditional memories such as SRAM and DRAM. Therefore, we must use NVM as efficiently as possible.

- If the application's system state is not completely backed up across all power failures, the application will re-run from the start, i.e., from the main() function [7, 12, 13].

- During a power failure, we must save the entire program state to NVM. If power comes back, we need to restore the state before power failure; otherwise, we could end up with inconsistent results.

- Even if an application has appropriate checkpoints and a system that maintains a consistent state between NVM and volatile memory, it may perform differently than expected. When the capacitor discharges and there is a power failure, the energy harvesting device is turned off for a certain period of time, and all peripherals and their system states are cleared. This behavior may violate the designer's assumptions about the atomicity of operations and the timeliness of data when compared to continuously powered devices.

  – **Atomicity:** Certain code regions must execute sequentially (with no checkpoints in the middle), ensuring the application runs correctly.

  – **Timeliness:** Some information loses value over time. Because the device may remain on for a prolonged period of time if the power goes out, placing checkpoints between the gathering and using data restricts its usefulness.

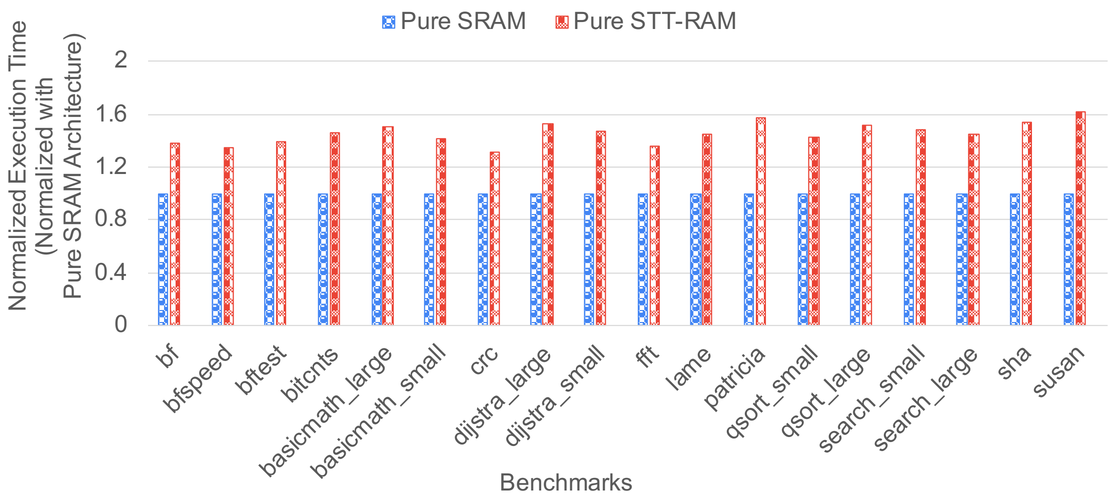
- Another important challenge is deciding where and when to place a checkpoint procedure. Placing unnecessary checkpoints degrades the system performance very badly and consumes more energy than in the standard scenario.

1.3.2 Research Questions

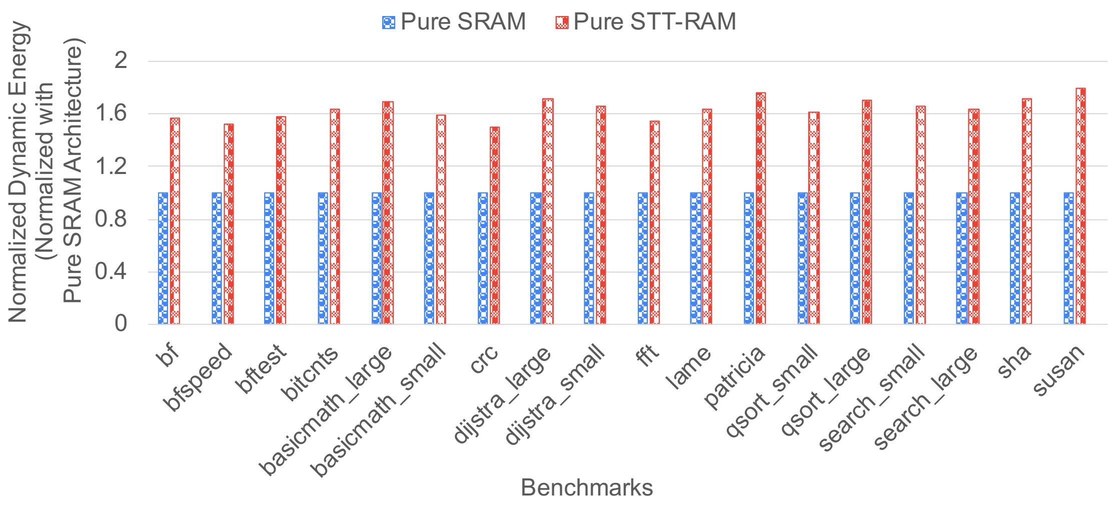
Question 1: Does a pure NVM-based architecture in the L1 cache provide more benefits during frequent power failures?
Analysis: Introducing STT-RAM as a cache can deteriorate system performance due to its long access time and high dynamic energy. We modeled two cache architectures in gem5 [41], pure SRAM cache (only SRAM at L-1) and pure STT-RAM cache (only STT-RAM at L-1), to compare their performance and energy consumption.

In Fig. 1.5, the performance and energy consumption of the cache architectures is normalized based on the pure SRAM cache architecture. Fig. 1.5 (a) shows that the STT-RAM cache architecture takes 45.93% more execution time than the pure SRAM cache architecture. Thus, a hybrid cache consisting of SRAM and STT-RAM cache at L1 will potentially be useful.

Question 2: Hybrid caches offer more benefits than pure NVM-based architectures during a power failure; how do we decide which cache blocks should be placed in which cache region?
Analysis: In the case of hybrid cache architectures, the movement between two cache regions was explored in the literature, i.e., migration-based policies for hybrid caches [42], [43], [44], [45], [46]. Moving cache blocks from one cache region to another cache region creates additional overhead, i.e., migration overhead. This overhead increases the



(a) Execution Time



(b) Dynamic Energy Consumption

Figure 1.5: Comparisons between Pure SRAM and Pure STT-RAM Architectures in terms of Execution time and Dynamic Energy Consumption

number of read and write operations to the NVM and requires additional cycles and energy, causing the system to be inefficient by consuming more energy and deteriorating the overall system performance. Thus, our observation is to reduce these additional migration overheads.

Contributions made to the use of NVM in L1: We introduce the following techniques and architectures to answer questions 1 and 2.

- We are convinced that the hybrid cache architecture benefits more than the pure NVM-based architecture in L1. Thus, we proposed a hybrid NVM-based architecture that consists of SRAM and STT-RAM at L1.
- We proposed efficient placement and migration policies to decide which cache blocks should be placed in which cache region.

Question 3: As mentioned in question 1, what design options will NVM have in LLCs?
Analysis: SRAM is used at L1 and LLC, and PCM is used in the main memory, as shown in Fig. 1.4 (a), i.e., traditional architecture. SRAM is used at L1, STT-RAM at LLC, and PCM is used at the main memory, as shown in Fig. 1.4 (b), i.e., baseline architecture. We analyze these two architectures under both stable and unstable power supply scenarios.

Under stable power supply: Both architectures shown in Fig. 1.4 (a) and Fig. 1.4 (b) give an equal number of writes to the PCM during regular operation. In architecture (Fig. 1.4 (b)), STT-RAM takes more cycles to execute than architecture-1 because the CPU must stall to complete each STT-RAM write. So, we implement the LLC cache so that the LLC gets fewer writes by passing the writes to PCM to hide the LLC latency. We observe that architecture (Fig. 1.4 (b)) takes 5.88% more execution time than architecture (Fig. 1.4 (b)) during stable power. Thus, the use of STT-RAM in the cache should have a minimal impact on overall system performance.

Under unstable power supply: During frequent power failures, a writeback of volatile contents is essential. In architecture (Fig. 1.4 (a)), the PCM gets more writes due to frequent power failures, which consumes more energy. In architecture (Fig. 1.4 (b)), PCM gets fewer writes because STT-RAM can save cache blocks during a power failure, which consumes less energy than in architecture (Fig. 1.4 (a)). We compared architecture (Fig. 1.4 (a)) and architecture (Fig. 1.4 (b)) in terms of energy and performance during frequent power failures. We used the same set of benchmarks used in section 5.3 and the experimental setup shown in Table 3.1. We observed that architecture (Fig. 1.4 (a)) takes 8.13% more execution time than architecture (Fig. 1.4 (b)) during power failures. On average, architecture (Fig. 1.4 (b)) saves energy of 0.07% for every power failure. If the number of power failures is 200, we save 9.04% of the system energy. Thus, NVM at LLC and main memory save energy during frequent power failures.

Question 4: What if we have a capacitor that can store only a fixed amount of energy? Is this fixed amount of energy enough to backup volatile contents during power failures?

Analysis: When using battery-less hardware, the device must be turned off when the harvested power is no longer available. To avoid sudden power failures and fluctuations, such devices accumulate energy in a capacitor that smoothens power availability and provides energy during power failures [5, 47, 48]. Thus, during a power failure, the energy stored in a capacitor is used to backup the processor state. The entire process state in volatile memory must be backed up to ensure correctness. Furthermore, cache lines store a copy of data elements present in memory; therefore, cache lines that are not modified need not be backed up. However, in the worst case, all cache lines could be dirty.

Since the capacitor energy storage capacity is limited and fixed, only fixed SRAM contents can be copied to NVM during a power failure. A sub-optimal solution is to constrain the cache size based on the energy available in a capacitor or design the capacitor to store the entire cache.

Contributions made to use NVM in LLC under certain energy constraints: To answer these questions, we introduce the following techniques and architectures.

- We are convinced that the use of pure NVM-based architecture benefits more than the hybrid cache architecture at LLC during frequent power failures. Thus, we proposed a pure NVM-based architecture at LLC.
- We proposed an efficient cache management and replacement policy that ensures that during a power failure, we can perform a safe backup using a fixed amount of energy.

Question 5: For a particular hybrid main memory-based architecture like MSP430FR6989, how do we choose appropriate sections of the program and map them to SRAM or FRAM regions? A major challenge is mapping the program's stack, code, and data sections to SRAM or FRAM.

Analysis: SRAM is 2KB, and FRAM is 128KB in FRAM-based MCU, MSP430FR6989. The first naive approach is to use the entire 128KB of FRAM in both stable and unstable power scenarios, resulting in longer execution cycles and higher energy consumption. Similarly, we have a second naive approach to use the entire 2KB SRAM for small applications (whichever fits within the SRAM size), which has advantages during regular operation. Unfortunately, it loses all 2KB SRAM data during a power failure and takes more time to backup 2KB contents to FRAM during a power failure. As shown in Fig. 1.6, for the FRAM-only configuration, we map the three sections to FRAM and the three sections to SRAM for the SRAM-only configuration.

We compared the FRAM-only configuration and the SRAM-only configuration in both stable and unstable power scenarios. FRAM-only configuration performs better during frequent power failures, whereas the SRAM-only configuration performs better during regular operations (without any power failures), as shown in Fig. 1.7. On the other hand, the FRAM-only configuration consumes 47.9% more energy than the SRAM-only

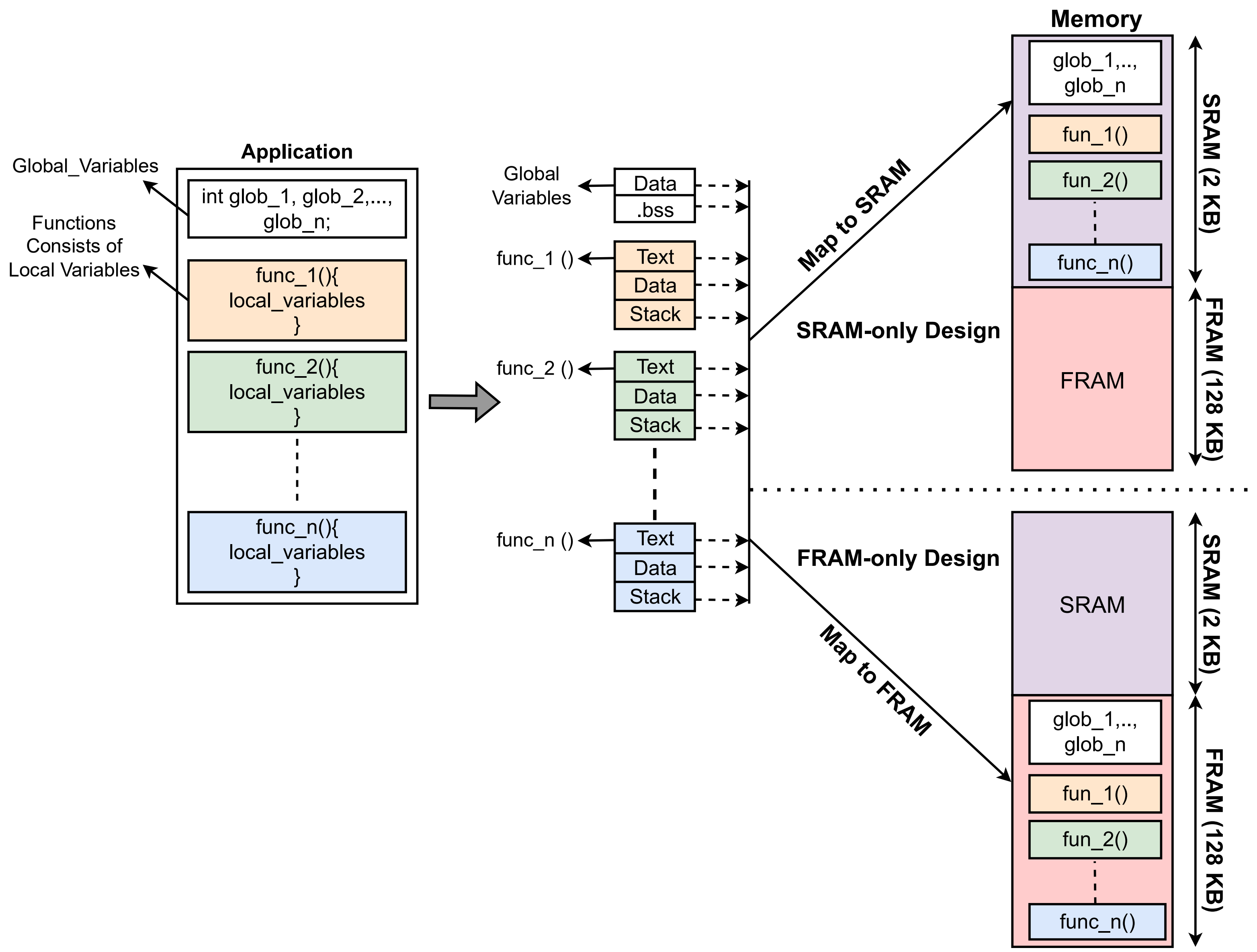


Figure 1.6: Overview of the FRAM-only configuration and SRAM-only configuration memory mappings in MSP430FR6989

configuration during stable power, as shown in Fig. 1.7 (a). On average, the SRAM-only configuration consumes 32.7% more energy than the FRAM-only configuration during unstable power, as shown in Fig. 1.7 (b). For large-size applications that cannot run using only SRAM, it requires FRAM as well. Thus, large applications consume more energy in the SRAM-only configuration during a stable power scenario.

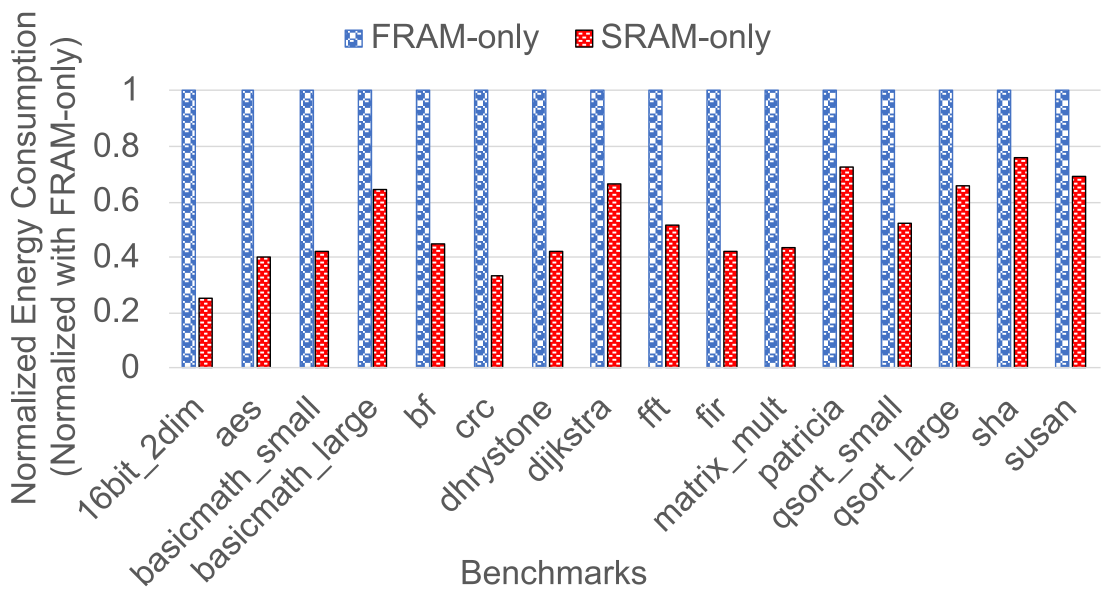
Table 1.3: Analysis of the Empirical Methods Used by Jayakumar et al. [1] for qsort_small Under Stable and Unstable Power Scenarios

Configuration	**Text**	**Data**	**Stack**	$Energy_{stable}(mJ)$	$Energy_{unstable}(mJ)$
1. {SSS}	SRAM	SRAM	SRAM	16.70	79.56
2. {SSF}	SRAM	SRAM	FRAM	21.08	66.34
3. {SFS}	SRAM	FRAM	SRAM	28.75	33.79
4. {SFF}	SRAM	FRAM	FRAM	35.97	52.10
5. {FSS}	FRAM	SRAM	SRAM	39.48	68.24
6. {FSF}	FRAM	SRAM	FRAM	57.64	54.75
7. {FFS}	FRAM	FRAM	SRAM	64.14	45.83
8. {FFF}	FRAM	FRAM	FRAM	92.09	36.07

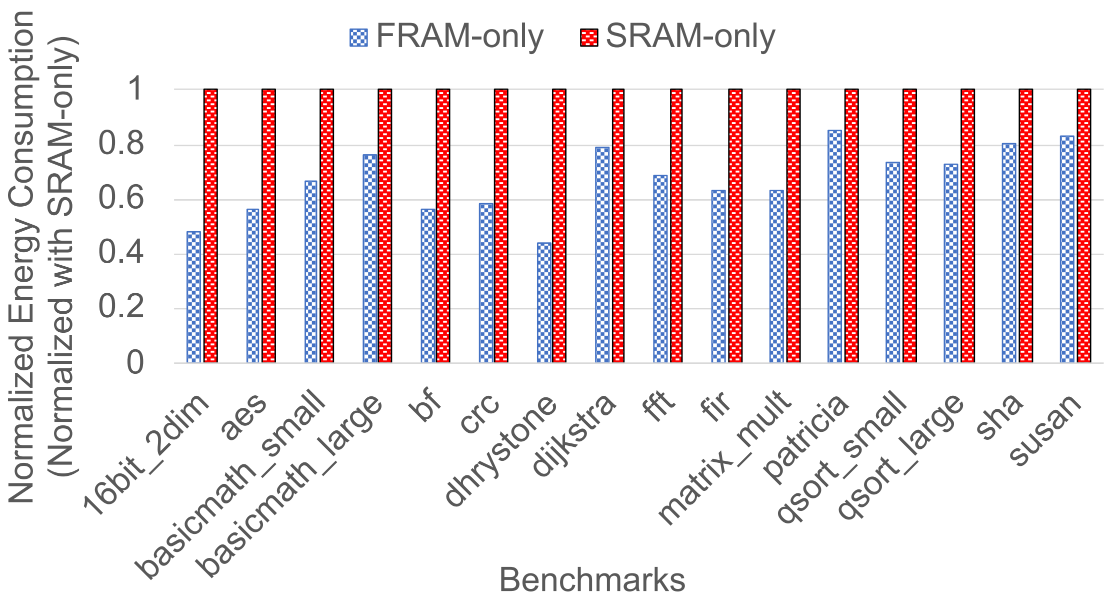
These two designs motivate us to propose a hybrid memory design that effectively uses both SRAM and FRAM. We also found that the SRAM-only configuration is ineffective for larger applications. As a result, we had to use a hybrid memory and figure out how and where to place the sections. To the best of our knowledge, only one work explored the memory mapping issue for these MCUs [1]. We analyze mapping decisions using their empirical model. Jayakumar et al. [1] calculated the energy consumption values for each configuration. Jayakumar et al. empirical method suggested that the sections be assigned to either SRAM or FRAM based on energy values.

We introduced an empirical method that was proposed by the Jayakumar et al. is as follows. Jayakumar et al. empirical method consider functions as the basic unit. They explored all configurations and calculated the energy values, as shown in Table 1.3. Jayakumar et al. empirical method have eight configurations because they have two memory regions (SRAM or FRAM) and need to map three sections (stack, data, text). Using the Jayakumar et al. empirical method, we calculated the energy values for the qsort_small application. For instance, the {SSS} configuration performs better during a stable power supply, and during a power failure, {SFS} consumes less energy than all other configurations. As a result, Jayakumar et al. empirical method allocate text and stack sections to SRAM and data sections to FRAM.

We observed that this empirical method becomes ineffective as the number of configurations increases. Jayakumar et al. empirical method considered all global variables, arrays, and constants as data sections. Instead, why can't we map each global variable or array to either SRAM or FRAM? This increases the number of configurations, and calculating/tracking energy values is challenging. Our design space grows enormously and makes our mapping problem challenging.



(a) Under Stable Power



(b) Under Unstable Power

Figure 1.7: Comparison between FRAM-only and SRAM-only configurations under Stable and Unstable Power

Contributions made to use NVM at main memory level in these MSP430FR6989 devices:

- We proposed the Integer-Linear Programming (ILP) based memory mapping technique for intermittently powered IoT devices.
- We incorporated energy harvesting scenarios into the ILP model so that the frequency of power failures is considered as an input for our ILP model.

- We formulated the memory mapping problem to cover all the possible design choices. We also formulated our problem in such a way that it supports large-size applications.
- We proposed a framework that efficiently consumes low energy during regular operation and frequent power failures. Our proposed framework supports intermittent computing.

1.4 Research Gaps

As per the challenges and various research questions mentioned above and reviewed literature, the following research gaps are observed:

- **At L1:** We found very few works on NVM-based L1 caches in the literature. We analyzed the NVM-based architectures at L1 with different NVM technologies.

  – **Prediction and Migration Policies:** We observed that when we used pure NVM at L1, the system's performance was degraded and consumed more energy. This behavior motivated us to explore hybrid caches to benefit from SRAM and NVM. However, we need to answer specific questions about deciding which cache blocks should be placed in which cache region, either in SRAM or NVM. To answer these questions, we have to explore efficient prediction and migration policies.

    In existing architectures, Xie et al. [32] also introduced a similar hybrid cache architecture that consists of STT-RAM in the L1 cache. Some of the observations that we reported from the work of Xie et al. are listed below.

    1. Xie et al. used a pattern sampler for the prediction table, which does not collect all of the application's details, resulting in inaccurate predictions. When the prediction information is incomplete, placement and migration policies cannot provide accurate predictions. Inaccurate predictions increase the number of reads and write operations, consuming more execution time and energy.
    2. Xie et al. used a standard LRU replacement policy to identify the cache block for eviction. What if the evicted block turns out to be a write-intensive block the next time? The used replacement policy may result in unnecessary writes to NVM and consumes more energy.

    All of the above challenges and observations motivated us to explore efficient prediction and migration policies for an NVM-based hybrid cache architecture.

- **Under Energy Constraints:** During a power failure, what if we have a fixed amount of energy to do a backup? This question motivated us to study relevant research. Unfortunately, there is no work associated with this study that inspires us to explore efficient NVM-based architectures and cache management policies to answer this question and also to support NVM-based architectures under these energy constraints.

- **Checkpointing Overhead:** Checkpoint is a procedure that invokes a backup procedure that copies the volatile state of the system to the NVM. We noticed that unnecessary checkpoints increase the number of writes to the NVM that consume more energy. Therefore, in stable and unstable power scenarios, we need to decide when and where to checkpoint. There are many checkpoint techniques in the literature. However, we still need intelligent backup/restore policies that reduce backup/restore content size during a checkpoint or power failure.
- **Memory Partitioning in MSP430-based MCUs:** MSP430-based MCUs such as MSP430FR6989 consist of SRAM and FRAM-based main memory. As we already mentioned, for any hybrid memory architecture, it is necessary to decide which part of the application goes to which memory region, whether SRAM or FRAM; essentially, this is a memory mapping problem. There is only one work on hybrid main memory architectures that supports recent MCUs, which has given us enough motivation to investigate efficient memory mapping techniques for hybrid main memory architectures.

1.5 Research Aim and Objectives

The main aim of the thesis is the design of non-volatile processors for intermittent computing scenarios. In addition, the objectives of the thesis are given as follows:

- **Objective 1:** To design an NVM-based architecture at L1 cache for intermittent computing.
- **Objective 2:** To design an NVM-based architecture at Last-Level cache (LLC) for intermittent computing under energy constraints.
- **Objective 3:** To design an NVM-based architecture at main memory for intermittent computing.

1.6 Thesis Organization

The research work provided in this study is on designing non-volatile processors for intermittent computing. Chapter-wise organization of this study is summarized below.

Chapter 1 gives an introduction to non-volatile memory (NVM), its application, and intermittent computing. This chapter also discusses the various challenges associated with NVMs and devices that are intermittently powered. The research gaps and objectives of this study have been summarized hereafter.

Chapter 2 gives a detailed review of the literature on different memory technologies, the literature on NVM-based architectures, the literature on NVM-based architectures that support intermittent computing, and different checkpointing techniques have been discussed.

Chapter 3 focuses on designing an NVM-based hybrid cache architecture at L1 that supports intermittent computing. Efficient prediction and migration policies have been introduced and thoroughly discussed in order to benefit from the benefits of both SRAM and STT-RAM. The proposed architecture is evaluated and compared to existing and baseline architectures under both stable and unstable power scenarios.

Chapter 4 focuses on designing an NVM-based hybrid cache architecture at LLC that supports intermittent computing. Efficient cache management policies have been introduced and thoroughly discussed in order to use fixed energy for backup operations during frequent power failures. The proposed architecture is evaluated and compared to existing and baseline architectures under both stable and unstable power scenarios.

Chapter 5 focuses on designing an efficient memory mapping technique for a recent microcontroller that consists of NVM-based hybrid main memory. ILP-based memory mapping techniques have been introduced and discussed in detail to get benefits from both SRAM and FRAM. The proposed framework and techniques are evaluated under stable and unstable power scenarios and compared with the existing and baseline frameworks.

Chapter 6 finally presents conclusions and possible directions for further research.

Chapter 2

Literature Review

This chapter gives a detailed literature review on different memory technologies; the state-of-the-art works on NVM-based architectures, the state-of-the-art works on NVM-based architectures that support intermittent computing, and different checkpointing techniques have been discussed. This chapter begins with recent advancements in various memory technologies in Section 2.1, followed by related works on NVM-based architecture described in Section 2.2. We present the detailed literature survey on developments in intermittent aware designs in Section 2.3. Recent advances in real-time MCUs are described in Section 2.4. It is followed by detailed state-of-the-art works on memory mapping techniques described in Section 2.5.

2.1 Developments in Memory Technologies

Static Random Access Memory (SRAM) and Dynamic Random Access Memory (DRAM) are used to design registers, caches, and main memory for conventional processors. Recent advances in NVM technologies include Spin-Transfer Torque RAM (STT-RAM) [22], Phase Change Memory (PCM) [9], NAND Flash [49] and Ferroelectric RAM (FRAM) [11]. These NVM technologies motivated researchers for their appealing characteristics, such as non-volatility, low cost, and high density. NVM is used to design flip-flops [50, 51, 52]. S. Thirumala et al. [50] proposed reconfigurable ferroelectric transistors to design energy-efficient intermittent devices. NVM is used to design the L1 cache [32], the last-level cache (LLC) [53, 54, 55], and the main memory [11, 9] in the literature. Many studies have used NVMs to build even hybrid memories [22, 23, 46, 32], saving significant energy when configured and used correctly.

Overview of SRAM Developments (Table 2.1): Mittal et al. [56] use a typical SRAM cell with a four-transistor (4T). Rabaey et al. and Majumdar et al. [13, 57] designed 5T cells with two and four PMOS and NMOS transistors, respectively. Zhang et al. [58] presented a machine learning classifier that achieves computation performance in a typical 6T SRAM array. 8T SRAM cell has been introduced [59, 60, 61] to hold data from the 6T and 7T SRAM cells. Vaknin et al. [41] designed a DF-SRAM that utilizes less power and relatively low voltage. Aggregate power improves with 14. 5% when combined with traditional 6T SRAM cells [62, 63, 64]. Akerman et al. [65] introduced SRAM-based ternary content addressable memory (TCAM). Kitagata et al. [66] designed a technique for NV-SRAM that uses magnetic tunnel junctions (MTJs) to reduce current leakage and thus improve energy efficiency.

Table 2.1: Overview of SRAM Developments

Classification	**References**
Read/Write Performance	[57, 32, 58, 61, 67, 68, 64], [69, 70, 71, 72, 73]
Power Optimizations	[56, 12, 13, 41, 74, 68]
Read/Write Security	[59, 60, 62, 63]
NVM using SRAM	[65, 66, 75]
Cache Optimizations	[76]

Overview of DRAM Developments (Table 2.2): Seshadri et al. [77] propose a novel DRAM structure that internally achieves bulk data copy and startup processes at a low hardware cost. Advani et al. [78] used REVA, a refresh scheme that usually refreshes merely the imperative region of interest (ROI). Choi et al. [79] presented a method to automatically synthesize the proportional circuit of the dielectric relaxation (DR) trademark in DRAM. Kim et al. [80] discussed the issues, fabrication technologies, current development, and imminent nature of DRAM materials. Lee et al. [81] describe Tiered-Latency DRAM (TL-DRAM). Chang et al. [82] introduced the Flexible Latency DRAM (FLY-DRAM), which characterizes the fluctuation in latency present in each DRAM module. Lee et al. [83] proposed Adaptive Latency DRAM, which has designed a mechanism to identify and safely reduce the timing margin to speed up DRAM accesses.

Table 2.2: Overview of DRAM Developments

Classification	**References**
Latency Improvements	[80, 81, 83, 82, 84, 85]
Read/Write Performance	[86, 87, 88]
DRAM Security	[89, 90, 91, 92]
Power Optimizations	[79, 93, 94, 95]
Bank Optimizations	[77, 96, 97]
DRAM Refresh	[98, 78]
Density Improvements	[99, 100, 101]

Background on Spin Transfer Torque RAM (STT-RAM:) STT-RAM is a new kind of magnetoresistive RAM (MRAM). STT-RAM cell uses magnetic tunnel junction (MTJ) for the data storage, as shown in Fig. 2.1 (a), (b) and (c). An MTJ consists of two ferromagnetic films divided by an oxide barrier layer (e.g., MgO). One of the films has a fixed magnetic orientation, known as a fixed layer, and the other film's magnetic orientation can be altered, known as a free layer. When the free and fixed layer directions are in parallel, MTJ signifies 0 (low state). Whenever two layers were in anti-parallel directions, MTJ signifies 1 (high state). Idle energy is required to read the MTJ state, while enormous energy is used to flip the magnetic state causing high write energy consumption and longer write latency. STT-RAM can retain its stored data for at least ten years by using MTJ, where MTJ has been modeled for the highest operating temperatures, so MTJ cannot change its polarity stored in the junction. Hence STT-RAM stores data indefinitely and doesn't need a periodic refresh.

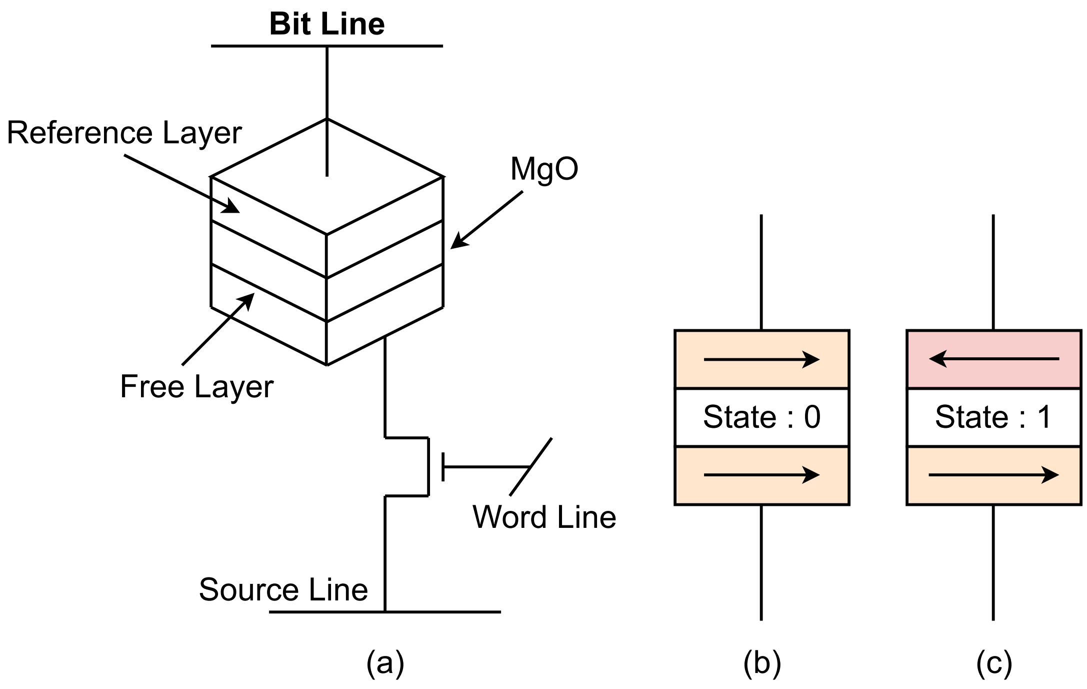


Figure 2.1: (a) Overview of STT-RAM cell, (b) Representing 0 state, and (c) Representing 1 state

Overview of STT-RAM Developments (Table 2.3): Zhu et al. and Lepak et al. [24, 25] described STT-RAM, an innovative version of MRAM. Takemura et al. [26] discussed the challenges faced by STT-RAM due to the high write energy. Kawahara et al. [102] described NV-RAM, which reduces power utilization by introducing a method of on/off processing. Bell et al. [103] deployed a complex free layer in an STT-RAM cell for ultra-high-thickness memory. Choi et al. [104] use a cache management strategy to reduce unequal write requests to STT-RAM to overcome the long-latency issue. Mittal [105] deployed SHIELD in STT-RAM LLCs to mitigate read-disturbance error (RDE). The works [106, 107] used a Vertical Transport MRAM (VMRAM), where current passes through the vertical column to alter magnetic orientation, thus reducing write disturbances.

Table 2.3: Overview of STT-RAM Developments

Classification	**References**
MTJ Optimizations	[108, 109, 110, 111, 106, 107, 112, 113, 24, 25, 114, 102]
Layers Optimization	[76, 115, 103, 116]
Energy Improvements	[26, 117, 118, 119, 120]
Usage as Cache	[104, 121, 105]

Overview of NAND Flash Developments (Table 2.4): Kang et al. [122] presented a 48-word line (WL) loaded 256-Gb V-NAND flash memory with a 3b multi-level cell (MLC). Park et al. [123] presented a genuine 3D, 128 Gb, 2-bit/cell vertical-NAND (V-NAND) Flash. The chip achieves 50 MB/s write throughput and has an endurance of 3K for conventional embedded applications [124]. Tanaka et al. [125] dealt with the NAND scaling problem with a Floating Gate (FG) of 3D NAND and a charge trap flash (CTF) of 3D NAND. Parat et al. [126] presented a 3b/cell NAND flash memory with a 3D FG mechanism to limit the scaling problem. Yoon et al. [127] describe the improvements in V-NAND flash memory technology. Park et al. [128] demonstrate Solid State Drives (SSDs).

Table 2.4: Overview of NAND Flash Developments

Classification	**References**
Performance Improvements	[124, 129, 122, 130, 131]
Scalability	[132, 125, 123]
SSD Compatibility	[126, 127]

Overview of PCM-RAM Developments (Table 2.5): The PCM architecture has a phase change material and a small electrode called a heater [27, 28, 29, 30, 31]. Park et al. [133] implemented sub-20 nm PCM technology with the TCAD simulator and a heat-aware write method (HAW) to regulate write current and voltage. Reduce energy consumption by about 36%. [134] uses a 512Mb diode switch PCM-RAM on a 90nm CMOS.

Table 2.5: Overview of PCM-RAM Developments

Classification	**References**
Performance Improvements	[27, 134, 135]
Energy Improvements	[136, 133]
Density Improvements	[28, 29, 30, 31]

Overview of Re-RAM Developments (Table 2.6): Chi et al. [137, 138] adopted ReRAM as an alternative to DRAM-based main memory. However, PCM shows better qualities than ReRAM, and it is promising because of its higher density, lower write energy, and shorter read latency. Xu et al. [139] deploy a multi-directional write mechanism to reduce hardware overhead. Beigi et al. [140] used a 3D-loaded ReRAM to achieve an adaptable memory framework design.

Table 2.6: Overview of Re-RAM Developments

Classification	**References**
Performance Improvements	[139, 141, 142, 138, 143]
Material Optimizations	[144, 145, 146, 147, 148]
Density Improvements	[137, 140, 149]

2.2 Developments in NVM-based Architectures

This section reviews the related work in hybrid cache architectures (HCAs) and NVM for last-level caches and L1 caches.

2.2.1 NVM for LLC

STT-RAM offers better features than the existing NVM technologies [22, 23, 8]. To use STT-RAM at LLC, we have two possible and distinct preferences. First, we replace the entire SRAM cache with STT-RAM at LLC. Second, we use HCA (SRAM + STT-RAM) at LLC, where this design takes advantage of both SRAM and STT-RAM.

Wu et al. [150] modeled a 3-level cache architecture by replacing SRAM with STT-RAM in L3 and using SRAM in L1 and L2. This architecture achieved an improvement in instructions per cycle (IPC) of approximately 4%, and compared to the traditional 3-level SRAM cache design, Wu et al. achieved a reduction of 63% in power consumption.

Usually, for hybrid caches, block placement and movement between caches are the main challenges. Classifying cache blocks based on write frequencies [151] and write access behavior [37] [38] [40] [13] helps decide where to place the respective cache block. Many HCA architectures use table-based prediction techniques to predict and place the cache block in an appropriate cache region and migrate from one cache to another [40]. The challenges in L1 HCA are different from those of LLC. In LLC, input traffic is due to L1 / L2, while read/write requests in L1 are due to load/store instructions.

2.2.2 HCA for L1 caches

The write access latency of STT-RAM is higher than the SRAM, which creates the primary limiting factor of using STT-RAM at the L1 cache. For this concern, there are two possible alternatives. First, relax the nonvolatility of STT-RAM to reduce the overall STT-RAM's write access latency [36], [39]. Relaxing the nonvolatility of STT-RAM is achievable by reducing the MTJ planar area and the MTJ switching current [152]. Second, reduce the STT-RAM's write latency and energy consumption by limiting the number of writes to STT-RAM. Usually, the number of reads and write operations in the L1 cache is more than in the LLC.

Xie et al. [32] introduce an HCA that consists of STT-RAM in the L1 cache. During power failures, the program state is backed up from the SRAM cache to the STT-RAM cache. Xie et al. use an access pattern-based predictor that predicts block behavior. Based on the prediction, Xie et al. placed the cache block in the respective cache region. During an eviction or on a wrong prediction, Xie et al. propose a migration policy that migrates a cache block from one cache region to another. Whenever power comes back, Xie et al. restore the cache contents from the STT-RAM cache to the SRAM cache.

2.2.3 Reducing Writes to NVM in a Hybrid Architectures

Many researchers are working to reduce the number of NVM writes at the cache or main memory. Choi et al. [46] proposed a way allocation scheme to reduce write counts to NVM in their hybrid LLC. Lee et al. [153] introduced PCM buffers to overcome the overheads, i.e., write latency and energy. Qureshi et al. [154] proposed a write cancelation and write pausing technique to give more priority to read requests than write requests. Hybrid main memory architectures [155], [156] have been introduced to efficiently use DRAM and PCM to reduce write latency and energy.

2.3 Developments in Intermittent Aware Designs

To develop an intermittent aware design, we should also change the execution model of a conventional processor by incorporating additional backup/restore procedures [7]. We require an efficient backup/restore procedure that restores and backups volatile contents during power failures. The size of volatile contents determines the energy required to backup/restore during a power failure [1]. We may get inconsistent results if we only have a small amount of available energy in our capacitor, which is insufficient to cover all volatile contents. As a result, we must reduce backup/restore overhead during frequent power failures.

Many researchers are working to reduce NVP backup and restore overheads. Recent checkpointing techniques [157, 158, 159, 50] have also been proposed to reduce the size of volatile contents that must be backup/restored during frequent power failures. Priyanka et al. [160] give an overview of recent checkpointing techniques. In situ checkpointing has gained popularity in recent times [50, 161], using unified NVM architectures. Lee et al. [10] proposed an adaptive NVP that prioritizes data retention to reduce the frequency of backup/restore operations. The number of power failures can be reduced by adjusting the voltage and frequency [162], [163]. Rather than reducing the number of backups and restore operations, researchers [164] focused on reducing the size of the backup contents. Architectures [165], [166] based on the comparison and compression strategies are proposed to reduce the number of bits/contents to be stored in the non-volatile flip flop (NVFF)-based NVP, which reduces the dynamic energy consumption.

Checkponting Techniques for Intermittent power devices: NVM-based NVPs [167], [168], [169] are proposed by storing the contents of the registers, volatile on-chip data to the non-volatile registers, and non-volatile memories, respectively. Whenever the power comes back, the system uses data from the NVM region to continue and complete the application execution.

Checkpoint-based approaches for HCA are the other alternative for supporting intermittently powered IoT devices. In these checkpoint-based approaches, volatile data is checkpointed to NVM at regular intervals to store the application program state [170]. Mementos [171] was one of the initial checkpointing techniques. It used periodic voltage checks to decide when to back up the program state. Hibernus [172] extended the work of [171] by introducing NVM. These checkpointing schemes do not consider the timely execution of the applications. TICS [173] overcomes this problem by introducing timely execution, branching, and efficient automatic checkpoints.

Checkpoints are placed using either software procedures or hardware components. Checkpointing approaches such as [171], [174], and [175] were proposed to backup and restore a consistent program state. The compiler or software procedures were primarily responsible for placing software-based checkpoints. Whenever a checkpoint is identified, the system initiates a backup procedure that stores the program state to NVM. In [175], checkpoints are placed based on the expiration of a timer. Hardware-based checkpoints were mainly associated with external devices. In [174], hardware-based checkpoints were placed using a voltage detector that triggers a backup mechanism for an NVP.

2.4 Developments in Real-time NVM-based MCUs

FRAM consumes less energy than other NVM technologies, such as Flash. FRAM can be helpful for low-power IoT devices. These NVM technologies motivated researchers due to their attractive characteristics, such as non-volatility, low cost, and high density [176, 34, 33, 177, 178].

Researchers started using real-time NVM-based MCUs for intermittent computing [179, 180, 11, 181, 182]. Researchers observed that using only NVMs at the cache or main memory level degrades the system's performance and consumes more energy, which motivates the use of hybrid memories. Recent NVM-based MCUs such as MSP430FR6989 [11] consist of both SRAM and FRAM. We need to utilize SRAM and FRAM efficiently and correctly; otherwise, we may degrade system performance and consume extra energy. To make the system more efficient, we need to map the application contents to either SRAM or FRAM. This is actually a memory mapping problem, similar to scratch-pad memories.

Recent works focused on incorporating NVMs as virtual memory during frequent power failures. Andrea et al. [183] propose Alfred, a mapping technique that maps virtual memory to volatile or NVM. Andrea et al. use machine-level codes for these mappings that achieve 2x improvement compared to existing techniques and systems. However, the Andrea et al. technique does not discuss the complete design choices or consider the real-time power scenarios.

Including NVMs in systems needs to answer the following research questions: when to checkpoint and where to checkpoint the volatile data. Researchers proposed efficient checkpointing techniques [184, 185] incorporating user-defined function calls that help determine how much energy is still available in the capacitor. Based on that analysis, the system invokes the checkpoint. These techniques even predict power failures and support intermittent computing.

2.5 Memory Mapping Techniques from Scratch Pad Memories to Modern MCUs

Researchers explored a similar mapping problem in scratch-pad memories (SPMs) [186, 187, 188]. Chakraborty et al. [189] documented the existing and standard memory mapping techniques on SPMs. In earlier works, memory mapping was done mainly between SPMs and main memory. Memory mapping can be done statically and dynamically [190, 191]. In static memory mapping, either ILP or the compiler can help to determine the best placement [186, 187, 192, 188]. ILP-solver takes inputs obtained from profilers and memory sizes as constraints in ILP-based memory mapping works. The ILP-solver provides the best placement option based on the objective function. In dynamic memory allocation [193, 194, 195, 196], either the user-defined program or the compiler will decide on an optimal placement choice at run-time.

However, our problem differs from the memory mapping techniques in SPMs because intermittent computing brings new constraints. During intermittent computation, the challenges were the forward progress of an application, the consistency of data, the consistency of the environment, and the concurrency between tasks. Due to these challenges, the execution model and development environment differ from SPM-based memory mapping techniques. As a result, we require a memory mapping technique that supports intermittent computation.

Researchers have explored memory mapping techniques and analysis for the MSP430FR6989 MCU. Kasım et al. [197] proposed a task-based mapping mechanism considering all event-driven paradigms that support intermittent computing and battery-less sensing devices. In FRAM-based MCUs, Jayakumar et al. [154] implement a checkpointing policy. They save the system state to FRAM during a power failure. Jayakumar et al. [198, 1] propose an energy-efficient memory mapping technique for TI-based applications in FRAM-based MCUs. Kim et al. [199] present a detailed analysis of energy consumption for all memory sections in FRAM-based MCUs with different memory mappings.

2.6 Challenges Identified from the Literature

We observed some challenges from the state-of-the-art works that motivated us to propose architectures and techniques to solve these challenges. We list these challenges below:

- **Challenge-1 related to NVMs write energy:** Writes to NVMs consume more latency and energy compared to volatile memory. We must optimize NVM utilization by reducing the usage of NVM or reducing the number of writes to NVM.

- **Challenge-2 related to HCAs:** Usually, for hybrid caches, block placement and movement between caches are the main challenges. In these hybrid caches, incorrectly placing a cache block in any region causes migration overhead. Migration overhead increases the number of writes to NVM and consumes more write energy. Introducing NVM in the L1 cache shows an impact on performance and energy consumption.

- **Challenge-3 related to checkpointing or backup/restore Overheads:** The main problem with the architectures and techniques mentioned above is the extra computation caused by multiple checkpoints. Another issue with checkpointing during a power failure is data inconsistency, leading to a corrupted output. Another disadvantage of the checkpointing approach is that whenever power comes back, we must restore the contents of non-volatile main memory to the cache. Whenever power comes back, we must implement a restoration procedure that restores the saved checkpoint from the NVM. Restoring the program state introduces one more extra overhead. These additional overheads degrade system performance very badly and consume more energy.

- **Challenge-4 related to energy constraints for backup/restore operations:** As per our knowledge, there were no works explored on this challenge. Our main concern is how we can guarantee a safe backup during a power failure. Is capacitor energy sufficient to perform this backup? As per our analysis of the related works, we observed that no work is using a constant amount of energy for backing up/restoring the volatile contents.

- **Challenge-5 related to Mapping contents in Hybrid Main Memory Architectures:** As per our knowledge, there were very few works that explored hybrid main memory architectures on real-time boards. Earlier works investigated this problem by analyzing the possibilities to make the system efficient. The works [198, 1, 199] have not covered all the possibilities and design choices. In addition, there is significantly less contribution toward memory mappings in FRAM-based MCUs that support intermittent computation.

2.7 Works Proposed to address the Challenges

- In order to solve challenges 1, 2, and 3 in section 2.6, we proposed an efficient HCA to address the challenges mentioned in the literature [34]. We use SRAM and STT-RAM at the L1 cache to reduce these migration overheads during both stable and unstable power scenarios. We proposed placement and migration policies, which also have a prediction table to predict the correct placement to reduce additional backup/restore overheads. We discussed the proposed architecture and techniques in the section 3 in detail.

- In order to solve challenges 1 and 4 in section 2.6, we proposed an architecture that uses constant energy for backing up/restoring the volatile contents to/from NVM during frequent power failures [33]. Efficient cache management policies have been introduced and thoroughly discussed in order to use fixed energy for backup operations during frequent power failures. We discussed the proposed architecture and techniques in the section 4 in detail.

- In order to solve challenges 1 and 5 in section 2.6, Our work [35] proposes an energy-efficient memory mapping technique for intermittently powered IoT devices that experience frequent power failures. We have discussed the proposed mapping technique in the section 5 in detail.

Chapter 3

Placement and Migration Policies for NVM based Hybrid L1 Caches

The research work presented in this chapter focuses on designing NVM-based hybrid cache architecture at L1 that supports intermittent computing. The proposed architecture introduces NVM at the L1 cache and main memory levels. NVM-based caches reduce system performance and consume more energy than SRAM-based caches. In order to get the benefits of both volatile and non-volatile memories, the proposed architecture employs a hybrid cache consisting of both SRAM and NVM. This chapter discusses the efficient placement and migration policies for hybrid cache architecture that uses SRAM and STT-RAM at the first-level cache. The proposed architecture includes cache block placement and migration policies to reduce the number of writes to STT-RAM. During a power failure, the backup strategy identifies and migrates the important blocks from SRAM to STT-RAM. This chapter begins with an introduction in Section 3.1. Section 3.2 describes the proposed hybrid NVM-based architecture, followed by a discussion on efficient placement and migration policies. Experimental setup and discussions on results have been given in section 3.4. The summary of the chapter is given in Section 3.5.

3.1 Introduction

In a conventional processor, all registers, caches, and main memories are volatile, which means that we lose all the data stored in these during a power failure. Data lost include the application's program state and progress. The program state includes the contents of registers, cache, and main memory. As a result, when a power failure occurs, some parts of the application must re-execute, slowing execution and consuming extra energy. This type of computing is defined as intermittent computing [7, 47, 48].

The solution is to store the application's program state at a precise restart point before a power failure. The question is, where should the program state be stored? Using Non-volatile memories (NVM), we can save the application's program state during a power failure. Recently, several new NVMs have been proposed, such as STT-RAM [200, 22, 23, 8], PCM [9, 201], Re-RAM [10], and FRAM [11].

NVM cache enhances the application's execution progress even during frequent power failures. The write access latency of NVM is higher than the SRAM, which creates the primary limiting factor of using NVM at the L1 cache. It is not a good idea to replace the entire SRAM with NVM; instead, we can integrate both SRAM and NVM to make a hybrid architecture [32] [202] at the cache level. STT-RAM offers better features than existing NVM technologies [22, 23, 8] at the cache level. PCM offers better features than the existing NVM technologies [9] at the main memory level. To design a hybrid cache, we observed two main challenges associated with this kind of design.

The first challenge is that introducing STT-RAM as a cache can deteriorate the system's performance due to its long access time and consumes more dynamic energy. We observed that the STT-RAM-based architecture consumes 49.53% more energy than the SRAM-based architecture. We need to use STT-RAM efficiently so that the system performs better and consumes less energy. In the case of a hybrid cache, we need to utilize SRAM and STT-RAM as efficiently as possible to benefit from both.

The second challenge is placing a cache block in the wrong memory region in these hybrid caches causes unnecessary movements between cache regions and yields extra overheads, i.e., migration overheads. These overheads increase the number of reads and write operations and require additional cycles and energy, making the system inefficient by consuming more energy and deteriorating the overall system's performance.

In the existing architectures, Xie et al. [32] also introduced a similar hybrid cache architecture that consists of STT-RAM at the L1 cache. The main observations that we reported from the Xie et al. work and the main challenges associated with the existing hybrid cache architecture [32] are listed below.

- For the prediction table, Xie et al. used a pattern sampler, which doesn't gather the complete details of the application.
- Where the placement and migration policies cannot provide accurate predictions if the prediction information is incomplete. As previously stated, inaccurate predictions increase the number of reads and write operations, which consume more execution time and energy.
- Xie et al. used a checkpointing scheme, which uses more energy because we need to write/read to/from the NVM for every checkpoint.
- Xie et al. used a standard LRU replacement policy to identify the cache block for eviction. What if the evicted block turns out to be a write-intensive block the next time? The used replacement policy may result in unnecessary writes to NVM and consume more energy.
- Xie et al. backup all volatile contents during a power failure, which is not always necessary, and push more writes to NVM during frequent power failures.

All the above challenges and observations motivated us to propose an efficient hybrid cache architecture that considers these issues.

In this chapter, we address challenge-1, 2, and 3 described in chapter 2, and we achieve objective 1 discussed in chapter 1 by proposing a hybrid cache architecture that efficiently uses both SRAM and STT-RAM by gaining benefits from SRAM during regular operation and from STT-RAM during frequent power failures. The main contributions of this work are summarized below:

- We propose efficient placement and migration policies for the hybrid cache architecture at the L1. The proposed placement policy suggests which cache block should be placed in which memory region, whether SRAM or STT-RAM. We proposed a prediction mechanism that helps in block placement. The proposed prediction mechanism helps to reduce unnecessary migrations between cache regions.
- The proposed architecture supports intermittent computing. To reduce the backup time during frequent power failures, the proposed architecture identifies the important blocks that need to be backed up to NVM instead of saving the entire volatile content.

3.2 Proposed Architecture

The proposed architecture uses the proposed placement, migration, and backup policies [34]. Fig. 3.1 shows the proposed architecture. Every cache set in the proposed architecture contains a mix of SRAM and STT-RAM cache blocks. Along with the valid bit (V), dirty bit (D), tag, and data in each cache block, we added three more entries: i) Read-Intensive Counter (RIC), ii) Write-Intensive Counter (WIC), and iii) Confidence bits (CONF). These three entries are beneficial for cache placement and migration policies. We classified the blocks into two types: read-intensive blocks and write-intensive blocks. Read-intensive (RI) blocks have more read accesses than a predefined threshold at a given point in time, whereas write-intensive (WI) blocks have more write accesses than a predefined threshold at a certain point in time.

We keep two counters for each block, RIC and the WIC. Furthermore, we added a 2-bit CONF field that tracks important blocks; important block information is helpful during power failures. A prediction table has also been included. Each prediction table entry has a previous region (PR) bit. During the replacement/eviction process, this PR bit is updated. The PR bit stores the block's most recent cache region.

3.2.1 Placement and Migration Policies

We describe the proposed block placement and migration policies in this section. Because STT-RAM has higher read/write latency and consumes more energy than SRAM, the placement policy aims to reduce the number of writes to STT-RAM. The write latency of STT-RAM is ten times more than its read latency [200]. Therefore, we would like to place write-intensive blocks in the SRAM cache and read-intensive blocks in the STT-RAM cache. We use the PR bit to check the prediction table and place the block in the appropriate cache region based on whether it is read-intensive or write-intensive.

The algorithm 1 demonstrates the placement policy in case of a cache miss. Line 1 uses a tag to check the prediction table on a read/write miss. We access the PR bit associated

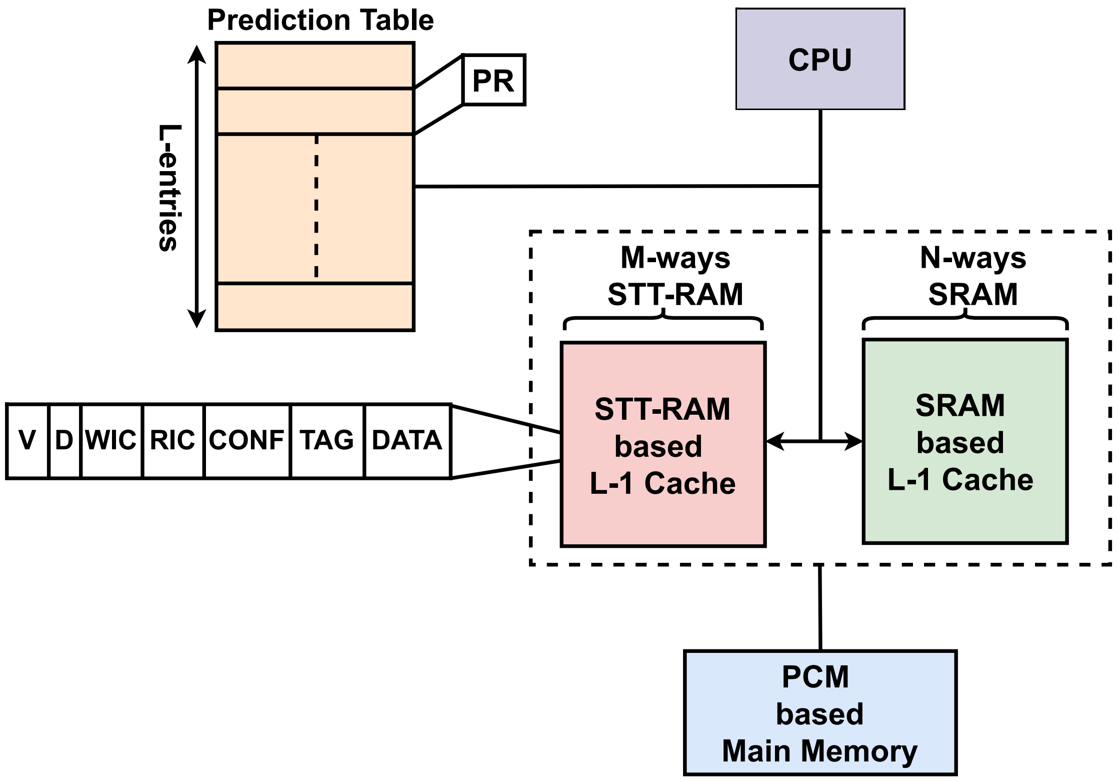


Figure 3.1: Overview of the Proposed Architecture, which consists of SRAM and STT-RAM-based caches at L1 and PCM at the main memory level.

with the tag entry. We keep the PR bit to note the previous block placement information for that tag entry. If PR = 0, lines 3-5 in algorithm 1 check whether the corresponding STT-RAM cache set is full or not. If it is full, we replace the block with the lowest RIC value; otherwise, we place it in the STT-RAM cache. Suppose PR!=0, lines 10-12 in algorithm 1 check whether the SRAM cache set is full or not. We replace the block with the lowest WIC value; otherwise, we place the block in the SRAM cache.

The algorithm 2 describes the placement and migration policies whenever there is a read hit. Line 1 checks the block's RIC value with the empirically determined threshold. We fixed the threshold limit empirically. If the block's RIC is equal to the threshold, we call that block an RI block. The proposed placement policy suggests that all RI blocks should be placed in STT-RAM. If the block is present in the SRAM cache, we migrate from SRAM to STT-RAM and re-initialize RIC, WIC, and CONF to zero. If the block is not in the SRAM cache, we place the block in the STT-RAM cache and increment CONF by 1. If the threshold does not equal the block's RIC value, we increment RIC by 1. The block chosen for replacement has to update its PR bit in the prediction table. If RIC reaches the threshold and CONF reaches 11 state, then we do not increment RIC.

The algorithm 3 describes the placement and migration policies whenever there is a write hit. Line 1 checks the block's WIC value with the threshold. If the block's WIC equals the threshold, we call that block a WI block. The proposed placement policy suggests that all WI blocks should be placed in SRAM. If the block is already present in

Algorithm 1: Placement Algorithm in case of Cache miss

```
1: Check Prediction Table.
2: if $PR == 0$ then
3:    if STT-RAM set is full then
4:       Replace block with lowest $b.RIC$
5:       Update the replaced block's $PR$ bit in the Prediction Table.
6:    else
7:       Place in the STT-RAM cache.
8:       Re-Intialize $b.RIC$, $b.WIC$ to zero.
9:    end if
10: else
11:    if SRAM set is full then
12:       Replace block with lowest $b.WIC$.
13:       Update the replaced block's $PR$ bit in the Prediction Table.
14:    else
15:       Place in the SRAM cache.
16:       Re-Intialize $b.RIC$, $b.WIC$ to zero.
17:    end if
18: end if
```

the STT-RAM cache, we migrate from STT-RAM to SRAM and re-initialize RIC, WIC, and CONF to zero. This case reduces the number of writes to the STT-RAM cache. If the threshold does not equal the block's WIC value, we increment WIC by 1. The block chosen for replacement has to update its PR bit in the prediction table. If WIC reaches the threshold and CONF reaches 11 state, then we do not increment WIC.

3.2.2 Prediction Table Design

The importance of the prediction table in the proposed architecture is to store the previous region for the respective tag entry. The prediction table has L entries, where L denotes the number of entries in the prediction table. This table acts as a direct-mapped buffer, indexed using (Address/block_size) % L. Each entry in the prediction table has a PR (Previous Region) field. The prediction table does not store the tag bits in order to save area; its size is L bits. Initially, all bits in the prediction table are set to 1.

We update the PR field whenever there is a replacement in the cache due to the SRAM/STT-RAM set being full. If PR is 1, the block is a WIC because its WIC was greater than RIC during replacement. Place the WIC block into the SRAM cache region. If PR is zero, the block is a RIC because its RIC is greater than WIC during replacement. Place the RIC block into the STT-RAM cache region.

3.2.3 Support for Intermittent Power Supply

Our proposed architecture supports intermittent computing and performs well during frequent power failures. We define important blocks as those with high CONF values. We used RIC/WIC values to update the CONF field. When power is restored in a traditional architecture, we begin execution by accessing blocks from the main memory and copying

Algorithm 2: Placement and Migration Algorithm in case of Read hit

```
 1: if $b.RIC$ == threshold then
 2:    if Block is in SRAM then
 3:       if STT-RAM set is full then
 4:          Replace block with lowest $b.RIC$ and $b.CONF$
 5:          Update the replaced block's $PR$ bit in the Prediction Table.
 6:          Migrate to STT-RAM.
 7:       else
 8:          Migrate to STT-RAM.
 9:       end if
10:       Re-Intialize $b.RIC$, $b.WIC$, $b.CONF$ to zero.
11:    end if
12:    $b.CONF = b.CONF + 1$
13:    Re-Initialize $b.RIC$ to zero.
14: else
15:    $b.RIC = b.RIC + 1$
16: end if
```

Algorithm 3: Placement and Migration Algorithm in case of Write hit

```
 1: if $b.WIC$ == threshold then
 2:    if Block is in STT-RAM then
 3:       if SRAM set is full then
 4:          Replace block with lowest $b.WIC$ and $b.CONF$
 5:          Update the replaced block's $PR$ bit in the Prediction Table.
 6:          Migrate to SRAM.
 7:       else
 8:          Migrate to SRAM.
 9:       end if
10:       Re-Intialize $b.RIC$, $b.WIC$, $b.CONF$ to zero.
11:    end if
12:    $b.CONF = b.CONF + 1$
13:    Re-Initialize $b.WIC$ to zero.
14: else
15:    $b.WIC = b.WIC + 1$
16: end if
```

them to the cache. We save important blocks in STT-RAM that help to start execution without restoring blocks from the main memory to SRAM.

We propose a state model to assist in determining the most important blocks. Using the CONF field, we can determine which blocks should be present in STT-RAM during a power failure. Initially, CONF is in the 00 state and supports four states, ie 00, 01, 10, and 11 states, as shown in Fig. 3.2. To represent the proposed state model, we need a 2-bit CONF field. The algorithmic process of updating the CONF field has already been described in algorithms 2 and 3.

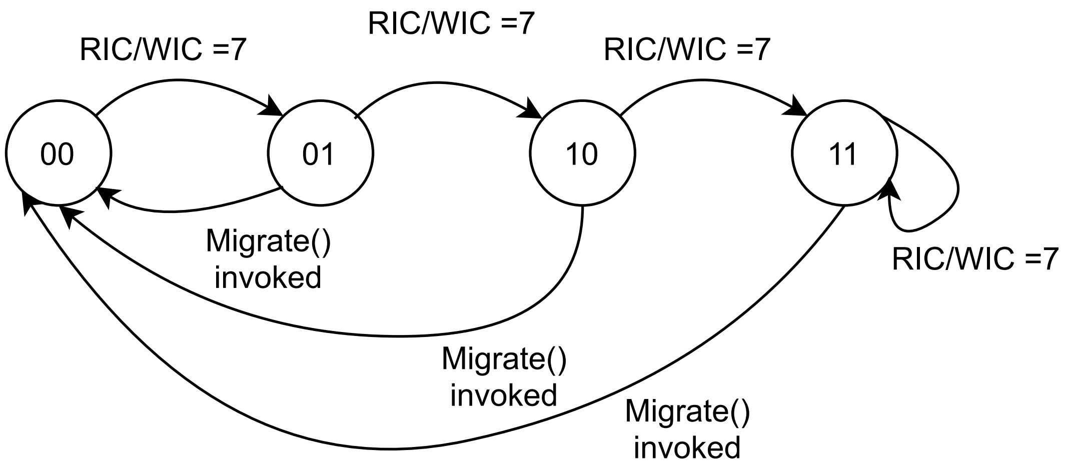


Figure 3.2: State Diagram for Updating CONF.

In summary, if RIC/WIC exceeds the threshold, CONF is increased by one and advances to the next state. When CONF is in the 11 state and crosses the threshold, it remains in that state. If any migration occurs from SRAM/STT-RAM to STT-RAM/SRAM cache, then CONF resets to the 00 state along with the RIC and WIC values.

During a power failure, the proposed backup policy is triggered to save important blocks from SRAM to STT-RAM. According to the proposed backup policy, the blocks with CONF field 11 are the most important blocks. Therefore, we prioritize blocks with the CONF field on the order of 11 > 10 > 01 > 00. If any SRAM block has a CONF field of 11, we replace that block with the least priority block in STT-RAM. If there is no block with 11 state in the SRAM, we decrement our priority order by 1.

Now our priority becomes 10. If there are blocks with 10 state in the SRAM cache line, we replace the blocks with the least priority block in STT-RAM. If there is no block with 10 state in the SRAM line, we decrement our priority order by one. Similarly, we check blocks with 01 and 00 states. Priority with 00 is the case where we copy the SRAM contents to STT-RAM and the STT-RAM contents to PCM. Whenever the power comes back, the STT-RAM contents are accessed automatically without copying to SRAM. Our migration policy automatically migrates from STT-RAM to SRAM if needed and vice versa.

3.2.4 Storage Overhead

We analyze the storage overhead because we added extra bits, a prediction table, and backup logic. For the same system configuration shown in table 3.1, we evaluate the overhead area of the proposed architecture. We showed the area overhead as an example. There are two aspects of the proposed architecture that cause storage overhead.

- The proposed architecture has two 3-bit counters and two confidence bits per block. The data cache has 256 blocks, each with 8 bits, so the data cache requires 256*8=2048 bits.
- The proposed prediction table has 4K byte entries with 1-bit per entry, resulting in a total storage overhead of 1024*4 = 4096 bits.

The overall storage overhead of the proposed architecture will be 2048 + 4096 = 6144 bits=0.75KB. The total percentage of area overhead is about 0.75KB/32KB=2.34%.

3.3 Detailed Example

Fig. 3.3 illustrates the operation of the proposed architecture. In figure 3.3, Initially, the SRAM cache has (a,c) blocks, and the STT-RAM cache has (b,d) blocks. We defined all counters and CONF as a tuple [RIC, WC, CONF] and initialized it to [0, 0, 00]. A prediction table has a PR field. We take a sequence of access requests; read requests are labeled as rd_i (i.e., read block i), and write requests are labeled as wr_i (i.e., write block i). We labeled different timing points as A, B, C, .., and K. In this section, we discuss how the proposed architecture works after every timing point.

In Fig A of Fig. 3.3, we update the RIC of ‘a’ to 2 because of two consecutive reads. In Fig B, the WIC of ‘b’ has become 2. In Fig C, [RIC, WIC] of ‘a’ updates to [3, 1]. In Fig D, the WIC of ‘b’ becomes 7, which equals the threshold and becomes a write-intensive block. Our placement policy suggests that write-intensive blocks should be placed in SRAM. SRAM set is full; to replace the block, we find the block having the lowest WIC. The block ‘c’ has a low WIC value; we replace ‘c’ with ‘b’ and reset all ‘b’ counters to [0, 0, 00]. In Fig E, the RIC of ‘a’ becomes 7, which equals the threshold and becomes a read-intensive block. Our placement policy suggests that read-intensive blocks should be placed in STT-RAM. STT-RAM set had one empty slot; we migrated ‘a’ from SRAM to STT-RAM. Reset all ‘a’ counters to [0, 0, 00] and update the WIC of ‘b’ to 2.

In Fig F, the RIC of ‘a’ updates to 4. A new block request, ‘c,’ occurred between the timing points F and G. The block request ‘c’ is not present in both caches. Check the prediction table for index 2, associated with tag_c, to find the c’s PR field. We found an entry in the prediction table of index 2 with PR = 1. If the PR value is 1, the block is placed in the SRAM cache during the last eviction. We place ‘c’ in the SRAM cache. In Fig G, the WIC of ‘c’ updates to 7, which equals the threshold, becomes a write-intensive block and updates the RIC of ‘a’ to 4. Our placement policy suggests that write-intensive

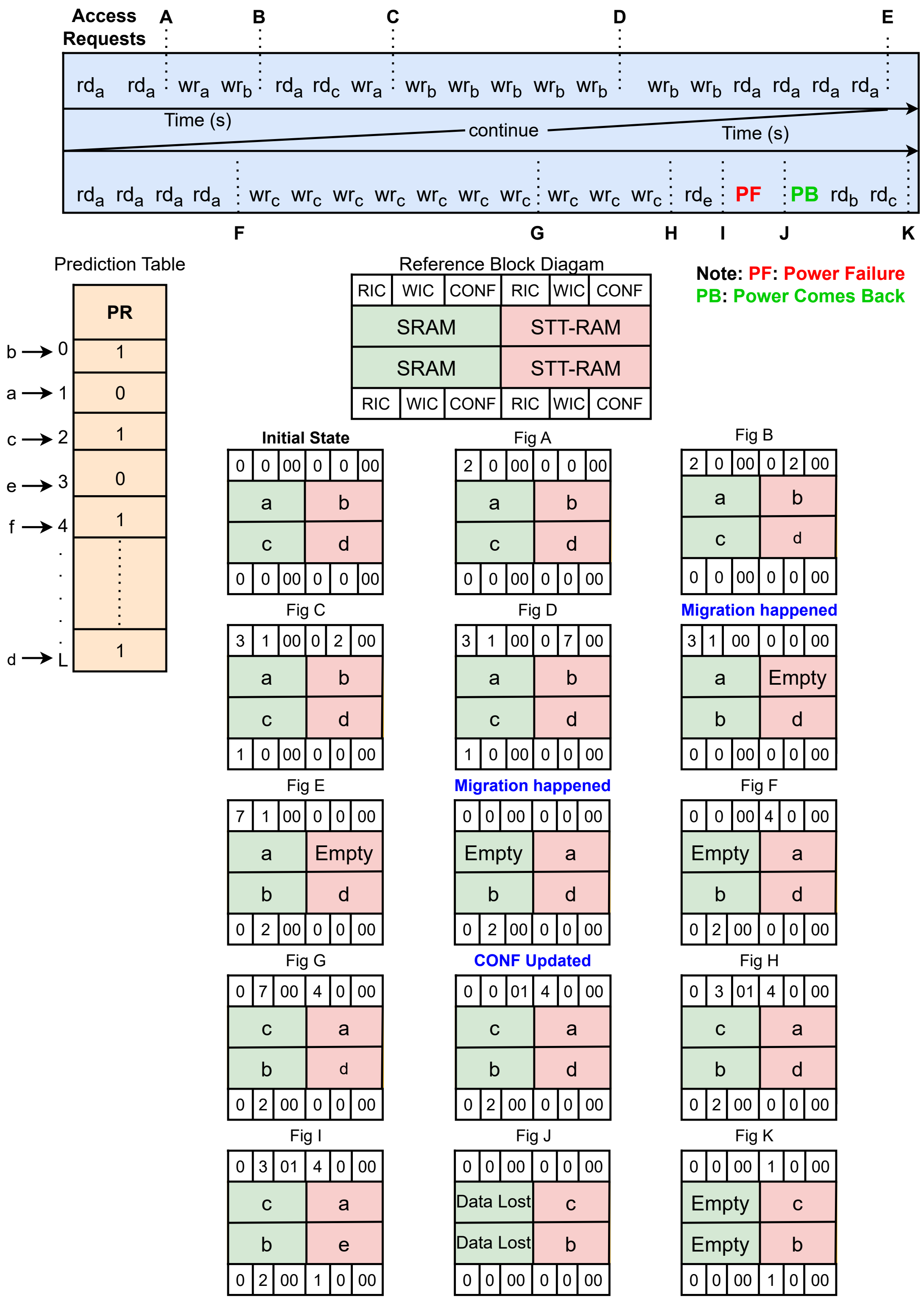


Figure 3.3: Working Example of the Proposed Architecture for a given set of Access Requests.

blocks should be placed in the SRAM; ‘c’ is already in the SRAM. We update the CONF of ‘c’ to 01 and reset the counter values. After H, the WIC of ‘c’ updates to 3.

A new block request ‘e’ occurred between the timing points H and I. Block request ‘e’ is not present in both caches. Check the prediction table for index 3, associated with tag_e, to find the PR field of e. We found an entry in the prediction table of index 3 with PR = 0. If the PR value is 0, the block is placed in the STT-RAM cache during the last eviction. We place ‘e’ in the STT-RAM cache. STT-RAM set is full; we find the lowest RIC to replace the block. The block ‘d’ has a low RIC value; we replace ‘e’ with ‘d’. Update all ‘e’ counters to {1, 0, 00}.

Power failure (PF) occurred; our backup policy saves important blocks using the CONF field. Where the CONF of ‘c’ has 01 and ‘a’ has 00, our priority order suggests that 01 has the highest priority compared to 00. We place ‘a’ in main memory and backup ‘c’ in STT-RAM. We prefer write-intensive blocks compared to read-intensive ones during a power failure. So ‘b’ replaces ‘e.’ In Fig J, ‘c’ and ‘b’ are saved to STT-RAM. Whenever power comes back (PB), we do not require any restoration process. Fig K shows the RIC of ‘c’ and ‘b’ updates to 1.

3.4 Experimental Setup and Results Analysis

3.4.1 Experimental Setup

We evaluate the proposed architecture using the gem5 [41] simulator and 18 benchmarks from the MiBench suite [74]. Our targeting applications primarily work with embedded devices. Based on the literature survey, Mi-Bench Suite is the preferred set of applications for embedded devices. As a result, we compared the proposed architectures to baselines and existing architectures using the Mi-Bench suite. The general microarchitectural parameters used for the implementation are shown in Table 3.1.

Table 3.1: System Configuration

Component	**Description**
CPU core	1-core, 480MHZ
L1 Cache	Block size - 64-byte, 4-way associative (2-way SRAM, 2-way STT-RAM); Private cache (16KB hybrid D-cache, and 16KB I-cache)
Size Parameters	VB-1bit, WIC and RIC-3bits, CONF-2bits, L- 4K bytes, threshold-7, and PR-1bit
Main memory	128MB PCRAM
Others	Clock Period: 2ns, SRAM Read: 1 Cycle, SRAM Write: 2 Cycles, STT-RAM Read: 2 Cycles, STT-RAM Write: 10 Cycles, PCM Read: 35 Cycles, and PCM Write: 100 Cycles

Table 3.2: Nvsim parameters of SRAM, STT-RAM Caches, PCM memory (350K, 22nm)

Parameter	**16KB SRAM**	**16KB STT-RAM**	**128KB STT-RAM**	**128MB PCM**
Read Latency	0.792 ns	1.994 ns	1.861 ns	204.584ns
Read Energy	0.006 nJ	0.081 nJ	0.123 nJ	1.553 nJ
Write Latency	0.772 ns	10.520 ns	10.446 ns	RESET - 134.923 ns
				SET - 264.954 ns
Write Energy	0.002 nJ	0.217 nJ	0.542 nJ	RESET - 6.946 nJ
				SET - 6.927 nJ

Table 3.2 shows the dynamic energy and latency for a single read and write operation on SRAM and STT-RAM, taken using Nvsim [70].

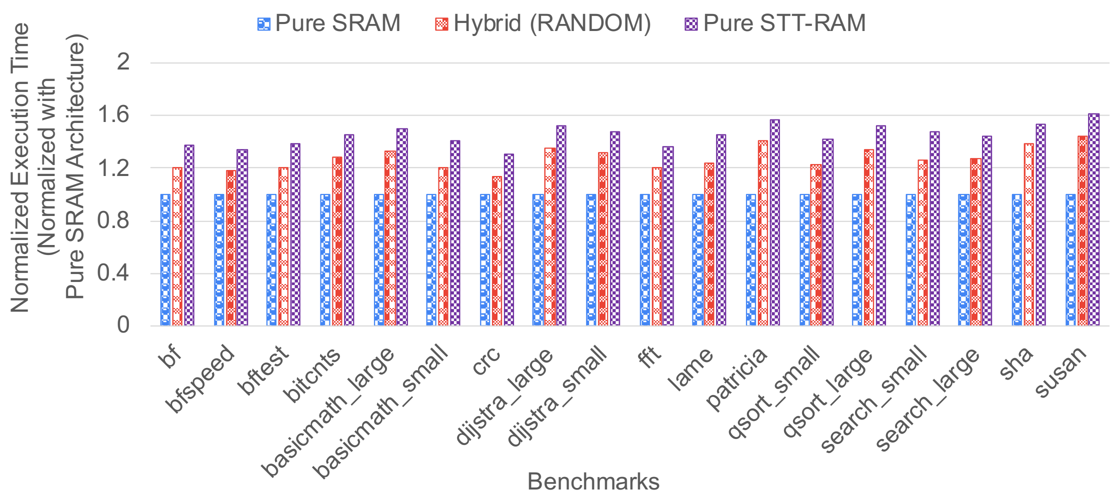
3.4.2 Baseline Architecture

We modeled a baseline architecture to compare with the proposed architecture. In order to choose the baseline architecture, we did a thorough investigation by comparing different possible approaches. If we have two memory regions, we will have three possible architectures, one is to use only SRAM, the second is to use only STT-RAM, and the third is to use both. Among these three architectures, we have to choose one relevant architecture. The chosen baseline architecture will be used for comparison purposes.

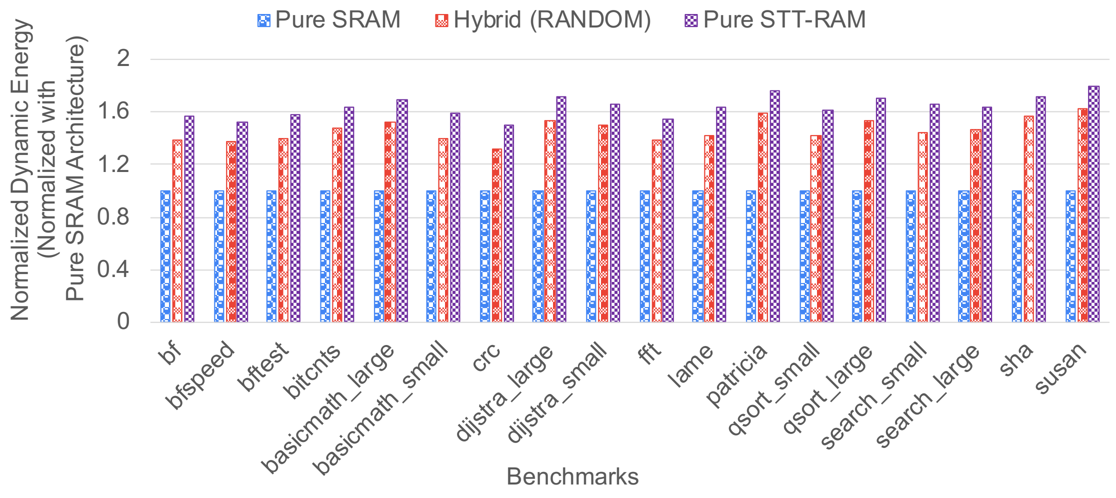
We first compared the performance and dynamic energy consumption of the pure SRAM, pure STT-RAM, and hybrid (SRAM and STT-RAM) cache architectures to determine the baseline architecture. Based on the analysis, we choose the relevant baseline architecture to compare the proposed and existing architectures throughout this work.

- **Pure SRAM cache**: We don't require any placement or migration policies in pure SRAM cache because we have only SRAM at L1.
- **Pure STT-RAM cache**: We do not require any placement or migration policies in pure STT-RAM cache because we have only STT-RAM at L1.
- **Hybrid cache**: At L1, the hybrid cache architecture includes both SRAM and STT-RAM. We use a random placement policy in this architecture. In the random placement policy, the block is randomly placed in either SRAM or STT-RAM. We use the migration policy based on counters. We empirically determined the threshold as 7 and the size of the counters as 3 bits.

We set the L1 size to 32KB in all three architectures. In all three cache architectures, we did not use any prediction mechanisms. In Fig. 3.4, the performance and energy consumption of the cache architectures are normalized with the pure SRAM cache



(a) Execution Time



(b) Dynamic Energy Consumption

Figure 3.4: Comparisons between Pure SRAM, Pure STT-RAM, and Hybrid Cache Architectures.

architecture. We observed that the hybrid architecture performs between pure SRAM and STT-RAM architecture from Fig. 3.4. Thus, we used the normal hybrid architecture as our baseline architecture. We termed only one architecture as a baseline throughout this chapter.

Baseline Architecture: We selected the hybrid architecture as our base architecture throughout this work with the modeling details mentioned above.

We experimented with the baseline architecture to determine the threshold value. When the respective counter crosses its threshold, we move the block from one cache to another to check energy values. The size of the counters is determined by the threshold value. For example, if the threshold is 3, the counter size is $log4$. (counts from 0 to 3). We experimented with threshold values of 1, 3, 7, and 15. We observed that threshold value 7 consumes less energy than the other threshold values on average. We set the threshold to

7 because we noticed that migrations between cache regions increase when the threshold is exceeded. We also observed that NVM gets more writes if the threshold is higher than 7, increasing the hybrid cache architecture's energy consumption. Fig. 3.5 shows that threshold value 7 consumed less energy than the other threshold values.

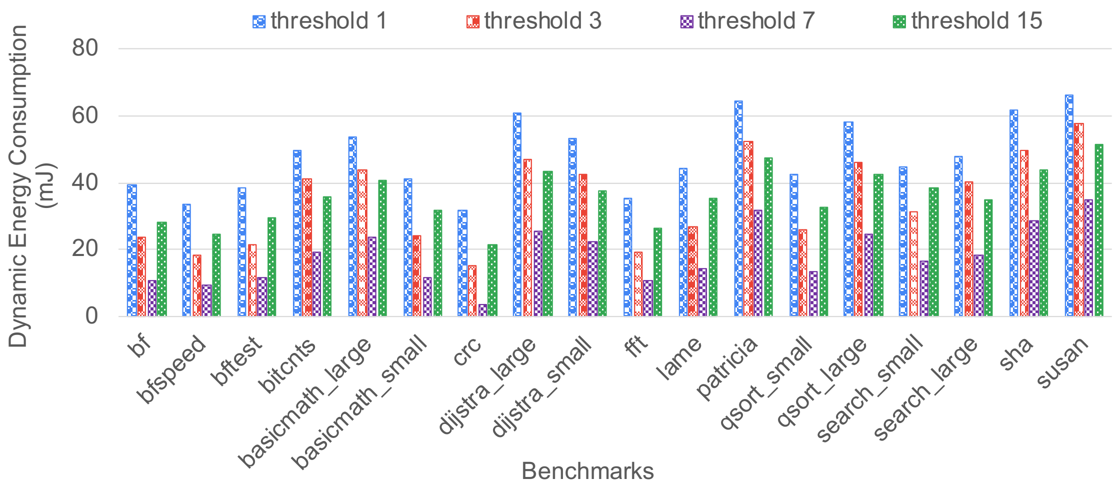


Figure 3.5: Dynamic Energy Consumption for Various Threshold Values.

3.4.3 Results & Analysis

This section evaluates the proposed architecture under stable power and frequent power failures. We also evaluated the proposed architecture for backup and energy efficiency w.r.t. baseline and existing architectures.

3.4.3.1 Number of Writes to NVM

One of the main objectives of the proposed architecture is to reduce the number of writes to the STT-RAM cache. To achieve this, we place the write-intensive blocks in the SRAM cache. We have shown the ratio of write operations to STT-RAM with total write accesses in Fig. 3.6. A lower number of writes to STT-RAM shows the effectiveness of the proposed architecture. The percentage of writes to the STT-RAM cache is normalized with the baseline architecture shown in Fig. 3.6. In general, the proposed architecture helps reduce STT-RAM write operations from 63.35% to 35.93% compared to the baseline architecture. If our prediction accuracy increases, the number of migrations decreases. If the number of migrations decreases, we observe fewer writes to NVM. Thus, our proposed architecture decreases the number of writes to STT-RAM.

3.4.3.2 Analysis for Execution Time and Dynamic Energy Consumption

Under Stable Power Scenarios: We compare our proposed architecture with the baseline and the architecture proposed by Xie et al. [32]. We implemented the work of Xie et al. [32] to analyze both stable power and intermittent power systems. For a fair

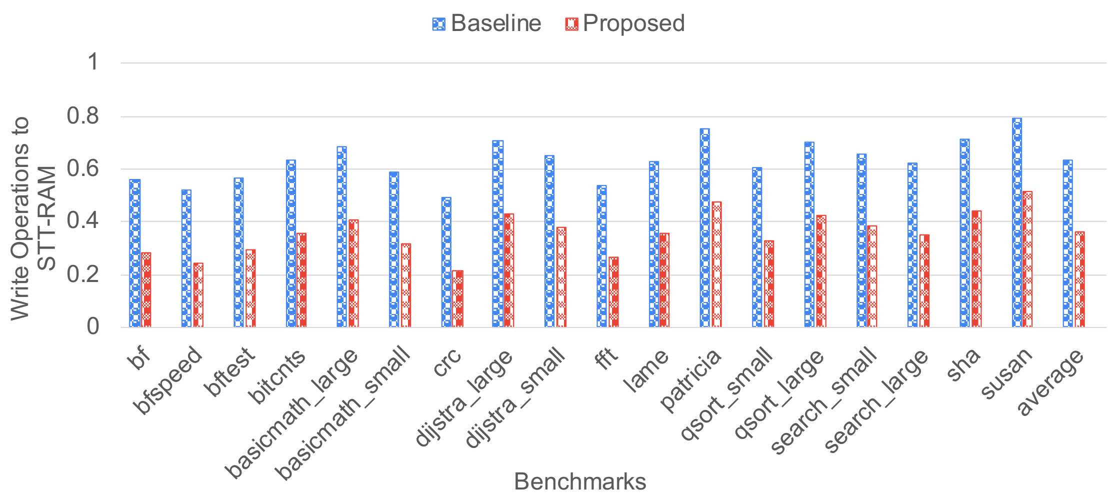


Figure 3.6: Write operations to STT-RAM

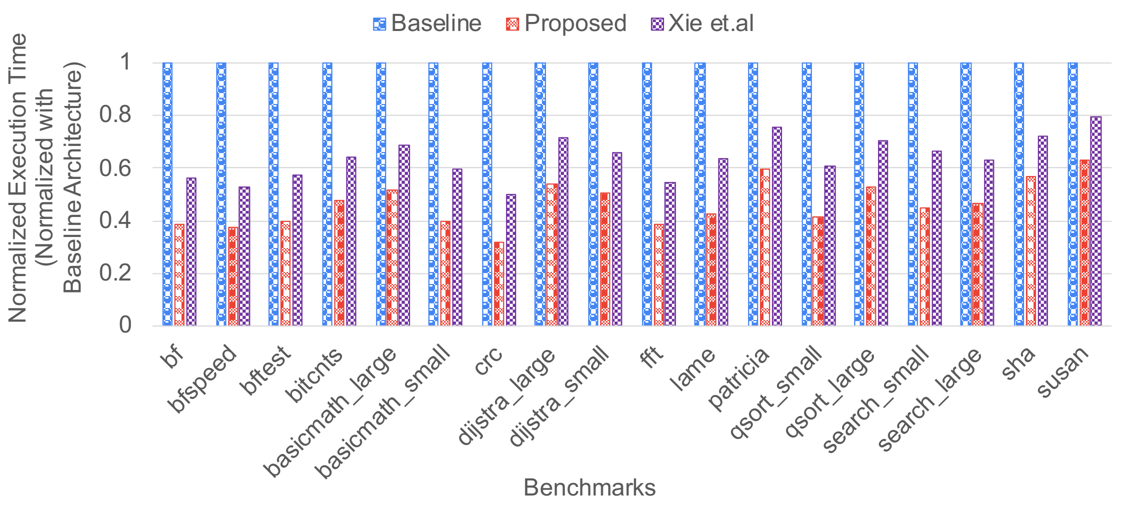
comparison, all these architectures use the same system configuration shown in Table 3.1 and the energy/delay values of STT-RAM and PCM from Table 3.2.

Reducing the STT-RAM writes also guarantees better endurance and a lifetime of IoT nodes. Performance and energy consumption values are normalized with the baseline architecture in Fig. 3.7 (a) and Fig. 3.7 (b). Figs. 3.7 (a) and 3.7 (b) show a better execution time and dynamic energy consumption than the existing and baseline architectures. We achieve better values because of accurate prediction when we compare the proposed architecture with the Xie et al. architecture. In addition, the proposed prediction table helps decrease the number of migrations and accesses. Therefore, our architecture results in 32.85% better execution time and saves 23.42% of dynamic energy consumption than the baseline architecture.

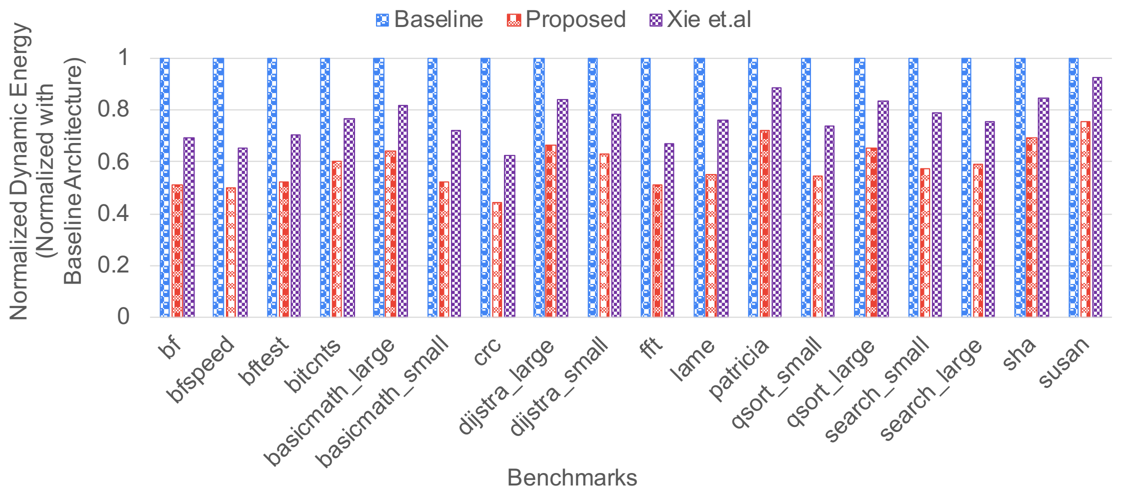
Under Unstable Power Scenarios: We assume that frequent power failures occur every 2 and 4 million instructions. We performed all experiments with one billion instructions in the Gem5 simulator. We modeled three scenarios of power failure, as shown in Table 3.3. In case 1, power failures occur every 2 million instructions. In case 2, power failures occur every 4 million instructions. In case 3, power failures occur randomly in between 2 to 4 million instructions.

Considering energy-harvesting sources, such as piezoelectric and vibration-based sources, they extract much less energy from the surroundings. In these cases, the capacitor cannot store enough energy, leading to frequent power failures. As a result, our proposed architecture supports these worst-case scenarios. However, the existing work by Xie et al. [32] made similar assumptions, assuming that each power failure occurs every 500 ms.

We also compared the SRAM+PCM-based architecture to show how much performance improved during intermittent power supply. In the SRAM + PCM architecture, SRAM is the L1 cache, and PCM is the main memory. We introduced a power failure randomly and a safe point for every 4 million instructions. When a power failure occurs, we back up all SRAM contents to PCM. Whenever power comes back, we start the application's execution



(a) Execution Time



(b) Dynamic Energy Consumption

Figure 3.7: Comparisons between Proposed, baseline, and Existing Architectures under Stable Power.

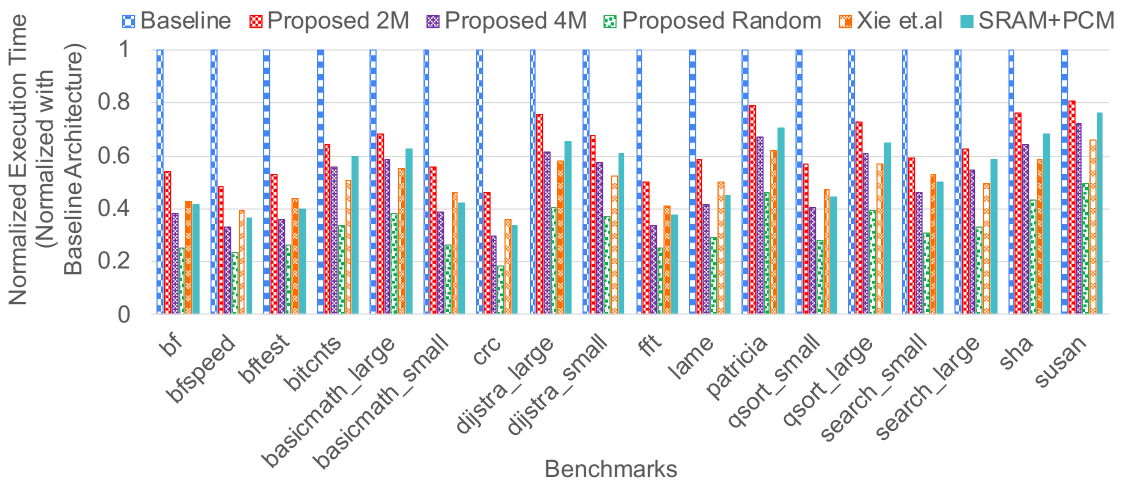
from the nearest safe point. When we compared the SRAM + PCM architecture with the proposed architecture, the proposed architecture gave better results because the proposed architecture saves data in the L1 cache itself (by using STT-RAM). proposed architecture saves the re-execution time of the application and reduces the number of writes to PCM during a power failure. The performance and energy consumption values are normalized to the baseline architecture. We compare execution time and energy consumption with the baseline architecture during these frequent power failures, as shown in Fig. 3.8 (a) and Fig. 3.8 (b).

We also compared the proposed architecture with existing work, i.e., Xie et al. They checkpoint only selective dirty blocks from SRAM to STT-RAM during power failures. This type of checkpointing increases writes to PCM, which increases the dynamic energy consumption of their architecture. Thus, the proposed architecture achieves better

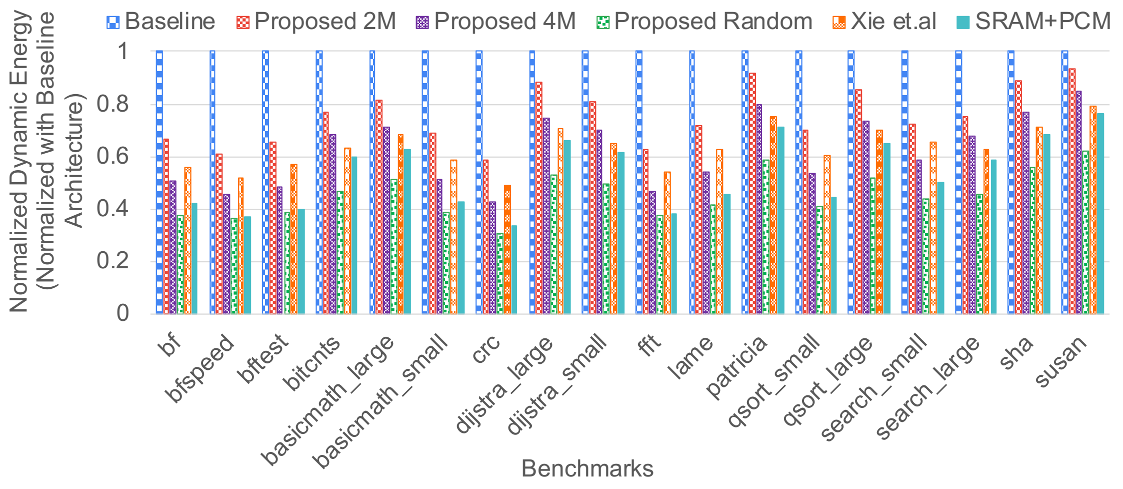
Table 3.3: Different Power Failure Scenarios

Configuration	Power Failure (PF) Scenario
Case-1 (Proposed 2M)	PF for every 2-Million Instructions
Case-2 (Proposed 4M)	PF for every 4-Million Instructions
Case-3 (Proposed Random)	Random PF between every 2 to 4-Million Instructions

execution time and energy values than the existing architecture. Whenever power comes back, the proposed architecture uses blocks from STT-RAM directly. In the work of Xie et al., STT-RAM consists of fewer blocks than the proposed architecture, which increases the execution time in existing work.



(a) Execution Time



(b) Dynamic Energy Consumption

Figure 3.8: Comparisons between Proposed, baseline, and Existing Architectures under Frequent Power Failures.

3.4.3.3 Analysis for Dynamic Energy with Traditional Checkpointing

Under Stable Power Scenarios: During stable power, we performed experiments to compare the traditional checkpointing approach with the proposed architecture. We use a traditional checkpointing method, creating a safe point for every 4 million instructions. We save the program state for every 4 million instructions to the main memory. Our proposed architecture outperforms traditional checkpointing. In the traditional checkpointing technique, backup occurs for each safe point, but in the proposed architecture, backup occurs only during a power failure. We normalize the energy consumption values with the traditional checkpointing approach. The proposed architecture reduces energy consumption by 22.95%, as shown in Fig. 3.9 (a).

Under Unstable Power Scenarios: We compare the traditional checkpointing approach with the proposed architecture during power failures. We save the program state for every 4 million instructions. We retrieve the program state from the main memory at every safe point to continue with the remaining execution of the application. For instance, if a random power failure occurs at 7^{th} million instructions. We re-execute the application from 4^{th} million instruction because the nearest safe point is at 4^{th} million instruction. Energy consumption values are normalized on the basis of the traditional checkpointing approach. We compared the proposed architecture with the traditional checkpointing approach, which reduces energy consumption by 31.03%, as shown in Fig. 3.5 (b).

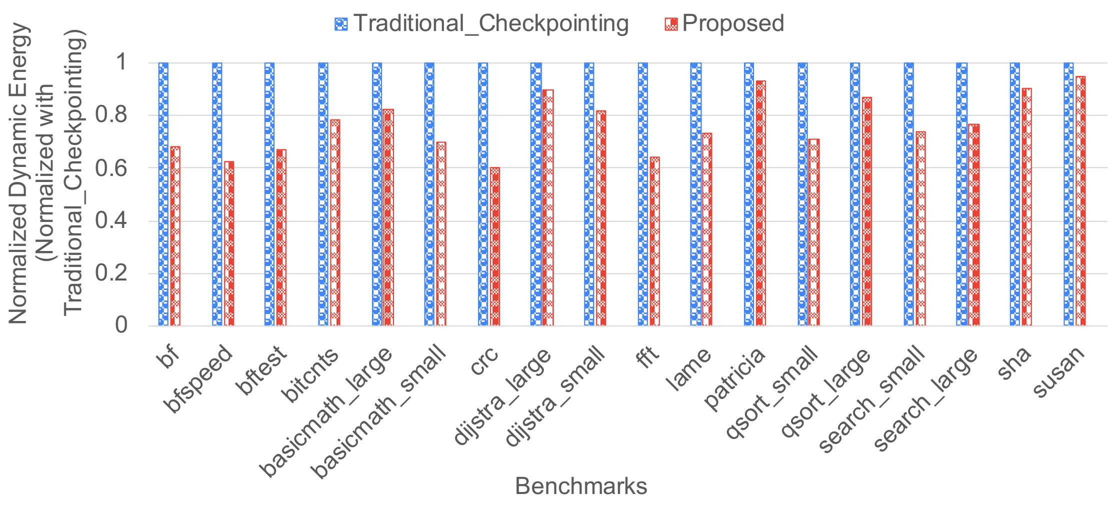
3.4.3.4 Analysis for Different Cache Settings

We also performed experiments by changing the system configurations, such as cache sizes and associativity, that are different from the system configuration shown in Table 3.1. We used six different cache settings for these experiments, as shown in Table 3.4.

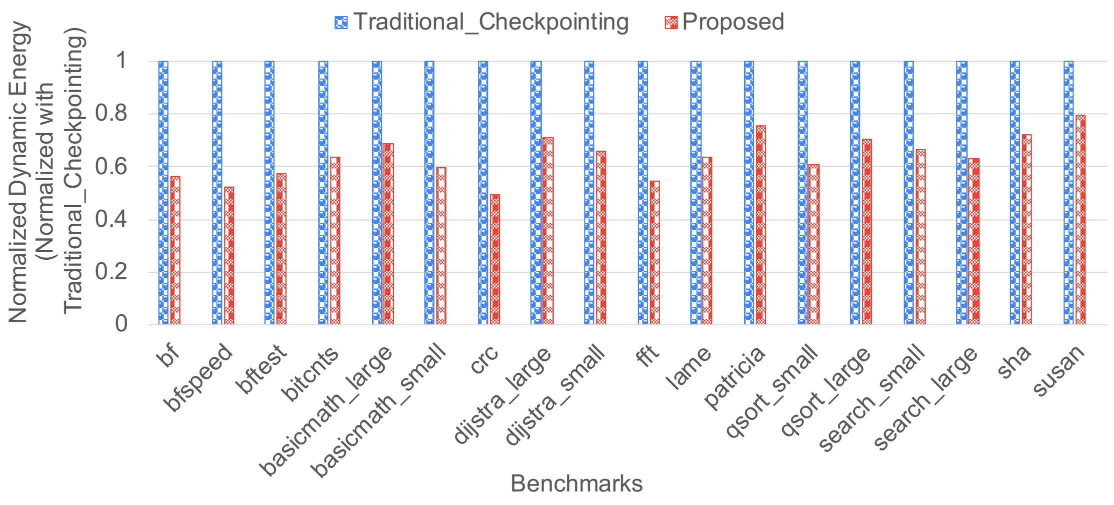
Here, we used two different cache sizes; one is 16 KB, and the other is 32 KB. We compared the energy consumption for these 10 configurations under stable power. We used all the proposed policies and techniques in these 10 sets of configurations.

Under Stable Power Scenarios: We observed that for the 16KB cache size, configuration-1 consumes more energy than the other 4 configurations during stable power because STT-RAM consumes more energy than SRAM. During a stable power supply, we observed that the configuration with more SRAM ways consumes less energy than others without relating to cache sizes. We observed that the 16K{8,0} setting consumes less energy than all the other 4 configurations because it is like a pure SRAM-based architecture), and after this, the 16K{6,2} setting consumes less energy than all the other 3 configurations. After 16K{6,2}, the 16K{4,4} setting consumes less energy than the other two configurations, i.e., 16K{2,6} and 16K{0,8}. Compared to the pure STT-RAM cache architecture, the 16K{6,2} setting consumes 38.10% less energy, and the 16K{4,4} setting consumes 17.97% less energy. Compared to the pure STT-RAM cache architecture, the 16K{2,6} setting consumes 5.82% less energy.

Similarly, we observed that the 32K{8,0} setting consumes less energy than all other



(a) Dynamic Energy Consumption under Stable Power



(b) Dynamic Energy Consumption under Frequent Power Failures

Figure 3.9: Comparisons between Proposed and Baseline Architectures with Traditional Checkpointing Approach.

4 configurations because it is like a pure SRAM-based architecture) and after this, the 32K{6,2} setting consumes less energy than all other 4 configurations for the 32KB cache size. The order is the same as the 32KB cache size because a large SRAM size gives more benefits during stable power. Compared to the pure STT-RAM cache architecture, the 32K{6,2} setting consumes 42.91% less energy, and the 32K{4,4} setting consumes 26. 01% less energy. Compared to the pure STT-RAM cache architecture, the 32K{2,6} setting consumes 13.37% less energy.

Under Unstable Power Scenarios: We observed that for the 16KB cache size, configuration-5 consumes more energy than the other four configurations during an unstable power due to backing up the SRAM contents to SRAM. During an unstable power supply, we observed that the configuration with more STT-RAM ways consumes less energy than others without relating to cache sizes. We observed that the 16K{0,8}

Table 3.4: Different Cache Configurations Used to Analyze Proposed Policies

Configuration	Cache Setting	Cache Size	Associativity
1	16K {0;8}	16KB	8-way {0-way SRAM, 8-way STT-RAM}
2	16K {2,6}	16KB	8-way {2-way SRAM, 6-way STT-RAM}
3	16K {4,4}	16KB	8-way {4-way SRAM, 4-way STT-RAM}
4	16K {6,2}	16KB	8-way {6-way SRAM, 2-way STT-RAM}
5	16K {8,0}	16KB	8-way {8-way SRAM, 0-way STT-RAM}
6	32K {0,8}	32KB	8-way {0-way SRAM, 8-way STT-RAM}
7	32K {2,6}	32KB	8-way {2-way SRAM, 6-way STT-RAM}
8	32K {4,4}	32KB	8-way (4-way SRAM, 4-way STT-RAM}
9	32K ({6,2}	32KB	8-way {6-way SRAM, 2-way STT-RAM}
10	32K {8,0}	32KB	8-way {8-way SRAM, 0-way STT-RAM}

setting consumes less energy than all other four configurations because it is like a pure STT-RAM-based architecture), and after this, the 16K{2,6} setting consumes less energy than all three configurations. After 16K{2,6}, the 16K{4,4} setting consumes less energy than the other two configurations, i.e., 16K{6,2} and 16K{8,0}. Compared to the pure SRAM cache architecture, the 16K{2,6} setting consumes 16.70% less energy, and the 16K{4,4} setting consumes 12.19% less energy. Compared to the pure STT-RAM cache architecture, the 16K{6,2} setting consumes 7.11% less energy.

Similarly, we observed that the 32K{0,8} setting consumes less energy than all other four configurations because it is like a pure STT-RAM-based architecture) and after this, the 32K{2,6} setting consumes less energy than the three configurations for the 32KB cache size. The order is the same as the 16KB cache size because a large STT-RAM size gives more benefits during unstable power, where it backups more data and reduces both backup and restore overhead. Compared to the pure SRAM cache architecture, the 32K{2,6} setting consumes 21.10% less energy, and the 32K{4,4} setting consumes 15.49% less energy. Compared to the pure STT-RAM cache architecture, the 32K{6,2} setting consumes 9.14% less energy.

3.4.3.5 Analysis for Backup & Energy Efficiency

For the hybrid architecture model, our design performs a backup during a power failure, and when the power comes back, it performs a memory restore operation. As a result, we define the energy required to execute the application in Equation 3.1.

$$E_{overall} = E_{exec} + E_{backup} + E_{restore} \tag{3.1}$$

Where $E_{overall}$ is the energy required to execute the overall application, E_{exec} is the energy required to execute the program. We backup the system state by copying all register contents and SRAM cache blocks to NVM. The energy consumed by the backup procedure is E_{backup}, where it depends on the number of bytes to be backed up to NVM as shown in Equation 3.2.

$$E_{backup} = N_{w\_L1} * e_{w\_sttram} + N_{w\_main} * e_{w\_pcm} \tag{3.2}$$

Where N_{w_L1} is the number of writes to STT_RAM, N_{w_main} is the number of writes to main memory, e_{w_sttram} is the energy per write for the STT-RAM and e_{w_pcm} is the energy per write for PCM RAM

The energy required to restore volatile contents from NVM is $E_{restore}$, where it depends on the number of bytes to be restored from NVM and can be defined as follows:

$$E_{restore} = N_{r\_L1} * e_{r\_sttram} + N_{r\_main} * e_{r\_pcm} \tag{3.3}$$

Where N_{r_L1} is the number of reads to STT_RAM, N_{r_main} is the number of reads to main memory, e_{r_sttram} is the energy per read for the STT-RAM and e_{r_pcm} is the energy per read for PCM RAM. Whenever the power comes back, the size of the restored contents is the same as the content that was backed up during a power failure. Thus, Equation 3.2 and Equation 3.3 were interrelated in terms of sizes, and as we are doing automatic restoration, so we don't have any restore overhead in our proposed hybrid cache architecture.

one of our objectives is to maximize the backup efficiency (η), which is defined as shown in Equation 3.4.

$$\eta = \frac{N_{w\_L1}}{N_{w\_L1} + N_{w\_main}} \tag{3.4}$$

If we achieve less N_{writes}, our η increases. Thus, we achieve one of our objectives by reducing N_{writes}.

Lastly, we define energy efficiency as the ratio of the energy consumed during normal execution without any power failures to the energy consumed during power failures. Let θ be the energy efficiency as defined in Equation 3.5.

$$\theta = \frac{E_{normal}}{E_{overall}} \tag{3.5}$$

Here, E_{normal} is the energy required for normal execution without any power interruptions.

As shown in Fig. 3.11 and Fig. 3.12, we performed experiments to analyze the backup

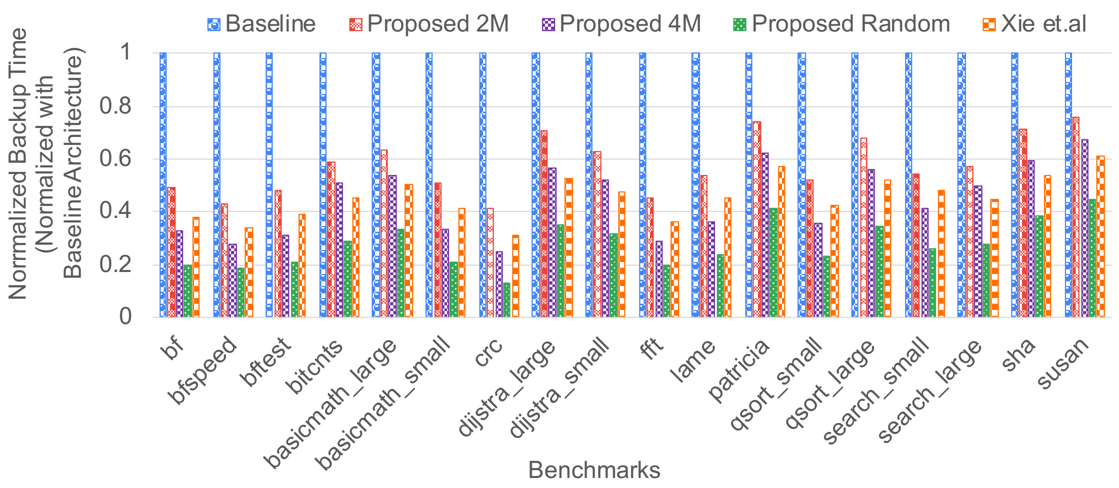


Figure 3.10: Backup Time

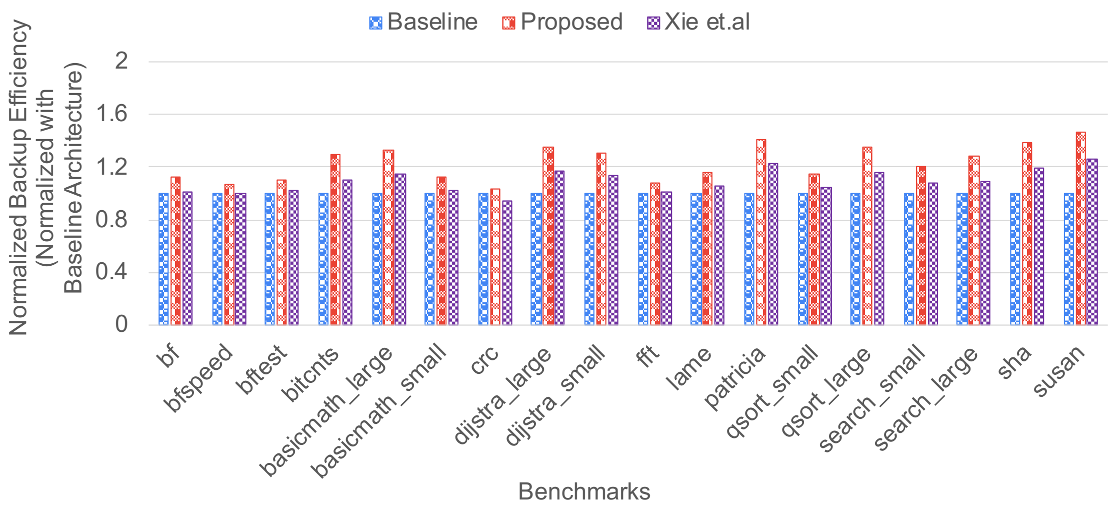


Figure 3.11: Comparison of Backup Efficiency (η) during Power failures

efficiency (η) and energy efficiency (θ) for both proposed and existing architectures. The values of η and θ are normalized with the baseline architecture. Our proposed architecture improves η by 32.52% and θ by 43.41% due to the proposed backup strategy. The other reason for the improvement in η and θ is a reduction in E_{backup} and B_t.

We calculated the average backup time (B_t), i.e., the time required to backup all the SRAM contents to NVM. We also evaluated a random intermittent power system, where power failure occurs very often and randomly, to check B_t and the efficiency of the proposed architecture. The performance and energy consumption values are normalized based on the baseline architecture. We compare the average B_t with the baseline, as shown in Fig. 3.10.

The limitations that we observed in this chapter are as follows: (1) This chapter limits the NVM at the L1 and main memory levels; (2) This work does not support fixed energy constraints, which means that the proposed architecture does not perform safe backup

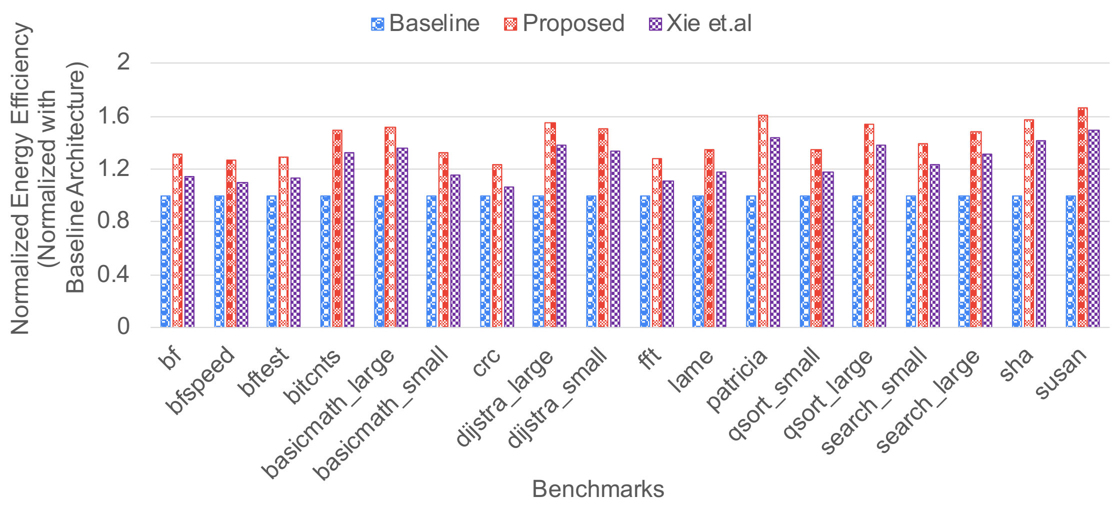


Figure 3.12: Comparison of Energy Efficiency (θ) during Power failures

during power failures using a fixed amount of energy; and (3) The proposed architecture consists of a prediction table, implying that optimized prediction-based techniques for these hybrid cache architectures may still exist.

3.5 Summary

In this chapter, a hybrid cache architecture at L1 has been proposed for intermittently powered embedded systems. The proposed architecture is beneficial for IoT applications, where power failures are often unpredictable. Due to its high write latency and energy consumption, NVM introduces overhead in hybrid caches. We proposed an efficient prediction-based placement policy and an intelligent migration policy that efficiently uses SRAM and STT-RAM. We reduce the number of writes to STT-RAM using the proposed prediction table. Compared to the baseline architecture, the proposed architecture reduces STT-RAM writes from 63.35% to 35.93%. As a result, our energy consumption and execution time are reduced.

We compared the proposed architecture with the state-of-the-art and baseline architectures. The proposed architecture improves energy and backup efficiency. We proposed a backup strategy to ensure efficient backup of the program state. During a power failure, the proposed backup strategy helps to recognize important blocks and migrate them to the STT-RAM cache. Compared to baseline and existing architectures, the proposed architecture requires less backup time.

Chapter 4

NVM based Last-Level Cache Architecture under Energy Constraints

This chapter comprises an architecture that ensures safe backup of volatile contents during a power failure under a given energy constraint. Non-volatile memory (NVM) based processors were explored to store the program state during a power failure. The energy stored in a capacitor is used for a backup during a power failure. Since the size of a capacitor is fixed and limited, the available energy in a capacitor is also limited and fixed. Thus, the capacitor energy is insufficient to store the entire program state during frequent power failures. Using a proposed dirty block table (DBT) and a writeback queue (WBQ), this work limits the number of dirty blocks in the L1 cache at any given time. This chapter begins with an introduction in Section 4.1. Section 4.2 describes the proposed architecture in detail, which is finally followed by a detailed experimental setup and results and analysis in the context of stable power and intermittent power supply scenarios in Section 4.3. We write the summary of this work in Section 4.4.

4.1 Introduction

Non-volatile processors (NVP) have been proposed [10, 203, 204] as a solution in the past. An NVP stores the processor state in NVM during a power failure. Thus, NVP resumes the application's computational tasks once the power supply is restored, thus achieving faster recovery and backup speeds compared to traditional processors.

During a power failure, NVP needs to store volatile memory content (registers/SRAM caches) in an NVM such that the application can restart from the same point. The size of the registers and the content of the SRAM caches determine the time and energy required for the backup. Although using no SRAM caches reduces backup time and energy to almost negligible, it significantly impacts performance.

When using battery-less hardware, the device must be turned off when the harvested power is no longer available. To avoid sudden power failures and fluctuations, such devices accumulate energy in a capacitor that smoothens power availability and provides energy during power failures [5, 47, 48]. Thus, during a power failure, the energy stored in a capacitor is used to backup the processor state. The entire process state in volatile memory must be backed up to ensure correctness. In addition, cache lines store a copy of data elements present in memory; therefore, cache lines that are not modified need not be backed up. However, in the worst case, all cache lines could be dirty.

For example, if the base architecture consists of NVM in LLC and main memory as shown in Fig. 1.3 (b). So, we require a capacitor that helps to backup the entire L1 dirty blocks to PCM or STT-RAM. Equation 4.1 formulates the backup energy required for architecture-2. In the worst case, we need to backup all L1 contents to STT-RAM or PCM.

$$E_{backup} = N_{B/L1} \times (e_{w\_sttram}) \tag{4.1}$$

Where E_{backup} is the backup energy required for the base architecture during a power failure. $N_{B/L1}$ is the number of blocks at L1, and e_{w_sttram} is the energy per write for the STT-RAM cache block.

Since the capacitor energy storage capacity is limited and fixed, only fixed SRAM contents can be copied to NVM during a power failure.

Usually, a capacitor has fixed energy ($E_{capacitor}$) that can only backup a fixed number of cache blocks (K) during a power failure. In the worst case, we need to backup the entire L1 content to either STT-RAM or PCM for the base architecture. We need a larger capacitor to backup all L1 contents, as shown in Equation 4.1, which is infeasible in practice. A large capacitor requires more time to charge. Thus, maintaining the larger capacitor will not help us during frequent power failures, resulting in faulty computations. This observation motivated us to propose an architecture that uses fixed energy to backup the L1 dirty cache contents during a power failure.

We defined $E_{capacitor}$ in Equation 4.2. Where C is the capacitance, and V is the operating voltage.

$$E_{capacitor} = \frac{1}{2}CV^2 \tag{4.2}$$

A sub-optimal solution is to constrain the cache size based on the energy available in a capacitor or design the capacitor to store the entire cache. Therefore, given $E_{capacitor}$ as constant, our objective is to maximize the cache blocks K. We define K using Equation 4.3.

$$K = \frac{E_{capacitor} - E_{reg\_file}}{e_{w\_sttram}} \tag{4.3}$$

Where K is the maximum number of blocks that can backup to NVM during a power failure. Here E_{reg_file} is the energy required to backup the register file to STT-RAM.

Instead of saving entire SRAM contents during a power failure, we backup only K blocks from the L1 cache to NVM. Where $K << \left(N_{dirty/L1}\right)$ w.r.t base architecture during a backup procedure.

Thus, we require a capacitor of size C that can provide energy of $\geq E_{capacitor}$; this ensures that the K blocks are completely and safely backed up to NVM. We assumed that the capacitor energy was fixed and could not be replaced again. As a result, $E_{capacitor}$ is sufficient to backup K blocks from the L1 cache and the register file contents to NVM, which we explained in section 4.2.2.

The main objective here is to use the given $E_{capacitor}$ efficiently and complete the backup within the given $E_{capacitor}$ during a power failure. We need to restrict the number of dirty blocks to K to achieve the above objective. We need to address the following issues that help to restrict the number of dirty blocks to K:

1. The number of dirty blocks at any point in time should be counted and tracked.
2. The write time to LLC is longer than the L1 cache. Every $(K + 1)th$ dirty block would require additional time to write back to the LLC. The processor would stall during this time, which degrades the system's performance. How can we avoid this unnecessary stalling?
3. We need to decide which block should be replaced when the dirty blocks are more than K.
4. Where should all dirty blocks be stored during a power failure?

In this chapter, we address challenge-1 and 4 described in chapter 2, and we achieve objective 2 discussed in chapter 1 by proposing an NVM-based architecture. In order to solve the above issue mentioned, we brief our contributions as follows:

- In the proposed architecture, we divided K into $M + N$ blocks. To address the first issue, we propose a dirty block table (DBT) that tracks dirty blocks M [33]. We discuss DBT in section 4.2.
- We introduced a write-back queue (WBQ) at the L1 cache that tracks N dirty blocks, which resolves the second issue. We discuss WBQ in the section 4.2.
- To address the third issue, we explore two different replacement policies. These replacement policies are discussed in section 4.2.1.
- The proposed architecture supports intermittent computing, and we introduced an STT-RAM-based backup region at LLC, which provides additional storage space for volatile data during power failures.

4.2 Proposed Architecture

The proposed architecture is shown in Fig. 4.1. Each cache block contains a valid bit (V), a dirty bit (D), a tag, and data.

The algorithm 4 refers to whenever there is a write hit in the L1 cache. Line 1 checks; if a write hit occurs and the dirty bit is 0, we set the dirty bit to 1 and create an entry in DBT. Line 3 checks whether the number of valid DBT blocks is equal to M. If the number of valid blocks in DBT equals M, we use the DBT replacement policy to make space for the new entry. Line 6 checks if there are more than N entries in WBQ; the processor stalls to complete a writeback to LLC. Line 12 determines whether the number of valid blocks in DBT is less than M and inserts an entry into DBT. Line 15 determines whether there is a write request and if the dirty bit is already set to 1. If Line 15 becomes true, we update the WC field in DBT.

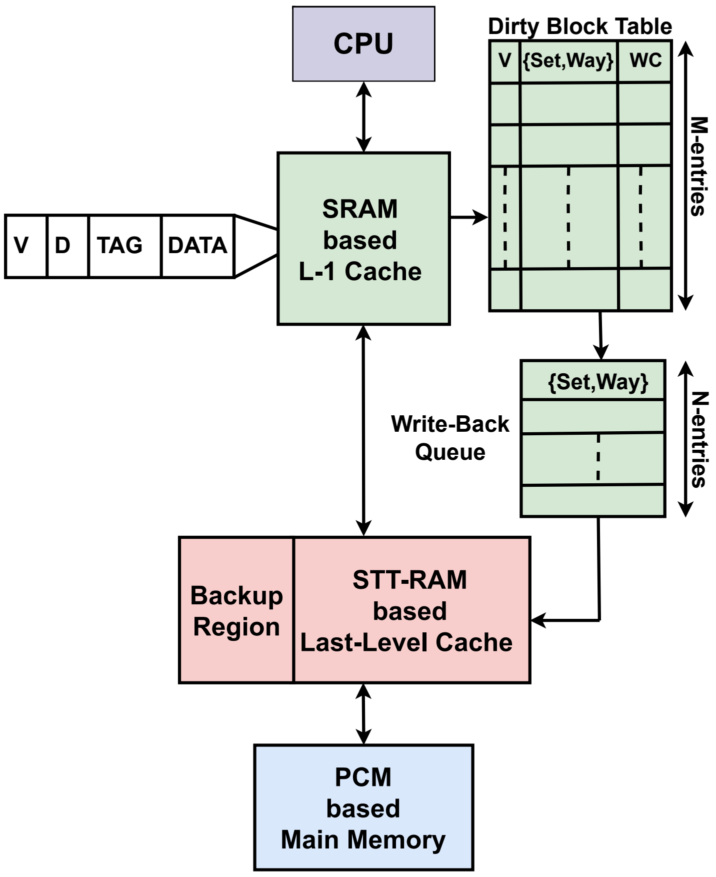


Figure 4.1: Overview of Proposed Architecture that consists of SRAM-based cache at L1, STT-RAM-based cache at LLC, and PCM at main memory, which tracks and backup the finite Dirty blocks from L1 to LLC using the proposed DBT and WBQ.

If we find any dirty block, we make an entry in DBT. DBT stores dirty block information in four fields: valid bit (V), set, way, and write counter (WC). DBT is implemented as an associative buffer of M entry. DBT does not impact the clock period because it does not come under the critical path. When a victim entry is chosen from DBT for replacement, the tag information of that entry is moved into WBQ.

WBQ has N entries that store the {set, way} field information. To hide the latency of STT-RAM writes, we use a writeback queue in a standard mechanism. WBQ works as a queue and writes the data to LLC to maintain N entries. When the data are written from WBQ, the dirty bit in the cache is cleared. We update the modified value in WBQ whenever there is a write hit to the WBQ entry.

The primary importance of WBQ is seen in a scenario where we need to write back the data to LLC, which takes an additional number of cycles for every $(K + 1)th$ block. During this time, the processor would halt for the 'X' number of cycles to write one of the $(K + 1)th$ blocks to LLC, where X is the number of cycles required to complete one

STT-RAM write. This additional stalling degrades system performance and consumes a significant amount of energy.

Rather than saving the entire contents of the SRAM, we backup $M + N$ dirty blocks to the STT-RAM. This reduces writes to STT-RAM and PCM during a power failure. In case of a miss at the L1 cache, the architecture is the same as the conventional architecture.

Algorithm 4: L1 cache hit access

```
On access to block b in set s
 1: if (b.write) & !(b.D) then
 2:    b.D =1
 3:    if DBT.size() == M then
 4:       invokes replacement policy.
 5:       replacement policy returns a victim DBT entry.
 6:       if WBQ.isFull() then
 7:          STALL
 8:       else
 9:          Make an entry in WBQ.
10:       end if
11:    else
12:       Make an entry in DBT.
13:       Update WC field in DBT.
14:    end if
15: else if b.write then
16:    Update WC.
17: else
18:    This is a read-hit case; provide the data.
19: end if
```

4.2.1 Replacement Policy in DBT

When the DBT size exceeds >M, we require a replacement policy in the DBT to replace any of the M entries. The traditional LRU replacement policy does not work for our architecture because we want to replace a block based on the number of writes or the write behavior. For the proposed architecture, we explore two replacement policies. First, the least frequently written (LFW) policy replaces an entry that has received the least number of writes compared to all other entries in the DBT. Second, the least recently written (LRW) policy maintains the recency information for each block. Instead of replacing the block based on write counts, the LRW policy recommends replacing the most recently written block to preserve write access recency information.

We introduce a write counter (WC) field in the DBT to identify the LFW block; the size of the WC depends on 'M.' If a write request is made to any DBT entry, we increment the WC by one. For instance, the WC has a size of 5 bits. If the WC of the requested entry is equal to the maximum value $(2^5 - 1)$, we do the logical right shift, i.e., we decrement $(2^5/2)$ from all DBT entries. For example, the size of the WC is 5 bits, and the DBT has four entries with the WC values as follows {19, 17, 31, 3}. Suppose that a new write request is received for the third entry. The WC value for the third entry exceeds 31; we subtract 16 from all DBT entries, which becomes $\{3, 1, 15, 0\}$. We replace the entry with the lowest WC value during a replacement request.

We used the LRW field in the DBT to implement the LRW replacement policy. The LRW field has a size of $\lceil logM \rceil$ bits. For example, if M is 16, we require a 4-bit LRW field. The implementation of LRW is identical to that of the 4-bit priority queue. When we use the LRW policy, we replace the WC field in DBT with the LRW field.

4.2.2 During Intermittent power supply

Apart from the STT-RAM-based LLC, we introduce a backup region (BR). STT-RAM is used to implement the backup region. The backup region can always have a maximum size of K blocks + reg file. To read/update the backup region, we used the same access latency and energy values as the STT-RAM cache. During a power failure, we have registers and K block (M+N) contents to backup. When the power comes back, we move the contents of the backup region to the L1 cache. With these contents, we begin the execution.

Validity of Proposed work: The following activities occur in the system during a power failure and before initiating the backup procedure.

1. Except for the processor, all peripherals of the system receive power-off signals.
2. As the processor is stalled, no new instructions are carried out by the processor.
    (a) Since all peripherals are switched off, no peripheral can issue an interrupt during this time.
3. Because the processor is stalled, it consumes no dynamic energy. As a result, we can use the energy of the entire capacitor to backup the volatile contents.

After completing the previous three phases, we initiate the backup procedure. During backup, volatile contents are stored in the register file and the SRAM-based cache; a specially designed controller is responsible for saving the volatile contents to the STT-RAM-based BR at LLC. During backup, the controller reads the (set, way) fields in DBT and WBQ one entry after another to identify a block in the L1 cache, then writes the dirty data to the STT-RAM-based BR at LLC. The controller performs the following operations: reading from the register file, reading from the SRAM-based cache ('K' block data), and writing to the STT-RAM-based BR at LLC. In the proposed architecture, the register file size and the number of dirty blocks at the L1 cache are minimal, and the backup procedure requires a constant and fixed number of cycles. As a result, the proposed architecture requires a fixed overhead for backup.

The processor has been set to shutdown mode once the backup procedure has been completed. We use the energy of the capacitor ($E_{capacitor}$) to perform the backup. When power is restored, we perform the same operations as in a conventional processor.

Therefore, for a given capacitor energy, system configuration, register files, and read/write parameters for the memory system, we define the required number of K blocks in section 4.1 using Equation 4.3.

4.3 Experimental Setup and Results

4.3.1 Experimental Setup

The proposed architecture is evaluated using the gem5 simulator [41] and 14 MiBench benchmarks [74]. Our targeting applications primarily work with embedded devices. Based on the literature survey, Mi-Bench Suite is the preferred set of applications for embedded devices. As a result, we compared the proposed architectures to baselines and existing architectures using the Mi-Bench suite. Table 4.1 shows the micro-architectural parameters used for implementation. We collected dynamic energy and latency values for a single read and write operation to SRAM and STT-RAM using Nvsim [70], as shown in Table 3.2.

Table 4.1: System Configuration

Component	**Description**
CPU core	1-core, 480MHZ
L1 Cache	Block size is 64-byte, 4-way associative Private cache (16KB D-cache,and 16KB I-cache)
Last-Level Cache	Block size is 64-byte, 16-way associative Private cache (128KB D-cache, and 128KB I-cache), write-back cache policy
Size Parameters	VB - 1bit, WC - 6bits K- 16; M- 12, N- 4, LRW- 4bits, and C should be $\geq$ 1.92 nF
Main memory	128MB PCM
Others	Clock Period: 2ns, SRAM Read: 1 Cycle, SRAM Write: 2 Cycles, STT-RAM Read: 2 Cycles, STT-RAM Write: 10 Cycles, PCM Read: 35 Cycles, and PCM Write: 100 Cycles

4.3.2 Baseline Architecture

We modeled three baseline architectures to compare them with the proposed architecture, as shown in Table 4.2.

- In L1, Baseline-1 uses a write-through policy. As a result, L1 contains no dirty blocks. Baseline-1 uses the least amount of backup energy during a power failure.

Table 4.2: Overview of Baseline Architectures Configurations

Architecture	Memory	Policy
Baseline-1	L1: SRAM (32 KB)	Write-through
	LLC: STT-RAM (256 KB)	Write back
	Main Memory: PCM (128 MB)	-
Baseline-2	L1: SRAM (32 KB)	Write back
	LLC: STT-RAM (256 KB)	Write back
	Main Memory: PCM (128 MB)	
Baseline-3	L1: SRAM (4 KB)	Write back
	LLC: STT-RAM (256 KB)	Write back
	Main Memory: PCM (128 MB)	

- At L1, Baseline-2 uses a write-back policy. As a result, the number of LLC writes decreases. Baseline-2 improves system performance over baseline-1.
- Baseline-3 has a 4 KB L1 cache and the same LLC and main memory sizes as in baseline-2. Baseline-3 helps to determine whether using small-size volatile memory at L1 improves performance during a stable power supply. During a power failure, our proposed backup contents are the same size as baseline-3.

Under a stable power supply, we compared baselines 1, 2, and 3 with the proposed architecture. We compared baseline-2 with the proposed architecture during frequent power failures. Proposed policies such as DBT, WBQ, and replacement policies are not included in the baselines 1,2, and 3 architectures.

4.3.3 Results & Analysis

The proposed architecture is evaluated in this section under stable power and power failures. The proposed architecture is compared with two baseline architectures.

4.3.3.1 Number of Writes to NVM

In order to reduce the system's energy consumption, we first need to reduce the number of writes to both STT-RAM and PCM. We performed experiments to compare the number of writes for NVM in the baseline and the proposed architectures. All values shown in Fig. 4.2 and Fig. 4.4 are normalized with the baseline-1 architecture. All values shown in Fig. 4.3 are normalized with the baseline-3 architecture. Baseline-2 receives fewer writes than the proposed architecture at LLC and PCM by 14.11% and 7.84%, as shown in Fig. 4.2. Compared to baseline-1, the proposed gets 18.97% fewer writes in the LLC and 10.66% in the PCM, as shown in Fig. 4.2.

4.3.3.2 Analysis for Execution Time and Dynamic Energy Consumption & Experiments to Compare the Replacement Policies

Under Stable Power Scenarios: During a power failure, the proposed architecture only backs up the content of DBT and WBQ to LLC.

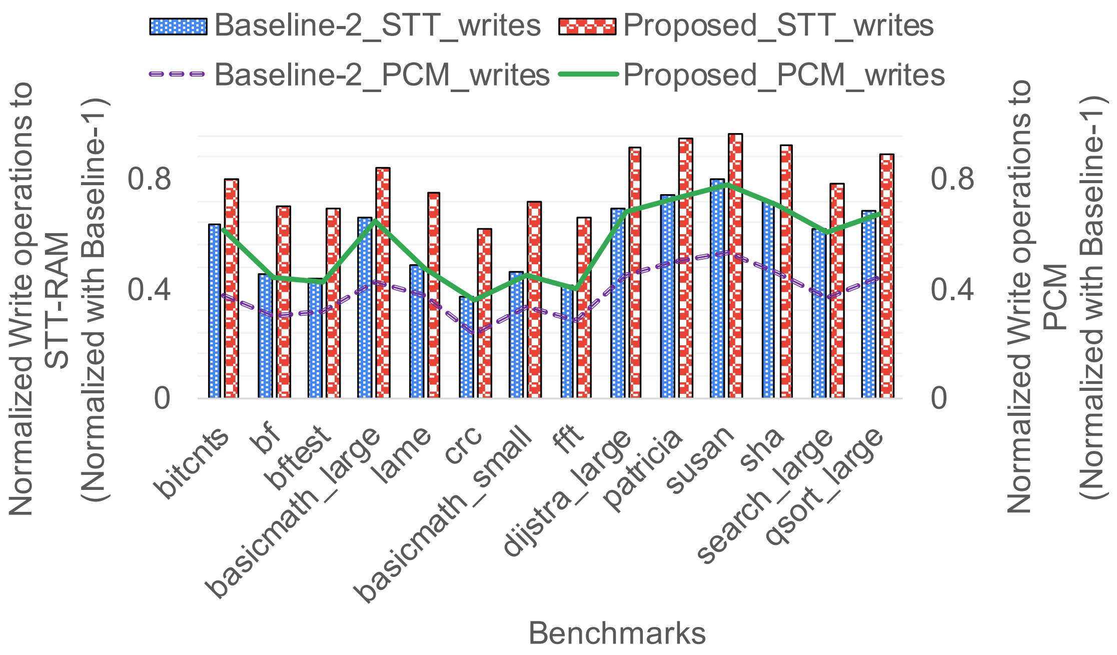


Figure 4.2: Write operations for STT-RAM, PCM under Stable Power.

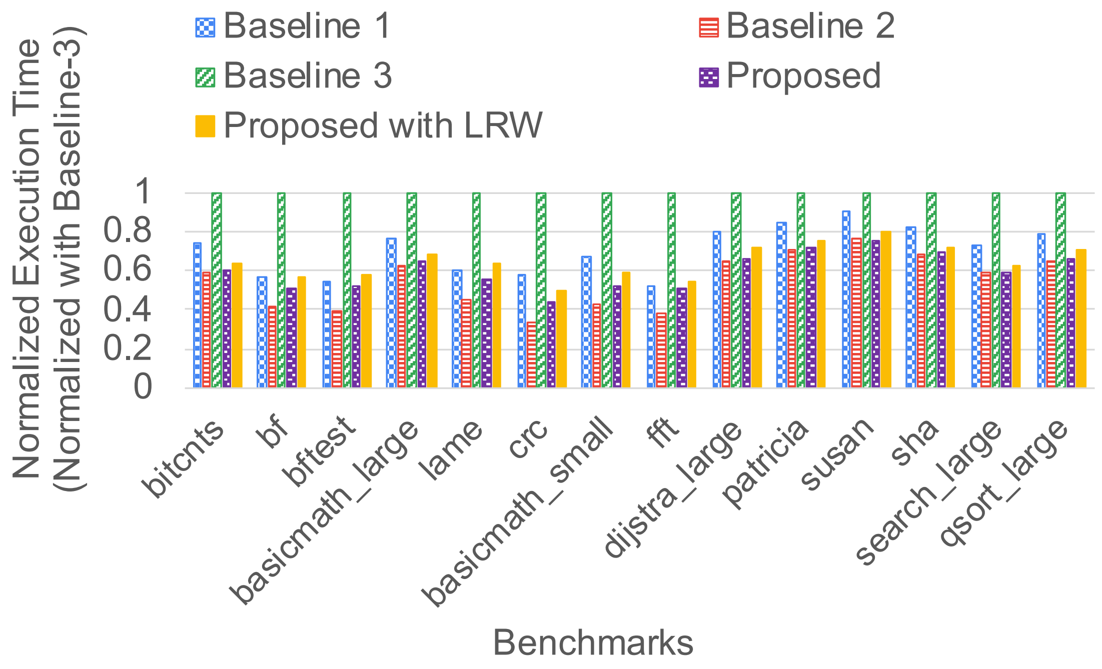


Figure 4.3: Comparisons between Proposed and Baseline Architectures for Execution Time under Stable Power.

The combined size of DBT and WBQ is approximately less than or equal to 4 KB. As a result, we compare the proposed architecture to baseline 3 to see how it performs in normal operations. As shown in Fig. 4.3, architecture-3 performs poorly compared to the proposed and baseline architectures. Compared to baseline-3, the proposed architecture takes 38.79% less execution time during regular operation. Thus, using a small L1 size cache does not improve performance during a stable power supply.

As shown in Fig. 4.3, we compared the architectures with the LRW and LFW policies. As shown in Table 4.3, the proposed architecture uses the LFW replacement policy, and the proposed LRW architecture uses the LRW replacement policy instead of LFW. As shown in Fig. 4.3, the proposed architecture takes 13.11% less execution time than the baseline-1 architecture and 5.10% more execution time than the baseline-2 architecture. The proposed architecture with the LFW policy performs better than the architecture with the LRW policy.

Table 4.3: Overview of the Different Possibilities for the Proposed Architecture that are Used for the Comparisons

Proposed Architecture	**Proposed Base Architecture (L1: SRAM, LLC-STT-RAM, Memory: PCM)**	**DBT & WBQ**	**LFW**	**LRW**	**BR**
Proposed with LRW	✓	✓	✗	✓	✓
Proposed without BR	✓	✓	✓	✗	✗
Proposed with LRW without BR	✓	✓	✗	✓	✗
Proposed Architecture	✓	✓	✓	✗	✓
✓- Supported , ✗- Not Supported					

As shown in Fig. 4.4, the proposed architecture consumes 17.56% less energy than the baseline-1 architecture and 4.93% more energy than the baseline-2 architecture. Under a stable power supply scenario, the proposed architecture consumes less energy than baseline-1 and more energy than baseline-2.

Under Unstable Power Scenarios: We simulate frequent power failures assuming that a power failure occurs every 2 million instructions. The open-source version of the Gem5 core does not model an intermittent power supply processor. By introducing interrupts, we modified gem5 to support intermittent power supply processors. So, for every 2 million instructions, there is an interrupt, which the processor model admits as a power failure. The Gem5 simulator is used to run all the experiments with one billion instructions. We assumed that a power failure occurs every 2 million instructions because, on average, 2 million instructions take approximately 25-30 ms of time to execute. In another way, there is a power interruption every 30 ms, so these power failures are not as frequent as they would be in real life. Therefore, the results are rather conservative.

The proposed architecture outperforms the baseline-2 architecture by 19.61% during

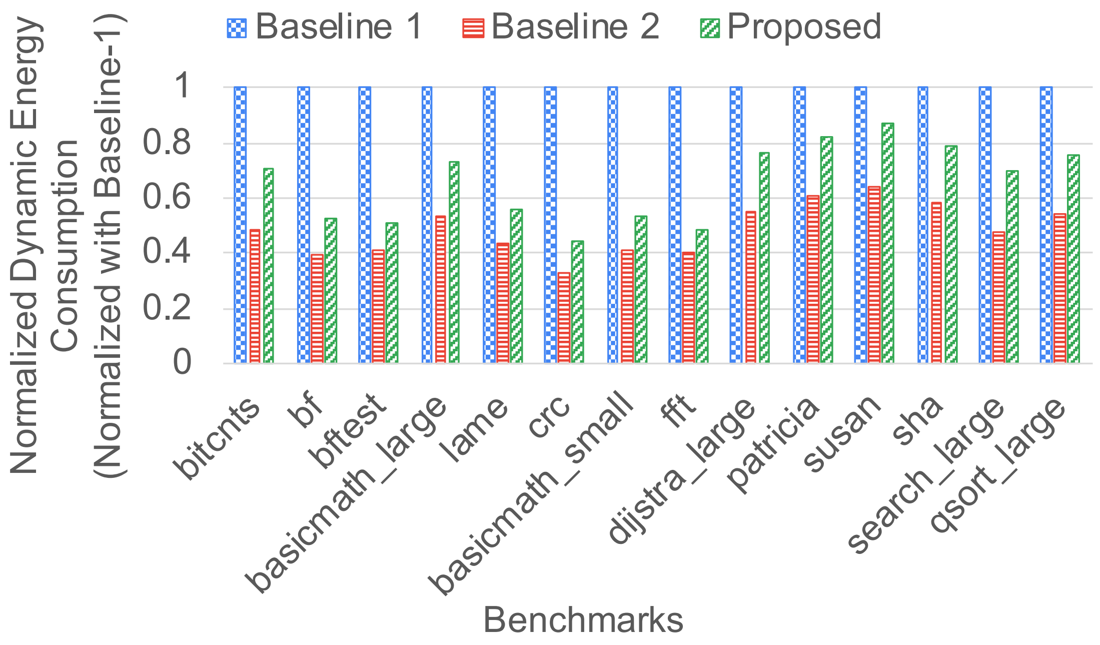


Figure 4.4: Comparisons between Proposed and Baseline Architectures for Dynamic Energy Consumption under Stable Power.

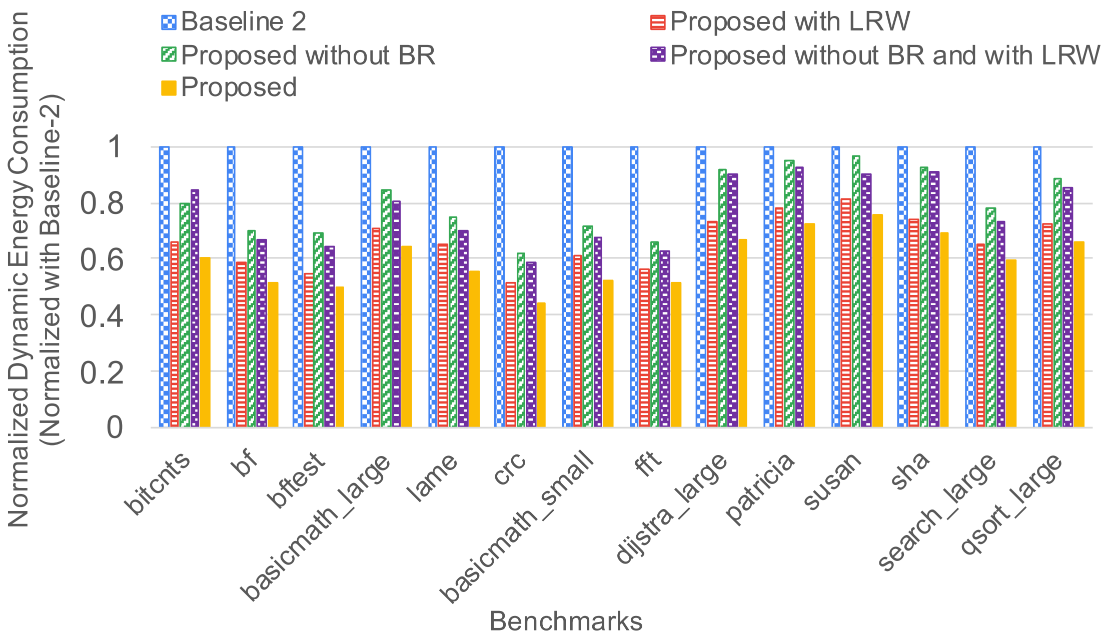


Figure 4.5: Comparisons between Proposed and Baseline Architectures for Energy Consumption under Unstable Power.

frequent power failures for execution time. This advantage is due to DBT and WBQ; most L1 requests result in hits, which reduces performance overhead. Another significant advantage of the proposed architecture is the explored replacement policies. During a stable power supply, the proposed replacement policy helps save important blocks (write-intensive blocks) at the L1 cache rather than evicting them. Thus, incorporating DBT and WBQ improves the proposed architecture's performance under intermittent power.

All values shown in Fig. 4.5 and Fig. 4.6 are normalized with the baseline-2 architecture. The proposed architecture consumes 20.94% less energy than the baseline-2 architecture, as shown in Fig. 4.5. We evaluated the proposed architecture with both replacement policies. The architecture that uses LFW performs better than LRW, as shown in Fig. 4.6.

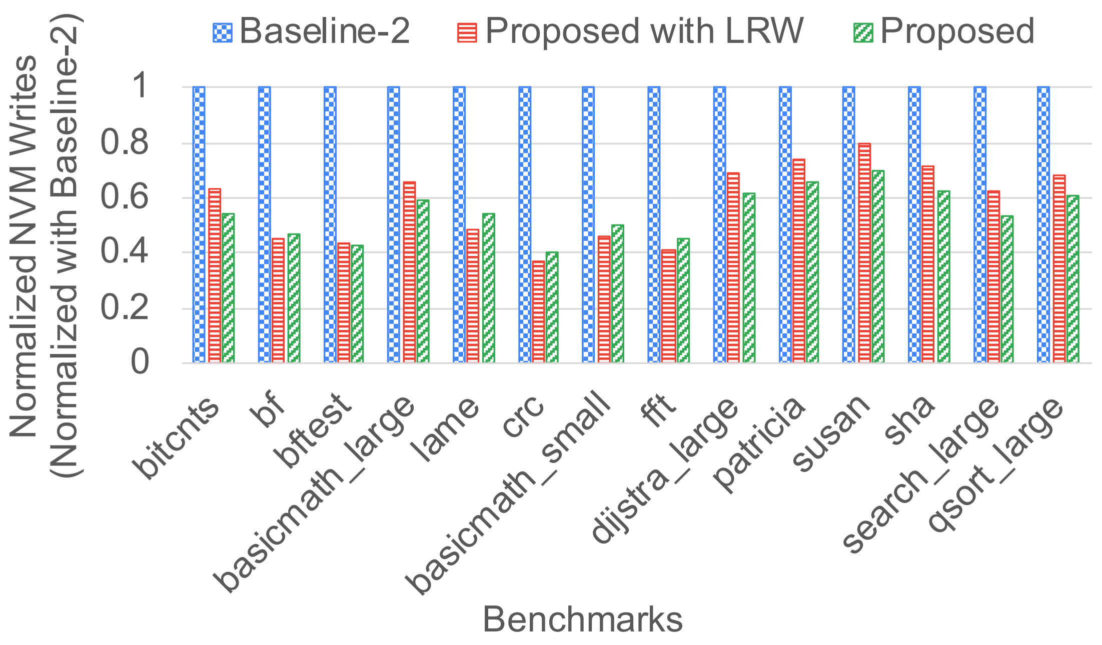


Figure 4.6: Comparisons between LRW and LFW Replacement Policies with Proposed and Baseline Architectures under Unstable Power.

4.3.3.3 Experiments to determine the K, M, and N Values

We performed experiments to determine the K, M, and N values as shown in Fig. 4.7 and Fig. 4.8. Where 'K' is the total number of dirty blocks at any time in the L1 cache. 'K' is denoted as M+N, where M is the total number of entries in DBT and N is the total number of entries in WBQ.

We performed experiments for various K values, as shown in Fig. 4.7. We performed experiments with various K values from K=8 to K=128. For all experiments shown in Fig. 4.7, we assumed that (M, N) was equal. For example, if K=16, M=N=8. As shown in Fig. 4.7, increasing the K value consumes higher energy values. We assume the capacitor energy as input; thus, K is also input to our design based on Equation 3.3. We analyze

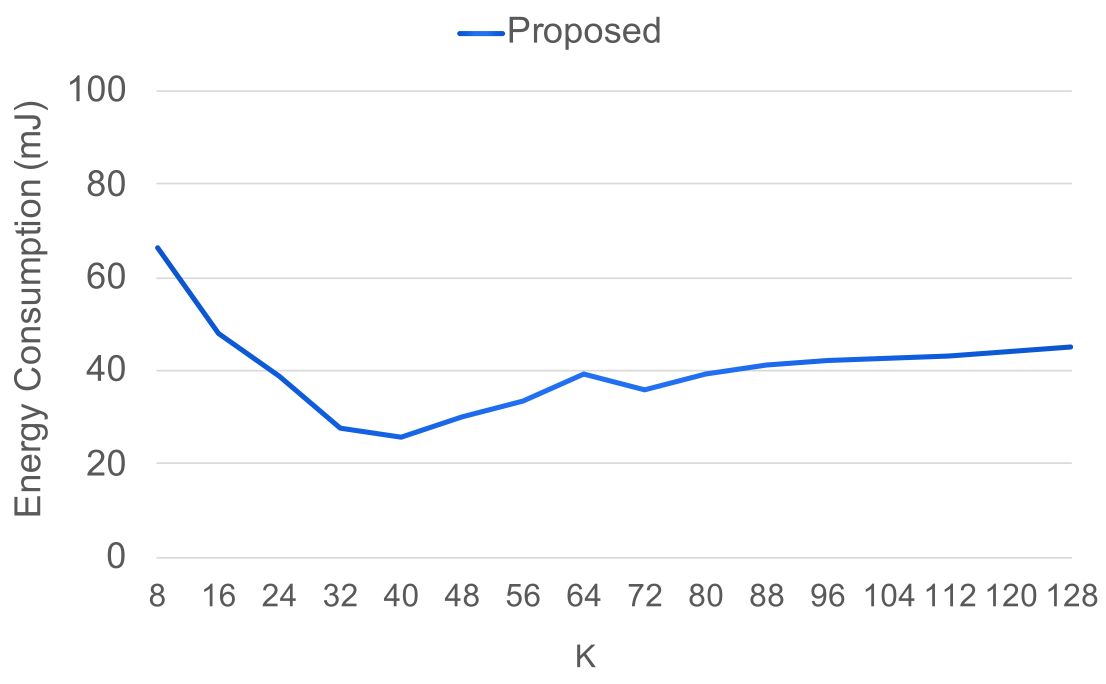


Figure 4.7: Energy Consumption for Different K Values under Stable Power.

various K values because different sizes of capacitors are available on the market. The size of M entirely decides the size of the LRW field. Fig. 4.8 already shows the energy consumption values for various M values. On the basis of the M value, we can set the LRW field's size. If we look at Fig. 4.7, we observe that the system uses less energy up to K = 42, then gradually consumes more energy as K increases.

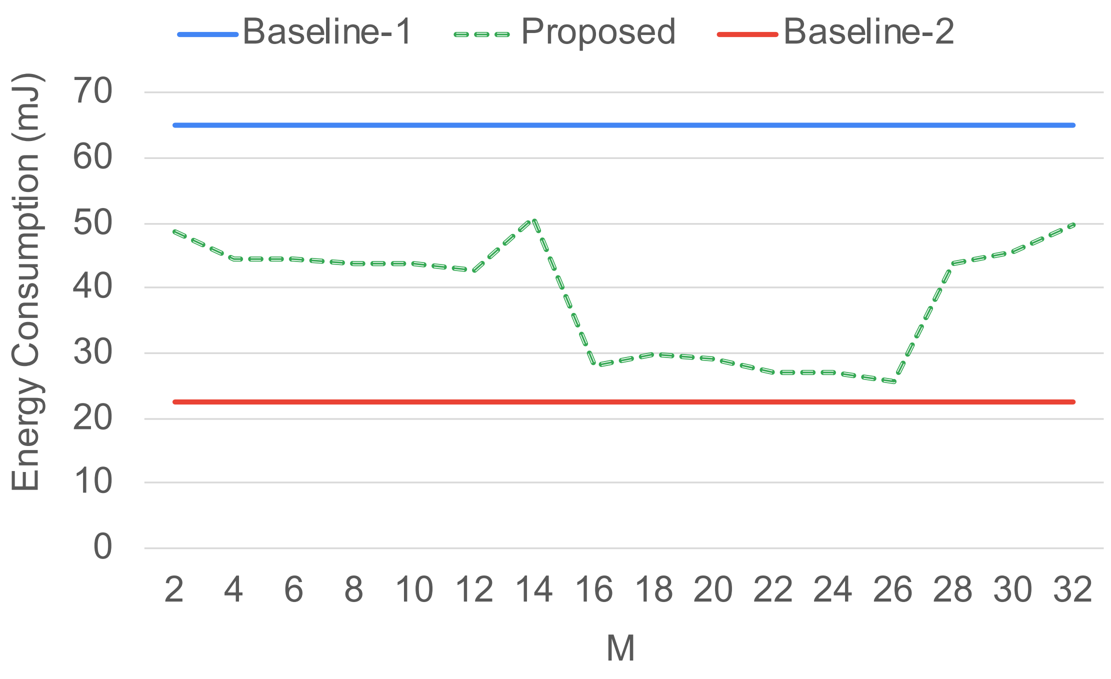


Figure 4.8: Energy Consumption for Different M, N values under Stable Power.

As shown in Fig. 4.8, we used K as 32 in these experiments. Within K = 32, we experimented with various pairs (M, N). In Fig. 4.8, when M=2, N becomes 14 (16-M), and when M=12, N becomes 4. We performed experiments with various possible (M, N) pairs such as (6, 10), (16, 16), (24, 8), and (32, 0). We observe that (26, 6) uses less energy than the other pairs. As a result, for K=32, we used (M, N) as (26, 6) throughout this work. In the same way, we experimented to identify the best pair (M, N) for K = 64. We observe that (54, 10) uses less energy than all other pairs for K=64.

We performed experiments on the same benchmark suite used in section 4.3.1 to determine the K, M, and N values. We selected the K, M, and N values on the basis of the average values. The optimal K, M, and N values vary depending on the application behavior and the system configuration. We also observed that for the selected K, M, and N values, most of the benchmarks (11 out of 14) outperformed others. The variation between the other K, M, and N values and the selected values is negligible for the other three benchmarks. As a result, we selected the values of K, M, and N after considering all the benchmarks. We have shown the overall average values in Fig. 4.7 and Fig. 4.8. Table 4.1 lists the selected values of K, M, and N.

4.3.3.4 Analysis for Dynamic Energy with Unified NVM Architecture

We chose NVM for LLC and main memory based on previous discussions. We use STT-RAM at LLC and PCM at main memory throughout this work. In architecture-1, SRAM is used at L1 and LLC, and PCM is used at the main memory, as shown in Fig. 4.9 (a), i.e., traditional architecture.

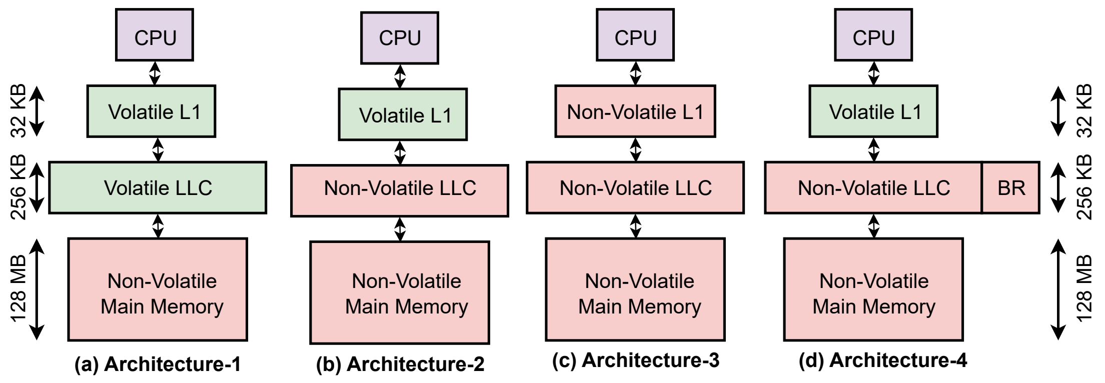


Figure 4.9: Architecture Designs to integrate NVM (a) Introduce NVM at Main-memory, and (b) Introduce NVM at the last-level cache and main-memory levels, (c) Introduce NVM at both the cache levels and main-memory level, and (d) Introduce NVM based Backup-Region (BR) at the last-level cache.

Architecture-2 is shown in Fig. 4.9 (b), SRAM is used at L1, STT-RAM at LLC, and PCM is used at the main memory, i.e., our base architecture. c) STT-RAM at L1, LLC, and PCM is used at the main memory, but it does not consist of any STT-RAM-based backup region. d) SRAM is used at L1, STT-RAM at LLC, and PCM is used at the main memory, but this architecture consists of any STT-RAM-based backup region. These comparisons help to evaluate how bad or good our base architecture is under stable and unstable power scenarios.

Under Stable Power Scenarios: We performed experiments with the unified NVM architecture, i.e., architecture-3. We compared the unified NVM architecture with the proposed architecture to determine how good or bad architecture-3 will perform under a stable power supply. We compare architecture-3 with architectures 1 and 2 for dynamic energy consumption, as shown in Fig. 4.10. Under a stable power supply, architecture-1 outperforms architectures-2, 3, and the proposed architecture. This advantage is due to the use of volatile memory at L1 and LLC in architecture-1.

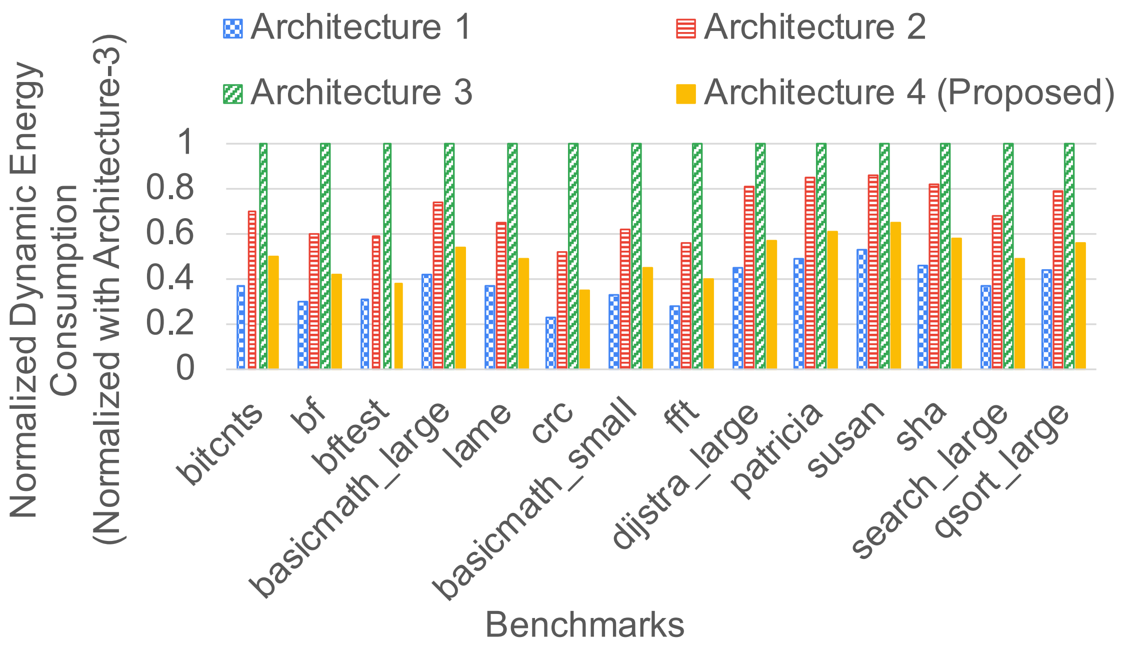


Figure 4.10: Comparisons between Proposed Architecture and Unified NVM-based Architectures for Dynamic Energy Consumption under Stable Power.

We observed that NVM receives more read/write accesses when the entire memory at L1 is NVM-based. Compared to the proposed architecture, architecture-3 consumes 43.25% more energy and has a performance overhead of 38.99% under a stable power supply. This analysis motivated us to compare the proposed architecture with one that uses NVM-based memory for only DBT and WBQ at the L1 cache. The proposed architecture consumes 19.41% less energy than this design.

We performed experiments to compare architecture-3 with the architecture that uses NVM-based DBT and WBQ for the number of NVM writes, as shown in Fig. 4.11. Under stable power, the NVM-based DBT and WBQ design consumes 37.17% fewer writes than architecture-3 and 21.79% more writes than the proposed architecture. As a result, a unified NVM architecture is not a good choice to use for regular operations.

Under Unstable Power Scenarios: We want to examine whether all these benefits are due to PCM at the main memory level. Therefore, we compare the energy consumption for four architectures shown in Fig. 4.12. As shown in Fig. 4.12, architecture-1 consumes

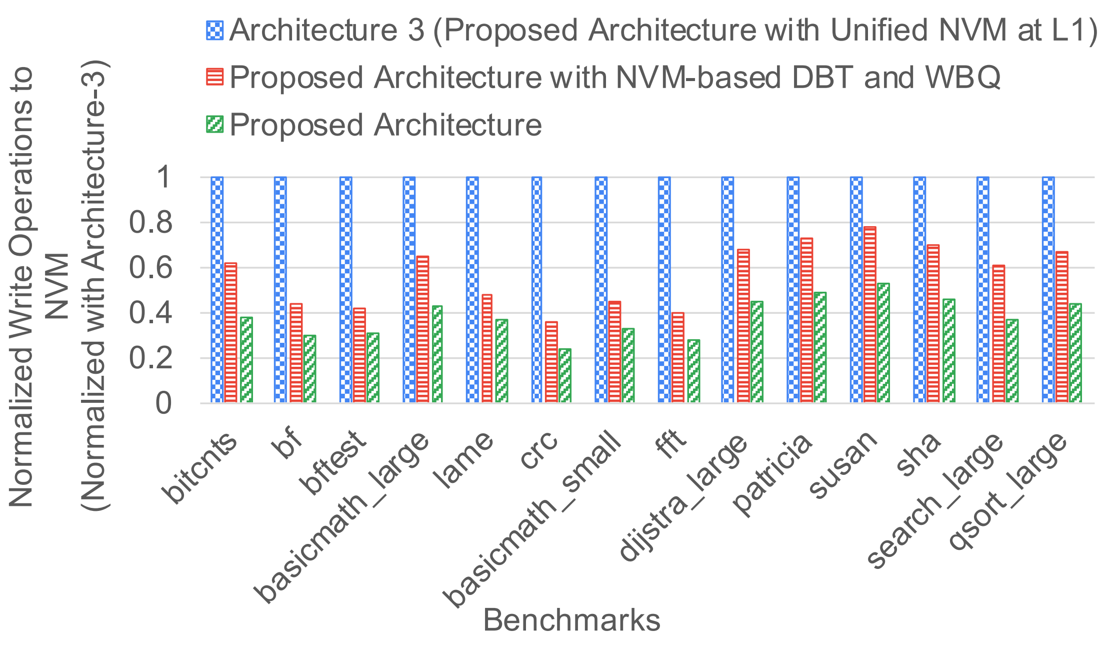


Figure 4.11: Comparisons between Proposed Architecture and Unified NVM-based Architectures for NVM Write operations under Stable Power.

more energy than the other three architectures because the number of writes to PCM is greater in architecture-1. The only difference between architecture-2 and architecture-4 is that architecture-4 is BR-enabled.

The performance of BR-enabled architecture is more effective than that of non-BR-enabled architecture, i.e., architecture-2. With a BR at LLC, we can directly place those K blocks in BR and quickly restore the contents of the L1 cache. When we remove BR from LLC, we must first update LLC. If LLC is full, we need to replace some blocks at LLC to make space for the L1 dirty blocks. We use the LRU replacement policy at LLC, which increases the number of writes to PCM compared to the BR-enabled architecture. We assume that our system has a fixed-energy capacitor that can only backup the K blocks and the register file. When we remove BR from LLC, the capacitor no longer supports safe backup because we have to lose K or $< K$ blocks (either from L1 or LLC). As a result, we either end up with the wrong results or have to restart the application. As illustrated in Fig. 4.5, the BR-enabled architecture consumes less energy than the architecture without BR at LLC. This benefit is due to the increased number of writes to PCM. The proposed architecture consumes less energy than the baseline-2 architecture because frequent power failures increase reads/writes to the NVM, as shown in Fig. 4.5.

Compared to architecture-1, architecture-2 consumes 19.02% less energy, and architecture-4 consumes 32.64% less energy. Fig. 4.12 shows that architecture-4 is better than the other three architectures. Therefore, not all of these improvements are solely related to PCM since our proposed policies also help to achieve better performance and energy.

We performed experiments with the unified NVM architecture, i.e., architecture-3.

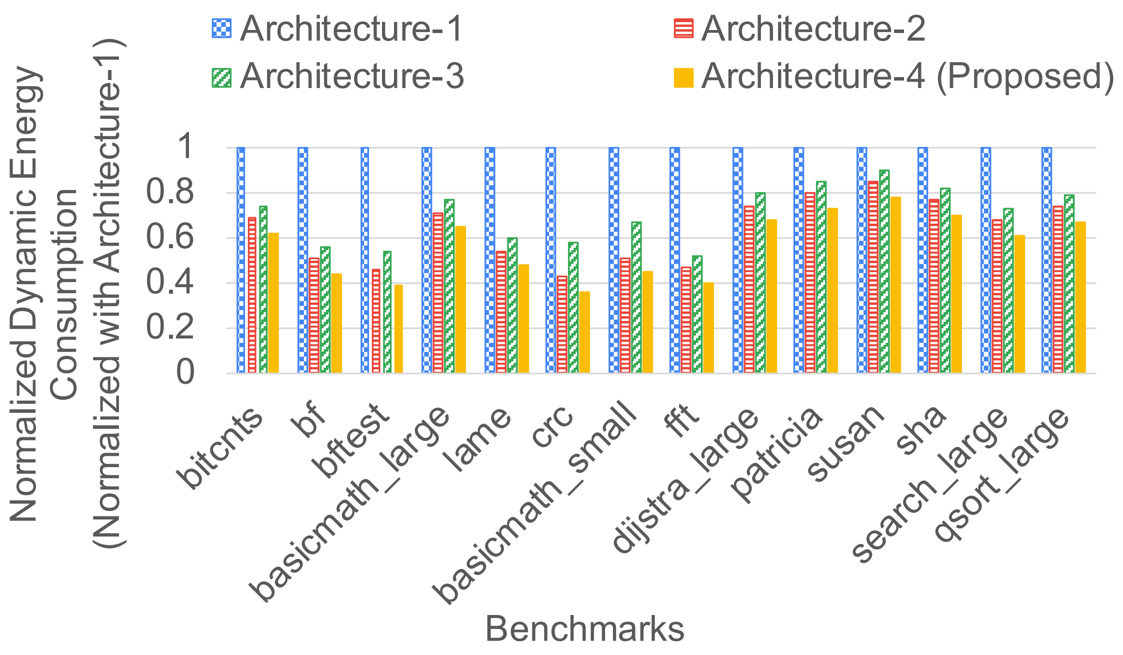


Figure 4.12: Comparisons between Architectures-1, 2, 3, and 4 for Overall Dynamic Energy Consumption under Intermittent Power Supply.

All values shown in Fig. 4.12 are normalized with architecture-1. Under frequent power failures, architecture-3 outperforms architecture-1, but not architecture-2 and the proposed architecture. This advantage for architecture-3 is due to the usage of NVM at L1 and LLC.

Architecture-3 consumes 23.01% more energy than the proposed architecture, i.e., architecture-4, as shown in Fig. 4.12. This benefit for the proposed architecture is because the energy required for backing up the K blocks is less than that of the energy required for architecture-3 during regular operations. Fig. 4.10 and Fig. 4.11 show that architecture-3 consumes more energy and attracts more NVM reads and writes during regular operations. Architecture-3 is not suitable for intermittent power supply scenarios due to these overheads during normal operations. The unified NVM architecture requires additional procedures to make the system more energy efficient.

4.3.4 Analysis for Backup Energy During Intermittent Power Supply

We performed a series of experiments to analyze the backup energy.

During frequent power failures, architecture-3 outperforms architecture-1 and the proposed architecture for backup energy consumption. Compared to architecture-1, architecture-4 consumes 35.57% less energy, as shown in Fig. 4.13. Because we need to backup the entire L1 and LLC to PCM, architecture-1 requires more backup energy than architecture-3. Due to the unified NVM architecture, architecture-3 only needs to backup volatile register contents. Architecture-4 requires constant backup energy because we only need to backup volatile register contents and K blocks to BR during a power failure. Thus, architecture-4 consumes more energy than architecture-3. We have shown the required backup energy for architecture-1 and 4 in Equation 4.1. Architectures 3 and 4 use constant energy to store volatile contents.

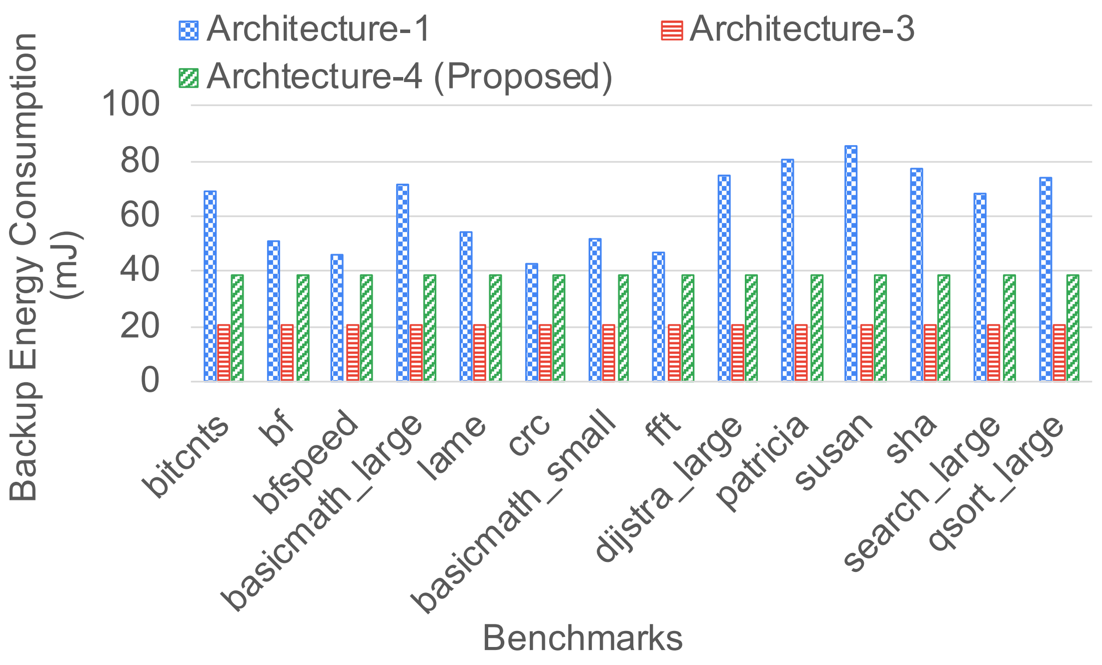


Figure 4.13: Comparisons between Architecture-1, 3, and 4 for Backup Energy Consumption under Unstable Power.

4.3.5 Analysis for Execution Time & Dynamic Energy with Existing Checkpointing Techniques

The proposed backup/restore strategy looks similar to checkpointing. We performed experiments with a stable and an unstable power supply to compare the proposed backup/restore strategy with existing checkpointing approaches. We used four different checkpointing methods for these experiments.

First, we designed a traditional checkpointing technique, introducing a safe point at every 4 million instructions. For every 4 million instructions, we backup the system state to NVM. We restore from the main memory at each safe point and continue with the application's execution. Our proposed architecture outperforms traditional checkpointing because backup occurs only during power failure.

Second, we used a checkpointing method proposed by Xie et al. [32]. Xie et al. identified important volatile blocks that needed to be backed up during a power failure using STT-RAM-based counters (DBCounter and MCounter). Xie et al. used the LRU policy for replacement in NVM-based caches and volatile caches. Updating and accessing these counters is similar to NVM writes and reads, which use more energy and slow down the system. Because Xie et al. perform backup during a power failure, we compared performance and energy consumption based on total NVM reads/writes.

Third, we implemented a checkpointing procedure that creates a checkpoint whenever the system state changes. For every write request, we increment the write counter (WC). Once WC reaches the threshold, the checkpoint procedure is triggered, size of the WC size is the same as the proposed system configuration.

Fourth, we implemented a checkpointing procedure that initiates a backup procedure whenever the system state changes. During a power failure, the checkpoint procedure is triggered and we compare the previous checkpoint data with the new checkpoint data, block by block. Only volatile contents that differ from the previous checkpoint data are backed up. Copy-by-Change checkpointing procedures are proposed by S. Ahmed et al. [157, 158].

Under Stable Power Scenarios: We used the first three checkpointing methods for these experiments to compare with the proposed architecture under a stable power supply.

All values shown in Fig. 4.14 and Fig. 4.15 are normalized with the traditional checkpointing technique. The proposed architecture reduces performance overhead and energy consumption by 41.27% and 37.95% compared to the traditional checkpointing technique, the proposed architecture reduces performance overhead and energy by 19.03% and 14.72% compared to the Xie et al. checkpointing technique, and the proposed architecture reduces performance overhead and energy consumption by 39. 11% and 33. 10% compared to the third checkpointing technique as shown in Fig. 4.14 and Fig. 4.15.

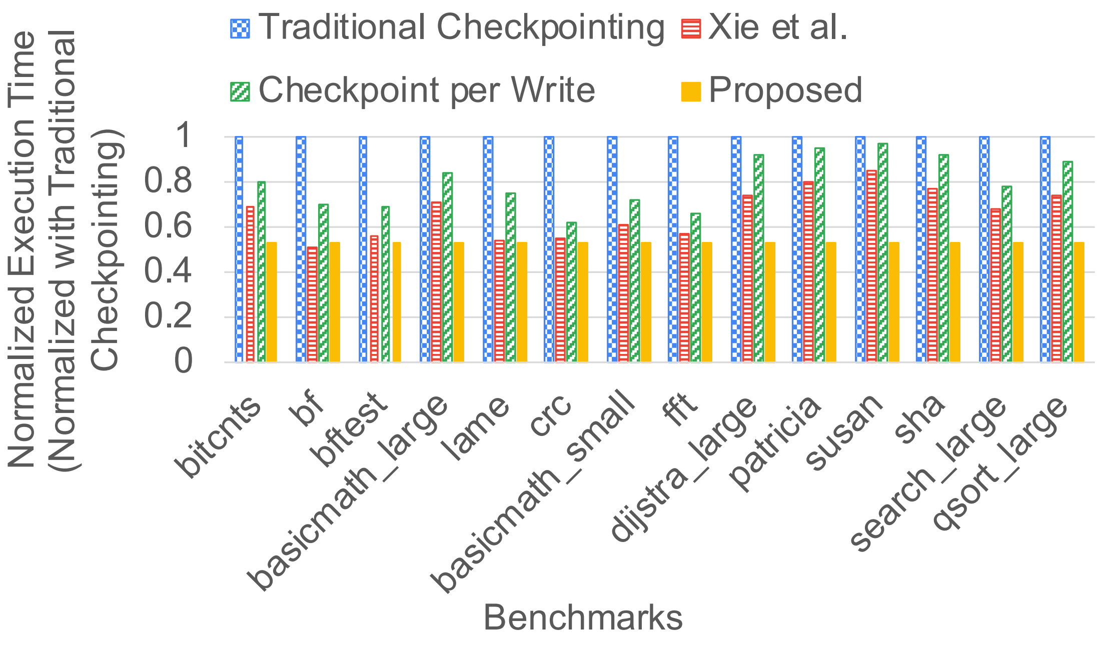


Figure 4.14: Comparisons between Proposed Backup and Different Existing Checkpointing Techniques for Execution Time under Stable Power.

The traditional checkpointing technique is the standard and the worst-case scenario for these intermittently powered systems. If we want to add another level of filtering checkpoints, we can add a counter per cache block, and if any cache block reaches the defined threshold, the checkpoint procedure is triggered. This checkpointing technique

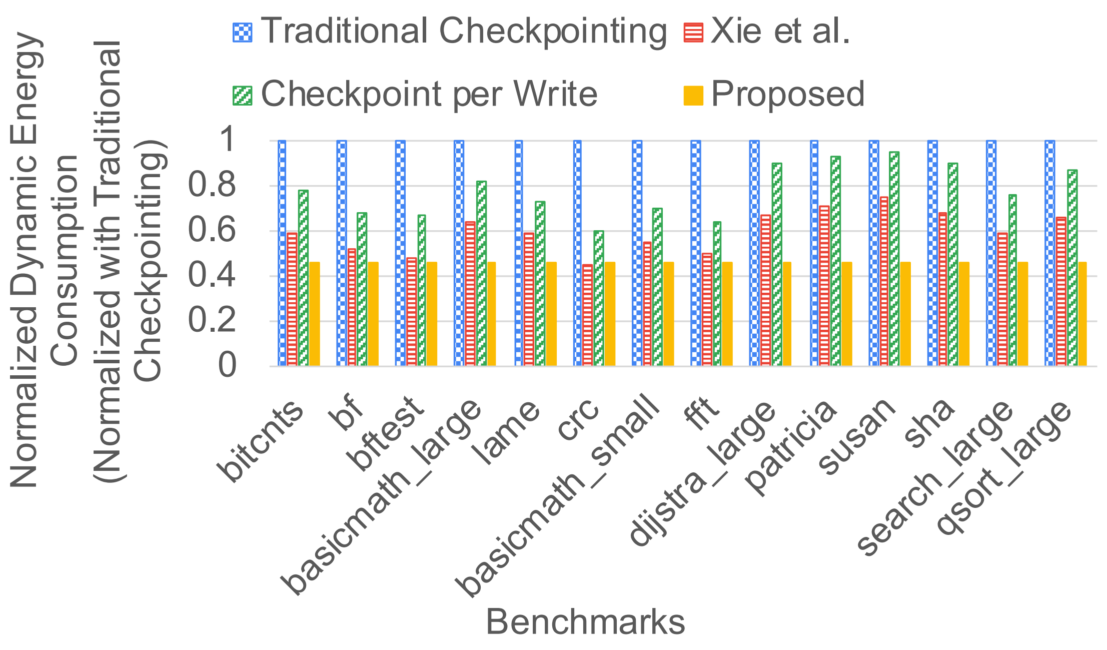


Figure 4.15: Comparisons between Proposed Backup and Different Existing Checkpointing Techniques for Energy Consumption under Stable Power.

performs better than traditional checkpointing. However, the second checkpointing technique increases the number of NVM accesses. Instead of checkpointing for every safe point or for every $> WC$ case, we can add another level to reduce checkpoints by placing a checkpoint only during a power failure. Xie et al. proposed a checkpointing policy that only backups the dirty blocks selected during a power failure. Xie et al. outperform the other two checkpointing policies. However, the proposed backup strategy reduces performance and energy overhead compared to Xie et al., as shown in Fig. 4.14 and Fig. 4.15.

Under Unstable Power Scenarios: We used all four checkpointing methods for these experiments to compare with the proposed architecture under an unstable power supply.

All values shown in Fig. 4.16 and Fig. 4.17 are normalized with the traditional checkpointing technique. For example, suppose that an unexpected power failure occurs at the $7th$ million instruction. We re-execute the application from the $4th$ million instruction because it is the closest safe point. These unnecessary executions increase NVM writes/reads, consume more energy, and degrade system performance. Compared to the traditional checkpointing technique, the proposed architecture reduces performance overhead and energy consumption by 48.70% and 40.19%, as shown in Fig. 4.16 and Fig. 4.17.

Compared to Xie et al., the proposed architecture reduces performance overhead and energy consumption by 27.99% and 20.07%, as shown in Fig. 4.16 and Fig. 4.17. This advantage is due to the selection of an LRU-based replacement policy in Xie et al. work for updating the counters, which we observed as insufficient to reduce the number of writes

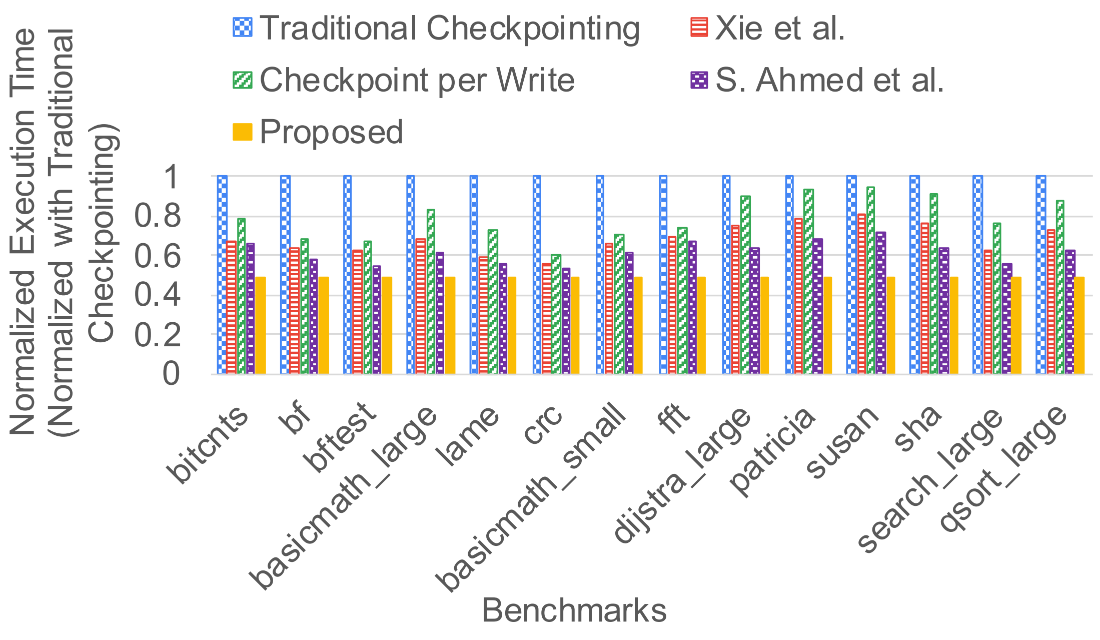


Figure 4.16: Comparisons between Proposed Backup and Different Existing Checkpointing Techniques for Execution Time under Unstable Power.

to NVM. Another reason is that Xie et al. did not backup the entire dirty block to NVM, which requires the application to be re-executed, which consumes more energy during frequent power failures.

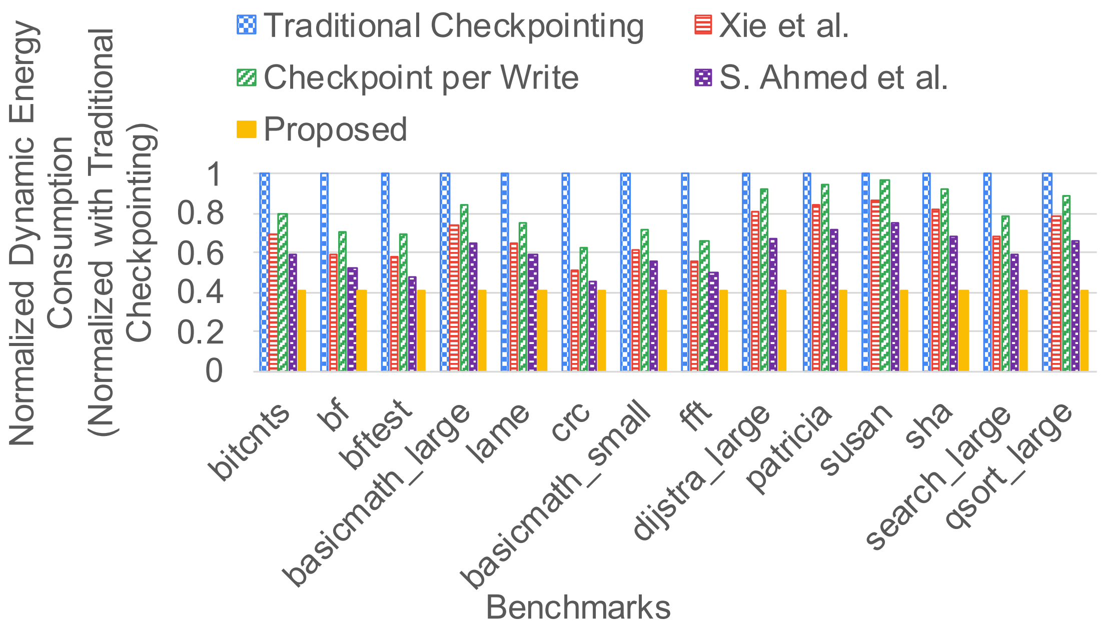


Figure 4.17: Comparisons between Proposed Backup and Different Existing Checkpointing Techniques for Energy Consumption under Unstable Power.

Compared with the third checkpointing technique, the proposed architecture reduces performance overhead and energy consumption by 38.71% and 32.13%, as shown in Fig. 4.16 and Fig. 4.17. During frequent power failures, backup/restore sizes increase in this technique, potentially causing NVM reads and writes to increase, which degrades system performance and consumes more energy. The proposed architecture reduces performance overhead and energy consumption by 21.93% and 16.55% compared to S. Ahmed et al. checkpointing policy, as shown in Fig. 4.16 and Fig. 4.17.

4.3.6 Analysis for Multi-Cache Levels and Multi-Core Designs

The proposed architecture consists of an SRAM-based L1 cache and an STT-RAM-based LLC. We found two possibilities if we want to add one or two levels of cache to the proposed architecture.

First, suppose that we introduce one or two levels of STT-RAM-based caches. In that case, our proposed architecture has no impact or complications with this design because we proposed a backup strategy that can only track and backup K dirty blocks from the L1/L2/L3 caches. Everything is the same as in the proposed architecture during a stable power supply. We don't need a backup of L2/L3 during frequent power failures because these are already NVM-based caches. According to our proposed backup policy, we must back up the contents of the registers and the L1 dirty contents to LLC. We must track the number of dirty block contents in the L1 cache. This appears to be similar to our proposed architecture and techniques. As a result, our proposed architecture does not require additional methods and data structures for the first possibility.

Second, imagine that we have one or two levels of SRAM-based caches. In that case, our proposed architecture makes a difference. This design is complicated because it is necessary to limit the total number of dirty blocks from all levels of SRAM-based caches, i.e., from L1/L2/L3 caches to K at any time. This issue requires several changes to the proposed architecture for tracking and maintaining K dirty data at each level. If the processor is multi-core, the problem becomes even more complex and introduces new complications, such as cache coherence issues. Selecting volatile data and backing it up to NVM requires significant energy and additional techniques during frequent power failures. These extra procedures add various overheads, such as size and performance. However, these complex systems are not required for embedded devices or applications.

NVM-enabled microcontrollers do not contain a second-level cache. Texas Instruments (TI)-based NVPs, such as the MSP430FR6989 and MSP430F5529, do not have a cache and contain only main memory. The main memory of the MSP430FR6989 contains 2KB of SRAM and 128KB of FRAM, while the MSP430F5529 contains SRAM and flash. For these intermittently powered IoT systems to run embedded applications, we don't need a multi-core processor with complex cache hierarchies; instead, one/two levels of caches and a 1-core processor are sufficient. As a result, in this work, we did not examine the proposed architecture for higher levels of cache or multi-core processors.

4.3.7 Exploring Design Space for Various Parameter Combinations

In this section, we evaluate the proposed architecture for various combinations of parameters. The parameters in our work are the cache size, main memory size, and sizes of K, M, N, and WC. We intend to allow the end user to choose a suitable architecture with an appropriate parameter size.

As shown in Table 4.1 and Table 3.2, we used the same experimental setup and latency/energy values for NVM. Our design depends on the number of power failures that occur. We performed experiments to observe the energy gains when the number of power failures increased from 500 to 1000 and decreased from 500 to 200. Usually, we introduce a power failure for every 2 million instructions, which means that by executing 1 billion instructions, we experience 500 power failures. We introduce a power failure for every 1 million instructions to increase the number of failures from 500 to 1000. Similarly, if we introduce a power failure every 5 million instructions, the total number of power failures becomes 200.

So, our design space depends on (cache_size, Memory_size, K, M, N, WC, Replacement_policies, BR, number of power failures). If we fix cache, memory, and WC sizes w.r.t our experimental setup, our design space becomes large enough. Let us take an instance to see how large our design space will become. For K = 16, the possible K values are 16. If K = 16, possible pairs (M, N) are 16 * 16. For BR, possible conditions are 2 (BR-enabled or not). For replacement policies, the possible ways are 2 (LRW or LFW). The number of power failures; the possible cases are 4 (as per the experiments we performed) if we combine all these parameters to calculate the design space size (16*16*16*2*2*4) equals 65,536 design choices for the given cache size, main memory size, WC and K = 16.

We can notice from Table 4.4 that only one design choice performs better in all the given combinations. For the given cache size, the main memory size, and K = 16, we find that the combination of M = 12, N = 4, WC = 6, enabled by BR, and the number of power failures=1000 is preferable to all other 65,536 design choices.

The limitations that we observed in this chapter are as follows: (1) This chapter limits the NVM at the LLC and main memory levels; and (2) This work does not predict the power failures beforehand, which means that the proposed architecture is not able to identify the power failure in advance and what if the energy in a capacitor is not full and cannot perform the safe backup during power failures.

4.4 Summary

In this chapter, we describe an NVM-based architecture. Using the proposed DBT and WBQ, we see fewer writes to STT-RAM (LLC) and PCM (main memory). The proposed architecture decreases STT-RAM writes by 18.97% and PCM writes by 10.66% compared to the baseline-1 architecture. As a result, we have reduced energy consumption by approximately 17.56%. However, the proposed architecture has 5.10% execution

Table 4.4: Comparison of Different Possibilities for a given Cache size(L1/LLC) as 32KB/256KB and Main memory size as 64MB

Cache size (L1/LLC)	**Main Memory size**	**K**	**M**	**N**	**WC**	**LRW/LFW**	**BR-enabled**	**Number of power failures**	**Energy Gain (%)**	**Compared with**
32KB/256KB	64MB	8	6	2	6	LFW	Yes	500	13.60	Baseline-2
		8	6	2	0	LRW	Yes	500	12.35	Baseline-2
		8	6	2	6	LFW	No	500	7.15	Baseline-2
		8	6	2	6	LFW	Yes	200	11.76	Baseline-2
		8	6	2	6	LFW	No	200	6.84	Baseline-2
		8	6	2	6	LFW	Yes	1000	14.32	Baseline-2
		8	6	2	6	LFW	No	1000	7.58	Baseline-2
		16	8	8	6	LFW	Yes	500	15.37	Baseline-2
		16	12	4	0	LRW	Yes	500	15.56	Baseline-2
		16	12	4	6	LFW	No	500	8.230	Baseline-2
		16	12	4	6	LFW	Yes	200	15.64	Baseline-2
		16	**12**	**4**	**6**	**LFW**	**Yes**	**1000**	**18.04**	**Baseline-2**
		16	8	8	6	LFW	Yes	200	14.45	Baseline-2
		16	8	8	6	LFW	No	1000	9.43	Baseline-2
		16	12	4	0	LRW	No	200	8.60	Baseline-2
		16	12	4	0	LRW	Yes	1000	16.63	Baseline-2

overhead and 4.93% energy overhead compared to the baseline-2 architecture under a stable power supply. We also compared the existing checkpointing policies with the proposed architecture. We introduced an STT-RAM-based backup region at LLC that helps for backup from L1 during a power failure. We also evaluated and analyzed the unified NVM architecture with the proposed architecture. We explored various design spaces to determine how our proposed architecture behaves when changing parameter sizes.

Until now, we have only explored the challenges and the proposed policies for NVM-based caches, either at L1 or LLC. Therefore, we also explore NVMs at the main memory level by using the real-time micro-controllers that consist of NVM at the main memory level in the next chapter.

Chapter 5

An Energy Efficient Memory Mapping Technique for NVM based Hybrid Main Memories

This chapter explores an ILP-based energy-efficient memory mapping technique for NVM-based hybrid main memories. Our proposed technique gives an optimal mapping choice that reduces the system's Energy-Delay Product (EDP). We validated our system using TI-based MSP430FR6989 and MSP430F5529 development boards. TI-based MSP430FR6989 has a small SRAM and large non-volatile-based main memory, i.e., FRAM. To make the system energy efficient, we need to use SRAM efficiently. So, we must select some portions of the application and map them to SRAM or FRAM. This chapter is organized as follows. The introduction is provided in Section 5.1. Section 5.2 explains the system model and gives an overview of the problem definition. We discuss the proposed work in Section 5.3, which explains the proposed ILP-based memory mapping technique and framework that supports intermittent computing. The experimental setup and results are described in Section 5.4. We write the summary in Section 5.5.

5.1 Introduction

For intermittently powered IoT devices, energy harvesting is the main source of energy. Energy-harvesting sources such as piezoelectric materials and radio-frequency devices extract a small amount of energy from their surroundings. We must use energy efficiently in both stable and unstable power supply scenarios.

In order to utilize energy efficiently and to make the system energy efficient, we primarily have two choices. The first choice is to reduce energy consumption by proposing new techniques that use energy efficiently. The second choice is to increase the number of different energy harvesters, which will accumulate more energy while increasing maintenance costs. We need to maintain these many energy harvesters, which is not a feasible solution. Thus, our main concern is to reduce energy consumption by proposing new techniques that help to design an energy-efficient system.

Gonzalez et al. [205] mentioned energy as not an ideal metric to evaluate the efficiency of the system. By simply reducing the supply voltage or load capacitance, energy can be reduced. Instead of using energy as a metric, they suggested using the Energy-Delay Product (EDP) as the energy-efficient design metric. The EDP considers both performance and energy simultaneously in a design. If a design minimizes the EDP, we can call such a design energy-efficient. We define EDP in Equation 5.1.

$$EDP = E_{system} \times Num\_cycles \tag{5.1}$$

Where E_{system} is the energy consumption of the system, Num_cycles is the number of CPU cycles.

During these frequent power failures, executing IoT applications becomes more difficult because all computed data may be lost, and the application's execution must restart from the beginning. During power failures, we need an additional procedure to backup/checkpoint the volatile memory contents to NVM.

Flash memory was the prior NVM technology used by modern microcontrollers at the main memory level, such as MSP430F5529 [181]. Flash is ineffective for frequent backups and checkpointing because its erase/write operations require a lot of energy. Emerging NVMs outperform flash, including STT-RAM [206, 22], PCM [207], Re-RAM, and FRAM [11]. Previous work has been demonstrated by incorporating these emerging NVMs into low-power-based microcontrollers (MCUs) [11, 181, 154]. Recent NVM-based MCUs, such as the flash-based MSP430F5529 and the FRAM-based MSP430FR6989, encourage the use of hybrid main memory. The flash-based MCU, MSP430F5529, is made up of SRAM and flash, while the FRAM-based MCU, MSP430FR6989, is made up of SRAM and FRAM at the main memory level. The challenges associated with hybrid main memory-based architectures, such as MSP430FR6989, are as follows.

1. FRAM consumes 2x more energy and latency than SRAM. This design degrades system performance and consumes extra energy even during normal operations.

2. SRAM loses contents during a power failure and needs to run the application from the beginning, which consumes extra energy and time. For large-size applications, this design will not be helpful. Anyway, using only SRAM performs better during regular operations.

3. We can design a hybrid main memory to get the benefits of both SRAM and FRAM. The following questions must be answered and analyzed to use the hybrid main memory design.

    (a) How do we choose the appropriate sections of a program and map them to either SRAM or FRAM regions? A significant challenge is mapping a program's stack, code, and data sections to either SRAM or FRAM.

    (b) How and where should volatile contents be backed up to the NVM region during frequent power failures?

The main question is which section of an application should be placed in which memory region; this is essentially a memory mapping problem. In this chapter, we address challenge-1 and 5 described in chapter 2, and we achieve objective 3 discussed in chapter

1 by proposing an energy-efficient memory mapping technique. Concerning all of the challenges mentioned above, we briefly review our contributions in this chapter as follows.

- To the best of our knowledge, this is the first work on the integer linear programming (ILP)-based memory mapping technique for intermittently powered IoT devices.
- We incorporated energy harvesting scenarios into the ILP model such that the frequency of power failures is considered as an input for our ILP model.
- We formulated the memory mapping problem to cover all the possible design choices. We also formulated our problem in such a way that it supports large-size applications.
- We proposed a framework that efficiently consumes low energy during regular operation and frequent power failures. Our proposed framework supports intermittent computing.
- We evaluated the proposed techniques and frameworks on actual hardware boards.

5.2 System model and Problem Definition

This section discusses the system model for embedded MCUs and defines the mapping problem for these MCUs.

5.2.1 System Model

We consider a simple, customized RISC instruction set with a Von Neumann architecture, where the instructions and data share the same address space that supports at least 16-bit addressing. Base architecture does not have a cache to avoid uncertainty. To make things simple, we assume a single-cycle execution of the processor. The base architecture has a small SRAM memory and a larger NVM.

The MSP430 is an example of such a processor. Non-volatile memory sizes range from 1 kilobyte (KB) to 256 KB, while volatile RAM sizes range from 256 bytes to 2KB. Both SRAM and NVM can be accessed by instructions using a compiler/linker script. We can modify the linker script to map memory according to the memory ranges specified by the user. MSP430 does not have any operating system.

5.2.2 Problem Definition

Definition 4.1: **Optimal Memory Mapping Problem**: Given a program that consists of various functions and global variables, sizes of SRAM and FRAM, the number of reads and writes for each function/variable, frequency and duration of power failures, and the energy required per read/write to the SRAM/FRAM. What is the optimal memory mapping for these functions/variables in order to reduce the system's EDP?

The inputs are : The number of functions; the number of global variables; energy per write to SRAM and FRAM; energy per read to SRAM and FRAM; SRAM and FRAM sizes; Number of CPU cycles per function; the number of reads; the number of writes; frequency and duration of power failures.

The output is: Mapping information for all functions and global variables, under which the system's EDP is minimized.

Definition 4.2: **Support for Intermittent Computing**: During power failures, we must safely backup volatile contents to NVM. As stated previously, we must use SRAM efficiently for energy savings; but again, how can we save the contents of SRAM? There are two significant issues with intermittent computation. First, during a power failure, all SRAM's mapping information and register contents are lost, causing the system to become inconsistent. Second, how do we backup/restore the mapping information and register contents to ensure system consistency?

Our first objective is to minimize the energy of the overall system and the EDP of the system. We need to support the proposed system even during frequent power failures. Our second objective is to maximize the execution progress of the application during frequent power failures. The progress of the application is a function of both the execution time and the frequency of power failures (η), as shown in equation 5.2.

$$\text{Progress } = F\ (NC_{Execute} * \eta) \tag{5.2}$$

We define the frequency of power failures in equation 5.3. It is the ratio between the time consumed during regular execution without any power failures to the time consumed during power failures, where $NC_{Execute}$ is the number of cycles required to execute the application during regular operation.

$$\eta = \frac{NC_{Execute}}{NC_{Intermittent}} \tag{5.3}$$

We define the number of cycles required during power failures in equation 5.4. We need to perform the backup and restore operations during a power failure.

$$NC_{Intermittent} = NC_{Backup} + NC_{Execute} + NC_{Restore} \tag{5.4}$$

Where NC_{Backup} is the number of cycles required for the backup operation and $NC_{Restore}$ is the number of cycles required for the restore operation.

5.3 Proposed Memory Mapping Technique

In this section, we discuss the details of the proposed mapping technique. Our main objective is to pick the optimal mapping choice among all the design choices, which reduces the system's EDP. To achieve this, we proposed an ILP-based mapping technique [35]. The overview of the proposed mapping technique is shown in Fig. 5.1. We also discuss how we support intermittent computing for these MCUs.

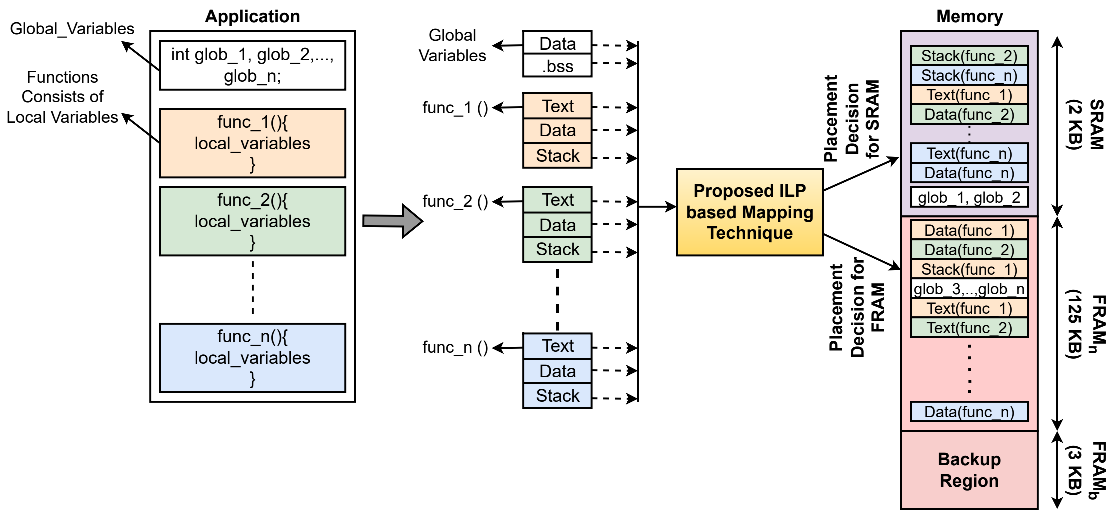


Figure 5.1: Overview of the proposed memory mappings in MSP430FR6989

5.3.1 ILP Formulation for Data Mapping for Intermittent Computing

We present the ILP formulation for the memory mapping problem mentioned in definition 4.1. We divide this ILP formulation into two parts, one for global variables and the second for the functions. We have shown the overview block diagram of the proposed ILP framework in Fig. 5.2. During the profiling and characterization process, we consider the branch instructions and their behavior from the generated assembly code.

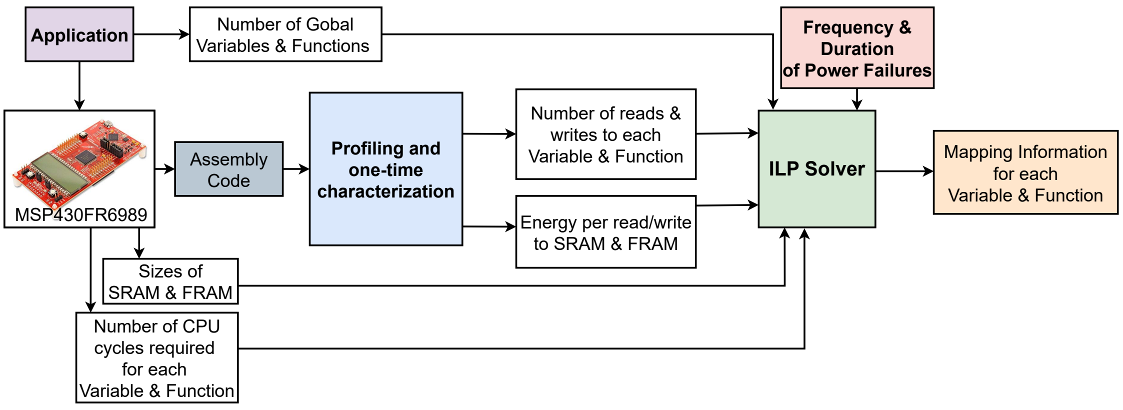

Application
Number of Gobal Variables & Functions
Frequency & Duration of Power Failures
MSP430FR6989
Assembly Code
Profiling and one-time characterization
Number of reads & writes to each Variable & Function
Energy per read/write to SRAM & FRAM
ILP Solver
Mapping Information for each Variable & Function
Sizes of SRAM & FRAM
Number of CPU cycles required for each Variable & Function

Figure 5.2: Overview of the Proposed ILP Framework

For Global Variables: Let the number of global variables in a program be 'G'. Let the number of reads and writes to the variable 'i' be r_i and w_i. We divided FRAM's 128 KB into two regions, i.e., $FRAM_n$ and $FRAM_b$, $FRAM_n$ memory region has 125 KB, and the $FRAM_b$ memory region has 3 KB.

We have two memory regions represented as Mem_j as shown in Equation 5.5; When j = 1, we select the memory region as SRAM, and we use $FRAM_n$ for j=2.

$$Mem_j = \begin{cases} j = 1 & ;\text{SRAM} \\ j = 2 & ;FRAM_n \end{cases} \tag{5.5}$$

Let the SRAM / FRAM sizes be $Size(Mem_j)$ as shown in Equation 5.6; When j = 1, we refer to the SRAM memory size in bytes, and when j = 2, we refer to the $FRAM_n$ memory size in bytes.

$$Size(Mem_j) = \begin{cases} j = 1 & ;\text{SRAM} \\ j = 2 & ;FRAM_n \end{cases} \tag{5.6}$$

Let the energy required for each read/write to Mem_j be E_{r_j} and E_{w_j}. Let the number of CPU cycles required to execute a global variable v_i be NC_{v_i}, where $\forall i \in [1, G])$. Using one-time characterization and static profiling, we gathered data such as per read/write energy to SRAM/FRAM and the number of cycles.

We define a binary variable (BV); $I_j\,(v_i)$, which refers to a variable v_i allocated to the memory region j. If $I_j\,(v_i)$=1 then the variable v_i is allocated and $I_j\,(v_i)$=0 indicates that the variable v_i is not allocated. $I_j\,(v_i)$, where $(\forall j \in [1, Mem_j], \forall i \in [1, G])$ is defined as shown in Equation 5.7.

$$I_j\,(v_i) = \begin{cases} 1 & v_i \text{ is allocated to memory region } j \\ 0 & \text{otherwise} \end{cases} \tag{5.7}$$

Constraints: There are two constraints, one is for BV; $I_j\,(v_i)$ and one is a memory size constraint. In any case, a variable v_i is allocated to only one memory region, which means that v_i is allocated to either SRAM or FRAM but not both. This constraint is defined in Equation 5.8.

$$\sum_{j=1}^{Mem_j} I_j\,(v_i) = 1 \quad (\forall i \in [1, G]) \tag{5.8}$$

The other constraint is related to the size of the memory. The allocated variables v_i and its $Size(v_i)$; $\forall i \in [1, G])$ should not be greater than $Size(Mem_j)$. This constraint is defined in Equation 5.9.

$$\sum_{i=1}^{G} I_j\,(v_i) * Size(v_i) \leq Size(Mem_j) \quad (\forall j \in [1, Mem_j]) \tag{5.9}$$

Objective 4.1: The challenge of mapping global variables in a program to SRAM or FRAM is to reduce EDP and improve system performance. E_{global} is defined in Equation 5.10. Here E_{global} is the energy required to allocate global variables to SRAM or FRAM and execute those from their respective memory regions.

$$E_{global} = \sum_{j=1}^{Mem_j} \sum_{i=1}^{G} [E_{r\_j} \times r_i + E_{w\_j} \times w_i] \tag{5.10}$$

EDP_{global} is defined in Equation 5.11. Where EDP_{global} is the energy-delay product required to allocate global variables to SRAM or FRAM.

$$EDP_{global} = \sum_{j=1}^{Mem_j} \sum_{i=1}^{G} I_j\left(v_i\right)\left[E_{global} \times NC_{v_i}\right] \tag{5.11}$$

For Functions: Let the number of functions in a program be 'N_f'. Let the number of reads and writes to i^{th} function be $r(F_i)$ and $w(F_i)$, where $\forall i \in [1, N_f]$. The functions consist of procedural parameters, local variables, and return variables. Internally, the code/data of functions are divided into the text, data, and stack sections. We map at least one section among these three sections to either SRAM or FRAM regions, i.e., Mem_j and $Sec_k(i)$ defines section 'k' of i^{th} function as shown in Equation 5.12, when k=1, we refer to the text section of i^{th} function when k=2, we refer to the data section of i^{th} function, and when k=3, we refer to the stack section of i^{th} function.

$$Sec_k(i) = \begin{cases} k = 1 & ;\text{Text} \\ k = 2 & ;\text{Data} \quad ;\forall i \in [1, N_f] \\ k = 3 & ;\text{Stack} \end{cases} \tag{5.12}$$

We define a BV; $I_j\left(Sec_k(i)\right)$, which refers to a section Sec_k of i^{th} function is allocated to only one memory region j. If $I_j\left(Sec_k(i)\right)$=1 then the section Sec_i is allocated and $I_j\left(Sec_k(i)\right)$=0 that indicates the section Sec_i is not allocated. $I_j\left(Sec_k(i)\right)$, where ($\forall j \in [1, Mem_j], \forall i \in [1, N_f]$), $\forall k \in [1, Sec_k(i)]$) is defined as shown in Equation 5.13.

$$I_j\left(Sec_k(i)\right) = \begin{cases} 1 & Sec_k \text{ of } i^{th} \text{ function is allocated to } j \\ 0 & \text{otherwise} \end{cases} \tag{5.13}$$

Constraints: There are two constraints, one is for BV; $I_j\left(Sec_k(i)\right)$ and one is a memory size constraint. In any case, a Sec_k of the i^{th} function is allocated to only one memory region, which means that the Sec_k of the i^{th} function is allocated to either SRAM or FRAM but not both. This constraint is defined in Equation 5.14.

$$\sum_{k=1}^{3} \sum_{j=1}^{Mem_j} I_j\left(Sec_k(i)\right)) = 1 \quad (\forall i \in [1, N_f]) \tag{5.14}$$

The other constraint is related to memory sizes. The allocated sections $Sec_k(i)$ and its $Size(F_i)$; $\forall k \in [1, Sec_k(i)]$), $\forall j \in [1, Mem_j]$, $\forall i \in [1, N_f]$ should not be greater than the $Size(Mem_j)$. This constraint is defined in Equation 5.15.

$$\sum_{i=1}^{G} I_j\left(v_i\right) * Size(v_i) + \sum_{k=1}^{3}\sum_{i=1}^{N_f} I_j\left(Sec_k(i)\right) * Size(F_i) \leq Size(Mem_j) \tag{5.15}$$

Objective 4.2: The challenge of mapping sections of these functions in a program to either SRAM or FRAM is to minimize EDP and improve system performance. E_{func} is defined in Equation 5.16, where M_{c_i} is the number of the times i^{th} functions called.

$$E_{func} = \sum_{j=1}^{Mem_j}\sum_{i=1}^{N_f}[E_{r\_j} \times r(F_i) + E_{w\_j} \times w(F_i)] \times M_{c_i} \tag{5.16}$$

EDP_{func} is defined in Equation 5.17. Where EDP_{func} is the energy-delay product required to allocate all functions to either SRAM or FRAM. Where E_{func} is the energy required to allocate functions to either SRAM or FRAM. Where NC_{F_i} is the number of CPU cycles required to execute a function F_i.

$$EDP_{func} = \sum_{k=1}^{3}\sum_{j=1}^{Mem_j}\sum_{i=1}^{N_f} I_j\left(Sec_k(i)\right)\left[E_{func} \times NC_{F_i}\right] \tag{5.17}$$

The overall system EDP, EDP_{system}, is the sum of both EDP_{global} and EDP_{func} as shown in the equation 5.18.

$$EDP_{system} = \eta(EDP_{global} + EDP_{func}) \tag{5.18}$$

Our objective function is shown in Equation 5.19. Our main objective is to minimize the system's EDP by choosing the optimal placement choice.

$$\textbf{Objective Function: Minimize } EDP_{system} \tag{5.19}$$

5.3.2 Implementing Mapping Technique in MSP430FR6989

Once we obtain the placement information from ILP_solver, we map the respective variables and the sections of a function to SRAM or FRAM. We modify the linker script accordingly to map the sections or variables to SRAM or FRAM. In our proposed mapping policy, placing global variables is straightforward, i.e., mapping the respective variable to either SRAM or FRAM based on the ILP decision.

We observed that from the linker script, we could map the whole stack section of each function to either SRAM or FRAM. We analyzed the stack section mappings for each function by modifying the linker script. We used built-in attributes to differentiate the mappings between SRAM and FRAM; for example, we used the built-in attribute ($\_\_\_attribute\_\_\_((ramfunc))$ that maps this function to SRAM. If we want to place the stack section in SRAM, we modify the linker script by replacing the default setting with " .stack: {} > RAM (HIGH) ". If we want to place the stack section in FRAM, we modify the linker script by replacing the default setting with " .stack: {} > FRAM".

Similarly, for the text section, we observed that placing the text section in either SRAM or FRAM shows an impact on EDP. This effect is because the majority of accesses in the text section are read accesses, as we observed that the energy consumption for each read access to SRAM/FRAM differs. Table 3.2 shows that approximately FRAM consumes 2x more read energy than SRAM. Thus, we analyzed each application to map the text section based on the available free space. If we have enough space available in SRAM, we place the text section in SRAM itself; otherwise, we place the text section in FRAM. We included the following four lines in our linker script to check the above condition and map the text section.

1. $\#ifndef\_\_\_LARGE\_CODE\_MODEL\_\_\_$
2. .text : {} > FRAM
3. #else
4. .text : {} » SRAM

We modified the linker script for mapping the data section by using the inbuilt compiler directives. We followed the following three steps.

1. Allocate a new memory block, for example, $NEW\_DATASECTION$. We can declare the start address and size of the data section in the linker script.
2. Define a segment (.Localvars) that is stored in this memory block ($NEW\_DATASECTION$).
3. Use #pragma $DATA\_SECTION(funct\_name, seg\_name)$ in the program to define functions in this segment. Here $funct\_name$ is the function name, and seg_name is the segment name created. For instance, #pragma $DATA\_SECTION(func\_1, .Localvars)$

Once we have created the different sections, we can assign these sections to either SRAM or FRAM based on ILP decisions. For example, placing "$NEW\_DATASECTION$: {} > FRAM" in the linker script, which maps $NEW\_DATASECTION$ to FRAM.

5.3.3 Support for Intermittent Computing

When the power is stable, everything works properly. Because of the static allocation scheme, we map all functions and/or variables to SRAM/FRAM for the first time. During a power failure, SRAM and registers lose all of their contents, including mapping information. When power is restored, we don't know what functions/variables were allocated to SRAM before the failure. As a result, we must either restart the execution from the beginning or end up with incorrect results. Restarting the application consumes extra energy and time, making our system inefficient in terms of energy consumption and performance.

We propose a backup strategy during frequent power failures. FRAM was divided into $FRAM_n$ and $FRAM_b$ as shown in Fig. 5.1. $FRAM_n$ has a size of 125 KB and is used for regular mappings. $FRAM_b$ has a size of 3 KB that serves as a backup region (BR) during power failures. So, during a power failure, we back up all register and SRAM contents to FRAM. Whenever power is restored, we restore the register and SRAM contents from $FRAM_b$ to SRAM and resume the application execution. The proposed backup strategy reduces additional energy consumption and makes the system more energy efficient.

Implementation details of Flash-based Programming for Intermittent Computing: MSP430F5529 consists of SRAM and Flash in the main memory. SRAM is the only memory on the chip where the CPU can read code to execute the application during Flash programming. We need to copy the Flash program function onto the stack whenever we want to use only SRAM for mapping the application. Whenever we want to switch between SRAM to Flash, we need to restore the stack pointer, and as well as we need to map the program counter register to the Flash memory region.

During a power failure scenario, we must perform the backup operation to copy the SRAM data to the Flash memory region. For the backup operation, we made some changes to the inbuilt MSP430 functions, such as void Flash_wb(char *Data_ptr, char byte) and void Flash_ww(int *Data_ptr, int word). Here Flash_wb() helps to write the byte to the Flash memory region, and Flash_ww() helps to write the word to the Flash memory region.

Whenever power comes back, we must restore the contents from the Flash-based backup region to the SRAM memory region. We used the inbuilt functions, i.e., ctpl() functions for copying from Flash to SRAM, and after restoring, we needed to clear the Flash-based backup region; for this, we made changes to the inbuilt function, i.e., void Flash_clr(int *Data_ptr) to clear the Flash data.

5.4 Experimental Setup and Results

5.4.1 Experimental Setup

We used TI's MSP430FR6989 for all experiments. We experimented with mixed benchmarks, which have both Mi-Bench [74] and TI-based benchmarks. We have shown the experimental setup in Table 5.1. The development platform and experimental setup are shown in Fig. 5.3. We performed experiments to determine the energy required for a single read/write to SRAM/FRAM, as shown in Table 5.2. We collected the number of reads/writes for each global variable and function as part of a one-time characterization. We also used TI's MSP430F5529 for comparing flash with FRAM. We performed experiments to determine the energy required for a single read/write to flash, as shown in Table 5.2.

MCU, which we have tested, has MSP430 architecture, which is more suitable for IoT devices. Most MSP430 software is written in C and compiled with one of TI's recommended compilers (IAR Embedded Code Bench, Code-Composer Studio (CCS), or

Table 5.1: Experimental Setup

Component	**Description**
Target Board	TI MSP430FR6989 Launchpad
Core	MSP430 (1.8-3.6 V; 16 MHz)
Memory	2KB SRAM and 128KB FRAM
IDE	Code Composer Studio
Energy Profiling	Energy Trace++
ILP Solver	LPSolve_IDE
Benchmarks	Mixed benchmarks (MiBench and TI-based)

msp430-gcc). The IAR Embedded Code Bench and CCS compilers are part of integrated development environments (IDEs). We use the widely used, freely available, and easily extended tool, i.e., CCS, for all experiments in this work. EnergyTrace++ technology allows us to calculate energy and power consumption directly. According to the datasheet for the MSP430FR6989, the number of cycles required to read/write in FRAM is twice that of SRAM, which means the access penalty of FRAM is twice that of SRAM at this specific operating point of 16 MHz. The latency penalty disappears when operating at/below 8 MHz and gets worse above 16 MHz.

Table 5.2: Energy Values for each read/write to SRAM and FRAM

Memory	**Per Read Energy (nJ)**	**Per Write Energy (nJ)**
SRAM	5500	5600
FRAM	10325	13125
Flash	23876	31198

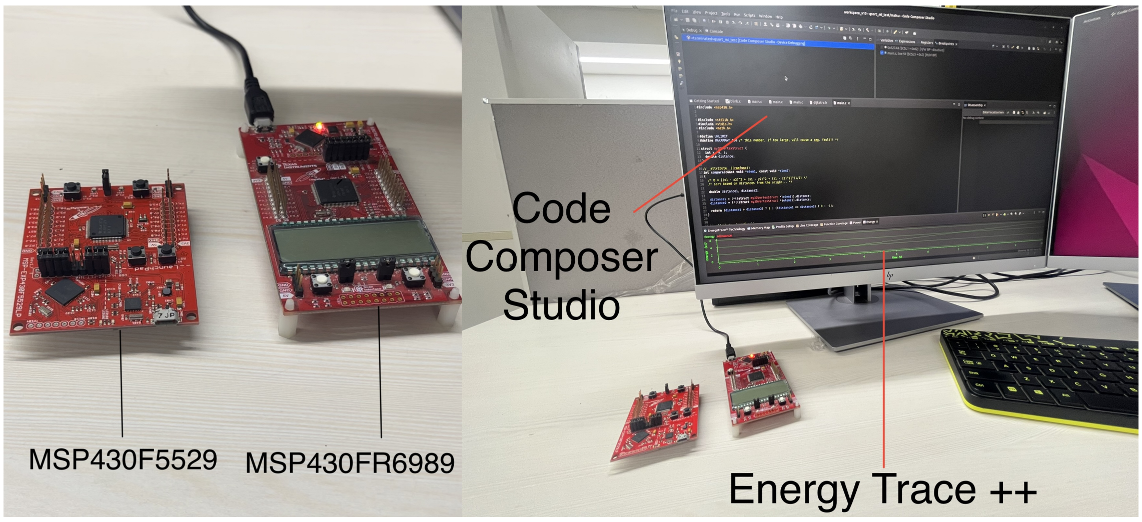


Figure 5.3: (a) TI-based MSP430 Launchpad Development Boards (b) Working with EnergyTrace++ on CCS

5.4.2 Evaluation Benchmarks

We chose benchmarks from both the MiBench suite and TI benchmarks. One of the primary motivations for using the MiBench suite is that most of the TI-based benchmarks were small in size and easily fit into either SRAM or FRAM. In these cases, we don't require any hybrid memory design. Most TI-based benchmarks have only one or two functions and 3-4 global variables, which is not useful for the hybrid main memory design. Thus, we used mixed benchmarks consisting of 4 TI-based benchmarks and 12 from the MiBench suite.

For the MiBench suite, we first make MCU-compatible benchmarks by adding MCU-related header files and watchdog timers. All benchmarks may not be compatible with the MCU. Thus, we need to choose the benchmarks from the MiBench suite that are compatible with the MSP430 boards. Once we have benchmarks, we execute them on board for the machine code. Using the .asm file, we calculate the inputs that are required by the ILP solver, as shown in Fig. 5.2.

5.4.3 Baseline Configurations

We chose five different memory configurations to compare with the proposed memory configuration.

1. We directly map all the functions/variables to FRAM in the FRAM-only configuration. We use the FRAM-only configuration to compare our proposed memory configuration during stable and unstable power scenarios.

2. We directly map all functions/variables to SRAM in the SRAM-only configuration. We use the SRAM-only configuration to compare our proposed memory configuration during stable and unstable power scenarios.

3. We used the empirical method of Jayakumar et al. [1]. We compare this configuration with our proposed configuration during stable and unstable power scenarios to observe the importance of the proposed work rather than the existing work.

4. In the SRAM+Flash with ILP configuration, we used the proposed ILP technique for the flash-based msp430 board [181]. We compare this configuration with our proposed configuration during stable and unstable power scenarios to observe the difference between the FRAM and Flash technologies.

5. In SRAM+FRAM with ILP configuration, we have the proposed memory mapping technique that does not support BR. We compare this configuration with our proposed configuration during frequent power failures to observe the importance of BR. The overview of all baseline configurations is shown in Table 5.3. The experimental setup for all the above five configurations is the same as the one proposed.

Table 5.3: Overview of the Different Memory Configurations for Comparing with the Proposed Memory Configuration

Configuration	**FRAM**	**SRAM**	**Flash**	**BR**	**ILP**
FRAM-only	✓	✗	✗	✗	✗
SRAM-only	✗	✓	✗	✗	✗
Jayakumar et al. [1]	✓	✓	✗	✗	✗
SRAM+Flash with ILP	✗	✓	✓	✗	✓
SRAM+FRAM with ILP	✓	✓	✗	✗	✓
Proposed	✓	✓	✓	✓	✓
✓- Supported , ✗- Not Supported					

5.4.4 Results & Analysis

The proposed memory configuration is evaluated in this section under stable and unstable power. The proposed memory configuration is compared with five different memory configurations as discussed in section 5.4.3.

5.4.4.1 Analysis for System's Energy Delay Product

Under Stable Power Scenarios: Our main objective of the proposed memory configuration is to minimize the system's EDP. All values shown in Fig. 5.4 are normalized with the FRAM-only configuration. Compared to the FRAM-only configuration, the proposed gets 38.10% lesser EDP, as shown in Fig. 5.4. Because there are no power interruptions in this scenario, this improvement is totally due to the proposed ILP model. In configuration-1, we place everything on FRAM, where FRAM consumes more energy and the number of cycles than SRAM, as shown in the table 5.2. Our proposed memory configuration incorporates the placement recommendation from the proposed ILP model and suggests utilizing both SRAM and FRAM.

Under a stable power scenario, the proposed gets 9.30% less EDP than Jayakumar et al., as shown in Fig. 5.4. The empirical method used by Jayakumar et al. is as follows. Jayakumar et al. considered functions as a basic unit. They explored all configurations and calculated the energy values. Jayakumar et al. method has eight configurations because they have two memory regions (SRAM or FRAM) and need to map three sections (stack, data, text). Jayakumar et al. assumed that the data section included all global variables, constants, and arrays. As a result, our proposed ILP-based mapping differs from the author's mapping in that our proposed mapping outperforms the existing work. Under stable power, Jayakumar et al. receive 24.57% less EDP than the FRAM-only configuration, as shown in Fig. 5.4. This advantage is primarily due to hybrid memory from Jayakumar et al.

Compared to SRAM + Flash with ILP configuration, the proposed reduces EDP by 18.55%, as shown in Fig. 5.4. In this configuration, we used flash + SRAM with our proposed ILP framework. As shown in Table 5.2, the above benefit is mainly due to FRAM because flash consumes more energy. Jayakumar et al. outperform SRAM +

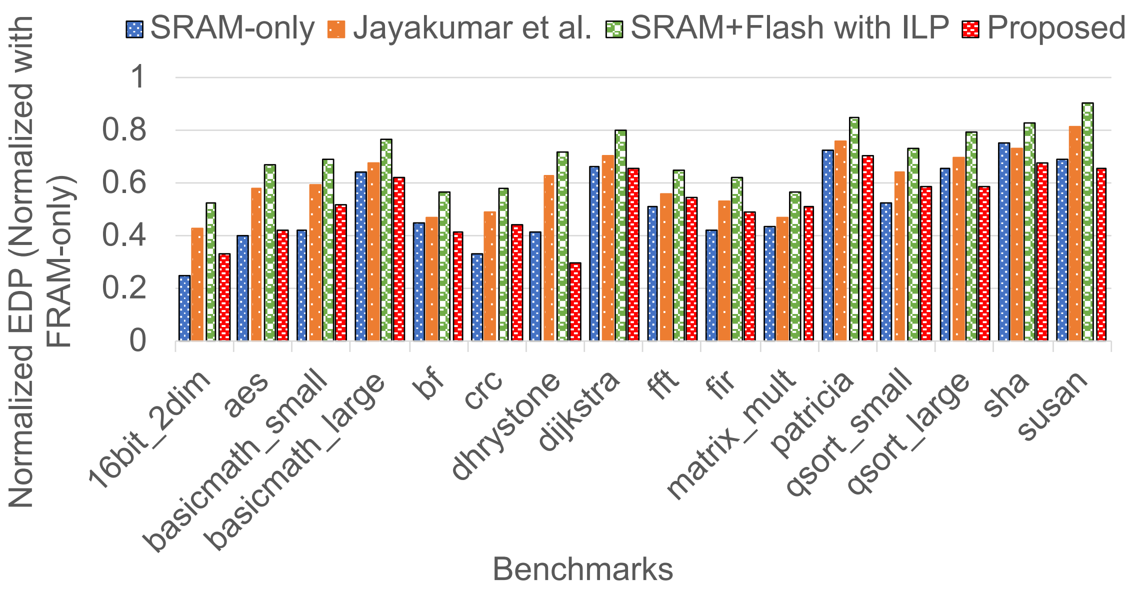


Figure 5.4: Comparison between Baseline configurations and the Proposed under Stable Power.

Flash with the ILP configuration during stable power. Due to FRAM in Jayakumar et al., even our proposed ILP model is ineffective in this case. We found that Jayakumar et al. achieved 9.19% less EDP than SRAM+Flash with ILP configuration, and this benefit is due to smaller applications. From Fig. 5.4, SRAM+Flash with ILP configuration performs better for large applications than SRAM+Flash with ILP configuration. Jayakumar et al. empirical method suggests placing more content on SRAM because SRAM is sufficient for placing the entire small-size application. As a result, the performance of Jayakumar et al. depends on the application size, as for large-size applications, even FRAM does not outperform Flash.

Comparing the proposed memory configuration with the empirical method of Jayakumar et al. helps us understand the role of the ILP model. This comparison also clarifies whether these advantages stem from mapping granularity or ILP. The proposed memory configuration outperforms the existing one, demonstrating that it benefits from mapping granularity and the ILP model.

SRAM-only configuration outperforms the proposed and all other memory configurations under stable power conditions. We noticed that this benefit is primarily due to SRAM, but it only applies to smaller applications. SRAM-only achieves 36.19% less EDP than the proposed for smaller applications, as shown in Fig. 5.4. We also looked at large applications where the proposed configuration outperforms the SRAM-only configuration by a small margin. When the SRAM is full, the MCU must wait for the space to be released, which consumes extra energy and cycles. For more extensive applications, the SRAM-only configuration achieves 2.94% more EDP than proposed.

Under Unstable Power Scenarios: We used the default TI-based compute through power loss (ctpl) tool for migration. During a power failure, we need to migrate the

SRAM contents to a FRAM-based backup region ($FRAM_b$), i.e., the backup process. Whenever the power comes back, we need to migrate the ($FRAM_b$) contents to SRAM, i.e., the restoration process. Therefore, all these migrations are done using ctpl() functions. We introduce a power failure by changing the low-power modes mentioned in the MSP430FR6989 design document. We use ctpl() to create power failures. We assume that the number of power failures is spread equally within the execution period. For instance, if the total execution period for an application is 20 milliseconds (ms), and let us say the number of power failures is four, then for every 5 ms, we experience a power failure.

We performed experiments under unstable power to compare the proposed memory configuration with five memory configurations, as shown in Table 5.3. All values shown in Fig. 5.5 are normalized with the SRAM-only configuration. Compared to the SRAM-only configuration, the proposed one gets a 15.97% lower EDP, as shown in Fig. 5.5. We observe that the migration overhead is less than the energy consumed to execute the FRAM application, and this migration overhead depends on the number of power failures. For example, a backup migration consumes approximately 16.88 mJ of energy, and a restore migration consumes approximately 11.606 mJ of energy in a qsort application. The above benefit to our proposed configuration is using a hybrid memory.

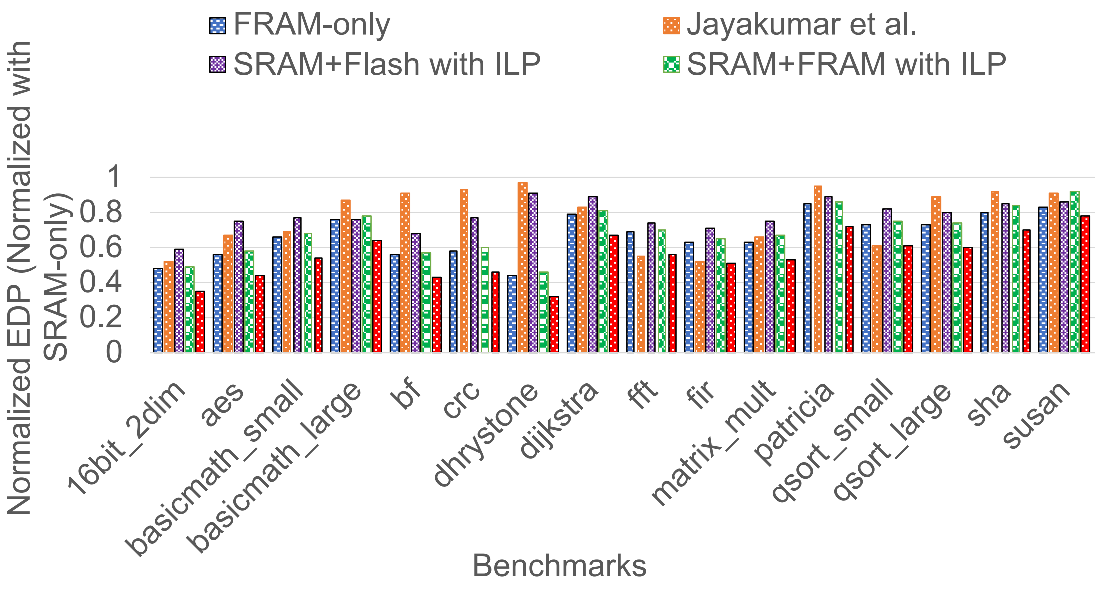


Figure 5.5: Comparison between Baseline configurations and the Proposed under Unstable Power.

Under an unstable power scenario, the proposed gets 21.99% less EDP than Jayakumar et al., as shown in Fig. 5.5. As already stated, the empirical method by Jayakumar et al. is more beneficial for small applications. In contrast, the author's empirical method suggests placing more content on SRAM because SRAM is sufficient for placing the entire small-size application. Thus, for Jayakumar et al. work, backup/restore operations take more energy during a power failure. Our proposed mapping outperforms existing work. During frequent power failures, Jayakumar et al. receive 6.91% less EDP than the FRAM-only configuration, as shown in Fig. 5.5. This advantage is primarily due to hybrid memory by Jayakumar et al.

Compared to SRAM+Flash with ILP configuration, the proposed reduces EDP by 23.05%, as shown in Fig. 5.5. As shown in Table 5.2, the above benefit is mainly due to FRAM because flash consumes more energy. Jayakumar et al. outperform SRAM + Flash with the ILP configuration during stable power. Because of the FRAM in Jayakumar et al., even our proposed ILP model is ineffective for this comparison. We found that Jayakumar et al. achieved 6.28% less EDP than SRAM+Flash with ILP configuration for smaller applications. The above benefit for Jayakumar et al. is minimal because the size of backup/restore increases, which even neutralizes the flash for some applications, as shown in Fig. 5.5. SRAM+Flash with ILP configuration achieves 2.69% less EDP than Jayakumar et al. for large applications, as shown in Fig. 5.5. As a result, the performance of Jayakumar et al. depends on the application size, as for large-size applications, even FRAM does not outperform Flash.

The proposed memory configuration outperforms all memory configurations under unstable power conditions. This benefit is primarily due to hybrid memory and the proposed mapping technique. SRAM-only achieves 42.98% less EDP than the proposed, as shown in Fig. 5.5.

When we remove BR, all the mapping information from the SRAM is lost because our model is static. We introduce a BR in the FRAM memory region to save this mapping information. During a power failure, we migrate the SRAM contents to $FRAM_b$; whenever power comes back, we restore the $FRAM_b$ contents to the SRAM.

We experimented to know the importance of BR, where we compared the proposed memory configuration with SRAM+FRAM with ILP configuration. Compared to SRAM+FRAM with ILP configuration, the proposed gets 23.94% lower EDP, as shown in Fig. 5.5. This benefit is because we need to re-execute the application four times from the beginning, which consumes extra time and energy. The number of times re-executing the application is equal to the number of power failures.

5.4.4.2 Comparing Flash-based MCU (MSP430F5529) with FRAM-based MCU (MSP430FR6989)

Under Stable Power Scenarios: We also evaluated our proposed framework with another MSP430F5529 MCU with flash and SRAM for completeness. This comparison helps the user in selecting the most appropriate NVM technology, such as FRAM or Flash, as needed. To be fair, we used the same sizes of SRAM (2 KB) and Flash (128 KB) in this comparison. We compared FRAM-based and flash-based MCUs under stable power conditions. We used the proposed frameworks and techniques in both MCUs. We discovered that the proposed FRAM-based configuration outperforms the flash-based configuration. Flash-based configurations consume 26.03% more EDP than FRAM-based configurations, as shown in Fig. 5.6. Flash consumes more energy, as shown in Table 5.2.

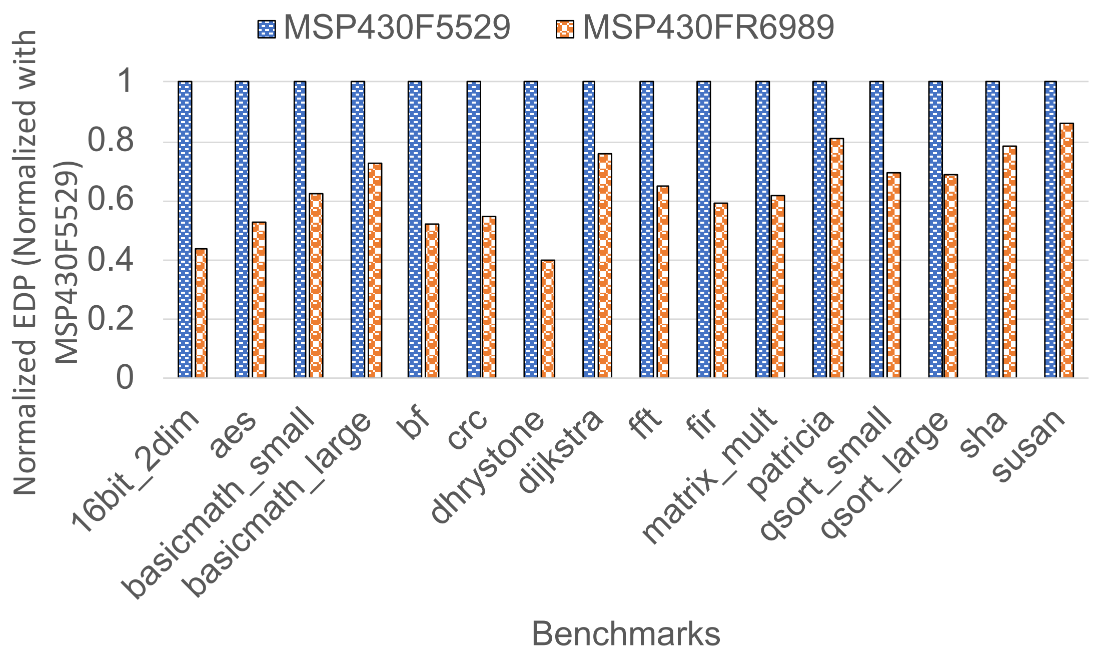


Figure 5.6: Comparison between MSP430FR6989 (FRAM-based MCU) and MSP430F5529 (Flash-based MCU) under Stable Power.

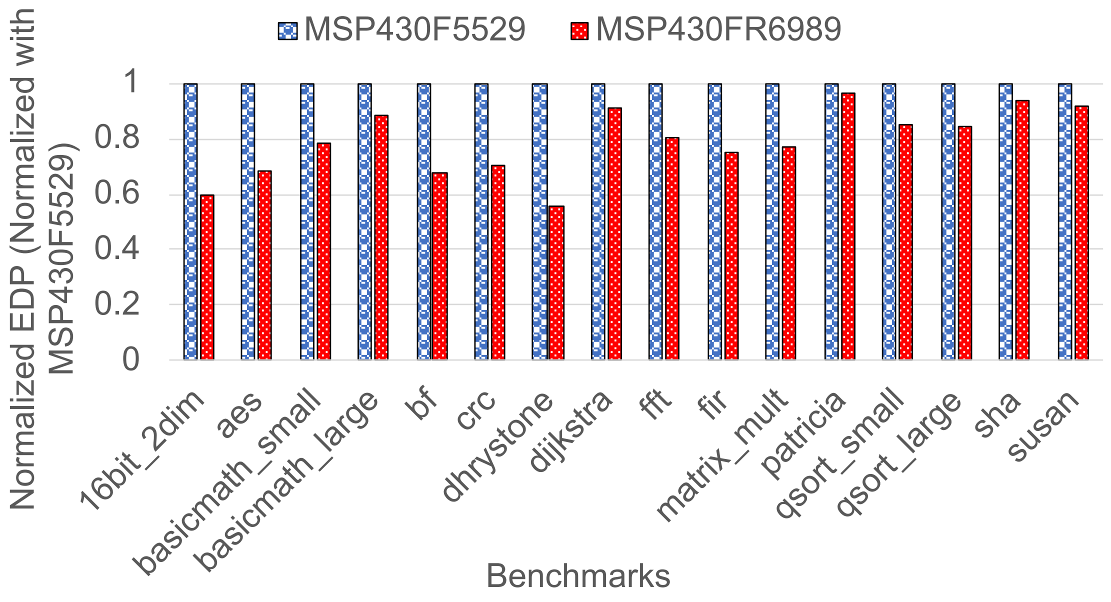


Figure 5.7: Comparison between MSP430FR6989 (FRAM-based MCU) and MSP430F5529 (Flash-based MCU) under Unstable Power.

Under Unstable Power Scenarios: We also evaluated our proposed framework with another MSP430F5529 MCU, which consists of flash and SRAM for completeness. This comparison helps the user in selecting the most appropriate NVM technology, such as FRAM or Flash, as needed. To be fair, we used the same sizes of SRAM (2 KB) and Flash (128 KB) in this comparison. We also used BR in these experiments; the only difference is that we replaced the FRAM with flash in the proposed configurations, and everything is the same. We compared FRAM-based and flash-based MCUs under unstable power conditions. We used the proposed frameworks and techniques in both MCUs. We found that the proposed FRAM-based configuration outperforms the flash-based configuration. Flash-based configurations consume 16.50% more EDP than FRAM-based configurations, as shown in Fig. 5.7. Flash consumes more energy, as shown in Table 5.2.

5.4.5 Summary of the Proposed Mapping Technique

In this section, we outline the proposed ILP-based memory mapping technique. Following all of these analyses, we observed that the mappings shown below consume less EDP than other design choices, as shown in the table. To keep things simple, we only show the final mapping configurations for each application's stack, data, and text sections, keeping out the final mappings for global variables.

Table 5.4: Optimal Placement for different Applications in MSP430FR6989

Benchmarks	**Stack**	**Text**	**Data**
16bit_2dim	SRAM	SRAM	SRAM
aes	SRAM	FRAM	FRAM
basicmath_small	SRAM	SRAM	FRAM
basicmath_large	SRAM	FRAM	FRAM
bf	SRAM	SRAM	FRAM
crc	SRAM	FRAM	SRAM
dhrystone	FRAM	SRAM	FRAM
dijkstra	SRAM	FRAM	SRAM
fft	SRAM	SRAM	FRAM
fir	SRAM	SRAM	FRAM
matrix_mult	SRAM	SRAM	SRAM
patricia	SRAM	FRAM	SRAM
qsort_small	SRAM	SRAM	FRAM
qsort_large	SRAM	FRAM	FRAM
sha	SRAM	FRAM	FRAM
susan	SRAM	FRAM	FRAM

Table 5.4 shows that, with the exception of the dhrystone application, the remaining three TI benchmark applications (fir, matrix, and 16bit_2dim) are very small and can easily be placed in SRAM. We don't need FRAM for these types of smaller applications, but there is a disadvantage during frequent power failures. The backup and restore sizes to FRAM are larger for these applications during frequent power failures. As a result, our proposed backup/recovery strategy should be intelligent enough to reduce EDP. The

dhrystone application, on the other hand, has a larger stack section that requires FRAM to accommodate the entire stack section.

As we can see in Table 5.4, many applications used both SRAM and FRAM for Mi-Bench applications. As a result, we can conclude that a hybrid main memory design is required for many applications. Using a hybrid main memory design helps reduce EDP during stable power scenarios. However, determining how and where to backup volatile contents can be difficult during frequent power outages. However, our proposed memory mapping technique and the framework suggest using a hybrid main memory design that supports intermittent computing.

The limitations that we observed in this chapter are as follows: (1) This chapter limits the NVM at the main memory level; (2) This work does not support the architecture that consists of caches; (3) The proposed memory mapping technique is a static memory mapping technique, which opens the door for exploring dynamic memory mapping techniques for hybrid main memory architectures; and (4) We all know that ILP is an optimization problem and NP-complete problem, there could be a good possibility to explore the better-optimized solutions for mapping the contents to hybrid main memory.

5.5 Summary

In this chapter, an ILP-based memory mapping technique is proposed to reduce the system's energy-delay product. For both global variables and functions, we formulate an ILP model. The functions consist of data, stack, and code sections. Our ILP model suggests placing each section on SRAM or FRAM. Under both stable and unstable power scenarios, we compared the proposed memory configuration to the baseline memory configurations. We evaluated our proposed frameworks and techniques on actual boards. We added a backup region to FRAM to support intermittent computing. We compared the proposed framework with recent related work.

Under stable power, our proposed memory configuration consumes 38.10% less EDP than the FRAM-only configuration and 9.30% less EDP than the existing work. Under unstable power, our proposed configuration achieves 15.97% less EDP than the FRAM-only configuration and 21.99% less EDP than the existing work. Under stable power, our proposed memory configuration consumes 18.55% less EDP than SRAM + Flash with the ILP configuration. We also compared the FRAM-based MSP430FR6989 with the flash-based MSP430F5529. Compared to flash, the FRAM-based hybrid main memory design consumes less EDP. FRAM-based design consumes 26.03% less EDP than flash-based design during stable power and 16.50% less EDP than flash-based design during frequent power failures.

Chapter 6

Conclusion and Future Possibilities

This chapter provides an overall conclusion of the work presented in various chapters of this thesis. Section 6.1 summarizes the main findings and provides an overall conclusion, followed by the scope of future work in Section 6.2.

6.1 Major Research Contributions

The following are major research contributions discussed in this study:

– Designing a non-volatile processor (NVP) involves challenges at the register files, caches, and main memory level. In this thesis, we analyzed the challenges at the first-level cache, last-level cache (LLC), and main memory levels and proposed unique/novel architectures to address some of these challenges.

– We investigated the suitability of Non-Volatile Memory (NVM) based hybrid cache architecture for the first-level cache. We proposed efficient migration and prediction techniques that would ultimately be helpful in determining which cache block should be placed in which cache region. The proposed architecture and policies reduce the number of writes to NVM, which reduces energy consumption compared to the baseline and existing architectures.

– In intermittent computing devices, the energy to perform data backup is fixed and limited, constraining the backup content size. We proposed efficient cache management policies to track and maintain the fixed number of dirty blocks at L1 using a dirty block table (DBT) and write-back queue (WBQ). The proposed architecture uses fixed energy for backup operations during power failures. The proposed architecture and policies reduce the number of writes to NVM, which reduces energy consumption compared to the baseline architectures.

– We designed an efficient memory mapping technique for recent microcontrollers (MSP430FR6989 and MSP430F5529) that consists of NVM-based hybrid main memory (SRAM+FRAM and SRAM+Flash). Integer linear programming (ILP) based memory mapping techniques have been introduced to get benefits from both SRAM and NVM. The proposed ILP model gives us the optimal placement decision to achieve less energy-delay product (EDP) than the baseline and existing memory mapping models during power failures.

– The code and related data are made available to the research community using GitHub public repositories.

6.2 Future Possibilities

However, more research in this field is needed until universal adoption is achieved. The identified future research directions are described below.

6.2.1 NVMs at Register Level

We completed deploying NVMs at the cache and main memory levels. We plan to work on "Inserting NVMs at the Flip-Flop Level." This topic presents several challenges that must be thoroughly investigated.

6.2.2 Energy Harvesting Devices

We currently have many assumptions for the simulation environment. We intend to deploy real energy harvesting devices in order to create true battery-free devices. When we deploy real energy-harvesting devices, we may face new challenges and goals. As a result, it generates a potential research direction to investigate further.

6.2.3 Dynamic Memory Partioning

To map the application to SRAM or FRAM, we explored a static memory partitioning technique. In a static memory mapping technique, we need all the information ahead of time, and we profiled each application to know the information of all inputs. However, application scenarios can change, and using the static memory partitioning technique may result in inaccurate memory mapping decisions. As a result, we can investigate the feasibility of using dynamic memory partitioning techniques for MSP430-based microcontrollers in frequent power failure scenarios.

6.2.4 Secure NVM-based Architectures

When we were exploring NVM-based architectures, we encountered many security concerns in these types of architectures. Since the main memory of the computing system is now integrated with NVM, attackers can easily extract the data from powered-off devices with physical access to the device. At the same time, in an NVM-based system, the memory state is maintained, and during critical security calculations, key memory states, such as cryptographic functions, can reveal secret information. One such direction we observed was to encrypt the data exposed to the attacker or encrypt the data stored in NVM. In addition to encryption, we can even lock the NVM data during a power failure or before a power failure. These two operations should be in an atomic execution. As a result, it generates a potential research direction to investigate it further.

6.2.5 Predicting Power Failures

When we deal with unstable power scenarios, we notice that power failures are random. With this observation, we posed two questions: "Is the energy stored by the capacitor at the time of power failure less than the energy required to back up the SRAM contents, even for a fixed number of SRAM contents?" "Does predicting power failures save dynamic energy consumption and execution time, or does it worsen the situation"? These questions inspired us to conduct further studies to investigate them and develop efficient techniques and energy models. As a result, these questions provide an easily identifiable research direction.

6.2.6 NVM-based GPU Architectures

Transactional memory (TM) is a programming model developed by the database community to simplify the use of fine-grained parallelism. NVM-based main memory has changed the existing hierarchy of memory and storage. So, the technique that initially considered volatile and non-volatile memory separately now has to consider the persistence throughout the stack. Persistence means that any operation, including an error, is stored on the device throughout its lifetime. Therefore, to make the system's state always valid, consistency must be ensured for it to be valid. Several applications and algorithms have already addressed the issues related to consistency, such as databases that depend on lock-free data structures, such as hash tables, skip lists, and B+ trees.

Graphics processing units (GPUs) are widely used as standard accelerators for high-performance computing. GPUs are designed to use many light parallel threads for speed-oriented computation. For this purpose, they are constructed with a computing unit that can accommodate many resident threads along with their register state, as well as a deeply pipelined memory subsystem capable of handling a large number of parallel memory accesses. The use and maintenance of NVM-based main memory on CPUs are being studied intensively, but how to maintain the GPUs along with NVMs is still in the early stages.

If we observe, the combination of TM + NVM + GPU for intermittent computing will become an exciting research direction to explore and propose efficient NVM-based GPU architectures.

6.2.7 Potential Applications

Some of the potential future applications of the proposed architectures and frameworks are listed below.

– **Defence Applications:** Intermittent computing systems frequently lack the energy to process all input data in a single cycle. When a process is interrupted, the current outcome is most likely incomplete. In such cases, the approximate results can be a complete and accurate representation of the output. In such military applications, some sensors/cameras are deployed at the border to check the movement between the regions. When one of these sensors/cameras is damaged, or the power supply to these devices is disrupted, the proposed frameworks and architectures will assist in retrieving or processing the data using external capacitors and NVMs.

– **Health Monitoring Applications:** If the processor cannot keep up the data from devices that harvest energy from continuous input signals, but it can sample inputs. A sample with precise computations is preferable to one with approximate results from larger inputs. Dropping samples precisely to maintain data flow runs the risk of missing critical data contained in dropping samples. The proposed frameworks and architectures avoid this by saving the previous data and results that help to generate approximate results for all samples to avoid the loss of important information.

  Consider monitoring blood glucose levels for diabetic patients. Regular monitoring is required to detect dangerously low concentrations and rapidly implement corrective measures. Energy harvesting and wearable monitoring devices have been developed to meet this demand. However, these devices must balance their rigorous energy requirements with as much reading as possible. However, if you have sufficient energy to continue processing the same data, our proposed frameworks and architectures offer the benefit of increasing accuracy over time by storing the results in NVMs. The proposed frameworks and architectures rapidly trade off the sampling frequency's precision based on the requirements and constraints of each application and system.

References

[1] Hrishikesh Jayakumar, Arnab Raha, Jacob R Stevens, and Vijay Raghunathan. Energy-aware memory mapping for hybrid fram-sram mcus in intermittently-powered iot devices. *ACM Transactions on Embedded Computing Systems (TECS)*, 16(3):1–23, 2017.

[2] Hêriş Golpîra, Syed Abdul Rehman Khan, and Sina Safaeipour. A review of logistics internet-of-things: Current trends and scope for future research. *Journal of Industrial Information Integration*, page 100194, 2021.

[3] Xiaosong Hu, Le Xu, Xianke Lin, and Michael Pecht. Battery lifetime prognostics. *Joule*, 4(2):310–346, 2020.

[4] Dong Ma, Guohao Lan, Mahbub Hassan, Wen Hu, and Sajal K Das. Sensing, computing, and communications for energy harvesting iots: A survey. *IEEE Communications Surveys & Tutorials*, 22(2):1222–1250, 2019.

[5] Attapong Mamen and Uthane Supatti. A survey of hybrid energy storage systems applied for intermittent renewable energy systems. In *2017 14th ECTI-CON*, pages 729–732. IEEE, 2017.

[6] Yongpan Liu, Hehe Li, Xueqing Li, Jason Chun Xue, Yuan Xie, and Huazhong Yang. Self-powered wearable sensor node: Challenges and opportunities. In *International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, pages 189–189. IEEE, 2015.

[7] Brandon Lucia, Vignesh Balaji, Alexei Colin, Kiwan Maeng, and Emily Ruppel. Intermittent computing: Challenges and opportunities. *2nd Summit on Advances in Programming Languages (SNAPL 2017)*, 2017.

[8] Sukarn Agarwal and Shounak Chakraborty. Abaca: Access based allocation on set wise multi-retention in stt-ram last level cache. In *2021 IEEE 32nd International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 171–174. IEEE, 2021.

[9] Sparsh Mittal and Jeffrey S Vetter. A survey of software techniques for using non-volatile memories for storage and main memory systems. *IEEE Transactions on Parallel and Distributed Systems*, 27(5):1537–1550, 2015.

[10] Albert Lee, Chieh-Pu Lo, et al. A reram-based nonvolatile flip-flop with self-write-termination scheme for frequent-off fast-wake-up nonvolatile processors. *IEEE Journal of Solid-State Circuits*, 52(8):2194–2207, 2017.

[11] Texas Instruments. Msp430fr5969 launchpad development kit, 2018.

[12] Hengyu Zhao and Jishen Zhao. Leveraging mlc stt-ram for energy-efficient cnn training. In *Proceedings of the International Symposium on Memory Systems*, pages 279–290. ACM New York, NY, USA, 2018.

[13] Lan Gao, Rui Wang, Yunlong Xu, Hailong Yang, Zhongzhi Luan, Depei Qian, Han Zhang, and Jihong Cai. Sram-and stt-ram-based hybrid, shared last-level cache for on-chip cpu–gpu heterogeneous architectures. *The Journal of Supercomputing*, 74 (7):3388–3414, 2018.

[14] Satyajaswanth Badri, Mukesh Saini, and Neeraj Goel. Design of energy harvesting based hardware for iot applications. *arXiv preprint arXiv:2306.12019*, 2023.

[15] Tim Daulby, Anand Savanth, Alex S Weddell, and Geoff V Merrett. Comparing nvm technologies through the lens of intermittent computation. In *Proceedings of the 8th International Workshop on Energy Harvesting and Energy-Neutral Sensing Systems*, pages 77–78, 2020.

[16] Joel Van Der Woude and Matthew Hicks. Intermittent computation without hardware support or programmer intervention. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 17–32, 2016.

[17] Sparsh Mittal and Jeffrey S Vetter. A survey of software techniques for using non-volatile memories for storage and main memory systems. *IEEE Transactions on Parallel and Distributed Systems*, 27(5):1537–1550, 2015.

[18] Jalil Boukhobza, Stéphane Rubini, Renhai Chen, and Zili Shao. Emerging nvm: A survey on architectural integration and research challenges. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 23(2):1–32, 2017.

[19] Saad Ahmed, Naveed Anwar Bhatti, Martina Brachmann, and Muhammad Hamad Alizai. A survey on program-state retention for transiently-powered systems. *Journal of Systems Architecture*, 115:102013, 2021.

[20] Sumanth Umesh and Sparsh Mittal. A survey of techniques for intermittent computing. *Journal of Systems Architecture*, 112:101859, 2021.

[21] Jalil Boukhobza, Stéphane Rubini, Renhai Chen, and Zili Shao. Emerging nvm: A survey on architectural integration and research challenges. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 23(2):1–32, 2017.

[22] Sheel Sindhu Manohar and Hemangee K Kapoor. Capmig: Coherence aware block placement and migration in multi-retention stt-ram caches. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2022.

[23] Arindam Sarkar, Newton Singh, Varun Venkitaraman, and Virendra Singh. Dam: Deadblock aware migration techniques for stt-ram-based hybrid caches. *IEEE Computer Architecture Letters*, 20(1):62–4, 2021.

[24] Jian-Gang Zhu. Magnetoresistive random access memory: The path to competitiveness and scalability. *Proceedings of the IEEE*, 96(11):1786–1798, 2008.

[25] Kevin M Lepak and Mikko H Lipasti. Silent stores for free. In *Proceedings 33rd Annual IEEE/ACM International Symposium on Microarchitecture. MICRO-33 2000*, pages 22–31. IEEE, 2000.

[26] Riichiro Takemura, Takayuki Kawahara, Katsuya Miura, Hiroyuki Yamamoto, Jun Hayakawa, Nozomu Matsuzaki, Kazuo Ono, Michihiko Yamanouchi, Kenchi Ito, Hiromasa Takahashi, et al. A 32-mb spram with 2t1r memory cell, localized bi-directional write driver and1'/0'dual-array equalized reference scheme. *IEEE Journal of Solid-State Circuits*, 45(4):869–879, 2010.

[27] Tetsuo Endoh, Hiroki Koike, Shoji Ikeda, Takahiro Hanyu, and Hideo Ohno. An overview of nonvolatile emerging memories—spintronics for working memories. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 6(2):109–119, 2016.

[28] Chengen Yang, Manqing Mao, Yu Cao, and Chaitali Chakrabarti. Cost-effective design solutions for enhancing pram reliability and performance. *IEEE Transactions on Multi-Scale Computing Systems*, 3(1):1–11, 2017.

[29] Chengen Yang, Yunus Emre, Zihan Xu, Hsingmin Chen, Yu Cao, and Chaitali Chakrabarti. A low cost multi-tiered approach to improving the reliability of multi-level cell pram. *Journal of Signal Processing Systems*, 76(2):133–147, 2014.

[30] Manqing Mao, Chengen Yang, Zihan Xu, Yu Cao, and Chaitali Chakrabarti. Low cost ecc schemes for improving the reliability of dram+ prammain memory systems. In *2014 IEEE Workshop on Signal Processing Systems (SiPS)*, pages 1–6. IEEE, 2014.

[31] Xin Chen, Yonghui Zheng, Min Zhu, Kun Ren, Yong Wang, Tao Li, Guangyu Liu, Tianqi Guo, Lei Wu, Xianqiang Liu, et al. Scandium doping brings speed improvement in sb 2 te alloy for phase change random access memory application. *Scientific reports*, 8(1):6839, 2018.

[32] Mimi Xie, Chen Pan, Youtao Zhang, Jingtong Hu, Yongpan Liu, and Chun Jason Xue. A novel stt-ram-based hybrid cache for intermittently powered processors in iot devices. *IEEE Micro*, 39(1):24–32, 2018.

[33] Satyajaswanth Badri, Mukesh Saini, and Neeraj Goel. An efficient nvm-based architecture for intermittent computing under energy constraints. *IEEE*

Transactions on Very Large Scale Integration (VLSI) Systems, pages 1–13, 2023. doi: 10.1109/TVLSI.2023.3266555.

[34] SatyaJaswanth Badri, Mukesh Saini, and Neeraj Goel. Efficient placement and migration policies for an stt-ram based hybrid l1 cache for intermittently powered systems. *Design Automation for Embedded Systems*, 2023. doi: 10.1007/s10617-023-09272-w.

[35] SatyaJaswanth Badri, Mukesh Saini, and Neeraj Goel. Mapi-pro: An energy efficient memory mapping technique for intermittent computing. *arXiv preprint arXiv:2301.11967*, 2023.

[36] Zhenyu Sun, Xiuyuan Bi, Hai Li, Weng-Fai Wong, Zhong-Liang Ong, Xiaochun Zhu, and Wenqing Wu. Multi retention level stt-ram cache designs with a dynamic refresh scheme. In *proceedings of the 44th annual IEEE/ACM international symposium on microarchitecture*, pages 329–338. ACM New York, NY, USA, 2011.

[37] Namhyung Kim, Junwhan Ahn, Kiyoung Choi, Daniel Sanchez, Donghoon Yoo, and Soojung Ryu. Benzene: An energy-efficient distributed hybrid cache architecture for manycore systems. *ACM Transactions on Architecture and Code Optimization*, 15 (1):1–23, 2018.

[38] Jishen Zhao, Cong Xu, Tao Zhang, and Yuan Xie. Bach: A bandwidth-aware hybrid cache hierarchy design with nonvolatile memories. *Journal of Computer Science and Technology*, 31(1):20–35, 2016.

[39] Clinton W Smullen, Vidyabhushan Mohan, Anurag Nigam, Sudhanva Gurumurthi, and Mircea R Stan. Relaxing non-volatility for fast and energy-efficient stt-ram caches. In *IEEE 17th International Symposium on High Performance Computer Architecture*, pages 50–61. IEEE, IEEE, 2011.

[40] Junwhan Ahn, Sungjoo Yoo, and Kiyoung Choi. Prediction hybrid cache: An energy-efficient stt-ram cache architecture. *IEEE Transactions on Computers*, 65 (3):940–951, 2015.

[41] Nathan Binkert, Bradford Beckmann, et al. The gem5 simulator. *ACM SIGARCH computer architecture news*, 39(2):1–7, 2011.

[42] Guangyu Sun, Xiangyu Dong, Yuan Xie, Jian Li, and Yiran Chen. A novel architecture of the 3d stacked mram l2 cache for cmps. In *IEEE 15th International Symposium on High Performance Computer Architecture*, pages 239–249. IEEE, 2009.

[43] Xiaoxia Wu, Jian Li, Lixin Zhang, Evan Speight, Ram Rajamony, and Yuan Xie. Hybrid cache architecture with disparate memory technologies. *ACM SIGARCH computer architecture news*, 37(3):34–45, 2009.

[44] Jianhua Li, Chun Jason Xue, and Yinlong Xu. Stt-ram based energy-efficiency hybrid cache for cmps. In *IEEE/IFIP 19th International Conference on VLSI and System-on-Chip*, pages 31–36. IEEE, 2011.

[45] Amin Jadidi, Mohammad Arjomand, and Hamid Sarbazi-Azad. High-endurance and performance-efficient design of hybrid cache architectures through adaptive line replacement. In *IEEE/ACM International Symposium on Low Power Electronics and Design*, pages 79–84. IEEE, 2011.

[46] Ju-Hee Choi and Gi-Ho Park. Nvm way allocation scheme to reduce nvm writes for hybrid cache architecture in chip-multiprocessors. *IEEE Transactions on Parallel and Distributed Systems*, 28(10):2896–2910, 2017.

[47] Milijana Surbatovich, Brandon Lucia, and Limin Jia. Towards a formal foundation of intermittent computing. *Proceedings of the ACM on Programming Languages*, 4 (OOPSLA):1–31, 2020.

[48] Josiah Hester and Jacob Sorber. The future of sensing is batteryless, intermittent, and awesome. In *Proceedings of the 15th ACM conference on embedded network sensor systems*, pages 1–6, 2017.

[49] Texas Instruments. Msp430f5529 datasheet, 2011.

[50] Sandeep Thirumala, Arnab Raha, Sumeet Gupta, and Vijay Raghunathan. Exploring the design of energy-efficient intermittently powered systems using reconfigurable ferroelectric transistors. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 30(4):365–378, 2021.

[51] Aika Kamei, Hideharu Amano, Takuya Kojima, Daiki Yokoyama, Kimiyoshi Usami, Keizo Hiraga, Kenta Suzuki, and Kazuhiro Bessho. A variation-aware mtj store energy estimation model for edge devices with verify-and-retryable nonvolatile flip-flops. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2023.

[52] Abdullah Ash Saki, Sung Hao Lin, Mahabubul Alam, Sandeep Krishna Thirumala, Sumeet Kumar Gupta, and Swaroop Ghosh. A family of compact non-volatile flip-flops with ferroelectric fet. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 66(11):4219–4229, 2019.

[53] Kunal Korgaonkar, Ishwar Bhati, Huichu Liu, Jayesh Gaur, Sasikanth Manipatruni, Sreenivas Subramoney, Tanay Karnik, Steven Swanson, Ian Young, and Hong Wang. Density tradeoffs of non-volatile memory as a replacement for sram based last level cache. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 315–327. IEEE, 2018.

[54] Candace Walden, Devesh Singh, Meenatchi Jagasivamani, Shang Li, Luyi Kang, Mehdi Asnaashari, Sylvain Dubois, Bruce Jacob, and Donald Yeung. Monolithically

integrating non-volatile main memory over the last-level cache. *ACM Transactions on Architecture and Code Optimization (TACO)*, 18(4):1–26, 2021.

[55] Ahmet Inci, Mehmet Meric Isgenc, and Diana Marculescu. Deepnvm++: Cross-layer modeling and optimization framework of non-volatile memories for deep learning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2021.

[56] Yasuhiro Takahashi, Nazrul Anuar Nayan, Toshikazu Sekine, and Michio Yokoyama. Low-power adiabatic 9t static random access memory. *The Journal of Engineering*, 2014(6):259–264, 2014.

[57] Fen Ge, Lei Wang, Ning Wu, and Fang Zhou. A cache fill and migration policy for stt-ram-based multi-level hybrid cache in 3d cmps. *Electronics*, 8(6):639, 2019.

[58] Christopher Münch, Rajendra Bishnoi, and Mehdi B Tahoori. Multi-bit non-volatile spintronic flip-flop. In *Design, Automation & Test in Europe Conference & Exhibition*, pages 1229–1234. IEEE, IEEE, 2018.

[59] Robert Perricone, Ibrahim Ahmed, Zhaoxin Liang, Meghna G Mankalale, X Sharon Hu, Chris H Kim, Michael Niemier, Sachin S Sapatnekar, and Jian-Ping Wang. Advanced spintronic memory and logic for non-volatile processors. In *Design, Automation & Test in Europe Conference & Exhibition*, pages 972–977. IEEE, IEEE, 2017.

[60] Jaydeep P Kulkarni, Ashish Goel, Patrick Ndai, and Kaushik Roy. A read-disturb-free, differential sensing 1r/1w port, 8t bitcell array. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 19(9):1727–1730, 2011.

[61] K Dhanumjaya, M Sudha, MN Giri Prasad, and K Padmaraju. Cell stability analysis of conventional 6t dynamic 8t sram cell in 45nm technology. *International Journal of VLSI Design & Communication Systems*, 3(2):41, 2012.

[62] Dhananjaya Tripathy, Tarangini Manasneha, and Varun Das. A single-ended tg based 8t sram cell with increased data stability and less delay. In *2017 2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*, pages 1282–1285. IEEE, 2017.

[63] Takashi Ohsawa, Fumitaka Iga, Shoji Ikeda, Takahiro Hanyu, Hideo Ohno, and Tetsuo Endoh. High-density and low-power nonvolatile static random access memory using spin-transfer-torque magnetic tunnel junction. *Japanese Journal of Applied Physics*, 51(2S):02BD01, 2012.

[64] K Madhukar, V Shiva Prasad Nayak, N Ramchander, Govind Prasad, and K Manjunathachari. Optimized proposed 9t sram cell. In *2016 IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*, pages 170–174. IEEE, 2016.

[65] Johan Åkerman. Toward a universal memory. *Science*, 308(5721):508–510, 2005.

[66] Shuu'ichirou Yamamoto and Satoshi Sugahara. Nonvolatile static random access memory using magnetic tunnel junctions with current-induced magnetization switching architecture. *Japanese Journal of Applied Physics*, 48(4R):043001, 2009.

[67] K Dhanumjaya, M Sudha, MN Giri Prasad, and K Padmaraju. Cell stability analysis of conventional 6t dynamic 8t sram cell in 45nm technology. *International Journal of VLSI Design & Communication Systems*, 3(2):41, 2012.

[68] CB Kushwah and Santosh Kumar Vishvakarma. A single-ended with dynamic feedback control 8t subthreshold sram cell. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 24(1):373–377, 2016.

[69] Sayeed Ahmad, Mohit Kumar Gupta, Naushad Alam, and Mohd Hasan. Low leakage single bitline 9 t (sb9t) static random access memory. *Microelectronics Journal*, 62: 1–11, 2017.

[70] Xiangyu Dong, Cong Xu, Yuan Xie, and Norman 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*, 31 (7):994–1007, 2012.

[71] Guru Shamanna, Bhunesh S Kshatri, Raja Gaurav, YS Tew, Percy Marfatia, Y Raghavendra, and V Naik. Process technology and design parameter impact on sram bit-cell sleep effectiveness. In *23rd IEEE International SOC Conference*, pages 313–316. IEEE, 2010.

[72] CDAC Scientist-E and UP Noida. Characterization and comparison of low power sram cells. *J. Electron Devices*, 11:560–566, 2011.

[73] Mitashra Gupta and Ashutosh Nandi. Enhancing the sram performance of gate-stacked dg-mosfet. In *2017 2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*, pages 336–339. IEEE, 2017.

[74] Matthew R Guthaus et al. Mibench: A free, commercially representative embedded benchmark suite. In *Proceedings of the fourth annual IEEE international workshop on workload characterization*, pages 3–14. IEEE, IEEE, 2001.

[75] Noboru Sakimura, Yukihide Tsuji, Ryusuke Nebashi, Hiroaki Honjo, Ayuka Morioka, Kunihiko Ishihara, Keizo Kinoshita, Shunsuke Fukami, Sadahiko Miura, Naoki Kasai, et al. 10.5 a 90nm 20mhz fully nonvolatile microcontroller for standby-power-critical applications. In *IEEE International Solid-State Circuits Conference Digest of Technical Papers*, pages 184–185. IEEE, 2014.

[76] Wang Kang, Liuyang Zhang, Jacques-Olivier Klein, Youguang Zhang, Dafiné Ravelosona, and Weisheng Zhao. Reconfigurable codesign of stt-mram under process variations in deeply scaled technology. *IEEE Transactions on Electron Devices*, 62 (6):1769–1777, 2015.

[77] Vivek Seshadri, Yoongu Kim, Chris Fallin, Donghyuk Lee, Rachata Ausavarungnirun, Gennady Pekhimenko, Yixin Luo, Onur Mutlu, Phillip B Gibbons, Michael A Kozuch, et al. Rowclone: fast and energy-efficient in-dram bulk data copy and initialization. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 185–197. ACM, 2013.

[78] Siddharth Advani, Nandhini Chandramoorthy, Karthik Swaminathan, Kevin Irick, Yong Cheol Peter Cho, Jack Sampson, and Vijaykrishnan Narayanan. Refresh enabled video analytics (reva): Implications on power and performance of dram supported embedded visual systems. In *2014 IEEE 32nd International Conference on Computer Design (ICCD)*, pages 501–504. IEEE, 2014.

[79] Sujin Choi, Wookyung Sun, and Hyungsoon Shin. New modeling method for the dielectric relaxation of a dram cell capacitor. *Solid-State Electronics*, 140:29–33, 2018.

[80] Seong Keun Kim and Mihaela Popovici. Future of dynamic random-access memory as main memory. *MRS Bulletin*, 43(5):334–339, 2018.

[81] Donghyuk Lee, Yoongu Kim, Vivek Seshadri, Jamie Liu, Lavanya Subramanian, and Onur Mutlu. Tiered-latency dram: A low latency and low cost dram architecture. In *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, pages 615–626. IEEE, 2013.

[82] Kevin K Chang, Abhijith Kashyap, Hasan Hassan, Saugata Ghose, Kevin Hsieh, Donghyuk Lee, Tianshi Li, Gennady Pekhimenko, Samira Khan, and Onur Mutlu. Understanding latency variation in modern dram chips: Experimental characterization, analysis, and optimization. In *ACM SIGMETRICS Performance Evaluation Review*, volume 44, pages 323–336. ACM, 2016.

[83] Donghyuk Lee, Yoongu Kim, Gennady Pekhimenko, Samira Khan, Vivek Seshadri, Kevin Chang, and Onur Mutlu. Adaptive-latency dram: Optimizing dram timing for the common-case. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, pages 489–501. IEEE, 2015.

[84] Kevin K Chang, A Giray Yağlıkçı, Saugata Ghose, Aditya Agrawal, Niladrish Chatterjee, Abhijith Kashyap, Donghyuk Lee, Mike O'Connor, Hasan Hassan, and Onur Mutlu. Understanding reduced-voltage operation in modern dram devices: Experimental characterization, analysis, and mechanisms. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 1(1):10, 2017.

[85] Hasan Hassan, Nandita Vijaykumar, Samira Khan, Saugata Ghose, Kevin Chang, Gennady Pekhimenko, Donghyuk Lee, Oguz Ergin, and Onur Mutlu. Softmc: A flexible and practical open-source infrastructure for enabling experimental dram studies. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 241–252. IEEE, 2017.

[86] Karthik Chandrasekar, Sven Goossens, Christian Weis, Martijn Koedam, Benny Akesson, Norbert Wehn, and Kees Goossens. Exploiting expendable process-margins in drams for run-time performance optimization. In *Proceedings of the conference on Design, Automation & Test in Europe*, page 173. European Design and Automation Association, 2014.

[87] Chih-Wei Tsai, Yu-Ting Chiu, Yo-Hao Tu, and Kuo-Hsing Cheng. A wide-range all-digital delay-locked loop for double data rate synchronous dynamic random access memory application. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4. IEEE, 2018.

[88] Yiying Zhang and Steven Swanson. A study of application performance with non-volatile main memory. In *2015 31st Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–10. IEEE, 2015.

[89] Jan Lucas, Mauricio Alvarez-Mesa, Michael Andersch, and Ben Juurlink. Sparkk: Quality-scalable approximate storage in dram. In *The memory forum*, pages 1–9, 2014.

[90] Moinuddin K Qureshi, Dae-Hyun Kim, Samira Khan, Prashant J Nair, and Onur Mutlu. Avatar: A variable-retention-time (vrt) aware refresh for dram systems. In *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 427–437. IEEE, 2015.

[91] Matthias Jung, Deepak M Mathew, Christian Weis, and Norbert Wehn. Efficient reliability management in socs-an approximate dram perspective. In *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 390–394. IEEE, 2016.

[92] Hrishikesh Jayakumar, Arnab Raha, Woo Suk Lee, and Vijay Raghunathan. Quickrecall: A hw/sw approach for computing across power cycles in transiently powered computers. *ACM Journal on Emerging Technologies in Computing Systems*, 12(1):1–19, 2015.

[93] Milan Pešić, Steve Knebel, Maximilian Geyer, Sebastian Schmelzer, Ulrich Böttger, Nadiia Kolomiiets, Valeri V Afanas' ev, Kyuho Cho, Changhwa Jung, Jaewan Chang, et al. Low leakage zro2 based capacitors for sub 20 nm dynamic random access memory technology nodes. *Journal of Applied Physics*, 119(6):064101, 2016.

[94] Sunil kumar Ojha, OP Singh, GR Mishra, and PR Vaya. Design of dram having dummy cell sensing structure. In *2015 National Conference on Recent Advances in Electronics & Computer Engineering (RAECE)*, pages 234–236. IEEE, 2015.

[95] Ankush Kumar, Akanksha Pandey, Praveen Kumar Sahu, Lalit Chandra, R Dwivedi, and VN Mishra. Design of dram sense amplifier using 45nm technology. In *2018 International Symposium on Devices, Circuits and Systems (ISDCS)*, pages 1–5. IEEE, 2018.

[96] Song Liu, Karthik Pattabiraman, Thomas Moscibroda, and Benjamin G Zorn. Flikker: saving dram refresh-power through critical data partitioning. *ACM SIGPLAN Notices*, 47(4):213–224, 2012.

[97] Arnab Raha, Hrishikesh Jayakumar, Soubhagya Sutar, and Vijay Raghunathan. Quality-aware data allocation in approximate dram. In *Proceedings of the 2015 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, pages 89–98. IEEE Press, 2015.

[98] Karthik Chandrasekar, Christian Weis, Benny Akesson, Norbert Wehn, and Kees Goossens. Towards variation-aware system-level power estimation of drams: an empirical approach. In *Proceedings of the 50th Annual Design Automation Conference*, page 23. ACM, 2013.

[99] Matthew Poremba, Tao Zhang, and Yuan Xie. Nvmain 2.0: A user-friendly memory simulator to model (non-) volatile memory systems. *IEEE Computer Architecture Letters*, 14(2):140–143, 2015.

[100] Hyungjin Kim, Jong-Ho Lee, and Byung-Gook Park. Asymmetric dual-gate-structured one-transistor dynamic random access memory cells for retention characteristics improvement. *Applied Physics Express*, 9(8):084201, 2016.

[101] Fei Xia, Dejun Jiang, Jin Xiong, and Ninghui Sun. Hikv: a hybrid index key-value store for dram-nvm memory systems. In *2017 {USENIX} Annual Technical Conference ({USENIX}{ATC} 17)*, pages 349–362, 2017.

[102] Takayuki Kawahara, Kenchi Ito, Riichiro Takemura, and Hideo Ohno. Spin-transfer torque ram technology: Review and prospect. *Microelectronics Reliability*, 52(4): 613–627, 2012.

[103] Roy Bell, Jiaxi Hu, and RH Victora. Dual referenced composite free layer design for improved switching efficiency of spin-transfer torque random access memory. *IEEE Electron Device Letters*, 37(9):1108–1111, 2016.

[104] Ju Hee Choi and Jong Wook Kwak. Fast writeable block-aware cache update policy for spin-transfer-torque ram. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(4):1236–1249, 2017.

[105] Sparsh Mittal. Mitigating read disturbance errors in stt-ram caches by using data compression. In *Nanoelectronics*, pages 133–152. Elsevier, 2019.

[106] R Sbiaa, H Meng, and SN Piramanayagam. Materials with perpendicular magnetic anisotropy for magnetic random access memory. *physica status solidi (RRL)–Rapid Research Letters*, 5(12):413–419, 2011.

[107] Fabian Oboril, Rajendra Bishnoi, Mojtaba Ebrahimi, and Mehdi B Tahoori. Evaluation of hybrid memory technologies using sot-mram for on-chip cache hierarchy. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(3):367–380, 2015.

[108] Said Tehrani, JM Slaughter, E Chen, M Durlam, J Shi, and M DeHerren. Progress and outlook for mram technology. *IEEE Transactions on Magnetics*, 35(5): 2814–2819, 1999.

[109] Sabpreet Bhatti, Rachid Sbiaa, Atsufumi Hirohata, Hideo Ohno, Shunsuke Fukami, and SN Piramanayagam. Spintronics based random access memory: a review. *Materials Today*, 20(9):530–548, 2017.

[110] Tai Min and Po-Kang Wang. Thermally assisted integrated mram design and process for its manufacture, February 3 2009. US Patent 7,486,545.

[111] Jimmy Zhu and Gary A Prinz. Vmram memory holds both promise and challenge. *Data Storage*, 40, 2000.

[112] Bing-Chen Wu and Tsung-Te Liu. Circuit sensing techniques in magnetoresistive random-access memory. *Journal of Low Power Electronics*, 14(2):206–216, 2018.

[113] M Hosomi, H Yamagishi, T Yamamoto, K Bessho, Y Higo, K Yamane, H Yamada, M Shoji, H Hachino, C Fukumoto, et al. A novel nonvolatile memory with spin torque transfer magnetization switching: Spin-ram. In *IEEE InternationalElectron Devices Meeting, 2005. IEDM Technical Digest.*, pages 459–462. IEEE, 2005.

[114] AV Khvalkovskiy, D Apalkov, S Watts, R Chepulskii, RS Beach, A Ong, X Tang, A Driskill-Smith, WH Butler, PB Visscher, et al. Basic principles of stt-mram cell operation in memory arrays. *Journal of Physics D: Applied Physics*, 46(7):074001, 2013.

[115] Hiroki Noguchi, Kazutaka Ikegami, Keiichi Kushida, Keiko Abe, Shogo Itai, Satoshi Takaya, Naoharu Shimomura, Junichi Ito, Atsushi Kawasumi, Hiroyuki Hara, et al. 7.5 a 3.3 ns-access-time 71.2 μw/mhz 1mb embedded stt-mram using physically eliminated read-disturb scheme and normally-off memory architecture. In *2015 IEEE International Solid-State Circuits Conference-(ISSCC) Digest of Technical Papers*, pages 1–3. IEEE, 2015.

[116] Sara Choi, Taehui Na, Seong-Ook Jung, Jung Pill Kim, and Seung H Kang. Area-optimal sensing circuit designs in deep submicrometer stt-ram. In *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1246–1249. IEEE, 2016.

[117] Hong-xi Liu, Yusuke Honda, Tomoyuki Taira, Ken-ichi Matsuda, Masashi Arita, Tetsuya Uemura, and Masafumi Yamamoto. Giant tunneling magnetoresistance in epitaxial co2mnsi/mgo/co2mnsi magnetic tunnel junctions by half-metallicity of co2mnsi and coherent tunneling. *Applied Physics Letters*, 101(13):132418, 2012.

[118] Ki-Seung Lee, Seo-Won Lee, Byoung-Chul Min, and Kyung-Jin Lee. Threshold current for switching of a perpendicular magnetic layer induced by spin hall effect. *Applied Physics Letters*, 102(11):112410, 2013.

[119] Jeffrey S Vetter and Sparsh Mittal. Opportunities for nonvolatile memory systems in extreme-scale high-performance computing. *Computing in Science & Engineering*, 17(2):73–82, 2015.

[120] Fatemeh Arezoomand, Arghavan Asad, Mahdi Fazeli, Mahmood Fathy, and Farah Mohammadi. Energy aware and reliable stt-ram based cache design for 3d embedded chip-multiprocessors. In *2017 12th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, pages 1–8. IEEE, 2017.

[121] Kangwook Jo and Hongil Yoon. Variation-tolerant sensing circuit for ultralow-voltage operation of spin-torque transfer magnetic ram. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 64(5):570–574, 2017.

[122] Dongku Kang, Woopyo Jeong, Chulbum Kim, Doo-Hyun Kim, Yong Sung Cho, Kyung-Tae Kang, Jinho Ryu, Kyung-Min Kang, Sungyeon Lee, Wandong Kim, et al. 256 gb 3 b/cell v-nand flash memory with 48 stacked wl layers. *IEEE Journal of Solid-State Circuits*, 52(1):210–217, 2017.

[123] Ki-Tae Park, Dae-Seok Byeon, and Doo-Hyun Kim. A world's first product of three-dimensional vertical nand flash memory and beyond. In *2014 14th Annual Non-Volatile Memory Technology Symposium (NVMTS)*, pages 1–5. IEEE, 2014.

[124] Yu Cai, Yixin Luo, Erich F Haratsch, Ken Mai, and Onur Mutlu. Data retention in mlc nand flash memory: Characterization, optimization, and recovery. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, pages 551–563. IEEE, 2015.

[125] Tomoharu Tanaka, Mark Helm, Tommaso Vali, Ramin Ghodsi, Koichi Kawai, Jae-Kwan Park, Shigekazu Yamada, Feng Pan, Yuichi Einaga, Ali Ghalam, et al. 7.7 a 768gb 3b/cell 3d-floating-gate nand flash memory. In *2016 IEEE International Solid-State Circuits Conference (ISSCC)*, pages 142–144. IEEE, 2016.

[126] K Parat and A Goda. Scaling trends in nand flash. In *2018 IEEE International Electron Devices Meeting (IEDM)*, pages 2–1. IEEE, 2018.

[127] Kyung Jean Yoon, Yumin Kim, and Cheol Seong Hwang. What will come after v-nand—vertical resistive switching memory? *Advanced Electronic Materials*, page 1800914, 2019.

[128] Ziqi Fan and Dongchul Park. Extending ssd lifespan with comprehensive non-volatile memory-based write buffers. *Journal of Computer Science and Technology*, 34(1): 113–132, 2019.

[129] Yu Cai, Saugata Ghose, Yixin Luo, Ken Mai, Onur Mutlu, and Erich F Haratsch. Vulnerabilities in mlc nand flash memory programming: experimental analysis, exploits, and mitigation techniques. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 49–60. IEEE, 2017.

[130] Zhibo Wang, Fang Su, Yiqun Wang, Zewei Li, Xueqing Li, Ryuji Yoshimura, Takashi Naiki, Takashi Tsuwa, Takahiko Saito, Zhongjun Wang, et al. A 130nm feram-based parallel recovery nonvolatile soc for normally-off operations with 3.9× faster running speed and 11× higher energy efficiency using fast power-on detection and nonvolatile radio controller. In *Symposium on VLSI Circuits*, pages C336–C337. IEEE, 2017.

[131] Tetsuo Endoh. Embedded nonvolatile memory with stt-mrams and its application for nonvolatile brain-inspired vlsis. In *2017 International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*, pages 1–3. IEEE, 2017.

[132] Laura M Grupp, John D Davis, and Steven Swanson. The bleak future of nand flash memory. In *Proceedings of the 10th USENIX conference on File and Storage Technologies*, pages 2–2. USENIX Association, 2012.

[133] Hyun Kook Park, Tae Woo Oh, and Seong-Ook Jung. A novel heat-aware write method with optimized heater material and structure in sub-20 nm pram for low energy operation. In *2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pages 273–276. IEEE, 2018.

[134] Kwang-Jin Lee, Beak-Hyung Cho, Woo-Yeong Cho, Sangbeom Kang, Byung-Gil Choi, Hyung-Rok Oh, Chang-Soo Lee, Hye-Jin Kim, Joon-Min Park, Qi Wang, et al. A 90 nm 1.8 v 512 mb diode-switch pram with 266 mb/s read throughput. *IEEE Journal of Solid-State Circuits*, 43(1):150–162, 2008.

[135] W Zhao, E Belhaire, V Javerliac, C Chappert, and B Dieny. A non-volatile flip-flop in magnetic fpga chip. In *International Conference on Design and Test of Integrated Systems in Nanoscale Technology, 2006. DTIS 2006.*, pages 323–326. IEEE, 2006.

[136] Junyoung Ko, Younghwi Yang, Jisu Kim, Younghoon Oh, HK Park, and Seong-ook Jung. Incremental bitline voltage sensing scheme with half-adaptive threshold

reference scheme in mlc pram. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 64(6):1444–1455, 2017.

[137] Richard Fackenthal, Makoto Kitagawa, Wataru Otsuka, Kirk Prall, Duane Mills, Keiichi Tsutsui, Jahanshir Javanifard, Kerry Tedrow, Tomohito Tsushima, Yoshiyuki Shibahara, et al. 19.7 a 16gb reram with 200mb/s write and 1gb/s read in 27nm technology. In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pages 338–339. IEEE, 2014.

[138] Akifumi Kawahara, Ryotaro Azuma, Yuuichirou Ikeda, Ken Kawai, Yoshikazu Katoh, Yukio Hayakawa, Kiyotaka Tsuji, Shinichi Yoneda, Atsushi Himeno, Kazuhiko Shimakawa, et al. An 8 mb multi-layered cross-point reram macro with 443 mb/s write throughput. *IEEE Journal of Solid-State Circuits*, 48(1):178–185, 2013.

[139] Ping Chi, Shuangchen Li, Cong Xu, Tao Zhang, Jishen Zhao, Yongpan Liu, Yu Wang, and Yuan Xie. Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory. In *ACM SIGARCH Computer Architecture News*, volume 44, pages 27–39. IEEE Press, 2016.

[140] Majed Valad Beigi and Gokhan Memik. Thor: Thermal-aware optimizations for extending reram lifetime. In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 670–679. IEEE, 2018.

[141] Cong Xu, Pai-Yu Chen, Dimin Niu, Yang Zheng, Shimeng Yu, and Yuan Xie. Architecting 3d vertical resistive memory for next-generation storage systems. In *2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 55–62. IEEE, 2014.

[142] Alessandro Grossi, Elisa Vianello, Mohamed M Sabry, Marios Barlas, Laurent Grenouillet, Jean Coignus, Edith Beigne, Tony Wu, Binh Q Le, Mary K Wootters, et al. Resistive ram endurance: Array-level characterization and correction techniques targeting deep learning applications. *IEEE Transactions on Electron Devices*, 66(3):1281–1288, 2019.

[143] Boris Hudec, I-Ting Wang, Wei-Li Lai, Che-Chia Chang, Peter Jančovič, Karol Fröhlich, Matej Mičušík, Mária Omastová, and Tuo-Hung Hou. Interface engineered hfo2-based 3d vertical reram. *Journal of Physics D: Applied Physics*, 49(21):215102, 2016.

[144] Ting-Chang Chang, Kuan-Chang Chang, Tsung-Ming Tsai, Tian-Jian Chu, and Simon M Sze. Resistance random access memory. *Materials Today*, 19(5):254–264, 2016.

[145] Po-Hsun Chen, Ting-Chang Chang, Kuan-Chang Chang, Tsung-Ming Tsai, Chih-Hung Pan, Chih-Cheng Shih, Cheng-Hsien Wu, Cheng-Chi Yang, Yu-Ting Su,

Chih-Yang Lin, et al. Obtaining lower forming voltage and self-compliance current by using a nitride gas/indium–tin oxide insulator in resistive random access memory. *IEEE Transactions on Electron Devices*, 63(12):4769–4775, 2016.

[146] Zizhen Jiang, Yi Wu, Shimeng Yu, Lin Yang, Kay Song, Zia Karim, and H-S Philip Wong. A compact model for metal–oxide resistive random access memory with experiment verification. *IEEE Transactions on Electron Devices*, 63(5):1884–1892, 2016.

[147] Po-Hsun Chen, Kuan-Chang Chang, Ting-Chang Chang, Tsung-Ming Tsai, Chih-Hung Pan, Yu-Ting Su, Cheng-Hsien Wu, Wan-Ching Su, Chih-Cheng Yang, Min-Chen Chen, et al. Modifying indium-tin-oxide by gas cosputtering for use as an insulator in resistive random access memory. *IEEE Transactions on Electron Devices*, 63(11):4288–4294, 2016.

[148] Hyojung Kim, Ji Su Han, Sun Gil Kim, Soo Young Kim, and Ho Won Jang. Halide perovskites for resistive random-access memories. *Journal of Materials Chemistry C*, 2019.

[149] Nianduan Lu, Pengxiao Sun, Ling Li, Qi Liu, Shibing Long, Lv Hangbing, and Ming Liu. Thermal effect on endurance performance of 3-dimensional rram crossbar array. *Chinese Physics B*, 25(5):056501, 2016.

[150] Xiaoxia Wu, Jian Li, Lixin Zhang, Evan Speight, Ram Rajamony, and Yuan Xie. Design exploration of hybrid caches with disparate memory technologies. *ACM Transactions on Architecture and Code Optimization*, 7(3):1–34, 2010.

[151] Zhe Wang, Daniel A Jiménez, Cong Xu, Guangyu Sun, and Yuan Xie. Adaptive placement and migration policy for an stt-ram-based hybrid cache. In *IEEE 20th International Symposium on High Performance Computer Architecture*, pages 13–24. IEEE, IEEE, 2014.

[152] Jun Yao, Jianliang Ma, Tianzhou Chen, and Tongsen Hu. An energy-efficient scheme for stt-ram l1 cache. In *IEEE 10th International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing*, pages 1345–1350. IEEE, IEEE, 2013.

[153] N Do, J Kim, S Lemke, L Tee, Y Tkachev, X Liu, P Ghazavi, F Zhou, B Villard, S Jourba, et al. Scaling split-gate flash memory technology for advanced mcu and emerging applications. In *2019 IEEE 11th International Memory Workshop (IMW)*, pages 1–4. IEEE, 2019.

[154] Hrishikesh Jayakumar, Arnab Raha, and Vijay Raghunathan. Quickrecall: A low overhead hw/sw approach for enabling computations across power cycles in transiently powered computers. In *2014 27th International Conference on VLSI*

Design and 2014 13th International Conference on Embedded Systems, pages 330–335. IEEE, 2014.

[155] HanBin Yoon, Justin Meza, Rachata Ausavarungnirun, et al. Row buffer locality aware caching policies for hybrid memories. In *2012 IEEE 30th ICCD*, pages 337–344. IEEE, 2012.

[156] Moinuddin K Qureshi, John Karidis, , et al. Enhancing lifetime and security of pcm-based main memory with start-gap wear leveling. In *2009 42nd Annual IEEE/ACM MICRO*, pages 14–23. IEEE, 2009.

[157] Saad Ahmed, Naveed Anwar Bhatti, Muhammad Hamad Alizai, Junaid Haroon Siddiqui, and Luca Mottola. Efficient intermittent computing with differential checkpointing. In *Proceedings of the 20th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems*, pages 70–81, 2019.

[158] Saad Ahmed, Naveed Anwar Bhatti, Muhammad Hamad Alizai, Junaid Haroon Siddiqui, and Luca Mottola. Fast and energy-efficient state checkpointing for intermittent computing. *ACM Transactions on Embedded Computing Systems (TECS)*, 19(6):1–27, 2020.

[159] Mimi Xie, Chen Pan, and Chun Jason Xue. A novel stt-ram-based hybrid cache for intermittently powered processors in iot devices. *IEEE Micro*, 39(1):24–32, 2018.

[160] Priyanka Singla and Smruti R Sarangi. A survey and experimental analysis of checkpointing techniques for energy harvesting devices. *Journal of Systems Architecture*, page 102464, 2022.

[161] Sandeep Krishna Thirumala, Arnab Raha, Vijay Raghunathan, and Sumeet Kumar Gupta. Ips-cim: Enhancing energy efficiency of intermittently-powered systems with compute-in-memory. In *2020 IEEE 38th International Conference on Computer Design (ICCD)*, pages 368–376. IEEE, 2020.

[162] Fan Zhang, Yanqing Zhang, et al. A batteryless 19μw mics/ism-band energy harvesting body area sensor node soc. In *2012 IEEE International Solid-State Circuits Conference*, pages 298–300. IEEE, 2012.

[163] Domenico Balsamo, Anup Das, et al. Graceful performance modulation for power-neutral transient computing systems. *IEEE TCAD*, 35(5):738–749, 2016.

[164] Hehe Li, Yongpan Liu, Qinghang Zhao, et al. An energy efficient backup scheme with low inrush current for nonvolatile sram in energy harvesting sensor nodes. In *2015 DATE*, pages 7–12. IEEE, 2015.

[165] Xiao Sheng, Yiqun Wang, Yongpan Liu, and Huazhong Yang. Spac: A segment-based parallel compression for backup acceleration in nonvolatile processors. In *2013 DATE*, pages 865–868. IEEE, 2013.

[166] Yiqun Wang, Yongpan Liu, Shuangchen Li, et al. Pacc: A parallel compare and compress codec for area reduction in nonvolatile processors. *IEEE TVLSI*, 22(7): 1491–1505, 2013.

[167] Mimi Xie, Mengying Zhao, Chen Pan, Jingtong Hu, Yongpan Liu, and Chun Jason 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*, pages 1–6. ACM New York, NY, USA, 2015.

[168] Yongpan Liu, Fang Suy, Zhibo Wangy, and Huazhong Yang. Design exploration of inrush current aware controller for nonvolatile processor. In *2015 IEEE Non-Volatile Memory System and Applications Symposium*, pages 1–6. IEEE, IEEE, 2015.

[169] Yang Zhou, Mengying Zhao, Lei Ju, Chun Jason Xue, Xin Li, and Zhiping Jia. Energy-aware morphable cache management for self-powered non-volatile processors. In *IEEE 23rd International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 1–7. IEEE, IEEE, 2017.

[170] Mimi Xie, Mengying Zhao, Chen Pan, Hehe Li, Yongpan Liu, Youtao Zhang, Chun Jason Xue, and Jingtong Hu. Checkpoint aware hybrid cache architecture for nv processor in energy harvesting powered systems. In *International Conference on Hardware/Software Codesign and System Synthesis*, pages 1–10. IEEE, IEEE, 2016.

[171] Benjamin Ransford, Jacob Sorber, and Kevin Fu. Mementos: System support for long-running computation on rfid-scale devices. In *Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems*, pages 159–170, 2011.

[172] Domenico Balsamo, Alex S Weddell, Anup Das, Alberto Rodriguez Arreola, Davide Brunelli, Bashir M Al-Hashimi, Geoff V Merrett, and Luca 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.

[173] Vito Kortbeek, Kasim Sinan Yildirim, Abu Bakar, Jacob Sorber, Josiah Hester, and Przemysław Pawełczak. Time-sensitive intermittent computing meets legacy software. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 85–99, 2020.

[174] Fang Su, Yongpan Liu, Yiqun Wang, and Huazhong Yang. A ferroelectric nonvolatile processor with 46 \$\mu$ s system-level wake-up time and 14\$\mu$ s sleep time for energy harvesting applications. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 64(3):596–607, 2016.

[175] Jongouk Choi, Hyunwoo Joe, Yongjoo Kim, and Changhee Jung. Achieving stagnation-free intermittent computation with boundary-free adaptive execution. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 331–344. IEEE, 2019.

[176] Jue Wang, Xiangyu Dong, Yuan Xie, and Norman P Jouppi. Endurance-aware cache line management for non-volatile caches. *ACM Transactions on Architecture and Code Optimization (TACO)*, 11(1):1–25, 2014.

[177] Yiran Chen, Weng-Fai Wong, Hai Li, and Cheng-Kok Koh. Processor caches built using multi-level spin-transfer torque ram cells. In *IEEE/ACM International Symposium on Low Power Electronics and Design*, pages 73–78. IEEE, 2011.

[178] Yu-Ting Chen, Jason Cong, Hui Huang, Bin Liu, Chunyue Liu, Miodrag Potkonjak, and Glenn Reinman. Dynamically reconfigurable hybrid cache: An energy-efficient last-level cache design. In *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 45–50. IEEE, 2012.

[179] Sophiane Senni, Lionel Torres, Gilles Sassatelli, and Abdoulaye Gamatie. Non-volatile processor based on mram for ultra-low-power iot devices. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 13(2):1–23, 2016.

[180] Keni Qiu, Mengying Zhao, Zhenge Jia, Jingtong Hu, Chun Jason Xue, Kaisheng Ma, Xueqing Li, Yongpan Liu, and Vijaykrishnan Narayanan. Design insights of non-volatile processors and accelerators in energy harvesting systems. In *Proceedings of the 2020 on Great Lakes Symposium on VLSI*, pages 369–374, 2020.

[181] Markus Koesler and Franz Graf. Programming a flash-based msp430 using the jtag interface. *SLAA149, TEXAS INSTRUMENTS*, pages 4–13, 2002.

[182] Ferhat Erata, Eren Yildiz, Arda Goknil, Kasim Sinan Yildirim, Jakub Szefer, Ruzica Piskac, and Gokcin Sezgin. Etap: Energy-aware timing analysis of intermittent programs. *ACM TECS*, 22(2):1–31, 2023.

[183] Andrea Maioli and Luca Mottola. Alfred: Virtual memory for intermittent computing. In *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*, pages 261–273, 2021.

[184] Benjamin Ransford, Jacob Sorber, and Kevin Fu. Mementos: System support for long-running computation on rfid-scale devices. In *Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems*, pages 159–170, 2011.

[185] Mohammad Alshboul, Hussein Elnawawy, Reem Elkhouly, Keiji Kimura, James Tuck, and Yan Solihin. Efficient checkpointing with recompute scheme for non-volatile main memory. *ACM Transactions on Architecture and Code Optimization (TACO)*, 16(2):1–27, 2019.

[186] Vivy Suhendra, Chandrashekar Raghavan, and Tulika Mitra. Integrated scratchpad memory optimization and task scheduling for mpsoc architectures. In *Proceedings of the 2006 international conference on Compilers, architecture and synthesis for embedded systems*, pages 401–410, 2006.

[187] Lian Li, Hui Feng, and Jingling Xue. Compiler-directed scratchpad memory management via graph coloring. *ACM Transactions on Architecture and Code Optimization (TACO)*, 6(3):1–17, 2009.

[188] Yibo Guo, Qingfeng Zhuge, Jingtong Hu, Juan Yi, Meikang Qiu, and Edwin H-M Sha. Data placement and duplication for embedded multicore systems with scratch pad memory. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(6):809–817, 2013.

[189] Prasenjit Chakraborty, Preeti Ranjan Panda, and Sandeep Sen. Partitioning and data mapping in reconfigurable cache and scratchpad memory–based architectures. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 22(1): 1–25, 2016.

[190] Mahmut Kandemir, J Ramanujam, Mary Jane Irwin, Narayanan Vijaykrishnan, Ismail Kadayif, and Amisha Parikh. Dynamic management of scratch-pad memory space. In *Proceedings of the 38th Design Automation Conference (IEEE Cat. No. 01CH37232)*, pages 690–695. IEEE, 2001.

[191] Morteza Mohajjel Kafshdooz and Alireza Ejlali. Dynamic shared spm reuse for real-time multicore embedded systems. *ACM Transactions on Architecture and Code Optimization (TACO)*, 12(2):1–25, 2015.

[192] Reem Elkhouly, Mohammad Alshboul, Akihiro Hayashi, Yan Solihin, and Keiji Kimura. Compiler-support for critical data persistence in nvm. *ACM Transactions on Architecture and Code Optimization (TACO)*, 16(4):1–25, 2019.

[193] Sumesh Udayakumaran, Angel Dominguez, and Rajeev Barua. Dynamic allocation for scratch-pad memory using compile-time decisions. *ACM Transactions on Embedded Computing Systems (TECS)*, 5(2):472–511, 2006.

[194] Jean-Francois Deverge and Isabelle Puaut. Wcet-directed dynamic scratchpad memory allocation of data. In *19th Euromicro Conference on Real-Time Systems (ECRTS'07)*, pages 179–190. IEEE, 2007.

[195] Bernhard Egger, Jaejin Lee, and Heonshik Shin. Dynamic scratchpad memory management for code in portable systems with an mmu. *ACM Transactions on Embedded Computing Systems (TECS)*, 7(2):1–38, 2008.

[196] Manish Verma, Lars Wehmeyer, and Peter Marwedel. Dynamic overlay of scratchpad memory for energy minimization. In *Proceedings of the 2nd IEEE/ACM/IFIP*

international conference on Hardware/software codesign and system synthesis, pages 104–109, 2004.

[197] Kasım Sinan Yıldırım, Amjad Yousef Majid, Dimitris Patoukas, Koen Schaper, Premise Pawelczak, and Josiah Hester. Ink: Reactive kernel for tiny batteryless sensors. In *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*, pages 41–53, 2018.

[198] Moinuddin K Qureshi, Michele M Franceschini, et al. Preset: Improving performance of phase change memories by exploiting asymmetry in write times. 40(3):380–391, 2012.

[199] Mirae Kim, Jungkeol Lee, Youngil Kim, and Yong Ho Song. An analysis of energy consumption under various memory mappings for fram-based iot devices. In *2018 IEEE 4th World Forum on Internet of Things (WF-IoT)*, pages 574–579. IEEE, 2018.

[200] Adwait Jog, Asit K Mishra, Cong Xu, Yuan Xie, Vijaykrishnan Narayanan, Ravishankar Iyer, and Chita R Das. Cache revive: Architecting volatile stt-ram caches for enhanced performance in cmps. In *Design Automation Conference*, pages 243–252. IEEE, 2012.

[201] Chen Pan, Mimi Xie, Jingtong Hu, Yiran Chen, and Chengmo Yang. 3m-pcm: Exploiting multiple write modes mlc phase change main memory in embedded systems. In *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis*, pages 1–10, 2014.

[202] Kaisheng Ma, Yang Zheng, Shuangchen Li, Karthik Swaminathan, Xueqing Li, Yongpan Liu, Jack Sampson, Yuan Xie, and Vijaykrishnan Narayanan. Architecture exploration for ambient energy harvesting nonvolatile processors. In *IEEE 21st International Symposium on High Performance Computer Architecture*, pages 526–537. IEEE, IEEE, 2015.

[203] Hrishikesh Jayakumar, Arnab Raha, and Vijay Raghunathan. Energy-aware memory mapping for hybrid fram-sram mcus in iot edge devices. In *2016 29th International Conference on VLSI Design and 2016 15th International Conference on Embedded Systems (VLSID)*, pages 264–269. IEEE, 2016.

[204] Fang Su, Yongpan Liu, Yiqun Wang, and Huazhong Yang. A ferroelectric nonvolatile processor with 46 $\backslash\mu$ s system-level wake-up time and 14$\backslash\mu$ s sleep time for energy harvesting applications. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 64(3):596–607, 2016.

[205] Ricardo Gonzalez and Mark Horowitz. Energy dissipation in general purpose microprocessors. *IEEE Journal of solid-state circuits*, 31(9):1277–1284, 1996.

[206] Yu-Der Chih, Yi-Chun Shih, Chia-Fu Lee, Yen-An Chang, Po-Hao Lee, Hon-Jarn Lin, Yu-Lin Chen, Chieh-Pu Lo, Meng-Chun Shih, Kuei-Hung Shen, et al. 13.3 a 22nm 32mb embedded stt-mram with 10ns read speed, 1m cycle write endurance, 10 years retention at 150 c and high immunity to magnetic field interference. In *2020 IEEE International Solid-State Circuits Conference-(ISSCC)*, pages 222–224. IEEE, 2020.

[207] Benjamin C Lee, Engin Ipek, Onur Mutlu, and Doug Burger. Architecting phase change memory as a scalable dram alternative. In *Proceedings of the 36th annual international symposium on Computer architecture*, pages 2–13, 2009.