Optimizing Embedded Software of Self-Powered IoT Edge Devices for Transient Computing

by

Chen Pan

BS, Wuhan University of Science and Engineering, 2009

MS, Oklahoma State University, 2017

Submitted to the Graduate Faculty of

the Swanson School of Engineering in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2019

UNIVERSITY OF PITTSBURGH

SWANSON SCHOOL OF ENGINEERING

This dissertation was presented

by

Chen Pan

It was defended on

May 30, 2019

and approved by

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

Zhi-Hong Mao, Ph.D., Professor
Department of Electrical and Computer Engineering

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

Alex Jones, Ph.D., Professor
Department of Electrical and Computer Engineering

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

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

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

Zhi-Hong Mao, Ph.D., Professor
Department of Electrical and Computer Engineering

Optimizing Embedded Software of Self-Powered IoT Edge Devices for Transient Computing

Chen Pan, PhD

University of Pittsburgh, 2019

IoT edge computing becomes increasingly popular as it can mitigate the burden of cloud servers significantly by offloading tasks from the cloud to the edge. Currently, there are trillions of edge devices all over the world, and a majority of them work under power-constrained scenarios such as outdoor environmental monitoring. Considering the cost and sustainability, in the long run, self-powering through energy harvesting technology is preferred for these IoT edge devices. Nevertheless, a common and critical drawback of self-powered IoT edge devices is that their runtime states in volatile memory such as SRAM will be lost during the power outage. Thanks to the state-of-the-art non-volatile processor (NVP), the volatile states can be saved into the on-chip non-volatile memory before the power outage and recovered when harvesting power becomes available. Yet the potential of a self-powered IoT edge device is still hindered by the intrinsic low energy efficiency and reliability.

To fully exert the potentials of existing self-powered IoT edge devices, this dissertation aims at optimizing the energy efficiency and reliability of self-powered IoT edge devices through several software approaches. First, to prevent execution progress loss during the power outage, NVP-aware task schedulers are proposed to maximize the overall task execution progress especially for the atomic tasks of which the unfinished progress is subjected to loss regardless of having checkpoints. Second, to minimize checkpointing overhead on non-volatile memory, an intelligent checkpointing scheme is proposed which can not only ensure a successful checkpointing but also predict the necessity of conducting checkpointing to avoid excessive checkpointing. Third, to avoid inappropriate runtime MCU clock frequency with low energy utility, a CPU frequency modulator is proposed which adjusts the runtime CPU clock frequency adaptively. Finally, to thrive in ultra-low harvesting power scenarios, a light-weight software paradigm is proposed to help maximize the energy extraction rate. Besides, checkpointing is also optimized for more energy-efficient and light-weight operation.

Table of Contents

1.0 Introduction 1

1.1 Research Motivation 1

1.2 Existing Work 3

1.2.1 Optimizing Power Regulator 3

1.2.2 Optimizing Checkpointing 4

1.2.3 Optimizing Run-time Execution 5

1.3 Research Contributions 6

1.4 Dissertation Organization 7

2.0 Technical Background 8

2.1 System Architecture 8

2.2 Basic Checkpointing 9

2.3 Voltage Monitoring 10

2.4 Sleep/Wake-up Managing 11

3.0 Atomic Task Aware Task Scheduling 13

3.1 Motivation 13

3.2 Related Work 16

3.3 System Overview 17

3.4 Power Prediction 18

3.5 Independent NVP Task Scheduler 21

3.6 Dependent NVP Task Scheduler 26

3.7 Experiments 30

3.7.1 Evaluation of NVP Scheduler 30

3.7.1.1 Hardware Platform 30

3.7.1.2 Power Traces 31

3.7.1.3 Software Setup 31

3.7.1.4 Energy Consumption Analysis 32

3.7.1.5 Sleep Mode vs. Working Mode 32
3.7.1.6 Progress Comparison 34
3.7.1.7 Efficiency and Overhead 38
3.7.2 Evaluation of NTS Scheduler 40
3.7.2.1 Hardware Platform 40
3.7.2.2 Power Trace 41
3.7.2.3 Software Setup 41
3.7.2.4 Benchmark Setup 43
3.8 Summary 45
4.0 Intelligent Checkpointing Scheme 46
4.1 Motivation 46
4.2 Related Work 46
4.3 Statisitic based Checkpointing Avoidance 48
4.3.1 Secure Checkpointing 52
4.4 Experiments 53
4.4.1 Experimental Setup 53
4.4.1.1 Hardware Platform 53
4.4.1.2 Power Trace 53
4.4.1.3 Software Setup 53
4.4.1.4 Performance of TCW 54
4.5 Summary 56
5.0 CPU Frequency Modulation 57
5.1 Motivation 57
5.2 Modeling and Analysis 58
5.3 Frequency Modulator 60
5.3.1 Off-line Stage 60
5.3.2 On-line Stage 61
5.4 Experiments 61
5.4.1 Experimental Setup 61
5.4.1.1 Hardware Platform 61

5.4.1.2 Power Trace . . . 62
5.4.1.3 Benchmarks . . . 62
5.4.2 Energy Utility Evaluation . . . 63
5.4.2.1 Overhead Analysis . . . 64
5.5 Summary . . . 65
6.0 Thriving on Ultra-Low Harvesting Power . . . 66
6.1 Motivation . . . 66
6.1.1 Wake-up Voltage Determines Efficiency . . . 68
6.1.2 Routines vs. Efficiency . . . 71
6.2 Related Work . . . 73
6.2.1 System Architecture . . . 75
6.3 Modeling and Analysis for $\xi(v_{Wak})$ . . . 75
6.3.0.1 Hardware Energy Efficiency . . . 75
6.3.0.2 Software Energy Efficiency . . . 78
6.3.0.3 Execution Energy Efficiency . . . 78
6.3.0.4 Influence of Checkpointing Data Size ω . . . 79
6.3.0.5 Influence of Wake-up Voltage v . . . 79
6.4 Modeling and Analysis for $\xi(v_{Wak}, v_{slp})$ . . . 82
6.4.1 Energy Modeling . . . 82
6.4.1.1 Harvested Energy . . . 82
6.4.1.2 Hardware Energy Overhead . . . 82
6.4.1.3 Software Overhead . . . 83
6.4.1.4 Effective Energy . . . 84
6.4.2 Efficiency Analysis . . . 84
6.4.2.1 Formulation of ξ . . . 84
6.4.2.2 Optimization of ζ . . . 85
6.4.2.3 Validation of modeling . . . 86
6.5 Optimizing Voltages for Sleep/Wakeup . . . 88
6.5.1 Routine Handler . . . 88
6.6 Experiments . . . 92

6.6.1 Experimental Setup 92
6.6.1.1 Hardware Platform 92
6.6.1.2 Power Trace 93
6.6.1.3 Benchmarks 93
6.6.2 Experimental Evaluation 93
6.6.2.1 Observation of $\zeta(v_{slp}, v_{wak})$ 93
6.6.2.2 Energy Efficiency Evaluation 95
6.7 Summary 96
7.0 Conclusion 97
Bibliography 98

List of Tables

1 Task Information (Tab) 23
2 Task Information 32

List of Figures

1 System Architecture of Self-Powered Non-Volatile IoT Edge Device 8
2 NVP Architecture 9
3 Atomic Task B is Unfinished Before The Power Outage Resulting in Progress Setback. 14
4 Prioritizing atomic Task B to Finish Before The Power Outage Results in Progress Maximization. 15
5 System Architecture 18
6 NVP Scheduler Overview 22
7 Example DAG 29
8 Power Trace Used for Experiments 31
9 Power Consumption of Benchmarks 33
10 Power Consumption of Low Power Mode vs Working Mode 33
11 Execution Speed of Register Operation 34
12 Execution Speed of SRAM Writes 35
13 Execution Speed of FRAM Writes 36
14 Execution Speed of Thermometer 37
15 Execution Speed of Accelerometer 38
16 Execution Speed of UART communication 39
17 Energy Efficiency 40
18 Energy and Time Overhead 41
19 Power Traces 42
20 Required Energy of Each Benchmark. 42
21 Average Single Execution Period with NTS. 44
22 Average Execution Power of Benchmarks Under Different Power Traces. . . . 45
23 Four Out of Nine Checkpointings Can Be Avoided without Progress Setback. 50
24 Dual-Backup Checkpointing Handler. 52

25 Single Execution Period with TCW. 55
26 Average Execution Power of Benchmarks Under Different Power Traces. . . . 55
27 Energy efficiency with different clock frequency 58
28 Frequency matching based task schedule 61
29 Required energy of each benchmark 62
30 Energy Utility under Available Frequencies 63
31 Time and energy overhead 64
32 Changes of Charging Efficiency and Voltage on the Edge Device with Ultra-low Harvesting Power 68
33 Influence of Wake-up Voltage on Charging Cycle and Efficiency of the Edge Device with Ultra-low Harvesting Power 69
34 Influence of the Wake-up Voltage on Checkpointing Frequency of the Edge Device with Ultra-low Harvesting Power 70
35 Measurements of the Execution Progress with Different Working Voltages . . 71
36 Energy efficiency with different voltage combinations 72
37 Architecture of self-powered edge device 75
38 Capacitor Charging Circuit 76
39 Concavity of ζ in respect of v_{slp} and v_{wak} 86
40 Optimal ζ and v_{wak} with given v_{slp} 87
41 Execution speed (theoretical vs experimental) 87
42 Maximum Energy Efficiency and Optimal Wakeup Voltages with Different Sleep Voltages 94
43 Average execution power 95
44 ENZYME vs. Baselines regarding energy efficiency 96

1.0 Introduction

In this era of the Internet of Things (IoT), an increasing number of electronics "things" that embedded with software, sensors, and connectivity forms a seamless web to facilitate our daily life. Generally, the majority of these "things", such as wireless sensor nodes, are at the edge of IoT network, which bridges between the physical and cyber world by sensing the real-life information and converting it into the digital representations for IoT applications. It is estimated that IoT will consist of almost 50 billion devices by 2020, which brings a challenge of how to power these vast number of edge devices sustainably and efficiently.

1.1 Research Motivation

While battery power is not a favorable solution in the long run due to size, longevity, safety, and recharging concerns, energy harvesting, out of all possible energy sources, is one of the most promising techniques to meet both the size and power requirements of edge devices. A typical self-powered IoT edge device is usually composed of an energy harvester, a power regulator, a storage capacitor, and an embedded edge device. Energy harvesters can harvest different kinds of ambient energy, such as kinetic, electromagnetic radiation (light and RF), and thermal energy. The harvested energy will be converted into electric energy and flow through the power regulator to provide targeting voltage for charging the storage capacitor or directly powering the edge devices.

Although the future of self-powered IoT edge devices is promising, a critical drawback of most energy harvesters is that the harvesting power is often weak and intermittent, resulting in frequent power interruptions to edge devices. Frequent system turning on and off will jeopardize the data integrity causing significant progress setback if a program's intermediate execution state is not saved. With the help of non-volatile processors (NVP), both the program's execution state and active contents in its stack can be saved into non-volatile memory (NVM), before each power outage. This process is known as checkpointing. The

next time when the harvesting power recovers, the execution state can be restored to volatile RAM and program execution resumes. In this way, edge devices are resilient to power outages and the execution progress can be "intermittently" accumulated.

Nevertheless, even with the help of NVP, the potential of a self-powered IoT edge device is still severely hindered by the intrinsic low energy efficiency and reliability making it challenging to deliver reliable and efficient edge service for satisfying IoT service requests. These drawbacks of self-powered IoT edge devices mainly come from the following four aspects.

- First, although all tasks' runtime execution state can be saved successfully before power outage by NVP, not all tasks can be resumed correctly. Tasks, such as sensing and communication, are susceptible to the time delay. Their execution states, although can be checkpointed successfully, become useless when being resumed after a period of time. Therefore, extra energy and computation resources are required to compensate for their progress loss during the power outage.
- Second, as the checkpointing involves write operation on non-volatile memory, which is both time and energy-consuming, extra time and energy are required to support checkpointing. Considering the limited harvesting power, checkpointing can significantly affect the energy overhead especially when harvesting power becomes extremely small and unstable. What's more, as the endurance of non-volatile memory gradually wears out with each new write operation, checkpointing also threatens the lifetime of edge devices with NVP. Further, if checkpointing is unsuccessful, a significant progress setback would occur.
- Third, each tasks has its own optimal CPU clock frequency for which neither too small nor too large can bring optimal energy efficiency. Therefore, an inappropriate configuration of run-time CPU clock frequency can escalate the already low energy efficiency.
- Finally, under the ultra-low harvesting scenario, the energy harvester and power regulator bundle suffers from severe degradation in energy efficiency making energy efficiency much lower and checkpointing much frequent which further escalates the energy efficiency and QoS of IoT services.

These four challenges need to be addressed before the self-powered IoT edge device can provide a reliable, energy-efficient, and environment-friendly IoT end service.

1.2 Existing Work

In order to solve the aforementioned challenges, existing work mainly focuses on three optimization perspectives including improving power regulator efficiency, reducing checkpointing overhead, and improving runtime execution efficiency.

1.2.1 Optimizing Power Regulator

As the power supply of self-powered IoT edge devices, the energy harvester extracts energy from the ambient environment. Solar [38, 40, 48, 15], wind [38], footsteps [45, 47, 20], breathing [47], blood pressure [47], and body heat [21, 38, 47] are all possible energy harvesting sources. The power harvested from these energy sources has different magnitudes. For example, solar energy can generate a large magnitude of power at a power density of $15mW/cm^2$. The footstep is controllable human power, and the amount of harvested power can be as much as $67W$ [20] during a brisk walk. For ultra-low-power devices, the sources with low power densities, such as breathing ($0.42W$), and body heat ($2.4{\sim}4.8W$), can provide sufficient power to drive the devices at low duty cycles [17, 21, 48, 30, 52]. A common drawback of these energy sources is that they are often weak or intermittent, When an edge device is powered directly by an energy harvester, the supply voltage may be too low to drive the edge device. Even if the supply voltage is higher enough such as solar power, the intermittent nature would cause frequent power interruption resulting in data corruption and execution progress setback. Therefore, the power regulator is a key component, which harnesses the harvested energy for supplying targeting voltage to powering IoT edge device.

For a power regulator, upon supplying power to the IoT edge device, it also consumes power rendering lower energy efficiency for powering edge devices. The situation will be even worse when harvesting power is ultra-low and unstable. A considerable amount of research has been conducted for reducing the hardware overhead by improving the regulator efficiency through impedance matching techniques [8, 19, 9, 18, 46, 24]. These work mainly focus on power regulator optimization, for instance [8] proposes effective switching frequency technique for voltage converter to deliver maximum output power. [46] conducts circuit-level

design which enables the regulator to extract power from multiple low-power energy harvesting sources with maximum efficiency. [9] proposes a duty cycle based impedance adjustment technique for the maximum power extraction from a thermoelectric energy source without sacrificing power conversion efficiency. what's more [51, 49, 50] propose through-silicon-via inductors which can be used by energy harvesting circuits with minimum footprint. Further, [63] proposes a run-time simulation framework of both power delivery and architecture and captures their interactions for energy efficiency optimization.

1.2.2 Optimizing Checkpointing

After harvesting power has been delivered to the edge device, a portion will be used to support checkpointing which saves runtime state from volatile memory into the non-volatile region before a power outage happens. Checkpointing involves write operation on nonvolatile memory, which not only is time and energy consuming but also reduces the limited endurance of NVP resulting escalation of reliability. Therefore, many pieces of research work focus on reducing checkpointing overhead through both hardware and software techniques.

Hardware checkpointing stores the system state and data automatically by hardware. For example, Yu et al. [16] propose a non-volatile processor architecture that integrates non-volatile elements into volatile memory at bit granularity. Wang et al. [53] design a FRAM based processor, which attaches an NV-FRAM cell to each volatile standard flip-flop. The flip-flops are accessed for normal execution while the FRAM cells are used to checkpoint the states in flip-flops at power failure. This processor can backup and restore the processor state and data within $3\mu s$. Sakimura et al. propose the non-volatile magnetic flip-flops [43] and a 20MHz non-volatile micro-controller with STT-RAM [44]. Recently, Liu et al. propose an enhanced NVP based on ReRAM which has the highest integration level [27]. Also, Li et al. propose the non-volatile I/O enabling automatic reconfiguration of I/O interfaces [23].

Besides hardware checkpointing, there are also software mechanisms that checkpoint the processor's state and other volatile data into non-volatile memories. For example, Mementos [42] is a software mechanism for transiently powered RFID-scale devices. Some trigger points are placed after each call instruction or at each loop latch. At run-time, when these

trigger points are reached, the supply voltage is checked with an ADC. If this voltage is below a threshold, a snapshot of the system state is saved to the flash memory. Quickrecall [14] integrates FeRAM into the main memory to increase the checkpointing efficiency which reduces the backup data size and lowers the failure voltage threshold. Hibernus [6] and Hibernus++ [6] propose interrupt-based checkpointing mechanisms. In these mechanisms, the system stated is checkpointed only once immediately before the power failure, then the system hibernates. This mechanism requires frequent voltage checking and automatic interrupts to checkpoint or restore the system state. MEU [34] proposes prototyping techniques for a joint reduction of software and hardware overhead with software solutions.

1.2.3 Optimizing Run-time Execution

Even though the power regulator and checkpointing operation are optimized, inappropriate runtime clock frequencies can significantly reduce the runtime energy utility. Previous research on DVFS-based techniques [58, 12, 29] mainly focus on how to lower down the execution frequency appropriately so that the power consumption of the embedded system can be reduced. However, low CPU frequency doesn't necessarily guarantee high energy efficiency. In fact, when CPU frequency becomes lower, the reduction of execution power may not be significant. Yet the execution time could become significantly longer. As a result, energy efficiency may become lower. Besides, accessing hardware with a large latency such as non-volatile memory or GPIO can be the bottleneck of the actual execution speed. As such, it is necessary to optimize execution frequency accordingly for each program. Therefore runtime frequency should be adjusted accordingly.

All existing optimization techniques mentioned above that can be used to optimize the energy efficiency of self-powered IoT edge devices can be categorized into hardware and software solutions. Compared with hardware solutions, software solutions have advantages including low cost, full of flexibility, significant reduction of development cost, better adaptability, etc. Besides, what is even more important is that software solutions are able to provide equally competitive performance optimization compared with hardware solutions. Therefore, software solutions become increasingly popular to optimize the performance of

self-powered IoT edge devices. This dissertation is one of such kind that proposed several embedded software approaches to address the aforementioned challenges so that the potential of self-powered IoT edge devices can be fully exerted for better IoT edge services.

1.3 Research Contributions

This dissertation proposes embedded software solutions to address the aforementioned challenges. Compared with the aforementioned research work, this dissertation includes a project that focuses on maximizing execution progress of the atomic task which is subjected to progress loss during the power outage regardless of having NVP equipped. What's more, this dissertation also covers the projects that optimize checkpointing and runtime clock frequency accordingly for low-powered edge devices uniquely through software approaches. Further, this dissertation includes a project which is the first to propose a software paradigm dedicated to self-powered IoT edge devices to thrive in ultra-low harvesting power scenarios. In a nutshell, this dissertation focuses on four different angles to address the aforementioned critical issues challenging self-powered IoT edge devices.

- First, to prevent execution progress loss during the power outage, NVP-aware task schedulers are proposed to maximize the overall task execution progress especially for the atomic tasks which are better to finish execution before the power outage.
- Second, to minimize both the time and energy overhead of checkpointing operation, an intelligent checkpointing scheme is proposed which not only can ensure a successful checkpointing but also can predict the necessity of conducting checkpointing to avoid excessive checkpointing overhead.
- Third, to avoid inappropriate runtime CPU clock frequency, which consumes extra energy while delivering less execution progress, a CPU frequency Modulator is proposed which adjusts the runtime CPU clock frequency accordingly.
- Finally, to thrive in ultra-low harvesting power scenarios, a light-weight software-based paradigm is proposed to help maximize the energy extraction rate of energy harvester

and power regulator bundle. Besides, checkpointing under such an ultra-low scenario is also optimized for more energy-efficient and light-weight operation.

1.4 Dissertation Organization

The remainder of this dissertation is organized as follows. Chapter 2 provides the technical background. After that, Chapter 3 proposes an NVP-aware task scheduler to prioritize the atomic task before power outage so that the overall task execution progress can be maximized. To minimize the checkpointing overhead, Chapter 4 proposes an intelligent checkpointing scheme to avoid unnecessary checkpointing. To improve runtime energy efficiency, Chapter 5 proposes a CPU frequency modulator to adjusts the runtime CPU clock frequency accordingly for different tasks. To thrive in ultra-low harvesting power scenarios, Chapter 6 proposes a light-weight software paradigm to optimize energy extraction rate as well as a checkpointing scheme for more energy-efficient and light-weight operation. Finally, Chapter 7 concludes this dissertation.

2.0 Technical Background

This chapter first proposes the targeting system architecture and then introduces the fundamental supporting techniques of this dissertation including checkpointing, voltage monitoring, and sleep/wakeup managing.

2.1 System Architecture

In this section, a general architecture of the self-powered non-volatile IoT edge device is introduced which is shown in Figure 1.

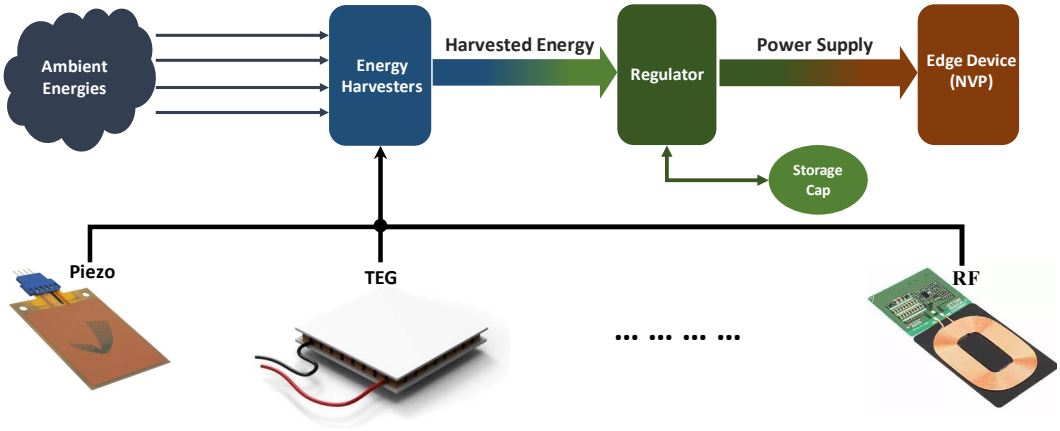


Figure 1: System Architecture of Self-Powered Non-Volatile IoT Edge Device

The targeting self-powered self-powered IoT edge device is equipped with a non-volatile processor and can harvest energy from multiple energy sources in the ambient environment and convert them into electrical energy. Typical energy sources include piezoelectric, thermal (TEG), radio frequency (RF), etc. As most kinds of harvested energy are weak and intermittent, in our target harvesting scenario, the energy harvester cannot power the edge device continuously. Once the program execution has used up the stored energy in the capacitor,

the non-volatile processor conducts checkpointing for maintaining execution progress. After that, the edge device goes to sleep and waits for the storage capacitor to be charged up again by the regulator which harnesses the harvesting power and provides the target supply voltage to the edge device. Once the capacitor has been charged up, the edge device can start or resume the execution until the stored energy is used up again.

2.2 Basic Checkpointing

In this section, the basic checkpointing mechanism will be introduced. The targeting edge device is equipped with NVP. Based on the structure difference of NVP, there are two kinds of checkpointing operations which are software-based checkpointing and hardware-based checkpointing. This dissertation only focuses on software-based checkpointing which is flexible and easy for implementation. Figure 2 shows the targeting NVP architecture for implementing software-based checkpointing.

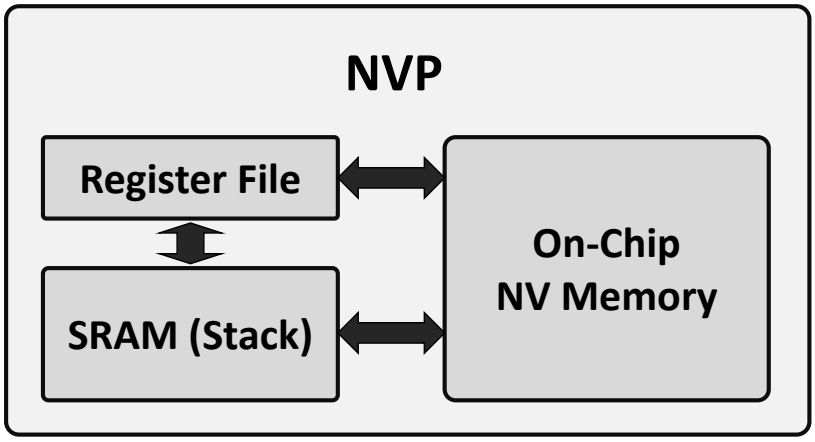


Figure 2: NVP Architecture

As we can see from Figure 2, for NVP aside from containing traditional on-chip volatile memory, including register file and SRAM, NVP also includes on-chip non-volatile memory. When a power outage occurs, all contents of the processor registers and SRAM should be

checkpointed to on-chip non-volatile memory. Hence, all the computation states can be saved. Once the power comes back on again, the computation can resume with the saved computation states.

Since the targeting NVP only support software-based checkpointing, there is no hardware to automatically save computation states. Hence, the software should take this responsibility. When detecting a low voltage below the preset threshold, the system will enter the checkpoint stage. First, all registers will be pushed onto the stack. Then the whole stack, which includes contents of all registers and all temporary variables, will be checkpointed from SRAM to non-volatile memory. One challenge here is how to save the program counter (PC) since we are not allowed to move PC explicitly. The trick we used is that, whenever a callee is called or interrupt service routine is invoked, the PC will be automatically pushed onto the stack. Then, we can safely save the PC together with other registers to the on-chip NVM

2.3 Voltage Monitoring

In this section, the basic voltage monitoring procedure is introduced. For voltage monitoring, there are mainly two categories which are hardware-based and software-based voltage monitoring. For hardware-based voltage monitoring techniques, there is dedicated hardware, known as voltage monitor, to keep monitoring the voltage of the target. If the voltage reaches the threshold, the voltage monitor will generate an I/O interrupt for preemption on NVP which then will know the power interruption. For the software-based voltage monitoring technique, the voltage detection is initiated by NVP which configures ADC accordingly for different scenarios and even coordinate with timer for voltage monitoring in a timely manner.

Software-based voltage monitoring techniques have significant advantages over hardware-based voltage monitoring mainly due to its flexibility and similar energy consumption. For a hardware-based voltage monitor, it only warns NVP systems about the "poor" quality of the harvesting power through I/O interrupts based on the hardwired voltage threshold. However, this threshold has a large impact on the overall performance of self-powered IoT edge devices. On the one hand, If the threshold is too high the NVP system wastes energy with frequent

checkpointing. On the other hand, if the threshold is too low, the checkpointing may fail. For that matter, after deployment, if the condition of the environment keeps changing, the hardwired threshold will degrade the performance of the edge devices. Therefore, for self-powered IoT edge devices, the software-based voltage monitoring technique is preferable where the NVP system needs to proactively initiate voltage detection. As the ADC detection time is trivial compared with the execution time, this software-based voltage monitoring overhead is negligible.

For the voltage monitoring target, many traditional techniques choose Vcc as the monitoring voltage source. However, Vcc is not suitable for monitoring and this dissertation uses the voltage on the storage capacitor for monitoring. The disadvantages of monitoring Vcc is that Vcc remains stable due to the power regulator unless the stored energy is almost drained out. In this way, once the voltage monitor detects the Vcc drop, NVP may not have enough time to respond for checkpointing, resulting in the progress setback. Therefore, Vcc is ruled out as the voltage detection source. For the voltage on the storage capacitor, it reflects the actual power supply of the energy harvester. Specifically, if the voltage on the storage capacitor drops, the energy harvester generates less power than the edge device consumed. Otherwise, the harvested power is sufficient to drive the edge device.

2.4 Sleep/Wake-up Managing

Once the checkpoint handler saves the run-time states into NVM, the edge device should be in sleep mode other than continue execution or shut down the entire system. There are mainly three reasons that make sleep mode a better option than the other two options.

1) After checkpointing, the remaining energy is insufficient to support another checkpointing. Therefore any further execution wastes energy as the state cannot be saved.
2) It is much faster for the NVP system to recover from sleep mode than cold reboot after being shut down.
3) Once the system is dead, it will automatically restart once the input voltage is beyond the cold start voltage. However, if the power supply is still insufficient, the system will

fail again at the very beginning. However, in sleep mode, ADC and timer are still allowed to monitor the voltage with negligible energy consumption. So the system can wake up when power is sufficient

.

Once the harvesting power comes back on again, the edge device will wake up given a preset wake-up voltage V_{wak}, which should at least support one checkpointing and a period of execution allowing new progress to be accumulated. Notice that, if the edge device is dead and once the power comes back on again, the edge device will initiate the cold start which requires much higher power than it was in low-power mode. In this case, if the harvesting power is insufficient, the edge device will be stuck at the very beginning as the harvested energy will always be drained out and cannot be preserved. Our solution is that at the very beginning of the startup stage, the NVP system needs to measure V_{cap}. If $V_{cap} < V_{wak}$, the NVP system needs to go back to sleep mode. This can avoid the system from stagnating at the cold starting stage.

3.0 Atomic Task Aware Task Scheduling

This chapter presents the research topic regarding task scheduler design for self-powered IoT edge devices with NVP. The organization of this chapter is as follows. First, research motivation and related work are provided. Then, the targeting system overview is given. After that, the proposed techniques including power prediction, independent task scheduler, and dependent task scheduler will be introduced in detail. Finally, the corresponding experiments and summary are provided.

3.1 Motivation

For self-powered IoT edge devices, before the power outage, programs' runtime execution state needs to be saved from volatile SRAM into on-chip NVM through checkpointing operation. However, not all tasks' execution states need to be pushed into stacks for checkpointing. Tasks, such as sensing and communication, are susceptible to the time delay. Their execution states, although can be checkpointed, become useless when being resumed after a period of time. These tasks should finish before the power outage once scheduled. Otherwise, their execution has to be started over when the power comes back on again. These tasks are defined as atomic tasks. Other tasks that can be checkpointed such as computation and data processing are defined as non-atomic tasks. To reduce the checkpointing overhead, the task scheduler should try its best to finish schedule as many unfinished atomic tasks as possible for completion before the power outage.

The following example will demonstrate how the unfinished atomic tasks affect system performance. The system that we target in this example is checkpointing enabled and equipped with both volatile and non-volatile memories. Assume that successful checkpointing can be ensured. For illustration purposes, task A is a non-atomic task and task B is an atomic task. Initially, tasks are executed concurrently with a round-robin scheduler. Given a harvesting power trace, the execution progress of both tasks are shown in Figure. 3.

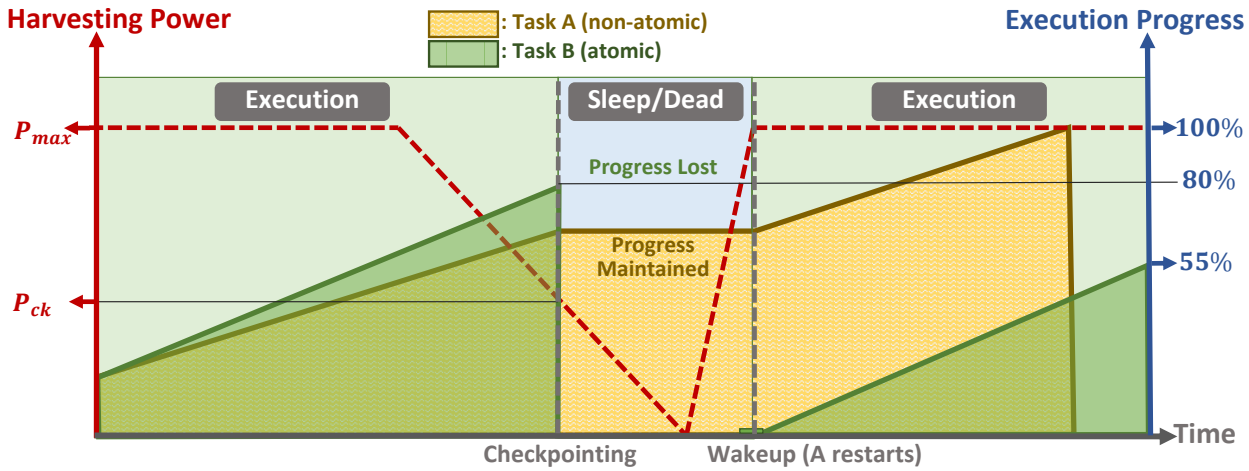


Figure 3: Atomic Task B is Unfinished Before The Power Outage Resulting in Progress Setback.

In Figure. 3, the red dash line represents the energy harvesting power trace, the yellow shadow represents the execution progress of task A, and the green bars represent the execution progress of task B. From the beginning, the execution progresses of both tasks are gradually accumulated until the harvesting power drops below P_{ck}. Since A is non-atomic, its execution status including all the register files and stack can be stored into non-volatile memory when checkpointing is triggered. Therefore, once the power comes back on again the system wakes up and the program execution of A resumes.

However, Since task B is atomic, it loses entire execution status during the power outage. Once the power comes back on again, B has to restart from the very beginning. As we can see, before the power outage, B is almost finished and has the execution progress of 80%, which is a significant setback once this progress is completely lost during the power outage. One simple and intuitive solution to avoid the progress setback is to prioritize B once before the power outage happens. In this way, as A will not suffer from progress setback, if B can finish before the power outage, the overall progress can be maintained. Given the same power trace, figure. 4 shows the execution progress of both A and B with prioritized scheduling.

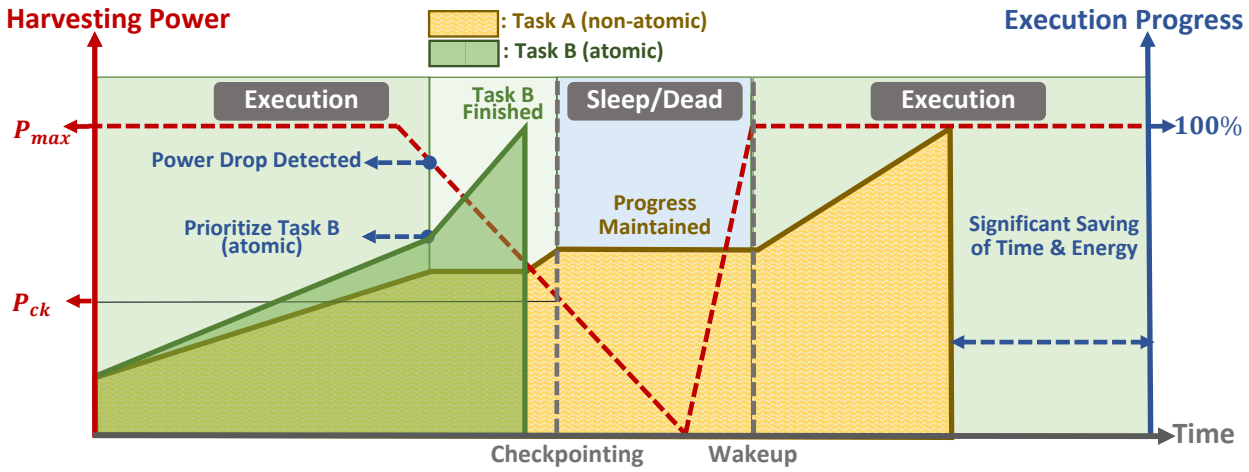


Figure 4: Prioritizing atomic Task B to Finish Before The Power Outage Results in Progress Maximization.

From Figure. 4, we can see that after detecting the power drop, B has been prioritized immediately and it finishes right before the system conducts checkpointing. In this way, the execution progress of both A and B can be maintained. What is even more appealing is that, compared with Figure. 3, a significant amount of time and energy are saved, which can be further used for other program executions. Overall, the execution progress can be significantly improved.

Although prioritizing B can maximize the overall execution progress, before conducting the priority-based scheduling, it is crucial to know whether B can finish before the power outage. This is because, if B cannot finish, the system will waste even more energy and have a severe progress setback during the power outage. To evaluate whether task B can finish before the power outage, the power analysis should be taken place for evaluation, which is proposed in section 3.4

3.2 Related Work

Energy harvesting extracts power from the ambient environment and is often used to deploy long lifetime battery-less devices. Solar, wind, footsteps, breathing, blood pressure, and body heat [21, 38, 47] are all promising energy harvesting sources. They have different characteristics of predictability, controllability, and magnitude. For example, solar energy is predictable and can generate a large magnitude of power at a power density of $15mW/cm^2$. The footstep is a controllable human power and the amount of harvested power can be as much as $67W$ [20] during a brisk walk. For ultra-low-power devices, the sources with low power densities, such as breathing ($0.42W$), and body heat ($2.4\sim4.8W$), can provide sufficient power to drive the devices at low duty cycles [48]. With proper configuration and management [17], it is feasible to operate a whole system with purely harvested energy.

In order to make systems power-failure proof, non-volatile memory based processors (NVP) [56] are developed. In these processors, non-volatile memory [35, 32] is attached to the processor, and the volatile execution state is checkpointed into the non-volatile memory upon the power outage. Researchers showed that checkpointing is a feasible method to save the runtime state [41, 42] with nonvolatile memory for energy-harvesting devices. Microcontrollers such as TI's MSP430 series [2] employ FRAM as an on-chip memory. Ransford et al. [42] present a software system, *mementos*, for transiently powered RFID-scale devices with energy-aware state checkpointing method. This system deploys Flash memory to back up the volatile content. Registers are pushed into the stack and then saved to the Flash memory. Since Flash memory has a limited write endurance and slow write speed, the time and energy overhead is large. Similarly, Wireless Identification and Sensing Platform (WISP) [4] was developed to achieve a similar goal.

Instead of checkpointing the execution state into on-chip or off-chip memory at a low speed, ferroelectric non-volatile register-based processors are proposed for energy-harvesting devices [53, 64]. This kind of processor attaches a nonvolatile memory cell to each volatile element and therefore allows fast local backup of intermediate results and fast recovery. FRAM-based processors [64, 31, 57, 11, 65, 13, 31, 53], present a great potential to be deployed in energy-harvesting devices. For example, Yu et al. [59] propose a non-volatile processor ar-

chitecture that integrates non-volatile elements into volatile memory at bit granularity. To reduce the backup overhead and energy, different technologies have been proposed including instruction scheduling [55], register reduction [62], and instruction selection [54].

Since harvested energy is limited for each power cycle, efficiently utilizing harvested energy is critical. Smart task scheduling techniques can save a significant amount of energy. [60] proposes a long-term deadline-aware scheduling algorithm to reduce energy consumption and deadline miss rates of tasks. [22] proposes a performance-aware task scheduling strategy for energy harvesting nonvolatile processors considering the power switching overhead. This paper will propose a lightweight scheduler that can maximize task progress and thus reduce energy consumption considering all types of tasks.

3.3 System Overview

In this section, the targeting system architecture of self-powered IoT edge devices regarding software and hardware layers is given in Figure 5.

As shown in Figure 5, the architecture of the targeted energy harvesting system includes both hardware and software components from bottom to top. The power supply consists of energy harvesters, a regulator, and a capacitor, which supplies energy for the whole system. The energy harvester converts harvested energy from ambient sources, such as solar energy, thermal energy, and radio frequency (RF) to electrical energy. The converted energy is stored in a small-size storage capacitor, which is used to supply energy for checkpointing upon power outages. The regulator is used to maintain a constant voltage level. The other hardware components consist of MCU, timer, ADC, sensors, and I/O ports. The software layer has full control of the onboard hardware to collect data and make decisions.

The software layer includes the proposed progress maximization scheduler assisted with three auxiliary functional modules: voltage monitor, checkpoint handler, and routine handler. These four functional modules interact with each other to maximize the task execution progress and can be easily integrated into any existing energy harvesting embedded systems. Further, all tasks to be scheduled can be divided into atomic tasks and non-atomic tasks.

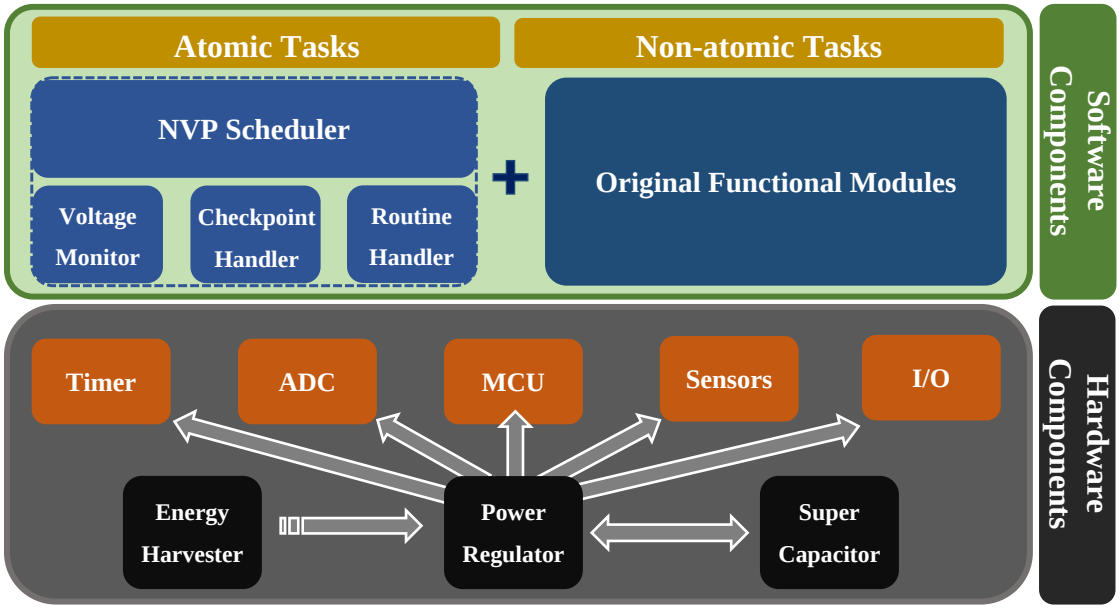


Figure 5: System Architecture

3.4 Power Prediction

In this section, a voltage monitor designed to conduct voltage measurement and power analysis will be presented. From Figure 5, we know that detecting the voltage of the storage capacitor requires collaboration between hardware and software. On the hardware side, the output of the storage capacitor needs to connect to an analog to digital converter (ADC) channel of the MCU. On the software side, the ADC needs to periodically sample the storage capacitor's voltage. Here we define V_{cap} as the voltage on the storage capacitor.

Knowing the voltage of the storage capacitor V_{cap} is not enough to determine whether the harvesting power can support the NVP system and there are mainly two reasons:

1) First, for the storage capacitor, even if V_{cap} is low, the harvesting power supply can still be sufficient to support the NVP system if it is recovering;

2) Second, the power consumption of tasks varies. Therefore, even if the harvesting power can support the current task, it cannot guarantee to support other tasks which consume more power and energy.

These two concerns can be eased by calculating the discharging power of the capacitor. This is because the working status of the system is directly determined by the amount of energy on the capacitor. Also, given the same working power, the changes of the discharging power equal the changes in the harvesting power. So given current ongoing execution of $Tsk[cur]$, the calculation of discharging power on the capacitor can estimate how long (T^S) the system can be in an execution before the outage happens.

Therefore, with current task $Tsk[cur]$, task information table Tab, and ADC sample period T_{ADC}, DPA algorithm is able to calculate the discharging power of $Tsk[cur]$. With the discharging power, DPA can further estimate the duration of T^S for the system to work before the outage happens. The estimation also comes with the trustiness $Trust^P$ evaluation.

DPA is called each time by the NVP scheduler to evaluate the discharging power on the capacitor before actually conducting task scheduling. At the beginning, DPA initiates parameters including $< \Delta E_{cap} >_N$: an array to store N times of energy reductions on the capacitor, $< \Delta E^{est}_{cap} >_N$: an array to store N times of estimated energy reduction on the capacitor, $< \Delta P_{cap} >_N$: an array to store N times of discharging power reduction on the capacitor, $< \Psi >_N$: N times of the correctness of the estimation, and P'_{cap} : the estimated reduction of the discharging power. The initialization process is shown in line 1 to 5.

After initialization, DPA calls ADC to sample $N + 1$ voltage samples on the capacitor. After each sampling, DPA first calculates the reduction of the energy ΔE_{cap} on the storage capacitor which is shown in line 8. Then, in line 9, DPA further calculates the reduction of the discharging power ΔP_{cap}. To signal the status of the discharging power, DPA introduces two coefficients υ and ν ($\upsilon > \nu$ and $\upsilon + \nu = 1$), If $\Delta P_{cap} > 0$, the discharging power is reduced and DPA assigns ν to the weight factor K, otherwise,DPA assigns υ to K. The rationale behind the weight assignment is to let the system be more aware of the most recent increase of the discharging power, which indicates that the harvesting power is reducing. The comparing process is shown in line 10 - 14.

After calculating ΔP_{cap} from the real samples of V_{cap}, in line 15 to 21, DPA conducts the evaluation procedure. At first, in line 15, DPA estimates the reduction of the discharging power as P'_{cap}. The estimation uses the weight factor of K. If the real discharging power is reduced (harvesting power recovers), DPA gives more weight to estimation, otherwise, DPA

Algorithm 3.4.1 Discharging Power Analysis (DPA)

Input: Tab, and T_{ADC}.
Output: T^S and $Trust^P$.

1: $< \Delta E_{cap} >_N \leftarrow$ actual reduction of the capacitor energy;
2: $< \Delta E^{est}_{cap} >_N \leftarrow$ estimated reduction of the capacitor energy
3: $< \Delta P_{cap} >_N \leftarrow$ actual reduction of the discharging power;
4: $< \Psi >_N \leftarrow$ correctness of the estimations;
5: $P'_{cap} \leftarrow$ the estimated reduction of the discharging power;
6: **for** each ADC cycle i from 1 to $N+1$ **do**
7: **if** $i > 1$ **then**
8: $\Delta E_{cap}(i-1) = C[V^2_{cap}(i-1) - V^2_{cap}(i)]/2$;
9: $\Delta P_{cap}(i-1) = \Delta E_{cap}(i-1) - \Delta E_{cap}(i)$;
10: **if** $\Delta P_{cap}(i) > 0$ **then**
11: $K = \nu$; /* discharging power reduced */
12: **else**
13: $K = \upsilon$; /* discharging power increased */
14: **end if**
15: $P'_{cap} = (1-K)P'_{cap} + K\Delta P_{cap}(i)$;
16: $\Delta E^{est}_{cap}(i-1) = \Delta E_{cap}(i-2) + P'_{cap}$;
17: **if** $\Delta E^{est}_{cap}(i-1) < \Delta E_{cap}(i) - 1$ **then**
18: $\Psi(i) = 0$; /* underestimate */
19: **else**
20: $\Psi(i) = 1$; /* overestimate */
21: **end if**
22: **end if**
23: **end for**
24: $E^{rem}_{cap} = C[V_{cap}(N+1)^2 - V^2_{ck}]/2$
25: $\overline{\Delta E^{est}_{cap}} = \sum_{i=1}^{N} \Delta E^{est}_{cap}(i)/N$
26: $T^S = T_{ADC} * E^{rem}_{cap} / \overline{\Delta E^{est}_{cap}}$
27: $Trust^P = \sum_{i=2}^{N} \Psi(i)/(N-1)$;
28: **return** T^S and $Trust^P$; =0

gives more weight to the real discharging power. After the calculation of P'_{cap}, DPA further estimates the energy reduction ΔE^{est}_{cap} on the capacitor, which is shown in line 16. After estimation, in line 17 to 21, DPA compares the estimation ΔE^{est}_{cap} with the real energy reduction ΔE_{cap}. If DPA underestimates the reduction of the discharging power, the estimation is invalid and DPA assigns $\Psi = 0$ to indicate the invalidation. If DPA overestimates the reduction, it accepts the estimation and assigns $\Psi = 1$ to indicate the acceptance. In this way, the estimation can be more aware of the loss of the harvesting power.

After evaluation, DPA calculates the remaining energy E^{rem}_{cap} in line 24 and the average estimated energy reduction $\overline{\Delta E^{est}_{cap}}$ in line 25. Then, based on E^{rem}_{cap} and $\overline{\Delta E^{est}_{cap}}$, in line 26, DPA estimates the duration T^S for system to work before the power outage happens. Accompanied by the estimation of T^S, DPA also calculates the trustiness $Trust^P$ of the estimation in line 27. After the calculation, DPA returns T^S and $Trust^P$ for NVP scheduler to use as references to conduct task scheduling for progress maximization.

3.5 Independent NVP Task Scheduler

In this section, we will present a lightweight scheduler, which is specifically designed for NVPs. The goal is to maximize system progress under unstable power supply.

Figure 6 shows the overview. The core component is the NVP scheduler and it is assisted with three functional modules: voltage monitor, checkpoint handler, and routine handler. The NVP scheduler can run on its own or be incorporated into an existing embedded OS. If working with the existing OS, the modules in existing OS remain intact and will be invoked when $V < V_{warn}$, where V_{warn} is the preset voltage threshold. The voltage monitor is active all the time and responsible for voltage detection and analysis of power consumption of the NVP system. Once voltage V drops below V_{warn}, the NVP scheduler is triggered to maximize task progress. If V continues to drop until it is below threshold V_{ck}, the checkpoint handler is triggered to ensure a successful checkpointing. After checkpointing, the routine handler will put the system into sleep mode until the power recovers. By then, the routine handler will wake up the entire system. These functional modules interact with each other.

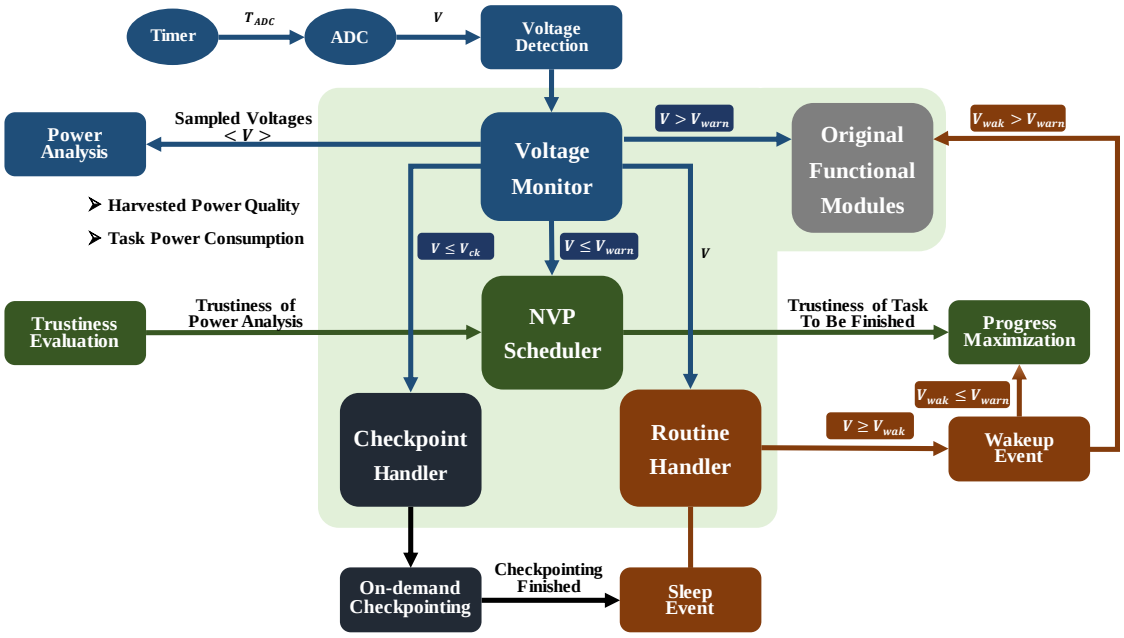


Figure 6: NVP Scheduler Overview

After the voltage monitor detects that the voltage on the storage capacitor drops below V_{warn}, in anxious of a potential power outage, the NVP Scheduler is triggered to maximize progress with the remaining energy supply. The key idea of the scheduler is to differentiate different tasks and all tasks can be categorized into two groups:

- Atomic tasks: tasks that even can be checkpointed before power outage still have to start over when the power comes back on again. Typical examples include communication, sensing, etc.
- Non-atomic tasks: tasks that can be checkpointed and resumed correctly during a power outage without any problem. Typical examples include computation, data fusion, etc.

Several important pieces of information regarding each task also need to be maintained. Of the most important are the task execution power, current progress, progress to time ratio, and atomicity, as shown in Table 1.

For each task, the execution power is measured at offline stage. The execution progress and the progress to time ratio are maintained by the OS and are updated when the task is switched out by the scheduler. When voltage on the storage capacitor drops below V_{warn},

Table 1: Task Information (Tab)

Tasks	Execution Power	Current Progress	Progress to Time Ratio	Atomicity
$Tsk[i]$	P_i	Pg_i	Pg'_i	Non-atomic
$Tsk[j]$	P_j	Pg_j	Pg'_j	Atomic

in anxious of a potential power outage, NVP scheduler is triggered to prioritize the atomic tasks. Given the task ready queue Q and the task information table Tab as the inputs, Algorithm 3.5.1 shows details of NVP scheduler.

Algorithm 3.5.1 NVP Scheduler

Input: Q and Tab
Output: schedule tasks for maximum task progress

1: **while** $(V_{ck} < V_{cap} < V_{warn})$ **do**
2: $T^S, Trust^P \Leftarrow DPA(Tab, T_{ADC})$; /*Algorithm 3.4.1*/
3: $Trust^{max} = 0$;
4: **for** each atomic $Tsk[i]$ in ready queue Q **do**
5: $Trust_i \Leftarrow TCE(T^S, Trust^P, Tsk[Cur], Tsk[i], Tab)$; /* Algorithm 3.5.2 */
6: **if** $Trust_i > Trust^{max}$; **then**
7: $Trust^{max} = Trust_i$;
8: $Sel = Tsk[i]$;
9: **end if**
10: **end for**
11: **if** $Trust^{max} > Trust^{TH}$ **then**
12: execute Sel
13: **else**
14: break;
15: **end if**
16: **end while**
17: **while** $(V_{ck} < V_{cap} < V_{warn})$ **do**
18: round-robin scheduling for non-atomic tasks$\in Q$;
19: **end while** =0

The purpose of NVP Scheduler is to achieve overall task progress maximization by giving more scheduling priority to atomic tasks in the ready queue Q when a potential power outage may happen. The inputs include ready task queue Q and task information table Tab. NVP Scheduler proceeds with a voltage range of $[V_{ck}, V_{warn}]$. Once $V_{cap} < V_{warn}$, in anxious of

a possible power outage, NVP Scheduler starts scheduling. When $V_{cap} \leq V_{ck}$, the NVP Scheduler stops and initiates checkpointing procedure.

At first, the NVP scheduler needs to prioritize the scheduling for atomic tasks, which is shown in line 1 to 16. Before conducting scheduling, preparation needs to be made. In line 2, NVP scheduler calls the Discharging Power Analysis (DPA) algorithm to assess the harvesting power and output the estimated execution time T^S that the system can support to the current ongoing task $Tsk[cur]$, along with which DPA also provides the trustiness value $Trust^P$ of the estimation. DPA is detailed in Algorithm 3.4.1. After calling DPA, in line 3, NVP scheduler initiates the parameter $Trust^{max}$ which will be further used to select the best atomic task candidate for scheduling.

After preparation stage, in line 4 to 10, NVP scheduler evaluates the possibility of each task $Tsk[i]$ to finish before the power outage. Given the parameter of T^S and $Trust^P$ from DPA, the current task $Tsk[cur]$, targeting task $Tsk[i]$, and the task information table Tab, the Task Completion Evaluation (TCE) algorithm is called by NVP scheduler to evaluate the possibility that $Tsk[i]$ is able to finish before the power outage. This possibility for $Tsk[i]$ is defined as $Trust_i$. Details of TCE is given in Algorithm 3.5.2. During the calculation of $Trust_i$ for each atomic task, NVP scheduler is able to find the best $Tsk[i]$ with the maximum $Trust_i$, which will be selected by NVP scheduler for further evaluation. The best $Tsk[i]$ and its $Trust_i$ are assigned to Sel and $Trust^{max}$ respectively.

With Sel and $Trust^{max}$, in line 11 to 15, NVP scheduler compare $Trust^{max}$ with $Trust^{TH}$ which is a threshold to decide whether it is worthwhile to execute Sel. If $Trust^{max} > Trust^{TH}$, NVP scheduler believes that Sel can finish before the power outage and proceeds the scheduling for Sel. Otherwise, the NVP scheduler will break out the while loop to schedule for atomic tasks. This is because, if the best atomic candidate Sel is not worthwhile for scheduling, no other atomic tasks will be.

Finally, once there is no suitable atomic candidate for scheduling, NVP will conduct round-robin scheduling for non-atomic tasks in the task ready queue Q until V_{cap} drops below V_{ck}, when the checkpointing starts. This is shown in line 17 to 19

During the scheduling, NVP scheduler needs to call its sub-algorithm TCE to conduct task completion evaluation for each atomic task. The evaluation is to find out the trustiness

$Trust_i$ of $Tsk[i]$ to be able to finish before the power outage. Given a targeting task $Tsk[i]$, the current task $Tsk[cur]$, task information table Tab, and the calculated values of T^S and $Trust^P$ from DPA, Algorithm 3.5.2 shows the details of the the evaluation.

Algorithm 3.5.2 Task Completion Evaluation (TCE)

Input: $Tsk[i]$, $Tsk[cur]$, Tab, T^S, and $Trust^P$
Output: $Trust_i$: the trustiness of $Tsk[i]$ to be able to complete;
1: $P_{cur}, P_i \leftarrow$ powers of $Tsk[cur]$ and $Tsk[i]$ from Tab;
2: $Pg_i \leftarrow$ progress of $Tsk[i]$ from Tab;
3: $Pg'_i \leftarrow$progress to time ration of $Tsk[i]$ from Tab;
4: $t_i = (1 - Pg_i)/Pg'_i$;
5: $T_i^S = Trust^P * T^S * P_{cur}/P_i$;
6: $Trust_i = T_i^S/t_i$;
7: **if** $Trust_i > 1$ **then**
8: $\quad Trust_i = 1$;
9: **end if**
10: **return** $Trust_i$ =0

At the beginning, TCE acquires several pieces of necessary information from Tab. For current task $Tsk[cur]$, TCE needs the execution power P_{cur}. For the targeting task $Tsk[i]$, TCE needs to access the execution power P_i, the execution progress Pg_i, and the progress to time ratio Pg'_i. The entire information fetching process is given in line 1 - 3.

With these pieces of information, TCE calculates how long that $Tsk[i]$ still needs before completion and how long the system can offer $Tsk[i]$ for execution. Here t_i is defined as the time required for $Tsk[i]$ to finish which is calculated in line 4 using parameters Pg_i and Pg'_i. T_i^S is defined as the time that the system is able to provide for $Tsk[i]$ before the power outage. T_i^s is calculated in line 5. Also because T_S is an estimated value, we need to consider about its trustiness $Trust^P$.

Finally, Given t_i and T_i^S, TCE is able to calculate $Trust_i$ in line 6. Then, TCE trims the value of $Trust_i$ to be between 0 and 1 and delivers $Trust_i$ to the NVP scheduler to make a final decision.

For auxiliary modules, the NVP scheduler also has a checkpointing handler and a routine handler. For the checkpointing handler, instead of checkpointing run-time data periodically, it is only triggered once the voltage on the storage capacitor V_{cap} drops below V_{ck}. After

checkpointing, the routine handler will take over the system for sleep. In the sleep mode, ADC and timer are still allowed to monitor the voltage with negligible energy consumption. If $V_{cap} < V_{wak}$, the edge device needs to go back to sleep mode. Otherwise, the edge device can wake up. This can save energy by avoiding the system from stagnating at the cold starting stage.

3.6 Dependent NVP Task Scheduler

In this section, an NVP-aware task scheduler (NTS) considering data dependence is proposed to maximize the execution progress and reduce the energy consumption on checkpointing. The main goals of NTS are bifold. First, to maximize the chance for tasks to meet their deadlines, the earliest deadline first (EDF) based list scheduling is used to schedule NVP tasks. Then, NTS differentiates tasks' atomicity based on their IO characteristics and schedules a task whose execution state can be stored and recovered successfully before the power outage. In this way, the harvested energy can be fully used for execution and the progress of the unfinished task can be maintained successfully without wasting energy to start over the execution.

Besides, NTS also considers the fact that tasks on the self-powered IoT edge device are usually iterative. For instance, the temperature sensing task on a self-powered IoT edge sensor node will keep collecting data periodically after the node has been deployed. This is true as the use of an energy harvesting system as the power source is mainly for devices to have long-term autonomous operations. Therefore, the type of tasks on these devices and their goals tends to be unchanged.

Aside from considering atomicity and task iteration, NTS also considers the data dependency, all father tasks should also be scheduled before their children. All tasks and the dependencies among these tasks are represented with Directed Acyclic Graph (DAG) where a parent task always requires data from its child tasks. Each task also has its own time constraint and energy consumption. Having all these concerns addressed, with the task set $\{tsk\}$ which contains the data dependency and energy requirements of each task. The task

ready queue Q, wake-up voltage v_{wak}, and current delay set $\{d\}$ as inputs, algorithm 3.6.1 provides details for task grouping and scheduling (NTS).

Given a DAG, scheduled task list sch, wake-up voltage v_{wak}, time constraint set $\langle t \rangle$, and energy consumption set $\langle e \rangle$ of each task's energy requirement as inputs, Algorithm 3.6.1 provides the details of the NVP-aware Task Scheduling (NTS) algorithm. NTS is executed when the edge device is temporarily awake to monitor the capacitor voltage. The purpose of the NTS algorithm is to schedule tasks for the next execution cycle such that the size of checkpointing data can be as small as possible. Initially, NTS searches for all subtrees in DAG and removes the scheduled tasks in sch from subtrees which are then stored into task tree set $\langle tree \rangle$ as shown in line 1. Then, in line 2, NTS calculates the amount of energy E that can be stored in a capacitor-based on the wake-up voltage v_{wak}. E will first be used to finish the previous checkpointed task tsk_p, if any. This is shown in line 4. Next, NTS initiates parameter Ψ which is used to indicate whether the scheduling is finished. After that, the deadline of each tree is calculated by adding its time constraint and the current time, which is shown in line 7 to 9. In line 10, NTS sorts all trees in ascending order of their deadlines. With the sorted tree sequence, in line 11, NTS starts picking up tasks for scheduling from the tree with the most imminent deadline.

For tasks on the selected tree, NTS keeps scheduling tasks as long as the stored energy E is beyond the required energy of all scheduled tasks. Because of data dependencies, leaf tasks need to be scheduled first. When a leaf task tsk_i^j on $tree_i$ has been chosen, NTS adds tsk_i^j into both tsk_{nx} and sch. After that, NTS removes tsk_i^j from the current task tree and subtracts the required energy e_i^j from E so that the following scheduling has updated information. These steps are shown in line 13 to 16. Once the energy in the capacitor is not enough to support a new task, in line 18 to 27, NTS adds an existing non-atomic leaf task, if any, to the end of the scheduling list and marks $\Psi = 1$ to indicate the completion of scheduling process.

Before breaking out the scheduling process, NTS checks whether all tasks in $tree_i$ have been scheduled. If so, NTS will release them from sch. In this way, these tasks can continue their next iteration in the coming execution cycle. In the end, NTS returns the schedule list tsk_{nx}. These steps are shown in line 30 to 37.

Algorithm 3.6.1 NVP-aware Task Scheduling (NTS)

Input: DAG, sch, v_{wak}, $\langle t \rangle$, and $\langle e \rangle$
Output: The task schedule list for the next execution cycle, tsk_{nx}

Build up unscheduled task tree set $\langle tree \rangle$ based on DAG and sch;
Calculate energy storage E based on v_{wak};
if previous task tsk_p has been checkpointed **then**
  $E = E - e_p$;
end if
$\Psi = 0$ /* indicator of completing scheduling */
for each $tree_i$ **do**
  $d_i = t^{cur} + t_i$; /* deadline of $tree_i$ */
end for
Sort trees in ascending order of d_i;
for each $tree_i$ **do**
  for each leaf task tsk_i^j in $tree_i$ **do**
    if $E > e_i^j$ **then**
      Add tsk_i^j to tsk_{nx} and sch;
      Remove tsk_i^j from $tree_i$;
      Update $E = E - e_i^j$;
    else
      for each leaf task tsk_i^k in $tree_i$ **do**
        if tsk_i^k is a non-atomic task **then**
          Add tsk_i^k to tsk_{nx} and sch;
          Remove tsk_i^k from $tree_i$;
          Update $E = E - e_i^k$;
          Break;
        end if
      end for
      $\Psi = 1$;
      Break;
    end if
  end for
  if $tree_i$ is empty **then**
    Release all tasks of $tree_i$ from sch;
  end if
  if $\Psi = 1$ **then**
    Break;
  end if
end for
return tsk_{nx}; =0

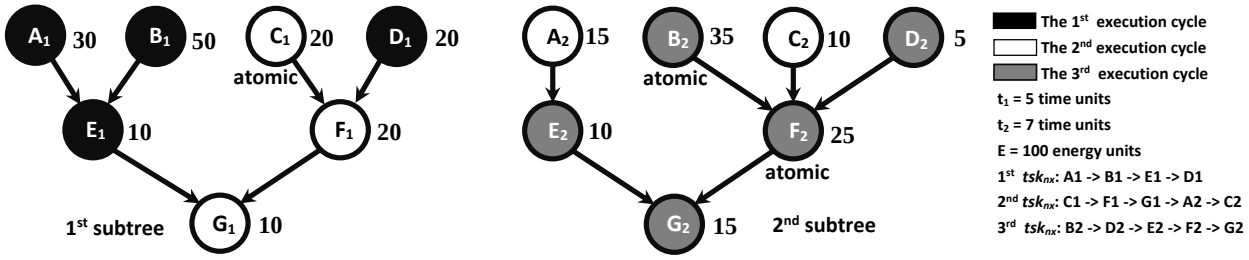


Figure 7: Example DAG

The example DAG in Fig. 7 will be used for better understanding of how NTS works. As we can see, this DAG has two subtrees. The first tree is on the left with the time constraint of 5 time units. The second tree is on the right with the time constraint of 7 time units. Both trees have seven nodes representing tasks. The atomic tasks are labeled at the bottom of each node and the unlabeled tasks are non-atomic. Each task's energy consumption is given on the right of each node. The energy in the capacitor is 100 energy units.

For the first execution cycle, NTS selects the 1^{st} tree for scheduling because of its earlier deadline. Then, NTS starts to schedule leaf task A_1 and B_1, which leaves E_1, C_1, and D_1 as the leaf tasks. After that, NTS selects E_1 for scheduling. The energy left in the capacitor after the execution of E_1 becomes 100-90=10. When NTS selects C_1 for scheduling, it finds out that energy consumption of C_1 is more than the energy available in the capacitor. At this point, NTS starts searching for unscheduled non-atomic leaf tasks, if any, in the 1^{st} tree and selects D_1 for scheduling. The scheduled tasks in the first execution cycle are in black.

For the second execution cycle, D_1 needs to be resumed at the beginning. NTS assumes that D_1 has to be started over which will consume all its required energy. Therefore, the energy left for the rest of the tasks is 100-20=80. Then, NTS schedules C_1, F_1, and G_1, which completes the scheduling of the 1^{st} tree and leaves 80-50=30 energy units for scheduling tasks on the 2^{nd} tree. For this subtree, NTS first schedules A_2, which leaves 15 energy units in the capacitor. When NTS selects B_2 for scheduling, it finds out that the energy consumption of

B_2 is more than the energy available in the capacitor. At this point, NTS starts searching for unscheduled non-atomic leaf tasks in the 2^{nd} tree and selects C_2 for scheduling. The scheduled tasks in the second execution cycle are in white.

For the third execution cycle, C_2 needs to be resumed at the beginning, which leaves 100-10=90 energy units for the rest of the unscheduled tasks. Since the total amount of required energy of the remaining unscheduled tasks on the 2^{nd} tree is also 90 energy units, all tasks on the 2^{nd} tree can be scheduled. The scheduled tasks in the third execution cycle are in grey.

NTS has the time complexity of $O(n)$ where n is the number of tasks. Due to the limited number of runtime programs on an edge device, the energy consumption of NTS is trivial compared with that of task execution. Experiments will show the performance and overhead of the NTS algorithm.

3.7 Experiments

In this section, the experiments are conducted to evaluate the performance of both the NVP scheduler and the NTS scheduler.

3.7.1 Evaluation of NVP Scheduler

3.7.1.1 Hardware Platform The experiment platform of the NVP embedded system is TI's MSP430FR5739 ultra-low-power evaluation board, which consists of a 16-bit MCU, a 10-bit ADC module, a 1kB volatile SRAM, a 16KB nonvolatile FRAM memory, and different peripherals for sensing and wireless sensing applications. The Bq25570 ultra-low-power regulator is used to supply a stable voltage, which can work with a minimum of the 120mV input voltage and a maximum of 4.2V boost voltage. The only hardware overhead of our design is that there should be an extra wire to connect between the storage capacitor and the ADC channel so that the voltage on the capacitor can be constantly monitored.

3.7.1.2 Power Traces To evaluate the performance of the NVP scheduler, five different power traces are generated as shown in Figure 8.

	Trace 1	Trace 2	Trace 3	Trace 4
High	9.1 s	3 s	1 s	always
Low	0.9 s	2 s	4 s	0 s

Figure 8: Power Trace Used for Experiments

The source power is provided by the MSP430FR5969 evaluation board which is programmed to generate different power traces through a GPIO pin which can provide a $3.6V$ output once the pin is set to be high, otherwise the output is 0. Then, power will be harnessed by the power regulator Bq25570 to power the MSP430FR5739. Power source 1 is for the output pin to set low for 0.9 seconds in every 10 seconds period; Power source 2 is for the output pin to set low for 2 seconds in every 5 seconds period; Power source 3 is for the output pin to set low for 4 seconds in every 5 seconds period; Power source 4 is for the output pin to set low for 1 second in every 1.3 seconds period; Power source 5 is for the output pin to set high all the time.

3.7.1.3 Software Setup Experimental software includes a lightweight NVP scheduler and the proposed three functional modules of Voltage Monitor, Checkpoint Handler, and Routine Handler. The original scheduler of the system is a round-robin scheduler. For the NVP scheduler, the Voltage Monitor keeps monitoring the voltage of the capacitor. Specifically, for every T_{ADC} period, the timer will wake up the ADC module for a short period during which V_{cap} will be sampled and analyzed, and the progress to time ratio of the executing task will be updated. If $V_{cap} > V_{warn}$, the original round-robin scheduler is in charge. Otherwise, the NVP scheduler will take over task scheduling for progress maximization. Before presenting experimental results, parameters are listed in Table 2.

Table 2: Task Information

C	T_{ADC}	V_{warn}	V_{ck}	V_{safe}	ν	υ	φ	ψ
$470\mu F$	$37.5ms$	$2.5V$	$1.2V$	$0.5V$	0.2	0.8	0.9	0.1

The benchmarks include three atomic tasks of acceleration measurement (the results are written in SRAM), temperature sensing (the results are written in SRAM), and UART communication (the data are written in FRAM). There are also three non-atomic tasks including register operations, SRAM writes, and FRAM writes. These six benchmarks are iterative and independent of each other.

With power traces, the experiments start with the power measurements of these six benchmarks. The baselines of this experiment are NoCP and RRCP. NoCP represents a round-robin scheduler without a checkpoint handler. The RRCP represents the round-robin scheduler with checkpointing ability. The proposed NVP scheduler which is incorporated into the baseline RRCP is defined as NVCP. For NVCP, the voltage monitor is always active to sample the voltage V on the capacitor. Once $V < V_{warn}$, NVP scheduler is activated and takes over the task scheduling, otherwise, it remains inactive.

3.7.1.4 Energy Consumption Analysis The power consumption of the six benchmarks is measured with a stable power supply. Figure 9 shows the measurements. All benchmarks are iterative and the duration of each iteration is trivial. For instance, the benchmark 6 has the largest iteration period of $24ms$ and benchmark 1 has the lowest period of $0.18ms$. During these periods, the changes in voltage on the capacitor are negligible. So it is reasonable to use $\mu J/Iteration$ representing the power consumption of each task.

3.7.1.5 Sleep Mode vs. Working Mode The first experiment is to show the power consumption of NVP system in sleep mode. In Figure 10, we adjust the Vcc from 3.7 to 1.4 gradually and measure the input current.

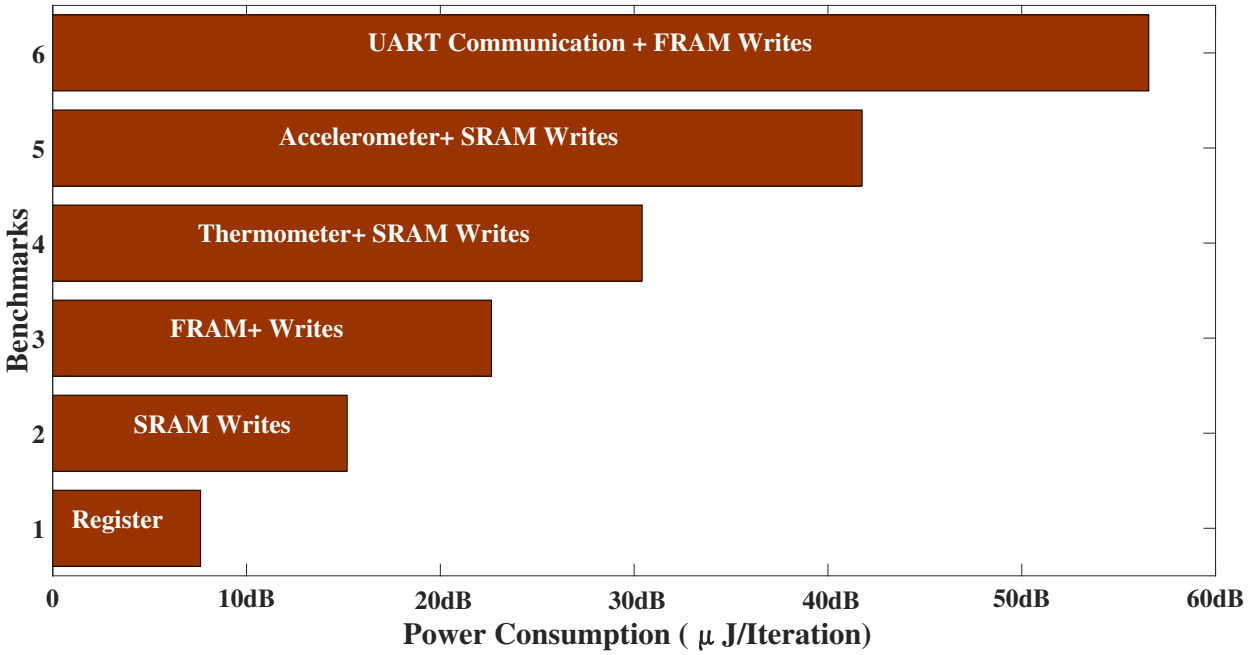


Figure 9: Power Consumption of Benchmarks

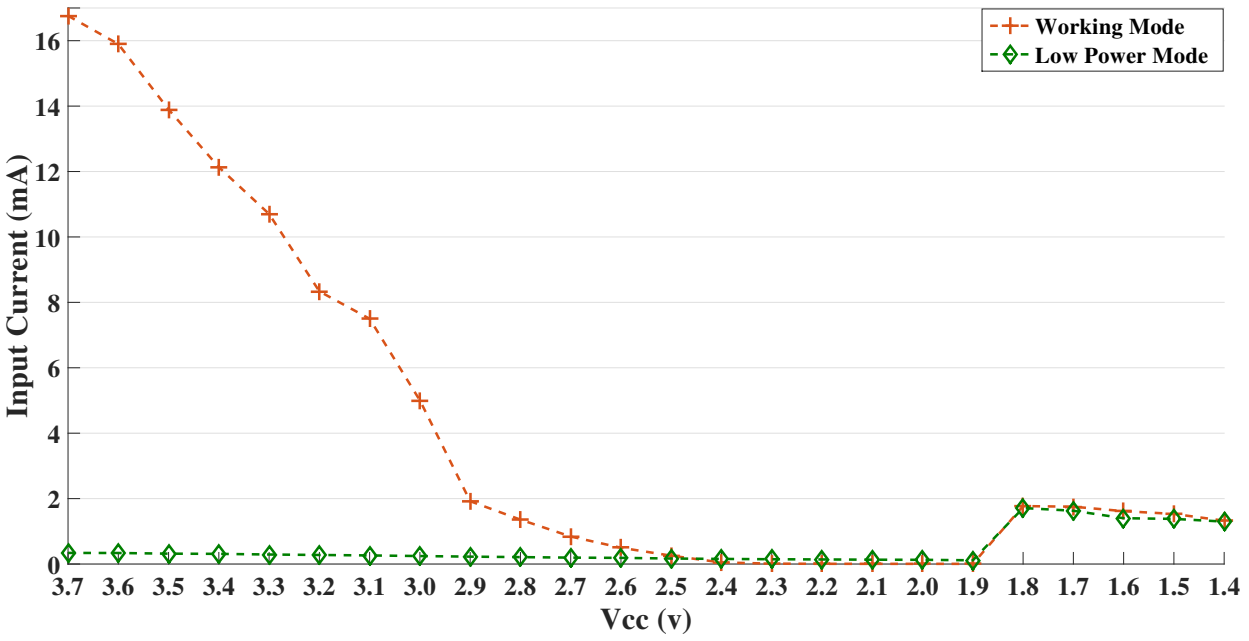


Figure 10: Power Consumption of Low Power Mode vs Working Mode

As we can see from Figure 10, in sleep mode, the maximum current is $0.3392mA$ which is significantly smaller than $16.724mA$ in working mode. Once the voltage drops, the current in sleep mode remains almost the same. Once the voltage drops below 1.8V, the system is dead and MCU loses control of all the peripherals and previously terminated GPIOs can allow current to go through, which increases the current. We can see that the NVP system in sleep mode consumes much less power than in working mode, therefore, it is better to go to sleep after conducting checkpointing.

3.7.1.6 Progress Comparison Figure 11-16 show the average execution speed of each task in term of iterations per second given different power traces. The larger the speed, the more energy is used for execution, and vice versa.

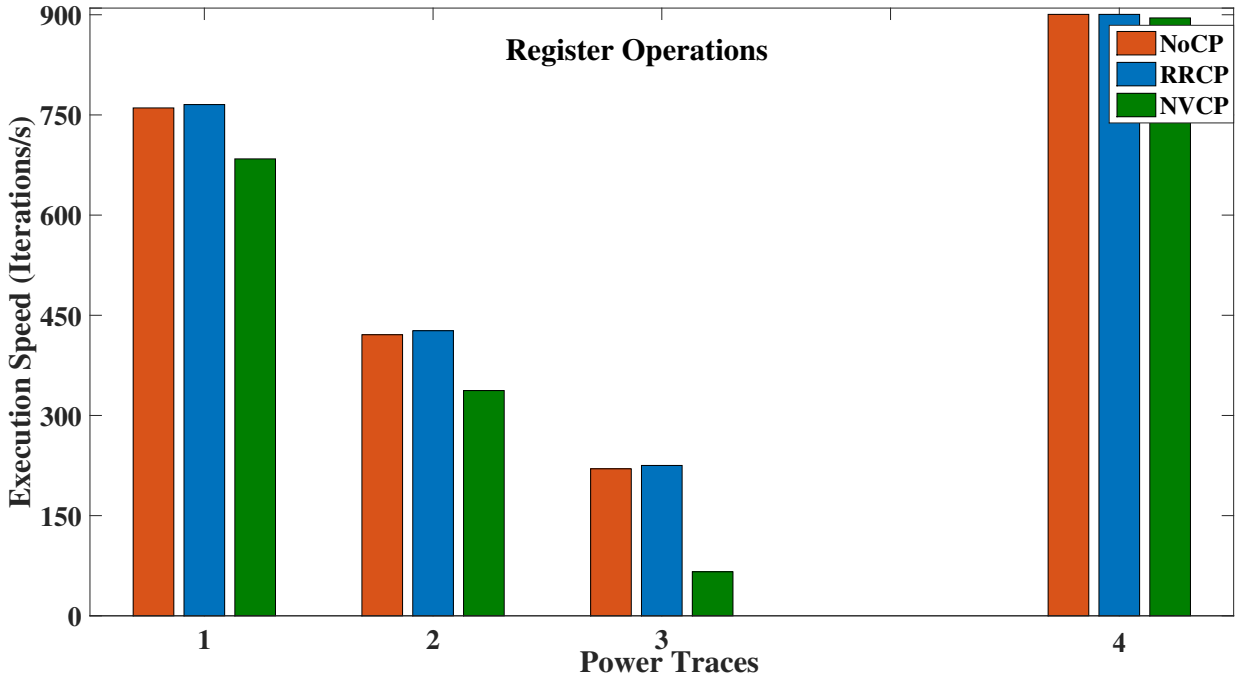


Figure 11: Execution Speed of Register Operation

Figure 11 shows the execution speed of register operation. As we can see from trace 4, when power is sufficient and stable, the NVP scheduler will not be activated. Therefore, the execution speeds of all comparing techniques are almost the same, except that NVCP

requires a constant voltage monitoring, however, the overhead of which is trivial and will be evaluated in section 3.7.1.7. When power becomes weaker, the execution speed for using NVCP is reduced more drastically. Take a look at power trace 3 and 4, under which, the source powers are extremely weak. Trace 3 enables NVCP to deliver the execution speed of 65.9 $Iterations/s$ which is only 29.9% of what the NoCP can deliver and 29.3% of what RRCP can deliver. The reason why the execution is slower when NVCP is active is that more energy will be used to execute atomic tasks. The execution speed of non-atomic tasks such as register operation will be reduced because of reduced energy. Under the worst-case scenario of power traces 4 where the source power is minimum and sparse, only NVCP shows the execution speed of 58.2 $Iterations/s$ and baselines show on speed. This is because the source power is so small and without sleep mechanism, NoCP and RRCP will cause the NVP system to stuck at cold starting once the stored energy has been drained out.

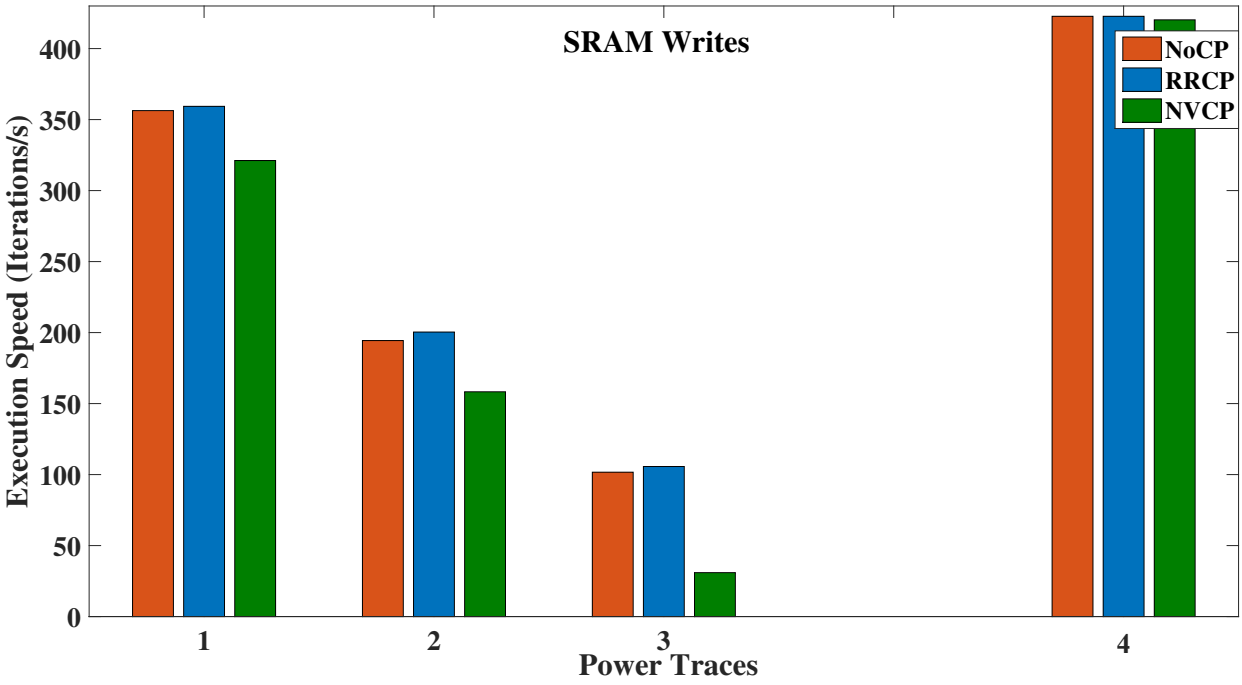


Figure 12: Execution Speed of SRAM Writes

Figure 12 shows the execution speed of SRAM writes. Similar to register operation, when power is sufficient and stable, NVCP will not be activated. When power becomes weaker,

the execution speed for using NVCP is reduced more drastically. Taking a look at trace 3, as more energy is used by atomic tasks, for SRAM writes, NVCP only delivers the execution speed of 30.95 $Iterations/s$ which is only 29.2% of what RRCP can deliver and 30.44% of what NoCP can deliver. Yet under the worst-case scenario of power trace, 4 where the source power is extremely weak and sparse, NoCP and RRCP cause NVP system to stuck at cold starting and only NVCP shows the execution speed of 27.32 $Iterations/s$.

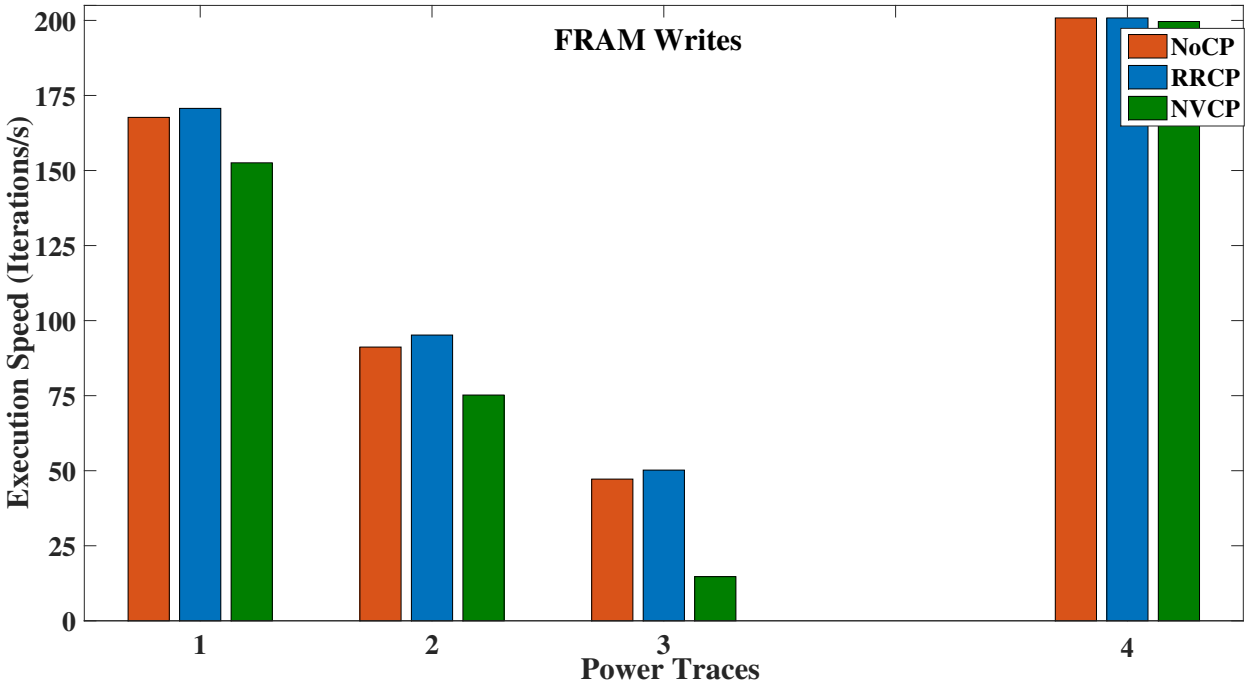


Figure 13: Execution Speed of FRAM Writes

Figure 13 shows the execution speed of FRAM writes. As we can see, similar to SRAM writes, the non-atomic tasks share less energy when power becomes weaker, thus, the execution speed of FRAM writes for using NVCP is reduced more drastically. under power trace 3, NVCP can only deliver the execution speed of 14.7 $Iterations/s$ which is only 29.28% of what RRCP can deliver and 31.14% of what NoCP can deliver.

Figure 14 shows the execution speed of the Thermometer which samples proximity temperature and transfers it into digital representations. The thermometer is an atomic task as the unfinished temperature data is inaccurate and useless. Therefore, when source power

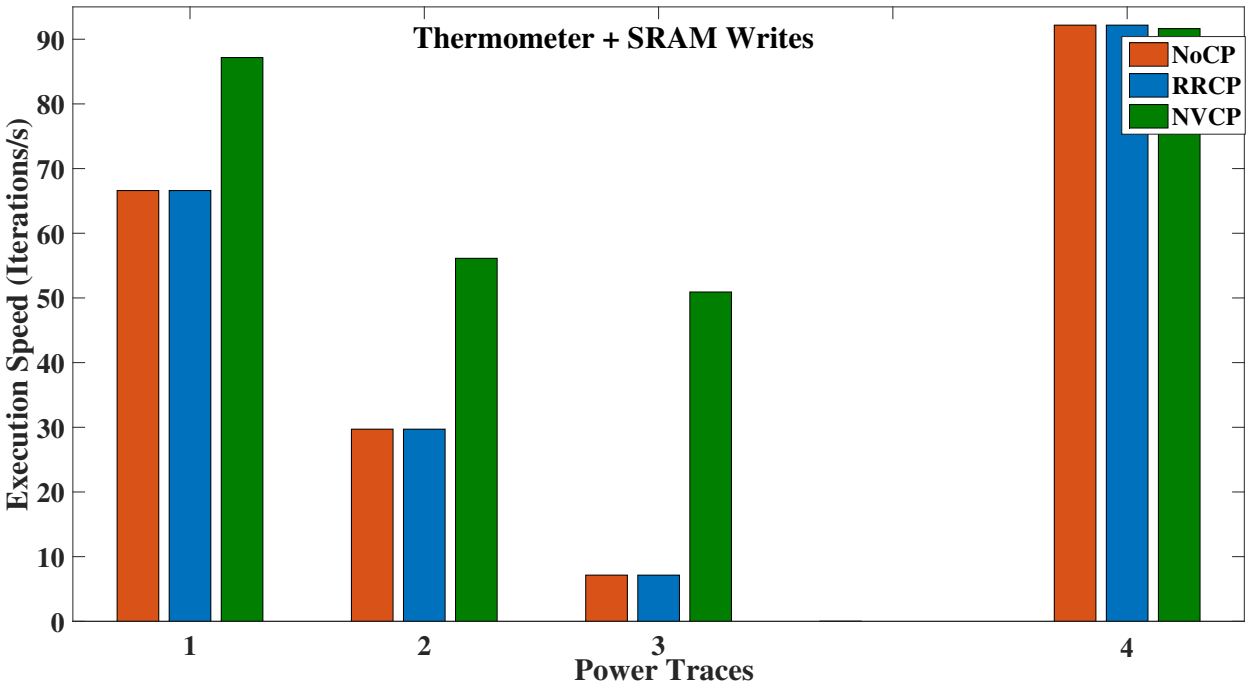


Figure 14: Execution Speed of Thermometer

becomes weaker, more energy portions will be given to atomic tasks to finish them before the power fails. As we can see, when power is sufficient and stable, NVCP will not be activated. When power becomes weaker, the execution speed for using NVCP is increased significantly over baseline techniques. Under power trace 3, NoCP and RRCP won't be able to keep the progress for the unfinished atomic tasks, so the execution speeds are the same. However, NVCP assigns a large portion of energy for atomic tasks and the execution speed of NVP is 50.92 $Iterations/s$ which is 712.77% of what baseline can deliver.

Figure 15 shows the execution speed of Accelerometer which samples 3-axis accelerations. Similar to Thermometer, Accelerometer is also an atomic task and the unfinished sensing data are inaccurate and useless. Therefore, when source power becomes weaker, a larger part of the energy will be used to execute Accelerometer for it to complete before power fails. As we can see, when power becomes weaker, the execution speed for using NVCP is increased significantly over baseline techniques. Under power trace 3, NVCP delivers the execution speed of 11.8 $Iterations/s$ which is 513.53% of what the baseline can deliver.

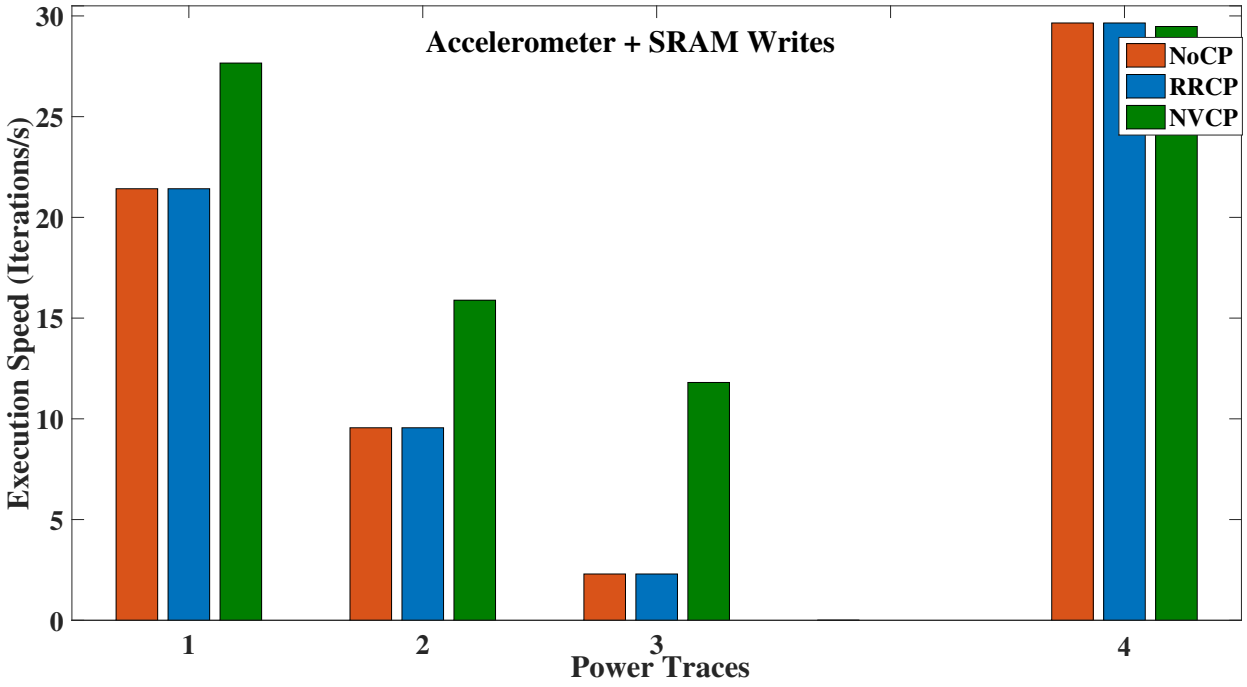


Figure 15: Execution Speed of Accelerometer

Figure 16 shows the execution speed of UART communication which constantly sends data through serial port given the baud rate of 9600. Similar to the other two atomic tasks, unfinished UART communication is useless and should be forfeited. The unfinished transmission data should be transmitted all over again when power is recovered. Therefore, when source power becomes weaker, a larger part of the energy will be used to execute UART communication to finish it before the power fails. As we can see, when power becomes weaker, the execution speed for using NVCP is increased significantly over baseline techniques. Under power trace 3, NVCP delivers the execution speed of 2 $Iterations/s$ which is 381.24% of what the baseline can deliver.

3.7.1.7 Efficiency and Overhead In this section, the efficiency and overhead of NVCP is evaluated. Considering the power consumption of each benchmark in Figure 9, we compare the energy efficiency of NVCP with baseline techniques. For trace 1, NVCP shows 10.87% improved energy efficiency than NoCP and 10.26% improved energy efficiency than

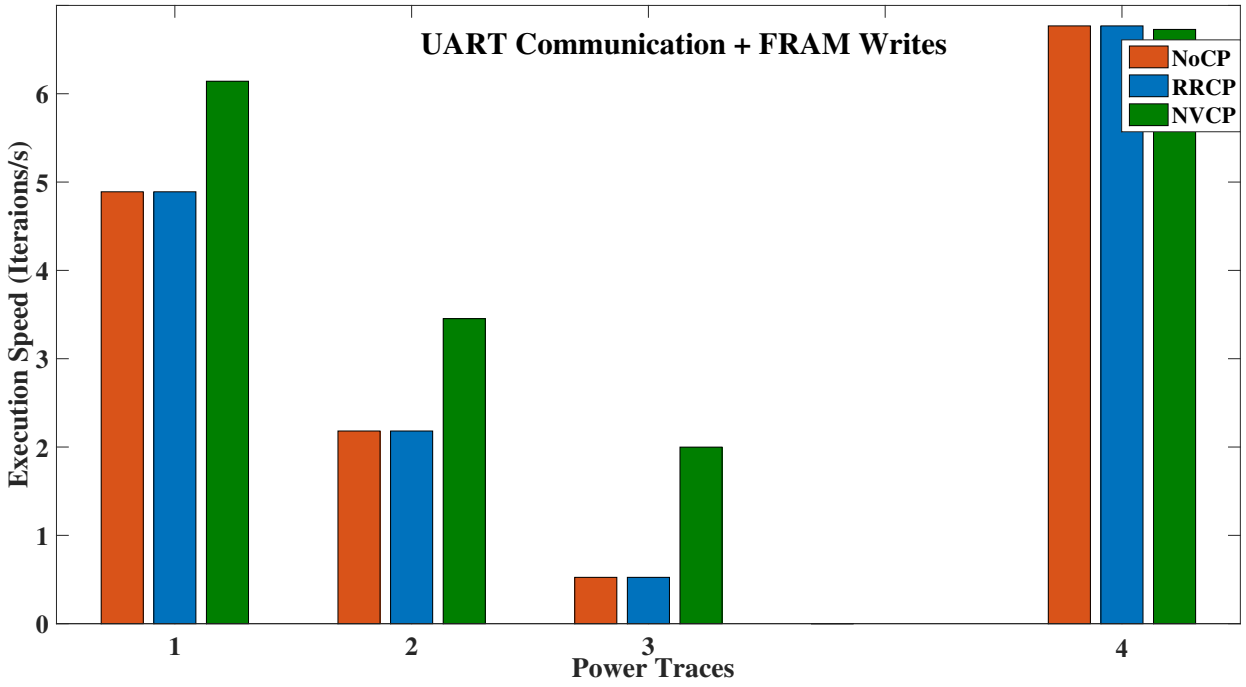


Figure 16: Execution Speed of UART communication

RRCP. For trace 2, NVCP shows 24.46% improved energy efficiency than NoCP and 22.48% improved energy efficiency than RRCP. For trace 3, NVCP shows 79.44% improved energy efficiency than NoCP and 74.79% improved energy efficiency than RRCP. For trace 4 NVP scheduler has not been triggered, there is no obvious difference in terms of energy efficiency. Overall, the advantages are significant.

Due to the periodical voltage monitor, there are extra amounts of energy and time overhead compared with baseline techniques. From Figure 18, both energy and time overhead increase when the source power becomes weaker. For the worst-case scenario in power trace 3, NVP scheduler generates an extra 3.52% energy overhead and 7.79% time overhead compared with baseline round-robin scheduler. However, considering the extra gained progress, the influence of overhead is negligible. Notice that based on the energy and time consumed by voltage monitor, the power consumption of voltage monitor is $0.08mW$ which is more than enough to support by source power if it can the charge up the capacitor.

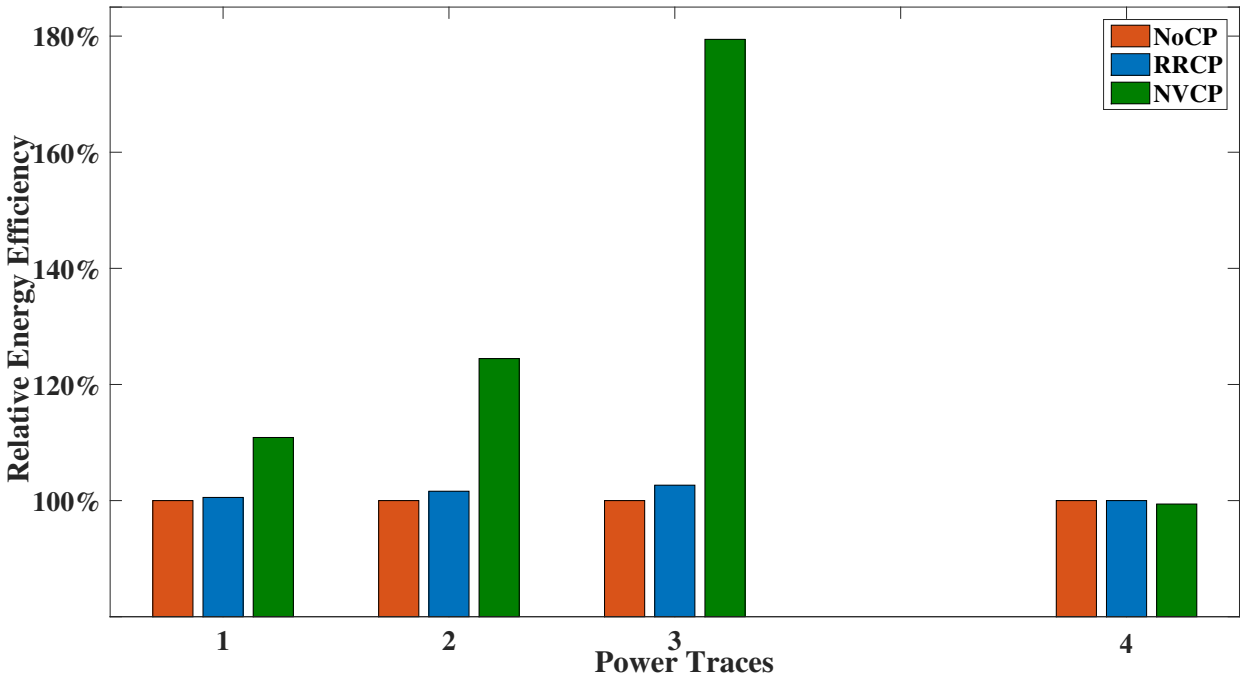


Figure 17: Energy Efficiency

3.7.2 Evaluation of NTS Scheduler

3.7.2.1 Hardware Platform The hardware platform includes a non-volatile IoT edge device and an energy harvesting module, which are detailed as follows.

- The experimental platform of a non-volatile IoT edge device is TI's MSP430FR5739 ultra-low-power evaluation board, which consists of a 16-bit MCU, a 10-bit ADC, a 1kB volatile SRAM, a 16KB nonvolatile FRAM memory, and different peripherals for sensing and data communication.

- For the energy harvesting module, a signal generator is used to generate ultra-low power. Then, the power regulator Bq25570 + LTC3459EDC harnesses the power and supply a constant voltage of 3.3V to power the edge device and a maximum voltage of 4.2V to charge the capacitor.

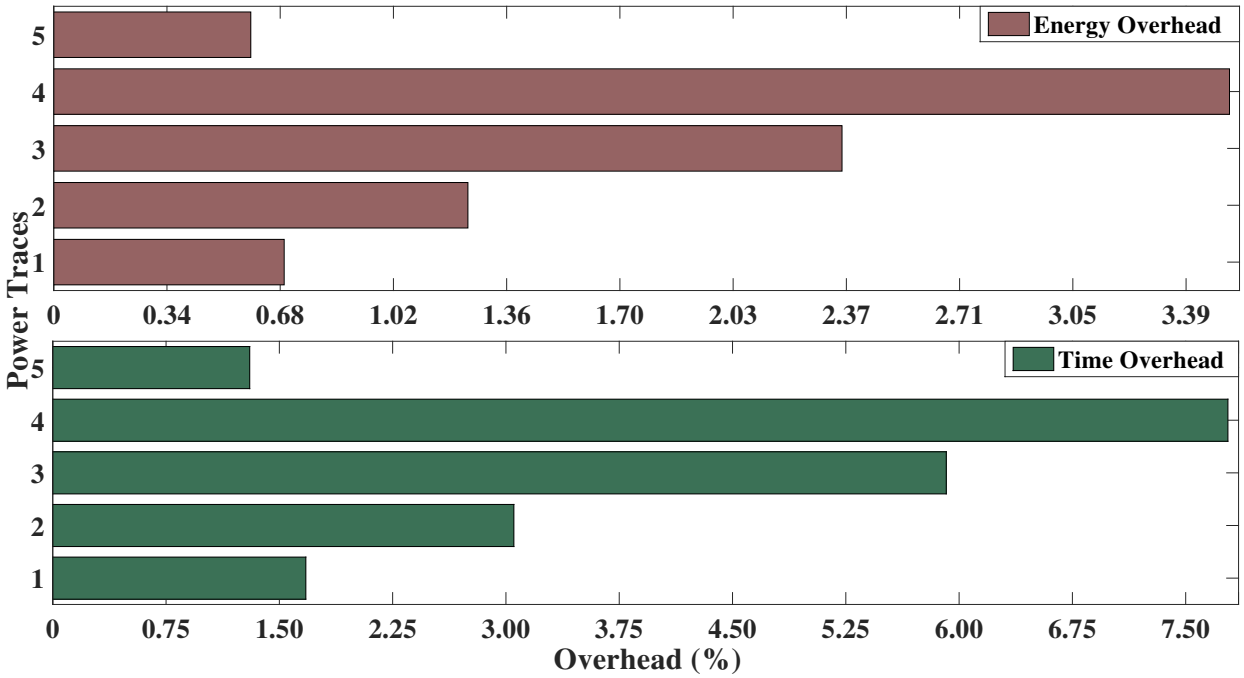
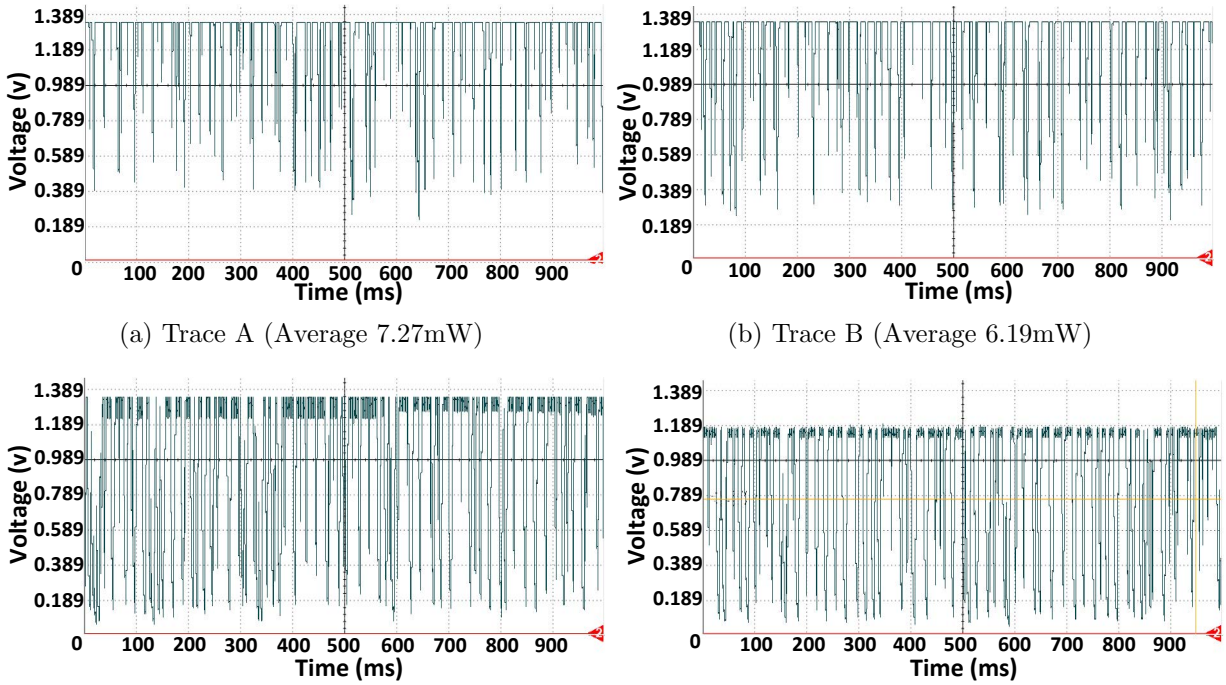


Figure 18: Energy and Time Overhead

3.7.2.2 Power Trace Four power traces in Figure 19 using signal generators for performance evaluation. The four power traces with different power magnitudes are recorded from the oscilloscope at the sample rate of $240KSa/s$. On average, the power they can provide is 7.27mW, 6.19mW, 4.94mW, and 4.11mW, respectively. The harvesting power is far less than the working power of edge devices.

3.7.2.3 Software Setup The parameter settings of the following experiments includes $C = 470\mu F$, $v_{ck} = 2.1V$, $m = 10$, and $T_{ADC} = 37.5\mu s$ For experimental evaluation, five benchmarks, SRAM, FRAM, Thermometer, Accelerometer, and UART, are used. Their required energy is show in Figure 20.

Here, SRAM writes data into sram; FRAM writes data into fram; Thermometer and Accelerometer sense temperature and acceleration; UART send data with the baud rate of 9600. All five benchmarks are iterative and make up two task trees. The first tree consists of Thermometer, SRAM, and UART, which is represented by Tree (therm). At first,



(a) Trace A (Average 7.27mW)

(b) Trace B (Average 6.19mW)

(c) Trace C (Average 4.94mW)

(d) Trace D (Average 4.11mW)

Figure 19: Power Traces

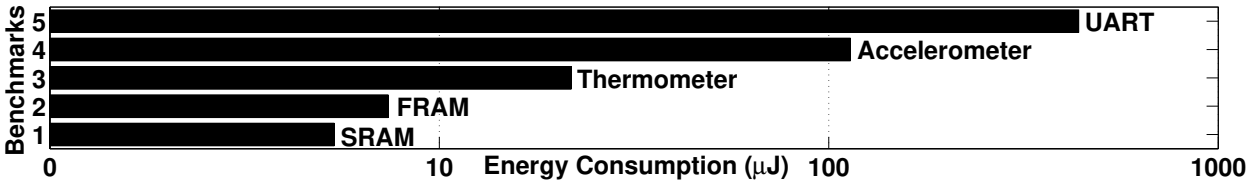


Figure 20: Required Energy of Each Benchmark.

Thermometer senses temperature data; then the data is written into a particular location on sram by SRAM. This sense-and-store operation continues for ten iterations. Then, these sensed temperature data is sent out by UART. After seven times of data transmission, a single execution period of Tree (therm) completes. Similarly, FRAM, Accelerometer, and UART make up the second tree, which is represented by Tree (accel). At first, Accelerometer and FRAM conduct sense-and-store operation repetitively ten times, during which the sensed acceleration data are written into a particular location on fram by FRAM. After that, UART sends out the acceleration data after seven times of data transmission, a single execution period of Tree (accel) completes. We also set up different execution deadlines for each task tree and measure the execution speed of each tree as well as the execution power regarding energy per iteration.

3.7.2.4 Benchmark Setup In this section, the performance of NTS will be evaluated regarding time and energy efficiency. The given five benchmarks make up two trees and are executed with the proposed NTS algorithm. Under each power trace, five pairs of deadlines are selected for each tree, which are $(5s, 2.5s)$, $(2.5s, 5s)$, $(8s, 4s)$, $(4s, 8s)$, and $(8s, 8s)$, respectively. Here, the first number of each pair represents the deadline for Tree (accel), and the second number of each pair represents the deadline for Tree (therm). The baselines for evaluating NTS include the Round-Robin (RRB) scheduler which executes each tree sequentially and the Priority-based Task Scheduling (PTS) proposed in [34] where the whole system is defined as MEU (maximizing energy utility). Fig. 21 shows the single execution period of each task trees with NTS vs. RRB and MEU (PTS).

In Fig. 21, the experiments are represented in five regions that have different deadlines for the targeting task trees. Regions are separated by the gray bars. The left side of each region shows the average single execution period of Tree (accel) and the right side shows the average single execution period of Tree (therm). The deadline for each tree is marked with a bold horizontal line. Notice that, for both trees, when supply power is weak, all techniques result in more time to finish a single execution period. For RRB, the deadlines of each task tree will not change the scheduling order which is the same situation when applying MEU (PTS). For MEU (PTS), the scheduling priority of is based on the stack size of each task

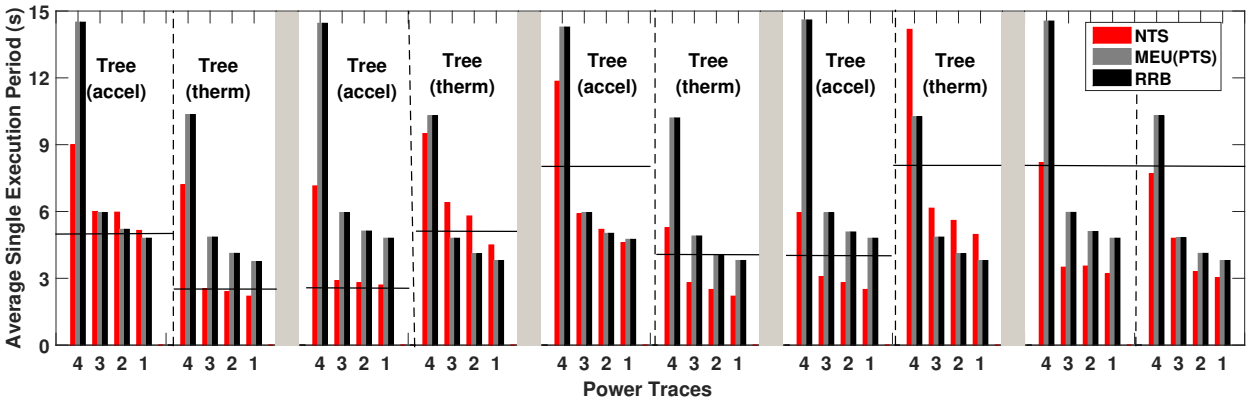


Figure 21: Average Single Execution Period with NTS.

which is ideal for scheduling tasks which are independent to each other. However, the data dependency of each task within the same tree renders the PTS useless.

For the first deadline pair $(5s, 2.5s)$ of NTS, Tree (accel) has a longer single execution period for Tree (therm), which is because Tree (Therm) has a shorter deadline, thus higher scheduling priority. For deadline pair, $(2.5s, 5s)$, Tree (accel) has a significantly shorter execution period with NTS over RRB and MEU (PTS) due to a higher scheduling priority. Due to the tight deadline $(2.5s)$, All three techniques cannot enable Tree (accel) to meet the deadline. For Tree (therm), due to a lower priority, only under power trace 1 can it meet the deadline with NTS. For deadline pair $(8s, 4s)$, the advantage of NTS over RRB and MEU (PTS) becomes salient, that is NTS only fails twice, yet RRB and MEU (PTS) fails four times. For deadline pairs, $(4s, 8s)$ and $(8s, 8s)$, NTS only fails twice and once respectively. Overall, both RRB and MEU (PTS) result in 23 times of fails to meet the deadlines while NTS only results in 18 times of fails.

Considering the execution power of each benchmark shown in Figure 20, the average execution power of each benchmark under different power traces is shown in Fig. 22.

NTS utilizes 36.1% more energy on average than RRB and MEU (PTS). Since the amount of harvested energy is the same for both schedulers, NTS has a higher energy efficiency than the round-robin scheduler. The reason for NTS to have a higher efficiency is that both

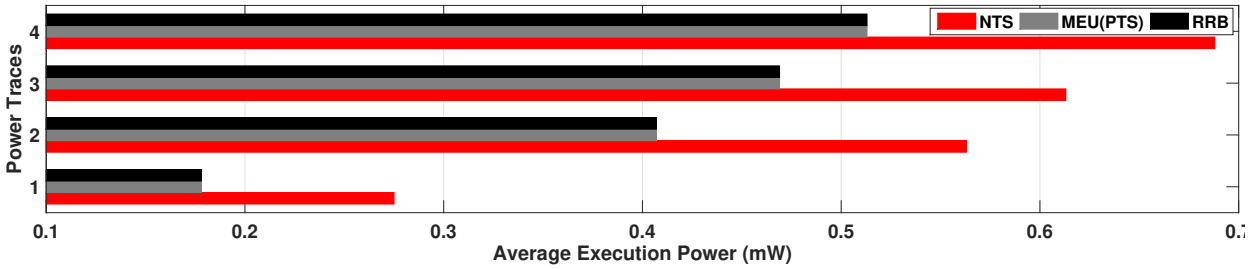


Figure 22: Average Execution Power of Benchmarks Under Different Power Traces.

RRB and MEU (PTS) are unaware of task classification. If the last executing task before conducting checkpointing is an atomic task such as Acceleration or Thermometer, the ongoing execution has to start over once the power comes back on again. However, for NTS, the last task before conducting checkpointing is always a non-atomic task such as SRAM and FRAM, no progress setback will happen. This results in better energy utilization.

3.8 Summary

This chapter proposes NVP-aware task schedulers to maximize the overall task execution progress. Considering the tasks' atomicity, the unfinished atomic tasks will be scheduled for completion as many as possible before the power outage to mitigate the progress loss. The proposed NVP scheduler can be easily incorporated into embedded systems as an auxiliary to the existing scheduler with great compatibility which takes effect when harvesting power is not enough. Experiments confirm the effectiveness of the proposed techniques.

4.0 Intelligent Checkpointing Scheme

In this chapter, the basic checkpointing technique will be optimized for better reliability and energy-efficiency

4.1 Motivation

Although the progress of atomic tasks can be saved. The checkpointing operation also needs to be optimized mainly due to three reasons. First, as the checkpointing involves write operation on non-volatile memory, which is both time and energy-consuming, extra time and energy are required to support checkpointing. Considering the limited harvesting power, checkpointing can significantly affect the energy overhead especially when harvesting power becomes extremely small and unstable. Second, as the endurance of non-volatile memory gradually wears out with each new write operation, checkpointing also threatens the lifetime of edge devices with NVP. Third, if checkpointing is unsuccessful, a significant progress setback would occur. To tackle the aforementioned challenges, an intelligent checkpointing scheme is proposed which not only can ensure a successful checkpointing but also can predict the necessity of conducting checkpointing to avoid excessive checkpointing overhead.

4.2 Related Work

The intermittent nature of energy harvester increases the energy consumption of program execution. This is because, when the power outage happens, the on-going processor' states along with all the other volatile data will be lost. When the power comes back on, all the unfinished program executions have to start over again resulting in reduced energy efficiency. Therefore, it is necessary to maintain the execution progress for each program. To enable continuous execution of programs across different power cycles, both hardware [16, 53, 43,

44, 27, 23] and software [14, 26, 28, 41, 42] recovery mechanisms have been proposed to support an automatic recovery of computation tasks on the processor.

Hardware checkpointing stores the system state and data automatically by hardware. For example, Yu et al. [16] propose a non-volatile processor architecture that integrates non-volatile elements into volatile memory at bit granularity. Wang et al. [53] design a FRAM based processor, which attaches an NV-FRAM cell to each volatile standard flip-flop. The flip-flops are accessed for normal execution while the FRAM cells are used to checkpoint the states in flip-flops at power failure. This processor can backup and restore the processor state and data within $3\mu s$. Sakimura et al. propose the non-volatile magnetic flip-flops [43] and a 20MHz non-volatile micro-controller with STT-RAM [44]. Recently, Liu et al. propose an enhanced NVP based on ReRAM which has the highest integration level [27]. Also, Li et al. propose the non-volatile I/O (NVIO) enabling efficient automatic reconfiguration of I/O interfaces [23].

Besides hardware checkpointing, there are also software mechanisms that checkpoint the processor's state and other volatile data into non-volatile memories. For example, Mementos [42] is a software mechanism for transiently powered RFID-scale devices. Some trigger points are placed after each call instruction or at each loop latch. At run-time, when these trigger points are reached, the supply voltage is checked with an ADC. If this voltage is below a threshold, a snapshot of the system state is saved to the flash memory. Quickrecall [14] integrates FeRAM into the main memory to increase the checkpointing efficiency which reduces the backup data size and lowers the failure voltage threshold. Hibernus [6] and Hibernus++ [6] propose interrupt-based checkpointing mechanisms. In these mechanisms, the system stated is checkpointed only once immediately before the power failure, then the system hibernates. This mechanism requires frequent voltage checking and automatic interrupts to checkpoint or restore the system state. MEU [34] proposes prototyping techniques for a joint reduction of software and hardware overhead with software solutions. MEU requires constant harvesting power analysis which is expensive and tends to be less accurate compared with the proposed techniques.

4.3 Statisitic based Checkpointing Avoidance

In this subsection, a tentative checkpointing Withdrawal (TCW) algorithm is proposed based on statistic evaluation which allows the edge device to tentatively avoid checkpointing so that further reduction of checkpointing overhead upon NTS can be achieved.

Normally, the system needs to checkpoint before it goes to sleep mode when detecting a low voltage. However, if the harvesting power is enough to charge up the capacitor after the edge device enters sleep mode, this checkpointing is unnecessary and can be avoided. In this way, a substantial amount of energy for checkpointing can be used for program execution instead. However, the challenge is that the edge device needs to know whether the harvesting power is enough to charge up the capacitor when the edge device is in sleep mode. Otherwise, the edge device is at risk of losing all the data during sleep mode, which incurs progress setback. Therefore, TCW is proposed to evaluate the risk and avoid checkpointing properly, which is detailed in Algorithm 4.3.1.

The inputs of TCW include initial charging voltage v_{ini}, wake-up voltage v_{wak}, charging period τ, the previous wake-up state s_p, the current wake-up state s_c, estimated charging speed $\widehat{cs}$, and the confidence indicator of power goodness $\lambda \in [0, 1]$. Here, the wake-up state indicates whether the edge device wakes up from the sleep mode or reboots from the power outage (1: waking up from the sleep mode; 0: rebooting from a power outage). Since the harvesting power is inherently weak and unstable, at the very beginning, TCW conservatively initiates λ as 0. The output of TCW is *out* which indicates the necessity of the next checkpointing (1: necessary; 0: unnecessary).

The key of TCW is using the latest two wake-up states (s_c and s_p) to determine the necessity of the next checkpointing. Specifically, if current wake-up state s_c and previous wake-up state s_p are both 1, TCW considers that the harvesting power is sufficient and could last longer. In this case, the next checkpointing is more likely to be avoided than in other cases. If current wake-up state $s_c = 1$ but previous wake-up state $s_p = 0$, TCW thinks that the harvesting power of the current charging cycle is only sufficient temporarily and may become insufficient at any moment shortly. In this case, the next checkpointed is less likely to be avoided than in the previous case. Finally, if the current wake-up state $s_c = 0$, the

Algorithm 4.3.1 Tentative Checkpointing Withdrawal (TCW)

Input: v_{ini}, v_{wak}, τ, s_p, s_c, $\widehat{cs}$, and λ
Output: *out* /* the necessity of the next checkpointing (0: unnecessary; 1: necessary) */

```
1: λ = (λ + s_c)/2;                                   /* update the confidence of power goodness */
2: if s_c = 1 then                                    /* current wake-up is from sleep mode */
3:    cs = (v_wak^2 − v_ini^2)/τ;                     /* calculate the charging speed */
4:    if s_p = 1 then                                 /* previous wake-up was from sleep mode */
5:       if cs ≥ λ ∗ ĉs then
6:          out = 0;
7:       else
8:          out = 1;
9:       end if
10:   else                                            /* previous wake-up was from reboot */
11:      if λ ∗ cs ≥ ĉs then
12:         out = 0;
13:      else
14:         out = 1;
15:      end if
16:   end if
17: else                                              /* current wake-up is from reboot */
18:   cs = 0;
19:   out = 1;
20: end if
21: ĉs = (cs + ĉs)/2;                                 /* update the estimated charging speed */
22: return out =0
```

harvesting power is insufficient to charge up the capacitor when the edge device is in the sleep mode and has drained out all the stored energy and rebooted after the harvesting power recovery. In this case, TCW cannot acquire the actual charging period of τ. Therefore, TCW conservatively forces the next checkpointing to be mandatory no matter how large cs is. An auxiliary example in Fig. 23 details TCW.

s	s_{ini}=0	s_1=1	s_2=1	s_3=0	s_4=0	s_5=1	s_6=1	s_7=1	s_8=1
cs	cs_{ini}=0	cs_1=10	cs_2=3	cs_3=0	cs_4=0	cs_5=4	cs_6=8	cs_7=12	cs_8=5
λ	λ_{ini}=0.250	0.625	0.813	0.406	0.203	0.602	0.801	0.900	0.950
$\widehat{cs}$	$\widehat{cs}_{ini}$=8.000	4.000	7.000	5.000	2.500	1.250	2.625	5.313	8.656
out	out_{ini}=1	0	1	1	1	0	0	0	1

Figure 23: Four Out of Nine Checkpointings Can Be Avoided without Progress Setback.

Fig 23 shows nine times of continuous decision-making for checkpointing, $s_{ini} \rightarrow s_8$ represents the wake-up states, $cs_{ini} \rightarrow cs_8$ represents the charging speeds during each charging cycle, λ represents confidence indicator, $\widehat{cs}$ represents estimated charging speed, and out represents the checkpointing decision.

Initially, $s_c = s_{ini} = 0$, according to line 17, the edge device wakes up from reboot. In this case, the charging speed is unmeasurable, hence, TCW considers that the charging power is insufficient to charge up the capacitor and power outage could continue happening. Then, it set $cs_{ini} = 0$ and $out = 1$ to indicate that the next checkpointing is necessary. Before returning out, given $\widehat{cs_{ini}} = 8$, TCW updates $\widehat{cs} = 4$ based on line 21 which will be used during the next cycle.

After the 1^{st} charging cycle, the $s_c = s_1 = 1$. In this case, given λ_{ini}, TCW first updates λ based on line 1. Since currently $s_c = 1$ and $s_p = s_{ini} = 0$, TCW considers that the harvesting power is only temporarily sufficient and may become insufficient at any moment shortly. In this case, TCW calculates out based on the criterion in line 11. Specifically, On one hand, if λ is small, TCW is less confident that the current sufficient harvesting power could last long, hence cs needs to be higher to convince TCW to avoid checkpointing. On the other hand, if λ is large, TCW is more confident that current sufficient harvesting power could last long,

hence a smaller cs can convince TCW to avoid checkpointing. Yet, overall, cs needs to be larger than $\widehat{cs}$ as TCW still holds the idea that the harvesting power may become insufficient shortly. Here, because $cs_1 = 10$, $\widehat{cs} = 4$, and $\lambda = 0.625$, we have $\lambda * cs > \widehat{cs}$, hence, the $out = 0$, which means the next checkpointing can be avoided. Finally, TCW updates $\widehat{cs} = 7$ before returning the output.

After the 2^{nd} charging cycle, $s_c = 1$. In this case, TCW first updates λ. Since currently $s_p = s_c = 1$, TCW considers that the harvesting power is sufficient to charge up the capacitor when the edge device is in sleep mode. In this case, TCW calculates out based on the criterion in line 5. As harvesting power gets better, λ and $\widehat{cs}$ become larger, even if cs is smaller than $\widehat{cs}$, the harvesting power can be still sufficient to charge up the capacitor when the edge device is in sleep mode. In this case, cs can allow being smaller than $\widehat{cs}$ to avoid more checkpointing operations. Here, because $cs_2 = 3$, $\widehat{cs} = 7$, and $\lambda = 0.813$, we have $cs < \lambda * \widehat{cs}$, hence, the $out = 1$, so the checkpointing is necessary. The reason behind this checkpointing is that TCW noticed the sharp drop in charging power which is normally a sign of an imminent power outage. So it is necessary to conduct checkpointing. Finally, TCW updates $\widehat{cs}$ before returning the output.

From the following 3^{rd} to 8^{th} charging cycles, TCW determines out following the same procedure as before. In the end, four out of nine checkpointing operations were avoided without inducing progress setback, which saves a significant amount of energy for program execution.

Notice that, for harvesting power, TCW is cautious about the improvements while sensitive to the drops. Specifically, on the one hand, when harvesting power changes from weak to strong, unless the improvements are significant (e.g. $cs_4 \rightarrow cs_5$), TCW will cautiously consider that checkpointing is necessary. On the other hand, when harvesting power changes from strong to weak, even if harvesting power still seems to be far beyond the sleep power of edge devices (e.g. $cs_7 \rightarrow cs_8$), TCW will sensitively consider that checkpointing is necessary. This can significantly reduce the misprediction of checkpointing, yet it cannot always guarantee the correctness of the prediction. For sudden complete power outage from a stable power supply ($s_p = 1$ and $s_c = 1$), the edge device will lose the previous computation state by failing to checkpoint. However, the sleep power can be infinitesimally small compared

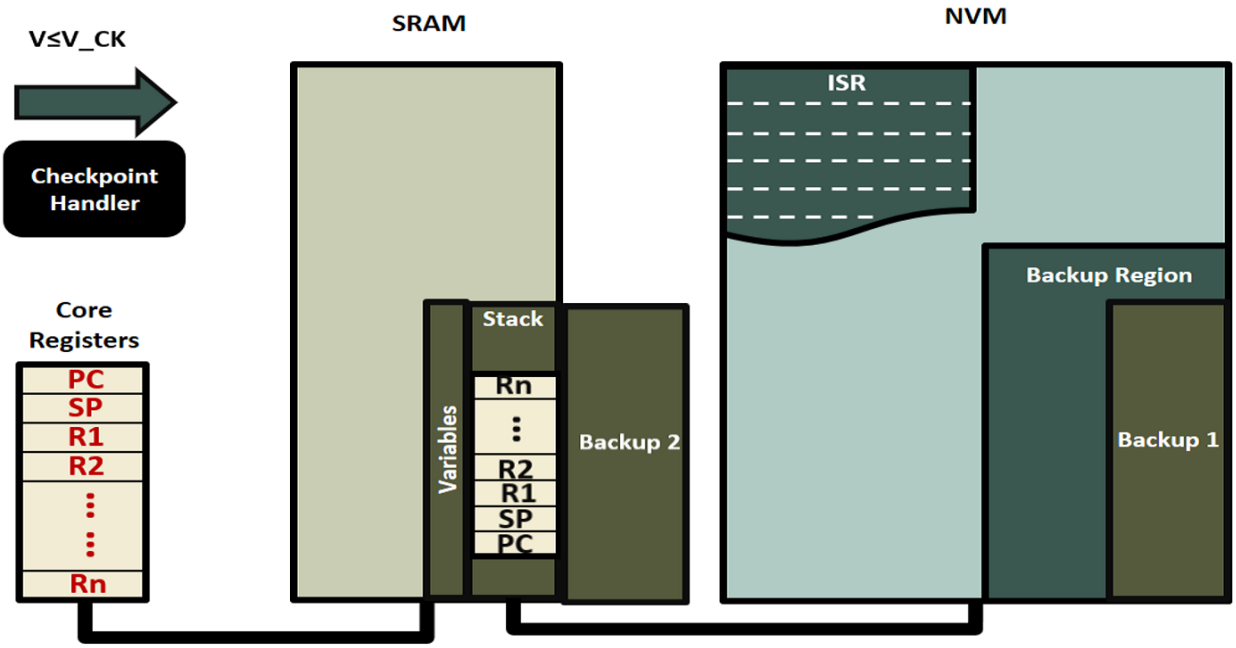


Figure 24: Dual-Backup Checkpointing Handler.

with the working power, which allows the edge device to stay alive for a significant amount of time by using residual energy in the storage before harvesting power comes back on again. For instance, with our experimental settings, the residual energy in the capacitor after checkpointing is around $300\mu J$ and the standby power of edge device in low power mode can be as low as $20\mu W$. In the worst case, the edge device can last more than two minutes by totally sustaining on its residual energy, which makes our system robust enough to cope with most intermittent energy harvesting scenarios where the power outage is usually temporary lasting for less than a minute.

4.3.1 Secure Checkpointing

In the case of checkpointing failures, two areas in NVM should be reserved for checkpointing alternatively. With double backups, even if current checkpointing fails, the NVP system can still roll back to the previous successful checkpoint other than start over from the very beginning as shown in Figure 24.

During each checkpointing, checkpointing handler should also compare the new data with the latest successful backup and only checkpoints the differences. This is to minimize the energy consumption of checkpointing.

4.4 Experiments

4.4.1 Experimental Setup

This section details the experimental setup both on hardware and software.

4.4.1.1 Hardware Platform The hardware platform includes a non-volatile IoT edge device and an energy harvesting module, which are detailed as follows.

- The experimental platform of a non-volatile IoT edge device is TI's MSP430FR5739 ultra-low-power evaluation board, which consists of a 16-bit MCU, a 10-bit ADC, a 1kB volatile SRAM, a 16KB nonvolatile FRAM memory, and different peripherals for sensing and data communication.
- For the energy harvesting module, a signal generator is used to generate ultra-low power. Then, the power regulator Bq25570 + LTC3459EDC harnesses the power and supply a constant voltage of 3.3V to power the edge device and a maximum voltage of 4.2V to charge the capacitor.

4.4.1.2 Power Trace Four power traces in Figure 19 using signal generators for performance evaluation. The four power traces with different power magnitudes are recorded from the oscilloscope at the sample rate of $240KSa/s$. On average, the power they can provide is 7.27mW, 6.19mW, 4.94mW, and 4.11mW, respectively. The harvesting power is far less than the working power of edge devices.

4.4.1.3 Software Setup The parameter settings of the following experiments includes $C = 470\mu F$, $v_{ck} = 2.1V$, $m = 10$, and $T_{ADC} = 37.5\mu s$ For experimental evaluation, five

benchmarks, SRAM, FRAM, Thermometer, Accelerometer, and UART, are used. Their required energy is show in Figure 20.

Here, SRAM writes data into sram; FRAM writes data into fram; Thermometer and Accelerometer sense temperature and acceleration; UART send data with the baud rate of 9600. All five benchmarks are iterative and make up two task trees. The first tree consists of Thermometer, SRAM, and UART, which is represented by Tree (therm). At first, Thermometer senses temperature data; then the data is written into a particular location on sram by SRAM. This sense-and-store operation continues for ten iterations. Then, these sensed temperature data is sent out by UART. After seven times of data transmission, a single execution period of Tree (therm) completes.

Similarly, FRAM, Accelerometer, and UART make up the second tree, which is represented by Tree (accel). At first, Accelerometer and FRAM conduct sense-and-store operation repetitively ten times, during which the sensed acceleration data are written into a particular location on fram by FRAM. After that, UART sends out the acceleration data after seven times of data transmission, a single execution period of Tree (accel) completes. We also set up different execution deadlines for each tree and measure the execution speed of each tree as well as the execution power regarding energy per iteration.

4.4.1.4 Performance of TCW In this section, the performance of TCW will be evaluated upon applying NTS. TCW avoids unnecessary checkpointing to reduce the further energy consumption of checkpointing. The baseline for comparison includes NTS and tentative checkpointing avoidance (TCA) in MEU which avoids checkpointing based on a more complicated evaluation of harvesting power. Fig. 25 shows the single execution period of each task with TCW vs. NTS and MEU (TCA).

In Fig. 25, for the first deadline pair $(5s, 2.5s)$, upon implementing NTS, TCW can further improve the energy efficiency, which meets deadlines four times. For deadline pair, $(2.5s, 5s)$, TCW further meets the deadline four times. For deadline pair, $(8s, 4s)$, $(4s, 8s)$, and $(8s, 8s)$, TCW enables both trees to meet their deadline under all power traces. Considering the execution power of each benchmark, the utilized energy of each method is shown in Fig. 26. With TCW, energy efficiency can be further improved by 77.95% on average compared with

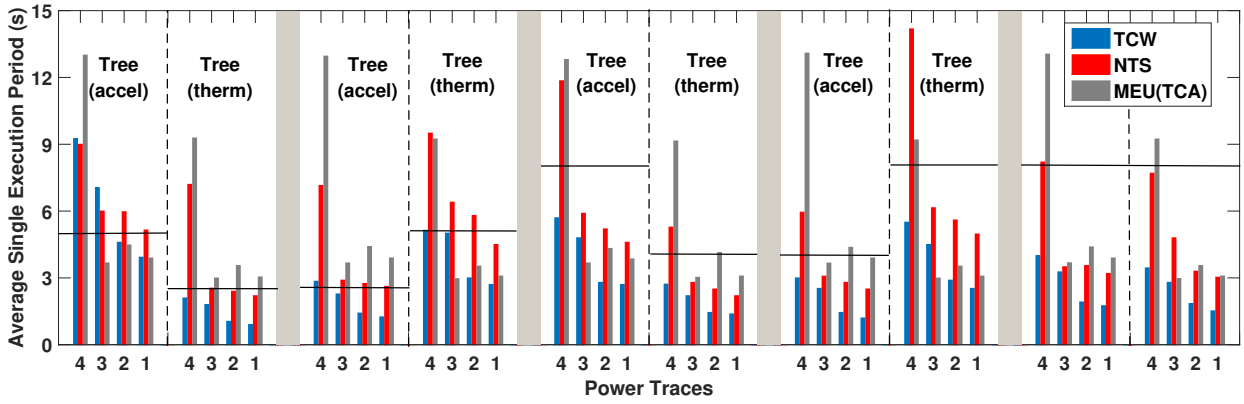


Figure 25: Single Execution Period with TCW.

applying NTS only. While with MEU (TCA), the energy efficiency can only be improved by 16.38% on average upon applying MEU (PTS). Hence the improvement of TCW is significant Compared with MEU (TCA). The main reason for such improvement is that MEU (TCA) responds to the harvesting power fluctuation much slower than TCW, which results in more progress setback and unnecessary checkpointing. Overall, after applying TCW, more energy can be used for execution which significantly improves energy efficiency.

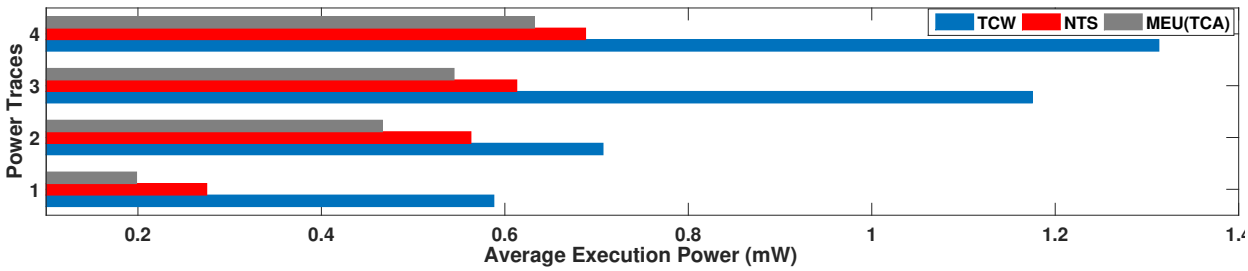


Figure 26: Average Execution Power of Benchmarks Under Different Power Traces.

4.5 Summary

This chapter proposes an intelligent checkpointing scheme that not only can ensure a successful checkpointing but also can predict the necessity of conducting checkpointing to avoid excessive checkpointing overhead. The experimental results show that the proposed checkpointing scheme not only can significantly reduce the energy overhead but also be able to help the system meet the time bound for execution.

5.0 CPU Frequency Modulation

In this chapter, a CPU frequency Modulator is proposed which adjusts the runtime CPU clock frequency adaptively to reduce energy consumption due to inappropriate runtime CPU clock frequency configuration.

5.1 Motivation

Even though the checkpointing overhead can be minimized, inappropriate execution frequencies can also significantly reduce runtime energy efficiency which is defined as energy utility. The optimal frequency varies from program to program. This is because, for task execution, the higher clock frequency can help reduce the execution time yet it also induces extra power consumption. If the reduction of time is lower than the increment of power, the energy efficiency becomes lower, In this case, the higher clock frequency is not advisable. Similarly, the lower clock frequency can help reduce the power consumption yet it also induces extra time for execution. If the reduction of power is lower than the increment of time, energy efficiency also becomes lower. In this case, the lower clock frequency is not advisable. To further understand the implication of clock frequency to the energy efficiency, an observation is conducted on TI' MSP430FR6989 with seven fixed frequencies available. The testing benchmark is 16_bit_2dim which reads from ADC as an input data source. Figure 27 shows the execution speed of the device when running this benchmark with seven available clock frequencies.

As we can see, under different clock frequencies, the execution speed varies greatly and thus the energy efficiency. Specifically, the frequency that achieves the best energy efficiency is neither the maximum nor the minimum frequency. Instead, when the clock frequency equals to $4MHz$ (half of the maximum frequency), maximum energy efficiency is achieved. The truth behind this observation is that, when the clock frequency is high, the delay of ADC sampling becomes a bottleneck of the execution speed. When the clock frequency is

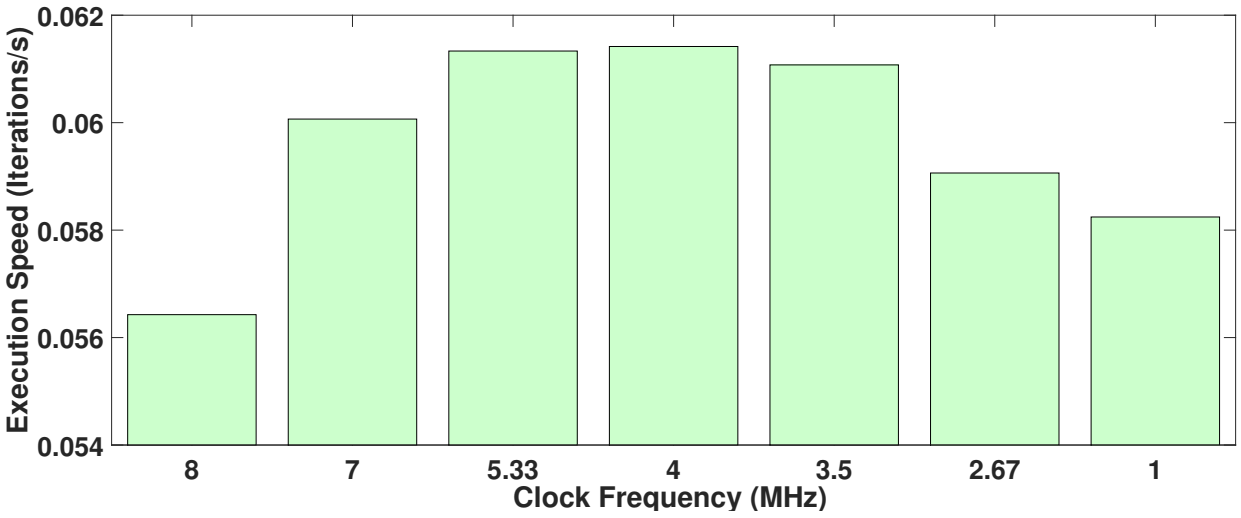


Figure 27: Energy efficiency with different clock frequency

low, the working power is not reduced with the same magnitude to the runtime execution speed. Therefore, selecting a clock frequency appropriately is crucial to each program.

5.2 Modeling and Analysis

This subsection will explore the potential of energy utility maximization through runtime clock frequency modulation.

Assume there are n tasks running on the edge device and the runtime clock frequency for the i^{th} task is denoted by f_i. Let $S^i_{exe}(f_i)$ denote the amount of execution progress per time unit and $P^i_W(f_i)$ denote the execution power, $S^i_{exe}(f_i)$ and $P^i_w(f_i)$ are both in a positive correlation to f_i. When frequency switches, the energy ε_{sw} is required for oscillator settling. With these parameters, the energy utility λ can be calculated as

$$\lambda\langle f_1, f_2, ...f_n\rangle = \frac{\sum_{i=1}^{n} \overline{S^i_{exe}(f_i)}\tau^i_{ex}}{\sum_{i=1}^{n}(\overline{P^i_W(f_i)}\tau^i_{ex} + m\kappa_i\varepsilon_{sw})} \tag{5.1}$$

Let m represent the number of execution rotations for all tasks. $\kappa_i \in \{0, 1\}$ is a frequency switch indicator. If $f_i = f_{i-1}$, then $\kappa_i = 0$; otherwise, $\kappa_i = 1$. For the i^{th} task, let $\overline{S^i_{exe}(f_i)}$ denote the average execution speed, $\overline{P^i_W(f_i)}$ denote the average working power, f_i denote the clock frequency, and τ^i_{ex} denote the amount of execution time, given the progress that the i^{th} task has made as g_i, we have

$$\lambda\langle f_1, f_2, ...f_n\rangle = \frac{\sum_{i=1}^{n} g_i}{\left[\sum_{i=1}^{n}\left(\frac{\overline{P^i_W(f_i)}g_i}{\overline{S^i_{exe}(f_i)}} + m\kappa_i\varepsilon_{sw}\right)\right]} \tag{5.2}$$

For edge devices, since task routine is preset, the progress ratio between any two tasks tends to be a constant in a long run. In this case, we have $G_i = g_i/\sum_{i=1}^{n} g_i$ as a constant. In addition, during each execution rotation, the total amount of progress of all tasks is a constant which is defined as $G^s = \sum_{i=1}^{n} g_i/m$. Hence, eq. (5.2) can be transformed into

$$\lambda\langle f_1, f_2, ...f_n\rangle = \left(\sum_{i=1}^{n} \lambda_i^{\alpha} G_i + \frac{\varepsilon_{sw}}{G^s}\sum_{i=1}^{n}\kappa_i\right)^{-1} \tag{5.3}$$

Here, $\lambda_i^{\alpha} = \frac{\overline{P^i_W(f_i)}}{\overline{S^i_{exe}(f_i)}}$. As G_i and G^s are constants, in order to maximize λ, both $\sum_{i=1}^{n}\kappa_i$ and $\sum_{i=1}^{n}\lambda_i^{\alpha}G_i$ should be minimized. Achieving such optimality could be time consuming. Yet, considering the fact that $\frac{\varepsilon_{sw}}{G^s}\sum_{i=1}^{n}\kappa_i \ll \sum_{i=1}^{n}\lambda_i^{\alpha}G_i$, the utility optimization priority should be focused on

$$Minimize \quad \lambda_i^{\alpha} = \frac{\overline{P^i_W(f_i)}}{\overline{S^i_{exe}(f_i)}} \quad \forall i \in \{1, 2, .., n\} \tag{5.4}$$

Generally, $P^i_w(f_i)$ increases when f_i becomes larger. However, this is not true for $S^i_{exe}(f_i)$ which may be affected by the latency of accessing hardware modules such as memory, I/O, and ADC. Therefore, a minimum of λ_i^{α} varies from task to task. Aside from minimizing λ_i^{α}, $\sum_{i=1}^{n}\kappa_i$ can also be reduced by properly scheduling independent tasks so that more adjacent tasks could have the same optimal frequency. In this way, less frequency switch overhead is required and further energy utility can be achieved.

5.3 Frequency Modulator

In this section, the Frequency Modulator (FM) is introduced which picks up the optimal clock frequency for each task so that maximum energy utility can be achieved. The implementation of FM contains both off-line and on-line stages which are detailed as follows.

5.3.1 Off-line Stage

At the off-line stage, the FM first needs to measure the optimal MCU frequency for each task. Then, a task is scheduled based on these optimal frequencies.

5.3.1.1 Optimal Frequency Measurement: As tasks on edge devices are generally iterative and involve accessing different hardware modules, the bottleneck of execution utility varies from task to task. In this case, based on the analysis in section 5.2, for each task $tsk_i \in \langle Tsk \rangle$, FM needs to measure the average power $\overline{P^i_W(f^j)}$ and execution speed $\overline{S^i_{exe}(f^j)}$ with any operable frequency f^j. Then for each tsk_i, $f^{opt}_i = \arg\min \frac{\overline{P^i_W(f^j)}}{\overline{S^i_{exe}(f^j)}}$.

5.3.1.2 Optimal Frequency Aware Task Scheduling: After acquiring the optimal frequency $\langle f^{opt} \rangle$, based on section 5.2, the utility can be further improved with the reduction of frequency switching overhead. Considering task dependency, all tasks can be divided into independent task chains. Therefore, for any two chains TC_α and TC_β, if $f^{opt}_{TC_\alpha.tail} = f^{opt}_{TC_\alpha.head}$, then TC_α and TC_β can form a new chain with $TC_\alpha.tail.next \leftarrow TC_\alpha.head$.

Figure 28 shows an ideal frequency matching based task schedule where tasks within the same dash rectangle box are with data dependencies and form an independent task chain to others. As we can see, $f^{opt}_h = f^{opt}_{h+1}$, $f^{opt}_k = f^{opt}_{k+1}$, $f^{opt}_m = f^{opt}_{m+1}$, $f^{opt}_n = f^{opt}_1$ allow four independent task chains to connect together and form a iterative schedule. In a real application, the number of operable frequencies is usually limited. However, there is still a high chance that chains cannot match frequencies with all others. In this case, these chains will be randomly connected with others at the end of the matching process, so that minimum frequency switching overhead can be achieved.

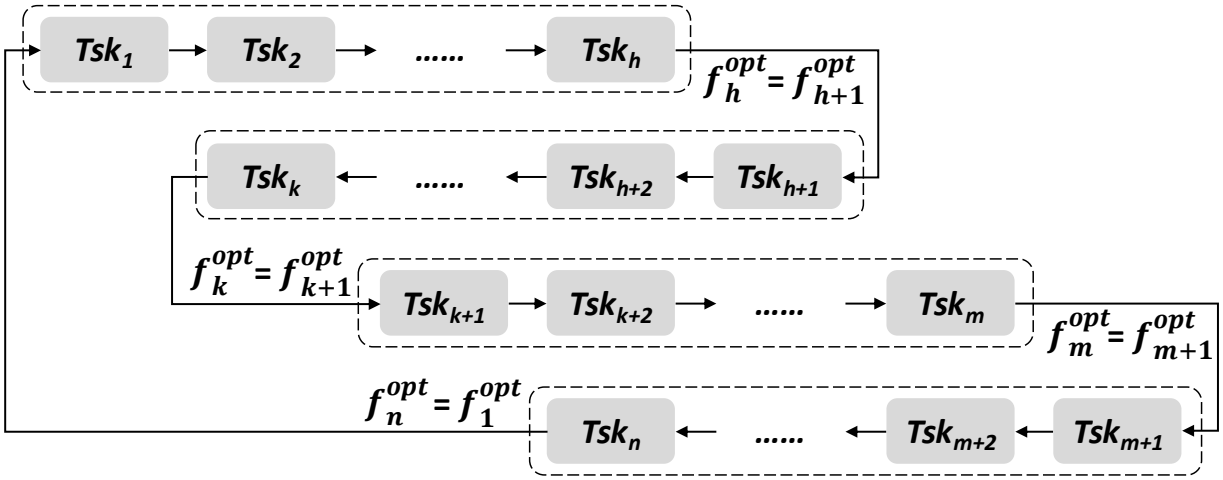


Figure 28: Frequency matching based task schedule

5.3.2 On-line Stage

During the online stage, FM acts as an auxiliary module of the existing scheduler by setting the frequency to the optimal value for each new task during task switching.

5.4 Experiments

5.4.1 Experimental Setup

In this section, the details of the hardware platform, power traces, and benchmarks are described.

5.4.1.1 Hardware Platform The targeted ultra-low-power edge device is TI's ultra-low-power evaluation board with MSP430FR6989, which consists of a 16-bit MCU, a 12-bit ADC, a 2KB volatile SRAM, a 128KB nonvolatile FRAM memory, and different peripherals for sensing and data communication. The power regulator is Bq25570 + LTC3459EDC

which is able to supply a constant voltage of 3.3V to power the edge device and a maximum voltage of 4.2V to charge up the capacitor with the capacitance of $470\mu F$.

5.4.1.2 Power Trace We collect the power traces from Powercast RF energy harvester. Then, we synthesize four power traces in Figure 19 using signal generators for performance evaluation.

The four power traces with different power magnitudes are recorded from the oscilloscope at the sample rate of $240KSa/s$. On average, the power they can provide is 7.27mW, 6.19mW, 4.94mW, and 4.11mW, respectively. The harvesting power is far less than the working power of edge devices.

5.4.1.3 Benchmarks Ten MSP430 benchmarks from Texas Instruments [1] are used for performance evaluation. Among them, four benchmarks 16_bit_2dim, $Floating\_Point\_Math$, $Matrix\_Multiplication$, and Fir_Filter are modified with ADC sampling and FRAM access in order to evaluate the execution speed bottleneck caused by hardware modules. Figure 29 shows the energy requirement of each task when the clock frequency is $4MHz$.

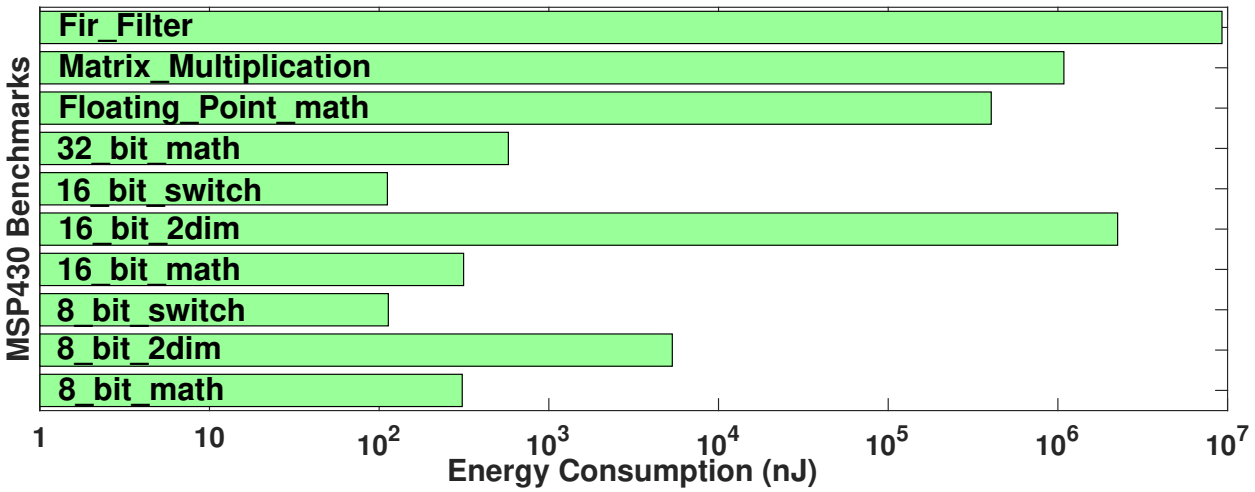


Figure 29: Required energy of each benchmark

To evaluate the performance of Routine Handler, each benchmark is executed repetitively to measure the average execution speed. To measure the performance of Frequency Modulator, all benchmarks are scheduled one after another in a round-robin style.

5.4.2 Energy Utility Evaluation

In this section, the performance of ENZYME is evaluated in terms of the energy utility. The baseline is the ENZYME with the default constant clock frequency of $4MHz$. First, each benchmark is evaluated with the comparison among all available frequencies. The results are shown in Figure 30.

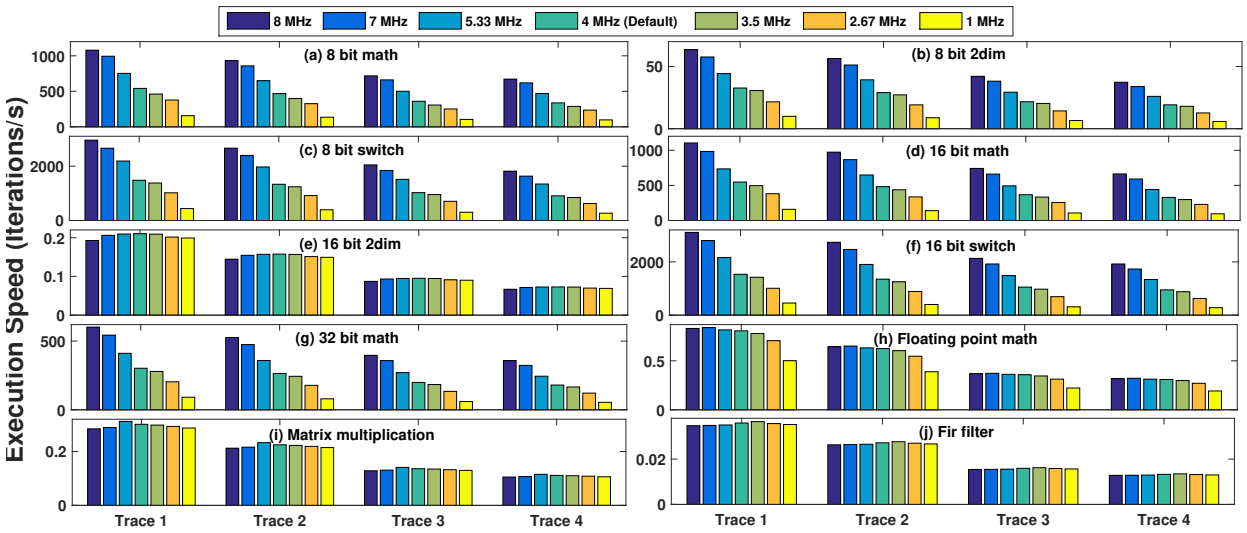
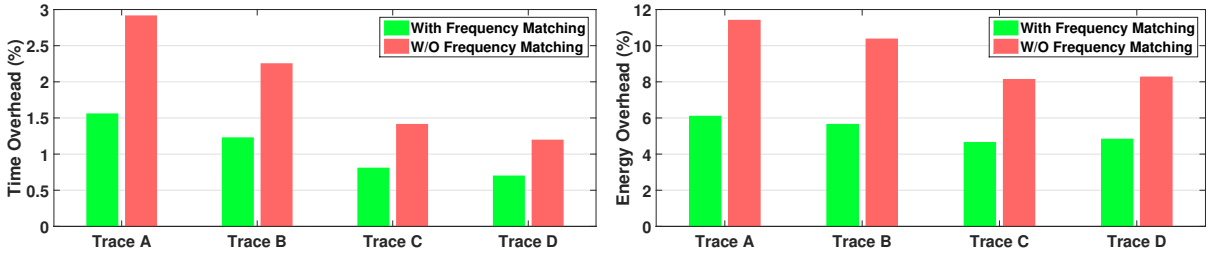


Figure 30: Energy Utility under Available Frequencies

As we can see from Figure 30, given the same amount of energy for execution, with proper selection of clock frequency, the execution can be further speeded up, which means extra energy utility can be gained. For benchmarks (e) 16 bit 2dim, (h) Floating-point math, (i) Matrix multiplication, and (j) Fir filter, due to the access of ADC and FRAM with long latency, a faster clock frequency cannot reduce the run-time delay. Instead, it brings extra energy overhead. Hence, their optimal frequency is lower than the maximum frequency of

$8MHz$. For all other benchmarks, the access only involves register, SRAM, etc. which have short access latency and the maximum clock frequency can deliver the optimal energy utility. After applying Frequency Modulator, the system can achieve 35.71% improvement regarding energy utility which further speeds up the execution.

5.4.2.1 Overhead Analysis In this section, the overhead of frequency modulation is evaluated in terms of energy and time. For ENZYME, the overhead mainly comes from voltage detection and frequency modulation. In our experiments, each voltage detection takes 100 clock cycles and each frequency switch takes 400 clock cycles. When benchmarks are executed in a round-robin style with their optimal frequency, averagely, each voltage detection takes 25.9μs, each frequency switch takes 103.8μs, and voltage detection happens every $5 * 10^4$ clock cycles. In our experiments, we compare the overhead with and without frequency matching technique and the results are shown in Figure 31.



(a) Time Overhead (%)

(b) Energy Overhead (%)

Figure 31: Time and energy overhead

Without frequency matching, the time overhead is 1.94% and energy overhead is 9.54% on average. With frequency matching, the time and energy overhead are reduced to 1.07% and 5.30% respectively. The overhead does not downgrade the performance. Instead, ENZYME can achieve maximum energy efficiency and energy utility for edge devices.

5.5 Summary

This chapter proposes a frequency modulator that can avoid inappropriate runtime CPU clock frequency by adjusting the runtime CPU clock frequency accordingly. The frequency modulator is build upon the observation and comprehensive analysis that different tasks have their own optimal clock frequencies for execution which can yield the maximum energy efficiency. Besides, MCU frequency switch overhead is considered for optimization. The overall performance from the experiments shows the effectiveness of the the proposed frequency modulator for energy utilization.

6.0 Thriving on Ultra-Low Harvesting Power

This chapter proposes a software paradigm, ENZYME, to improve the energy efficiency of edge devices for transient computing with ultra-low energy harvesting power supplies. ENZYME is a lightweight yet highly efficient software module that can maximize power extraction from energy harvesters with proper operation routines. The lightweight and highly efficient natures enable ENZYME to be integrated into ultra-low-power IoT edge devices easily and efficiently.

6.1 Motivation

In this era of the Internet of Things (IoT), all "things" tend to be embedded with electronics, software, sensors, and connectivity in our daily life. The edge of the IoT network, consisting of many interconnected small devices such as wireless sensor nodes, shoulders the responsibility of converting the sensed real-life information into digital representations for IoT. Edge devices usually work under power-constrained scenarios like outdoor environmental monitoring, where energy harvesting technology is preferred, considering the cost and sustainability in the long run. Although the future of implementing energy harvesting system is promising, the weak and transient nature of the harvesting power jeopardizes the data integrity and performance of edge devices. With the help of emerging non-volatile processor (NVP) [55, 31, 57, 10], transient computing [25, 5] becomes practical through checkpointing techniques [16, 53, 42] which ensure data integrity and aggregated progress. When a power outage is imminent, the edge device will conduct checkpointing by backing up all execution state from volatile memory into non-volatile memory (NVM) [33, 39, 37]. Then the edge device is put to sleep mode until the capacitor is charged up. After power recovery, the edge device will wake up and resume execution from the previous checkpoint.

Without appropriate power management, however, edge devices suffer from severe performance degradation during transient computing, especially when the source power is ex-

tremely low. The underlying reason is the low energy efficiency as only a small amount of the harvested energy is used for program execution. Besides, a large amount of the harvested energy is consumed by the supporting hardware and software. The hardware energy consumption is from the power regulator and the energy harvester itself due to the internal resistance. The software energy consumption is from checkpointing and resuming.

Reducing either hardware or software energy overhead can increase energy efficiency. Conventional energy efficiency optimization mainly focuses on reducing hardware energy overhead such as through optimizing regulator [8, 46, 9]. Besides reducing hardware overhead, it is equally important to reduce software overhead of edge devices, especially under ultra-low-power scenarios. This is because, during each duty cycle, the supporting software such as checkpointing tend to consume the same amount of energy. When the harvesting power is ultra-low, the duty cycle of edge devices becomes extremely small. During such an ephemeral active period, software overhead could be gigantic compared with effective energy, which severely undermines the performance of edge devices. Current research regarding such software overhead mainly focuses on reducing the energy consumption of checkpointing. [55, 34, 61, 36]. These methods largely improve the performance of embedded systems.

Orthogonal to reducing hardware or software overhead directly, managing the routine for the edge device is another effective yet rarely explored realm for reducing energy overhead. When harvesting power is ultra-low, the edge device has to periodically checkpoint, sleep, wake up, resume, and execute. These periodical events together make up a routine of transient computing. The key events that define a unique routine are sleep and wakeup events, which are determined by sleep and wakeup voltages. [34] has recently discovered that by dynamically adjusting the wakeup voltage based on the quality of harvesting power, the overall overhead on hardware and software can be minimized. However, to implement the dynamic wakeup strategy, edge devices need to constantly monitor the voltage and analyze the quality of harvesting power. This could cost a tremendous percentage of harvested energy considering its scarcity. Aside from the wakeup event, sleep event also dominates energy efficiency. Therefore, this paper first explores the relationship of wakeup/sleep events and energy efficiency and then proposes a lightweight and highly efficient routine handler to deliver the maximum energy efficiency for edge computing.

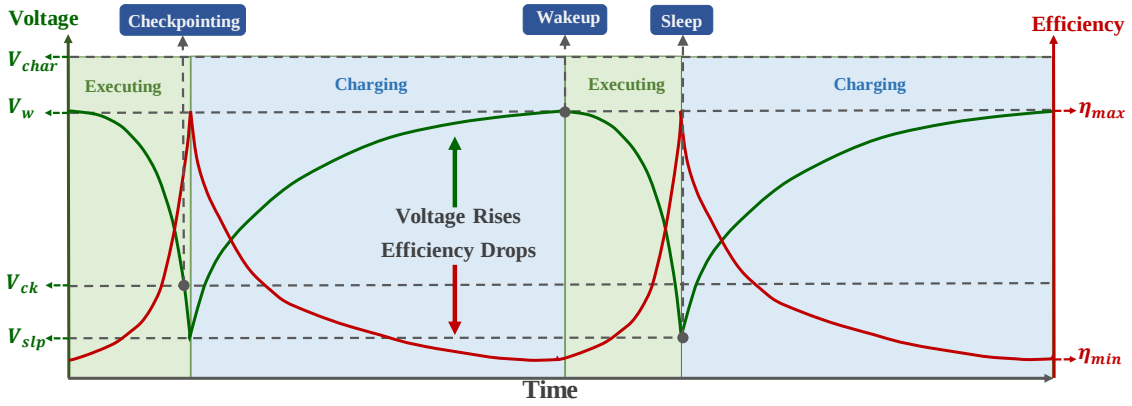


Figure 32: Changes of Charging Efficiency and Voltage on the Edge Device with Ultra-low Harvesting Power

6.1.1 Wake-up Voltage Determines Efficiency

Fig. 32 shows the changes of both the charging efficiency (red line) and the voltage (green line) on a self-powered IoT edge device through the entire executing and charging cycles with ultra-low harvesting power supply. In this figure, the power regulator is assumed to be able to maintain a constant input voltage for the edge device and the harvesting power is far smaller than the working power but higher than the idle power of the edge device.

Initially, when the storage capacitor is charged up to the target voltage V_w, the edge device wakes up and starts to work. Since the harvesting power is smaller than the working power of the edge device, the energy in the storage capacitor drops quickly and so does the voltage on the capacitor. Once the voltage on the capacitor drops to V_{ck}, the checkpointing needs to be conducted which backs up the execution state. After checkpointing, the voltage drops to V_{slp} and the edge device enters the sleep mode with low power consumption so that the harvesting power is sufficient to charge up the storage capacitor. As shown in Fig. 32, for the storage capacitor, there is always a negative correlation between the charging efficiency and the voltage. Therefore, achieving a higher wake-up voltage of V_w is time-consuming with poor energy efficiency.

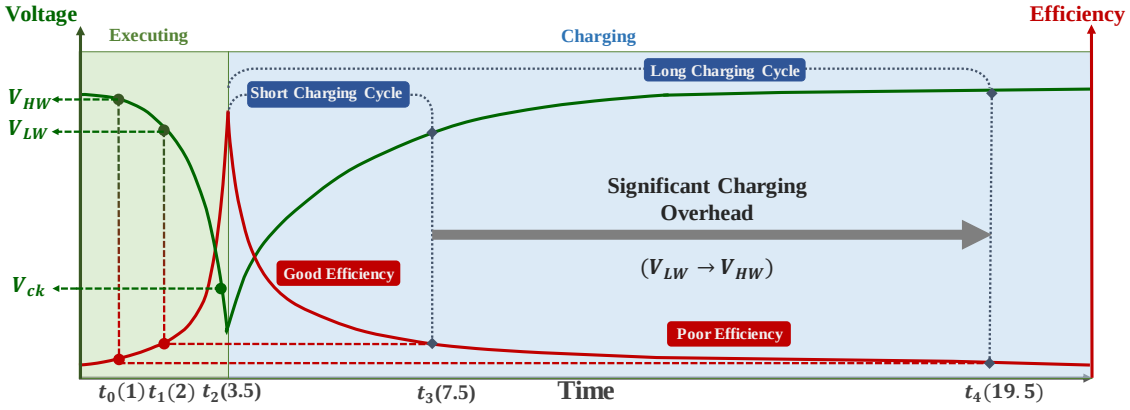


Figure 33: Influence of Wake-up Voltage on Charging Cycle and Efficiency of the Edge Device with Ultra-low Harvesting Power

Fig. 33 shows the influence of wake-up voltage on the duration of the charging cycle and the efficiency of the edge device with ultra-low harvesting power. The harvesting power P_H and working power P_W are assumed to remain constant for simplicity. There are two different wake-up voltages: V_{HW} represents the higher wake-up voltage and V_{LW} represents the lower wake-up voltage.

If the edge device wakes up at V_{LW}, the execution time is $t_2 - t_1 = 1.5$ time units. Then it takes $t_3 - t_2 = 4$ time units to charge the capacitor until it reaches the wake-up voltage V_{LW}. The energy efficiency for the wake-up voltage V_{LW} is $\frac{1.5P_W}{4P_H}$. If the edge device wakes up at V_{HW}, the execution time is $t_2 - t_0 = 2.5$ time units. Then it takes $t_4 - t_2 = 16$ time units to charge the capacitor until it reaches the wake-up voltage V_{HW}. The energy efficiency for wake-up voltage V_{HW} is $\frac{2.5P_W}{16P_H}$. Therefore, although increasing the wake-up voltage up from V_{LW} to V_{HW} leads to one extra time unit of execution time, it takes 12 more time units from t_3 to t_4 to charge up the capacitor from V_{LW} to V_{HW} to wake up the system again. The energy efficiency when waking up at V_{LW} is $\frac{1.5P_W}{4P_H} / \frac{2.5P_W}{16P_H} = 2.4$ times of the energy efficiency when waking up at V_{HW}. Therefore, the wake-up voltage is better to be low. The experimental results in section 6.6 show that with the appropriate setting

of wake-up voltages, the charging efficiency can remain in the good efficiency zoom which significantly improves the efficiency of energy extraction from the ambient environment. The results show that the charging efficiency can be boosted up by 54.03% by solely selecting appropriate wake-up voltages, which is significant for self-powered IoT edge devices with ultra-low energy harvesting supply.

Nevertheless, with a low wake-up voltage, checkpointing becomes more frequent. Fig. 34 shows the influence of the wake-up voltage on the checkpointing frequency.

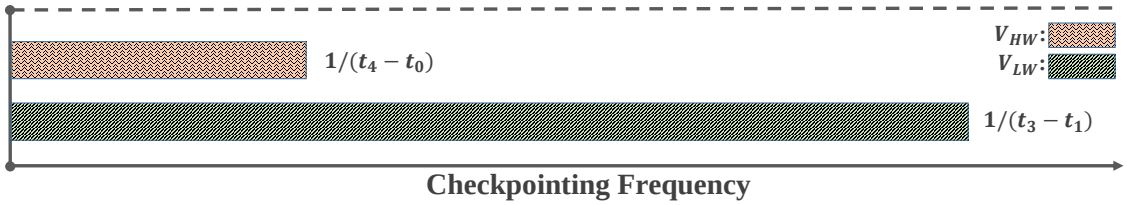


Figure 34: Influence of the Wake-up Voltage on Checkpointing Frequency of the Edge Device with Ultra-low Harvesting Power

Based on Fig. 33, for the wake-up voltage of V_{HW}, the entire executing/charging period is $t_4 - t_0 = 18.5$ time units during which only one checkpointing needs to be conducted. Therefore, the checkpointing frequency can be computed as $f^H = \frac{1}{t_4 - t_0} = \frac{1}{18.5}$. Similarly, for the wake-up voltage of V_{LW}, the checkpointing frequency is $f^L = \frac{1}{t_3 - t_1} = \frac{1}{5.5}$. As f^L is more than three times of f^H, a low wake-up voltage results in a more intensive checkpointing accompanied by a large energy overhead. Therefore, the wake-up voltage should be high enough to maintain a low checkpointing frequency.

From the above analysis, we can see that a conflict exists between the reduction of checkpointing (software) overhead and the improvement of charging (hardware) efficiency. Since the wake-up voltage determines both aspects, an appropriate wake-up voltage which strikes an optimal tradeoff between these two optimization goals should be determined so that the overall energy efficiency can be maximized. With the experimental setup described in Section 6.6.1, Fig. 35 shows how the selection of different wake-up voltages influences the overall energy efficiency. In this figure, more program execution progress means better overall energy efficiency.

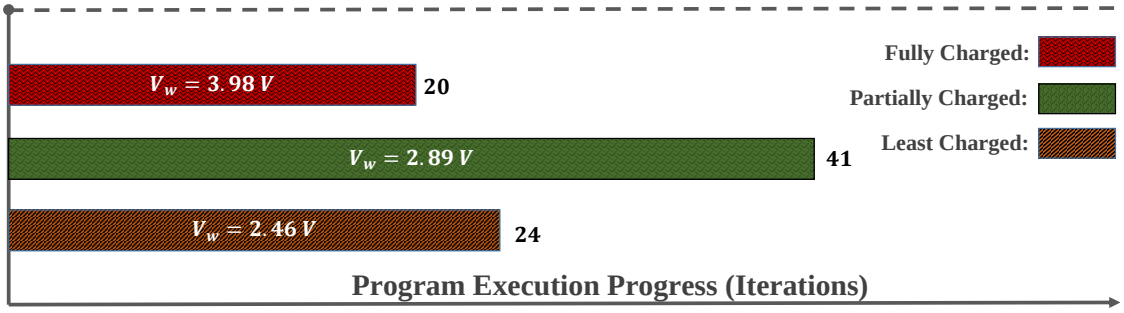


Figure 35: Measurements of the Execution Progress with Different Working Voltages

When the three testing wake-up voltages are 2.46V, 2.89V, and 3.98V, the benchmark program has the corresponding execution progress of 24, 41, and 20 iterations respectively. Therefore, the energy efficiency is low with both the lowest wake-up voltage 2.46V and the highest wake-up voltage 3.98V. When the wake-up voltage is 2.46V, the high checkpointing frequency predominantly affects energy efficiency. When the wake-up voltage is 3.98V, the low charging efficiency predominantly affects energy efficiency. However, if the wake-up voltage is set to be in between the two voltages such as 2.89V in this example, the energy efficiency is significantly increased, which is 41 iterations. Therefore, between 2.46V and 3.98V, there should be an optimal solution which achieves the maximum energy efficiency.

6.1.2 Routines vs. Efficiency

After observation the relationship between wake-up voltage and energy efficiency. It is time to further observe the influence of the voltage selections regarding sleep and wake-up events together (routine) to energy efficiency. The observation is conducted on TI' MSP430FR6989 [3], with an ultra-low source power of $4.11mW$. The average working power of MSP430 benchmark 8_bit_math from [1] is $28.95mW$ which is far greater than $4.11mW$. Hence, transient computing is necessary to achieve progress accumulatively. Here four routines with different sleep and wakeup voltage combinations are created for evaluating the execution speed which can reflect energy efficiency. For each routine, the benchmark is exe-

cuted repetitively to measure the average execution speed, i.e. the number of iterations per second. Higher speed means more energy is extracted from the energy harvester for program execution within the same amount of time. Figure 36 shows the program execution speed corresponding to four routines with four green bars. The sleep voltage and wake up voltage for each routine are labeled in each bar.

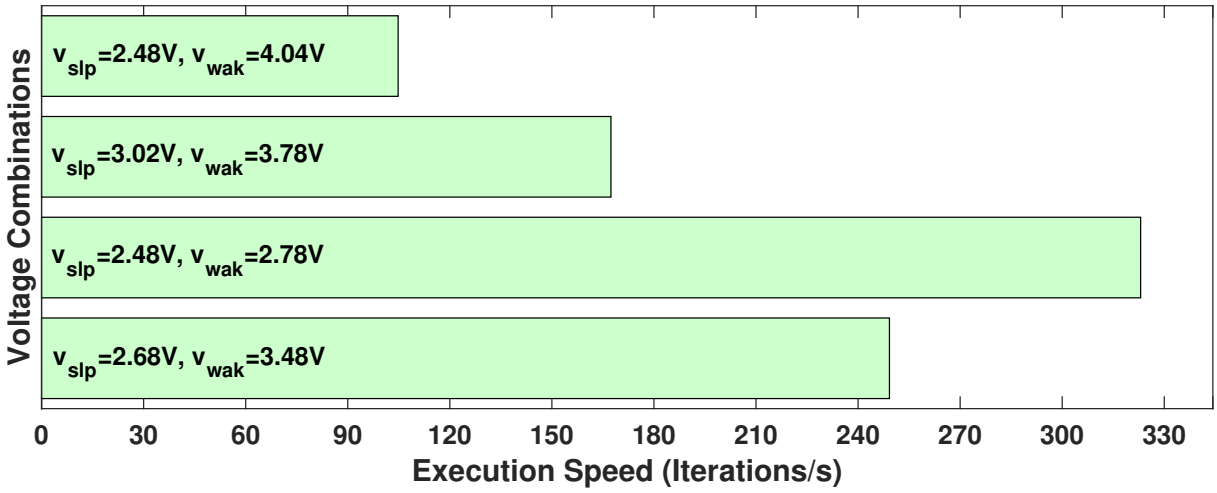


Figure 36: Energy efficiency with different voltage combinations

From Figure 36, different routines result in significant differences in execution speed and thus the energy efficiency. The reason is that different routines cause different charging efficiency for transient computing. For example, although routine 1 and routine 3 have the same sleep voltage of $2.48V$, the energy efficiency of the third routine with wakeup voltage $2.78V$ is almost triple of the first routine which has the wakeup voltage of $4.04V$. The reason is that the charging efficiency of the first routine is significantly smaller than the third routine due to the high wakeup voltage, hence, the third routine enables a faster execution speed. Therefore, it is crucial to devise the edge computing routine appropriately.

6.2 Related Work

For self-powered edge devices, the harvested energy, other than being solely consumed by the edge device itself, is also consumed by the power regulator resulting in significant hardware energy overhead. A considerable amount of research has been conducted for reducing the hardware overhead by improving the regulator efficiency through impedance matching techniques. [8] proposes effective switching frequency technique for voltage converter to deliver maximum output power. [46] conducts circuit-level design which enables the regulator to extract power from multiple low-power energy harvesting sources with maximum efficiency. [9] proposes a duty cycle based impedance adjustment technique for the maximum power extraction from a thermoelectric energy source without sacrificing power conversion efficiency. what's more [51, 49, 50] propose through-silicon-via inductors which can be used by energy harvesting circuits with minimum footprint. Further, [63] proposes a run-time simulation framework of both power delivery and architecture and captures their interactions for energy efficiency optimization.

Aside from hardware overhead, software overhead such as checkpointing and resuming also degrades the performance of edge devices. The situation will become even worse when harvesting power is extremely small as the percentage of software overhead will be amplified. Current research regarding software overhead mainly focuses on reducing the energy consumption of checkpointing. [55] proposes a checkpoint aware instruction scheduling algorithm to reduce the writes to nonvolatile registers. [34] proposes a priority-based task scheduling which prioritizes the execution of tasks with less checkpointing content to lower down checkpointing overhead. [61] observes the runtime stack size variation and inserts checkpoints for a long program when its stack size is small so that checkpointing overhead can be reduced. These methods largely improve the performance of embedded systems.

Orthogonal to reducing overhead directly, managing the routine for the embedded system is another effective yet rarely explored realm for reducing energy overhead. There are only a few research work that has explored this topic. [36] proves that the energy harvesting powered embedded system is better to be put to sleep mode instead of being shut down completely upon power outage. After exploring the sleep event, [7] observes that when harvesting power

is getting better, it is unnecessary for the system to stay, if any, in sleep mode. Based on this idea, a tentative wakeup strategy is proposed which allows the self-powered embedded system to temporarily wake itself up, evaluate the quality of harvesting power, and determine whether the system can wake up earlier. In this way, the charging efficiency can be improved. This method is effective when a power outage happens sporadically. However, with ultra-low harvesting power, checkpointing could happen intensively which incurs extra software overhead. After realizing such a problem, [34] further discovers that checkpointing overhead is in a negative correlation to the charging efficiency while both subjects are determined by wakeup voltages.

However, when the harvesting power becomes extremely low, the aforementioned techniques may not achieve desirable performance. There are mainly two reasons. First, the harvesting power, most of the time, should be far less than the active power of an edge device. In this case, tentative wakeup would further lower down the energy efficiency for execution. Second, to implement a dynamic wakeup strategy, an edge device needs to constantly monitor the voltage and analyze the quality of source power. This could cost a tremendous amount of energy considering the scarcity of harvested energy. Therefore, this paper addresses the aforementioned issues and proposes a simple yet highly efficient routine handler to deliver the maximum energy efficiency for edge computing.

Even though overhead can be minimized with the aforementioned techniques, inappropriate execution frequencies can significantly reduce the runtime energy utility. Previous research on DVFS-based techniques [58, 12, 29] mainly focus on how to lower down the execution frequency appropriately so that the power consumption of the embedded system can be reduced. However, low execution frequency doesn't necessarily guarantee high energy utility. Besides, accessing hardware with a large latency such as non-volatile memory or GPIO can become be the bottleneck of the actual execution speed. Therefore, it is necessary to optimize the execution frequency accordingly for each program. Therefore, aside from improving energy efficiency, this paper also focuses on improving energy utility through runtime frequency modulation.

6.2.1 System Architecture

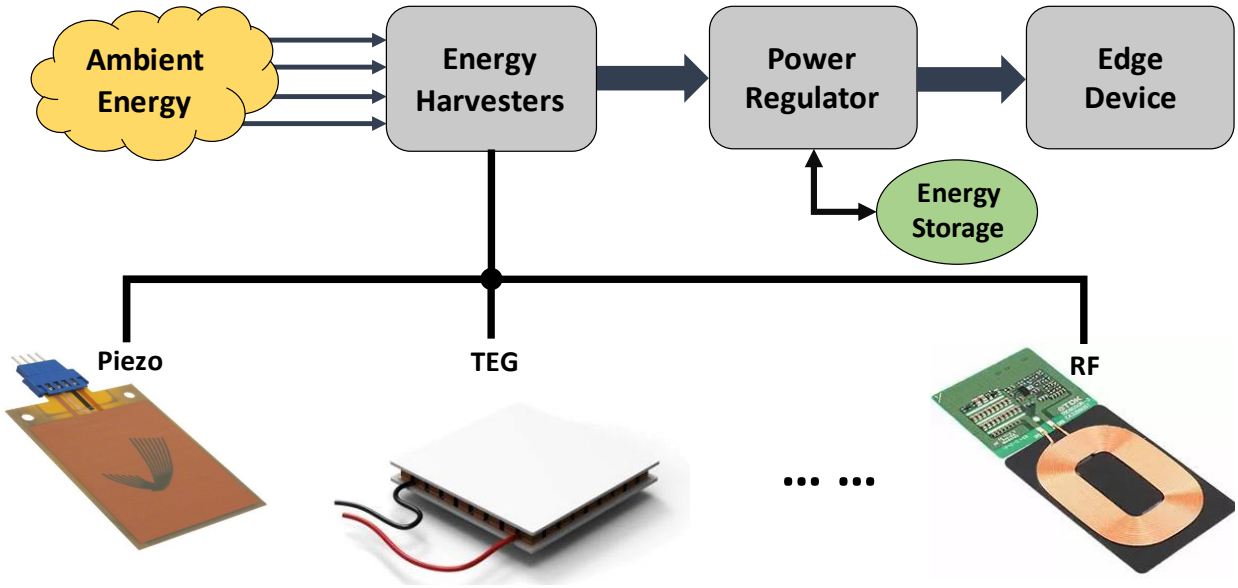


Figure 37: Architecture of self-powered edge device

Figure 37 shows a typical architecture of self-powered edge devices which consists of energy harvesters, power regulator, and storage capacitor. The energy harvesters harvest ambient energy and convert it into electrical energy. Then the power regulator harnesses the electrical energy and provides targeting voltage for charging the storage capacitor or directly powering the edge devices.

6.3 Modeling and Analysis for $\xi(v_{Wak})$

6.3.0.1 Hardware Energy Efficiency The charging circuit of the self-powered IoT edge device can be simplified as shown in Fig. 38.

In this figure, the storage capacitor has a time constant $R_C C$ where generally $R_C \ll R_N$ and the voltage of this capacitor is v. The energy harvester and the power regulator together

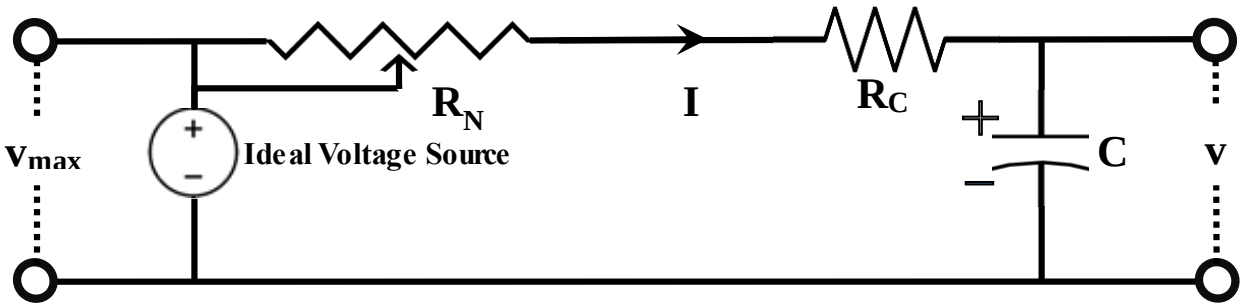


Figure 38: Capacitor Charging Circuit

can be considered as an ideal voltage source which connects the inner resistance R_N. The ideal voltage source maintains a constant voltage which equals the rated output voltage v_{max} of the regulator. R_N is not a constant and it becomes smaller when the harvesting power P_H increases (source power becomes stronger) and vice versa. To calculate the hardware energy efficiency ξ^H, Theorem 1 is proposed.

Theorem 1. *Given $P_H \ll P_W$ and τ as the charging period, the hardware energy efficiency ξ^H can be formulated in Eq.* (6.1)*,*

$$\xi^H = E_{cap}(V)/E_H(\tau) \tag{6.1}$$

where

$$\begin{cases} E_{cap}(v) = C(v^2 - v_{min}^2)/2 \\ E_H(\tau) = \int_0^\tau P_H(t)dt \end{cases} \tag{6.2}$$

.

Proof. Assume that the actual hardware energy efficiency is ξ_a^H which is defined as the amount of harvested energy over the maximum amount of energy that can be harvested during the entire execution and charging cycles. Therefore, ξ_a^H can be formulated in Eq. (6.3)

$$\xi_a^H = \frac{E_{cap}^\tau(v)}{E_H(\tau) + E_H(\gamma)} \tag{6.3}$$

Here, τ is the charging period and γ is the working period. Considering P_H and P_W, Eq. (6.3) can be reformulated as Eq. (6.4)

$$\xi_a^H = \frac{\overline{P_W}\gamma/\eta}{\overline{P_H}(\tau+\gamma)} \tag{6.4}$$

Where $\overline{P_H}$ is the average harvesting power, $\overline{P_W}$ is the average working power of the edge device, and η is the efficiency of the regulator to utilize the stored energy for execution. Assuming $\overline{P_W} = \Psi\overline{P_H}$, since $\xi_a^H < 1$ and $\eta < 1$, we have Eq. (6.5)

$$\frac{\gamma}{\tau} = \frac{\xi_a^H \eta}{\Psi - \xi_o^H \eta} < \Psi^{-1} \tag{6.5}$$

Therefore, we approximate the hardware energy efficiency as Eq. (6.6).

$$\xi^H = \frac{E_{cap}^{\tau}(v)}{E_H(\tau)} = \frac{\overline{P_W}\gamma/\eta}{\overline{P_H}\tau} \tag{6.6}$$

And the error ϵ of the approximation can be calculated in Eq. (6.7).

$$\epsilon = \frac{\xi^H - \xi_a^H}{\xi_a^H} = \frac{\gamma}{\tau} \tag{6.7}$$

Based on the assumption $P_w \gg P_H$, we have $\Psi^{-1} \to 0$. Therefore, $\epsilon \to 0$. We further analyze the influence of Ψ in the experiments. This completes the proof of Theorem 1. □

Theorem 1 explores the influence of the relationship between P_W and P_H to the accuracy of the formulation.

In Eq. (6.2), v_{min} represents the minimum voltage on the capacitor for the regulator to maintain the rated output voltage for the edge device. Since $E_{cap}(v)$ and $E_H(\tau)$ are with different independent variables v and τ, a further transformation is required. First Eq. (6.8) shows the calculation of P_H.

$$P_H(t) = v_{max}^2 / R_N(t) \tag{6.8}$$

Then, according to Kirchhoff Voltage Law, the parameters in Fig. 38 have the following relation:

$$v_{max} - v = C\frac{dv}{d\tau}(R_N + R_C) \tag{6.9}$$

Since $R_N \gg R_C$, R_C can be omitted in Eq. (6.9). Therefore, based on Eq. (6.8), we further have

$$v_{max} - v = C\frac{dv}{d\tau}\frac{v_{max}^2}{P_H} \tag{6.10}$$

Hence, $E_H(\tau)$ can be further transformed into Eq. (6.11).

$$E_H(\tau) = Cv_{max}^2\left[\ln\left(\frac{v_{max} - v_{min}}{v_{max} - v}\right)\right] \tag{6.11}$$

Considering Eq. (6.2) and Eq. (6.11) together, the hardware energy efficiency ξ^H in Eq. (6.1) can be further transformed into the function of v in Eq. (6.12).

$$\xi^H(v) = \frac{(v^2 - v_{min}^2)/2}{v_{max}^2\left[\ln(v_{max} - v_{min}) - \ln(v_{max} - v)\right]} \tag{6.12}$$

6.3.0.2 Software Energy Efficiency Based on the aforementioned analysis, the software energy efficiency ξ^S can be calculated as in Eq. (6.13).

$$\xi^S(v, \omega) = (E_{cap}(v) - E_{ck}(\omega) - E_{wak})/E_{cap}(v) \tag{6.13}$$

6.3.0.3 Execution Energy Efficiency Considering ξ^H and ξ^S together, the overall energy efficiency of program execution can be calculated as in Eq. (6.14)

$$\xi(v, \omega) = \frac{(v_{exe}^2 - v_{ck}^2)/2}{v_{max}^2\left[\ln(v_{max} - v_{min}) - \ln(v_{max} - v)\right]} \tag{6.14}$$

where,

$$\begin{cases} v_{ck} = \sqrt{2E_{ck}^{(1)}(\omega)/C + v_{min}^2} \\ v_{exe} = \sqrt{v^2 - 2E_{wak}/C} \end{cases} \tag{6.15}$$

Here, v_{ck} represents the voltage for starting checkpointing and v_{exe} represents the voltage for starting program execution.With Eq. (6.14), we can analyze the influences of both ω and v on the energy efficiency ξ.

6.3.0.4 Influence of Checkpointing Data Size ω Take a derivative of ξ with respect to ω, then we have Eq. (6.16).

$$\frac{\partial \xi(v,\omega)}{\partial \omega} = \frac{-(E_{ck}^{(1)} + E_{rs}^{(1)})/C}{v_{max}^2 \left[\ln(v_{max} - v_{min}) - \ln(v_{max} - v)\right]} < 0 \tag{6.16}$$

As we can see from Eq. (6.16), ω should be as small as possible in order to maximize the execution energy efficiency.

6.3.0.5 Influence of Wake-up Voltage v Under the circumstance that $P_w \gg P_H$, if we define v_{op} as the optimal wake-up voltage that maximizes the execution energy efficiency ξ, then we have the following properties.

Theorem 2. *The optimal solution v_{op} exists.*

Proof. Take a derivative of ξ with respect to v and we have Eq. (6.17)

$$\frac{\partial \xi(v,\omega)}{\partial v} = \frac{G(v) - F(v)}{M(v)} \tag{6.17}$$

where, $G(v)$, $F(v)$, and $M(v)$ are

$$\begin{cases} G(v) = v v_{max}^2 \ln(\dfrac{v_{max} - v_{min}}{v_{max} - v}) \\ F(v) = \left[(v_{exe}^2 - v_{ck}^2)/2\right] v_{max}^2/(v_{max} - v) \\ M(v) = \left[v_{max}^2 \ln(\dfrac{v_{max} - v_{min}}{v_{max} - v})\right]^2 \end{cases} \tag{6.18}$$

When the edge device starts program execution, the stored energy should be at least enough for checkpointing and waking up the edge device, therefore, v should satisfy Eq. (6.19)

$$\begin{cases} v \geq v_{ini} \\ v_{ini} = \sqrt{v_{ck}^2 + 2E_{wak}/C} \end{cases} \tag{6.19}$$

Also, because $v < v_{max}$, we have Eq. (6.20).

$$v_{ini} \leq v < v_{max} \tag{6.20}$$

Initially based on Eq. (6.15), $v = v_{ini} \Rightarrow G(v_{ini}) > F(v_{ini}) = 0$. In this case,

$$\frac{\partial \xi}{\partial v}|_{v=v_{ini}} > 0 \tag{6.21}$$

Since $\lim_{v \to v_{max}^-} \tau(v) = +\infty$, we have

$$\begin{cases} \lim_{v \to v_{max}^-} G(v) = +\infty \\ \lim_{v \to v_{max}^-} F(v) = +\infty \end{cases} \tag{6.22}$$

Therefore, to calculate $\frac{\partial \xi}{\partial v}|_{v \to v_{max}^-}$, we need to know the second derivative of both $G(v)$ and $F(v)$ with respect to v. Since any sudden loss or burst of P_H can be flattened by the storage capacitor, $P_H(\tau)$ can be considered irrelevant to τ. Given Eq. (6.18), we have Eq. (6.23).

$$\begin{cases} G(v)' = v_{max}^2 \left[\ln(\frac{v_{max} - v_{min}}{v_{max} - v}) + \frac{v}{v_{max} - v} \right] \\ F(v)' = v_{max}^2 \left[\frac{v}{v_{max} - v} + \frac{(v_{exe}^2 - v_{ck}^2)/2}{(v_{max} - v)^2} \right] \end{cases} \tag{6.23}$$

Since $G(v)' = o(F(v)')|_{v \to v_{max}^-}$, we further need to employ the *L'Hôpital's* rule which leads to the result shown in Eq. (6.24).

$$\lim_{v \to v_{max}^-} \frac{G(v)}{F(v)} = \lim_{v \to v_{max}^-} \frac{G(v)'}{F(v)'} = 0 \tag{6.24}$$

Further referring to Eq. (6.22), we have

$$\frac{\partial \xi}{\partial v}|_{v \to v_{max}^-} = -\infty \tag{6.25}$$

Hence, considering the Eq. (6.21) and (6.25), there should be at least one global maximum solution for ξ that satisfies

$$\frac{\partial \xi}{\partial v}|_{v=v_{op}} = 0 \qquad (v_{ck} \leq v_{op} < v_{max}) \tag{6.26}$$

This completes the proof of Theorem 2. □

Theorem 3. *For a given ω, v_{op} is unique.*

Proof. Since ξ, aside from wake-up voltage v, is also determined by the checkpointing data size ω. The changes of ω will affect the value of v_{op}. For a given ω, let's define $K(v) = G(v) - F(v)$, so the first derivative of $K(v)$ with respect to v is

$$K'(v) = v_{max}^2 \left[\ln(\frac{v_{max} - v_{min}}{v_{max} - v}) - \frac{(v_{exe}^2 - v_{ck}^2)/2}{(v_{max} - v)^2} \right] \tag{6.27}$$

Further, we have the second derivative of ξ with respect to v as

$$\frac{\partial^2 \xi}{\partial v^2} = \frac{K'(v)M(v) - K(v)M'(v)}{M(v)^2} \tag{6.28}$$

For optimal solution, we have $K(v_{op}) = 0$, hence

$$\frac{\partial^2 \xi}{\partial v^2}|_{v=v_{op}} = \frac{K'(v_{op})}{M(v)} \tag{6.29}$$

For $K'(v_{op})$, since $F(v_{op}) = G(v_{op})$, based on Eq. (6.18) and (6.27), we have

$$K'(v_{op}) = v_{max}^2 \left[\frac{(v_{max} - 2v_{op})(v_{op}^2 - v_{ini}^2)/2}{(v_{max} - v_{op})^2} \right] \tag{6.30}$$

Assume that there are multiple optimal solutions, since

$$\begin{cases} \frac{\partial \xi}{\partial v}|_{v=v_{ini}} > 0 \\ \frac{\partial \xi}{\partial v}|_{v \to v_{max}^-} < 0 \end{cases}$$

for the smallest local maximum solution v_{op1} that satisfies $\frac{\partial \xi}{\partial v}|_{v=v_{op1}} = 0$, it should also satisfy $\frac{\partial^2 \xi}{\partial v^2}|_{v=v_{op1}} < 0$. Since $M(v) > 0$, based on Eq. (6.29), we have

$$K'(v_{op1}) < 0 \tag{6.31}$$

Bring Eq. (6.31) into Eq. (6.30), we have $v_{op1} > v_{max}/2$. Since v_{op1} is the smallest local maximum solution, we have

$$\forall v_{op} > v_{op1} : K'(v_{op}) < 0 \Rightarrow \frac{\partial^2 \xi}{\partial v^2}|_{v=v_{op}} < 0 \tag{6.32}$$

Hence, all optimal solutions should be maximal. However, as ξ is continuous on v, if there are more than one solutions for $\xi' = 0$, the maximum and minimum solutions should appear alternatively. However, Eq. (6.32) means that all solutions are maximum, which is impossible unless there is only one solution v_{op1} which is also the global maximum for ξ.

This completes the proof of Theorem 3. □

6.4 Modeling and Analysis for $\xi(v_{Wak}, v_{slp})$

This section provides energy modeling and evaluates the influence of v_{wak} and v_{slp} together to the energy efficiency ξ. First, the architecture of a general self-powered edge device is given. Then, an energy consumption model for self-powered IoT devices is proposed. Based on this energy model, a thorough analysis is conducted regarding achieving maximum energy efficiency.

6.4.1 Energy Modeling

This subsection models the energy distribution of the harvested energy during a complete transient computing cycle. A cycle consists of four periods including charging τ_{ch}, resuming τ_{rs}, execution τ_{ex}, and checkpointing τ_{ck}. During a complete transient computing cycle, the harvested energy is distributed into hardware overhead, software overhead, and effective energy.

6.4.1.1 Harvested Energy Given the harvesting power $P_H(t)$, the four routine periods, then the harvested energy $E_H(\tau)$ can be defined as

$$E_H = \tau \overline{P_H(t)} \tag{6.33}$$

where $\tau = \tau_{ch} + \tau_{rs} + \tau_{ex} + \tau_{ck}$ represents a transient computing cycle and $\overline{P_H(t)}$ represents the average harvesting power during a complete transient computing cycle.

6.4.1.2 Hardware Energy Overhead On the hardware aspect, the harvested energy is consumed by the edge device and supporting hardware including the power regulator and the energy harvesters themselves. To find out hardware energy overhead E_{ho}, the power regulator and the energy harvester can be considered as a whole. Eq. (6.34) shows the definition of E_{ho}.

$$E_{ho} = E_H - E_{ed} \tag{6.34}$$

where E_{ed} represents the energy consumption of the edge device in a transient computing cycle. Given the efficiency η of the regulator and harvester, wakeup voltage v_{wak}, sleep voltage v_{slp}, and capacitance C, E_{ed} can be formulated as

$$E_{ed} = \eta \left\{ \psi \overline{P_H^{\psi}(t)} + \tau_{ch} \overline{P_C} \right\} \tag{6.35}$$

where $\overline{P_C} = \frac{C}{2\tau_{ch}}(v_{wak}^2 - v_{slp}^2)$ represents the average charging power of the capacitor during the charging period, $\psi = \tau_{re} + \tau_{rs} + \tau_{ex} + \tau_{ck}$ represents the active period, and $\overline{P_H^{\psi}(t)}$ represents the average harvesting power in active period. With Eq. (6.35), E_{ho} in Eq. (6.34) can be transformed into Eq. (6.36)

$$E_{ho} = (1-\eta)E_H + \eta \left\{ \tau_{ch} \overline{P_H^{\tau_{ch}}(t)} - \tau_{ch} \overline{P_C} \right\} \tag{6.36}$$

where $\overline{P_H^{\tau_{ch}}(t)}$ represents the average harvesting power in the charging period when edge device is in sleep mode.

6.4.1.3 Software Overhead The software overhead includes checkpointing overhead and resuming overhead. Among them, the first two are determined by the checkpointing data size ω. Given the energy consumption for checkpointing and resuming a data unit as $E_{ck}^{(1)}$ and $E_{rs}^{(1)}$ respectively, then energy for checkpointing ($E_{ck}(\omega)$) and resuming ($E_{rs}(\omega)$) can be defined as

$$\begin{cases} E_{ck}(\omega) = \omega E_{ck}^{(1)} \\ E_{rs}(\omega) = \omega E_{rs}^{(1)} \end{cases} \tag{6.37}$$

With the aforementioned definitions, the total software overhead E_{so} can be formulated as

$$E_{so} = E_{ck} + E_{rs} \tag{6.38}$$

Notice that, E_{so} is irrelevant to time, hence, lowering the frequency of checkpointing can reduce software overhead.

6.4.1.4 Effective Energy

The effective energy E_{ee} during a transient computing cycle is the part of the energy that is consumed for execution. Effective energy is provided both by the storage capacitor and the energy harvester. As the harvested energy is consumed in three ways (hardware overhead, software overhead, and effective energy), based on the previous modeling (6.34), the effective energy E_{ee} can be calculated as

$$E_{ee} = E_{ed} - E_{so} \tag{6.39}$$

Further taking eq. (6.35) into (6.39), we have

$$E_{ee} = \eta \left(\psi \overline{P_H^{\psi}(t)} + \tau_{ch} \overline{P_C} \right) - E_{so} \tag{6.40}$$

6.4.2 Efficiency Analysis

The purpose of this section is to explore strategies for maximizing the percentage of effective energy in the harvested energy. This percentage is defined as energy efficiency ξ.

6.4.2.1 Formulation of ξ

Based on Eq. (6.33) and (6.40), ξ can be formulated as

$$\xi = E_{ee}/E_H = \eta\chi \frac{\overline{P_H^{\psi}}}{\overline{P_H}} + \eta(1-\chi)\frac{\overline{P_C}}{\overline{P_H}} - \frac{E_{so}\chi}{\overline{P_H}\psi} \tag{6.41}$$

Here, $\chi = \psi/(\tau_{ch} + \psi)$ represents the duty cycle. Given the average working power of edge device during active mode as $\overline{P_W}$, we have

$$\eta(\psi \overline{P_H^{\psi}} + \tau_{ch}\overline{P_C}) = \psi \overline{P_W} \tag{6.42}$$

Hence, the duty cycle χ can be transformed into

$$\chi = \frac{\eta \overline{P_C}}{\overline{P_W} + \eta(\overline{P_C} - \overline{P_H^{\psi}})} \tag{6.43}$$

Under ultra-low harvesting power scenario, the harvesting power P_H is large enough to charge up the capacitor but far smaller than the working power P_W. In this case, we have $P_C < P_H \ll P_W$ and eq. (6.41) can be simplified as

$$\xi = \frac{\eta\overline{P_C} - E_{so}/\tau_{ch}}{\overline{P_H}} \tag{6.44}$$

From this equation, $\xi(\overline{P_C}, \tau_{ch})$ is a monotonically increasing function with respect to both $\overline{P_C}$ and τ_{ch}. Since $\overline{P_H}$ is uncontrollable and independent from other parameters, the optimization goal is

$$Maximize \qquad \zeta = \eta\overline{P_C} - E_{so}/\tau_{ch} \tag{6.45}$$

where $\overline{P_C}$ and τ_{ch} are tunable by edge devices. It's worth noting that, for the prerequisite condition $P_H \ll P_W$, P_H does not need to be infinitesimally smaller than P_H in order to make eq. (6.44) plausible. In fact, from the experimental observation, when $P_H \leq P_W/5$, the theoretic modeling turns out to be fairly effective, which will be detailed in section 6.6.

6.4.2.2 Optimization of ζ The equivalent charging circuit of the energy harvesting system can be simplified as shown in Figure 38. With such modeling, $\overline{P_C}$ and τ_{ch} can be considered as functions of both v_{slp} and v_{wak}.

$$\begin{cases} \overline{P_C} = C(v_{wak}^2 - v_{slp}^2)/(2\tau_{ch}) \\ \tau_{ch} = R_N C \ln\left(\dfrac{v_{max} - v_{slp}}{v_{max} - v_{wak}}\right) \end{cases} \tag{6.46}$$

With Eq. (6.46), ζ can be considered as a binary function of variable v_{slp} and v_{wak}. As long as $v_{wak} > v_{slp}$ and $E_{so} < \eta\tau_{ch}\overline{P_C}$ are satisfied, the unique optimal solution which is also the global maximum can be guaranteed.

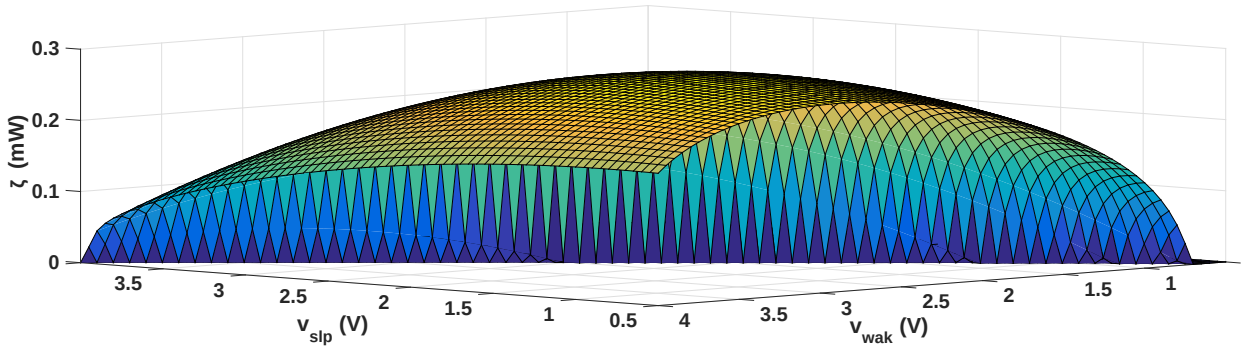


Figure 39: Concavity of ζ in respect of v_{slp} and v_{wak}

6.4.2.3 Validation of modeling We conduct both theoretical and experimental studies to validate the proposed modeling. First, we conduct a theoretical case study of $\zeta(v_{slp}, v_{wak})$. Given $v_{max} = 4.2V$, $R_N = 10K\Omega$, $C = 47\mu F$, $\eta = 0.7$, and $E_{so} = 4.15\mu J$, the graph of $\zeta(v_{slp}, v_{wak})$ can be observed in Figure 39. In this figure, ζ is an concave function in respect of both v_{slp} and v_{wak}. Therefore, for each v_{slp}, the optimal ζ is unique.

Figure 40 shows the maximum ζ and optimal v_{wak} for each given v_{slp}. From this figure, we observe that different V_{slp} yield different optimal V_{wak}. When $v_{slp} = v_{slp}^{opt} = 1.6V$, ζ achieves the global maximum of $0.285mW$ with the optimal $v_{wak}^{opt} = 2.75V$. Since different power regulators may have different v_{min}, $v_{slp} = \max\{v_{min}, v_{slp}^{opt}\}$ is able to yield the maximum ζ.

Further, we compare experimental results from a real-test bed (MSP430FR6989) with results from a theoretic study to validate the modeling, which is shown in Figure 41. The hardware parameters are shown in section 6.6.

For both theoretical and experimental evaluation, seven available sleep voltages are chosen. For each V_{slp}, we incrementally change V_{wak} and measure the execution speed (energy efficiency). The left figure shows the theoretical results and the right figure shows the experimental results. Due to the hardware constraints, the v_{slp} should be no less than $2.34V$. As we can see, left and right have very similar trends and distribution. This validates the effectiveness of the proposed modeling.

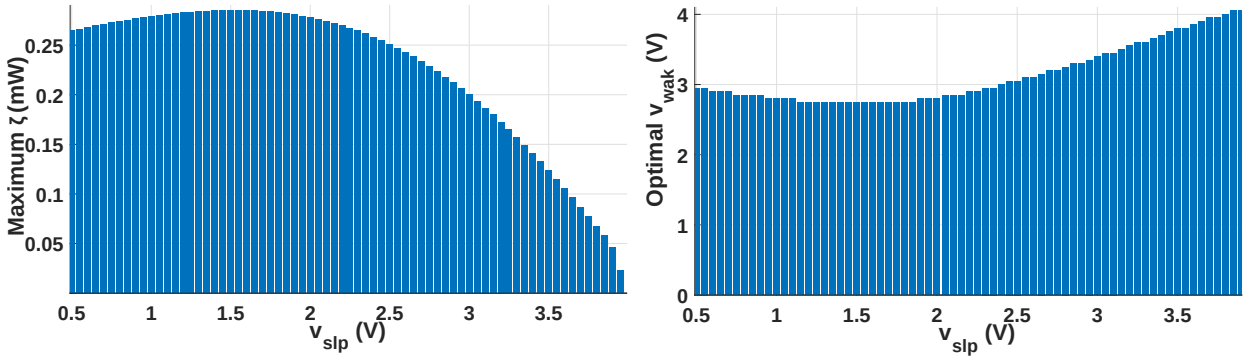


Figure 40: Optimal ζ and v_{wak} with given v_{slp}

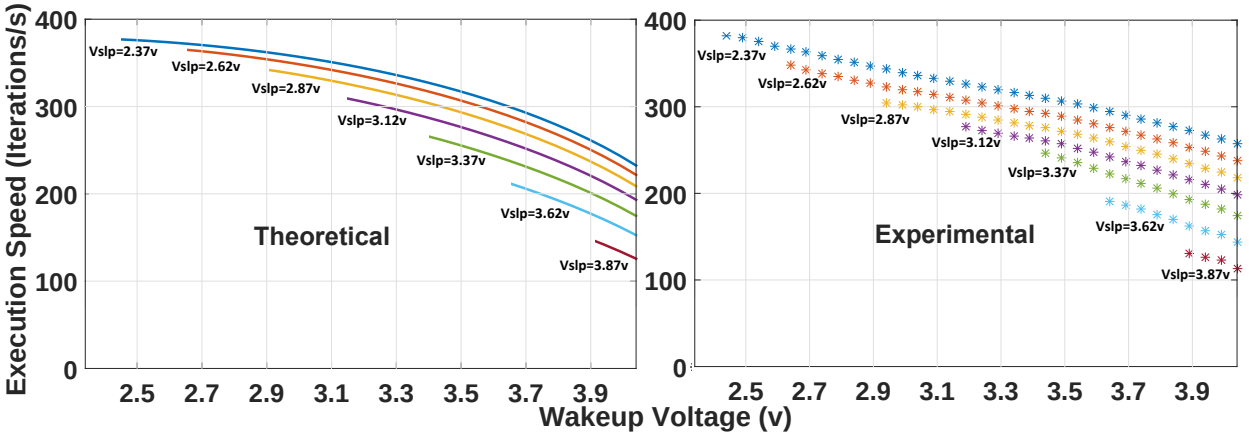


Figure 41: Execution speed (theoretical vs experimental)

6.5 Optimizing Voltages for Sleep/Wakeup

In this section, an energy-efficient routine management scheme is proposed to maximize the execution progress of ultra-low self-powered IoT edge devices.

From the previous energy modeling analysis, for a given sleep voltage, there exists an optimal wake-up voltage; for different pairs of sleep and wakeup voltages, there exists the best pair that generates the highest energy efficiency. Therefore, the voltage profile is collected offline which stores the optimal wake-up voltage concerning different sleep voltages. This voltage profile is referenced during online execution for designing the routine handler. Besides, for different tasks, there exists an optimal clock frequency to achieve the highest energy efficiency. Therefore, the frequency profile is also collected offline which stores the optimal clock frequency for each task. This frequency profile is referenced during online execution by the frequency modulator.

Routine Handler is in charge of selecting the best sleep voltage v_{slp} and wakeup voltage v_{wak} so that the maximum energy efficiency can be achieved for task execution. With optimized routines, the Frequency Modulator further maximizes energy utility by setting the optimal frequency for each task during the runtime execution.

6.5.1 Routine Handler

In this section, a routine handler is designed which manages the routine events of edge devices properly during transient computing, so that the maximum energy efficiency can be achieved for program execution. Before moving to the details of Routine Handler, it is necessary to find the optimal voltage set $\langle v_{slp}, v_{wak} \rangle$ for wakeup and sleep events which is shown in Algorithm 6.5.1.

The inputs of Algorithm 6.5.1 include all possible voltages $\langle v \rangle$, the maximum voltage v_{max}, and the minimum voltage v_{min} for power regulator to remain functional. The output is the optimal voltage set $\langle v_{slp}, v_{wak} \rangle$ for wakeup and sleep events.

Initially, voltages in $\langle v \rangle$ are filtered to satisfy $v_{min} \leq v \leq v_{max}$, which is shown in line 3. Then, from line 5 to 13, ζ is measured with all possible combinations of v_{slp} and v_{wak}. Here

Algorithm 6.5.1 Measure Optimal $\langle v_{slp}, v_{wak} \rangle$

Input: v_{max}, v_{min}, and $\langle v \rangle$
Output: optimal $\langle v_{slp}, v_{wak} \rangle$ set
1: **if** $\min \langle v \rangle < v_{min}$ or $\max \langle v \rangle > v_{max}$ **then**
2: remove any v which satisfies;
3: $v < v_{min} || v > v_{max}$;
4: **end if**
5: **for** each $v_i \in \langle v \rangle$ **do**
6: **for** each $v_j \in \langle v \rangle$ **do**
7: **if** $v_j > v_i$ **then**
8: measure $\zeta(v_i, v_j)$;
9: **end if**
10: **end for**
11: $v^i_{slp} \leftarrow v_i$;
12: $v^i_{wak} \leftarrow \arg\max_{v_j} \zeta(v^i_{slp}, v_j)$
13: add v^i_{slp} and v^i_{wak} into $\langle v_{slp} v_{wak} \rangle$;
14: **end for**
15: **return** $\langle v_{slp}, v_{wak} \rangle$. =0

ζ is reflected by the execution progress per time unit which is defined as execution speed. After that, for each possible sleep voltage $v_{slp} \in \langle v \rangle$, Algorithm 6.5.1 searches the optimal wakeup voltage pair $v_{wak} = \arg\max_{v_j} \zeta(v^i_{slp}, v_j)$. Finally, current v_{slp} and v_{wak} will be both added into $\langle v_{slp}, v_{wak} \rangle$.

After finding the optimal voltage set $\langle v_{slp}, v_{wak} \rangle$, the routine handling procedures are detailed in Algorithm 6.5.2, whose purpose is to maximize the energy efficiency. The inputs for this algorithm include the optimal voltage set $\langle v_{slp}, v_{wak} \rangle$, the required checkpointing energy E_{ck}, reserve energy ΔE, checkpointing indicator I^{nv}_{ck} (non-volatile), restart indicator I^{v}_{rs} (volatile), parameter-update indicator I^{nv}_{up} (non-volatile), and the ADC period T_{ADC}.

Initially, from line 1 to 5, Routine Handler initiates intermediate parameters including optimal sleep voltage v^{opt}_{slp}, optimal wakeup voltage v^{opt}_{wak}, checkpointing voltage v_{ck}, trigger-point voltage v_{tr}, and routine event ID R_{id} which can represent sleep (checkpointing), wakeup (resuming), or execution. Here, R_{id} is initially set to sleep.

With the above initialization, the procedure of Routine Handler is detailed as follows. For every T_{ADC} period in line 7, Routine Handler gets voltage v on the capacitor. Since initially $R_{id} = sleep$, the edge device first enters into the sleep mode. When $v > v^{opt}_{wak}$ (line 8 - 11) , the edge device wakes up with $R_{id} = wakeup$.

Algorithm 6.5.2 Routine Handler

Input: $\langle v_{slp}, v_{wak} \rangle$, E_{ck}, ΔE, I_{ck}^{nv}, I_{rs}^{v}, I_{up}^{nv}, and T_{ADC}
Output: Maximum energy efficiency

1: $v_{slp}^{opt} = \arg\max \zeta(v_{slp})$; /* optimal V_{sleep}; */
2: $v_{wak}^{opt} = \arg\max \zeta(v_{wak})$; /* optimal V_{wakeup}; */
3: $v_{ck} = (v_{slp}^{opt2} + \frac{2E_{ck}}{C})^{\frac{1}{2}}$; /* checkpointing */
4: $v_{tr} = (v_{slp}^{opt2} + \frac{2(E_{ck}+\Delta E)}{C})^{\frac{1}{2}}$; /* triggerpoint */
5: $R_{id} = Sleep$; /* routine event ID */
6: **while** $t\%T_{ADC} = 0$ **do**
7: $v \leftarrow$voltage on capacitor;
7: **switch** R_{id} **do**
7: **case** Sleep
8: **if** $v > v_{wak}^{opt}$ **then**
9: wake up the system;
10: $R_{id} \leftarrow Wakeup$;
11: **end if**
12: **if** $v \leq v_{ck}$ **then**
13: **if** $I_{ck}^{nv} = 0$ **then**
14: conduct checkpointing;
15: $I_{ck}^{nv} = 1$;
16: **end if**
17: **end if**
17: **case** Wakeup
18: **if** $I_{rs}^{v} \neq 1$ **then**
19: resume previous execution state;
20: **end if**
21: $I_{up}^{nv} = 0$;
22: $R_{id} \leftarrow Execution$;
22: **case** Execution
23: **if** $I_{up}^{nv} = 0$ **then**
24: $APU()$;
25: **end if**
26: **if** $v \leq v_{tr}$ **then**
27: $R_{id} \leftarrow Sleep$;
28: **else**
29: continue task execution;
30: **end if**
31: **end while**=0

During the wakeup stage (line 18 - 22), Routine Handler first checks the I_{rs}^{v}. $I_{rs}^{v} \neq 1$ means that the edge device has restarted and all the volatile data are lost. The trick here is that I_{rs}^{v} is a volatile variable which will be reset ($I_{rs}^{v} \neq 1$) after restarting automatically. In this case, the edge device needs to resume the previous execution state with an update indicator $I_{up}^{nv} = 0$ and $R_{id} = Execution$ before proceeding any further.

During the execution stage, since $I_{up}^{nv} = 0$, the adaptive parameter updating (APU) algorithm first needs to be executed for parameter updating (line 24) which will be detailed in Algorithm 6.5.3. After parameter updating, the edge device can continue execution. During the execution, the Routine Handler keeps monitoring the voltage on the capacitor. When $v \leq v_{tr}$, the edge device enters into sleep mode by setting $R_{id} = sleep$. During the sleep mode, if the voltage keeps dropping, it means that a power outage may happen. Therefore, once v continues to drop below v_{ck} (line 12 - 15), checkpointing needs to be conducted. However, in case of v fluctuating around v_{ck} due to unstable harvesting power, the Routine Handler sets checkpointing indicator $I_{ck}^{nv} = 1$ to prevent redundant checkpointing for energy saving. Once execution resumes, APU updates $I_{ck}^{nv} = 0$ to allow new checkpointing. Algorithm 6.5.3 gives details of APU.

Algorithm 6.5.3 Adaptive Parameter Update (APU)

Input: Δv, $\langle RS \rangle^{nv}$, and inputs of Routine Handler
Output: updated $I_{ck}^{nv}, I_{rs}^{v}, I_{up}^{nv}, v_{slp}^{opt}, v_{wak}^{opt}, v_{ck}$, and v_{tr}

1: $I_{rs}^{v} = 1$; $I_{ck}^{nv} = 0$; $I_{up}^{nv} = 1$;
2: **if** $\arg\max \zeta(v_{slp}) - \min\langle v_{slp} \rangle > \Delta v$ **then**
3: add I_{rs}^{v} into non-volatile array $\langle RS \rangle$;
4: **if** any $rs_i \in \langle RS \rangle^{nv}$ satisfies $rs_i \neq 1$ **then**
5: $v_{slp}^{opt} = \arg\max \zeta(v_{slp})$;
6: **else**
7: $v_{\alpha} = \{[\arg\max \zeta(v_{slp})]^2 - \frac{2(E_{ck}+\Delta E)}{C}\}^{\frac{1}{2}}$;
8: $v_{\beta} = \min\langle v_{slp} \rangle + \Delta v$;
9: $v_{slp}^{opt} = \max(v_{\alpha}, v_{\beta})$;
10: **end if**
11: $v_{ck} = (v_{slp}^{opt2} + \frac{2E_{ck}}{C})^{\frac{1}{2}}$;
12: $v_{tr} = (v_{slp}^{opt2} + \frac{2(E_{ck}+\Delta E)}{C})^{\frac{1}{2}}$;
13: $v_{wak}^{opt} = \arg\max \zeta(v_{wak}, v_{slp}^{opt})$;
14: **end if**
15: **return** $I_{ck}^{nv}, I_{rs}^{v}, I_{up}^{nv}, v_{slp}^{opt}, v_{wak}^{opt}, v_{ck}$, and v_{tr}; =0

The inputs of APU include Δv (a safe voltage distance), $\langle RS \rangle^{nv}$ (a non-volatile set to record restart incidences), and all inputs of Routine Handler. The outputs contain updated I_{ck}^{nv}, I_{rs}^{v}, I_{up}^{nv}, v_{slp}^{opt}, v_{wak}^{opt}, v_{ck}, and v_{tr}. The rationale of conducting APU is that even if v_{slp}^{opt} is optimal, the edge device enters into sleep mode when $v \leq v_{tr}$. If the harvesting power is large enough to charge up the capacitor, no checkpointing will be conducted. In this sense, the edge device does not sleep at v_{slp}^{opt}. Therefore, further optimization is required.

At the beginning (line 1), APU sets $I_{rs}^{v} = 1$ to monitor restart incidence, $I_{ck}^{nv} = 0$ to allow new checkpointing to happen, and $I_{up}^{nv} = 1$ to prevent redundant updating for energy saving. Then, APU evaluates where the edge device is eligible for updating v_{slp}^{opt} and v_{wak}^{opt} (line 3). Specifically, if the difference between v_{slp}^{slp} and minimum sleep voltage $\min\langle v_{slp} \rangle$ is larger than Δv, APU proceeds further voltage adjustment. Otherwise, APU stops parameter update.

Assume voltage updating is eligible, then, APU records the latest I_{rs}^{v} into $\langle RS \rangle^{nv}$. Here, any $rs_i \in \langle RS \rangle^{nv}$ satisfying $rs_i \neq 1$ means that a power outage has happened recently. In this case, APU sticks to original $v_{slp}^{opt} = \arg\max \zeta(v_{slp})$. However, all $rs_i \in \langle RS \rangle^{nv}$ satisfying $rs_i = 1$, indicates no power outage has happened recently. In this case, APU reduces v_{slp}^{opt} and tries to set the triggerpoint voltage v_{tr} as close to $\arg\max \zeta(v_{slp})$ as possible (line 6 - 9). After updating v_{slp}^{opt}, APU further updates v_{ck}, v_{tr}, and v_{wak}^{opt} (line 11-13). Finally APU returns the updated parameters (line 15).

6.6 Experiments

6.6.1 Experimental Setup

In this section, details of the hardware platform, power traces, and benchmarks are described.

6.6.1.1 Hardware Platform The targeted ultra-low-power edge device is TI's ultra-low-power evaluation board with MSP430FR6989, which consists of a 16-bit MCU, a 12-bit ADC, a 2KB volatile SRAM, a 128KB nonvolatile FRAM memory, and different peripherals

for sensing and data communication. The power regulator is Bq25570 + LTC3459EDC which is able to supply a constant voltage of 3.3V to power the edge device and a maximum voltage of 4.2V to charge up the capacitor with the capacitance of $470\mu F$.

6.6.1.2 Power Trace We collect the power traces from Powercast RF energy harvester. Then, we synthesize four power traces in Figure 19 using signal generators for performance evaluation.

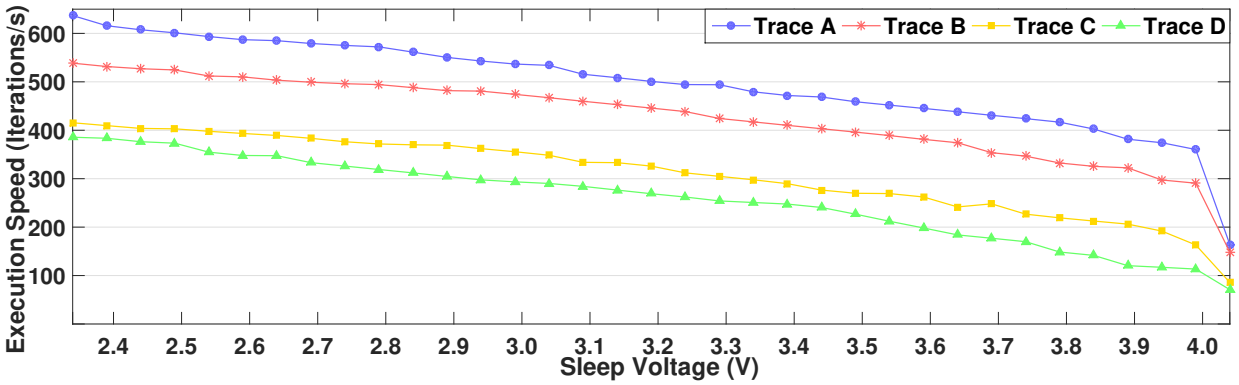
The four power traces with different power magnitudes are recorded from the oscilloscope at the sample rate of $240KSa/s$. On average, the power they can provide is 7.27mW, 6.19mW, 4.94mW, and 4.11mW, respectively. The harvesting power is far less than the working power of edge devices.

6.6.1.3 Benchmarks Ten MSP430 benchmarks in Figure 29 are from Texas Instruments [1] are used for performance evaluation. Among them, four benchmarks 16_bit_2dim, $Floating\_Point\_Math$, $Matrix\_Multiplication$, and Fir_Filter are modified with ADC sampling and FRAM access to evaluate the execution speed bottleneck caused by hardware modules. To evaluate the performance of Routine Handler, each benchmark is executed repetitively to measure the average execution speed. To measure the performance of Frequency Modulator, all benchmarks are scheduled one after another in a round-robin style.

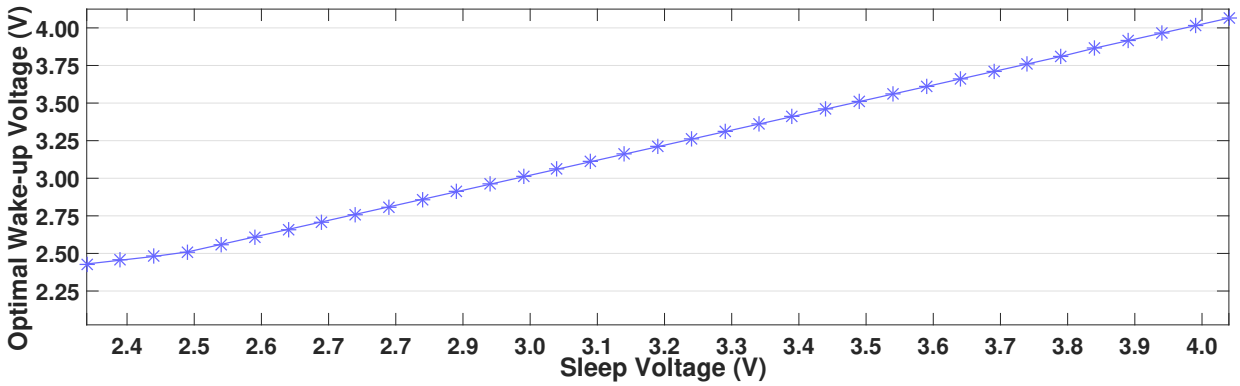
6.6.2 Experimental Evaluation

6.6.2.1 Observation of $\zeta(v_{slp}, v_{wak})$ In this section, we observe the energy efficiency with respect to v_{slp} and v_{wak} based on Algorithm 6.5.1 by executing benchmark 8_bit_math which has the average active power of $28.95mW$. Due to the discrete nature of ADC module, the entire voltage range $[v_{min}, v_{max}]$ is divided into 35 levels. For each $v_{slp} \in [v_{min}, v_{max}]$, the maximum energy efficiency (execution speed) and the respective v_{slp} are shown in Figure 42(a) and 42(b) respectively.

From Figure 42, when $v_{slp} = 2.37V$ and $v_{wak}^{opt} = 2.48V$, the maximum energy efficiency can be achieved, hence, they will be used by ENZYME to maximize the energy efficiency.



(a) Maximum Energy Efficiency with Different Sleep Voltages



(b) Optimal Wakeup Voltages with Different Sleep Voltages

Figure 42: Maximum Energy Efficiency and Optimal Wakeup Voltages with Different Sleep Voltages

Notice that the trends of the optimal v_{wak} and the respective maximum ζ agree with the theoretical analysis in section 6.4.2. Overall, referring to harvesting power and working power of edge device, when $P_H \leq P_W/5$, the proposed theory can be very effective. Due to the hardware constraint, the power regulator cannot maintain a constant output voltage if the voltage on the capacitor is lower than $2.2V$, which may increase the chance of runtime error. Therefore, we set the minimum sleep voltage to $2.34V$ to ensure a constant $3.3V$ output voltage when MSP430FR6989 is in active mode.

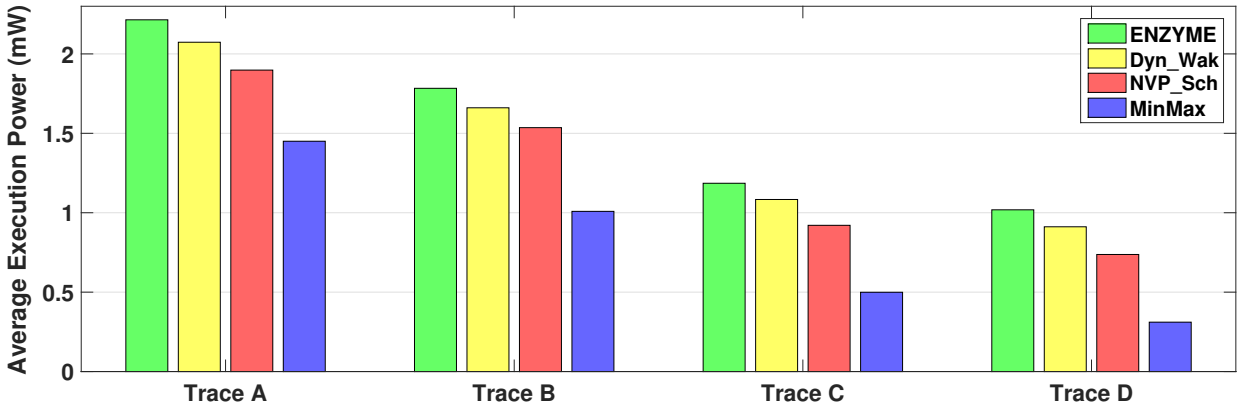
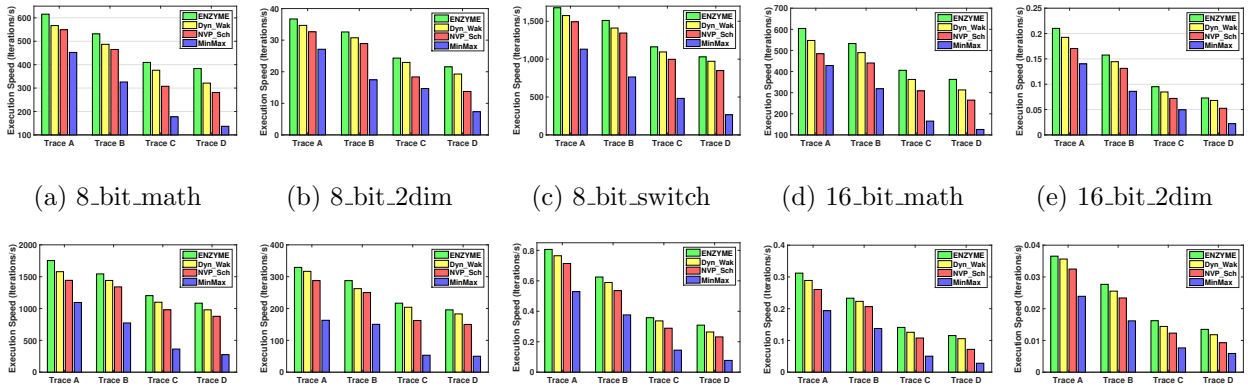


Figure 43: Average execution power

6.6.2.2 Energy Efficiency Evaluation In this section, the performance of ENZYME is evaluated in terms of energy efficiency. The baselines include Dyn_Wak which conducts dynamic wakeup strategy [34], NVP_Sch which has a preset runtime routine for execution [36], and $MinMax$ which has the minimum v_{slp} and the maximum v_{wak}. The results are shown in Figure 44.

As we can see from Figure 44, ENZYME shows significant improvements in energy efficiency over baselines. The reason for such superiority is bifold. First, the optimal sleep and wakeup voltage pairs can extract the maximum energy from energy harvester. Second, without a sophisticated and energy-consuming power analysis procedure, ENZYME is lightweight and can further save extra energy. Considering the energy consumption of each



(a) 8_bit_math (b) 8_bit_2dim (c) 8_bit_switch (d) 16_bit_math (e) 16_bit_2dim

(f) 16_bit_switch (g) 32_bit_math (h) Floating_point_math (i) Matrix_Multiplication (j) Fir_Filter

Figure 44: ENZYME vs. Baselines regarding energy efficiency

benchmark in a single iteration from Figure 29, the average execution power with different techniques is shown in Figure 43. Overall, ENZYME has 8.8% improvement over Dyn_Wak, 24.27% improvement over NVP_Sch, and 105.42% improvement over $MinMax$.

6.7 Summary

This chapter proposes a routine handler which is able to govern the runtime routine events of sleep and wake-up for self-powered IoT edge devices to help the energy harvester and power regualator bundle to maximize the energy extraction rate. The significance of this technique is that the the proposed routine handler is lightweight and highly efficient, which makes it especially suited for energy constraint IoT Edge devices. Experiments show that the routine handler can help the edge device to achieve an promising energy extraction rate wthout any extra hardware support.

7.0 Conclusion

This dissertation aims at solving urgent low-energy efficiency and reliability issues for self-powered IoT edge devices through software approaches, which are more flexible, cheaper, and equally effective compared with hardware solutions. Specifically, first, to prevent execution progress loss during the power outage, NVP-aware task schedulers are proposed to maximize the overall task execution progress especially for the atomic tasks which are better to finish execution before the power outage. Second, to minimize both the time and energy overhead of checkpointing operation, an intelligent checkpointing scheme is proposed which not only can ensure a successful checkpointing but also can predict the necessity of conducting checkpointing to avoid excessive checkpointing overhead. to avoid inappropriate runtime CPU clock frequency, which consumes extra energy while delivering less execution progress, a CPU frequency Modulator is proposed which adjusts the runtime CPU clock frequency accordingly. To thrive in ultra-low harvesting power scenarios, a light-weight software paradigm is proposed to help maximize the energy extraction rate of energy harvester and power regulator bundle. Besides, checkpointing under such an ultra-low scenario is also optimized for more energy-efficient and light-weight operation. The potential of this work can be significant on social and economic perspectives as the contributions of this work can help trillions of existing and more coming IoT edge devices thriving in power-constrained scenarios without tampering hardware structure.

Bibliography

[1] MSP430 Benchmarks. http://www.mcuzone.com/work/DIMM144-CPU-MSP430/slaa205a.pdf.

[2] MSP430 MCU. http://www.ti.com/lsds/ti/microcontrollers_16-bit_32-bit/msp/ultra-low_power/msp430frxx_fram/products.page.

[3] MSP430FR6989. http://www.ti.com/tool/MSP-EXP430FR6989.

[4] WISP Project. http://sensor.cs.washington.edu/WISP.html.

[5] D. Balsamo, A. Das, A. S. Weddell, D. Brunelli, B. M. Al-Hashimi, G. V. Merrett, and L. Benini. Graceful performance modulation for power-neutral transient computing systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35(5):738–749, May 2016.

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

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

[8] S. Bandyopadhyay and A. P. Chandrakasan. Platform architecture for solar, thermal, and vibration energy combining with mppt and single inductor. *IEEE Journal of Solid-State Circuits*, 47(9):2199–2215, Sept 2012.

[9] C. L. Chang and T. C. Lee. A compact multi-input thermoelectric energy harvesting system with 58.5 In *2014 International Symposium on Integrated Circuits (ISIC)*, pages 1–4, Dec 2014.

[10] C. Ding, N. Liu, Y. Wang, J. Li, S. Heidari, J. Hu, and Y. Liu. Multisource indoor energy harvesting for nonvolatile processors. *IEEE Design Test*, 34(3):42–49, June 2017.

[11] S. Ducharme, T. J. Reece, C. Othon, and R. K. Rannow. Ferroelectric polymer langmuir-blodgett films for nonvolatile memory applications. *IEEE Transactions on Device and Materials Reliability*, 5(4):720–735, 2005.

[12] V. Hanumaiah and S. Vrudhula. Energy-efficient operation of multicore processors by dvfs, task migration, and active cooling. *IEEE Transactions on Computers*, 63(2):349–360, 2014.

[13] Y. Horii, Y. Hikosaka, A. Itoh, K. Matsuura, M. Kurasawa, G. Komuro, K. Maruyama, T. Eshita, and S. Kashiwagi. 4 mbit embedded fram for high performance system on chip (soc) with large switching charge, reliable retention and high imprint resistance. In *International Electron Devices Meeting, 2002. IEDM '02.*, pages 539–542, 2002.

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

[15] X. Jiang, J. Polastre, and D. Culler. Perpetual environmentally powered sensor networks. In *Fourth International Symposium on Information Processing in Sensor Networks, 2005.*, pages 463–468, 2005.

[16] W. kei Yu, S. Rajwade, S.-E. Wang, B. Lian, G. Suh, and E. Kan. A non-volatile microcontroller with integrated floating-gate transistors. In *IEEE/IFIP 41st International Conference on Dependable Systems and Networks Workshops (DSN-W),*, pages 75–80, 2011.

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

[18] H. Kim, S. Priya, H. Stephanou, and K. Uchino. Consideration of impedance matching techniques for efficient piezoelectric energy harvesting. *IEEE transactions on ultrasonics, ferroelectrics, and frequency control*, 54(9):1851–1859, 2007.

[19] N. Kong, D. S. Ha, A. Erturk, and D. J. Inman. Resistive impedance matching circuit for piezoelectric energy harvesting. *Journal of Intelligent Material Systems and Structures*, 21(13):1293–1302, 2010.

[20] J. Kymissis, C. Kendall, J. Paradiso, and N. Gershenfeld. Parasitic power harvesting in shoes. *ISWC'98*, pages 132–139, 1998.

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

[22] H. Li, Y. Liu, C. Fu, C. J. Xue, D. Xiang, J. Yue, J. Li, D. Zhang, J. Hu, and H. Yang. Performance-aware task scheduling for energy harvesting nonvolatile processors considering power switching overhead. In *Design Automation Conference (DAC), 2016 53nd ACM/EDAC/IEEE*, pages 1–6. IEEE, 2016.

[23] Z. Li, Y. Liu, D. Zhang, C. J. Xue, Z. Wang, X. Shi, W. Sun, J. Shu, and H. Yang. Hw/sw co-design of nonvolatile io system in energy harvesting sensor nodes for optimal data acquisition. In *Proceedings of the 53rd Annual Design Automation Conference*, page 154. ACM, 2016.

[24] J. Liang and W.-H. Liao. Impedance modeling and analysis for piezoelectric energy harvesting systems. *IEEE/ASME Transactions on Mechatronics*, 17(6):1145–1157, 2012.

[25] C.-W. Liu and J. S. Thorp. New methods for computing power system dynamic response for real-time transient stability prediction. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 47(3):324–337, Mar 2000.

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

[27] Y. Liu, Z. Wang, A. Lee, F. Su, C.-P. Lo, Z. Yuan, C.-C. Lin, Q. Wei, Y. Wang, Y.-C. King, et al. A 65nm reram-enabled nonvolatile processor with 6x reduction in restore time and 4x higher clock frequency using adaptive data retention and self-write-termination nonvolatile logic. In *2016 IEEE International Solid-State Circuits Conference (ISSCC)*, pages 84–86. IEEE, 2016.

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

[29] A. Mishra and A. K. Tripathi. Energy efficient voltage scheduling for multi-core processors with software controlled dynamic voltage scaling. *Applied Mathematical Modelling*, 38(14):3456–3466, 2014.

[30] P. Mitcheson. Energy harvesting for human wearable and implantable bio-sensors. In *Engineering in Medicine and Biology Society (EMBC), 2010 Annual International Conference of the IEEE*, pages 3432–3436, 2010.

[31] H. Nakamoto, D. Yamazaki, T. Yamamoto, H. Kurata, S. Yamada, K. Mukaida, T. Ninomiya, T. Ohkawa, S. Masui, and K. Gotoh. A passive uhf rf identification cmos tag ic using ferroelectric ram in 0.35um technology. *IEEE Journal of Solid-State Circuits*, 42(1):101–110, 2007.

[32] C. Pan, S. Gu, M. Xie, Y. Liu, C. J. Xue, and J. Hu. Wear-leveling aware page management for non-volatile main memory on embedded systems. *IEEE Transactions on Multi-Scale Computing Systems*, 2(2):129–142, 2016.

[33] C. Pan, S. Gu, M. Xie, Y. Liu, C. J. Xue, and J. Hu. Wear-leveling aware page management for non-volatile main memory on embedded systems. *IEEE Transactions on Multi-Scale Computing Systems*, 2(2):129–142, April 2016.

[34] C. Pan, M. Xie, and J. Hu. Maximize energy utilization for ultra-low energy harvesting powered embedded systems. In *2017 IEEE 23rd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 1–6, Aug 2017.

[35] C. Pan, M. Xie, J. Hu, Y. Chen, and C. Yang. 3m-pcm: exploiting multiple write modes mlc phase change main memory in embedded systems. In *Proceedings of the 2014 International Conference on Hardware/Software Codesign and System Synthesis*, page 33. ACM, 2014.

[36] C. Pan, M. Xie, Y. Liu, Y. Wang, C. J. Xue, Y. Wang, Y. Chen, and J. Hu. A lightweight progress maximization scheduler for non-volatile processor under unstable energy harvesting. *SIGPLAN Not.*, 52(5):101–110, June 2017.

[37] C. Pan, M. Xie, C. Yang, Y. Chen, and J. Hu. Exploiting multiple write modes of nonvolatile main memory in embedded systems. *ACM Trans. Embed. Comput. Syst.*, 16(4):110:1–110:26, May 2017.

[38] C. Park and P. H. Chou. Ambimax: Autonomous energy harvesting platform for multi-supply wireless sensor nodes. In *SECON'06.*, pages 168–177, 2006.

[39] Z. Qin, Y. Wang, D. Liu, Z. Shao, and Y. Guan. Mnftl: An efficient flash translation layer for mlc nand flash memory storage systems. In *Proceedings of the 48th Design Automation Conference*, pages 17–22. ACM, 2011.

[40] V. Raghunathan, A. Kansal, J. Hsu, J. Friedman, and M. Srivastava. Design considerations for solar energy harvesting wireless embedded systems. In *Fourth International Symposium on Information Processing in Sensor Networks*, pages 457–462, 2005.

[41] B. Ransford, S. S. Clark, M. Salajegheh, and K. Fu. Getting things done on computational rfids with energy-aware checkpointing and voltage-aware scheduling. In *HotPower'08*, pages 5–5, 2008.

[42] B. Ransford, J. Sorber, and K. Fu. Mementos: system support for long-running computation on rfid-scale devices. *ACM SIGPLAN Notices*, 47(4):159–170, 2012.

[43] N. Sakimura, T. Sugibayashi, R. Nebashi, and N. Kasai. Nonvolatile magnetic flip-flop for standby-power-free socs. *IEEE Journal of Solid-State Circuits*, 44(8):2244–2250, 2009.

[44] N. Sakimura, Y. Tsuji, R. Nebashi, H. Honjo, A. Morioka, S. Fukami, S. Miura, N. Kasai, T. Endoh, H. Ohno, T. Hanyu, and T. Sugibayashi. 10.5a 90nm 20mhz fully nonvolatile microcontroller for standby-power-critical applications. *IEEE International Solid-State Circuits Conference*, 12(4):184–185, 2014.

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

[46] C. Shi, B. Miller, K. Mayaram, and T. Fiez. A multiple-input boost converter for low-power energy harvesting. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 58(12):827–831, Dec 2011.

[47] T. Starner. Human-powered wearable computing. *IBM systems Journal*, 35(3.4):618–629, 1996.

[48] J. Taneja, J. Jeong, and D. Culler. Design, modeling, and capacity planning for micro-solar power sensor networks. In *IPSN'08*, pages 407–418, 2008.

[49] U. R. Tida, R. Yang, C. Zhuo, and Y. Shi. On the efficacy of through-silicon-via inductors. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 23(7):1322–1334, July 2015.

[50] U. R. Tida, C. Zhuo, and Y. Shi. Novel through-silicon-via inductor-based on-chip dc-dc converter designs in 3d ics. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 11(2):16, 2014.

[51] U. R. Tida, C. Zhuo, and Y. Shi. Through-silicon-via inductor: Is it real or just a fantasy? In *19th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 837–842, Jan 2014.

[52] C. Wang, N. Chang, Y. Kim, S. Park, Y. Liu, and H. Lee. Storage-less and converter-less maximum power point tracking of photovoltaic cells for a nonvolatile microprocessor. *ASP-DAC*, pages 379–384, 2014.

[53] Y. Wang, Y. Liu, S. Li, D. Zhang, B. Zhao, M.-F. Chiang, Y. Yan, B. Sai, and H. Yang. A 3us wake-up time nonvolatile processor based on ferroelectric flip-flops. In *Proceedings of the ESSCIRC*, pages 149–152, 2012.

[54] M. Xie, C. Pan, J. Hu, C. J. Xue, and Q. Zhuge. Non-volatile registers aware instruction selection for embedded systems. In *2014 IEEE 20th International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 1–9, 2014.

[55] M. Xie, C. Pan, J. Hu, C. Yang, and Y. Chen. Checkpoint-aware instruction scheduling for nonvolatile processor with multiple functional units. In *The 20th Asia and South Pacific Design Automation Conference*, pages 316–321, 2015.

[56] M. Xie, C. Pan, M. Zhao, Y. Liu, C. J. Xue, and J. Hu. Avoiding data inconsistency in energy harvesting powered embedded systems. *ACM Trans. Des. Autom. Electron. Syst.*, 23(3):38:1–38:25, Mar. 2018.

[57] M. Xie, M. Zhao, C. Pan, J. Hu, Y. Liu, and C. J. Xue. Fixing the broken time machine: Consistency-aware checkpointing for energy harvesting powered non-volatile processor. In *Proceedings of the 52Nd Annual Design Automation Conference*, DAC '15, pages 184:1–184:6, 2015.

[58] H. Xu, R. Li, L. Zeng, K. Li, and C. Pan. Energy-efficient scheduling with reliability guarantee in embedded real-time systems. *Sustainable Computing: Informatics and Systems*, 2018.

[59] W.-k. Yu, S. Rajwade, S.-E. Wang, B. Lian, G. E. Suh, and E. Kan. A non-volatile microcontroller with integrated floating-gate transistors. In *Dependable Systems and*

Networks Workshops (DSN-W), 2011 IEEE/IFIP 41st International Conference on, pages 75–80. IEEE, 2011.

[60] D. Zhang, Y. Liu, X. Sheng, J. Li, T. Wu, C. J. Xue, and H. Yang. Deadline-aware task scheduling for solar-powered nonvolatile sensor nodes with global energy migration. In *Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE*, pages 1–6. IEEE, 2015.

[61] M. Zhao, C. Fu, Z. Li, Q. Li, M. Xie, Y. Liu, J. Hu, Z. Jia, and C. J. Xue. Stack-size sensitive on-chip memory backup for self-powered nonvolatile processors. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 36(11):1804–1816, Nov 2017.

[62] M. Zhao, Q. Li, M. Xie, Y. Liu, J. Hu, and C. J. Xue. Software assisted non-volatile register reduction for energy harvesting based cyber-physical system. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, DATE '15, pages 567–572, 2015.

[63] C. Zhuo, K. Unda, Y. Shi, and W. K. Shih. From layout to system: Early stage power delivery and architecture co-exploration. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 1–1, 2018.

[64] M. Zwerg, A. Baumann, R. Kuhn, M. Arnold, R. Nerlich, M. Herzog, R. Ledwa, C. Sichert, V. Rzehak, P. Thanigai, et al. An 82μa/mhz microcontroller with embedded feram for energy-harvesting applications. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2011 IEEE International*, pages 334–336. IEEE, 2011.

[65] M. Zwerg, A. Baumann, R. Kuhn, M. Arnold, R. Nerlich, M. Herzog, R. Ledwa, C. Sichert, V. Rzehak, P. Thanigai, and B. Eversmann. An 82ua/mhz microcontroller with embedded feram for energy-harvesting applications. In *IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pages 334–336, 2011.