A Lightweight Progress Maximization Scheduler for Non-volatile Processor under Unstable Energy Harvesting

Chen Pan[1] Mimi Xie[1] Yongpan Liu[2] Yanzhi Wang[3] Chun Jason Xue[4] Yuangang Wang[5]
Yiran Chen[6] Jingtong Hu[1]

[1]School of Electrical and Computer Engineering, Oklahoma State University, Stillwater, OK, 74078, USA.
[2]Department of Electronic Engineering, Tsinghua University, Beijing, P.R. China, 100084.
[3]Department of Electrical Engineering & Computer Science, Syracuse University, Syracuse, NY 13244.
[4]Department of Computer Science, City University of Hong Kong, Tat Chee Ave, Kowloon, Hong Kong.
[5]Huawei Technologies Co., Ltd. P.R. China.
[6]Department of Electrical and Computer Engineering, Duke University, Durham, NC 27708.
ypliu@tsinghua.edu.cn, ywang393@syr.edu, jasonxue@cityu.edu.hk, wangyuangang@huawei.com,
yiran.chen@duke.edu{chen.pan, mimix, jthu}@okstate.edu,

Abstract

Energy harvesting techniques become increasingly popular as power supplies for embedded systems. However, the harvested energy is intrinsically unstable. Thus, the program execution may be interrupted frequently. Although the development of non-volatile processors (NVP) can save and restore execution states, both hardware and software challenges exist for energy harvesting powered embedded systems. On the hardware side, existing power detector only signals the "poor" quality of the harvested power based on a preset threshold voltage. The inappropriate setting of this threshold will make the NVP based embedded system suffer from either unnecessary checkpointing or checkpointing failures. On the software side, not all tasks can be checkpointed. Once the power is off, these tasks will have to restart from the beginning. In this paper, a task scheduler is proposed to maximize task progress by prioritizing tasks which cannot be checkpointed when power is weak so that they can finish before the power outage. To assist task scheduling, three additional modules including voltage monitor, checkpointing handler, and routine handler, are proposed. Experimental results show increased overall task progress and reduced energy consumption.

CCS Concepts • **Computer systems organization → Embedded software**

Keywords Energy Harvesting, Task Scheduling, Progress Maximization, Non-volatile Memory, Non-volatile Processor

1. Introduction

Embedded systems tend to be smarter, smaller and more reliable in this era of Internet of Things (IoT) where all "things" are embedded with electronics, software, sensors, and connectivity in our daily life. These embedded "things" interleave and communicate with each other resulting in improved social and economic benefits and overall human well-being. Typical applications of IoT devices include power grids, structural health, automotive, logistic, healthcare, etc. There are about 9 billion IoT devices today and the number keeps growing. It is estimated that the IoT will consist of almost 50 billion objects by 2020 [1].

While the vision is promising and exciting, there are several challenges in achieving this goal. One of the challenges is how to power these 50 billion embedded devices. While battery power is not a favorable solution in a 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 embedded devices.

Energy harvesting devices harvest energy from their surrounding energy sources which include kinetic, electromagnetic radiation (including light and RF), thermal energy, etc. and then convert it into electric energy. The harvested energy can be used to recharge a capacitor or, in some cases, to directly power the electronics. However, there is an intrinsic drawback with harvested energy. They are mostly weak and unstable. With an unstable power supply, the processor execution will be interrupted frequently. Frequent turning-off and booting-up add an extra burden on the limited energy budget. Even worse, in some cases, large tasks can never get finished since the intermediate results cannot be saved. However, thanks to the non-volatile processors (NVP) [8, 19, 23], which enables instant on/off for these devices. NVP systems are equipped with a piece of non-volatile memory (NVM) such as FRAM [16, 23]. Every time there is a power outage, the processor's states will be saved from volatile memory into the NVM, which is known as checkpointing. Then the next time when input power becomes abundant, the processor's state can be restored and program execution resumes. In this way, we can make sure the program execution is "accumulative" and resilient to frequent power outages.

Nevertheless, both hardware and software challenges exist. On the hardware side, the power regulator of NVP systems is only able to warn NVP system about the "poor" quality of the harvesting power by signaling interrupts. However, the warning is based on a preset voltage threshold. Inappropriate settings of this threshold will degrade the performance of NVP system. On one hand, if the threshold is too high, the NVP system will waste energy with frequent checkpointing. On the other hand, if the threshold is too low, the checkpointing may fail. Therefore, it is necessary for

LCTES'17, June 21–22, 2017, Barcelona, Spain
© 2017 ACM. 978-1-4503-5030-3/17/06...$15.00
http://dx.doi.org/10.1145/3078633.3081038

NVP system to proactively initiate voltage detection which requires software coordination.

On the software side, all unfinished tasks should be checkpointed before a power outage. However, not all tasks can be checkpointed. We define tasks that cannot be checkpointed as Un_CK tasks and tasks that can be checkpointed as CK tasks. Un_CK tasks, such as communication, are susceptible to delay. A long time suspension may result in errors and the communication session needs to be restarted. Others, such as sensing, are unnecessary for checkpointing because incompletely sensed data might be useless. As Un_CK tasks will lose their progress after a power outage, we need a scheduler to prioritize Un_CK tasks to complete them before a power outage. In this way, task execution progress can be maximized. To assist task scheduling, additional three modules including voltage monitor, checkpointing handler, and routine handler are proposed to further maximize task progress. The proposed techniques can be easily incorporated into existing embedded system software to provide execution support under energy harvesting.

In a nutshell, the major contributions of this work include:

- A task scheduler is proposed to maximize overall task progress by incorporating the results of voltage monitor;
- A proactive voltage monitor is proposed to detect power interruptions and analyze the harvesting power with minimum hardware supports;
- An on-demand checkpoint handler is proposed to conduct checkpointing with energy efficiency and reliability;
- A routine handler is proposed which triggers sleep/wake-up events of energy harvesting systems to realize fast resuming.

The rest of this paper is organized as follows. Section 2 discusses related works. Section 3 introduces the system architecture and background. Section 4 provides motivational example of this work. Section 5 proposes NVP scheduler. Section 6 conducts the experiments. Finally, Section 7 concludes this paper.

2. Related Works

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 [6, 11, 14] 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 $67W$ [5] during a brisk walk. For ultra-low power devices, the sources with low power densities, such as breathing ($0.42W$), and body heat ($2.4{\sim}4.8W$), are able to provide sufficient power to drive the devices at low duty cycles [15]. With proper configuration and management [4], 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) are developed. In these processors, non-volatile memory [9, 10] 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 [12, 13] with nonvolatile memory for energy-harvesting devices. Microcontrollers such as TI's MSP430 series [2] employ FRAM as on-chip memory. Ransford et al. [13] 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) [3] was developed to achieve the 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 [16, 23]. This kind of processor attaches a nonvolatile memory cell to each volatile element and therefore allows fast local back-up of intermediate results and fast recovery. FRAM-based processors [8, 19, 23], present a great potential to be deployed in energy-harvesting devices. For example, Yu et al. [20] propose a non-volatile processor architecture which integrates non-volatile elements into volatile memory at bit granularity. To reduce the back-up overhead and energy, different technologies have been proposed including instruction scheduling [18], register reduction [22], and instruction selection [17].

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. [21] proposes a long-term deadline-aware scheduling algorithm to reduce energy consumption and deadline miss rates of tasks. [7] proposes a performance-aware task scheduling strategy for energy harvesting nonvolatile processors considering the power switching overhead. This paper will propose a lightweight scheduler which can maximize task progress and thus reduce energy consumption considering all types of tasks.

3. Architecture and Background

3.1 System Architecture

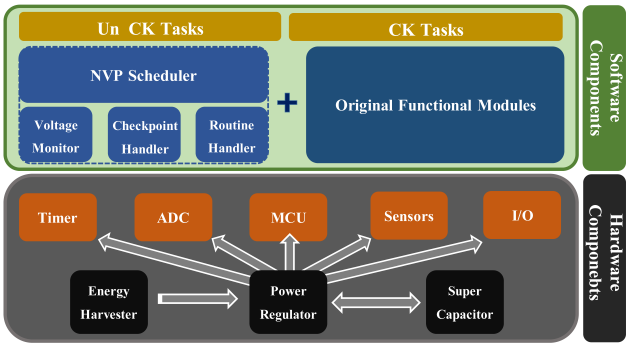

Figure 1. System Architecture

As shown in Figure 1, 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, piezoelectric, and radio frequency (RF) to electrical energy. The converted electrical 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 extra 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 incorporated into any existing energy harvesting embedded systems. Further, all tasks to be scheduled can be divided into CK tasks and Un_CK tasks.

3.2 Software Based CPU Checkpointing

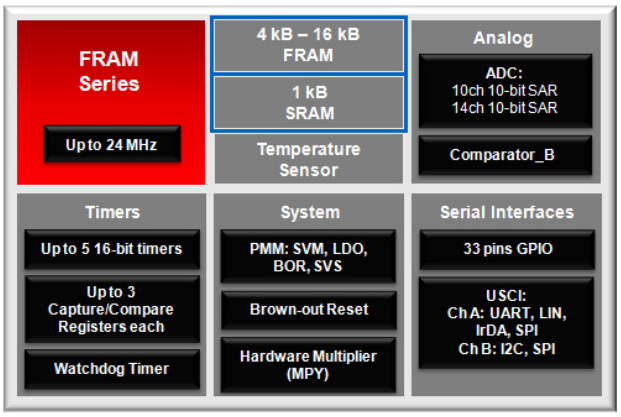

Figure 2. TI MSP430FRAM

Non-volatile processor, such as TI's MSP430 FRAM MCU shown in Figure 2, has both volatile and no-volatile on-chip memory. When a power outage occurs, all contents of the processor registers and SRAM should be checkpointed to FRAM. Hence, all the computation states can be saved. Once the power comes back on again, the computation can resume with the saved states.

Since there is no hardware to automatically save computation states, the software should take this responsibility. When detecting a low voltage below the preset threshold, the system will enter 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 non-volatile FRAM.

4. Motivational Examples

In this section, examples are given which show the motivation of this work.

The system we use 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, we employ two tasks. Task A can be checkpointed and task B cannot be checkpointed. 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.

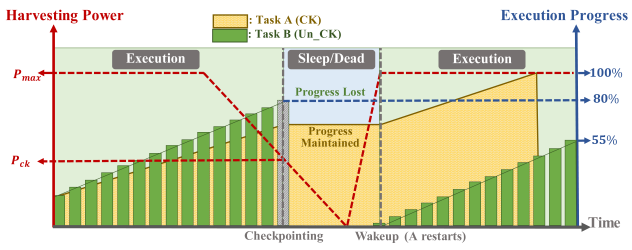

Figure 3. Un_CK Task B is Unfinished Before The Power Outage Resulting in Progress Setback.

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 can be checkpointed, its execution status including all the register files and stack can be stored into non-volatile memory when checkpointing is triggered. Therefore, once power comes back on again the system wakes up and the program execution of A resumes.

However, Since B is cannot be checkpointed, it loses entire execution status during the power outage. Once 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 the imminent power outage is detected. In this way, since A will not suffer from progress setback, if B can finish before the power outage, the overall execution progress can be maintained. Given the same power trace, figure. 4 shows the execution progress of both A and B which is prioritized after detecting the power drop.

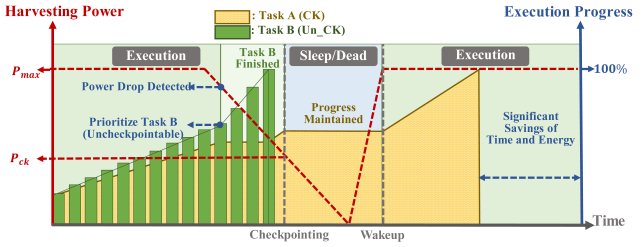

Figure 4. Prioritizing Un_CK Task B to Finish Before The Power Outage Results in Progress Maximization.

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 answer whether task B can finish before the power outage, the system needs several pieces of information. Here we conduct a serial of backward reasoning in Figure. 5 to find out what these pieces of information are and why the system needs to know them.

From Figure. 5, we can see that to answer whether B can finish before the power outage, we need to know the remaining time both for B to finish (t_B) and for the system to operate (T_S). If $T_S > t_B$, then prioritizing B could maximize the execution progress. In order to calculate t_B and T_S, the system needs the following four pieces of information.

1) The execution progress of task B (Pg_B);
2) The progress to time ratio of task B (Pg'_B);
3) The time when power drop happens (T_{dp});
4) The time when program execution stops (T_{sp}).

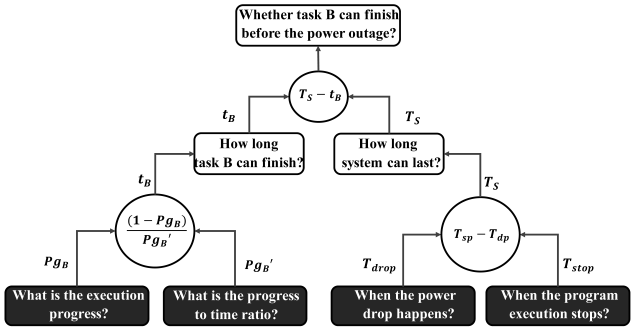

Figure 5. Required Information to Evaluate Whether Task B Can Finish Before The Power Outage.

Therefore, in Section. 5, aside from proposing a priority-based scheduling algorithm, techniques are also designed to achieve and maintain these four pieces of information for evaluating whether Un_CK tasks can be prioritized whenever power outages happen.

5. A lightweight NVP 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.

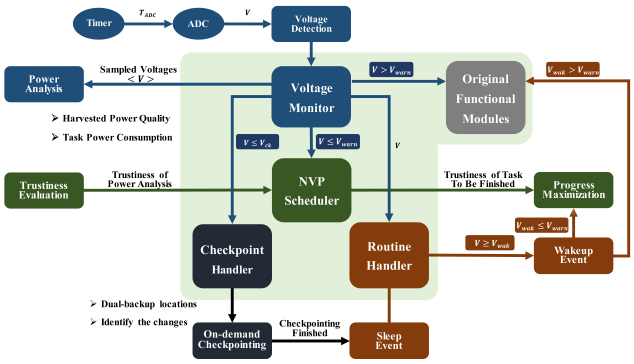

Figure 6. NVP Scheduler Overview

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 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 NVP system. Once voltage V drops below V_{warn}, 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 power recovers. By then, the routine handler will wake up the entire system. These functional modules interact with each other. The details of each component will be presented in Section 5.1, 5.2, 5.3, and 5.4, respectively.

5.1 NVP Scheduler

In this section, the NVP scheduler is proposed to maximize the overall task progress when the harvested power is insufficient to support NVP systems. 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:

- Un_CK tasks: tasks that cannot or do not need to be checkpointed which still have to start over when power comes back on again. Typical examples include communication, sensing, etc.
- CK tasks: tasks that can be checkpointed and resumed without causing 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 checkpointablility, as shown in Table 1.

Table 1. Task Information (Tab)

Tasks	Execution Power	Current Progress	Progress to Time Ratio	Checkpointablility
$Tsk[i]$	P_i	Pg_i	Pg_i'	True
$Tsk[j]$	P_j	Pg_j	Pg_j'	False

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}, in anxious of a potential power outage, NVP scheduler is triggered to prioritize the Un_CK tasks. Given the task ready queue Q and the task information table Tab as the inputs, Algorithm 5.1 shows details of NVP scheduler.

Algorithm 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 5.3*/
3: $Trust^{max} = 0$;
4: **for** each Un_CK $Tsk[i]$ in ready queue Q **do**
5: $Trust_i \Leftarrow TCE(T^S, Trust^P, Tsk[Cur], Tsk[i], Tab)$; /* Algorithm 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 CK tasks$\in Q$;
19: **end while**

The purpose of NVP Scheduler is to achieve overall task progress maximization by giving more scheduling priority to Un_CK tasks in the ready queue Q when 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 checkpointing starts.

At first, NVP scheduler needs to prioritize the scheduling for Un_CK 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 5.3. After calling DPA, in line 3, NVP scheduler initiates the parameter $Trust^{max}$ which will be further used to select the best Un_CK 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 5.2. During the calculation of $Trust_i$ for each Un_CK 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, NVP scheduler will break out the while loop to schedule for Un_CK tasks. This is because, if the best Un_CK candidate Sel is not worthwhile for scheduling, no other Un_CK tasks will be.

Finally, once there is no suitable Un_CK candidates for scheduling, NVP will conduct round-robin scheduling for CK 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 Un_CK 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 5.2 shows the details of the the evaluation.

Algorithm 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** $Trust_i$

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 NVP scheduler to make final decision.

In the next section, the voltage monitor is introduced, which is in charge of conducting voltage detection. Also, its sub-algorithm DPA needs to reply the calls from NVP system for analyzing the discharging power of the capacitor.

5.2 Voltage Monitor

In this section, a voltage monitor designed to conduct voltage measurement and power analysis will be presented.

5.2.1 Voltage Detection

Currently, the voltage regulators only warn 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. On 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. Therefore, the NVP system needs to proactively initiate voltage detection.

Many traditional techniques choose Vcc as the monitoring voltage source. However, because of the regulator, Vcc remains stable unless the stored energy is almost drained out. In this way, once the voltage monitor detects the Vcc drop, both NVP scheduler and checkpoint handler may not have enough time to respond, resulting in the progress setback. Therefore, Vcc is ruled out as voltage detection source.

One viable and appropriate monitoring source is the voltage on the storage capacitor which 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 NVP system consumes. Otherwise, the harvested power is sufficient to drive the NVP system.

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.

5.2.2 Power Analysis

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 be guaranteed to support others which consume more power.

These two concerns can be eased by calculating the discharging power of the capacitor. This is because 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 equals the changes of the harvesting power. So given current ongoing execution of $Tsk[cur]$, the calculation of discharging power on the capacitor is able to estimate how long (T^S) the system can be in execution before the outage happens, which is used as the input of the TCE Algorithm.

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 T^S for the system to work before the outage happens. The estimation also comes with the trustiness $Trust^P$ evaluation.

Algorithm 5.3 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 $Trust^P$;

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 K. If the real discharging power is reduced (harvesting power recovers), DPA gives more weight to estimation, otherwise, DPA 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 with 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 N-VP scheduler to use as references to conduct task scheduling for progress maximization.

5.3 On-Demand Dual-Backup Differential Checkpoint Handler (OD^2CH)

In this section, an On-Demand Dual-Backup Differential Checkpoint Handler (OD^2CH) is proposed to ensure successful checkpointing. During checkpointing, all registers will be pushed into the stack as described in Section 3. Then, all contents in the stack will be checkpointed to NVM.

To conduct OD^2CH, the energy stored in the storage capacitor is capable of supporting one successful checkpointing. Once $V < V_{ck}$ has been detected by the voltage monitor, checkpointing handler will be waken by the voltage monitor to conduct checkpointing. V_{ck} is crucial to ensure a successful checkpointing and is calculated in Eq. 1

$$V_{ck} = \sqrt{\frac{2E_{stack}}{C} + V^2_{safe}} \quad (1)$$

where E_{stack} is the energy required to checkpoint the entire stack region. V_{safe} is an safe voltage above minimum input voltage of the regulator. Once the checkpoint is finished, V_{cap} should be no less than V_{safe}. As the stack region is predefined in most embedded systems, E_{stack} is a constant, so are C and V_{safe}. Therefore, V_{ck} can be measured off-line. The parameters are selected for the worst-case scenario, which ensures the success of checkpointing.

Two areas in NVM should be reserved for OD^2CH to conduct checkpointing alternatively in case of a failed checkpoint. 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. During each checkpointing, OD^2CH also compares the new data with the old backup and only checkpoints the differences. This is to minimize the energy consumption of checkpointing.

After checkpointing, the Routine Handler will trigger sleep event for the NVP system.

5.4 Routine Handler

In this section, a routine handler is proposed to handle sleep and wake-up events of the embedded system.

5.4.1 Sleep Event

Once the checkpoint handler saves the system states into NVM, the routine handler will put the system into sleep mode other than continuing execution or shutting down the entire system. There are mainly three reasons that make sleep mode a better option than the other two.

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 NVP system to recover from sleep mode than cold reboot after being shut down.
3) Once the system is dead, it will automatically restarts 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.

5.4.2 Wake-up Event

Once power comes back on again, the routine handler will wake up the NVP system given a preset wake-up voltage V_{wak}, which should at least support one checkpointing and a period of execution. Notice that, if the system is dead and once power comes back on again, the NVP system will initiate the cold start with very low voltage. In this case, if the harvesting power is insufficient, the NVP system 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, 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.

6. Experiments

6.1 Experimental Setup

In this section, details of experimental setups on hardware and software are described.

6.1.1 Hardware Platform

The experiment platform of NVP embedded system is TIs 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 is able to 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.

6.1.2 Power Traces

To evaluate the performance of the NVP scheduler, five different power traces are generated as shown in Figure 7.

The source power is provided by MSP430FR5969 evaluation board which is programmed to generate different power traces through a GPIO pin which can provide $3.6V$ 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 seconds in every 1.3 seconds period; Power source 5 is for the output pin to set high all the time.

6.1.3 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 round-robin scheduler. For NVP scheduler, the Voltage Monitor keeps monitoring the voltage of the capacitor. Specifically, for every T_{ADC} time period, the timer will wake up the ADC module for a short period of time 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, NVP scheduler will take over task scheduling for progress maximization. Before presenting experimental results, experimental parameters are listed in Table 2.

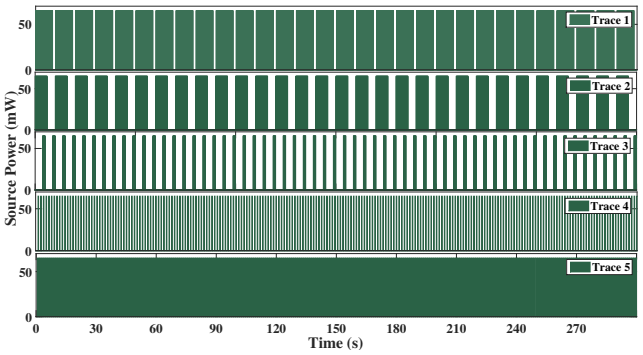

Figure 7. Power Trace Used for Experiments

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 Un_CK 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 CK tasks including register operations, SRAM writes, and FRAM writes. These six benchmarks are iterative and independent to each other.

6.2 Experimental Results

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 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 remain inactive.

6.2.1 Energy Consumption Analysis

The power consumption of the six benchmarks is measured in working mode with a stable power supply. Figure 8 shows the measurements.

All benchmarks are iterative and the duration of each iteration is trivial and negligible. 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 of voltage on the capacitor are negligible. So it is reasonable to use $\mu J/Iteration$ representing the power consumption of each tasks.

6.2.2 Sleep Mode vs. Working Mode

The first experiment is to show the power consumption of NVP system in sleep mode. In Figure 9, we adjust the Vcc from 3.7 to 1.4 gradually and measure the input current.

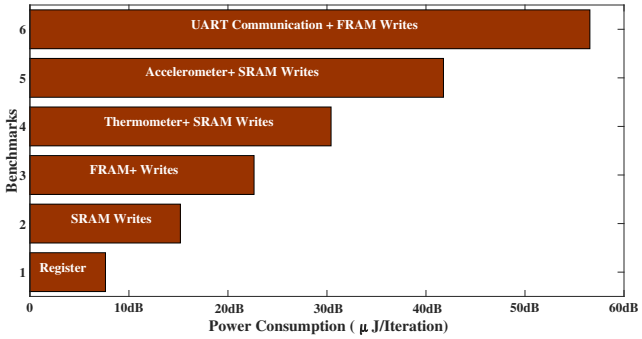

Figure 8. Power Consumption of Benchmarks

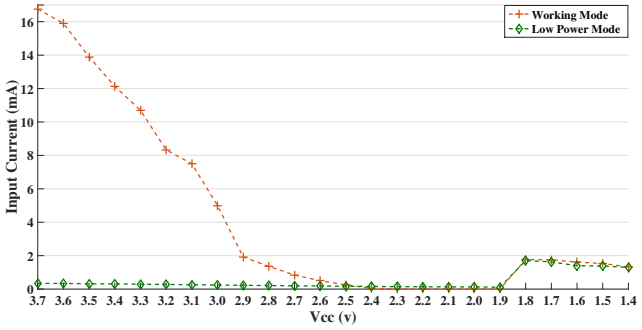

Figure 9. Power Consumption of Low Power Mode vs Working Mode

As we can see from Figure 9, 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.

6.2.3 Progress Comparison

Figure 10-15 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.

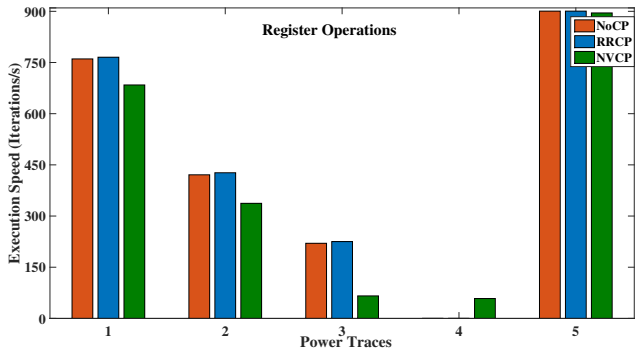

Figure 10. Execution Speed of Register Operation

Figure 10 shows the execution speed of register operation. As we can see from trace 5, when power is sufficient and stable, 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 6.2.4. 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 Un_CK tasks. The execution speed of CK task 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 NVP system to stuck at cold starting once the stored energy has been drained out.

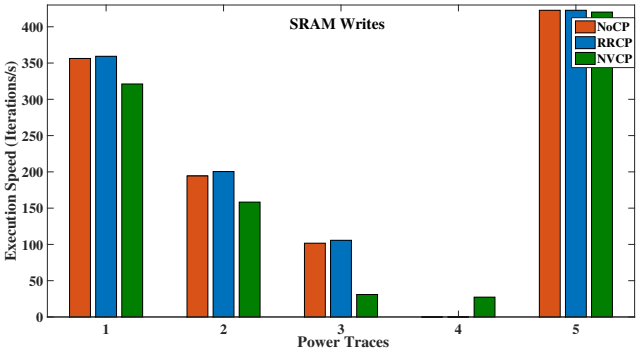

Figure 11. Execution Speed of SRAM Writes

Figure 11 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 Un_CK 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 $Iterations/s$.

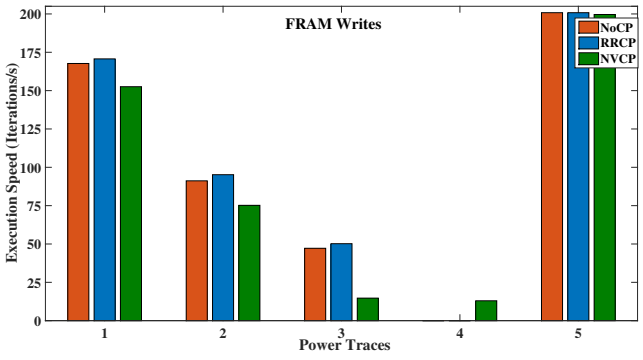

Figure 12. Execution Speed of FRAM Writes

Figure 12 shows the execution speed of FRAM writes. As we can see, similar to SRAM writes, the CK 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. And similarly, under the worst-case scenario of power trace 4, NoCP and RRCP shows no progress and NVCP shows the execution speed of 12.98 $Iterations/s$.

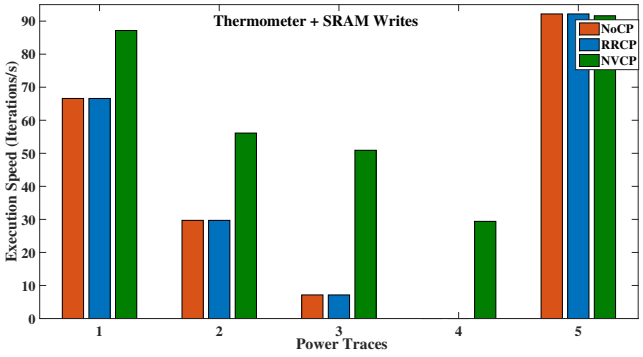

Figure 13. Execution Speed of Thermometer

Figure 13 shows the execution speed of Thermometer which samples proximity temperature and transfers it into digital representations. Thermometer is an Un_CK task as the unfinished temperature data is inaccurate and useless. Therefore, when source power becomes weaker, more energy portions will be given to uncheckponitable tasks in an effort 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 Un_CK tasks, so the execution speeds are the same. However, NVCP assigns a large portion of energy for Un_CK tasks and the execution speed of NVP is 50.92 $Iterations/s$ which is 712.77% of what baseline can deliver. Under the worst-case scenario of trace 4, NoCP and RRCP cause NVP system to stuck at the cold starting, hence no progress is made. Yet NVCP shows the speed of 24.41 $Iterations/s$.

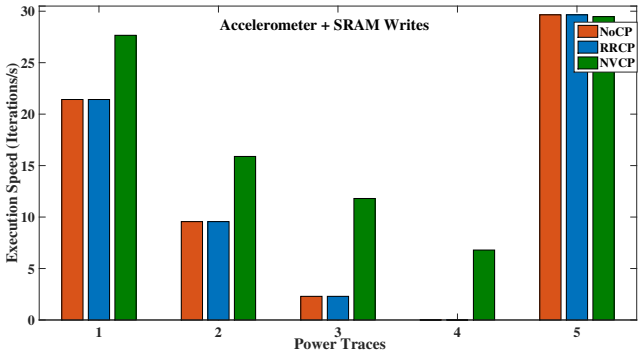

Figure 14. Execution Speed of Accelerometer

Figure 14 shows the execution speed of Accelerometer which samples 3-axis accelerations and transfers them into digital representations. Similar to Thermometer, Accelerometer is also an Un_CK task and the unfinished sensing data are inaccurate and useless. Therefore, when source power becomes weaker, a larger part of 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. And under power trace 4, only NVCP shows the execution speed of 6.79 $Iterations/s$, no progress is made by NoCP and RRCP.

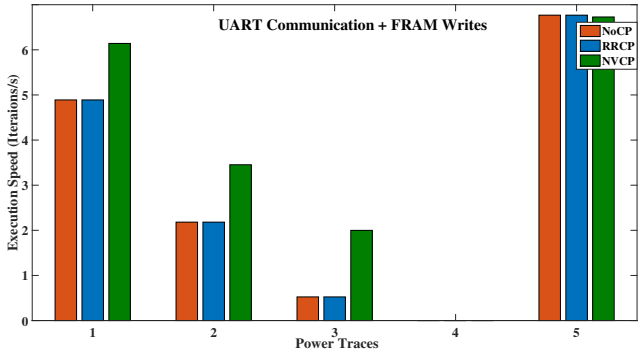

Figure 15. Execution Speed of UART communication

Figure 15 shows the execution speed of UART communication which constantly send data through serial port given the baud rate of 9600. Similar to the other two Un_CK task, 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 energy will be used to execute UART communication in an effort 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. Notice that under power trace 4 where the harvesting power is minimum and sparse, there is no progress been made by all techniques. This is because, UART consumes a significant amount of energy which is more than the stored energy can provide. Hence NVCP lowers down its priority and assigns gives more priorities to other tasks which is more likely to finish before the power fails.

6.2.4 Efficiency and Overhead

In this section, the efficiency and overhead of NVCP is evaluated. Considering the power consumption of each benchmark in Figure 8, 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 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

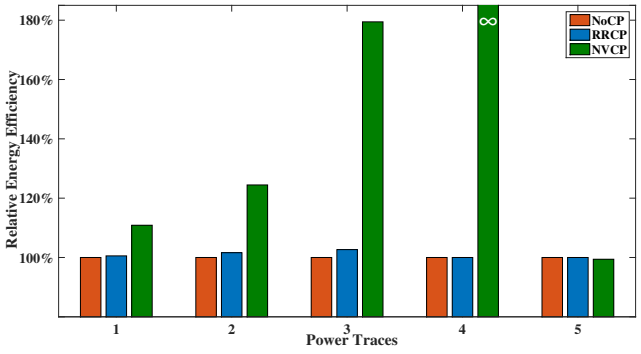

Figure 16. Energy Efficiency

RRCP. For trace 4, since NoCP and RRCP cause NVP system to stuck at the cold starting, the improved energy efficiency made by NVCP over the baselines is infinite. For trace 5 NVP scheduler has not been triggered, there is no obvious difference in terms of energy efficiency. Overall, the advantages are significant.

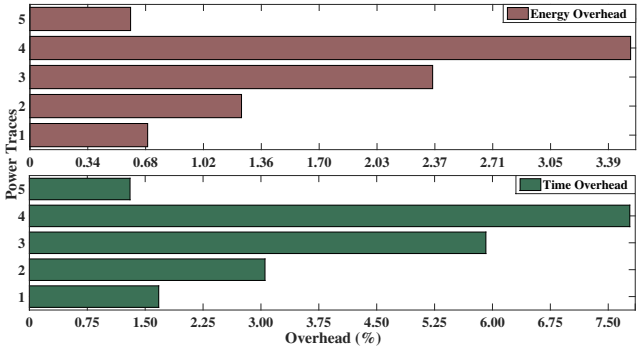

Figure 17. Energy and Time Overhead

Due to the periodical voltage monitor, there are extra amounts of energy and time overhead compared with baseline techniques. From Figure 17, both energy and time overhead increase when the source power becomes weaker. For the worst-case scenario in power trace 3, NVP scheduler generates 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.

7. Conclusion

This paper proposes a progress maximization multi-tasking scheduler, known as NVP scheduler, assisted with three functional modules including Voltage Monitor, Routine Handler, and Checkpoint Handler to maximize task progress of embedded system when power supply is frequently interrupted. The proposed NVP scheduler can be easily incorporated into embedded systems as an auxiliary to existing lightweight scheduler or OS with great compatibility which take effect when harvesting power is not enough. Experiments confirm the effectiveness of the proposed techniques.

Acknowledgments

This paper is partially supported by National Science Foundation CNS-1464429, CCF-1527506, and CCF-1615475. This paper is also supported by the National 863 Program 2015AA015304 and the Research Grants Council of the Hong Kong Special Administrative Region, China (No. CityU 11214115).

References

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

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

[3] `http://sensor.cs.washington.edu/WISP.html`.

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

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

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

[7] 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.

[8] H. Nakamoto, D. Yamazaki, et al. 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.

[9] 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.

[10] 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.

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

[12] 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.

[13] 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.

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

[15] 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.

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

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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. ISBN 978-3-9815370-4-8.

[23] 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.