Modeling and Optimization for Self-powered Non-volatile IoT Edge Devices with Ultra-low Harvesting Power

CHEN PAN and MIMI XIE, University of Pittsburgh, USA

SONG HAN, University of Connecticut, USA

ZHI-HONG MAO and JINGTONG HU, University of Pittsburgh, USA

Energy harvesters are becoming increasingly popular as power sources for IoT edge devices. However, one of the intrinsic problems of energy harvester is that harvesting power is often weak and frequently interrupted. Therefore, energy harvesting powered edge devices have to work intermittently. To maintain execution progress, execution states need to be checkpointed into the non-volatile memory before each power failure. In this way, previous execution states can be resumed after power comes back again. Nevertheless, frequent checkpointing and low charging efficiency generate significant energy overhead. To alleviate these problems, this article conducts a thorough energy efficiency analysis and proposes three algorithms to maximize the energy efficiency of program execution. First, a non-volatile processor-aware task scheduling algorithm is proposed to reduce the size of checkpointing data. Second, a tentative checkpointing avoidance technique is proposed to avoid checkpointing for further reduction of checkpointing overhead. Finally, a dynamic wake-up strategy is proposed to wake up the edge device at proper voltages where the total hardware and software overhead is minimized for further energy efficiency maximization. The experiments on a real testbed demonstrate that, with the proposed algorithms, an edge device is resilient to the extremely weak and intermittent power supply and the energy efficiency can be achieved more than $2\times$ higher than the fundamental baseline and $1.5\times$ higher than the state-of-the-art technique.

CCS Concepts: • **Computer systems organization** → **Embedded software**; • **Software and its engineering** → **Scheduling**; **Embedded software**; • **Hardware** → *Renewable energy*; *Power conversion*; *Hardware-software codesign*;

Additional Key Words and Phrases: IoT, edge devices, non-volatile processor, transient computing

ACM Reference format:

Chen Pan, Mimi Xie, Song Han, Zhi-Hong Mao, and Jingtong Hu. 2019. Modeling and Optimization for Self-powered Non-volatile IoT Edge Devices with Ultra-low Harvesting Power. *ACM Trans. Cyber-Phys. Syst.* 3, 3, Article 32 (August 2019), 26 pages.
<https://doi.org/10.1145/3324609>

This article is partially supported by National Science Foundation CNS-1830891.

Authors' addresses: C. Pan, M. Xie, Z.-H. Mao, and J. Hu, University of Pittsburgh, Pittsburgh, PA 15216; emails: {chen.pan, mix39}@pitt.edu, maozh@engr.pitt.edu, jthu@pitt.edu; S. Han, University of Connecticut, Storrs, CT 06269; email: song.han@uconn.edu.

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

© 2019 Association for Computing Machinery.

2378-962X/2019/08-ART32 \$15.00

<https://doi.org/10.1145/3324609>

1 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 the IoT network, which bridges the physical and cyber worlds 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.

While battery power is not a favorable solution in the long run due to size, longevity, safety, and recharging concerns, energy harvesting, 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 energy harvesting system is usually composed of an energy harvester, a power regulator, and a storage capacitor. 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 system.

However, an intrinsic common 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 cause significant progress setback if a program’s intermediate execution state is not saved. With the help of non-volatile processors (NVP) [1–4], both the program’s execution state and active contents in its stack can be saved into non-volatile memory (NVM) [5–7] before each power outage. This process is known as checkpointing. The next time the input 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, if the system is not carefully optimized, then energy overhead on both software and hardware will undermine limited energy utilization. On the software side, checkpointing mainly involves writing to NVM, which is both time- and energy-consuming. Therefore, a large size of checkpointing data and high checkpointing frequency will cause significant time and energy overhead. On the hardware side, once the stored energy is insufficient, the system has to wait until the storage capacitor is charged. The charging efficiency drops significantly when the voltage on the capacitor is approaching the charging voltage. Thus, the higher the capacitor’s target charging voltage, the more energy would be wasted for charging.

To improve the execution progress, overhead on both software and hardware needs to be reduced. On the software side, the checkpointing overhead can be reduced by reducing both the checkpointing data size and the checkpointing frequency. There are two solutions to reduce checkpointing frequency. First, the system can avoid unnecessary checkpointing if the harvesting power is sufficient to charge up the capacitor when the edge device is in sleep mode. Second, the system can increase the amount of stored energy on the capacitor before waking up so that program execution can last longer. However, increasing the stored energy means that the system has to set a higher threshold voltage for the capacitor, which reduces the charging efficiency and increases hardware overhead.

On the hardware side, reducing the capacitor’s target voltage can improve charging efficiency. As a result, the system will wake up earlier. However, due to the low voltage on the capacitor, only a small amount of energy can be harvested and stored. In this case, the system will soon drain out the stored energy. It then conducts checkpointing and goes to sleep or shuts down again. This makes checkpointing happen even more frequently, which incurs extra software overhead to the system. Hence, there is a performance tradeoff between the reduction of hardware and software

overhead. Thus, hardware and software joint optimization has to be considered to improve energy utilization efficiency.

In a nutshell, the major contributions of this article include the following:

- (1) An NVP-aware task scheduling (NTS) algorithm is proposed to reduce the software overhead through the reduction of checkpointing data size;
- (2) A tentative checkpointing withdrawal (TCW) algorithm is proposed to avoid unnecessary checkpointing for further reduction of software overhead;
- (3) A dual wake-up procedure (DWP) is proposed to balance between hardware and software overhead for further energy efficiency maximization.

The remainder of this article is organized as follows. Section 2 describes the related work of this article. Section 3 presents the architecture of the self-powered edge device and Section 4 gives examples that motivate this work. Then, Section 5 models the system and conducts energy efficiency analysis. After that, in Section 6, three techniques are presented to maximize execution progress. Section 7 presents performance evaluation results. Finally, Section 8 concludes this work.

2 RELATED WORK

Energy harvesting systems extract power from the ambient environment and are often used to deploy long lifetime battery-less devices. Solar, wind, footsteps, breathing, blood pressure, and body heat [8–10] 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 $15\text{mW}/\text{cm}^2$. The footstep is controllable human power, and the amount of harvested power can be as much as 67W [11] during a brisk walk. For ultra-low-power devices, the sources with low power densities, such as breathing (0.42W) and body heat ($2.4 \sim 4.8\text{W}$), can provide sufficient power to drive the devices at low duty cycles [12]. With proper configuration and management [13], it is feasible to operate a whole system with purely harvested energy.

Since many energy harvesters, such as TEG [14, 15] and Piezo [16, 17], can only generate ultra-low harvesting power, maximizing their energy efficiency is crucial to ensure the performance of the powered embedded devices. To improve the energy efficiency of the energy harvesting system, researchers mainly focus on hardware design. Some works, such as References [18] and [19], focus on designing low-power boost regulators that boost the harvesting energy efficiency with ultra-low harvesting power supply. Others, such as References [20] and [21], focus on improving the energy conversion rate of the energy harvesters. While exploiting the hardware design of energy harvester and power regulator can greatly improve the energy efficiency of the entire energy harvesting system, it is equally important to reduce the energy consumption of the program execution.

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 [22–27] and software [28–32] 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. [22] propose a non-volatile processor architecture that integrates non-volatile elements into volatile memory at bit granularity. Wang et al. [23] designed a FRAM-based processor that attaches an NV-FRAM cell to each volatile standard flip-flop. The flip-flops are accessed

Fig. 1. System architecture of self-powered non-volatile IoT edge device.

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 $3\mu\text{s}$. Sakimura et al. propose the non-volatile magnetic flip-flops [24] and a 20MHz non-volatile micro-controller with STT-RAM [25]. Recently, Liu et al. propose an enhanced NVP based on ReRAM that has the highest integration level [26]. Also, Li et al. propose the non-volatile I/O (NVIO) enabling efficient automatic reconfiguration of I/O interfaces [27].

Besides hardware checkpointing, there are also software mechanisms that checkpoint the processor' state and other volatile data into non-volatile memories. For example, Mementos [32] 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 runtime, when these trigger points are reached, the supply voltage is checked with an ADC. If this voltage is below a threshold, then a snapshot of the system state is saved to the flash memory. Quickrecall [28] integrates FeRAM into main memory to increase the checkpointing efficiency, which reduces the backup data size and lowers the failure voltage threshold. Hibernus [33] and Hibernus++ [33] propose interrupt-based checkpointing mechanisms. In these mechanisms, the system state 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.

In this article, upon applying the basic software checkpointing technique, we propose three software techniques that reduce the energy overhead of software checkpointing and improve energy efficiency for charging the storage capacitor. To our best knowledge, this is the first work that provides lightweight and effective software solutions to improve the energy efficiency of the self-powered IoT edge device with the ultra-low harvesting power supply.

3 ARCHITECTURE

In this section, we present a general architecture of the self-powered non-volatile IoT edge device (Figure 1).

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), and so on. 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

Fig. 2. Changes of charging efficiency and voltage on the edge device with ultra-low harvesting power.

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.

4 MOTIVATIONAL EXAMPLES

In this section, we will present examples to illustrate the motivation of this work.

4.1 Wake-up Voltage Determines Efficiency

Figure 2 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 Figure 2, for the storage capacitor, there is always a negative correlation between the charging efficiency and the voltage. Therefore, achieving a higher wake-up voltage V_w is time-consuming with poor energy efficiency.

Figure 3 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} , then 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} , then 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

Fig. 3. Influence of wake-up voltage on charging cycle and efficiency of the edge device with ultra-low harvesting power.

Fig. 4. Influence of the wake-up voltage on checkpointing frequency of the edge device with ultra-low harvesting power.

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 7.2.4 show that with appropriate setting of wake-up voltages, the charging efficiency can remain in the good efficiency zoom, which significantly improves 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. Figure 4 shows the influence of the wake-up voltage on the checkpointing frequency.

Based on Figure 3, 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 improvement of charging (hardware) efficiency. Since the wake-up voltage determines both aspects, an appropriate wake-up voltage that strikes an optimal tradeoff between these two optimization goals should be determined so that the overall energy efficiency

Fig. 5. Measurements of the program execution progress with different working voltages.

can be maximized. With the same experimental setup to be described in Section 7.1, Figure 5 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.

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, then the energy efficiency is significantly increased, which is 41 iterations. Therefore, between 2.46V and 3.98V, there should be an optimal solution that achieves the maximum energy efficiency.

Aside from choosing the optimal wake-up voltage, minimizing the size of checkpointing data can further reduce the checkpointing overhead. Section 6 will present the methods for minimizing the checkpointing data size and the calculation of optimal wake-up voltage.

4.2 Sensing Task Case Study

In this subsection, a real-life application case study is presented to illustrate the importance of wake-up voltage selection.

The target application scenario is a wireless sensor network where several sensor nodes powered by intermittent power supply need to conduct environmental monitoring. Each sensor node carries out sensing tasks on multiple monitoring targets that include temperature, humidity, light intensity, air pressure, and CO₂ level. There is no dependency or priority among tasks. There is, however, a time restriction that one round of all sensing tasks should be finished within 5 minutes. When the harvesting power is too weak to support sensor nodes to work, the capacitor should be charged first. Then the sensor node will be woken up; the sensing tasks will be started with the reserved energy in the capacitor. When the voltage of the capacitor drops down to the threshold, the execution contents will be checkpointed with the remaining energy.

In a traditional charging procedure as shown in Figure 6, the capacitor needs to be charged up to a high enough voltage: E_{max} , so that the harvested energy is sufficient to support one round of all sensing tasks. However, when the power is extremely weak, it takes more than 5 minutes for the capacitor to be charged up to the targeting voltage. In this case, no task will be able to meet the 5-minute time constraint.

Instead of waiting for a long time to charge up the capacitor to a very high voltage, which may take longer than 5 minutes, the sensor node can wake up much earlier with less but enough harvested energy for executing a single task (*Method 1*). Then it goes to sleep and executes the next task when the harvested energy is enough again. Figure 7 illustrates how reducing wake-up

Fig. 6. Traditional charging procedure may not be able to meet the time constraint.

Fig. 7. Reducing wake-up voltage enable sensing tasks to meet the time constraint.

voltage enables sensing tasks to meet the time constraint. In this figure, E_{Tmp} , E_{Hum} , E_{Lg} , E_{AP} , and E_{CO_2} indicate the energy needed for executing tasks *Tmp*, *Hum*, *Lg*, *AP*, and *CO₂*, respectively. E_{ck} indicates the energy for checkpointing. Each time, the capacitor is charged up to a low wake-up voltage level that is enough to complete a specific sensing task. As a result, all five tasks can finish within 5 minutes. This is because charging to lower wake-up voltage achieves higher charging efficiency, and thus there is more energy for sensing as discussed in Section 4.1.

Despite the superiority of *Method 1* over traditional charging procedures, the performance of this method still has room for improvement. This is because waking up the system when the power is just enough to execute a single task increases the checkpointing frequency, which introduces a significant amount of checkpointing energy overhead. To solve this problem, instead of harvesting enough energy to execute only one task, in this example, the capacitor is charged up to a voltage level for executing two tasks (*Method 2*). Figure 8 shows how *Method 2* further improves the energy efficiency while meeting the time constraint. Although the charging efficiency of *Method 2* is lower than *Method 1*, the checkpointing overhead is reduced. This is because *Method 1* introduces 5 times the checkpointing (Figure 7), while *Method 2* only introduces 3 times the checkpointing (Figure 8). Since all five tasks in a single iteration can finish much earlier than *Method 1*, the overall program progress is increased, which means the overall energy efficiency is improved. Therefore, *Method 2* has the highest total energy efficiency among the above three charging methods.

Fig. 8. Further improve energy efficiency while meeting the time constraint.

Based on the above analysis and case study, it is easy to see the necessity of choosing an appropriate wake-up voltage, so that the overall execution efficiency can be maximized.

5 ENERGY MODELING AND EFFICIENCY ANALYSIS

In this section, energy models of the edge device with ultra-low harvesting power will be built first. Then, based on these energy models, the analysis is conducted on how to achieve maximum energy efficiency for the edge device.

5.1 Energy Modeling

5.1.1 Harvested Energy. Given the harvesting power of $P_H(t)$ and the harvesting duration τ , the harvested energy $E_H(\tau)$ can be defined as in Equation (1),

$$E_H(\tau) = \int_0^{\tau} P_H(t) dt. \quad (1)$$

5.1.2 Stored Energy. Given minimum working voltage of power converter as v_{min} , the current voltage on the capacitor as v , and the capacitance C , the stored energy $E_{cap}(v)$ on the capacitor can be defined as in Equation (2),

$$E_{cap}(v) = \frac{C}{2} (v^2 - v_{min}^2). \quad (2)$$

5.1.3 Checkpointing Energy. Given the energy consumption for checkpointing a data unit from volatile memory to non-volatile memory as $E_{ck}^{(1)}$ and the size of the data that need to be checkpointed as ω , the energy $E_{ck}(\omega)$, which is required to conduct checkpointing, is defined as in Equation (3),

$$E_{ck}(\omega) = \omega E_{ck}^{(1)}. \quad (3)$$

5.1.4 Wake-up Energy. Given the energy E_{rs} for resuming the checkpointed data and the energy E_{act} for reactive the edge device from sleep mode into active mode, the total energy consumption E_{wak} for waking up the edge device is defined as in Equation (4),

$$E_{wak} = k E_{rs} + E_{act}. \quad (4)$$

Here, $E_{rs}(\omega) = \omega E_{rs}^{(1)}$, where $E_{rs}^{(1)}$ represents the energy consumption for resuming a data unit. k is a binary and if the edge device is dead before waking up, then $k = 1$; otherwise, $k = 0$. E_{act} is determined by the number of hardware modules on edge device that needs to be activated.

Fig. 9. Capacitor charging circuit.

The more modules that has been deactivated during the sleep mode the larger the E_{act} will be to reactivate these modules.

5.2 Energy Efficiency Analysis

In the edge device, assume that the working power is P_W and the harvesting power is P_H . Since the harvesting power is ultra low, it is assumed that $P_W \gg P_H$. Under such a circumstance, the edge device has to execute intermittently and the working period γ and harvesting period τ .

During a charging period, the utility rate of the energy harvester drops when the voltage on the capacitor rises, which will increase the energy overhead on hardware. During a working period, because $P_W \gg P_H$, the amount of the harvested energy is trivial and negligible compared with the amount of stored energy in the capacitor for program execution. Aside from being used for program execution, checkpointing also consumes a portion of the stored energy, which makes up the energy overhead on software. Therefore, energy efficiency on both hardware and software can be significantly be improved. The following provides detailed modeling and analysis.

5.2.1 Hardware Energy Efficiency. The charging circuit of the self-powered IoT edge device can be simplified as shown in Figure 9. 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 can be considered as an ideal voltage source that connects the inner resistance R_N . The ideal voltage source maintains a constant voltage that 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 5.1 is proposed.

THEOREM 5.1. *Given $P_H \ll P_W$ and τ as the charging period, the hardware energy efficiency ξ^H can be formulated in Equation (5),*

$$\xi^H = E_{cap}(V)/E_H(\tau), \quad (5)$$

where

$$\begin{cases} E_{cap}(v) = C(v^2 - v_{min}^2)/2 \\ E_H(\tau) = \int_0^\tau P_H(t)dt. \end{cases} \quad (6)$$

The proof the Theorem 5.1 is given in the Appendix, which explores the influence of the relationship between P_W and P_H to the accuracy of the formulation.

In Equation (6), 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 Equation (7) shows the calculation of P_H ,

$$P_H(t) = v_{max}^2/R_N(t). \quad (7)$$

Then, according to Kirchhoff Voltage Law, the parameters in Figure 9 have the following relation:

$$v_{max} - v = C \frac{dv}{d\tau} (R_N + R_C). \quad (8)$$

Since $R_N \gg R_C$, R_C can be omitted in Equation (8). Therefore, based on Equation (7), we further have

$$v_{max} - v = C \frac{dv}{d\tau} \frac{v_{max}^2}{P_H}. \quad (9)$$

Hence, $E_H(\tau)$ can be further transformed into Equation (10),

$$E_H(\tau) = C v_{max}^2 \left[\ln \left(\frac{v_{max} - v_{min}}{v_{max} - v} \right) \right]. \quad (10)$$

Considering Equation (6) and Equation (10) together, the hardware energy efficiency ξ^H in Equation (5) can be further transformed into the function of v in Equation (11),

$$\xi^H(v) = \frac{(v^2 - v_{min}^2)/2}{v_{max}^2 [\ln(v_{max} - v_{min}) - \ln(v_{max} - v)]}. \quad (11)$$

5.2.2 Software Energy Efficiency. Based on the aforementioned analysis, the software energy efficiency ξ^S can be calculated as in Equation (12),

$$\xi^S(v, \omega) = (E_{cap}(v) - E_{ck}(\omega) - E_{wak})/E_{cap}(v). \quad (12)$$

5.2.3 Execution Energy Efficiency. Considering ξ^H and ξ^S together, the overall energy efficiency of program execution can be calculated as in Equation (13),

$$\xi(v, \omega) = \frac{(v_{exe}^2 - v_{ck}^2)/2}{v_{max}^2 [\ln(v_{max} - v_{min}) - \ln(v_{max} - v)]}, \quad (13)$$

where

$$\begin{cases} v_{ck} = \sqrt{2E_{ck}^{(1)}(\omega)/C + v_{min}^2} \\ v_{exe} = \sqrt{v^2 - 2E_{wak}/C}. \end{cases} \quad (14)$$

Here, v_{ck} represents the voltage for starting checkpointing and v_{exe} represents the voltage for starting program execution. With Equation (13), we can analyze the influences of both ω and v on ξ .

5.2.4 Influence of Checkpointing Data Size ω . Take a derivative of ξ with respect to ω , then we have Equation (15),

$$\frac{\partial \xi(v, \omega)}{\partial \omega} = \frac{-(E_{ck}^{(1)} + E_{rs}^{(1)})/C}{v_{max}^2 [\ln(v_{max} - v_{min}) - \ln(v_{max} - v)]} < 0. \quad (15)$$

Therefore, ω should be as small as possible to maximize the execution energy efficiency. Section 6.1 provides a task scheduling method to reduce the checkpointing data size by reducing the number of executing tasks during the working cycle.

5.2.5 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 5.2. *The optimal solution v_{op} exists.*

THEOREM 5.3. *For a given ω , v_{op} is unique.*

Fig. 10. System overview.

Proofs of Theorems 5.2 and 5.3 are given in the Appendix. These theorems will be further used as the theoretical bases to maximize the energy efficiency in Section 6.

6 ENERGY EFFICIENCY MAXIMIZATION WITH ULTRA-LOW HARVESTING POWER

In this section, three techniques are proposed to maximize the energy efficiency of the edge device. The system overview is shown in Figure 10. At first, during the execution period, an NTS is proposed to improve the execution energy efficiency by finishing atomic tasks that cannot be interrupted before checkpointing and reducing the size of checkpointing data. Then, when power outages happen ($v < v_{ck}$), a TCW method is proposed allowing the edge device to tentatively withdraw the checkpointing for further reduction of checkpointing overhead. Finally, when the stored energy recovers ($v < v_{wak}$), a dual wake-up procedure (DWP) is proposed to optimize the wake-up voltage (v_{wak}) for further energy efficiency maximization.

6.1 NVP-aware Task Scheduling

In this section, an NVP-aware task scheduling technique is proposed to 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 NVP tasks 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. Before introducing the details of NTS, the following unique features of the self-powered IoT edge device needs to be considered:

- Not all tasks' execution states need to be pushed into stacks and therefore do not need to be checkpointed [35]. 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 number of unfinished non-atomic tasks during each execution cycle should be as small as possible.
- 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.

ALGORITHM 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}

```

1: Build up unscheduled task tree set  $\langle tree \rangle$  based on DAG and  $sch$ ;
2: Calculate energy storage  $E$  based on  $v_{wak}$ ;
3: if previous task  $tsk_p$  has been checkpointed then
4:    $E = E - e_p$ ;
5: end if
6:  $\Psi = 0$  /* indicator of completing scheduling */
7: for each  $tree_i$  do
8:    $d_i = t^{cur} + t_i$ ; /* deadline of  $tree_i$  */
9: end for
10: Sort trees in ascending order of  $d_i$ ;
11: for each  $tree_i$  do
12:   for each leaf task  $tsk_i^j$  in  $tree_i$  do
13:     if  $E > e_i^j$  then
14:       Add  $tsk_i^j$  to  $tsk_{nx}$  and  $sch$ ;
15:       Remove  $tsk_i^j$  from  $tree_i$ ;
16:       Update  $E = E - e_i^j$ ;
17:     else
18:       for each leaf task  $tsk_i^k$  in  $tree_i$  do
19:         if  $tsk_i^k$  is a non-atomic task then
20:           Add  $tsk_i^k$  to  $tsk_{nx}$  and  $sch$ ;
21:           Remove  $tsk_i^k$  from  $tree_i$ ;
22:           Update  $E = E - e_i^k$ ;
23:           Break;
24:         end if
25:       end for
26:        $\Psi = 1$ ;
27:       Break;
28:     end if
29:   end for
30:   if  $tree_i$  is empty then
31:     Release all tasks of  $tree_i$  from  $sch$ ;
32:   end if
33:   if  $\Psi = 1$  then
34:     Break;
35:   end if
36: end for
37: return  $tsk_{nx}$ ;

```

Aside from the above features, 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 has its own time constraint and energy consumption. 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 1 provides the details of the 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

Fig. 11. Example DAG.

in DAG and removes the scheduled tasks in sch from subtrees that 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 lines 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 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 completion of scheduling.

Before breaking out the scheduling process, NTS checks whether all tasks in $tree_i$ have been scheduled. If so, then 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 lines 30 to 37.

The example DAG in Figure 11 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 five 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 tasks. 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 first 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 first tree and selects D_1 for scheduling. The scheduled tasks in the first execution cycle are in black color.

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 first tree and leaves $80 - 50 = 30$ energy units for scheduling tasks on the second 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 the energy available in the capacitor. At this point, NTS starts searching for unscheduled non-atomic leaf tasks in the second tree and selects C_2 for scheduling. The scheduled tasks in the second execution cycle are in white color.

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 second tree is also 90 energy units, all tasks on the second tree can be scheduled. The scheduled tasks in the third execution cycle are in grey color.

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.

6.2 Tentative Checkpointing Withdrawal

In this subsection, a TCW algorithm is proposed that 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, then 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 2.

ALGORITHM 2: 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:  $\lambda = (\lambda + s_c)/2$ ; /* update the confidence of power goodness */
2: if  $s_c = 1$  then /* current wake-up is from sleep mode */
3:    $cs = (v_{wak}^2 - v_{ini}^2)/\tau$ ; /* calculate the charging speed */
4:   if  $s_p = 1$  then /* previous wake-up was from sleep mode */
5:     if  $cs \geq \lambda * \widehat{cs}$  then
6:        $out = 0$ ;
7:     else
8:        $out = 1$ ;
9:     end if
10:  else /* previous wake-up was from reboot */
11:    if  $\lambda * cs \geq \widehat{cs}$  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:  $\widehat{cs} = (cs + \widehat{cs})/2$ ; /* update the estimated charging speed */
22: return  $out$ 
```

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	$\lambda_{ini}=0.250$	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

Fig. 12. Four of nine checkpointings can be avoided without progress setback.

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 sleep mode; 0: rebooting from 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 $s_p = 0$, then TCW thinks that the harvesting power of current charging cycle is only sufficient temporarily and may become insufficient at any moment in the near future. In this case, the next checkpointed is less likely to be avoided than in the previous case. Finally, if the current wake-up state $s_c = 0$, then the 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 τ . Therefore, TCW conservatively forces the next checkpointing to be mandatory no matter how large *cs* actually is. An auxiliary example in Figure 12 details TCW.

Figure 12 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, and, 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 first 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 in the near future. In this case, TCW calculates *out* based on the criterion in line 11. Specifically, on one hand, if λ is small, then TCW is less confident that the current sufficient harvesting power could last long, and, hence *cs* needs to be higher to convince TCW to avoid checkpointing. On the other hand, if λ is large, then TCW is more confident that current sufficient harvesting power could last long, and, 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 in the near future. 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 second 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 $\widehat{cs}$, and 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 checkpointings. 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 third to eight charging cycles, TCW determines *out* following the same procedure as before. In the end, four of nine checkpointings 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. However, 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 previous computation state by failing to checkpoint. However, the sleep power can be infinitesimally small compared 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 μ J and the standby power of edge device in low power mode can be as low as 20 μ W. In the worst case, the edge device can last more than two minutes by totally sustaining on its own 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.

6.3 Dual Wake-up Procedure

In this section, a DWP is proposed that selects appropriate wake-up voltages for edge devices so that the overall energy overhead can be reduced.

According to the proposed theorems in Section 5.2.5, the optimal wake-up voltage v_{wak}^{opt} only determined by the checkpointing size ω . Since TCW in Section 6.2 withdraws unnecessary checkpointing, ω , in this case, is zero. Therefore, two optimal wake-up voltages, $v_{wak}^{opt}(ck)$ and $v_{wak}^{opt}(zero)$ for the ordinary checkpointing case and checkpointing avoidance case, will be referred to dynamically wake-up the edge device for achieving the maximum energy efficiency.

Given $v_{wak}^{opt}(ck)$, $v_{wak}^{opt}(zero)$, and the output of TCW *out* as inputs, the wake-up voltage is calculated according DWP as shown in Algorithm 3.

At the beginning, DWP checks if the previous checkpointing has been avoided by referring to the the output of TCW *out*. *out* = 1 indicates that the checkpointing has been conducted. In this case, $v_{wak} = v_{wak}^{opt}(ck)$ (line 2); otherwise, $v_{wak} = v_{wak}^{opt}(zero)$ (line 4). After that, while the system is in sleep mode, during each ADC cycle, ADC keeps sampling the voltage v on the capacitor until $v \geq v_{wak}$ by then, DWP wakes up the edge device and breaks out the while loop. This is shown from lines 8 to 11.

ALGORITHM 3: Dual Wake-up Procedure (DWP)

Input: $\widehat{v_{wak}^{opt}}(ck)$, $\widehat{v_{wak}^{opt}}(zero)$, and *out*
Output: Wake up the system when capacitor voltage is v_{wak}

```

1: if out = 1 then
2:    $v_{wak} = \widehat{v_{wak}^{opt}}(ck)$ ;                                /* initialize  $v_{wak}$  for checkpointed case */
3: else
4:    $v_{wak} = \widehat{v_{wak}^{opt}}(zero)$ ;                          /* initialize  $v_{wak}$  for checkpointing avoided case */
5: end if
6: while system is in sleep mode do
7:   if ADC is triggered then
8:     Sample capacitor voltage v;
9:     if  $v \geq v_{wak}$  then
10:      Wake up the system;
11:      Break;
12:   end if
13: end if
14: end while

```

Notice that, without sophisticated power evaluation mechanisms, DWP provides a simple, yet very effective, wake-up strategy that is especially suitable for an ultra-low self-powered IoT edge device. The key of DWP is that $\widehat{v_{wak}^{opt}}(ck)$ and $\widehat{v_{wak}^{opt}}(zero)$ should be measured precisely in order for DWP initiating a quick response for choosing an appropriate wake-up voltage. Due to the discrete nature of ADC sampling, there are only a limited number of voltages that we can measure. Therefore, at the vicinity of $\widehat{v_{wak}^{opt}}$ within $[-0.1, 0.1]V$, we obtain a near-optimal voltage $\widehat{v_{wak}^{opt}}$ and use it for experiments. The detailed measurements of $\widehat{v_{wak}^{opt}}$ and the validation of theorems will be described in Section 7.2.1.

7 EXPERIMENTS

7.1 Experimental Setup

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

7.1.1 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 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 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.

7.1.2 Power Trace. Four power traces in Figure 13 using signal generators for performance evaluation. The four power traces with different power magnitudes are recorded from oscilloscope at the sample rate of 240KSa/s. On average, the power they can provide is 7.27, 6.19, 4.94, and 4.11mW, respectively. The harvesting power is far less than the working power of edge devices.

Fig. 13. Power traces.

Fig. 14. Required energy of each benchmark.

7.1.3 Software Setup. The parameter settings of the following experiments includes $C = 470\mu\text{F}$, $v_{ck} = 2.1\text{V}$, $m = 10$, and $T_{ADC} = 37.5\mu\text{s}$. For experimental evaluation, five benchmarks, SRAM, FRAM, Thermometer, Accelerometer, and UART, are used. Their required energy is show in Figure 14.

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 10 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.

Fig. 15. Execution progress of benchmark SRAM with different wake-up voltages.

7.2 Experimental Results

In this section, a series of experiments is conducted. We first validate the proposed theorems in Section 5.2.5 and then evaluate the performance of the proposed algorithms accordingly by comparing with baselines.

7.2.1 Validation of Theorems. The experiments start with the validation of Theorem 5.2 and 5.3. The benchmark *SRAM* is chosen for evaluation where the executions with and without checkpointing will both be evaluated. Due to the discrete nature of ADC and the time complexity of validation, for each power trace, the wake-up voltage for the edge device is adjusted with the scale of 0.1V for every 10 minutes.

The top half of Figure 15 shows the execution with checkpointing. As we can see, the execution speed is severely affected by checkpointing overhead, which leads to a minimum execution progress when wake-up voltage is low. Once the wake-up voltage increases, the execution speeds up and reaches the maximum speed when waking up at 3.44V. However, when the wake-up voltage is approaching the charging voltage (4.2V), the execution speed keep dropping down, which is due to the reduction of charging efficiency of the capacitor. Also can be noticed is that despite the changes of source power magnitudes, the maximum execution speed is always constant with the corresponding wake-up voltage of 3.44V.

When checkpointing is avoided, the edge device only goes to sleep when the voltage drops below 2.1V. In this case, the maximum energy efficiency can be achieved when the wake-up voltage is 2.4V. This is because, without checkpointing, the dominant energy overhead is charging overhead and the charging efficiency keeps dropping down when the voltage is increasing. Under such circumstance, with a tiny amount of energy overhead for recovery, the maximum overall energy efficiency can be achieved when the edge device wakes up at the voltage of 2.4V.

For benchmark *SRAM*, the average execution power of the edge device is 29.18mW during the working period. Based on the above observations, when $\Psi = \frac{P_W}{P_H}$ is as low as 4, the optimal solution from the selected discrete wake-up voltage set still remains the same. Therefore, these observations validate the claims in Theorems 5.1, 5.2, and 5.3. These observations also imply that the proposed techniques are resilient and can be widely used in many energy harvesting scenarios to cope with different kinds of energy harvesting sources including in-door light, piezoelectric,

Fig. 16. Average single execution period with NTS.

radio-frequency, thermoelectric, and so on. Meanwhile, based on the observation, $\widehat{v_{wak}^{opt}(ck)} = 3.44V$ and $\widehat{v_{wak}^{opt}(zero)} = 2.40V$ will be used as wake-up voltage to evaluate the performance of Algorithm 3.

7.2.2 Performance of NTS. 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 Reference [34] where the whole system is defined as MEU (maximizing energy utility). Figure 16 shows the single execution period of each task trees with NTS vs. RRB and MEU (PTS).

In Figure 16, the experiments are represented in five regions that have different deadline pairs 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 that of Tree (therm). The deadline of 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 for a single execution period. For RRB, the deadlines of each task tree will not change the scheduling order, which is the same to MEU (PTS). For MEU (PTS), the priority is based on the stack size of each task that is ideal for scheduling independent tasks. 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. While with RRB and MEU (PTS), Tree (Therm) only fails under trace 4. 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 14, the average execution power of each benchmark under different power traces is shown in Figure 17.

Fig. 17. Average execution power of benchmarks under different power traces.

Fig. 18. Single execution period with TCW.

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 round-robin scheduler. The reason for NTS to have a higher efficiency is that both 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, then the on-going 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.

7.2.3 Performance of TCW. In this section, the performance of TCW will be evaluated upon applying NTS. TCW avoids unnecessary checkpointing to reduce 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. Figure 18 shows the single execution period of each task with TCW vs. NTS and MEU (TCA).

In Figure 18, 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 Figure 19.

With TCW, energy efficiency can be further improved by 77.95% on average compared with 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

Fig. 19. Average execution power of benchmarks under different power traces.

Fig. 20. Average execution power of benchmarks under different power traces.

Fig. 21. Single execution period with DWP.

unnecessary checkpointing. Overall, after applying TCW, more energy can be used for execution, which significantly improves energy efficiency.

7.2.4 Performance of DWP. In this section, the performance of DWP will be evaluated upon applying NTS and TCW. DWP is designed to adjust the wake-up voltage for maximizing energy efficiency. The baseline for comparison are TCW with the wake-up voltages $\widehat{v}_{wak}^{opt}(ck) = 3.44V$ and $\widehat{v}_{wak}^{opt}(zero) = 2.40V$, and dynamic wake-up strategy (DWS) in MEU. Figure 20 shows the single execution period of each task tree with DWP vs. TCW and MEU (DWS).

As we can see from Figure 20, upon implementing NTS and TCW, DWP further maximizes the energy efficiency and the improvements are salient. That is, DWP only results in one fail compared with five fails with only TCW and NTS. Thus, the improvement is significant.

From Figure 21, we can see that DWP further achieves energy efficiency by 54.03% on average upon TCW and NTS together. While MEU (DWS) only achieves 28.39% more energy efficiency on average upon MEU (TCA). The main reason for such improvement is that MEU (DWS) results in a large deviation from the optimal wake-up voltage and more computation overhead. Totally, the proposed techniques together can gain 273.05% energy efficiency improvements over RRB while

Fig. 22. Fail rates of each algorithm to meet the deadlines.

MEU scheme can only gain 49.42% energy efficiency improvements over RRB, which makes the proposed techniques $1.5\times$ more energy-efficient to implement under ultra-low harvesting power scenario than MEU. The improved energy efficiency can then be used to further maximize the execution progress.

The comparison of fail rates among all techniques is shown in Figure 22. As we can see from Figure 22, with the given four deadline pairs, DWP can lower down the fail rate to 2.5% while the fail rate of MEU is 30.0% and RRB is 57.5%, which are much higher the proposed techniques.

8 CONCLUSION

In this article, three lightweight and energy-efficient techniques are designed to improve the energy efficiency of the self-powered IoT edge device with the ultra-low-harvesting power supply. First, NTS is proposed to reduce the checkpointing data size, which helps reduce the checkpointing overhead. Then, TCW is proposed to avoid unnecessary checkpointing for further reduction of checkpointing overhead. Finally, DWP is proposed to balance hardware and software overhead for further energy efficiency maximization. The experiments on the real testbed show that the proposed algorithms can significantly improve the resilience of self-powered IoT edge devices to the extremely weak and intermittent power supply and the corresponding energy-efficiency can be achieved more than $2\times$ higher than the fundamental baseline and $1.5\times$ higher than the state-of-the-art technique.

REFERENCES

- [1] Mimi Xie, Chen Pan, Jingtong Hu, Chengmo Yang, and Yiran Chen. 2015. Checkpoint-aware instruction scheduling for nonvolatile processor with multiple functional units. In *Proceedings of the 20th Asia and South Pacific Design Automation Conference*. 316–321.
- [2] H. Nakamoto, D. Yamazaki, T. Yamamoto, H. Kurata, S. Yamada, K. Mukaida, T. Ninomiya, T. Ohkawa, S. Masui, and K. Gotoh. 2007. A passive UHF RF identification CMOS Tag IC using ferroelectric RAM in 0.35um technology. *IEEE J. Solid-State Circ.* 42, 1 (2007), 101–110.
- [3] Mimi Xie, Mengying Zhao, Chen Pan, Jingtong Hu, Yongpan Liu, and Chun Jason Xue. 2015. 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)*. 184:1–184:6.
- [4] C. Ding, N. Liu, Y. Wang, J. Li, S. Heidari, J. Hu, and Y. Liu. 2017. Multisource indoor energy harvesting for nonvolatile processors. *IEEE Des. Test* 34, 3 (Jun. 2017), 42–49. DOI: <http://dx.doi.org/10.1109/MDAT.2017.2682242>
- [5] Chen Pan, Mimi Xie, Chengmo Yang, Yiran Chen, and Jingtong Hu. 2017. Exploiting multiple write modes of non-volatile main memory in embedded systems. *ACM Trans. Embed. Comput. Syst.* 16, 4, Article 110 (May 2017), 26 pages. DOI: <http://dx.doi.org/10.1145/3063130>
- [6] C. Pan, S. Gu, M. Xie, Y. Liu, C. J. Xue, and J. Hu. 2016. Wear-leveling aware page management for non-volatile main memory on embedded systems. *IEEE Trans. Multi-Scale Comput. Syst.* 2, 2 (Apr. 2016), 129–142. DOI: <http://dx.doi.org/10.1109/TMSCS.2016.2525999>
- [7] C. Pan, M. Xie, C. Yang, Z. Shao, and J. Hu. 2015. Nonvolatile main memory aware garbage collection in high-level language virtual machine. In *Proceedings of the 2015 International Conference on Embedded Software (EMSOFT'15)*. 197–206. DOI: <http://dx.doi.org/10.1109/EMSOFT.2015.7318275>

- [8] Vladimir Leonov. 2013. Thermoelectric energy harvesting of human body heat for wearable sensors. *IEEE Sens. J.* 13, 6 (2013), 2284–2291. DOI : <http://dx.doi.org/10.1109/JSEN.2013.2252526>
- [9] Chulsung Park and Pai H. Chou. 2006. Ambimax: Autonomous energy harvesting platform for multi-supply wireless sensor nodes. In *Proceedings of the IEEE International Conference on Sensing, Communication and Networking (SECON'06)*. 168–177.
- [10] Thad Starner. 1996. Human-powered wearable computing. *IBM Syst. J.* 35, 3.4 (1996), 618–629.
- [11] John Kymissis, Clyde Kendall, Joseph Paradiso, and Neil Gershenfeld. 1998. Parasitic power harvesting in shoes. *Proceedings of the International Semantic Web Conference (ISWC'98)*, 132–139.
- [12] Jay Taneja, Jaemin Jeong, and David Culler. 2008. Design, modeling, and capacity planning for micro-solar power sensor networks. In *Proceedings of the International Conference on Information Processing in Sensor Networks (IPSN'08)*. 407–418.
- [13] Qadeer A. Khan and Sarvesh J. Bang. 2009. Energy Harvesting for Self Powered Wearable Health Monitoring System. Oregon State University.
- [14] Vladimir Leonov, Tom Torfs, Paolo Fiorini, and Chris Van Hoof. 2007. Thermoelectric converters of human warmth for self-powered wireless sensor nodes. *IEEE Sens. J.* 7, 5 (2007), 650–657.
- [15] Bert Gyselinckx, Julien Penders, and Ruud Vullers. 2007. Potential and challenges of body area networks for cardiac monitoring. *J. Electrocardiol.* 40, 6 (2007), S165–S168.
- [16] Fred H. Pollak and Manuel Cardona. 1968. Piezo-electroreflectance in Ge, GaAs, and Si. *Phys. Rev.* 172, 3 (1968), 816.
- [17] David N. Betts, H. Alicia Kim, Christopher R. Bowen, and D. J. Inman. 2012. Optimal configurations of bistable piezo-composites for energy harvesting. *Appl. Phys. Lett.* 100, 11 (2012), 114104.
- [18] E. J. Carlson, K. Strunz, and B. P. Otis. 2010. A 20 mV input boost converter with efficient digital control for thermoelectric energy harvesting. *IEEE J. Solid-State Circ.* 45, 4 (Apr. 2010), 741–750. DOI : <http://dx.doi.org/10.1109/JSSC.2010.2042251>
- [19] Y. K. Ramadass and A. P. Chandrakasan. 2011. A battery-less thermoelectric energy harvesting interface circuit with 35 mV startup voltage. *IEEE J. Solid-State Circ.* 46, 1 (Jan. 2011), 333–341. DOI : <http://dx.doi.org/10.1109/JSSC.2010.2074090>
- [20] Vinod R. Challa, M. G. Prasad, Yong Shi, and Frank T. Fisher. 2008. A vibration energy harvesting device with bidirectional resonance frequency tunability. *Smart Mater. Struct.* 17, 1 (2008), 015035.
- [21] Steve P. Beeby, R. N. Torah, M. J. Tudor, P. Glynne-Jones, T. O'donnell, C. R. Saha, and S. Roy. 2007. A micro electromagnetic generator for vibration energy harvesting. *J. Micromech. Microeng.* 17, 7 (2007), 1257.
- [22] Wing kei Yu, S. Rajwade, Sung-En Wang, B. Lian, G. E. Suh, and E. Kan. 2011. A non-volatile microcontroller with integrated floating-gate transistors. In *Proceedings of the IEEE/IFIP 41st International Conference on Dependable Systems and Networks Workshops (DSN-W'11)*. 75–80.
- [23] Yiqun Wang, Yongpan Liu, Shuangchen Li, Daming Zhang, Bo Zhao, Mei-Fang Chiang, Yanxin Yan, Baiko Sai, and Huazhong Yang. 2012. A 3us wake-up time nonvolatile processor based on ferroelectric flip-flops. In *Proceedings of the European Solid-State Circuits Conference (ESSCIRC'12)*. 149–152.
- [24] Noboru Sakimura, Tadahiko Sugibayashi, Ryusuke Nebashi, and Naoki Kasai. 2009. Nonvolatile magnetic flip-flop for standby-power-free SoCs. *IEEE J. Solid-State Circ.* 44, 8 (2009), 2244–2250.
- [25] Noboru Sakimura, Yukihide Tsuji, Ryusuke Nebashi, Hiroaki Honjo, Ayuka Morioka, Shunsuke Fukami, Sadahiko Miura, Naoki Kasai, Tetsuo Endoh, Hideo Ohno, Takahiro Hanyu, and Tadahiko Sugibayashi. 2014. 10.5A 90nm 20MHz fully nonvolatile microcontroller for standby-power-critical applications. *IEEE Int. Solid-State Circ. Conf.* 12, 4 (2014), 184–185.
- [26] Yongpan Liu, Zhibo Wang, Albert Lee, Fang Su, Chieh-Pu Lo, Zhe Yuan, Chien-Chen Lin, Qi Wei, Yu Wang, Ya-Chin King, et al. 2016. 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 *Proceedings of the 2016 IEEE International Solid-State Circuits Conference (ISSCC'16)*. IEEE, 84–86.
- [27] Zewei Li, Yongpan Liu, Daming Zhang, Chun Jason Xue, Zhangyuan Wang, Xin Shi, Wenyu Sun, Jiwu Shu, and Huazhong Yang. 2016. 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*. ACM, 154.
- [28] Hrishikesh Jayakumar, Arnab Raha, and Vijay Raghunathan. 2014. QuickRecall: A low overhead HW/SW approach for enabling computations across power cycles in transiently powered computers. In *Proceedings of the 2014 27th International Conference on VLSI Design and 2014 13th International Conference on Embedded Systems*. IEEE, 330–335.
- [29] Yongpan Liu, Zewei Li, Hehe Li, Yiqun Wang, Xueqing Li, Kaisheng Ma, Shuangchen Li, Meng-Fan Chang, Sampson John, Yuan Xie, Jiwu Shu, and Huazhong Yang. 2015. Ambient energy harvesting nonvolatile processors: From circuit to system. In *Proceedings of the 52nd Annual Design Automation Conference*. 150.

- [30] A. Mirhoseini, E. M. Songhori, and F. Koushanfar. 2013. Automated checkpointing for enabling intensive applications on energy harvesting devices. In *Proceedings of the 2013 IEEE International Symposium on Low Power Electronics and Design (ISLPED'13)*. 27–32.
- [31] Benjamin Ransford, Shane S. Clark, Mastooreh Salajegheh, and Kevin Fu. 2008. Getting things done on computational RFIDs with energy-aware checkpointing and voltage-aware scheduling. In *Proceedings of the Workshop on Power-Aware Computing and Systems (HotPower'08)*. 5–5.
- [32] Benjamin Ransford, Jacob Sorber, and Kevin Fu. 2012. Mementos: System support for long-running computation on RFID-scale devices. *ACM SIGPLAN Not.* 47, 4 (2012), 159–170.
- [33] Domenico Balsamo, Alex S. Weddell, Geoff V. Merrett, Bashir M. Al-Hashimi, Davide Brunelli, and Luca Benini. 2015. Hibernus: Sustaining computation during intermittent supply for energy-harvesting systems. *IEEE Embed. Syst. Lett.* 7, 1 (2015), 15–18.
- [34] C. Pan, M. Xie, and J. Hu. 2017. Maximize energy utilization for ultra-low energy harvesting powered embedded systems. In *Proceedings of the 2017 IEEE 23rd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'17)*. 1–6. DOI: <http://dx.doi.org/10.1109/RTCSA.2017.8046325>
- [35] Chen Pan, Mimi Xie, Yongpan Liu, Yanzhi Wang, Chun Jason Xue, Yuangang Wang, Yiran Chen, and Jingtong Hu. 2017. A lightweight progress maximization scheduler for non-volatile processor under unstable energy harvesting. *SIGPLAN Not.* 52, 5 (Jun. 2017), 101–110. DOI: <http://dx.doi.org/10.1145/3140582.3081038>

Received October 2018; revised February 2019; accepted April 2019