Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

MOPTIC-SM: Sleep mode-enabled multi-optimized intermittent computing for transiently powered systems

Kashif Javed [a], Naveed Anwar Bhatti [b,*], Mohammad Imran [b]

[a] Department of Avionics Engineering, Air University, Islamabad, Pakistan
[b] Department of Cyber Security, Air University, Islamabad, Pakistan

ARTICLE INFO

Keywords:
Transiently powered systems
Checkpointing
Approximate computing methods

ABSTRACT

The proliferation of Internet of Things (IoT) based devices has led to a significant increase in their usage across a broad range of application areas. However, the replacement and/or periodic maintenance of the billions of conventional power supplies used by these systems is required on an annual basis, which raises serious environmental and economic concerns. To subjugate these issues, transiently-powered embedded systems (TPESs) make use of an ambient energy resource. However, the non-uniform availability of ambient energy results in frequent system reboots. This problem can be mitigated by utilizing state checkpointing in non-volatile memory, yet a high number of checkpoints can lead to excessive energy consumption. In this research, a novel sleep mode-enabled multi-optimized intermittent computing method is proposed that combines data sampling and memoization to reduce the number of checkpoints. The proposed method is validated using the Microchip SAM-L11 embedded platform for the implementation of the Canny Edge Detection (CED) algorithm. The results of the experiments indicate that the proposed method effectively reduces the number of checkpoints for a given application by 50 percent in CED and maintains 70 percent accuracy in comparison to conventional TPES checkpointing. It is believed that the proposed solution will have several exciting applications in both the consumer market and industry.

1. Introduction

The systems of the future will involve trillions of Internet-of-Things (IoT) devices that will sense and compute pragmatic operations all around us [1–3]. The growth of IoT systems is augmented by factors such as the fourth industrial revolution [4], technical advancements in microelectronics [5,6], edge AI [7], and standardization of low power communication protocols [8], etc. However, as the number of IoT devices has increased in recent years, so has the volume of e-waste, with depleted batteries being the primary contributor. These depleted batteries contain a significant amount of hazardous, toxic, and corrosive materials such as mercury, cadmium, lithium, and lead which when soaked into the soil, during the coercion process, can have dire effects on the environment [9]. Furthermore, monitoring and replacing trillions of dead batteries every year not only increases the maintenance costs but also overshadows the benefits of IoT and other internet-connected devices. These issues prompted the scientific community to look for innovative ways to reduce reliance on batteries, particularly in the IoT domain.

Transiently-Powered Embedded Systems (TPES) have surfaced as a potential solution to this problem. TPESs are battery-less devices that rely on ambient energy harvested from the environment and stored in capacitors [10,11]. With these computing systems, the devices can be engineered to operate battery-less using capacitors, which utilize ambient energy to recharge [10].

However, ambient energy is not always consistent due to varying environmental conditions. For example, in the case of a solar-powered device, sunlight may not be available for a long period of time in cloudy conditions. This may cause the harvested energy to deplete to such a low level that the device stops functioning, leaving the computational tasks incomplete. Therefore, a critical challenge faced by battery-less energy harvesting systems is to ensure the completion of tasks in unpredictable environments. Since the energy-less periods are difficult to foresee therefore the task completion times are also hard to predict [12,13]. Due to frequent reboots caused by the loss of ambient energy, programs may corrupt or lose data, or fail to make forward progress. To address this issue, *checkpointing*, a method where applications save their current state to nonvolatile memory when the battery level drops to a predefined threshold, has been proposed. Upon system restart, the application restores the saved state and continues its computational tasks [14–17]. A major drawback of checkpointing

* Corresponding author.
E-mail addresses: 181701@students.au.edu.pk (K. Javed), naveed.bhatti@mail.au.edu.pk (N.A. Bhatti), m.imran@mail.au.edu.pk (M. Imran).

https://doi.org/10.1016/j.sysarc.2023.102850
Received 8 September 2022; Received in revised form 23 January 2023; Accepted 19 February 2023

is the computational overhead due to writing and reading the state of the system to and from memory. Excessive use of checkpointing can, therefore, drain the energy that would otherwise be used for computing and prevent the device from performing the desired computations. Hence, it is preferred to avoid the frequent use of checkpointing for TPESs [18,19].

Research has been carried out previously to identify methods for increasing the efficiency of systems, which in turn results in the reduction of checkpoints. The previously proposed approaches either do not target the energy harvesting systems [20], or demand hardware-based modifications in the systems [21,22]. We aim to propose a purely software-based solution that can be readily adopted on currently available hardware platforms. In our previous work [23], we applied memoization on a Kalman Filter application to show that optimization methods can be effectively used to reduce the number of checkpoints in a TPES. In this work, we propose a method that combines data sampling, an approximation technique, with memoization to further reduce the number of checkpoints. We evaluate the proposed Multi-OPtimized Intermittent Computing technique, MOPTIC, on a Canny Edge Detector application running on SAM-L11 platform. Our experiments show that the combination of data sampling and memoization significantly enhances the application performance by considerably reducing the number of checkpoints required for completing the computational task. The proposed method also provides the user with an extra degree of freedom to select the desired accuracy level by trading-off energy consumption.

We also propose a novel combination of the MOPTIC approach and the state-of-the-art sleep mode computing [24]. Experimental and computational analyses of the worst-case scenario that we model in our experiments demonstrate that the application of the proposed method results in a significant reduction in checkpoints, indicative of the guaranteed energy savings in the probable and best case. To the best of our knowledge, the proposed MOPTIC-SM is a novel software-based framework that we believe can have a profound impact on the energy consumption, performance, and sustainability of the battery-less IoT-based Transiently powered embedded systems.

The contributions of this work can be summarized as stated below:

- **[C1] We propose MOPTIC** – Multi-Optimized Intermittent Computing – using a novel combination of data sampling and memoization for reducing the number of checkpoints
- **[C2] We enhance MOPTIC** - with Sleep Mode enabled mechanism - MOPTIC-SM - which promises to be an effective method for achieving a greater reduction in the number of checkpoints
- **[C3] We implement MOPTIC** — on a real application and perform comprehensive experiments to demonstrate its effectiveness

This article proceeds as follows. In the subsequent section, the previous research carried out in the domain of intermittent computing is discussed. Section 3 elaborates multi-optimized intermittent approach for TPESs. The Sleep mode-enabled multi-optimized intermittent computing (MOPTIC-SM) approach on TPESs is presented in Section 4. In Section 5, experimental manifestation and results of the benchmark application are provided which is followed by the conclusion and future directions in Section 6.

2. Related work

Transiently powered embedded systems operate on ambient energy, and due to the inherent uncertainty in the availability of ambient energy, software and hardware-based improvements are often required to make these systems sustainable. We discuss some previous work on checkpointing techniques, which, while orthogonal to what we propose, is the key software-based technique that allows these systems to save their state during power outages. We also look at approximate computing techniques that improve the performance of transiently powered systems by sacrificing accuracy for throughput. Furthermore, we examine a sleep mode proposed in the literature to further reduce the number of checkpoints.

There are several checkpointing techniques proposed in the literature. Bhatti and Mottola [11,17] proposed techniques such as Split, Heap trekker, and Copy If-Change to checkpoint and restore a device's state on stable storage. Because of the distinctive performance versus energy trade-off for each application scenario, there is no "one-size-fits-all" solution. The performance of each presented technique depends on factors such as the amount of data to handle, the layout of data in memory, as well as an application's read/write patterns. However, they conclude that such "smarter" approaches for checkpointing have a favorable effect only when the application operates with a relatively small quantity of data between checkpoints.

Approximate computing techniques are commonly utilized for embedded applications, due to the low computational capacity and limited battery power available to such systems. The approximation method for a particular application needs to be very carefully balanced, keeping in view the performance versus energy trade-off. Mittal [25] presented a detailed analysis of various approximation techniques, their need, and respective application areas. Some of the presented techniques are precision scaling, loop perforation, load value approximation, memoization, skipping tasks and memory accesses, etc. All the above approximation techniques have been used in the past for the devices which are powered by the grid or battery. Mittal highlighted the opportunities and problems while using approximate computing for optimization, followed by a detailed literature survey on such techniques. Mittal also stated that researchers will have to turn their attention to general-purpose applications, thus extending the scope of approximate computing to the entire spectrum of computing applications. Ganesan et al. [22] proposed a hardware based approach for speeding up computations using approximation methods at a lower granularity. The authors also evaluated their technique on check-point based volatile processors as well as on non-volatile processors, and reported a significant speed-up of 300%. In a recent work, Mishra et al. [21] employed data sampling and memoization to reduce communication overheads in an IoT application, but their approach needs support from additional hardware for faster coreset formation.

The bulk of previous work done in approximate computing for embedded systems targets platforms that are either continuously powered by a grid/battery or in case of TPES employed hardware changes. In our previous work [23], we implemented memoization on transiently powered devices to analyze its effect on energy consumption. We highlighted that memoization is a machine-independent, cross-platform strategy for approximate computing that works during the run-time of the application instead of compile time. The use of memoization resulted in increased speed of the computation in exchange for storage space, as it requires memory to store precalculated values.

When someone is finding it difficult to make a decision, we advise them to *sleep over it*, and the idea seems to be applicable to TPES too! Lukosevicius et al. [24] presented a method for reducing reliance on checkpointing that employs a *'sleep'* state that is entered when the system's power supply is detected to be failing. It employs two distinct thresholds, namely the sleep threshold and the checkpoint threshold. The sleep threshold is set higher than the checkpointing threshold. If the TPES's power is restored and rises above the sleep threshold, the system wakes up and resumes computation. This enables the system to recover without needing to take a state checkpoint. As a result, the application can focus on useful computations rather than checkpointing. When the system is in sleep mode, a snapshot is taken if the supply voltage continues to fall and reaches the checkpoint threshold. This method is effective in situations where the harvester output fluctuates rapidly, for example when powering from vibrations or RF signals, etc.

The previous work discussed above separately focused on the use of checkpointing, approximate computing and sleep mode in different domains, while some of the previously proposed techniques addressed

the problem from a hardware point of view. In our research, We have taken a purely software-based approach, employing a combination of these techniques in a specific application area for achieving the best possible usage of ambient power.

3. Multi-optimized intermittent computing approach (MOPTIC)

Several optimization methods have been proposed and implemented for computing devices to reduce the computation expense [16]. Since IoT-based devices used for TPESs are low cost commercial-off-the-shelf (COTS) devices characterized by limited computational power and memory, therefore optimization techniques having high computational needs are not suitable for such TPESs [18,26]. On the other hand, approximation based optimization techniques such as precision scaling, loop perforation, down-sampling and skipping tasks show great potential for TPES related applications due to their low computational requirements. There are a few challenges associated with approximate computing methods such as ❶ limited application domain and inadequate gains from approximate computing, ❷ deteriorated accuracy of results, ❸ need for specific strategies per application and ❹ computational overhead and scalability issues [25]. Therefore, no specific approximation technique can be universally applied to all TPES applications. Instead, the approximation techniques need to be selected according to the computational requirements of the application under investigation either by the developer or by an intelligent program [25]. In summary, once the potential approximation processes and variables have been identified in an application, these variables can be approximated using any of the techniques stated above [25–27]. Conventionally, only one approximation technique is employed that best fits the application requirements. However, in this research, the applicability and advantage of collectively using multiple optimization techniques for a single target application is demonstrated.

The canny edge detection (CED) algorithm is selected for the application of the proposed approximation techniques. The major reason to choose this algorithm is its widespread utility in IoT domain applications, such as the detection of changes in surveillance scenarios. IoT devices that are used for detecting changes in the scene extract the features from images being recorded by surveillance cameras. Applying approximations on CED would enable the TPES to operate indefinitely under the limited resource environment. The approximation techniques that are selected for the application of canny edge detection are data sampling and memoization [18,27,28]. A brief description of these approximation techniques is provided below:

3.1. Data sampling

Input data sampling is an approximation technique in which only a subset of the input data is used for computation. An application may select a certain percentage of input data items for speeding up the computation, or down-sample the data by reducing the resolution or granularity of individual data items. For the canny edge detection application, the data sampling technique will save computation by applying the algorithm on low-resolution samples while achieving user specified accuracy requirements [29–31]. In this work, the input image of size 120 × 120 pixels is down-sampled to 6 × 6 pixels and then later used for comparison between two samples to detect a change. In this way, this technique not only helps reduce the computation of edges in 120 × 120 samples but also enables the application to run on low memory systems because of less space requirements.

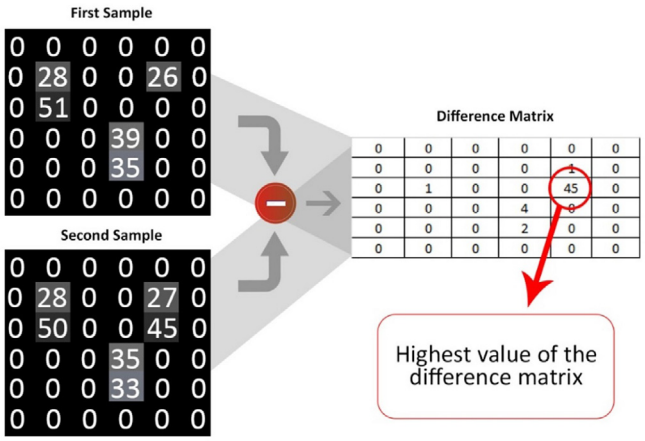

Fig. 1. Process for computing peak difference between images.

3.2. Memoization

Memoization is an optimization technique that helps in achieving a faster computation by reloading the previously acquired results against the same input variables instead of computing them anew. In other words, it reuses the previously calculated values instantly for a set of repeatedly used inputs [16,25,31–33]. In the selected application, the edges of the current down-sampled image are compared with the down-sampled version of the previous image to detect a change in the input data. As a result, the previously computed edges of a full-resolution image are loaded instantly in case of unchanged input conditions in the current image, resulting in considerable savings in energy that would otherwise be used in computing the edges of the current full-resolution image. This positively affects the energy efficiency of the system, and reduces the energy consumption by the application for completing its computational cycle.

Here we consider it necessary to introduce *Delta Threshold* which acts as a tuning parameter for MOPTIC. Delta Threshold is a user settable parameter that is used by MOPTIC to determine if an appreciable change has occurred between previous and current images. To compare two images, the difference between corresponding pixel values is calculated pixel by pixel, resulting in a difference matrix as shown in Fig. 1. The highest value in this matrix, termed as peak difference, is compared against Delta Threshold. If the peak difference is more than the Delta Threshold then the algorithm considers the two images to be considerably different. This process is described in Algorithm 1.

The value of the Delta Threshold plays an important part in the effective utilization of the proposed method. Let us consider the case when the value of Delta Threshold is kept 0; the algorithm will consider the images being compared as different even if the values of a corresponding pixel in the images differ by 1. This corresponds to the highest

Algorithm 1: GetPeakDiff

```
Input: Image_1, Image_2
Output: PeakDiff
1
2  PeakDiff = 0
3
4  for h = 1 to ImageHeight do
5      for d = 1 to ImageWidth do
6          Diff = |(Image_1[h][d] − Image_2[h][d])|
7          if (Diff) > PeakDiff then
8              PeakDiff = Diff
9  return PeakDiff
```

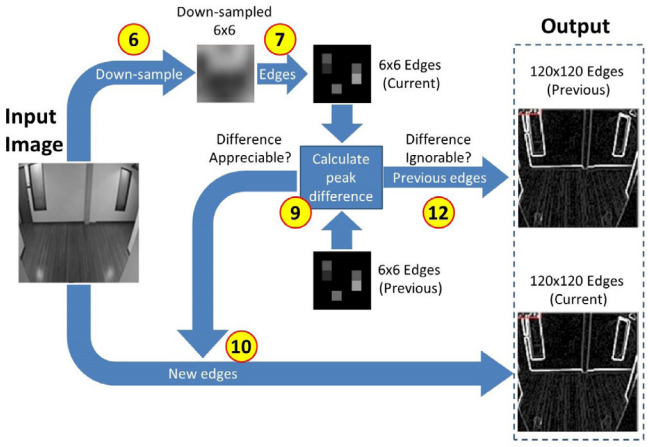

Fig. 2. Flow diagram for applying MOPTIC on canny edge detector.

accuracy, since the system is able to detect the smallest change in the surveillance images. Consequently, the system will opt for recomputing the edges of the current image instead of using memoization to load the previously computed edges, resulting in more computation and hence excessive energy consumption. Conversely, if the value of Delta Threshold is set to the maximum value of 255, then the algorithm will consider the two images as the same even if the white pixel in one image is changed to black in the other image. It is evident that this situation represents the worst accuracy because the system is not able to detect even the biggest change in the surveillance footage. As a result, the system will load the previously computed edges of the previous image and save computation and energy. Due to this inherent trade-off between accuracy and energy efficiency, the effectiveness of the proposed algorithm depends on the correct setting for this parameter as per the application requirements.

The proposed multi-optimized algorithm is described in Algorithm 2, while Fig. 2 provides its pictorial representation. The numbers within the circles in the figure represent the corresponding statements in the algorithm. We have not provided the description for Canny Edge and Image Down-sampling since these are standard algorithms and the interested reader will find many sources for further information on these.

Algorithm 2: MOPTIC

Input: $Image_{Curr}$ (120x120)
Output: $Edges_{Curr}$ (120x120)

$DT = DeltaThreshold$
$ELFI = Edgesof LastFullImage(120x120)$
$ELDI = Edgesof LastDownSampledImage(6x6)$

$Image_{Curr\_DS} = DownSample(Image_{Curr})$
$Edge_{Curr\_DS} = CannyEdge(Image_{Curr\_DS})$

if *GetPeakDiff* $(Edge_{Curr\_DS}, ELDI) > DT$ **then**
  $ELFI = CannyEdge(Image_{Curr})$
  $ELDI = Edge_{Curr\_DS}$
return $ELFI$

4. Multi-optimized intermittent computing with sleep mode (MOPTIC-SM)

In this section, different hypothetical scenarios are used to analyze the effect of various optimization approaches on a typical TPES application life cycle. A TPES device configured for conventional checkpointing has three different voltage thresholds – startup, checkpoint, and cut-off – during its operating cycle. When the capacitor voltage reaches the startup threshold, the TPES boots up and keeps operating until the voltage drops to the checkpointing threshold as shown in Fig. 3. The green portion illustrates the charging period of TPES while the brown color represents the operational period. When TPES voltage falls to the checkpointing threshold, TPES checkpoints its current state. Ideally, the checkpointing threshold should be set low enough that by the time the checkpointing operation gets completed, the voltage should have dropped to the cut-off voltage. Any operations performed after the checkpointing will be lost due to a power outage. When the next cycle of ambient energy arrives, the voltage level of the TPES rises up to the start-up threshold and the device restores its saved state which is represented by the white region in Fig. 3. Upon restoration, the device repeats the cycle. The device may undergo multiple checkpoints during a complete application life cycle.

Lukosevicius et al. [24] proposed the introduction of a fourth voltage threshold – the sleep threshold – above the checkpointing threshold. If the supply voltage drops to the sleep threshold, the device will enter hibernation mode as shown in Fig. 3. The main idea behind introducing this threshold is to keep the device on for a longer period of time hence preventing it from undergoing frequent checkpointing. In an ideal case, if the subsequent energy cycle arrives during the proposed intermediate sleep mode duration, the device will retain its current state and will start the operation from where it had hibernated (Sleep Mode in Fig. 3), unlike the aforementioned conventional checkpointing. One obvious advantage of using sleep mode is the eradication of excessive state checkpointing. The energy preserved as a result of this eradication will be available for performing the desired computing task. Another benefit is that the device can resume computation as soon as the supply voltage becomes available, in contrast with conventional checkpointing where the device undergoes long idle cycles while waiting for start-up threshold voltage attainment and during state restoration. In the worst case, the subsequent energy cycle will not arrive during the proposed sleep mode and the device voltage will eventually drop to the checkpointing threshold voltage, consequently forcing the device to turn-off as shown in Fig. 3 (Sleep Mode). The disadvantage in worst case scenario is that the device is limited to perform only 20 percent of the desired task as compared with the conventional checkpointing without sleep mode (30 percent) and the device will consume more energy cycles to complete the task.

Keeping in view the advantages of sleep mode, we propose that the MOPTIC approach discussed above may be supplemented by a sleep mode to further reduce checkpointing. The expected effect of multi-optimized intermittent computing approach on conventional checkpointing is that the device completes the task with 30 percent reduction in checkpointing as shown in Fig. 3 (Sleep Mode). We predict that the proposed MOPTIC approach combined with sleep mode can bring improvements in mitigation of overall checkpoints and performance of TPES during the worst case scenario as shown in Fig. 3 (MOPTIC-SM), respectively. Even in the worst case, the performance of MOPTIC-SM will be comparable to the conventional checkpointing case and better than the worst case of SM approach without MOPTIC. Therefore, it is expected that MOPTIC-SM will have significant improvement over SM and conventional checkpointing in terms of the overall computing efficiency and checkpointing reduction.

5. Methodology

5.1. Experimental setup

A commercially available Microchip SAM-L11 Xplained pro embedded platform [34] is used to experimentally implement the Canny Edge Detection algorithm using the proposed approach as shown in Fig. 4. SAM-L11 provides an inbuilt solution to measure the current and power

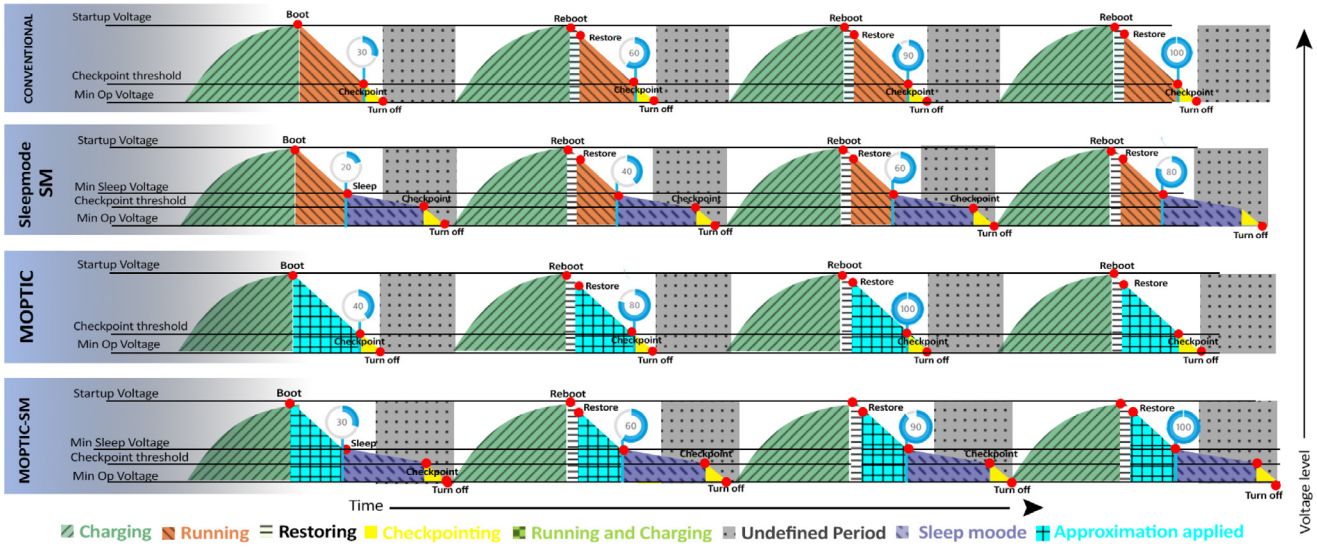

Fig. 3. The life-cycle of application running on transiently powered device with (a) conventional checkpointing, (b) sleep mode enabled conventional checkpointing — worst case, (c) multi-optimized intermittent computing approach, (d) sleep mode enabled multi-optimized intermittent computing approach — worst case. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

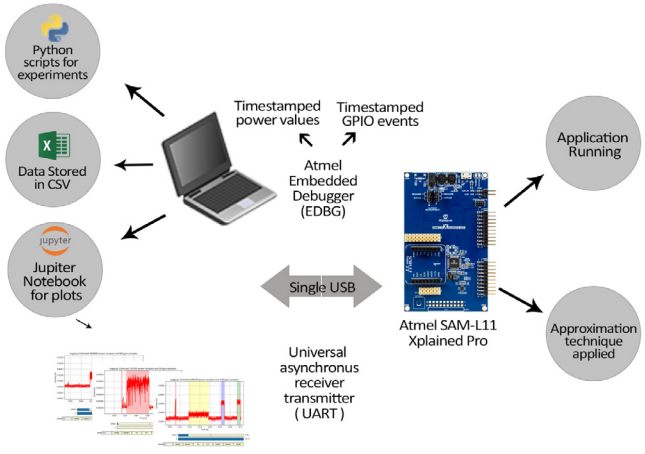

Fig. 4. Proposed experimental setup using SAM-L11 as evaluation platform.

drawn during the application execution, making it an ideal platform for measuring energy requirements for a given application. It may be noted that this particular platform is only used for demonstration of the effectiveness of the proposed approximation technique; the proposed technique can be implemented for any similar application to be executed on any hardware platform.

We selected 10 images from the target surveillance which capture a complete cycle of a scenario change. This kind of surveillance is most favorable for TPES as the changing scenario happens less often and the scene remains unchanged during no activity hours. The CED application is allowed to run on SAM-L11 which is interfaced with a computer using Atmel Studio. SAM-L11 deploys the canny edge detection algorithm which takes the input samples from target surveillance and generates the edge-detected images as output. One downside of using SAM-L11 is its limited memory, which prevented us from executing the application on all images in one run. To subjugate this issue, we performed the experiments on micro-scale by processing the input surveillance images in subsets, and then extrapolated the results to get the bigger picture. However, external memory can be used with SAM-L11 to resolve such issues. A Python measurement library is employed to extract the power readings from SAM-L11 and log to PC for further processing [35]. In the final step, Jupyter Lab notebooks were used to draw and analyze the output and power readings from SAM-L11.

To perform our experiments, we obtained ten images from a surveillance camera's prerecorded scenario for TPES [36]. The full images were composed of 120 × 120 pixels and the down-sampling factor was set as 20, so the size of the down-sampled image size was 6 × 6 pixels.

5.2. Calculation of checkpoints

In this section, we describe the methodology adopted for counting the number of checkpoints required by the application for completing one cycle. The process is explained separately for CED application running on TPES using conventional checkpointing and with MOPTIC.

5.2.1. Conventional CED application

As the first step, we measure the actual energy consumed by the platform for executing canny edge algorithm on 6 × 6 pixel input image, E_{CED_6x6}. This measurement is made using the total current drawn along with a timestamp, acquired from SAM-L11 kit as explained above. To calculate the energy consumption for the processing of one image of 120 × 120 pixels, $E_{CED\_120x120}$, we use the energy consumption for 6 × 6 image as reference. Hence the total energy required for the full application cycle, using conventional checkpointing, is calculated by multiplying the total number of images with energy consumption for a single 120 × 120 image. We call this energy E_{conv} as shown in Eq. (1).

$$E_{conv} = E_{CED\_120x120} \times 10 \tag{1}$$

In a TPES application, the required energy will be provided by the capacitor. The energy provided by a fully charged capacitor is calculated by the capacitor energy formula as $E_{cap} = \frac{1}{2}CV^2$, where C is the capacitance value and V stands for supply voltage. However, it may be noted that not all of this energy is usable for the computational tasks, as such a device requires a certain minimum voltage level for operation as shown in Fig. 5. Upon reaching this cut-off voltage threshold V_{cut_off}, the device simply shuts down. In conclusion, the total capacitor energy available for processing reduces to

$$E_{cap} = \frac{1}{2}CV^2 - \frac{1}{2}CV_{cut\_off}^2 \tag{2}$$

An application using checkpointing must set another voltage threshold, V_{ChkPt}, above the cut-off level as seen in Fig. 5. If the voltage level drops to this threshold, the application stops performing the desired

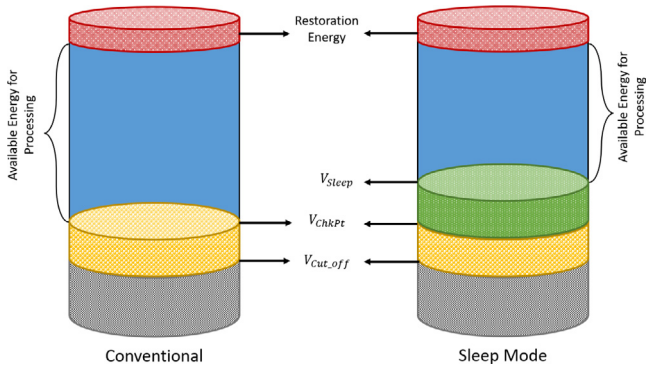

Fig. 5. Voltage thresholds and corresponding energy constraints in a TPES.

Table 1
Energy measured for computational tasks.

Parameter	Description	Value
$E_{CED\_60 \times 60}$	Applying CED on 6 × 6 image	1.86E−04 J
$E_{CED\_120 \times 120}$	Applying CED on 120 × 120 image	7.45E−02 J
E_{down}	Down-sampling	3.16E−04 J
E_{comp}	Comparing two 6 × 6 images	6.32E−06 J
$E_{checkpoint}$	Checkpointing	1.56E−04 J
$E_{restore}$	Restoration of state	1.85243E−05 J

computation and writes the system state to memory, consuming an amount of energy that we term as $E_{checkpoint}$. In addition, when the device reboots after a shutdown, it must restore the state which is also an energy consuming task. Let us call the energy spent on restoration $E_{restore}$. We treat checkpointing and restoring as overhead tasks, since these are not part of the CED application as such. It is, therefore, essential that energy used for these tasks be excluded from the total energy available for computations by the CED application. In essence, the total energy available for the application can be calculated as:

$$E_{avail} = E_{cap} - (E_{checkpoint} + E_{restore}) \tag{3}$$

Once we have the total energy required for a complete application cycle (Eq. (1)) and available capacitor energy (Eq. (3)), we can calculate the total number of checkpoints that occur during the full application cycle using conventional checkpointing as given by Eq. (4).

$$ChkPts_{conv} = \frac{E_{conv}}{E_{avail}} \tag{4}$$

An important assumption that we make here is that the ambient energy becomes unavailable once the capacitor is fully charged, and only becomes available when the voltage drops below the checkpointing threshold. In other words, we are modeling the worst case scenario.

Here it is pertinent to mention that all these calculations are based on actual values of E_{CED_6x6}, $E_{checkpoint}$ and $E_{restore}$ measured on the SAM-L11 platform. These values are provided in Table 1.

5.2.2. Conventional CED with sleep mode

If an application is configured to enter sleep mode in order to prevent frequent checkpointing, then another voltage threshold V_{sleep} must be set above the checkpointing threshold voltage as shown in Fig. 5. Therefore, in a sleep mode enabled application the total energy available for computation is less. In such a case, Eq. (2) can be modified as

$$E_{cap\_SM} = \frac{1}{2}CV^2 - \frac{1}{2}CV_{sleep}^2 \tag{5}$$

Calculation of total energy available for the application can be made by substituting E_{cap_SM} for E_{cap} in Eq. (3) and, subsequently, number of checkpoints can be computed using E_{avail} in Eq. (4).

It may be noted that we are modeling the worst case scenario for the sleep mode — when the application enters sleep mode it eventually checkpoints because of the unavailability of ambient energy. As pointed out earlier, the benefits of sleep mode become evident in TPES applications where there is irregular arrival of energy bursts for charging the capacitor. We believe that if the proposed algorithm is demonstrated to have completed the application with fewer number of checkpoints in the worst case scenario, then it can be inferred that the algorithm will show a similar advantage in situations where the source of ambient energy becomes available before reaching the checkpointing threshold.

Table 2
Parameters for experimental evaluation.

Parameter	Description	Value
V	Supply voltage	5 V
V_{cut_off}	Cut-off threshold voltage	1.8 V
V_{ChkPt}	Checkpointing threshold voltage	1.89 V
V_{sleep}	Sleep threshold voltage	2.2 V
C	Capacitor size	200/500/1000 µF

5.2.3. CED with MOPTIC

The calculation of the number of checkpoints for MOPTIC is slightly different. In each complete application cycle, tasks including down-sampling of the current 120 × 120 image into 6 × 6 image, calculation of 6 × 6 edges, and comparison of current 6 × 6 edges with the previous 6 × 6 edges are performed for each image. The energy consumption for each of these tasks is measured separately on SAM-L11, represented by E_{down}, E_{CED_6x6}, and E_{comp} respectively in Table 1. In addition to these tasks, the calculation of edges for the full current image may also need to be performed for images that are considerably different from the previous image. In such cases, the energy $E_{CED\_120x120}$ will be additionally consumed. So the total energy requirement for CED using MOPTIC can be formulated as:

$$\begin{aligned} E_{MOPTIC} &= (E_{down} + E_{CED\_6x6} + E_{comp}) \times n \\ &+ \quad E_{CED\_120x120} \times x, \end{aligned} \tag{6}$$

where n is the total number of images (10 in our experiments) and x represents the number of cases where the edges of the full 120 × 120 image were calculated. The value of x is determined on the basis of the Delta Threshold and peak difference between the previous and current images. The number of checkpoints that occur during the full application cycle using MOPTIC is calculated as:

$$ChkPts_{MOPTIC} = \frac{E_{MOPTIC}}{E_{avail}} \tag{7}$$

5.2.4. CED with MOPTIC and sleep mode

In order to count the number of checkpoints required by CED with MOPTIC in sleep mode, we need to use E_{cap_SM} (Eq. (5)) for calculating E_{avail} (Eq. (3)) and then apply Eq. (7) accordingly.

In our experiments we used there different capacitance values, i.e, 200 µF, 500 µF and 1000 µF, for calculating available energy. The operating voltage for the device was specified as 5 V and the cut-off voltage threshold V_{cut_off} was 1.8 V as per the device specifications. V_{ChkPt} was set to 1.89 V by incorporating the exact energy requirements for performing the checkpointing operation as measured on SAM-L11. Sleep voltage level V_{sleep} was set at 2.2 V following the work presented by Lukosevicius et al. [24]. These figures are provided in Table 2.

5.3. Evaluation

For evaluating the effectiveness of MOPTIC against conventional checkpointing, we used the energy measurements given in Table 1 and calculated the number of checkpoints required by the conventional checkpointing application for completing its task as explained in Section 5.2.1. We then applied the methodology described in Section 5.2.3

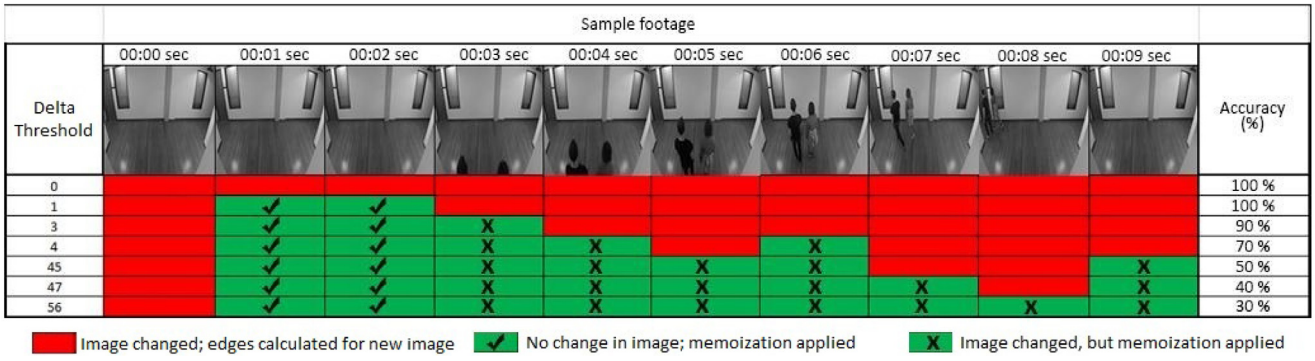

Fig. 6. Accuracy calculation for different values of delta Threshold.

to calculate the number of checkpoints required by MOPTIC against different values of Delta Threshold to complete the same task.

We also evaluated the sleep mode enabled MOPTIC-SM against the state-of-the-art sleep mode with conventional checkpointing (SM) as proposed by Lukosevicius et al. [24]. For performing this comparison, we modeled the worst-case scenario in terms of the availability of ambient energy; the calculations were performed with the assumption that in both SM and MOPTIC-SM the ambient energy cycle did not arrive before the system voltage dropped to cut-off level, and therefore both the applications had to enter sleep mode and then perform checkpointing. As described in related work section, a previous research [22] has also proposed the use of approximation techniques for energy harvesting systems and evaluated the proposed method on an image processing application, but we have not considered comparing our work with the aforementioned scheme because there exists a fundamental difference between the two methods; we have relied on software based approximation and the previous work suggests improvements at the hardware level.

For the sake of this research, the values of Delta Threshold were manually selected based on peak difference between consecutive images. Though naive, this approach does emphasize the point that IoT applications in general have various parameters that need to be fine-tuned keeping in view the trade-offs in a particular application scenario. In conclusion, the values of Delta Threshold for the same CED application in another environment may be entirely different than those used in this research.

The accuracy of results for MOPTIC was also computed against each value of Delta Threshold. For each new image, MOPTIC either uses memoization to load previously computed full edge or computes full edges of new image, based on the how it perceives the difference between previous and current images as per the value of Delta Threshold. The algorithm is considered to have taken correct action in two cases: (i) the new image was considerably different from the previous and the algorithm computed full edges of the new image, and (ii) there was no appreciable difference between the previous and new images and the algorithm used memoization to load full edges of the previous image. There are also incorrect actions that the algorithm might perform — the new image was considerably different but the algorithm reloaded full edge of the previous image, or the new image was not considerably different but its full edges were computed. Hence the accuracy of the algorithm for any value of Delta Threshold was calculated as the percentage of correct actions taken over the full application cycle. This process is depicted in Fig. 6.

5.4. Results and analysis

In this section, we present and analyze the results obtained from the experiments. First, we discuss the accuracy of the proposed algorithm as a function of the Delta Threshold. As we described earlier, Delta Threshold is a 'tuning' parameter that can be used to adjust the algorithm's energy efficiency in terms of reduction of checkpoints by trading off accuracy, and vice-versa. We ran the application with various values of Delta Threshold and computed the accuracy of results as per the methodology described in Section 5.3. Our experiments showed that increasing the values of Delta Threshold resulted in deterioration of accuracy, as shown in Fig. 6. We shall use these results subsequently in the discussion on the energy saving vs. accuracy trade-off.

We now compare and analyze the effectiveness of the proposed technique, MOPTIC, against the conventional checkpointing in terms of the number of checkpoints required by the two methods for completing one application cycle. This comparison, for a 200 μF capacitor, is shown in Fig. 7. The graph plots the number of checkpoints required by MOPTIC against different values of Delta Threshold which helps us to analyze the accuracy vs. energy-saving trade-off for MOPTIC.

Fig. 7 reveals that the number of checkpoints for MOPTIC is slightly more than the conventional checkpointing for Delta Threshold of 0, but drops significantly for all values of Delta Threshold above 0. As discussed earlier, the Delta Threshold of 0 forces MOPTIC to treat each image as different from the previous, hence causing a recalculation of edges for each new image regardless of the result of comparison. Therefore, for Delta Threshold = 0, MOPTIC acts just as conventional checkpointing. For this case, despite behaving like conventional checkpointing, MOPTIC is *additionally* performing the tasks such as down-sampling, 6 × 6 edge computation, and comparison operations for each image hence causing excessive energy consumption, resulting in more checkpoints.

The reduction in the number of checkpoints for MOPTIC comes at a cost. As we increase Delta Threshold, the algorithm's capability of detecting change in consecutive images is reduced, resulting in a decline in accuracy as explained earlier. Fig. 8 depicts the trade-off between improvement in energy saving due to reduced checkpoints vs. accuracy for 200 μF capacitor. It is observed that for this particular scenario, Delta Threshold value of 4 resulted in an optimal situation with almost 50% reduction in the number of checkpoints against conventional checkpointing with acceptable accuracy figure of 70%. Here we wish to emphasize the fact that the effectiveness of the proposed algorithm over conventional checkpointing also depends on the surveillance scenario. For input images where the change is not observed frequently, such as the footage taken from inside a bank vault, the propose method will show significant improvement over the conventional method. On the other hand, the benefits of proposed method will be on a lower scale for the images captured by surveillance camera at the bank's entrance because of rapidly changing scenes requiring frequent calculation of edges for new images.

When sleep mode is introduced, as shown in Fig. 9 for 200 μF, we observe the similar pattern as before: the number of checkpoints for MOPTIC-SM reduces with increasing values of Delta Threshold. Similarly, Fig. 10 shows that the accuracy line meets the improvement line roughly at a point that corresponds to Delta Threshold's value of

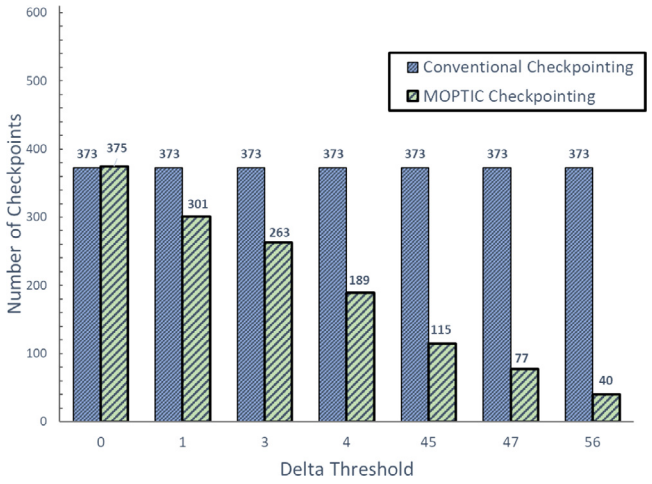

Fig. 7. Comparison of No. of checkpoints in conventional checkpointing and MOPTIC.

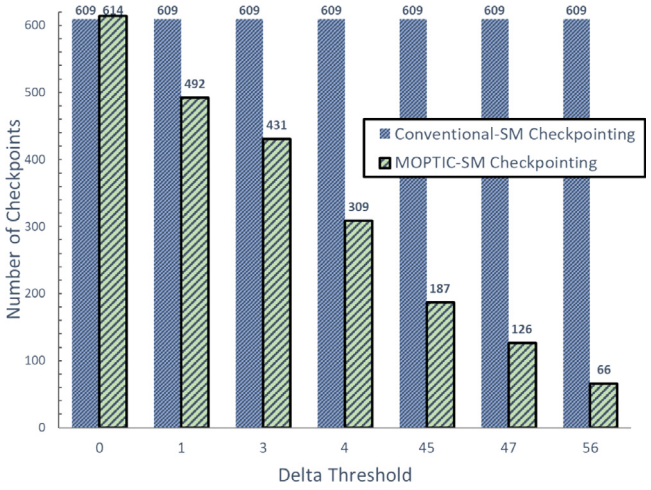

Fig. 9. Comparison of No. of checkpoints in conventional sleep mode and MOPTIC-SM.

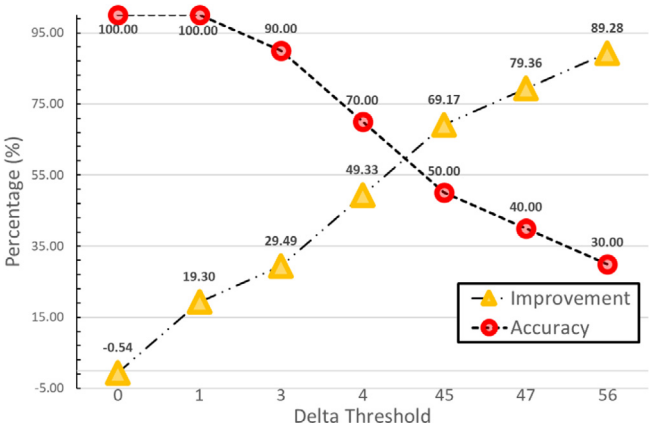

Fig. 8. MOPTIC: Trade-off between accuracy and reduction in checkpoints.

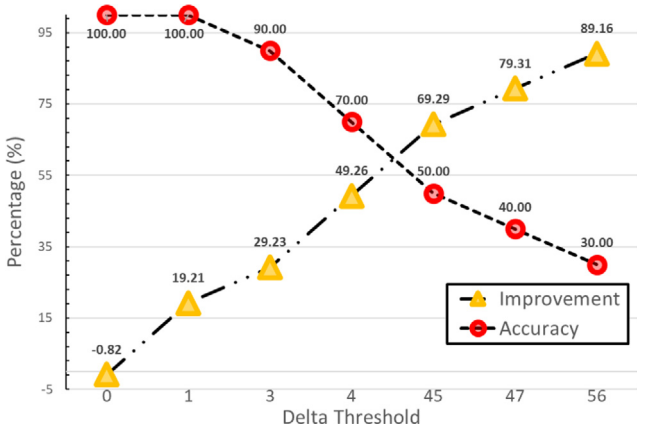

Fig. 10. MOPTIC-SM: Trade-off between accuracy and reduction in checkpoints.

4, marking it as the optimal setting for accuracy vs. energy saving trade-off just as without sleep mode.

An important observation to be made here is that the number of checkpoints increased for both the conventional sleep mode application and MOPTIC-SM as compared to without sleep mode. The reason behind this increase lies in the fact that the sleep voltage threshold is kept above the checkpointing threshold voltage, as shown in Fig. 5. Therefore, the total energy available for computation is less than in case of non-sleep mode application, requiring the application to undergo more checkpoints to complete one computational cycle.

As noted earlier, we modeled the worst case scenario for the sleep mode, representing situations where the application eventually checkpoints after going to sleep mode because of unavailability of ambient energy cycle. We believe that since the proposed algorithm has been demonstrated to complete the application with fewer number of checkpoints in the worst case scenario as compared with conventional sleep mode, therefore it is expected to perform even better in situations where the source of ambient energy becomes available before the device reaches the checkpointing threshold.

We carried out similar experiments with 500 μF and 1000 μF capacitors, as shown in the graphs in Fig. 11. For both these cases, the number of checkpoints is reduced as compared with 200 μF capacitor due to more available energy per charging cycle. The overall improvement between MOPTIC and traditional checkpointing remains unchanged.

In summary, the experimental results presented herein are fully in agreement with the hypothetical values predicted in Section 4, showing clear advantages of the proposed multi-optimized intermittent computing approach over conventional checkpointing in TPES with and without sleep mode.

6. Conclusion and future directions

This work presented a novel sleep mode-enabled multi-optimized computing approach and demonstrated its effectiveness on the benchmark application of canny-edge detection. Comprehensive experiments were conducted by applying conventional checkpointing and MOPTIC on a sample footage using SAM-L11, an embedded evaluation platform. Evaluation parameters including accuracy and number of checkpoints were numerically computed and experimentally verified. The experiments show that the proposed approach offers a unique trade-off between number of checkpoints and accuracy of the TPES. It is concluded that the proposed MOPTIC-SM approach outperforms the conventional checkpointing method with and without sleep mode with 50 percent reduction in checkpoints while maintaining 70 percent accuracy. To the best of our knowledge, the proposed MOPTIC-SM approach is the best possible software based solution for reducing the number of checkpoints with an extra degree of freedom in terms of accuracy selection.

The image processing application for TPESs was implemented for the first time. We had to evaluate the results one by one because

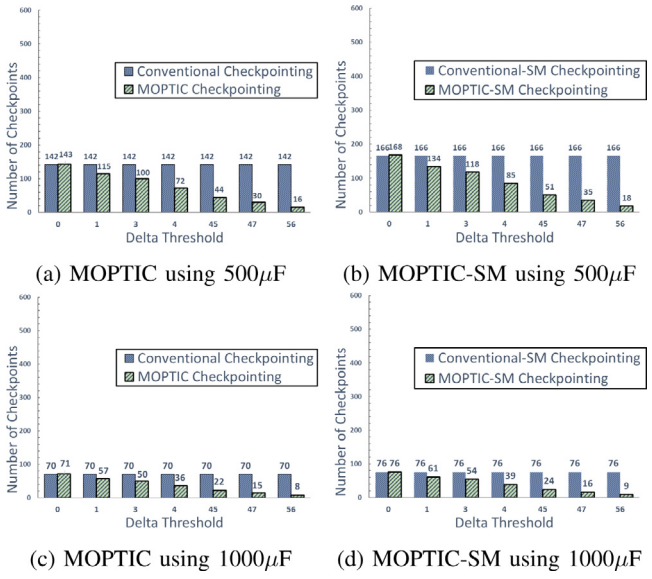

Fig. 11. Comparison of No. of checkpoints in conventional, MOPTIC, conventional sleep mode and MOPTIC-SM using 500 μF and 1000 μF capacitors.

we could not process all of the images at once due to memory constraints. Furthermore, we did not use the actual energy harvesting mechanism with SAM-L11 because we desired to introduce control and repeatability into our experiments.

A future direction for extending this work can be the implementation of proposed MOPTIC-SM for a larger input dataset with increased number of appreciable changes. This will assist in development and enhancement of the hashtag table for memoization under a repetitive input. Consequently, this will lead to considerably less computations in case of any unprecedented input change.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

References

[1] P. Fraga-Lamas, T.M. Fernández-Caramés, M. Suárez-Albela, L. Castedo, M. González-López, A review on internet of things for defense and public safety, Sensors 16 (10) (2016) 1644.

[2] J. Hester, J. Sorber, The future of sensing is batteryless, intermittent, and awesome, in: Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems, 2017, pp. 1–6.

[3] B. Lucia, V. Balaji, A. Colin, K. Maeng, E. Ruppel, Intermittent computing: Challenges and opportunities, in: 2nd Summit on Advances in Programming Languages (SNAPL 2017), Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[4] M. Skilton, F. Hovsepian, The 4th Industrial Revolution, Springer, 2018.

[5] R.D. Sochol, E. Sweet, C.C. Glick, S.-Y. Wu, C. Yang, M. Restaino, L. Lin, 3D printed microfluidics and microelectronics, Microelectron. Eng. 189 (2018) 52–68.

[6] N. Li, K. Han, W. Spratt, S. Bedell, J. Ren, O. Gunawan, J. Ott, M. Hopstaken, C. Cabral Jr., F. Libsch, et al., Dust-sized high-power-density photovoltaic cells on Si and SOI substrates for wafer-level-packaged small edge computers, Adv. Mater. 32 (49) (2020) 2004573.

[7] E. Li, L. Zeng, Z. Zhou, X. Chen, Edge AI: On-demand accelerating deep neural network inference via edge computing, IEEE Trans. Wireless Commun. 19 (1) (2019) 447–457.

[8] A. Nikoukar, S. Raza, A. Poole, M. Güneş, B. Dezfouli, Low-power wireless for the internet of things: Standards and applications, IEEE Access 6 (2018) 67893–67926.

[9] W. Mrozik, M.A. Rajaeifar, O. Heidrich, P. Christensen, Environmental impacts, pollution sources and pathways of spent lithium-ion batteries, Energy Environ. Sci. 14 (12) (2021) 6099–6121.

[10] N.A. Bhatti, M.H. Alizai, A.A. Syed, L. Mottola, Energy harvesting and wireless transfer in sensor network applications: Concepts and experiences, ACM Trans. Sensor Netw. 12 (3) (2016) 1–40.

[11] J. Van Der Woude, M. Hicks, Intermittent computation without hardware support or programmer intervention, in: 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), 2016, pp. 17–32.

[12] N.A. Bhatti, L. Mottola, HarvOS: Efficient code instrumentation for transiently-powered embedded sensing, in: 2017 16th ACM/IEEE International Conference on Information Processing in Sensor Networks, IPSN, IEEE, 2017, pp. 209–220.

[13] A. Maioli, L. Mottola, M.H. Alizai, J.H. Siddiqui, On intermittence bugs in the battery-less internet of things (wip paper), in: Proceedings of the 20th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems, 2019, pp. 203–207.

[14] M. Afanasov, N.A. Bhatti, D. Campagna, G. Caslini, F.M. Centonze, K. Dolui, A. Maioli, E. Barone, M.H. Alizai, J.H. Siddiqui, et al., Battery-less zero-maintenance embedded sensing at the mithræum of circus maximus, in: Proceedings of the 18th Conference on Embedded Networked Sensor Systems, 2020, pp. 368–381.

[15] S. Ahmed, M.H. Alizai, J.H. Siddiqui, N.A. Bhatti, L. Mottola, Towards smaller checkpoints for better intermittent computing, in: 2018 17th ACM/IEEE International Conference on Information Processing in Sensor Networks, IPSN, IEEE, 2018, pp. 132–133.

[16] S. Ahmed, N.A. Bhatti, M.H. Alizai, J.H. Siddiqui, L. Mottola, Fast and energy-efficient state checkpointing for intermittent computing, ACM Trans. Embedded Comput. Syst. (TECS) 19 (6) (2020) 1–27.

[17] N. Bhatti, L. Mottola, Efficient state retention for transiently-powered embedded sensing, in: International Conference on Embedded Wireless Systems and Networks, 2016, pp. 137–148.

[18] S. Ahmed, A. Bakar, N.A. Bhatti, M.H. Alizai, J.H. Siddiqui, L. Mottola, The betrayal of constant power× time: Finding the missing joules of transiently-powered computers, in: Proceedings of the 20th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems, 2019, pp. 97–109.

[19] S. Ahmed, N.A. Bhatti, M. Brachmann, M.H. Alizai, A survey on program-state retention for transiently-powered systems, J. Syst. Archit. 115 (2021) 102013.

[20] F. Samie, V. Tsoutsouras, D. Masouros, L. Bauer, D. Soudris, J. Henkel, Fast operation mode selection for highly efficient iot edge devices, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 39 (3) (2019) 572–584.

[21] C.S. Mishra, J. Sampson, M.T. Kandemir, V. Narayanan, Seeker: Synergizing mobile and energy harvesting wearable sensors for human activity recognition, 2022, arXiv preprint arXiv:2204.13106.

[22] K. Ganesan, J. San Miguel, N.E. Jerger, The what's next intermittent computing architecture, in: 2019 IEEE International Symposium on High Performance Computer Architecture, HPCA, IEEE, 2019, pp. 211–223.

[23] D.-S. Perju, Applying Memoization as an Approximate Computing Method for Transiently Powered Systems (Dissertation) (Ph.D. thesis), KTH Royal Institute of Technology, 2019.

[24] G. Lukosevicius, A.R. Arreola, A.S. Weddell, Using sleep states to maximize the active time of transient computing systems, in: Proceedings of the Fifth ACM International Workshop on Energy Harvesting and Energy-Neutral Sensing Systems, 2017, pp. 31–36.

[25] S. Mittal, A survey of techniques for approximate computing, ACM Comput. Surv. 48 (4) (2016) 1–33.

[26] A.A. Shah, N.A. Bhatti, K. Dev, B.S. Chowdhry, MUHAFIZ: Iot-based track recording vehicle for the damage analysis of the railway track, IEEE Internet Things J. 8 (11) (2021) 9397–9406.

[27] F. Bambusi, F. Cerizzi, Y. Lee, L. Mottola, The case for approximate intermittent computing, in: 2022 21st ACM/IEEE International Conference on Information Processing in Sensor Networks, IPSN, IEEE, 2022, pp. 463–476.

[28] S. Ahmed, Q. Ul Ain, J.H. Siddiqui, L. Mottola, M.H. Alizai, Intermittent computing with dynamic voltage and frequency scaling, in: EWSN, 2020, pp. 97–107.

[29] A.Y. Majid, C.D. Donne, K. Maeng, A. Colin, K.S. Yildirim, B. Lucia, P. Pawełczak, Dynamic task-based intermittent execution for energy-harvesting devices, ACM Trans. Sensor Netw. 16 (1) (2020) 1–24.

[30] W. Liu, F. Lombardi, M. Shulte, A retrospective and prospective view of approximate computing [point of view, Proc. IEEE 108 (3) (2020) 394–399.

[31] S. Venkataramani, S.T. Chakradhar, K. Roy, A. Raghunathan, Approximate computing and the quest for computing efficiency, in: 2015 52nd ACM/EDAC/IEEE Design Automation Conference, DAC, IEEE, 2015, pp. 1–6.

[32] M. Gao, Q. Wang, M.T. Arafin, Y. Lyu, G. Qu, Approximate computing for low power and security in the internet of things, Computer 50 (6) (2017) 27–34.

[33] S. Ahmed, M. Nawaz, A. Bakar, N.A. Bhatti, M.H. Alizai, J.H. Siddiqui, L. Mottola, Demystifying energy consumption dynamics in transiently powered computers, ACM Trans. Embedded Comput. Syst. (TECS) 19 (6) (2020) 1–25.

[34] Microchip Technology, 2022, https://www.microchip.com/, (Accessed 10 September 2022).
[35] GitHub - EWouters/pydgilib: Pydgilib provides python bindings for Atmel Data Gateway Interface (DGI) devices and allows collecting and visualizing power usage data, 2022, https://github.com/EWouters/pydgilib, (Accessed 10 September 2022).
[36] Intel IoT Development Kit - Sample Videos, 2022, https://github.com/intel-iot-devkit/sample-videos, (Accessed 10 September 2022).

Kashif Javed is an Avionics Engineering student at the Institute of Avionics and Aeronautics (IAA), Air University Islamabad. He received a bachelor's degree in Electrical Engineering from Air University Islamabad. His research areas of interest are Intermittent computing and Internet of things (IoT).

Dr. Naveed Anwar Bhatti is an Assistant Professor at Air University (Pakistan). He completed his Ph.D. from Politecnico di Milano (Italy) in 2018, and subsequently served as an ERCIM postdoctoral fellow at RI.SE in Sweden for one and a half years. His research centers on the cutting-edge field of modern networked embedded systems, with a specific focus on cyber security. Dr. Bhatti has published extensively in leading industry journals and conferences, including SenSys, IPSN, EWSN, ICC, IEEE IoT, TECS, and TOSN. He has also been recognized for his work, receiving the best Ph.D. presentation award at IPSN 2016. In addition to his research, Dr. Bhatti has served as a reviewer for several prestigious journals, including ACM Computing Survey, IEEE Transactions on Vehicular Technology, and IEEE Transactions on Very Large-Scale Integration Systems. He has also been a member of the technical program committee for several premier industry events, including ICPADS 2019, AlgoSensor 2019, ISIoT 2019, and IoTDI 2019. He has also been the recipient of numerous grants in the areas of cyber security and the Internet of Things from both local and international funding agencies.

Dr. Mohammad Imran is Assistant Professor at Air University, Pakistan since July 2020. Before joining academia he worked in the industry for over 20 years, his main involvement being in design and development of embedded systems. He obtained his Ph.D. in 2016 from Capital University of Science and Technology (Islamabad, Pakistan) specializing in cyber security. He has acted as reviewer for various IEEE conferences and journals including FIT, ICCWS, ICAI and IEEE Access. His research interests include security of embedded systems and cyber–physical systems, information security management, and malware analysis & classification.