ABSTRACT
Intermittent computing enables the functioning of computing systems under unstable power conditions. Designing such systems poses significant challenges due to the vast design space, including hardware and software parameters along with harvested energy availability. In this paper, we propose an analytical framework to predict an application’s execution time in intermittently powered systems, enabling rapid design space exploration. Our framework accounts for previously unexplored factors such as the effect of Equivalent Series Resistance (ESR) in capacitors and the choice of checkpoint strategies. With only one-time profiling, it estimates timings of different checkpoint techniques under various design configurations, with an average error of 4.7% under stable power and 10.4% under intermittent power. Furthermore, our evaluation reveals that neglecting checkpoint strategy in design can result in a 3.08x average slowdown compared to optimal setups.

CCS CONCEPTS
• Computer systems organization → Embedded and cyber-physical systems.

KEYWORDS
Intermittent Computing, Modeling, Design Space Exploration.

Permission to make digital or hard copies of part or all 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 third-party components of this work must be honored. For all other uses, contact the owner/author(s).
SISPAD ’24, August 5–7, 2024, Newport Beach, CA, USA
© 2024 Copyright held by the owner/authors.
ACM ISBN 979-8-4007-0688-2/24/08
https://doi.org/10.1145/3665314.3670799

1 INTRODUCTION
Advances in energy harvesting have enabled “battery-less” embedded systems, addressing challenges of batteries such as short lifetime and environmental impacts [10]. These systems require mechanisms to make forward progress amidst frequent power failures since environmental power sources are often inadequate for sustained continuous computing [18]. Intermittent computing addresses these challenges by utilizing Non-Volatile Memory (NVM) to regularly save the system states (checkpoint). Upon power restoration, the system can resume its execution from the saved context (recovery), rather than restarting from the program’s initial state.

Designing intermittent systems is often an optimization problem to find the best design configurations that meet specific requirements such as throughput or latency [7]. This search for optimal setups significantly impacts the cost and form factor of the resulting system. However, evaluating a wide range of configurations on real hardware is impractical due to the vast design space [5], which includes various factors such as capacitor size, core frequency, input power and harvesting efficiency. As a result, several modeling-based techniques have been proposed to evaluate the impact of different design configurations on the end-to-end execution time during early design stages, without relying on the actual hardware [5, 7, 20, 21].

In the meantime, recent studies have identified additional design aspects that significantly influence performance, but are often overlooked in the existing models. One example is the Equivalent Series Resistance (ESR) of capacitors, which acts as an inherent internal resistance. As demonstrated by a prior study [19], disregarding ESR can result in inefficient energy utilization or even failure of the system. It is also critical to include ESR when designing timing models, since ESR introduces extra voltage drops of the capacitor. This leads to more frequent power failures of the system, which in turn, significantly impact program execution time (Sec. 4.3.1).

Checkpoint techniques (CTs) are another important factor that is underexplored in the existing works. Various CTs, including static [12–14, 18] and Just-In-Time (JIT) checkpointing [11, 15, 16], have been proposed to minimize the cost of intermittent executions. As we will show, the choice of CT can heavily affect the application performance, by an order of magnitude or more (Sec. 4.3.1). This huge variation implies that analytical models should be able to explore different CTs to achieve truly optimal configuration. Unfortunately, existing works rely on profiling information from the checkpoint-equipped binaries, confining their timing models to the specific CT that is used for the profiling [5, 7, 20, 21]. To efficiently incorporate CT as a design parameter, overheads of CTs should be modeled based on the CT-neutral profiling information. This approach also eliminates the need for actual implementations of various CTs at design stages, easing the burden of system designers.

In this paper, we propose an analytical model to estimate the end-to-end execution time of intermittent applications in various configurations, enabling rapid early-stage design space exploration (DSE) of intermittent systems. Our model offers timing estimations for various CTs and different ESRs, based on just one-time profiling of the unmodified original program. Our validation shows that the proposed model achieves high accuracy in timing estimates, with an average error of 4.7% under stable power and 10.4% under...
2 BACKGROUND AND RELATED WORK

Intermittent Systems. An intermittent system typically consists of a power management subsystem and a computing subsystem. The power management subsystem includes an energy storage component, usually a small capacitor, where the harvested energy is stored. Once the capacitor voltage reaches a power-on threshold \( V_h \), it supplies stable voltage power to the computing system. When the capacitor voltage hits a power-off threshold \( V_l \), the power supply stops. This power cycle can be short (e.g., less than a second), leading to frequent power failures in the computing subsystem.

The computing subsystem consists of an NVM and a low-power microcontroller (MCU). To maintain progress under power failures, it regularly backs-up volatile system states (i.e., registers, TCB, and SRAM data) to NVMs. Such backup, known as checkpointing, involves a large number of power-intensive NVM accesses. Consequently, several checkpoint strategies have been studied to support forward progress with minimal overhead.

Equivalent Series Resistance (ESR). ESR has been an unexplored parameter in intermittent systems until the recent study [19]. ESR acts as a capacitor’s internal resistance, leading to a voltage drop (IR drop) when current is drawn from the capacitor. Fig. 1a shows a capacitor voltage trace obtained from our measurement. The voltage drops instantly at the time of power-on and is recovered when the MCU stops drawing the current (depicted in red). As a result, the system may use only a portion of energy in the capacitor (from \( V_h \) to \( V_l + \delta \) rather than \( V_l \)), leading to more frequent power-offs of the system than expected. This is particularly relevant in systems utilizing supercapacitors, which typically exhibit high ESR [19].

Checkpoint Techniques (CTs). JIT checkpointing [11, 15, 16] is a widely adopted strategy designed to perform checkpoint only once per power cycle, right before the power failure. To keep the checkpoint data size small and consistent, it utilizes NVM as the main memory, avoiding the entire need for checkpointing the memory. The checkpoint is initiated by an external power monitor that sends an interrupt when the capacitor voltage reaches \( V_l \). Upon receiving the interrupt, the system immediately executes checkpoint and is turned off to preserve memory consistency.

### Table 1: Configurable system design parameters.

<table>
<thead>
<tr>
<th>Symbols</th>
<th>Descriptions</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>( C )</td>
<td>capacitance of the capacitor</td>
<td>47\text{mF}</td>
</tr>
<tr>
<td>( R_{\text{cap}} )</td>
<td>ESR of capacitor</td>
<td>22\text{^}Ω</td>
</tr>
<tr>
<td>( P_l )</td>
<td>input power in W</td>
<td>20\text{mW}</td>
</tr>
<tr>
<td>( V_h )</td>
<td>capacitor voltage to turn on the system</td>
<td>5V</td>
</tr>
<tr>
<td>( V_l )</td>
<td>capacitor voltage to turn off the system</td>
<td>3.5V</td>
</tr>
<tr>
<td>( f )</td>
<td>core frequency</td>
<td>10\text{^}Hz</td>
</tr>
<tr>
<td>( V_{\text{op}} )</td>
<td>operating voltage</td>
<td>3.3V</td>
</tr>
<tr>
<td>( \eta_{\text{harrow}} )</td>
<td>energy harvester charge efficiency</td>
<td>0.93 [23]</td>
</tr>
<tr>
<td>( \eta_{\text{buck}} )</td>
<td>buck converter efficiency</td>
<td>0.89 [23]</td>
</tr>
</tbody>
</table>

Static checkpointing [12–14, 18] inserts checkpoint instructions directly in the program binary (e.g., at the start of loop iterations [12, 18]), typically with compiler assistance. To prevent excessive checkpoint executions, static checkpointing usually includes a mechanism to control their execution frequency. Two notable approaches are: voltage-ignorant (VI) and voltage-aware (VA). VI employs a counter that increments at each checkpoint trigger, and performs a checkpoint only when this counter reaches a user-configurable threshold. After the checkpoint execution, the counter is reset, ensuring that only a certain ratio \( \lambda_{\text{ct}} \) of checkpoints are executed. VA manages the checkpoint execution frequency with an aid of external voltage monitor [18], as in JIT. In VA, each checkpoint trigger involves checking the capacitor voltage, and checkpoint is performed if the voltage is below a predefined threshold \( V_{\text{trig}} \).

Related Work. Several modeling-based approaches have been developed to address the challenges of DSE in intermittent systems. EH Model [20, 21] proposed an energy model that predicts how much of the energy is spent on forward progress. PES [7] proposed a model to estimate the throughput of IoT applications with configurable sampling and communication frequencies. ETAP [5] introduced an approach to predict the energy and timing of a program from compiler analysis. However, they rely on detailed profiling data, which are often hard to collect (e.g., energy per each instruction [5], program phase [7], or backup [20, 21]). Furthermore, their abilities to explore various CTs are limited, as they require profiling data from the checkpoint-equipped binary. Also, none of these models consider the impact of ESR, thereby confining their utility to low-ESR systems. Our work addresses these limitations.

3 OUR APPROACH

3.1 Overview

3.1.1 Design Goal. The output of our model is an end-to-end latency of a program, including execution time and capacitor recharge time, under a given design configuration. We model a system that has a capacitor with a capacitance of \( C \) and an ESR of \( R_{\text{esr}} \), and collects \( P_l \) watt of power\(^1\). The system is turned on when the capacitor voltage becomes \( V_h \) and is turned off at \( V_l \). These are configurable design parameters, and the entire list can be found in Table 1.

Since the timing of an intermittent system is heavily affected by available power, our model comprises two interrelated components: a timing model and a power model. The timing model predicts the

\(^1\)We employ a single input parameter to represent average power as in [7, 20, 21] and leave the modeling for dynamic power profiles as a future work. Sec. 5.2 demonstrates that it does not limit our ability to assess various power requirements during DSE.
3.2 Base Model

3.2.1 Modeling ESR. ESR can be modeled as a resistor in series with the capacitor. Fig. 1b illustrates a circuit diagram of the load side of the system, at the moment of power-on. \( V_c \) is the internal voltage of the capacitor and \( V_0 \) is the voltage after the ESR effect that the system actually uses. To estimate the voltage drop \( d = V_c - V_0 \) due to ESR, we leverage the fact that power remains constant before and after the buck converter.

\[
V_1 \cdot I_1 \cdot \eta_{buck} = V_{op} \cdot I_2
\]

Also, \( V_1 \) can be represented with \( V_h \) and IR drop from \( R_{esr} \).

\[
V_1 = V_h - I_1 \cdot R_{esr} = V_h - \frac{V_{op} \cdot I_2 \cdot R_{esr}}{\eta_{buck} \cdot V_1}
\]

The solution of this quadratic equation of \( V_1 \) can be calculated as:

\[
V_1 = \frac{1}{2} \left( V_h \pm \sqrt{V_h^2 - \frac{4 \cdot I_1 \cdot V_{op} \cdot R_{esr}}{\eta_{buck}}} \right)
\]

We select \( V_1 \) that satisfies \( V_1 < V_i < V_h \). This simplified model significantly contributes to the accuracy of the timing model: reduction of validation error from 44.3\% to 10.4\% (Sec. 4.3.1).

3.2.2 Base Model. The base timing model addresses the timing aspects that are not related to the individual CTs. The end-to-end execution time \( T_{tot}^* \) for given the number of power-offs \( v \) consists of four components:

\[
T_{tot}^*(v) = T_{orig} + T_{ckpt}^*(v) + T_{recovery}^*(v) + T_{recharge}^*(v)
\]

\( T_{orig} \) is for time executing the original program, \( T_{ckpt}^*(v) \) is for overhead from checkpoints, \( T_{recovery}^*(v) \) denotes the time taken due to recovery and \( T_{recharge}^*(v) \) is the time spent for recharge. The + indicates that the term is dependent on the CTs, and should be extended by individual checkpoint models.

\( T_{orig} \) can be modeled as a linear combination of \( N_c \) and \( C_x \), and a constant initialization cost \( T_{init} \), given that the MCUs used in most intermittent systems in the literature follow an in-order architecture, e.g., TI MSP430 or ARM Cortex M0/M4. We use \( \tau = 1/f \) to denote the latency per cycle.

\[
T_{orig} = T_{init} + \tau \cdot \sum_{x \in \Gamma} N_x \cdot C_x
\]

The overhead of checkpoint consists of three components.

\[
T_{ckpt}^*(v) = O_{sta}^* + v \cdot (O_{dyn}^* + O_{rb}^*)
\]

\( O_{sta}^* \) denotes the static overhead, which is independent to \( v \), \( O_{dyn}^* \) represents the dynamic overhead that increases proportionally with \( v \) and \( O_{rb}^* \) accounts for the rollback cost, which is the lost progress that executed after the last checkpoint so that cannot be recovered. \( T_{recovery}^* \) is proportional to \( v \), system initialization cost and the checkpoint-wise recovery overhead \( O_{rec}^* \).

\[
T_{recovery}^*(v) = v \cdot (T_{init} + O_{rec}^*)
\]

When \( v > 0 \), the system experiences power-off and requires capacitor recharges. We denote the average current consumption of the system as \( I \), which is approximated as \( r \cdot I_{nom} + (1 - r) \cdot I_n \), given that \( r \) is a ratio of time consumed in NVM accesses over the total program execution. We use \( d_I \) to denote the voltage drop due to ESR when the current consumption of the board is \( I \), which
can be computed from Equation 4 (i.e., \(d_t = V_b - V_i\) with \(I_2 = I\)).

Also, \(E(V_b, V_i) = \frac{1}{2}C(V_b^2 - V_i^2)\) denotes the energy that capacitor holds between the voltage \(V_b\) and \(V_i\). Then, the time necessary to replenish the energy consumed can be modeled as:

\[
T_{\text{recharge}}(v) = \frac{\nu \cdot E(V_b, V_i + d_j)}{(P_i \cdot \eta_{\text{harv}})}
\]  

(9)

3.2.3 Power Model. Power model estimates \(\nu\) by modeling the power cycle duration \(T_{\text{pc}}\), which represents the continuous execution time without power failure. As the capacitor is charged concurrently with the system execution, we account for the recharging current and model the effective current consumption \(I_{\text{drain}}\).

\[
I_{\text{drain}} = \frac{\bar{I}}{(P_i \cdot \eta_{\text{harv}} \cdot \eta_{\text{back}}) / V_{\text{op}}}
\]  

Then, \(T_{\text{pc}}\) can be derived by dividing the usable energy of the system by its effective power consumption.

\[
T_{\text{pc}} = \frac{\eta_{\text{back}} \cdot E(V_b, V_i + d_j)}{I_{\text{drain}} \cdot V_{\text{op}}}
\]  

(11)

The number of poweroffs \(\nu\) is then calculated as the total time required for the computation divided by \(T_{\text{pc}}\).

\[
\nu = \frac{T_{\text{tot}}(v) - T_{\text{recharge}}(v)}{T_{\text{pc}}}
\]  

(12)

3.2.4 Finding \(\nu\). To solve Equation 12, we employ fixed-point iteration [3], an iterative method for finding roots of equation. If we denote the equation as \(f(v)\), the sequence \(\{v_n\}\) converges to the solution given that \(v_n = f(v_{n-1})\) and \(v_0\) is a random initial value. We use \(v_0 = 0\) and compute \(v_n\) until it converges. Since \(\nu\) is an integer, the exact solution can be found with a small number of iterations.

3.3 Checkpoint-Wise Models

Thanks to our base model, timing estimation of individual CT requires modeling of only four terms: \(O_{\text{sta}}^r\), \(O_{\text{dyn}}^r\), \(O_{\text{rb}}^r\) and \(O_{\text{rec}}^r\). We highlight the extensibility of our model by providing modelings for three different checkpoint styles.

3.3.1 JIT (Just-In-Time) Checkpointing (JIT). In JIT, accesses to SRAMs in original program turn to NVM accesses. As a result, static overhead can be modeled as follow. We denote the required number of NVM accesses for an word access as \(Q_{\text{nom}} = \text{(word size)} / B_{\text{nom}}\).

\[
O_{\text{sta}}^{\text{JIT}} = \tau \cdot \{N_{\text{frw}} \cdot (C_n - C_l) + N_i \cdot (C_n - C_s)\} \cdot Q_{\text{nom}}
\]  

(13)

Before poweroff, only the registers and TCB are checkpointed, as other program data is stored in NVM. \(S_{\text{ctb}} / B_{\text{nom}}\) represents the required number of NVM accesses to checkpoint TCB.

\[
O_{\text{dyn}}^{\text{JIT}} = \tau \cdot \{N_{\text{reg}} \cdot Q_{\text{nom}} \cdot C_n + S_{\text{ctb}} / B_{\text{nom}} \cdot (C_n + C_s)\}
\]  

(14)

Since the system instantly stops after checkpointing, no rollback overhead exists; thus \(O_{\text{rb}}^{\text{JIT}} = 0\). Recovery is a reverse of checkpointing in JIT, i.e., copying back the registers and TCB from NVM.

\[
O_{\text{rec}}^{\text{JIT}} = \tau \cdot \{N_{\text{reg}} \cdot Q_{\text{nom}} \cdot C_n + S_{\text{ctb}} / B_{\text{nom}} \cdot (C_n + C_s)\}
\]  

(15)

3.3.2 Voltage-Ignorant Static Checkpointing (VI). The static overhead of VI is modeled as follows, given that \(C_{\text{ct}}^n\) is a cost for checking counter and checkpoint instructions are inserted at loop headers.

\[
O_{\text{sta}}^{\text{VI}} = \tau \cdot (c_{\text{ct}}^n \cdot N_{\text{loop}} + \lambda_{\text{vi}} \cdot N_{\text{loop}}(O_{\text{stacking}} + O_{\text{copy}}))
\]  

(16)

The term \(\lambda_{\text{vi}} \cdot N_{\text{loop}}\) represents the number of checkpoint executions. Each checkpoint execution incurs overhead of system call setup \((O_{\text{stacking}})\) and data copy to NVM \((O_{\text{copy}})\). We modeled \(O_{\text{stacking}} = N_{\text{reg}} \cdot (C_n + C_s)\), accounting for costs of register saving and recovery. The cost of copying the registers, memory and TCB is modeled as:

\[
O_{\text{copy}} = N \cdot (C_s + C_{\text{nal}}),
\]  

where \(N = N_{\text{reg}} \cdot Q_{\text{nom}} + S_{\text{w}} + S_{\text{tcb}} / B_{\text{nom}}\)

(17)

VI has no additional checkpoint cost proportional to \(\nu\); thus \(O_{\text{rb}}^{\text{VI}} = 0\). We estimate the loss from rollback as half of the average interval between checkpoints. This can be modeled as:

\[
O_{\text{rb}}^{\text{VI}} = \tau \cdot (T_{\text{tot}}(v) / (2 \cdot \nu)) \cdot N_{\text{loop}}
\]  

(18)

The cost of recovery is a reverse of \(O_{\text{copy}}\); \(O_{\text{rec}}^{\text{VI}} = N \cdot (C_s + C_{\text{nal}})\).

3.3.3 Voltage-Aware Static Checkpointing (VA). Since VA and VI are in the same checkpoint style, they have the same model except for two parameters: the voltage checking cost \(C_{\text{v}}\), which substitutes \(C_{\text{ct}}^n\), and execution rate \(\lambda_{\text{va}}\) which is used instead of \(\lambda_{\text{vi}}\). We describe the modeling of \(\lambda_{\text{va}}\) and omit redundant model descriptions.

VA consists of two phases in terms of power consumption. In execution phase, where the voltage is higher than \(V_{\text{trig}}\), system does not access NVM and consumes current of \(I\). On the other hand, in checkpoint phase where the voltage is below \(V_{\text{trig}}\), heavy NVM accesses happen as all checkpoint instructions are executed. This results in rapid increase in the current draw up to approximately \(I_{\text{nom}}\), leading to further voltage drop due to ESR. As a result, when the system enters the checkpoint phase, the power-off threshold arises to \(V_i' = V_i + d_{\text{nom}}\). We model \(\lambda_{\text{va}}\) as a ratio of the time spent in checkpoint phase \((T_1)\) to the total execution time \((T_1 + T_2)\), where \(T_2\) is time consumed in execution phase. The internal capacitor voltage when system observes \(V_{\text{trig}}\) is denoted as \(V_i' = V_{\text{trig}} + d_j\).

\[
\lambda_{\text{va}} = \frac{T_1}{T_1 + T_2}, \quad \text{where} \quad T_1 = \frac{E(V_i', V_i')}{V_{\text{op}} \cdot I_{\text{nom}}} \quad \text{and} \quad T_2 = \frac{E(V_b, V_i')}{V_{\text{op}} \cdot I}
\]  

(19)

4 VALIDATION

4.1 Setup

We measure the execution time of four benchmarks (basicmath, fft, crc and sha) from MiBench [8] by running them under various environments and compare these against the predicted time by our work. Application-dependent parameters (Table 3) are obtained using the gem5 [2] simulator. Unless stated otherwise, the example values in Table 1 and 2 are used for the modeling. We implemented the three CTs (JIT, VI and VA) considered in Sec. 3.3 on FreeRTOS [6], a widely used real-time OS. The measurements are done on a custom-built board based on Arm Cortex-M4 core (STM32L496 [22]) equipped with 1MB FRAM. TI BQ25570 [23] is used to manage the capacitor charge and the power regulation.
4.2 Accuracy of Timing Model

First, we evaluate the timing model under stable power ($v = 0$), in order to exclude the effect of the power model. Fig. 2 shows the measured and modeled timings of the original program at 10 MHz, with different CTs. We observe an average error of 3.3% in the original program and of 4.7% across the CTs, for the two core frequencies.

The observed error in our model comes from several design choices. For example, we use single latency for each instruction class in $\Gamma$ to simplify profiling, despite the actual latency varying with execution context [1]. Also, static checkpointing modifies the code during compilation, potentially affecting register pressure and the subsequent compiler optimizations. Such impact is hard to predict as our design avoids profiling binaries other than the original program. Given these complexities, error under 5% offers a good tradeoff between accuracy and practicability, and the timing model provides a consistent base for estimating end-to-end timing.

4.3 Accuracy under Intermittent Power

4.3.1 Validation Results. To evaluate accuracy of model in unstable power, we measured the timing of each CT at configurations of $C = \{47, 104\}$ and $f = \{10, 50\}$, under input currents ranging from 10 to 35 mA at 2.6 V. To account for variability in measurements, 5 samples are measured and the average values are used.

Fig. 3 shows the detailed results for one configuration ($C = 47$ mF and $f = 10$ MHz). Different colors in the figure denote different CTs. Solid lines represent the measured timings, while dotted lines depict the model’s estimations. It shows several interesting observations. For instance, one can observe a significant variance in the efficiency of CTs, depending on the benchmarks (more discussions on Sec. 5.1). In terms of validation error, the figure shows that the error is distributed fairly even across different input currents, suggesting that our model effectively estimates the timing without high variance.

4.3.2 Source of Error. Beyond the design aspects of the model (Sec. 4.2), some other factors also influence model’s accuracy. First, there exists inherent variability in the measured times since the real execution involves non-deterministic physical behaviors [9]. Indeed, we observe 6.5% of the average variance in execution times measured under unstable powers, while measurements in stable power shows much more consistent value of 0.04%.

Secondly, since $v$ is an integer, a small prediction error in the number of power-offs can be amplified in $T_{tot}$. For example, consider a scenario that the model predicts the number of power-offs as 2.01 while the actual value is 1.99. This 1% error results in different $v$ values, i.e., $\lfloor 2.01 \rfloor = 2$ and $\lfloor 1.99 \rfloor = 1$, adding one more recharge cycle and associated checkpoint overhead, leading to greater errors especially when $v$ is small. Given the variability in the real systems, such errors are inevitable even if the model is highly accurate.

4.3.3 Comparison to Existing Works. Directly comparing our work with previous studies is not trivial due to a difference in methodology. While our work estimates the timing of CT-induced overheads, existing studies may directly obtain such information from checkpoint-equipped binaries. Despite this, our work presents an accuracy comparable to these prior works (e.g., 1.5–10.8% for ETAP [5], 1.6–10.4% for PES [7], and 1.6–7.0% for EH model [20, 21]) while requiring the most minimal profiling information (Sec. 2). Moreover, our design allows exploring impacts of different CTs, which is demonstrated to significantly affect the performance (Fig. 3), providing deeper insights when designing intermittent systems.

5 EVALUATION

5.1 Sensitivity of Checkpoint Techniques

In this evaluation, we assess the sensitivity of CTs for design configurations. Fig. 5a shows the modeled execution times of each CT, in 64 different design configurations, for each benchmark. The result clearly shows that efficiency of CT is dependent on both the design and the running program. For instance, VA outperforms the other CTs in fft but is the least efficient in crc. In sha, J1T and V1 exhibit the least overhead depending on the hardware configurations.

Fig. 5b compares normalized execution times in the same settings when a single CT is uniformly applied, against scenarios where the optimal strategy is chosen for each case. The line graph represents the ratio of each CT being the best choice across all configurations.

Figure 2: Validation results under stable power at $f = 10$ MHz.

Figure 3: Measured and modeled execution time of benchmarks under various input currents ($C = 47$ mF, $f = 10$ MHz).

Figure 4: Average validation errors of various configurations.
These results strongly support the importance of considering CTs, as a critical factor in design of intermittent systems.

5.2 Design Space Exploration Case Study

In this evaluation, we show that our model efficiently solves one of the challenges motivating this work: rapidly finding the optimal design points, including CTs, under specific requirements. To this end, we evaluate 22,464 design points under two example requirements. In the first case (R1), user wants to run sha and needs the worst-case response time, which is modeled as $T_{tot} + T_{recharge}$, to be less than 30 s. The second case (R2) requires fft to execute continuously for at least for 2.5 s ($P_{ct} > 2.5$) within total execution time of 60 s ($T_{tot} < 60$), under limited input power ($P_{ct} < 39$ mW) resembling an outdoor environment with a 12.9 cm$^2$ solar panel [17].

Table 4 shows some Pareto-optimal design points found from the exploration (5 out of 14 shown). The suggested points cover all different options of the parameters, such as various input powers or capacitor sizes, while meeting the requirements. Depending on the available hw/sw and design objectives, designers can choose from a wide range of distinct options, including different CTs. Another practical advantage of the model lies in its lightweight nature. For this evaluation, it only takes 8.3 secs for the entire search. This indeed helps designers easily experiment with different options and requirements, which is highly desired in early-stage DSE.

## 6 CONCLUSION

In this work, we present an analytical model to estimate the timings of an application in various design options of intermittent systems. Along with the common hardware parameters, our model incorporates previously unexplored parameters, such as ESR and various CTs. Our validation shows an average error of only 10.4% in unstable power environments, without actual implementation of the CTs. Evaluation results indicate that the CT should be considered as a critical design factor, as ignoring it may result in a 3.08x overhead in execution time. Also, we demonstrate the practicability of our model by showing rapid DSE results for various scenarios covering more than 20k design points in just a few seconds.

**ACKNOWLEDGEMENT**

This work was supported by IITP grant funded by the Korea government (MSIT) (No.2021-0-00360).

**REFERENCES**