Rapid Hardware/Software Design Space Exploration for Efficient Intermittent Systems

Youngbin Kim
yb.kim@etri.re.kr
Electronics and Telecommunications Research Institute
Daejeon, Republic of Korea

Hyoseung Kim
hyoseung@ucr.edu
University of California Riverside
Riverside, CA, USA

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.

ACM Reference Format:
Youngbin Kim and Hyoseung Kim. 2024. Rapid Hardware/Software Design Space Exploration for Efficient Intermittent Systems. In *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED '24), August 5–7, 2024, Newport Beach, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3665314.3670799

1 INTRODUCTION

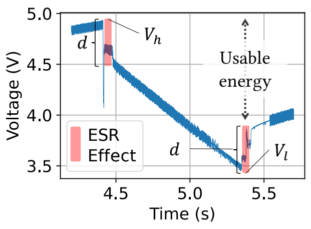
Advances in energy harvesting have enabled "battery-less" embedded systems, addressing challenges of batteries such as short lifetime and environmental impacts [10]. These systems require mechanisms to make forward progress amid frequent power failures since environmental power sources are often inadequate 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 application timings during early design stages, without relying on the actual hardware [5, 7, 20, 21].

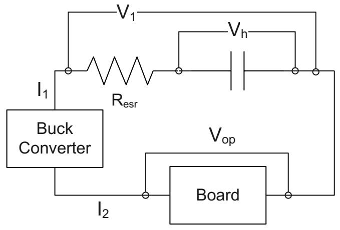
In the meantime, recent studies have identified additional design aspects that significantly influence performance, but are often overlooked in the existing models. One example is the Equivalent Series Resistance (ESR) of capacitors, which acts as an inherent internal resistance. As demonstrated by a 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



(a) Voltage trace of capacitor.



(b) Modeling ESR as a series resistor.

Figure 1: Modeling the voltage drop caused by ESR effect.

intermittent power, which is comparable to the results of existing work that relies on more detailed profiling. In addition, our results highlight that limiting the design space to a single CT can result in 3.08x average slowdown. In our DSE case studies, our model rapidly identifies optimal design configurations from a pool of over 20k options based on given requirements, within merely a few seconds.

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 $V_l + d$ 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 instantly executes checkpoint and is turned off to preserve memory consistency.

Table 1: Configurable system design parameters.

Symbols	Descriptions	Examples
$\mathbb{C}$	capacitance of the capacitor	47mF
R_{cap}	ESR of capacitor	22Ω
P_i	input power in W	20mW
V_h	capacitor voltage to turn on the system	5V
V_l	capacitor voltage to turn off the system	3.5V
f	core frequency	10Mhz
V_{op}	operating voltage	3.3V
η_{harv}	energy harvester charge efficiency	0.93 [23]
η_{buck}	buck converter efficiency	0.89 [23]

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 λ_{vi} 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_{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 $\mathbb{C}$ and an ESR of R_{esr}, and collects P_i 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.

Table 2: Architecture-dependent parameters.

Symbols	Descriptions	Examples
$C_x, x \in \Gamma$	Cycle for instruction class $x \in \Gamma$	$\{1, 3, 5, 2, 2\}$ [1]
C_{nl}	Cycle for load from NVM	2 [4, 22]
C_{ns}	Cycle for store to NVM	3 [4, 22]
N_{reg}	The number of registers (in words)	42 [1]
S_{tcb}	size of TCB (bytes)	88 [6]
B_{nvm}	NVM access bit-width (bytes)	2 [22]
I_n	avg current when not accessing NVM	25.01mA
I_{nvm}	avg current when accessing NVM	31.93mA

Table 3: Application-dependent parameters.

Symbols	Descriptions
$N_x, x \in \Gamma$	Instruction count of each instruction class $x \in \Gamma$
N_{loop}	total number of loop iterations
S_{ws}	the average working set size in bytes

total latency when the number of system power-offs, denoted as v, is given (Sec. 3.2.2). The power model estimates v, considering both power consumption and execution time of the program (Sec. 3.2.3). As these two models are coupled, we employ an iterative method to find v that satisfies the both equations (Sec. 3.2.4).

Another goal of our design is to provide the ability to evaluate various CTs from one-time profiling of the original program, i.e., no need to implement any CT for DSE. To achieve this, we structure our model into two parts: The base model, which covers checkpoint-neutral aspects and serves as a template for individual checkpoint models (Sec. 3.2); and the checkpoint-wise model, which extends the base model by providing modelings specific to each CT (Sec. 3.3). In this work, we showcase the extensibility of our model by presenting models for three representative CTs: `JIT`, `VI`, and `VA`.

3.1.2 Parameters and Notations. To account for varying latencies between instructions, we classify the instructions into five classes:

$$\Gamma = \{n, c, lro, lrw, s\} \tag{1}$$

where c is control, lro is load from the read-only section (e.g., `.rodata`), lrw is load from read-write section (e.g., `.data`), s is store and n is instructions not in the specified classes.

Architecture-dependent parameters are specific to the execution environment, such as the Instruction Set Architecture (ISA) and the underlying operating system (OS), not to the running program. Table 2 shows the list of these parameters. Apart from the cycles per each instruction class (i.e., $C_{x\in\Gamma}$), cycles for NVM access are modeled separately (i.e., C_{nl} and C_{ns}). We also parameterize the bit-width of NVM access (B_{nvm}) to reflect constraints on NVM access width. Most parameters are directly obtainable from manuals or datasheets, as denoted in Table 2. We only measured I_n and I_{nvm} for our custom board used in evaluation. Depending on the target hardware, these values can also be obtained without measurements.

Table 3 shows application-dependent parameters, which are obtained through profiling. These include the number of instructions per class (i.e., $N_{x\in\Gamma}$), the total number of loop iterations N_{loop}, and the average working set size S_{ws} (e.g. the size of stack and global data). Note that these parameters are from the original program, not modified one for checkpoint support, and can be easily obtained from simulation or measurement.

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_h is the internal voltage of the capacitor and V_1 is the voltage after the ESR effect that the system actually uses. To estimate the voltage drop $d = V_h - V_1$ due to ESR, we leverage the fact that power remains constant before and after the buck converter.

$$V_1 \cdot I_1 \cdot \eta_{buck} = V_{op} \cdot I_2 \tag{2}$$

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

$$V_1 = V_h - I_1 \cdot R_{esr} = V_h - \frac{V_{op} \cdot I_2 \cdot R_{esr}}{\eta_{buck} \cdot V_1} \tag{3}$$

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

$$V_1 = \frac{1}{2} \cdot \left(V_h \pm \sqrt{V_h^2 - \frac{4 \cdot I_2 \cdot V_{op} \cdot R_{esr}}{\eta_{buck}}} \right) \tag{4}$$

We select V_1 that satisfies $V_l < V_1 < V_h$. 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 Timing 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:

$$T^*_{tot}(v) = T_{orig} + T^*_{ckpt}(v) + T^*_{recovery}(v) + T_{recharge}(v) \tag{5}$$

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_x 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.

$$T_{orig} = T_{init} + \tau \cdot \textstyle\sum_{x\in\Gamma} N_x \cdot C_x \tag{6}$$

The overhead of checkpoint consists of three components.

$$T^*_{ckpt}(v) = O^*_{sta} + v \cdot (O^*_{dyn} + O^*_{rb}) \tag{7}$$

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^*_{recv}.

$$T^*_{recovery}(v) = v \cdot (T_{init} + O^*_{recv}) \tag{8}$$

When $v > 0$, the system experiences power-offs and requires capacitor recharges. We denote the average current consumption of the system as $\bar{I}$, which is approximated as $r \cdot I_{nvm} + (1 - r) \cdot I_n$, 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., $d_I = V_h - V_1$ with $I_2 = I$). Also, $E(V_a, V_b) = \frac{1}{2}\mathbb{C}\{V_a^2 - V_b^2\}$ denotes the energy that capacitor holds between the voltage V_a and V_b. Then, the time necessary to replenish the energy consumed can be modeled as:

$$T_{recharge}(\nu) = \nu \cdot E(V_h, V_l + d_{\bar{I}})/(P_i \cdot \eta_{harv}) \tag{9}$$

3.2.3 Power Model. Power model estimates ν by modeling the power cycle duration T_{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_{drain}.

$$I_{drain} = \bar{I} - (P_i \cdot \eta_{harv} \cdot \eta_{buck})/V_{op} \tag{10}$$

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

$$T_{pc} = \frac{\eta_{buck} \cdot E(V_h, V_l + d_{\bar{I}})}{I_{drain} \cdot V_{op}} \tag{11}$$

The number of poweroffs ν is then calculated as the total time required for the computation divided by T_{pc}.

$$\nu = \left\lfloor \frac{T_{tot}(\nu) - T_{recharge}(\nu)}{T_{pc}} \right\rfloor \tag{12}$$

3.2.4 Finding ν. To solve Equation 12, we employ fixed-point iteration [3], an iterative method for finding roots of equation. If we denote the equation as $f(\nu)$, the sequence $\{\nu_n\}$ converges to the solution given that $\nu_n = f(\nu_{n-1})$ and ν_0 is a random initial value. We use ν_0=0 and compute ν_n until it converges. Since ν 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^*_{sta}, O^*_{dyn}, O^*_{rb} and O^*_{recv}. 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_{nvm} = (word size)$/B_{nvm}$.

$$O^{jit}_{sta} = \tau \cdot \{N_{lrw} \cdot (C_{nl} - C_l) + N_s \cdot (C_{ns} - C_s)\} \cdot Q_{nvm} \tag{13}$$

Before poweroff, only the registers and TCB are checkpointed, as other program data is stored in NVM. S_{tcb}/B_{nvm} represents the required number of NVM accesses to checkpoint TCB.

$$O^{jit}_{dyn} = \tau \cdot \{N_{reg} \cdot Q_{nvm} \cdot C_{ns} + \frac{S_{tcb}}{B_{nvm}} \cdot (C_{ns} + C_l)\} \tag{14}$$

Since the system instantly stops after checkpointing, no rollback overhead exists; thus $O^{jit}_{rb} = 0$. Recovery is a reverse of checkpointing in `JIT`, i.e., copying back the registers and TCB from NVM.

$$O^{jit}_{recv} = \tau \cdot \{N_{reg} \cdot Q_{nvm} \cdot C_{nl} + \frac{S_{tcb}}{B_{nvm}} \cdot (C_{nl} + C_s)\} \tag{15}$$

3.3.2 Voltage-Ignorant Static Checkpointing (`VI`*).* The static overhead of `VI` is modeled as follows, given that C^{vi}_c is a cost for checking counter and checkpoint instructions are inserted at loop headers.

$$O^{vi}_{sta} = \tau \cdot \{C^{vi}_c \cdot N_{loop} + \lambda_{vi} \cdot N_{loop}(O_{stacking} + O_{copy})\} \tag{16}$$

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

$$O_{copy} = N \cdot (C_l + C_{ns}), \text{ where } N = N_{reg} \cdot Q_{nvm} + \frac{S_{ws} + S_{tcb}}{B_{nvm}} \tag{17}$$

`VI` has no additional checkpoint cost proportional to ν; thus $O^{vi}_{dyn} = 0$. We estimate the loss from rollback as half of the average interval between checkpoints. This can be modeled as:

$$O^{vi}_{rb} = \tau \cdot T^{vi}_{tot}(0)/(2 \cdot \lambda_{vi} \cdot N_{loop}) \tag{18}$$

The cost of recovery is a reverse of O_{copy}; $O^{vi}_{recv} = N \cdot (C_s + C_{nl})$.

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^{va}_c, which substitutes C^{vi}_c, and execution rate λ_{va} which is used instead of λ_{vi}. We describe the modeling of λ_{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_{trig}, system does not access NVM and consumes current of $\bar{I}$. On the other hand, in *checkpoint* phase where the voltage is below V_{trig}, heavy NVM accesses happen as all checkpoint instructions are executed. This results in rapid increase in the current draw up to approximately I_{nvm}, leading to further voltage drop due to ESR. As a result, when the system enters the checkpoint phase, the power-off threshold arises to $V'_l = V_l + d_{I_{nvm}}$. We model λ_{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_{trig} is denoted as $V'_t = V_{trig} + d_{\bar{I}}$.

$$\lambda_{va} = \frac{T_1}{T_1 + T_2}, \text{ where } T_1 = \frac{E(V'_t, V'_l)}{V_{op} \cdot I_{nvm}} \text{ and } T_2 = \frac{E(V_h, V'_t)}{V_{op} \cdot \bar{I}} \tag{19}$$

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.

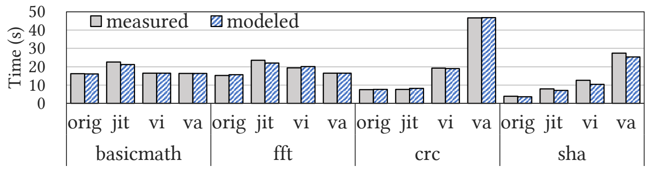


Figure 2: Validation results under stable power at f = 10Mhz.

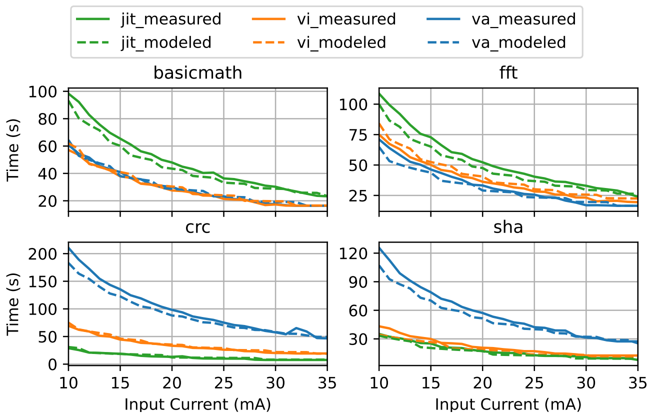


Figure 3: Measured and modeled execution time of benchmarks under various input currents ($\mathbb{C}$ = 47mF, f = 10Mhz).

4.2 Accuracy of Timing Model

First, we evaluate the timing model under stable power ($\nu = 0$), in order to exclude the effect of the power model. Fig 2 shows the measured and modeled timings of the original program at f=10, 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 Γ 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 $\mathbb{C} = \{47, 104\}$ and $f = \{10, 50\}$, under input currents ranging from 10 to 35mA at 2.6V. 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 ($\mathbb{C}$ = 47mF and f = 10Mhz). 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.

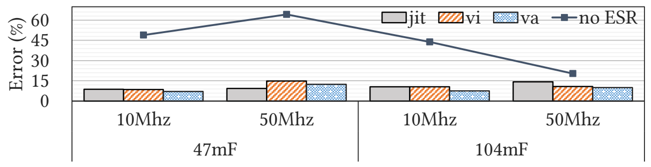


Figure 4: Average validation errors of various configurations.

Fig. 4 illustrates the average validation errors for all configurations. The line graph represents the average error of each configuration when ESR is not modeled. The figure shows that with ESR modeling, our model successfully estimates the timings under various configurations, with an average of 10.4% error and no case exceeding 15%. In contrast, the error increases to an average of 44.3% when ESR is not considered. This clearly reveals the importance of ESR effects in predicting behaviors of intermittent systems.

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 6.54% 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 ν 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 ν 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 ν is small. Given the variability in the real systems, such errors are inevitable even if the model is highly accurate.

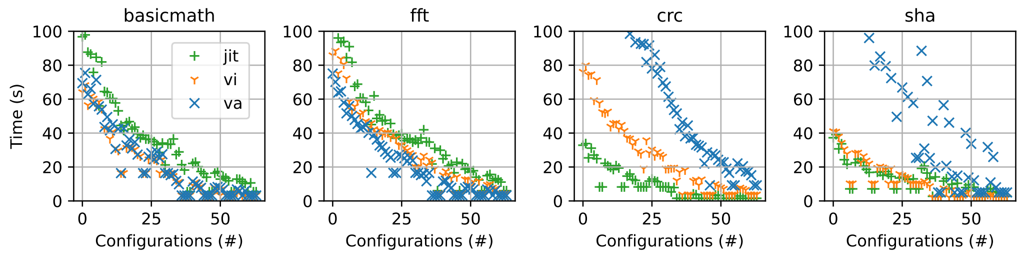
4.3.3 Comparison to Existing Works. Directly comparing our work with previous studies is not trivial due to a difference in methodology. While our work *estimates* the timing of CT-induced overheads, existing studies may directly obtain such information from checkpoint-equipped binaries. Despite this, 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

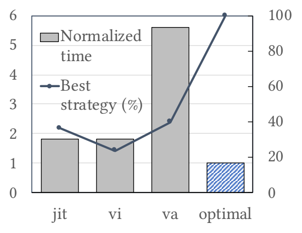
5.1 Sensitivity of Checkpoint Techniques

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`, `JIT` and `VI` 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.



(a) Modeled execution time of benchmarks under different configurations.



(b) Disadvantages of using single checkpoint technique.

Figure 5: Evaluation of the checkpoint efficiency for different benchmarks.

Table 4: Pareto-optimal design configurations under two example requirements, R1 and R2.

f	P_i (mW)	$\mathbb{C}$ (mF)	R_{cap}	Ckpt	T_{tot}	$T_{recharge}$	T_{pc}
R1: benchmark = sha, $T_{tot} + T_{recharge} < 30$							
50	13	20	20	vi	22.09	6.49	0.73
10	31	40	15	jit	25.13	4.39	1.53
R2: benchmark = `fft`, $P_i < 39, T_{pc} > 2.5, T_{tot} < 60$							
50	13	50	10	va	25.39	21.85	2.51
10	33	90	20	va	59.61	21.25	5.71
10	36	30	15	va	50.16	4.48	2.59

The result shows that applying a single CT for all design configurations leads to considerable overhead, with an average of 3.08x. Furthermore, we observe that no single CT consistently outperforms others in all scenarios. Even the most frequently optimal strategy, VA, emerges as the best choice in only 39.8% of the 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 30s. The second case (R2) requires `fft` to execute continuously for at least for 2.5s ($T_{pc} > 2.5$) within total execution time of 60s ($T_{tot} < 60$), under limited input power ($P_i < 39$mW) resembling an outdoor environment with a 12.9cm^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 of 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

[1] Arm 2021. *Armv7-M Architecture Reference Manual.* Arm.
[2] Nathan Binkert et al. 2011. The gem5 simulator. *ACM SIGARCH computer architecture news* 39, 2 (2011), 1–7.
[3] Richard L Burden. 2011. *Numerical analysis.* Brooks/Cole Cengage Learning.
[4] Cypress Semiconductor 2019. *FM22L16.* Cypress Semiconductor.
[5] Ferhat Erata et al. 2023. ETAP: Energy-aware timing analysis of intermittent programs. *ACM TECS* 22, 2 (2023), 1–31.
[6] FreeRTOS. 2023. FreeRTOS. https://freertos.org/
[7] Fatemeh Ghasemi et al. 2023. PES: An Energy and Throughput Model for Energy Harvesting IoT Systems. In *ISPASS.* IEEE, 13–23.
[8] Matthew R Guthaus et al. 2001. MiBench: A free, commercially representative embedded benchmark suite. In *WWC-4.* IEEE, 3–14.
[9] Josiah Hester et al. 2024. Ekho: Realistic and Repeatable Experimentation for Tiny Energy-Harvesting Sensors. In *SenSys.* ACM, 330–331.
[10] Josiah Hester and Jacob Sorber. 2017. The future of sensing is batteryless, intermittent, and awesome. In *SenSys.* 1–6.
[11] Hrishikesh Jayakumar et al. 2014. QuickRecall: A low overhead HW/SW approach for enabling computations across power cycles in transiently powered computers. In *VLSID.* IEEE, 330–335.
[12] Youngbin Kim et al. 2023. Liveness-Aware Checkpointing of Arrays for Efficient Intermittent Computing. In *DATE.* IEEE, 1–6.
[13] Vito Kortbeek et al. 2020. Time-sensitive intermittent computing meets legacy software. In *ASPLOS.* 85–99.
[14] Kiwan Maeng and Brandon Lucia. 2018. Adaptive dynamic checkpointing for safe efficient intermittent computing. In *OSDI.* 129–144.
[15] Kiwan Maeng and Brandon Lucia. 2019. Supporting peripherals in intermittent systems with just-in-time checkpoints. In *PLDI.* 1101–1116.
[16] Kiwan Maeng and Brandon Lucia. 2020. Adaptive low-overhead scheduling for periodic and reactive intermittent execution. In *PLDI.* 1005–1021.
[17] Powerfilm. 2024. Products. https://www.powerfilmsolar.com/products.
[18] Benjamin Ransford, Jacob Sorber, and Kevin Fu. 2011. Mementos: System support for long-running computation on RFID-scale devices. In *ASPLOS.* 159–170.
[19] Emily Ruppel et al. 2022. An architectural charge management interface for energy-harvesting systems. In *MICRO.* IEEE, 318–335.
[20] Joshua San Miguel et al. 2017. The EH model: Analytical exploration of energy-harvesting architectures. *IEEE Computer Architecture Letters* 17, 1 (2017), 76–79.
[21] Joshua San Miguel et al. 2018. The EH model: Early design space exploration of intermittent processor architectures. In *MICRO.* IEEE, 600–612.
[22] ST Microelctronics 2021. *RM0351 Reference Manual.* ST Microelctronics.
[23] Texas Instruments 2019. *bq25570 nano power boost charger and buck converter for energy harvester powered applications.* Texas Instruments.