Intermittent Systems at Small Scale: Execution Model and Design Guidelines

***Abstract*—Intermittent systems require software support to execute tasks amid frequent power failures. In designing such techniques, software designers rely on execution models that abstract hardware-level operations. In this paper, we propose an execution model that more accurately describes emerging intermittent systems with small energy storage. Our evaluation shows show that systems designed based on the traditional models can be up to 5.62x less power-efficient than expected and may result in unsafe checkpoint operations. Our design guidelines enhance the performance of existing static and dynamic checkpoint techniques by 3.04x and 2.85x on average, respectively.**

***Index Terms*—Intermittent Computing, Batteryless System.**

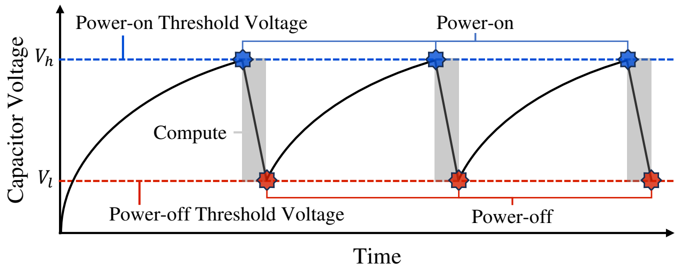


Fig. 1: Traditional execution model of intermittent systems.

I. Introduction

Batteryless systems are emerging as a promising future platform of Internet-of-Things (IoT) devices. These systems adopt a small capacitor as an energy storage and operate by harvesting power from environmental sources. This setup effectively addresses challenges associated with traditional battery-based systems, such as need for human intervention for recharging or replacement [1] and harmful environmental impacts [2]. They are also known as intermittent systems, since the computation happens intermittently during short periods only when there exist sufficient power to compute.

Intermittent systems require software supports to sustain long-running executions across power failures. During operation, volatile data (e.g., registers or SRAM data) must be saved to Non-Volatile Memory (NVM) through a process called checkpointing. When power is restored, this saved state is recovered to allow operations to resume the execution from the last checkpoint (recovery). In designing these state retention techniques, software designers rely on an *execution model* that abstracts hardware-level operations and represents behavior of intermittent systems necessary for software design.

Fig. 1 illustrates such execution model commonly adopted in the literature [3]–[10]. As energy accumulates, the voltage of the capacitor gradually increases. Once the voltage reaches the power-on threshold V_h, the collected power is supplied to the system. The system begins operation at this point, and execution is halted when the capacitor voltage reaches the power-off threshold V_l. Software designers aim to leverage this execution model to implement intermittent systems at minimal cost (e.g., by executing checkpoints just before reaching the power-off threshold [4], [5], [11]–[13]).

In the meantime, recent research on intermittent systems is increasingly exploring shorter operation times by using smaller capacitors. Operating on small capacitors is generally desirable, as it reduces device volume and enhances the responsiveness by enabling the system to wake up more frequently [5], [14], [15]. As a result, recent studies have targeted operation times in the range of tens of milliseconds [1], [13], [16]–[20] or even microseconds [16], [17]. However, as energy storage sizes decrease, the traditional execution model is failing to provide an accurate abstraction of actual execution behavior. The major source of this discrepancy is the buffering effects of the system's inherent capacitance, mostly coming from its decoupling capacitors. This factor has been overlooked in the traditional model, as the inherent capacitance was considered negligible compared to the main energy storage.

Decoupling capacitors are on-board capacitors that act as energy buffers. They are mandatory since the buffered energy prevent transient voltage drop when the system suddenly draws a large current, such as during checkpointing (Sec. II-A). However, their buffering effects also introduce discrepancies between the execution model and actual behavior. For example, during power-on, they rapidly charge from the energy storage, making capacitor voltage an unreliable estimate of available energy. This buffered energy also allows the system operate for a while at sub-normal voltages after the power supply is stopped. Additionally, between power cycles, decoupling capacitors discharge due to the resistance of the system, considerably lowering the power efficiency. In systems with smaller capacitors, these effects dominate the behaviors that are modeled in the traditional execution model. Consequently, highly efficient checkpoint techniques according to the traditional model may introduce substantial power overhead and even correctness issues in small-scale systems.

In this paper, we propose a new execution model for intermittent systems which accounts for the buffering effects of decoupling capacitors. In Sec. II, we demonstrate that understanding this model is critical for software designers: intermittent systems designed upon the traditional model can be up to 5.62x more energy-inefficient than expected and may fail to predict power-off timings accurately, leading to unsafe checkpointing. In Sec. III, we present design guidelines to

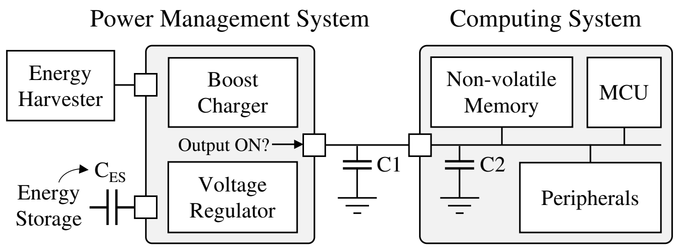


Fig. 2: A typical hardware setup of intermittent systems.

implement efficient intermittent systems with small energy storages, leveraging insights from our model. Our guidelines include designs which improve end-to-end execution latencies by 3.04x in static and 2.86x in dynamic checkpointing schemes on average, without incurring additional overhead.

II. Detailed Intermittent Execution Model

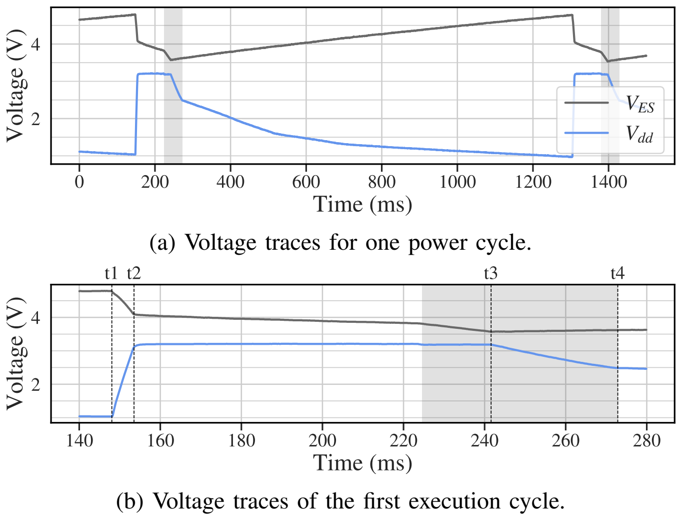
In this section, we describe our execution model and its implications for software design. Sec. II-A introduces the target architecture and the reference system used for evaluations. Sec. II-B presents our execution model, derived from key observations obtained through experimental results. In the following three sections, we discuss how this model affects both the power efficiency and correctness of software design. Finally, in Sec. II-F, we evaluate the effectiveness of our model across systems with various architectural configurations.

A. Target Architecture and Reference System

A typical intermittent system consists of two main components: a power management system and a computing system, as shown in Fig. 2. The power management system collects incoming energy into storage (C_{ES}) and supplies a stable-voltage current to the computing system. The computing system equips NVMs along with an MCU and peripherals, and utilize the NVMs for state retention between power failures.

This setup includes two notable decoupling capacitors that affect the execution model of intermittent systems. The first one (C1) is located in the power management system, required by voltage regulators to ensure stable operation. The second capacitor (C2) is part of the computing system and used to stabilize the operating voltage against sudden current draw.

Recent studies have increasingly explored 32-bit architectures for computing systems [17], [19], [21]–[29], as emerging applications on intermittent systems, such as Deep Neural Networks (DNNs) [7], [27]–[36], demand greater computational capabilities [14], [37]. In this context, we employ a custom-built board featuring a 32-bit ARM Cortex-M33 processor (STM32L5, operating at 16Mhz) with 512KB of Ferroelectric RAM (FRAM, Infineon FM22L16) as our reference system. For the power management system, we use a TI BQ25570-based board configured with V_h = 4.9V and V_l = 3.4V. We empirically select 22uF and 220uF capacitors for C1 and C2, respectively, as smaller capacitors fail to provide a reliable voltage for checkpoint and recovery. Sec. II-F evaluates the generality of our model across different architectures, such as systems with different NVM (e.g., Magnetic RAM, MRAM) and a 16-bit core (e.g., MSP430).



(a) Voltage traces for one power cycle.

(b) Voltage traces of the first execution cycle.

Fig. 3: Voltages trace of energy storage (V_{ES}) and V_{dd}.

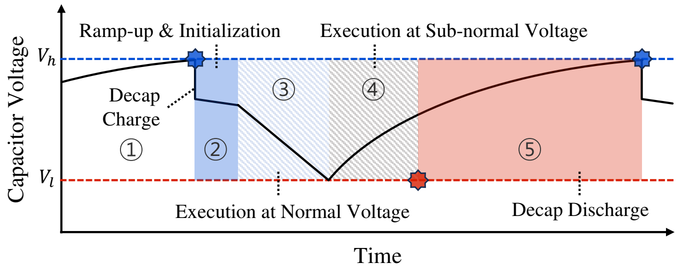


Fig. 4: Detailed execution model of intermittent systems.

B. Execution Model

To derive a general execution model with the effects of decoupling capacitors, we first present a sample measurement from our reference system. In this paper, we denote the voltage of the energy storage C_{ES} as V_{ES} and the MCU operating voltage as V_{dd}. To achieve an operation time of 50 ms under 1.5mA current supply, we use a 470uF capacitor for C_{ES}. Fig. 3a illustrates the voltage traces of V_{ES} and V_{dd} over a single power cycle. Note that V_{dd} is maintained by decoupling capacitors once the power supply from the power management system stops. The shaded areas represent the periods that system executes the application code.

Fig. 3b presents the first execution cycle in more detail. It reveals several differences between the traditional execution model and the actual operation. Among them, we highlight three key observations that affect software design decisions:

- **O1**: The capacitor voltage (V_{ES}) drops rapidly to charge decoupling capacitors when the system wakes up ($t1$–$t2$).
- **O2**: System operates at sub-normal voltage using decoupling capacitors, even after power supply stops ($t3$–$t4$).
- **O3**: Decoupling capacitors discharge while the system is powered off (after $t4$, as shown in Fig. 3a).

Fig. 4 illustrates our detailed execution model, incorporating these key observations. When V_{ES} reaches V_h, the voltage experience a rapid drop due to the buffering effects (①), instead of gradual decline. After initialization (②), the system

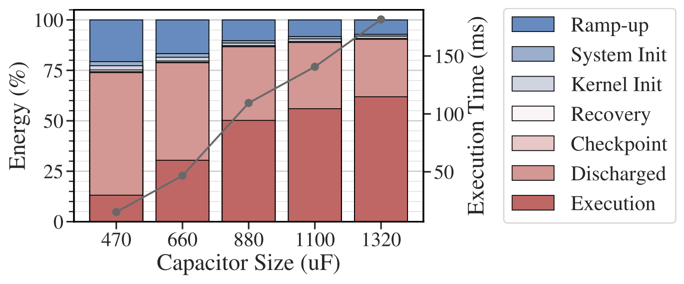
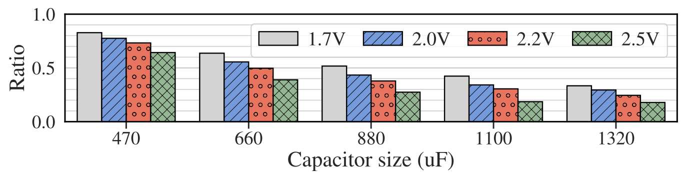
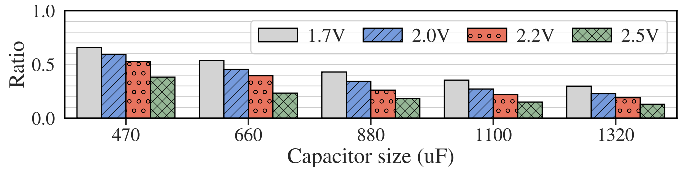


Fig. 5: Distribution of energy consumed in a power cycle.



(a) Input current = 1mA.



(b) Input current = 3mA.

Fig. 6: Ratio of sub-voltage operations in total execution time.

begins execution at normal operating voltage (③), 3.3V for example. When the voltage hits V_l, the power supply stops but system now starts to operate using the buffered energy (④). Since the voltage of the decoupling capacitors decreases as they discharge, the system executes at sub-normal voltage until it reaches the voltage level it cannot operate (e.g., 2.5V in Fig. 3). Finally, until the next power-on event, the remaining energy in decoupling capacitors continues to discharge (⑤).

When designing intermittent systems, particularly those utilizing small capacitors, understanding the effects described by this model is critical. In the following sections, we discuss impacts of our model to software design in more detail.

C. *Impact on Power Efficiency*

The traditional model implies that the energy consumed between V_h and V_l is entirely used in the computing system. However, our model reveals that considerable energy is used for charging the decoupling capacitors (**O1**) and dissipated during power-off durations (**O3**). This indicates that much smaller energy may be used for the useful computation compared to the designer's expectation.

Fig. 5 shows the distribution of the energy consumption for each stage of operation within one power cycle, averaged over 50 executions, where 1mA of input current is provided at 1.9V. The x-axis represents the size of C_{ES} and the line in the secondary axis represents the average operation times for application code. The checkpoint is executed by the interrupt from the power management system [4], [11], [20], [38], [39], which is generated when V_{ES} reaches V_l (3.4V). Note that this is the most efficient point for checkpoint execution according to the traditional model (i.e., just before the poweroff).

The results shows that significant energy is wasted in the decoupling capacitors. For example, in 470uF case, 60.7% of the energy is lost during the power-off duration (denoted as *Discharged*), leaving only 13.1% of the energy for computation. While the ratio of *Discharged* decreases with larger C_{ES}, it remains substantial; for example, in the 1320uF case, 28.5% of energy is discharged, which is still non-negligible.

This is because the discharging behavior can be modeled as an RC-discharging circuit (i.e., $q = CVe^{-\frac{t}{RC}}$), which exhibits an exponential discharge rate. Indeed, 50% of the energy is discharged within the first 161 ms in our measurements. Since recharging C_{ES} takes 2.13 secs even in 470uF configuration, most of the buffered energy is lost before the next power-on, regardless of the size of C_{ES}. As a result, the energy loss ratio due to discharging is larger with smaller capacitors.

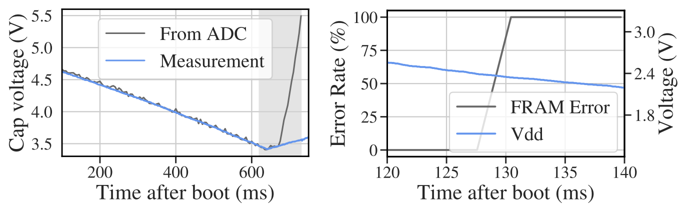
Another important observation is the error introduced by the traditional model. The traditional model expects both the energies, *Execution* and *Discharged*, are used for computation. This introduces huge errors, up to 5.62x in 470uF setup, for example. In the same context, the traditional model predicts that using a 470uF capacitor for C_{ES} instead of a 1320uF would result in only 1.22x overhead in energy efficiency, while the actual difference is 4.71x. This can significantly mislead system designers when they select capacitor sizes by considering tradeoffs between overall efficiency and reactiveness. In Sec. III, we explore strategies to minimize the inefficiencies caused by discharging when designing software techniques.

D. *Impact on Predicting Power Failures*

According to the traditional model, system states should be saved to NVM before V_{ES} reaches V_l, as the system is expected to halt at this point. On the other hand, our model shows that the system may continue operating using the energy stored in the decoupling capacitors (**O2**). Since modern MCUs can operate across a wide range of supply voltages (e.g., 1.7V to 3.6V in STM32L5 and MSP430), the computing system operates until the voltage of decoupling capacitors drops to the minimum operating level. This makes V_{ES} not a reliable indicator for the imminent power-off.

Fig. 6 presents the ratio of the times executed under sub-normal voltages to the total execution times, averaged over 30 measurements. The x-axis represents the sizes of C_{ES} and the colors indicate the voltage levels at which the system stops operation. We evaluate a range of stop voltages from 1.7V to 2.5V since not all components in the computing system may function at the lowest voltage level (Sec. II-E). Also, we examine two cases with input currents of 1mA (Fig. 6a) and 3mA (Fig. 6b), to assess the impact of input power.

The figure shows that a significant portion of MCU operation occurs at sub-normal voltages. For example, when 470uF capacitor is used at 1mA input current (Fig. 6a), 82.8% of computation takes place *after* the power-off threshold. This ratio decreases as the system stops earlier (reducing



(a) Analog-Digital Converter. (b) External FRAM.

Fig. 7: Incorrect operations at sub-normal voltages.

sub-voltage operation time) or the input current increases (extending operation time at normal voltage). However, at least 13.0% of computations are operated at sub-normal voltages even in highly optimistic configurations (1320uF in Fig. 6b).

These values can be directly translated to the inefficiencies of the systems based on the traditional model. For example, in the case of 470uF with a input current of 1mA, systems executing checkpoint at V_l may operate 16.3 ms. However, the system could operate for an additional 29.4 ms if the checkpoint can be delayed until 2.5V. At the next power-on, the decoupling capacitors discharge to similar voltage levels in both cases, as discussed in Sec. II-C. As a result, failing to utilize the buffered energy at sub-normal voltages introduces significant power inefficiency. In Sec. III-B, we validate this aspect and propose methods to fully utilize the buffered energy.

E. Impact of Sub-normal Voltage Execution

The traditional model leads software designers to assume that the system is executed under a stable voltage. However, a significant portion of execution may happen after the power-off threshold at sub-normal voltages (**O3**), as discussed in Sec. II-D. Being aware of this is crucial to software designers since analog components and peripherals may function differently at sub-normal voltages. Two relevant examples are Analog-Digital Converters (ADCs) and external NVMs.

ADCs are commonly used to determine when to execute a checkpoint by reading V_{ES}. It quantizes the input analog voltage into discrete 2^n values, ranging from 0 to the given reference voltage, where n is a resolution. Using smaller reference voltage increases sensitivity of ADC at the cost of reduced representation range.

Fig. 7a shows the behavior of ADCs, where the execution in sub-normal voltages are depicted in gray. Because STM32L5 uses V_{dd} as a reference voltage, accessing the ADC under sub-normal voltages produces inconsistent results. As shown in the figure, the ADC returns values higher than the actual measurements since its representation range decreases as V_{dd} drops. Consequently, ADC may mislead the system into overestimating the energy in C_{ES} during sub-normal voltage executions, potentially leading to checkpoint miss and a loss of progress for the entire power cycle.

Also, intermittent systems typically designed to operate with peripherals such as sensors [5], [40]–[42], wireless communication modules [6], [43], [44] or external NVMs [6], [23]–[25], which have their own minimum operating voltage requirements. Fig. 7b illustrates the error rate of FRAM in the reference system at different voltages, showing FRAM cannot operate reliably below 2.4V. Since the system continues operating until it reaches the lowest MCU operation voltage (e.g., 1.7V), software designers must ensure that peripherals are accessed only at safe voltage levels. Failing to do so can result in corrupted sensor data or unsafe checkpointing.

TABLE I: Architectures for generality evaluation

	Core	Core Freq.	Capacitance (uF)			Current	Memory
			C1	C2	C_{ES}		
A1	STM32L5	16MHz	22	220	1,320	3mA	MRAM (off-chip)
A2	MSP430FR5994	8MHz	22	10	40	100uA	FRAM (on-chip)

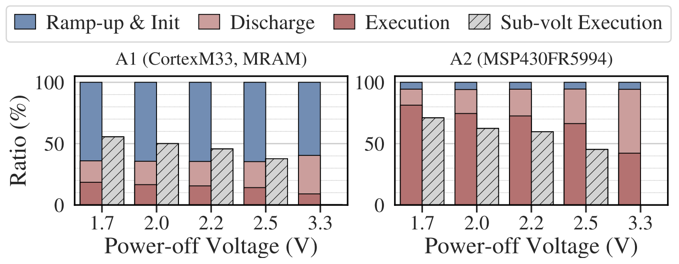


Fig. 8: Energy breakdown and the ratio of sub-voltage operations in different architectures.

F. Sensitivity to Architectural Designs

To evaluate the generality of the proposed model, we assess it across two additional architectural setups. Table I shows the detailed parameters of the target architectures. A1 shares the same configuration as the reference system but equips MRAM (Everspin MR5A16ACYS35), which is gaining attention as a next generation NVM [6], [14], [17], [23], instead of FRAM. Second target is MSP430 equipped with on-chip FRAM, a widely adopted 16-bit platform in intermittent system research. For both systems, the architectural parameters are configured to achieve an operation time of approximately 50 ms.

Fig. 8 shows the results for different power-off voltages. The bars on the left illustrate the energy breakdown in a single power cycle, and the bars on the right represent the ratio of the sub-normal voltage executions. The most noticeable difference is ratio of energy consumed during the *Ramp-up & Init* stage. While A1 consumes 63.4% power at this stage on average, only 5.6% of energy is consumed in A2. This is because A1 is configured with an external MRAM, which exhibits significantly higher leakage current, even compared to the FRAM used in the reference system. In contrast, A2 is equipped with on-chip FRAM, which has much lower leakage.

Despite these differences, both architectures exhibit high sub-normal voltage execution rates, up to 55.5% in A1 and 70.1% in A2. In addition, discharged energy takes considerable portion in both A1 (31.4%) and A2 (52.0%) at 3.3V power-off voltage configuration, which represents the techniques based on the traditional model that halt immediately at V_{ES}. In summary, the evaluation demonstrates that the modeled

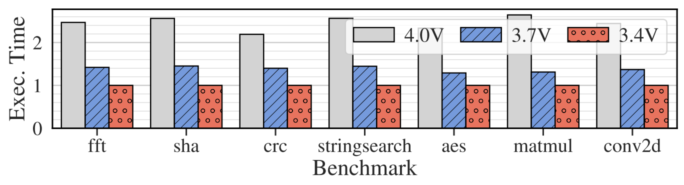


Fig. 9: Execution times across various checkpoint voltages, normalized to the 3.4V configuration.

buffering effects are general and their impacts are significant across different system architectures.

III. Design Guidelines

Based on the insights from our model, we propose design guidelines to implement efficient intermittent systems. The effectiveness of these guidelines is evaluated using seven benchmarks on the reference system used in Sec. II. We ported five benchmarks from miBench [45] benchmark suite and implemented two computation kernels (*matmul* and *conv2d*) commonly used in the evaluation of intermittent systems in the literature [11], [23], [24], [46], [47].

We evaluate two popular existing checkpointing schemes: *static* and *dynamic*. In *static*, checkpoint triggers are inserted at the back-edge of every loop in the program during compilation [3], [24], [25], [48]. At runtime, checkpoint triggers examine V_{ES} and execute checkpoint only when it is below a predefined threshold. In contrast, *dynamic* [4], [11], [20], [38], [39] does not modify the original program code. Instead, it executes checkpoints via interrupts from the power management system, generated when V_{ES} reaches V_l. These schemes are considered since most checkpoint techniques utilize V_{ES} either by actively polling it (as in *static*) or by receiving external signals generated based on V_{ES} levels (as in *dynamic*). All the evaluations are conducted with 470uF energy storage and 1mA of input current at 1.9V, unless otherwise stated.

A. *Delaying Checkpoint Executions*

The first design practice we propose is to delay checkpoint executions until the last possible moment. While this practice is generally regarded as desirable in existing works [3], [49], it has not been recognized as a critical property. Under the traditional execution model, early checkpoint execution is often considered acceptable as it makes the system wake up sooner, incurring only minor costs for initialization and recovery. For example, some approaches have explored proactive power-offs based on the program's worst-case execution time [1], [16], [50]. In contrast, our model reveals that significant energy is wasted each time the system powers off (Sec. II-C).

We evaluate the impact of delaying checkpoint executions in *dynamic*, by varying the interrupt voltage. A 1100uF capacitor is used for C_{ES}. Fig. 9 presents the average execution times of the benchmarks over 30 runs, normalized to the 3.4V configuration. The results show that executing checkpoints earlier is significantly inefficient as opposed to existing expectations: by 1.38x in 3.7V, and 2.45x in 4.0V setups, on average. Moreover, the overhead is consistent across all benchmarks since early checkpoint executions directly reduce the energy available for the computing system. Consequently, for maximum power efficiency, checkpoint techniques should be able to minimize the margin between the checkpoint execution and the power-off. Achieving this fundamentally depends on accurately predicting imminent power failures, which is the focus of the next section.

B. *Using V_{dd} with a Reference Voltage for Checkpoint Signals*

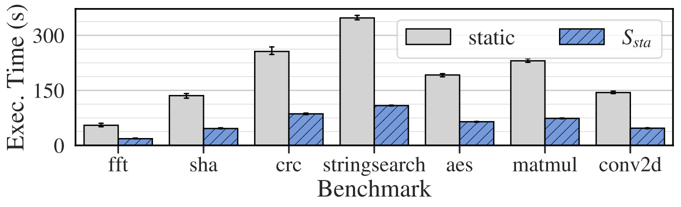
Sec. II-D demonstrates that V_{ES} is not a reliable estimate for the system's remaining execution time and that low V_{dd} is the direct cause of power-off. Based on this insight, we propose using V_{dd} to more accurately detect the imminent power failures, as in works without power management system (Sec. IV). We present two efficient implementations, S_{sta} and S_{dyn}, to accurately detect the imminent power-off events in approaches similar to *static* and *dynamic*, respectively.

Meanwhile, when designing techniques using V_{dd}, designers should account for the behaviors of analog components at subnormal voltages (Sec. II-E). For consistent operation of ADCs, we adopt a voltage source with a known value of V_{ref}. In STM32L5 and MSP430, an internal reference voltage source of 1.2V is available; alternatively, an external voltage reference (e.g., TI LVM431 [51]) can be used. Note that V_{ref} should be lower than the minimal operating voltage of MCU (e.g., 1.7V) as V_{ref} is generated by regulating V_{dd}.

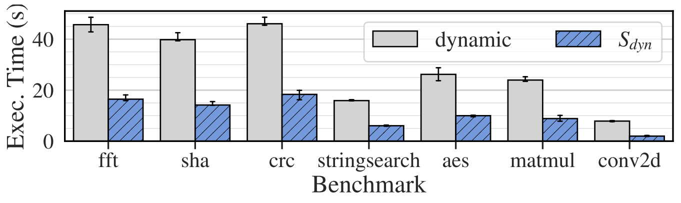
S_{sta} is designed for techniques similar to *static*, which query whether to execute a checkpoint at checkpoint triggers. Since directly reading V_{dd} is infeasible (i.e., V_{dd} itself is a reference voltage), S_{sta} reads V_{ref} instead. This results in the same value of $\lfloor V_{ref}/V_{dd} \cdot 2^n \rfloor$ when operating on normal voltage, where n is the ADC resolution. On the other hand, during sub-normal voltage executions, this value increases as V_{dd} decreases, as discussed in Sec. II-E. As a result, given that the target threshold voltage for checkpoint execution is V_{th}, software designers can compare the ADC value against $\lfloor V_{ref}/V_{th} \cdot 2^n \rfloor$ to determine whether to execute a checkpoint.

On the other hand, S_{dyn} utilizes an on-chip comparator, which is available in most modern MCUs including STM32L5 and MSP430. As V_{ref} is always lower than V_{dd}, we use a voltage divider consisting of two resistors, $R1$ and $R2$, to scale V_{dd} and compare it with V_{ref}. Specifically, we configure $R1$ and $R2$ to satisfy $\frac{R2}{R1+R2} \cdot V_{th} = V_{ref}$, so the comparator generates an interrupt when V_{dd} reaches the threshold V_{th}.

Fig. 10 compares the average execution times of the benchmarks over 30 iterations between traditional systems and the proposed setups. Fig. 10a and Fig. 10b illustrates the results of S_{sta} and S_{dyn}, respectively. The whiskers indicate the minimum and maximum execution times for each benchmark. The results show significant improvements in execution times for both systems, with average gain of 3.04x in S_{sta} and 2.85x in S_{dyn}. While the effectiveness of checkpoint schemes varies depending on application characteristics, our setups evenly enhance performance across all benchmarks. This underscores the importance of accurately detecting power-off events for efficient intermittent system operation.



(a) Static checkpointing with S_{sta}.



(b) Dynamic checkpointing with S_{dyn}.

Fig. 10: Impact of precise checkpoint timings to the end-to-end execution times.

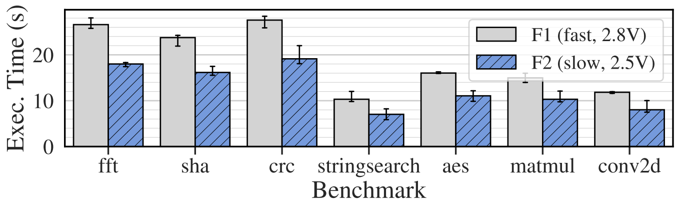


Fig. 11: Impact of peripheral operating voltage.

Another advantage of the proposed setups is their simplicity and practical applicability. Since the both setups only modify the method to detect imminent power failures and leave the checkpoint algorithms unchanged, it is straightforward to apply them in existing techniques. Furthermore, the proposed setups can reduce the system complexity, as they eliminate the need for communication between the energy storage system and the computing system (e.g., interrupt or access to V_{ES}).

C. On Selecting Hardware Components

Our model also helps designers in selecting efficient hardware components across various parameters. For example, it reveals that operating voltage of peripherals (e.g., external NVMs) is a critical design consideration (Sec. II-E), often more important than other factors such as latency.

To evaluate this tradeoff, we simulate two FRAM configurations, F1 and F2, in our reference system. F1 represents a slower setup capable of operating down to 2.5V. This is achieved by doubling the software-configurable wait time for FRAM accesses. F2 is set to have the lowest access latency but requires the system stop operating at 2.8V.

Fig. 11 presents the execution times of the benchmarks for the two configurations in S_{dyn}, averaged over 30 runs. Despite its doubled latency, F1 completes the workloads 1.46x faster on average, with consistent improvements across all benchmarks. These results imply that using slower FRAM that operates until 1.8V (e.g., [52]) could considerably improve the performance of our reference system. This example highlights that operating voltage, often overlooked in the traditional model, should be considered a critical design parameter.

Finally, our model highlights advantages of using smaller decoupling capacitors. Larger buffers not only increases the ratio of sub-normal voltage operations but also raise the amount of discharged energy during power-offs. Indeed, in our reference system with C_{ES} = 1100uF, we observe that completing benchmarks takes 1.18x and 1.36x longer on average, when 440uF and 660uF capacitors are used as C2, respectively, compared to our setup with a 220uF capacitor.

IV. Related Work

This work can be compared to existing modeling-based approaches that estimate the timing or efficiency of intermittent systems [7]–[10], [22], [53]. The primary focus of these works is identifying the most efficient design configurations (e.g., capacitor size, input power or checkpoint techniques [22]) for a given application. Zhan et al. [54] especially examined the trade-offs between capacitor sizes and forward progress. However, these works assume that the entire energy discharged from the capacitor is utilized by computing system, overlooking the buffering effects addressed in this work. Furthermore, our work proposes several practical guidelines to improve the efficiency of existing techniques with minimal efforts.

In some works that do not have a dedicated power management system and directly supply unregulated power to the computing system [13], [38], [39], [55], [56], V_{dd} has been used as a checkpoint signal. This is natural in these works since the voltage of the energy storage is always identical to V_{dd}. In contrast, our work demonstrates that accounting for sub-normal voltage operations is also critical in systems with regulated power supplies, which represent the majority of intermittent system setups. Also, we address the impacts of sub-normal voltage executions on the correctness and efficiency of software designs, along with suggestions to exploit such impacts for improved system performance.

V. Conclusion

As recent intermittent systems target smaller energy storages and shorter operation times, the traditional execution model for intermittent systems is failing to accurately represent actual system behaviors. In this paper, we propose a new execution model that incorporates the buffering effects from the system's inherent capacitance, which is a primary source of the discrepancies of the traditional model. Our model reveals that systems designed upon the traditional model can be up to 5.62x power-inefficient than expected and may result in unsafe checkpoint executions. Based on the insights from our model, we propose several design guidelines, including setups to improve performance of existing static and dynamic checkpoint techniques by 3.04x and 2.85x on average, respectively.

References

[1] J. Choi *et al.*, "Compiler-Directed High-Performance Intermittent Computation with Power Failure Immunity," RTAS '22, May 2022.

[2] S. Ahmed *et al.*, "The Internet of Batteryless Things," *Communications of the ACM*, vol. 67, pp. 64–73, Mar. 2024.

[3] B. Ransford *et al.*, “Mementos: System support for long-running computation on RFID-scale devices,” *ACM SIGARCH Computer Architecture News*, vol. 39, pp. 159–170, Mar. 2011.

[4] H. Jayakumar *et al.*, “QUICKRECALL: A Low Overhead HW/SW Approach for Enabling Computations across Power Cycles in Transiently Powered Computers,” VLSID ’14’, pp. 330–335, Jan. 2014.

[5] K. Maeng and B. Lucia, “Adaptive low-overhead scheduling for periodic and reactive intermittent execution,” PLDI ’20’, ACM, June 2020.

[6] J. de Winkel *et al.*, “Intermittently-powered bluetooth that works,” MobiSys ’22, pp. 287–301, ACM, June 2022.

[7] X. Hou *et al.*, “A Tale of Two Domains: Exploring Efficient Architecture Design for Truly Autonomous Things,” ISCA ’24, June 2024.

[8] F. Erata *et al.*, “ETAP: Energy-aware Timing Analysis of Intermittent Programs,” *ACM Transactions on Embedded Computing Systems*, vol. 22, pp. 23:1–23:31, Jan. 2023.

[9] F. Ghasemi *et al.*, “PES: An Energy and Throughput Model for Energy Harvesting IoT Systems,” ISPASS ’23, pp. 13–23, Apr. 2023.

[10] J. San Miguel *et al.*, “The EH Model: Early Design Space Exploration of Intermittent Processor Architectures,” MICRO ’18, Oct. 2018.

[11] K. Maeng and B. Lucia, “Supporting peripherals in intermittent systems with just-in-time checkpoints,” PLDI ’19, ACM, June 2019.

[12] V. Kortbeek *et al.*, “BFree: Enabling Battery-free Sensor Prototyping with Python,” *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 4, pp. 135:1–135:39, Dec. 2020.

[13] A. J. Neto *et al.*, “DiCA: A Hardware-Software Co-Design for Differential Check-Pointing in Intermittently Powered Devices,” ICCAD ’23, pp. 1–9, Oct. 2023.

[14] A. Bakar *et al.*, “Protean: An Energy-Efficient and Heterogeneous Platform for Adaptive and Hardware-Accelerated Battery-Free Computing,” SenSys ’23, pp. 207–221, ACM, Jan. 2023.

[15] A. Alsubhi *et al.*, “Stash: Flexible Energy Storage for Intermittent Sensors,” *ACM Trans. Embed. Comput. Syst.*, vol. 23, pp. 18:1–18:23, Mar. 2024.

[16] H. Reymond *et al.*, “SCHEMATIC: Compile-Time Checkpoint Placement and Memory Allocation for Intermittent Systems,” CGO ’24, pp. 258–269, Mar. 2024.

[17] Y. Wu *et al.*, “IntOS: Persistent Embedded Operating System and Language Support for Multi-threaded Intermittent Computing,” OSDI ’24, pp. 425–443, 2024.

[18] E. Yildiz *et al.*, “Efficient and Safe I/O Operations for Intermittent Systems,” in *Proceedings of the Eighteenth European Conference on Computer Systems*, pp. 63–78, ACM, May 2023.

[19] V. Kortbeek *et al.*, “WARio: Efficient Code Generation for Intermittent Computing,” p. 15, 2022.

[20] V. Kortbeek *et al.*, “Time-sensitive Intermittent Computing Meets Legacy Software,” ASPLOS ’20, pp. 85–99, ACM, Mar. 2020.

[21] H.-X. Shih *et al.*, “Intermittent Edge Computing for Green Agricultural Automation,” IoTDI ’24, pp. 219–220, May 2024.

[22] Y. Kim and H. Kim, “Rapid Hardware/Software Design Space Exploration for Efficient Intermittent Systems,” ISLPED ’24, pp. 1–6, ACM, Sept. 2024.

[23] K. Akhunov *et al.*, “Enabling Efficient Intermittent Computing on Brand New Microcontrollers via Tracking Programmable Voltage Thresholds,” ENSsys ’23, pp. 16–22, ACM, Nov. 2023.

[24] Y. Kim *et al.*, “LACT: Liveness-Aware Checkpointing to reduce checkpoint overheads in intermittent systems,” *Journal of Systems Architecture*, vol. 153, p. 103213, Aug. 2024.

[25] Y. Kim *et al.*, “Liveness-Aware Checkpointing of Arrays for Efficient Intermittent Computing,” DATE ’23, pp. 1–6, Apr. 2023.

[26] G. Park *et al.*, “Energy-Harvesting-Aware Adaptive Inference of Deep Neural Networks in Embedded Systems,” ISLPED ’23, pp. 1–6, Aug. 2023.

[27] O. Khan *et al.*, “DaCapo: An On-Device Learning Scheme for Memory-Constrained Embedded Systems,” *ACM Trans. Embed. Comput. Syst.*, vol. 22, pp. 142:1–142:23, Sept. 2023.

[28] R. Barjami *et al.*, “Intermittent Inference: Trading a 1% Accuracy Loss for a 1.9x Throughput Speedup,” SenSys ’24, pp. 647–660, ACM, Nov. 2024.

[29] W. Song *et al.*, “TaDA: Task Decoupling Architecture for the Battery-less Internet of Things,” SenSys ’24, pp. 409–421, ACM, Nov. 2024.

[30] C.-H. Yen *et al.*, “Keep in Balance: Runtime-reconfigurable Intermittent Deep Inference,” *ACM Transactions on Embedded Computing Systems*, vol. 22, pp. 124:1–124:25, Sept. 2023.

[31] G. Gobieski *et al.*, “Intelligence Beyond the Edge: Inference on Intermittent Embedded Systems,” ASPLOS ’19, pp. 199–213, ACM, Apr. 2019.

[32] S. Islam *et al.*, “Enabling Fast Deep Learning on Tiny Energy-Harvesting IoT Devices,” DATE ’22, pp. 921–926, Mar. 2022.

[33] C.-K. Kang *et al.*, “More Is Less: Model Augmentation for Intermittent Deep Inference,” *ACM Trans. Embed. Comput. Syst.*, vol. 21, pp. 49:1–49:26, Oct. 2022.

[34] S. Lee and S. Nirjon, “Neuro.ZERO: A zero-energy neural network accelerator for embedded sensing and inference systems,” SenSys ’19, pp. 138–152, ACM, Nov. 2019.

[35] B. Islam and S. Nirjon, “Zygarde: Time-Sensitive On-Device Deep Inference and Adaptation on Intermittently-Powered Systems,” vol. 4, pp. 82:1–82:29, Sept. 2020.

[36] L. L. Custode *et al.*, “Fast-Inf: Ultra-Fast Embedded Intelligence on the Batteryless Edge,” SenSys ’24, pp. 239–252, ACM, Nov. 2024.

[37] L. Caronti *et al.*, “Fine-grained Hardware Acceleration for Efficient Batteryless Intermittent Inference on the Edge,” *ACM Transactions on Embedded Computing Systems*, vol. 22, pp. 82:1–82:19, Sept. 2023.

[38] D. Balsamo *et al.*, “Hibernus++: A Self-Calibrating and Adaptive System for Transiently-Powered Embedded Devices,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 12, pp. 1968–1980, 2016.

[39] D. Balsamo *et al.*, “Hibernus: Sustaining Computation During Intermittent Supply for Energy-Harvesting Systems,” *IEEE Embedded Systems Letters*, vol. 7, pp. 15–18, Mar. 2015.

[40] E. Yildiz *et al.*, “Adaptable Runtime Monitoring for Intermittent Systems,” in *Proceedings of the Nineteenth European Conference on Computer Systems*, pp. 1175–1191, ACM, Apr. 2024.

[41] T. Dang *et al.*, “ioTree: A battery-free wearable system with biocompatible sensors for continuous tree health monitoring,” MobiCom ’22, pp. 352–366, ACM, Oct. 2022.

[42] M. Afanasov *et al.*, “Battery-less zero-maintenance embedded sensing at the mithræum of circus maximus,” SenSys ’20, pp. 368–381, ACM, Nov. 2020.

[43] M. Katanbaf *et al.*, “MultiScatter: Multistatic Backscatter Networking for Battery-Free Sensors,” SenSys ’21, pp. 69–83, ACM, Nov. 2021.

[44] S. Babatunde *et al.*, “Greentooth: Robust and Energy Efficient Wireless Networking for Batteryless Devices,” *ACM Transactions on Sensor Networks*, p. 3649221, Mar. 2024.

[45] M. R. Guthaus *et al.*, “MiBench: A free, commercially representative embedded benchmark suite,” WWC ’01, pp. 3–14, Dec. 2001.

[46] A. Bhattacharyya *et al.*, “NvMR: Non-volatile memory renaming for intermittent computing,” ISCA ’22, pp. 1–13, ACM, June 2022.

[47] K. Ganesan *et al.*, “The What’s Next Intermittent Computing Architecture,” HPCA ’19, pp. 211–223, Feb. 2019.

[48] K. Maeng and B. Lucia, “Adaptive Dynamic Checkpointing for Safe Efficient Intermittent Computing,” OSDI ’18, pp. 129–144, 2018.

[49] N. A. Bhatti and L. Mottola, “HarvOS: Efficient Code Instrumentation for Transiently-Powered Embedded Sensing,” IPSN ’17, pp. 209–220, Apr. 2017.

[50] P. Raffeck *et al.*, “WoCA: Avoiding Intermittent Execution in Embedded Systems by Worst-Case Analyses with Device States,” LCTES ’24, pp. 83–94, ACM, June 2024.

[51] Texas Instruments, “LMV431, 1.5%, low-voltage (1.24-V) adjustable precision shunt regulator.” https://www.ti.com/product/en-us/LMV431.

[52] Fujitsu Semiconductor, *MB85R4M2T*.

[53] J. San Miguel *et al.*, “The EH Model: Analytical Exploration of Energy-Harvesting Architectures,” *IEEE Computer Architecture Letters*, vol. 17, pp. 76–79, Jan. 2018.

[54] J. Zhan *et al.*, “Exploring the Effect of Energy Storage Sizing on Intermittent Computing System Performance,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, pp. 492–501, Mar. 2022.

[55] P. Raffeck *et al.*, “CO2CoDe: Towards Carbon-Aware Hardware/Software Co-Design for Intermittently-Powered Embedded Systems,” 2024.

[56] H. Reymond *et al.*, “EarlyBird: Energy belongs to those who wake up early,” RTCSA ’24, 2024.