The EH Model: Early Design Space Exploration of Intermittent Processor Architectures

Joshua San Miguel[†], Karthik Ganesan[‡], Mario Badr[‡], Chunqiu Xia[‡], Rose Li[‡], Hsuan Hsiao[‡]
and Natalie Enright Jerger[‡]
[†]*University of Wisconsin-Madison,* [‡]*University of Toronto*
jsanmiguel@wisc.edu, {karthik.ganesan, mario.badr, steven.xia, rose.li, julie.hsiao}@mail.utoronto.ca, enright@ece.utoronto.ca

***Abstract*—Energy-harvesting devices—which operate solely on energy collected from their environment—have brought forth a new paradigm of intermittent computing. These devices succumb to frequent power outages that would cause conventional systems to be stuck in a perpetual loop of restarting computation and never making progress. Ensuring forward progress in an intermittent execution model requires saving state in nonvolatile memory (backup) and potentially re-executing from the last saved state upon a power loss (restore). The interplay between spending energy on useful processing and spending energy on these necessary overheads yield unexpected trade-offs. To facilitate early design space exploration, the field of intermittent computing requires better models for 1) generalizing and reasoning about these trade-offs and 2) helping architects and programmers in making early-stage design decisions.**

We propose the *EH model*, which characterizes an intermittent system's ability to maximize how much of its available energy is spent on useful processor execution. The model parametrizes the energy costs associated with intermittent execution to allow an intuitive understanding of how forward progress can change. We use the EH model to explore how forward progress is impacted with the frequency of backups and the energy cost of backups and restores. We validate the EH model with hardware measurements on an MSP430 and characterize its parameters via simulation. We also demonstrate how architects and programmers can use the model to explore the design space of intermittent processors, derive insights, and model new optimizations that are unique to intermittent processor architectures.

I. Introduction

The batteryless operation of compute devices introduces new and challenging trade-offs to programmers and computer architects. Ambient energy sources (e.g., photovoltaic [23], thermal [47], RF [52], WiFi [8]) do not provide a constant stream of power to run the device [11]. Also, they typically provide less average power than the device requires. To overcome this, a common approach is to first store energy in a capacitor, which then powers the device for a period of time [11]. This sporadic power supply from ambient energy sources requires an execution model that must inherently support *intermittent computation*: computation may stop at any point in the application because the energy supply has depleted and cannot resume until the device has harvested sufficient energy from the environment.

Intermittent computation requires that application state be *backed up* in nonvolatile memory before energy is depleted and *restored* when energy is available again. A good intermittent architecture maximizes how much of the available energy is spent on useful work (i.e., *forward progress*), not on the necessary overheads of backups and restores. But building such an architecture is no simple task. Researchers have proposed a wide variety of processor designs ranging from simple in-order cores [6] to complex out-of-order cores [38] with a diverse mix of nonvolatile memory technologies (e.g., flash [43] and FRAM [12]). The variety of approaches being proposed demonstrates the vast design space of intermittent processor architectures. Should the architecture perform a backup every cycle [38], at set periods [43], only at critical power levels [6], when explicitly told by the program [12], or when implicitly inferred in hardware [22]? Should we save only the program counter and register file, or should we also save volatile memory data [6], cache data [31], or only dirty values [38], [56]? Should programmers manually write code that backs up to nonvolatile memory, or should they use task-based programming models [12], [34], checkpointing schemes [43], or simply rely on hardware to perform backups under the hood [22], [38]? Though simulators and tools exist for specific systems, we are in need of a way to rapidly explore the design space of intermittent processors to inform future architectures.

To address this need, we present and extend the *EH model*[1] [44], an analytical model for early design space exploration that generalizes and characterizes the complex, unconventional trade-offs that arise in intermittent processors. We validate the model and demonstrate its utility as an early-stage design tool with several explorations and case studies:

- Given the expected behaviour of applications, where should architects focus their efforts to maximize forward progress (e.g., backup vs. restore mechanisms)?
- Can a programmer estimate how well their application will perform under a specific architectural configuration (e.g., optimal task lengths for systems like Chain [12])?
- Under what circumstances are architectural optimizations (e.g., reduced bit-precision) beneficial to intermittent systems?
- How can programmers transform their code to perform better on state-of-the-art architectures (e.g., memory locality for mixed-volatility caches [31], [56])?

Our EH model helps architects and programmers derive new insights and rapidly explore this design space. We hope our model excites further research in the field of energy harvesting

[1]Why EH? Well, we are Canadian, after all.

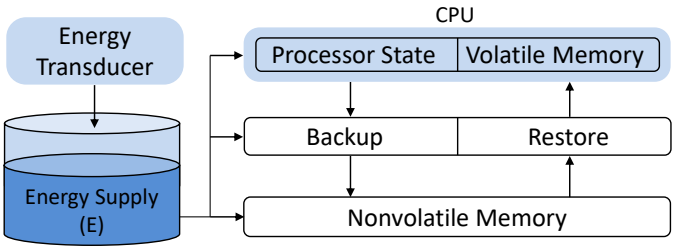

Fig. 1. An abstract energy-harvesting device.

by alleviating the complexity of designing efficient intermittent systems.

II. Background

Figure 1 represents an abstract energy-harvesting device. A transducer harvests energy from the environment to charge a capacitor. The energy is used by different hardware components: the CPU, nonvolatile memory, and the backup/restore mechanism. Typically, the system waits until the capacitor reaches a minimum threshold before it begins executing [6], [43]. Thus, no forward progress can be made during the *charging* phase. Once the minimum energy threshold is reached, an *active* phase begins until the capacitor is depleted. Transitioning between active and charging phases gives rise to intermittent computation, which is a function of how much energy can be harvested from the environment and how much energy is being used by the architecture.

The main challenge in intermittent computation is this: forward progress can only be made when the results of executed instructions are saved to nonvolatile memory. Intermediate results in volatile memory are lost during a power outage, and expending energy on instructions whose output is not saved before a power outage is wasteful. In this section, we provide an overview of prior art designed to allow intermittent computing systems to make forward progress in the face of frequent power outages. We divide the techniques into two categories: *multi-backup* and *single-backup* systems.

Multi-Backup Systems. A multi-backup system typically performs several backups within a single active period. Mementos [43] inserts checkpoints into the code (at the end of loop iterations or function calls) to back up volatile memory elements (i.e., SRAM) to nonvolatile memory (i.e., Flash). DINO [34], Chain [12], Alpaca [39] and Mayfly [21] break the program into atomic tasks, with backups being performed at the boundary between tasks, to maintain consistency while ensuring forward progress. Ratchet [54] and Clank [22] keep track of data that has been modified since the last power outage and periodically back up just this data to nonvolatile memory.

Single-Backup Systems. A single-backup system saves system state only once per active period. For example, Hibernus [5], [6] and QuickRecall [25] use an analog to digital converter (ADC) to sense when the voltage dips below a pre-defined threshold to back up. Several works build upon this idea to increase the efficiency of intermittent computing systems [1], [4], [7], [32], [35].

Nonvolatile Processors. The multi- and single-backup systems listed above augment conventional processors, which incorporate volatile memory such as SRAM, to work in the face of power outages. An alternative approach uses nonvolatile memory such as FRAM [28], [30], [36], [38], [53], [55], [58], ReRam [33] or MRAM [45]. A nonvolatile processor (NVP) does not require explicit checkpointing because all system state is preserved across power outages. Upon recovering from a power outage, the system resumes processing immediately without needing to restore state. However, nonvolatile memory suffers from higher power consumption. Thus, NVPs vary from backing up every cycle to backing up only when the supply voltage drops below a set threshold. Our EH model captures both conventional and NVP architectures; NVP designs that back up every cycle are characterized as multi-backup while those that back up based on a threshold are characterized as single backup.

As we demonstrate in the following sections, our EH model supports both multi-backup (Section IV-A) and single-backup architectures (Section IV-B) while still remaining general enough to characterize new designs (Section VI). In the next section, we provide a detailed description of the EH model.

III. The EH Model

The goal of our model is to enable early design space exploration of intermittent processor architectures by estimating forward progress. Prior work models the behaviour of energy-harvesting power sources, such as RF [51], vibration [27], [46], photovoltaic [14] and piezoelectric [29] power. In contrast, we focus on modeling a processor that is subject to intermittent execution caused by harvesting energy. The EH model focuses on the *active period*, during which the energy supply is expended on executing instructions and performing backups and restores. We differentiate between energy spent on forward progress, backups, restores, and re-execution (i.e., *dead energy*) caused when state was not saved before a power outage.

We list our model parameters in Table I. The output of our model is p, an estimate of forward progress represented as the fraction of the energy supply E expended on useful work (i.e., not spent on backups, restores and dead execution). To begin, we characterize the fundamental consumers of energy E:

$$E - \left(e_P + (n_B \cdot e_B) + e_D + e_R\right) = 0 \tag{1}$$

- e_P is energy spent on forward progress.
- e_B is energy spent on each of n_B backups.
- e_D is energy spent on dead execution.
- e_R is energy spent on restoring backed-up state.

Often the energy budget is not fixed at E but rather increases over time since the device can continue charging during an active period. We model this additional energy by including the device charging rate (ε_C) in the following equations for each of the energy consumers.[2]

[2]The charging rate can be modelled as a separate component of Equation 1; we opt to incorporate it into the individual equations to simplify the discussion without loss of fidelity.

TABLE I
EH MODEL INPUT PARAMETERS AND OUTPUTS.

General Parameters		
$E \in \mathbb{R}_{>0}$	joules	energy supply per active period
$\varepsilon \in \mathbb{R}_{>0}$	joules/cycle	execution energy per cycle
$\varepsilon_C \in \mathbb{R}_{\geq 0}$	joules/cycle	charging energy per cycle
Backup Parameters		
$\tau_B \in \mathbb{R}_{>0}$	cycles	time between backups
$\sigma_B \in \mathbb{R}_{>0}$	bytes/cycle	memory backup bandwidth
$\Omega_B \in \mathbb{R}_{\geq 0}$	joules/byte	backup energy cost
$A_B \in \mathbb{R}_{\geq 0}$	bytes	architectural state per backup
$\alpha_B \in \mathbb{R}_{\geq 0}$	bytes/cycle	application state per backup
Restore Parameters		
$\sigma_R \in \mathbb{R}_{>0}$	bytes/cycle	memory restore bandwidth
$\Omega_R \in \mathbb{R}_{\geq 0}$	joules/byte	restore energy cost
$A_R \in \mathbb{R}_{\geq 0}$	bytes	architectural state per restore
$\alpha_R \in \mathbb{R}_{\geq 0}$	bytes/cycle	application state per restore
Model Output		
$\tau_P \in \mathbb{R}_{\geq 0}$	cycles	time spent on forward progress
$p = \varepsilon \cdot \tau_P / E$	% of E	% energy spent on forward progress

Energy for Forward Progress (e_P). An application expends some amount of energy per cycle (ε) while executing. During an active period, a certain number of cycles will be used to make forward progress (τ_P); the total energy spent on forward progress is computed via Equation 2, along with any additional charging energy per cycle. The execution energy cost ε includes not only the processor but also any sensors and peripherals that are active. Techniques that reduce ε (e.g., duty cycling sensors, dynamic voltage scaling [37]) are always beneficial for forward progress.

$$e_P = (\varepsilon - \varepsilon_C) \cdot \tau_P \tag{2}$$

Energy for Backups (e_B). During active periods, computation must be backed up to nonvolatile memory to make forward progress. Some approaches may back up multiple times within an active period [6], [22], [38]; others only once [43]. We characterize how often backups occur with the number of cycles between them (τ_B). The total number of backups is:

$$n_B = \frac{\tau_P}{\tau_B} \tag{3}$$

To compute the energy spent on each backup (e_B), we multiply the cost of writing to nonvolatile memory (Ω_B) by the number of bytes written per backup. A system may back up a fixed number of bytes each time (A_B), such as architectural state (e.g., program counter, registers) [38]. A system may also incur a variable backup cost that is proportional to any changes in application state since the last backup (α_B bytes/cycle). For example, dirty data in a volatile cache [31], [56] must be saved, the amount of which depends on the application's write footprint (we explore this further in Section VI-A). Charging during the time spent performing the backup gains additional energy for the system; this energy is inversely proportional to the nonvolatile memory backup bandwidth σ_B (e.g., for the MSP430 CPU this corresponds to 2 cycles per word at 16MHz or above, and 1 cycle per word for speeds below 16MHz [6], [12], [43]). We calculate energy expended on backups as:

$$e_B = (\Omega_B - \frac{\varepsilon_C}{\sigma_B}) \cdot (A_B + \alpha_B \cdot \tau_B) \tag{4}$$

Dead Energy (e_D). Ideally, we would back up our data just before our energy supply (E) is depleted. If this is not the case, then the application has expended energy on computations that are never stored in nonvolatile memory. Thus, some number of cycles before the next backup (τ_D) contribute to dead energy (e_D). We calculate the amount of dead energy expended in a similar way to the energy spent on forward progress (Equation 2). In this case, we replace τ_P with τ_D since application state after dead computation is not saved for the next restore:

$$e_D = (\varepsilon - \varepsilon_C) \cdot \tau_D \tag{5}$$

The number of dead cycles can vary due to non-deterministic effects in the energy supply and in the application behaviour. To simplify the model, we consider the average case of dead cycles: $\tau_D = \frac{\tau_B}{2}$, shown in Equation 6; we discuss the impact of variability in Section IV-A2.

$$0 \leq \tau_D \leq \tau_B, \quad \tau_D = \frac{\tau_B}{2}, \text{ on average} \tag{6}$$

Energy for Restores (e_R). Before an application can resume execution, it must expend energy (e_R) to restore its last saved state. Mirroring the backup overhead (Equation 4), this incurs the cost of accessing nonvolatile memory (Ω_R) scaled by the amount of architectural (A_R) and application (α_R) state that must be restored. A_R represents a fixed number of bytes that the processor must restore at the start of each active period (e.g., register file). α_R represents the variable cost of reverting or cleaning up any uncommitted state left over from the dead execution (τ_D) of the previous active period (e.g., flushing dead instructions in nonvolatile processors [38]). Charging gains additional energy during the restore process, whose duration is inversely proportional to the nonvolatile memory restore bandwidth σ_R. From this, we compute the restore energy as:

$$e_R = (\Omega_R - \frac{\varepsilon_C}{\sigma_R}) \cdot (A_R + \alpha_R \cdot \tau_D) \tag{7}$$

Putting It All Together. Our model outputs the percentage of energy spent on forward progress p as $\varepsilon \cdot \tau_P / E$. Solving for p in Equation 1 yields:

$$p = \frac{1 - \frac{e_D}{E} - \frac{e_R}{E}}{(1 + \frac{e_B}{(\varepsilon - \varepsilon_C) \cdot \tau_B}) \cdot (1 - \frac{\varepsilon_C}{\varepsilon})} \tag{8}$$

In the first term of the denominator, forward progress is scaled down by the ratio of backup overhead (e_B) to how much useful work each backup commits ($(\varepsilon - \varepsilon_C) \cdot \tau_B$). This

represents the backup cost-reward ratio that governs the rate at which an application moves forward. The second term represents additional progress made due to charging in the active period. Note that the charging rate is generally much lower than the consumption rate; progress p goes to ∞ when ε_C approaches ε. In the numerator, one-time costs—dead energy and restore energy—incurred once per active period, limit progress. Thus, even if the cost of backups were completely eliminated in the denominator, these one-time costs impose an upper bound on performance and must be minimized.

Architects and programmers can garner many interesting implications from Equation 8. The next section explores a few of them in detail and discusses the insights they reveal.

IV. Explorations

In the previous section, we demonstrate how to estimate forward progress based on the energy consumption of different components (Equation 1). This section focuses on how architects can use our model to design a system that maximizes forward progress. We use the two common paradigms introduced in Section II: *multi-backup* and *single-backup* systems. Recall that, a multi-backup system may invoke several backups in a single active period whereas single-backup systems invoke only one. Next, we detail the application of the EH model for both these approaches to intermittent computing devices.

A. Multi-Backup Systems

We generalize multi-backup architectures to those that invoke periodic backups once every τ_B cycles on average. τ_B is defined by either periodic system backups, program-induced backups, or both. Periodic system backups can take the form of:

- Hardware watchdog timers that force a backup after τ_B cycles (e.g., Clank [22]);
- Compiler-inserted checkpoints (e.g., Mementos [43]);
- Nonvolatile processors [38] that back up every cycle ($\tau_B = 1$).

Examples of program-induced backups are: when tasks commit in transaction-based systems like Chain [12] or Mayfly [21] (τ_B is the average task length in this case) or when memory operations violate idempotency in Clank [22] (idempotency is further explored in Sections V and VI-B).

We formulate our model in Section III such that it is general enough to characterize an arbitrary number of backups per active period. Thus modelling a multi-backup system is as simple as setting the appropriate time between backups (τ_B). We look at how to optimize the time between backups, minimize dead cycles, and balance the restore-versus-backup cost (Sections IV-A1, IV-A2, and IV-A3).

1) Optimal Time Between Backups: How many cycles apart should backups be (τ_B) to maximize forward progress in a multi-backup system? Figure 2 shows how progress (normalized to ε) varies with the time between backups (τ_B) and backup cost (Ω_B). We set the execution energy (ε) to 1% of the active period's energy supply (E) for illustrative purposes, focusing on general trends as opposed to the exact values. We also assume no restore overhead nor charging energy for brevity; it is straightforward to extend our analysis to include them. The

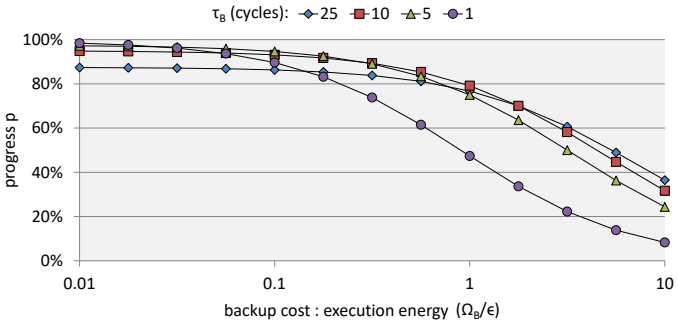

Fig. 2. Progress p for multi-backup system with varying τ_B and backup cost Ω_B (normalized to ε). Assumes $E = 100$, $e_C = 0$, $A_B = \varepsilon = 1$, $\alpha_B = 0.1$ and $\Omega_R = 0$.

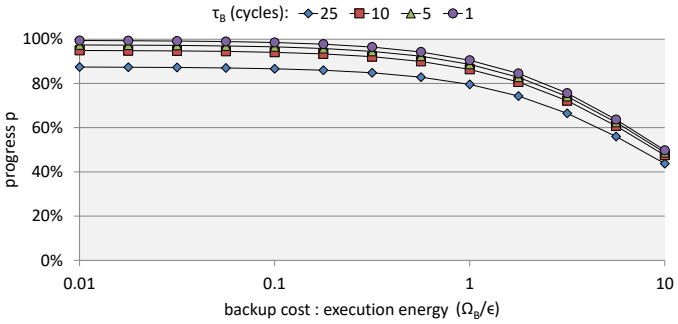

Fig. 3. Progress p for multi-backup system with varying τ_B and backup cost Ω_B (normalized to ε), with no architectural state ($A_B = 0$). Assumes $E = 100$, $e_C = 0$, $\varepsilon = 1$, $\alpha_B = 0.1$ and $\Omega_R = 0$.

first takeaway is that reducing backup cost is always better for performance, as expected. As the backup cost approaches 0, forward progress favours more frequent backups since 1) this minimizes dead cycles, and 2) backups are cheap.

The second takeaway is that the optimal time between backups is not stagnant but varies depending on the backup cost. Solving for the roots of $\frac{\partial p}{\partial \tau_B}$ (Equation 8), we obtain the optimal:

$$\tau_{B,opt} = \frac{\Omega_B \cdot A_B}{\Omega_B \cdot \alpha_B + \varepsilon} \cdot \left(\sqrt{2 \cdot \frac{E}{\varepsilon} \cdot \frac{\Omega_B \cdot \alpha_B + \varepsilon}{\Omega_B \cdot A_B} + 1} - 1 \right) \quad (9)$$

The ratio $\frac{\Omega_B \cdot A_B}{\Omega_B \cdot \alpha_B + \varepsilon}$ dictates the optimal number of cycles between backups. The numerator represents the compulsory energy cost per backup while the denominator represents the energy cost proportional to how much work was done since the last backup. There is a trade-off between 1) backing up less frequently if the compulsory cost is high and 2) backing up more frequently if the proportional cost is high. With Equation 9, programmers can estimate the optimal task length for their code (e.g., in systems like Chain [12], a task can be sized to match the optimal backup time), while system designers can configure the optimal period for checkpoints [43] and watchdog timers (e.g., in Clank [22], by choosing the appropriate duration between watchdog interrupts).

Note that in cases with very little architectural state (A_B approaches 0), there is no sweet spot in the time between backups, shown in Figure 3 (i.e., progress is monotonically non-increasing with τ_B). Though rare, needing to back up nearly

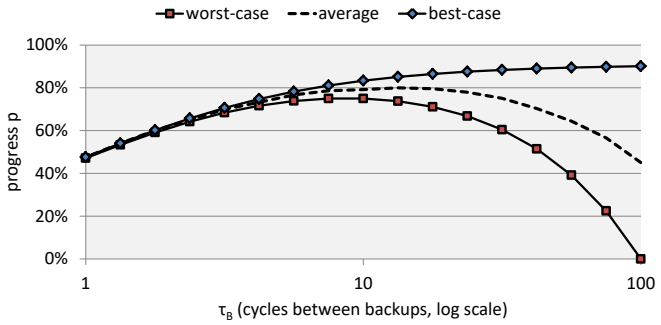

Fig. 4. Progress p over τ_B for multi-backup system, varying from worst-case to best-case τ_D. Assumes $E = 100$, $e_C = 0$, $\Omega_B = A_B = \varepsilon = 1$, $\alpha_B = 0.1$ and $\Omega_R = 0$.

zero architectural state is possible in systems that track dirty bits to save only the registers that have been modified (e.g., on-demand selective backups in nonvolatile processors [38]); only the program counter is compulsory on every backup. Assuming no cost of backing up architectural state ($A_B = 0$) in Equations 4 and 8 leaves us with a simple relationship: $\lim_{\tau_B \to 0} p = 1$. As a result, when considering only the cost of application state, it is always better to back up as frequently as possible since the overhead decreases proportionally with the time between backups (Equation 4).

2) Variability of Dead Cycles: So far our analysis for multi-backup systems assumes the average dead cycles (τ_D from Equation 6). Due to non-determinism in both the system (e.g., fluctuations in energy source) and the application (e.g., input-dependent program behaviour) [20], the number of dead cycles can vary dramatically across active periods. In this section, we explore how this variability affects important design decisions.

Figure 4 shows how progress varies under the worst-case ($\tau_D = \tau_B$) and best-case dead cycles ($\tau_D = 0$). We set similar system parameters as in Figure 2, assuming no restore or charging energy for simplicity. The first takeaway is that variability diminishes as the time between backups approaches 0; backing up more often decreases the likelihood of dead execution. Conversely, a long time between backups increases the risk of not backing up at all but opens up the possibility of the ideal scenario (i.e., a single backup invoked at the end of the active period). This introduces an interesting trade-off. If aggressive performance gains are desired, an architect can design a system with a long time between backups that are scheduled in an intelligent or speculative way to consistently minimize dead cycles. But if worst-case performance (i.e., tail latency) is important, an architect can sacrifice the average case and opt for a much lower τ_B. For example, Spendthrift [37] speculates the amount of dead energy per active period and employs voltage and frequency scaling to maximize efficiency. Our analysis provides an upper bound on forward progress for Spendthrift and related works that apply advanced speculative techniques.

Following from this, how many cycles apart should backups be invoked to maximize worst-case forward progress? At first glance, one may assume that Equation 9 is sufficient; however it assumes the average case of dead cycles. Solving instead for the worst case ($\tau_D = \tau_B$):

$$\tau_{B,opt(wc)} = \frac{\Omega_B \cdot A_B}{\Omega_B \cdot \alpha_B + \varepsilon} \cdot \left(\sqrt{\frac{E}{\varepsilon} \cdot \frac{\Omega_B \cdot \alpha_B + \varepsilon}{\Omega_B \cdot A_B} + 1} - 1 \right) \quad (10)$$

Though similar to the average case in Equation 9, the key takeaway is that $\tau_{B,opt(wc)}$ and $\tau_{B,opt}$ are never equal. The optimal time between backups in the worst case is always less than that of the average case, an important consideration when designing for tail latency.

3) Cost of Backups and Restores: Given the number of cycles between backups, should we focus on minimizing backup or restore overhead in a multi-backup system? This decision arises in situations where 1) an architect must optimize for the average task length [12], or 2) a programmer must optimize for periodic backups imposed by the system (e.g., watchdog timers [22]). At first glance, one may assume that it is always better to optimize backup cost since the system performs a restore only once per active period. But if the time between backups is too great, there may be insufficient energy to back up resulting in no progress. In this section, we explore the interplay between the time between backups and the backup/restore overhead.

As the time between backups becomes significantly large (τ_B approaches $+\infty$), the performance improvement of reducing the restore cost ($\frac{\partial p}{\partial e_R}$) outweighs that of the backup cost ($\frac{\partial p}{\partial e_B}$). Since progress is inversely proportional to backup and restore overhead, both partial derivatives are negative (i.e., a lower $\frac{\partial p}{\partial e_R}$ means that reducing e_R yields better performance). Investigating further, we solve for the number of cycles between backups at the break-even point ($\frac{\partial p}{\partial e_B} = \frac{\partial p}{\partial e_R}$):

$$\tau_{B,be} = \frac{2}{3} \cdot \frac{E - e_B - e_R}{\varepsilon} \quad (11)$$

From this, we have the following takeaways:

- If the time between backups is less than the break-even point ($\tau_B < \tau_{B,be}$), reduce the cost of backups.
- If the time between backups is greater than the break-even point ($\tau_B > \tau_{B,be}$), reduce the cost of restores.

As expected, with frequent backups (i.e., low τ_B), architects should focus on optimizing the backup mechanisms to improve performance. For example, a nonvolatile processor designer can choose to discard the state of some structures (e.g., instruction fetch queue, branch predictor) if they expect to back up often [38]. Conversely, as the time between backups increases, the restore overhead starts to dominate; it becomes more likely that no backup is invoked at all within an active period. When this happens, all execution is dead. As a result, we actually start to see more restore invocations than backup invocations when the time between backups exceeds the break-even point ($\tau_{B,be}$). With this analysis, architects and programmers gain a better understanding of where to focus their optimization efforts given the expected time between backups in their systems. For instance, in Clank [22], checkpoints occur due to idempotent violations and watchdog timers. Based on the

observed frequency of these checkpoints, the break-even point can inform the Clank architect to optimize the restore or backup overhead.

B. Single-Backup Systems

We characterize single-backup architectures as those that only invoke a single backup per active period when they observe that the energy supply has dropped too low, signaling an imminent power loss. Examples include Hibernus [6] and single-backup nonvolatile processor designs [38].

To model single-backup systems, the time between backups and number of dead cycles are set to extreme cases: $\tau_B = \tau_P$ and $\tau_D = 0$. We compute forward progress for single-backup systems with:

$$p = \frac{1 - \frac{(\Omega_B - \varepsilon_C/\sigma_B)\cdot A_B}{E} - \frac{e_R}{E}}{(1 + \frac{(\Omega_B - \varepsilon_C/\sigma_B)\cdot \alpha_B}{\varepsilon - \varepsilon_C}) \cdot (1 - \frac{\varepsilon_C}{\varepsilon})} \quad (12)$$

The key difference from the general case in Equation 8 is that there is no more dead execution energy; we effectively assume the best-case dead cycles from Equation 6. Though this is an advantage over multi-backup systems, this often comes at two costs. First, it introduces the risk of not having enough energy to perform a full backup and potentially leaving nonvolatile memory in an inconsistent state [42]. Second, the process of reading the voltage ADC and monitoring for imminent power losses is generally expensive, yielding up to 40% energy overhead [22].

C. Summary of Explorations

We present some implications from our EH model for designing efficient energy-harvesting systems. We show how our model characterizes state-of-the-art multi- and single-backup architectures. We explore the optimal time between backups and how it can help 1) programmers determine the granularity and size of tasks [43] and 2) architects configure optimal watchdog timers [22]. We provide architects with guidelines on when to optimize backup or restore overhead. Our model can compute lower and upper bounds on performance when accounting for the non-determinism of intermittent execution, useful when seeking aggressive performance gains [37]. Section VI dives deeper into additional cases studies to demonstrate the applications of our EH model. Next, we present experimental results of our model validation and characterizations.

V. Experimental Results

In this section, we present a hardware validation of the EH model's estimates of forward progress on a Texas Instruments LaunchPad platform [50] with an embedded MSP430FR5994 microcontroller. The MSP430 is used extensively in the energy-harvesting space [5], [6], [12], [34], [43]. First, we present a validation of a simple counter program to demonstrate that the EH model can capture the variation in forward progress seen when the cycles between backups is varied. Next, we show a validation of three state-of-the-art energy-harvesting systems: one single-backup system (i.e., Hibernus) and two multi-backup systems (i.e., Mementos and DINO). Finally,

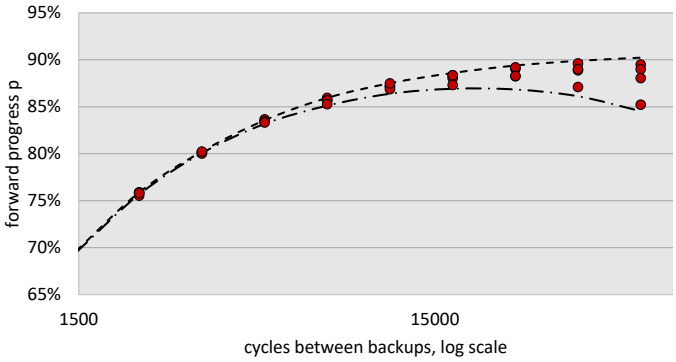

Fig. 5. Validating a multi-backup system. Points are measured from hardware. The dashed lines represent the upper and lower bounds estimated by the EH model.

obtaining model parameters (e.g., application state) for systems that do not exist in real hardware (such as Clank [22]) is only possible via simulation. Therefore, we extend a simulator from prior art [22] to characterize a range of applications. Namely, we profile the cycles between backups (τ_B), dead cycles (τ_D), and application state (α_B) parameters.

A. Validation

The first experiment mimics a multi-backup system, where backups are made at fixed time intervals (i.e., τ_B is constant throughout a single experimental run). We sweep time between backups through multiple runs, from 0.18ms to 7.1ms. The application increments a counter until an interrupt occurs every τ_B cycles, upon which the application backs up data representing its current state. The data backed up consists of an array and a timestamp, which are saved to the board's nonvolatile FRAM. The size of the array is equal to $\alpha_B \cdot \tau_B$ bytes, which varies as τ_B is varied. Application state (α_B) is set to be 0.1 bytes/cycle (Section V-B shows values for α_B from simulation—0.1 bytes/cycle is slightly below average). We measure the power consumption of instructions executing on the board using TI's EnergyTrace system [50]. Load and store operations to memory consume 1.2mW while all other instructions consume 1.05mW. We measure forward progress using onboard timers.

We also consider different active period lengths: 0.5s, 0.375s, 0.25s, and 0.125s based on prior work [6]. Figure 5 plots measured forward progress, with the dashed lines indicating the upper and lower bounds as predicted by the EH model (τ_P). The variance in the measured data points is due to the variation in dead cycles. That is, a backup may occur just before the end of an active period (very few dead cycles). Alternatively, the active period may end just before the next backup can occur (dead cycles is close to τ_B), the worst case for forward progress. The measured data points are within the bounds calculated from the EH model, demonstrating that it can capture the trends needed when exploring the design space of intermittent architectures.

The next experiment compares the forward progress measured on hardware vs. the forward progress predicted by the EH model for three energy-harvesting systems: Hibernus,

TABLE II
BENCHMARKS USED FOR HARDWARE VALIDATION.

Name	*Description*
RSA	Data encryption
CRC	Checksum calculation
SENSE	Statistics calculation of sensor data
AR	Activity recognition from sensor data
MIDI	Audio based data logging
DS	Key-value histogram based data logger

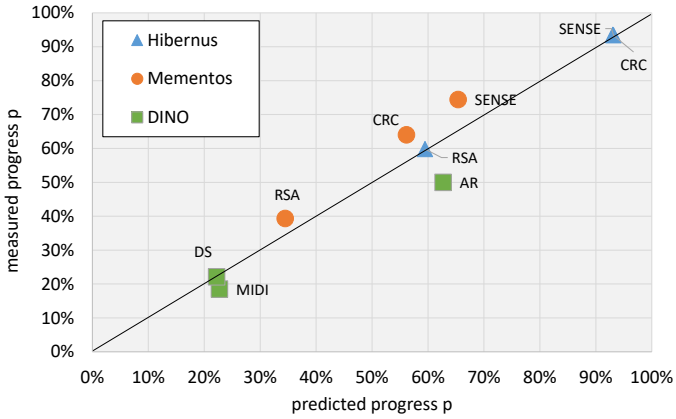

Fig. 6. Measured forward progress vs. predicted forward progress from EH model. The diagonal line corresponds to zero error in model estimate.

Mementos and DINO. We run the same benchmarks (Table II) as used in the original papers for each system. To track the exact cycles spent on each phase of execution (i.e., forward progress, backups, restores and dead cycles), we toggle a single-cycle pulse on a general purpose I/O (GPIO) pin at the start and end of each phase. These signals are captured and logged by a separate high-performance microcontroller. This allows us to faithfully capture how cycles are spent on the MSP microcontroller without affecting the energy of the platform (apart from the minimal energy cost of toggling a GPIO for 1 cycle each time). We also measure the exact duration of the active period by monitoring the voltage of the MSP microcontroller and comparing against the threshold voltages for power on and off specified by the device manufacturer.

Figure 6 shows the comparison of forward progress as predicted by the EH model vs. measurements carried out on the MSP Launchpad. The geometric mean error between predicted and actual values is 1.60%. Some applications such as *AR* and *MIDI* exhibit higher error for the DINO system. This is due to the fact that the EH model uses a single value for the cycles between backups (i.e., τ_B). However, these benchmarks use several different backup periods ranging from 17 cycles to over 14,000 cycles between backups. We use the average of these values as an input to the EH model, leading to larger error for these applications. Predicted forward progress is lower than the measured one for Mementos (geometric mean error of 6.97%). Recall that for a multi-backup system, the EH model uses

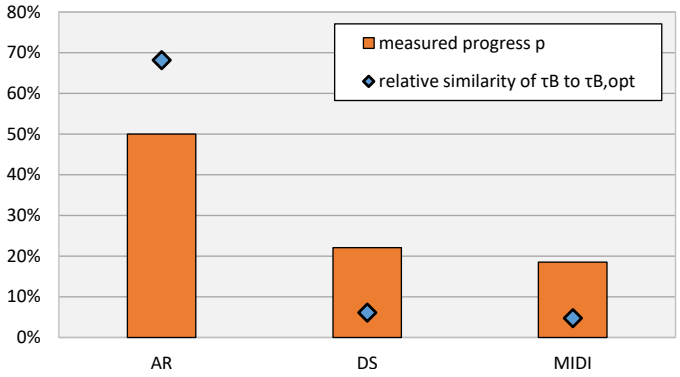

Fig. 7. Correlation between 1) measured forward progress and 2) relative similarity of measured τ_B to optimal $\tau_{B,opt}$ from EH model.

$\frac{\tau_B}{2}$ to estimate dead cycles (τ_D). However, the dead cycles in Mementos are dependent on the amount of energy left over after it has completed a backup. We account for this by subtracting the energy left over after hitting the minimum voltage threshold. However, Mementos continues to do work until it reaches the next checkpoint where it can perform another backup, which uses additional energy. This difference between the estimated energy left over at the minimum voltage threshold vs. the energy left over at the checkpoint leads to a lower forward progress prediction by the EH model.

Next, we validate that our estimate of an optimal time between backups ($\tau_{B,opt}$, Equation 9) correlates with the forward progress made by an application. Figure 7 shows the forward progress of the DINO benchmarks. We also plot the average τ_B using DINO compared to the optimal time between backups estimated by our model. The *AR* benchmark, which achieves an average τ_B that is nearly 70% of $\tau_{B,opt}$, has the highest forward progress. In contrast, *DS* and *MIDI* do not back up optimally and have significantly lower forward progress. Thus, the EH model can provide insights to the application designer on the ideal $\tau_{B,opt}$ they should aim for to maximize forward progress.

B. Characterization

We extend the simulator used in Clank [22] to better understand the values of the EH model parameters. We add a capacitor model to the simulator that supplies energy to the system. We supply harvested energy from recorded RF-based voltage traces [43] and evaluate a subset of the MiBench suite [18], compiled with GCC for ARM embedded processors (6-2017-q2-update). The simulator provides statistics for each active period (e.g., the number of backups, cycles between backups, dead cycles). We provide a brief description of Clank before presenting characterizations of the time between backups (τ_B) and (τ_D).

In Clank, backups are induced by idempotency violations [22]; an idempotent sequence of instructions is one that can be interrupted and re-executed yet still produce the same result. Idempotency violations are specifically caused by storing to a nonvolatile memory location that has been read at least once since the last backup. This forces a backup; if

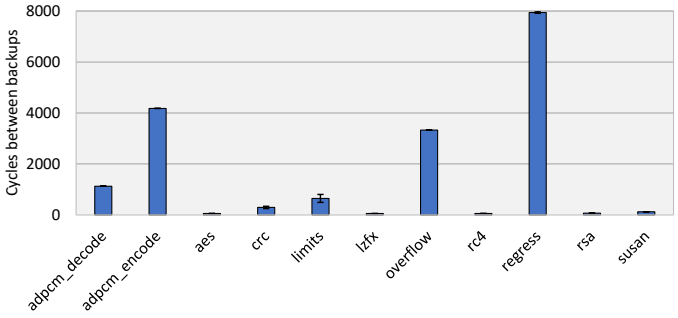

Fig. 8. The average τ_B with error bars for the standard error of the mean.

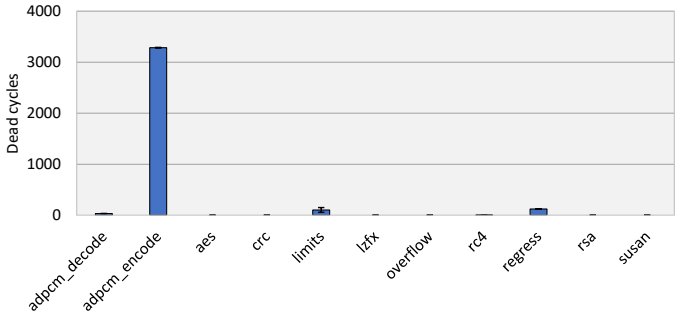

Fig. 9. The average τ_D with error bars for the standard error of the mean.

power were to be lost after the store instruction completes, re-executing from the previous checkpoint would read the new memory value (instead of the old one) and produce incorrect results. To identify violations, Clank uses volatile buffers and detection logic—the original paper proposed different buffers and optimizations to minimize backups. Our implementation of Clank employs an 8-entry read-first buffer and 8-entry write-first buffer to track violations. We use an 8000-cycle watchdog timer in case no idempotent violations occur in that time. Clank was based on an ARM Cortex-M0+, which requires 20 32-bit registers to be backed up and restored. We base our timings on the original paper [22] and our energy numbers on a datasheet for the ARM core [48].

Figures 8 and 9 plots the average time between backups (τ_B) and dead cycles (τ_D), respectively, across three voltage traces: 1) a trace that contains two short spikes of over 5V, while the troughs are very close to 0V; 2) a trace that gradually increases from a low voltage close to 0 to a voltage close to 2.5V; and 3) a trace that has multiple peaks (3.5-5.5V) and troughs (0-1.5V). The error bars represent the standard error of the mean, which is the standard deviation of the distribution divided by the square root of the sample size. The distributions we observed for both parameters were closely packed around the median, and while there was some variation caused by outliers, the error bars show that the variation is insignificant. We also see that the distributions across each voltage trace (not shown) were nearly identical. We believe this is a function of two things. First, active periods all have similar energy supplies (E) because the amount of energy that can be charged within an active period is very small. Second, because the energy supply is relatively stable, the sequence of idempotent violations observed will be

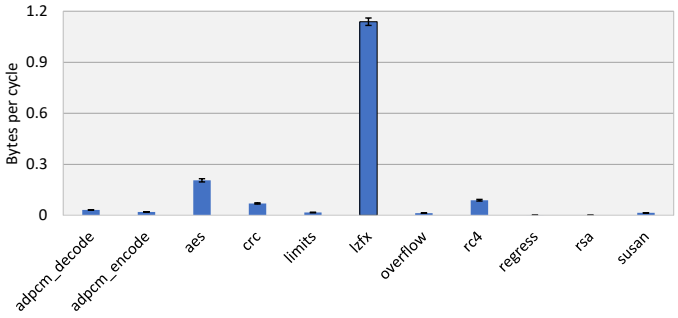

Fig. 10. The average α_B, with error bars for the standard error of the mean.

similar. Because idempotent violations signal backups, both τ_B and τ_D will see little variation. The very small values of dead cycles for some benchmarks reflects the similarly small values for time between backups; that is, τ_D cannot exceed τ_B.

To introduce greater variability, we simulate a hypothetical mixed-volatility processor that uses a parametrized watchdog timer to determine when to back up. We implement an unbounded store queue to track modifications to application state (α_B) within a backup period. Note that the store queue will require additional time (σ_B) and energy (Ω_B) to back up. Figure 10 plots the average application state (α_B) when using different watchdog timers for backups (250-3000 cycles between backups, in increments of 250 cycles). We can see that the bytes per cycle to be backed up is low across most benchmarks (on average 0.16 bytes per cycle).

Combined with the time between backups, Figure 10 helps us understand how much state needs to be backed up to make forward progress. The data can also help us understand the likelihood of idempotent violations. For example, *lzfx*'s frequent backups in Clank (Figure 8) are likely due to its very high rate of stores. Thus, the parameters in Table I can be chosen to recreate the scenarios exhibited in realistic benchmarks and evaluate future intermittent processing devices.

VI. Case Studies

In this section, we apply our EH model to case studies of state-of-the-art intermittent processor systems.

A. Store-Major Locality

In conventional architectures, high cache locality for load instructions is generally far more beneficial to performance than locality for store instructions. Unlike reading data, modifying data is typically off the critical path of execution. State-of-the-art intermittent processor systems may be equipped with volatile (or hybrid volatile/nonvolatile) caches [31], [56] between the processor and nonvolatile memory. Upon a backup, all dirty data must be saved to nonvolatile memory; this process often lies on the critical path of execution to prevent inconsistent nonvolatile updates. In these systems, programmers and architects must reconsider the trade-offs between load and store locality, which we explore in this section.

Consider the example matrix transpose program in Listing 1. With both arrays `A` and `B` encoded in standard row-major order in memory, the conventional approach is to iterate through

Listing 1. Store-major loop example.

```
1  // conventional load-major
2  for (i = 0; i < m; i++)
3      for (j = 0; j < n; j++)
4          B[j][i] = A[i][j];
5
6  // store-major
7  for (i = 0; i < m; i++)
8      for (j = 0; j < n; j++)
9          B[i][j] = A[j][i];
```

Listing 2. Circular buffer for idempotency example.

```
1  // conventional
2  for (i = 0; i < n; i++)
3      A[i] = f(A[i]);
4
5  // circular buffer for idempotency
6  for (i = 0; i < n; i++)
7      A[(A.head + n + i) % N] = f(A[(A.head + i) % N]);
8  A.head = (A.head + n) % N;
```

the nested loop in load-major order (i.e., the innermost loop iterates through the contiguous dimension of the array being read). On one hand, this yields maximum cache locality for load instructions, suffering a cache miss only once every $\beta_{block}/\beta_{load}$ loads on average, where β_{block} is the cache block size and β_{load} is the number of bytes per load. On the other hand, store instructions incur a cache miss on every access, assuming array `B` does not fit in the cache. In conventional architectures, this is of less concern for performance since the processor can often continue executing without waiting for the store miss to be serviced by memory. However, in an intermittent processor system, load-major ordering increases the number of bytes to back up by a factor of $\beta_{block}/\beta_{store}$ on average compared to store-major ordering, where β_{store} is the number of bytes per store instruction. This is due to the fact that dirty state is tracked at cache block granularity instead of byte granularity; the latter would be too expensive in metadata overhead. To illustrate this, we compare the load-major and store-major loops (Line 6) in our example program. With store-major ordering, if a backup is invoked after $\beta_{block}/\beta_{store}$ store instructions in the inner loop, there would only be one dirty block in the cache due to maximum store locality. However, with load-major ordering, after the same number of iterations, $\beta_{block}/\beta_{store}$ cache blocks must now be backed up, even though only β_{store} bytes have been modified in each of the blocks.

We can characterize the ratio of performance overhead (in cycles) with load-major loops ($\tau_{load\text{-}major}$) to store-major loops ($\tau_{store\text{-}major}$) as:

$$\frac{\tau_{load\text{-}major}}{\tau_{store\text{-}major}} = \frac{\frac{\alpha_{load}\cdot\tau_P}{\sigma_{load}} + \frac{(\beta_{block}/\beta_{store})\cdot n_B\cdot\alpha_B\cdot\tau_B}{\sigma_B}}{\frac{(\beta_{block}/\beta_{load})\cdot\alpha_{load}\cdot\tau_P}{\sigma_{load}} + \frac{n_B\cdot\alpha_B\cdot\tau_B}{\sigma_B}} \tag{13}$$

where α_{load} is the average number of bytes read by the application per cycle and σ_{load} is the nonvolatile memory bandwidth for load operations. Simplifying Equation 13, we find that store-major loops can improve performance over load-major loops only when:

$$\frac{\alpha_B\cdot(\frac{\beta_{block}}{\beta_{store}}-1)}{\alpha_{load}\cdot(\frac{\beta_{block}}{\beta_{load}}-1)} > \frac{\sigma_B}{\sigma_{load}} \tag{14}$$

The first term represents the ratio of unique dirty blocks that the application backs up to unique blocks that the application loads into the cache. The second term represents the ratio of backup bandwidth to read bandwidth on the nonvolatile memory device. Programmers should transform their loops to store-major order if either 1) the application is expected to have a larger write footprint than read footprint, or 2) backing up in nonvolatile memory is expensive relative to reading from it. In our example (Listing 1), the read and write footprints of the matrix transpose program are equal in size. If we assume the same latency for reading from and backing up to nonvolatile memory (i.e., $\sigma_{load} = \sigma_B$), we see that load-major and store-major ordering yield the same performance, a takeaway that does not apply to conventional architectures. For some memory devices, write latency can be much higher than read latency (e.g., 10× for STT-RAM [56]), making store-major loops beneficial. With this case study, we highlight the importance of reconsidering conventional trade-offs and present new insights for optimizing programs for cache locality on intermittent processor architectures.

B. Circular Buffers for Idempotency

In a conventional single-threaded system, there is generally no need for programmers to concern themselves with eliminating (or inducing) write-after-read memory dependencies. A store instruction to some memory location can come either immediately after or long after a load instruction from that same location; neither would be particularly better for performance. However, in the Clank processor [22], this is an important design consideration. As discussed in Section V, backups are induced by idempotency violations in Clank. Thus the average time between idempotency violations in a program (i.e., the time between each violating store instruction and its preceding load) dictates the frequency of backups in the system. Since there is a sweet spot in how frequently we should invoke backups to maximize forward progress (Section IV-A1), the ability to control idempotency violations is important for Clank.

Using Equation 9 from our model exploration, we introduce a general technique for programmers to tune the idempotent regions in their application (i.e., time between idempotency violations) to match the optimal time between backups ($\tau_{B,opt}$) of the Clank architecture. We propose storing program arrays in circular buffers in nonvolatile memory; the difference in buffer size to array size controls the length of idempotent regions. An example program snippet is shown in Listing 2. In the conventional case, each iteration of the loop invokes an idempotency violation due to first reading `A[i]` then writing to it next. As a result, backups are very frequent, which may be undesirable depending on the system parameters.

To address this, as long as the accesses to array `A` are in ascending order during the loop, the code can be transformed to use a circular buffer instead, shown in Line 7. With a larger circular buffer (of size `N`) relative to the array size (`n`),

idempotency violations are effectively postponed.[3] In general, the average number of store instructions to A between violations can be computed as $\mathtt{N}-\mathtt{n}+1$.[4] $\mathtt{N}=\mathtt{n}$ implies the conventional case with no circular buffering while $\mathtt{N}=2\cdot\mathtt{n}$ implies double buffering. With this analysis, given the configuration of the underlying Clank architecture, the programmer can maximize performance by solving for the optimal circular buffer size ($\mathtt{N}_{opt}$):

$$\Big(\mathtt{N}_{opt}-\mathtt{n}+1\Big)\cdot\tau_{store}=\tau_{B,opt} \tag{15}$$

where τ_{store} is the average number of cycles between store instructions, which can be obtained via application profiling. With this case study, we show how our EH model can reveal and characterize unconventional optimizations for intermittent processor systems.

C. Reduced Bit-Precision Backups

Recently there has been an increased interest in reduced bit-precision computation [16], [26], [36] and storage [24], [41] in the architecture community. This is motivated by two key trends:

1) The growing ubiquity of low-energy devices (e.g., mobile, internet-of-things, sensor networks) have forced designers to fight for every last nanojoule of energy under such tight resource constraints.
2) The widespread use of cognitive (e.g., deep learning, natural language processing) and approximate (e.g., computer vision, audio and video processing) applications have opened opportunities for reduced-precision techniques due to their natural resilience for bit errors.

Intermittent processor architectures provide a unique opportunity for exploiting reduced bit-precision in the backup process.[5] Unlike in conventional systems, intermittent processors frequently save state and thus incur more writes to nonvolatile memory. In this section, we dive deeper and analyze the factors that control the performance benefit of reducing bits.

First, we look at reducing bits in architectural (A_B) and application (α_B) state. In terms of architectural state, data words in the register file are attractive candidates for precision reduction since they are a fixed cost per backup. Application-specific data that has been modified since the last backup—either in volatile buffers [12], volatile caches [31], [56] or write-back buffers [22]—can be large when backups are infrequent. Solving for $\frac{\partial p}{\partial \alpha_B}$ and $\frac{\partial p}{\partial A_B}$ in Equation 8 for a multi-backup system (the analysis is straightforward in single-backup cases), we find that regardless of how large the architectural state is (for $A_B>0$) and how small the application state is (for $\alpha_B>0$), the performance benefit of reducing application state is always higher. Specifically, $\frac{\partial p}{\partial \alpha_B}\leq\frac{\partial p}{\partial A_B}$ for $\tau_B\geq 1$; both partial derivatives are negative since the amount of state to back up inversely impacts forward progress. In addition, approximating architectural state is risky and if not done carefully, can lead to incorrect program control flow. For example, architectural state such as the program counter (PC) and any registers might contain memory addresses must be saved precisely for correct program execution. Thus for the remainder of this section, we focus on approximating application state alone.

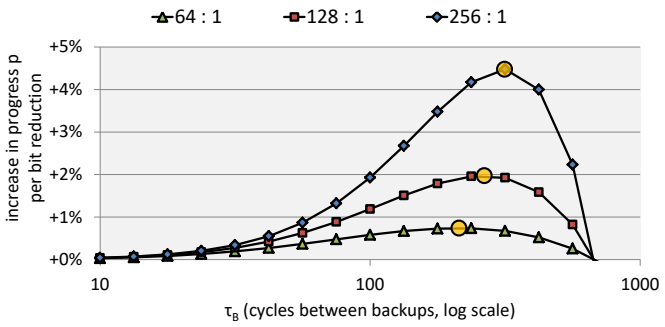

Fig. 11. Increase in progress per bit reduction ($|\frac{\partial p}{\partial \alpha_B}|$) with varying τ_B for benchmark *susan* on Clank [22]. Each curve corresponds to a different ratio of $\Omega_B\cdot A_B$ to $\Omega_B\cdot\alpha_B+\varepsilon$. Assumes $e_C=0$ and $\Omega_R=0$. The yellow dots indicate the optimal time between backups ($\tau_{B,bit}$).

Given these observations, under what circumstances is it most beneficial to reduce bit-precision in application state? The largest bit-precision reduction (i.e., maximum $|\frac{\partial p}{\partial \alpha_B}|$) is achieved when the time between backups is:

$$\tau_{B,bit}=\frac{3}{2}\cdot\frac{\Omega_B\cdot A_B}{\Omega_B\cdot\alpha_B+\varepsilon}\cdot\left(\sqrt{\frac{16}{9}\cdot\frac{E}{\varepsilon}\cdot\frac{\Omega_B\cdot\alpha_B+\varepsilon}{\Omega_B\cdot A_B}+1}-1\right) \tag{16}$$

As expected, this is primarily dictated by the ratio of compulsory architectural energy ($\Omega_B\cdot A_B$) to the energy cost proportional to how much work was done since the last backup ($\Omega_B\cdot\alpha_B+\varepsilon$). This is shown in Figure 11, which plots how the benefit of reduced precision ($|\frac{\partial p}{\partial \alpha_B}|$) varies with the time between backups for benchmark *susan* running on a Clank system [22], configured as in Section V-A. As the ratio $\frac{\Omega_B\cdot A_B}{\Omega_B\cdot\alpha_B+\varepsilon}$ decreases,[6] bit-precision reduction favours more frequent backups (i.e., $\tau_{B,bit}$ decreases) since the overhead of saving application state is more impactful. A small $\frac{\Omega_B\cdot A_B}{\Omega_B\cdot\alpha_B+\varepsilon}$ ratio is common when there are few architectural registers while a large ratio is common in applications with small volatile memory footprints. With Equation 16, we provide architects with the sweet spot for deciding when to employ reduced-precision optimizations. For example, suppose we have an architecture with a large register file, configured as in the top curve in Figure 11. If we reduce precision by just 1 bit, we can improve forward progress by up to 4.5% when the time between backups is at its optimal ($\tau_{B,bit}=315$ cycles). Note that this analysis does not factor in application error due to reduced bit-precision. These curves merely help architects determine if the benefit of reduced-precision will be sufficient to warrant further investigation.

[3]The extra overhead of indexing into the circular buffer is negligible when N is a power of 2.

[4]Accounting for the write-back buffer in Clank [22] simply involves adding the write-back buffer size w to the equation.

[5]The same analysis extends to the restore process as well; we only focus on backups to simplify the discussion.

[6]We control this ratio by varying α_B; all other parameters are set from our Clank experiments in Section V.

D. Summary of Case Studies

We demonstrate how our EH model can 1) expose unconventional insights and 2) catalyze new techniques and optimizations on state-of-the-art intermittent computing architectures. We show that programmers and designers need to rethink the trade-offs between load and store cache locality (Section VI-A). In Section VI-B, we propose a program transformation that tunes for the optimal time between backups on Clank processors [22]. We also explore the performance potential of reduced bit-precision, a popular optimization in emerging architectures (Section VI-C).

VII. Related Work

In this section, we look at evaluation methodologies for intermittent processor architectures and related work in modeling resilience for real-time and high-performance computing.

Analytical Models. *Energy driven computing* provides a taxonomy of energy-harvesting designs, spanning power-neutral, transient, and energy-driven systems [40]. Hibernus details a mathematical model comparing the time spent on execution for their system as well as Mementos and show that they achieve lower overhead [6]. Mathematical models of data sensed by energy-harvesting systems can aid designers [9]. Rodriguez et al. build on the Hibernus analytical model and expand it to show the breakdown of time and energy for three prior approaches, namely Mementos [43], Hibernus [5] and Quick Recall [25] across various application scenarios. They show that Hibernus achieves lower overhead than Mementos and is more energy efficient at lower interruption frequencies compared to QuickRecall, while QuickRecall is more energy efficient at higher frequencies. While these approaches aim to aid designers to pick the optimal system for their situation, they are limited to making a recommendation from a small set of approaches. In contrast, our model targets a wider range of intermittent computing architectures, including checkpoint-based and non-volatile processor systems.

Simulation. Simulation has been used to evaluate different devices [6], [22], [38]. Through simulation, Clank identified that re-execution cost (i.e., dead cycles) can outweigh the cost of checkpointing [22]. The EH model allows similar observations to be made without the implementation details of a backup/restore mechanism, revealing insights from simple formulas. Several simulators enable exploration of various designs in the energy-harvesting space. NVPsim explores NVP designs by varying the choice of nonvolatile memory for on-chip caches, the backup strategy and size of the energy buffer [17]. SIREN supplements cycle-level simulation with real-world energy-harvesting conditions, captured from real power sources [15]. In contrast to these simulators, the EH model allows for more rapid and wide ranging exploration of the intermittent computing design space. Ekho proposes an I–V curve to capture how current changes when checkpointing with a *variable* energy supply [20]. Our EH model is more architecture centric, focusing on the active period and requiring the energy supply to be fully charged before execution resumes.

Resilience. Checkpointing is a common mechanism for resilience against faults in real-time and high-performance computing. Early models for resilience [13], [57] bear similarities to the multi-backup components of our model. This is because faults were approximated as periodic events (based on mean time to failure); though recent work [2], [3], [10], [19], [49] opts for more sophisticated probabilistic models. The EH model addresses a fundamentally different problem in general: faults are spontaneous events while power losses are progressively decaying events. This means that imminent power losses can be anticipated (via ADC checks) and postponed (via charging), both of which are unique to the single-backup and charging components of our model. Our model is also unique in its ability to account for architectural/application state and non-volatile memory technologies, whereas resilience models often focus on I/O congestion and thread scheduling [2], [3], [10], [19], [49]. Still, the parallels between our multi-backup model and prior models pose an interesting exploration space that can bridge innovations between intermittence and resilience.

VIII. Conclusion

We propose the EH model, a step towards better understanding the implications and complex interactions that arise in intermittent processor systems. We derive our model to compute forward progress as the fraction of harvested energy that is spent on useful processor execution, as opposed to energy spent on backups, restores, dead execution and reading the supply level. This makes for a clear understanding of how well a given architecture performs and under what circumstances it performs better. We present several explorations, hardware validations and case studies that demonstrate the effectiveness of our model for both architects and programmers. In particular, we show how our model reveals new insights and unconventional design considerations pertaining to cache locality, idempotency and reduced bit-precision. Our EH model facilitates early design space explorations, to both assist and excite research in this field.

Acknowledgements

The authors thank the anonymous reviewers and members of the NEJ group for their valuable feedback. The authors also thank Alexei Colin for his assistance in replicating the hardware experiments. This research has been funded in part by the Computing Hardware for Emerging Intelligent Sensory Applications (COHESA) project. COHESA is financed under the National Sciences and Engineering Research Council of Canada (NSERC) Strategic Networks grant number NETGP485577-15. This work is further supported by a Percy Edward Hart Early Career Professorship, an IBM PhD Fellowship, the University of Toronto and the University of Wisconsin-Madison.

References

[1] F. A. Aouda, K. Marquet, and G. Salagnac, "Incremental Checkpointing of Program State to NVRAM for Transiently-Powered Systems," in *Proceedings of the 9th International Symposium on Reconfigurable and Communication-Centric Systems-on-Chip*. IEEE, 2014, pp. 1–4.

[2] S. Arunagiri, S. Seelam, R. Oldfield, M. Ruiz Varela, P. Teller, and R. Riesen, "Impact of checkpoint latency on the optimal checkpoint interval and execution time," University of Texas at El Paso, Departmental Technical Reports (CS) UTEP-CS-07-55a, 2008.

[3] G. Aupy, A. Benoit, T. Hérault, Y. Robert, and J. Dongarra, "Optimal checkpointing period: Time vs. energy," in *International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems*, 2014.

[4] D. Balsamo, A. Das, A. S. Weddell, D. Brunelli, B. M. Al-Hashimi, G. V. Merrett, and L. Benini, "Graceful Performance Modulation for Power-Neutral Transient Computing Systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 5, pp. 738–749, 2016.

[5] D. Balsamo, A. S. Weddell, A. Das, A. R. Arreola, D. Brunelli, B. M. Al-Hashimi, G. V. Merrett, and L. Benini, "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.

[6] D. Balsamo, A. S. Weddell, G. V. Merrett, B. M. Al-Hashimi, D. Brunelli, and L. Benini, "Hibernus: Sustaining Computation During Intermittent Supply for Energy-Harvesting Systems," *IEEE Embedded Systems Letters*, vol. 7, no. 1, pp. 15–18, 2015.

[7] G. Berthou, T. Delizy, K. Marquet, T. Risset, and G. Salagnac, "Peripheral State Persistence for Transiently-Powered Systems," in *2017 Global Internet of Things Summit*. IEEE, 2017, pp. 1–6.

[8] D. Bharadia, K. R. Joshi, M. Kotaru, and S. Katti, "BackFi: High Throughput WiFi Backscatter," in *Proceedings of the ACM Conference on Special Interest Group on Data Communication*. ACM, 2015, pp. 283–296.

[9] P. Bogdan, M. Pajic, P. P. Pande, and V. Raghunathan, "Making the internet-of-things a reality: From smart models, sensing and actuation to energy-efficient architectures," in *IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, 2016.

[10] G. Bosilca, A. Bouteiller, E. Brunet, F. Cappello, J. Dongarra, A. Guermouche, T. Herault, Y. Robert, F. Vivien, and D. Zaidouni, "Unified model for assessing checkpointing protocols at extreme-scale," *Concurrency and Computation: Practice and Experience*, 2014.

[11] A. P. Chandrakasan, D. C. Daly, J. Kwong, and Y. K. Ramadass, "Next Generation Micro-power Systems," in *Proceedings of the Symposium on VLSI Circuits*. IEEE, 2008, pp. 2–5.

[12] A. Colin and B. Lucia, "Chain: Tasks and Channels for Reliable Intermittent Programs," in *Proceedings of the International Conference on Object-Oriented Programming, Systems, Languages, and Applications*. IEEE, 2016, pp. 514–530.

[13] J. Daly, "A model for predicting the optimum checkpoint interval for restart dumps," in *Proceedings of the International Conference on Computational Science*, 2003.

[14] D. Dondi, A. Bertacchini, D. Brunelli, L. Larcher, and L. Benini, "Modeling and optimization of a solar energy harvester system for self-powered wireless sensor networks," *IEEE Transactions on Industrial Electronics*, 2008.

[15] M. Furlong, J. Hester, K. Storer, and J. Sorber, "Realistic Simulation for Tiny Batteryless Sensors," in *Proceedings of the 4th International Workshop on Energy Harvesting and Energy-Neutral Sensing Systems*. ACM, 2016, pp. 23–26.

[16] K. Ganesan, J. San Miguel, and N. Enright Jerger, "The What's Next Intermittent Architecture," in *Workshop on Approximate Computing Across the Stack*, 2018.

[17] Y. Gu, Y. Liu, Y. Wang, H. Li, and H. Yang, "NVPsim: A Simulator for Architecture Explorations of Nonvolatile Processors," in *Proceedings of the 21st Asia and South Pacific Design Automation Conference*. IEEE, 2016, pp. 147–152.

[18] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "MiBench: A free, commercially representative embedded benchmark suite," in *Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization*. IEEE, 2001, pp. 3–14.

[19] T. Herault, Y. Robert, A. Bouteiller, D. Arnold, K. Ferreira, G. Bosilca, and J. Dongarra, "Optimal cooperative checkpointing for shared high-performance computing platforms," in *Workshop on Advances in Parallel and Distributed Computational Models*, 2018.

[20] J. Hester, T. Scott, and J. Sorber, "Ekho: Realistic and Repeatable Experimentation for Tiny Energy-harvesting Sensors," in *Proceedings of the 12th Conference on Embedded Network Sensor Systems*. ACM, 2014, pp. 330–331.

[21] J. Hester, K. Storer, and J. Sorber, "Timely execution of intermittently powered batteryless sensors," in *Proceedings of the 15th Conference on Embedded Network Sensor Systems*, 2017, pp. 1–13.

[22] M. Hicks, "Clank: Architectural support for intermittent computation," in *Proceedings of the 44th International Symposium on Computer Architecture*. ACM, 2017, pp. 228–240.

[23] M. A. A. Ibrahim, M. M. Aboudina, and A. N. Mohieldin, "An ultra-low-power MPPT architecture for photovoltaic energy harvesting systems," in *Proceedings of the 17th International Conference on Smart Technologies*, 2017.

[24] A. Jain, P. Hill, S. C. Lin, M. Khan, M. E. Haque, M. A. Laurenzano, S. Mahlke, L. Tang, and J. Mars, "Concise loads and stores: The case for an asymmetric compute-memory architecture for approximation," in *Proceedings of the International Symposium on Microarchitecture*. IEEE, 2016, pp. 1–13.

[25] H. Jayakumar, A. Raha, and V. Raghunathan, "QuickRecall: A low overhead HW/SW approach for enabling computations across power cycles in transiently powered computers," in *27th International Conference on VLSI Design and 13th International Conference on Embedded Systems*, 2014.

[26] P. Judd, J. Albericio, T. Hetherington, T. M. Aamodt, and A. Moshovos, "Stripes: Bit-serial deep neural network computing," in *Proceedings of the International Symposium on Microarchitecture*. IEEE, 2016, pp. 1–12.

[27] T. Kazmierski and S. Beeby, *Energy Harvesting Systems: Principles, Modelling and Applications*. Springer, 2011.

[28] S. Khanna, S. C. Bartling, M. Clinton, S. Summerfelt, J. A. Rodriguez, and H. P. McAdams, "An FRAM-based nonvolatile logic MCU SoC exhibiting 100% digital state retention at VDD = 0 V achieving zero leakage with $<$ 400-ns wakeup time for ULP applications," *IEEE Journal of Solid-State Circuits*, vol. 49, no. 1, pp. 95–106, 2014.

[29] S. Kiran, D. Selvakumar, M. J, and H. Pasupuleti, "Modeling, simulation and analysis of piezoelectric energy harvester for wireless sensors," in *International Conference on Control, Electronics, Renewable Energy and Communications*, 2015.

[30] F. Li, K. Qiu, M. Zhao, J. Hu, Y. Liu, Y. Guan, and C. J. Xue, "Checkpointing-aware Loop Tiling for Energy Harvesting Powered Non-Volatile Processors," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. PP, no. 99, pp. 1–14, 2018.

[31] H. Li, Y. Liu, Q. Zhao, Y. Gu, X. Sheng, G. Sun, C. Zhang, M.-F. Chang, R. Luo, and H. Yang, "An energy efficient backup scheme with low inrush current for nonvolatile SRAM in energy harvesting sensor nodes," in *Proceedings of the Design, Automation and Test in Europe Conference*. EDA Consortium, 2015, pp. 7–12.

[32] Q. Liu and C. Jung, "Lightweight hardware support for transparent consistency-aware checkpointing in intermittent energy-harvesting systems," in *5th Non-Volatile Memory Systems and Applications Symposium*, 2016, pp. 1–6.

[33] Y. Liu, Z. Wang, A. Lee, F. Su, C. P. Lo, Z. Yuan, C. C. Lin, Q. Wei, Y. Wang, Y. C. King, C. J. Lin, P. Khalili, K. L. Wang, M. F. Chang, and H. Yang, "A 65nm ReRAM-enabled nonvolatile processor with 6x reduction in restore time and 4x higher clock frequency using adaptive data retention and self-write-termination nonvolatile logic," in *Proceedings of the International Solid State Circuits Conference*, 2016, pp. 84–86.

[34] B. Lucia and B. Ransford, "A Simpler, Safer Programming and Execution Model for Intermittent Systems," in *Proceedings of the 36th Conference on Programming Language Design and Implementation*. ACM, 2015, pp. 575–585.

[35] G. Lukosevicius, A. R. Arreola, and A. S. Weddell, "Using sleep states to maximize the active time of transient computing systems," in *Proceedings of the 5th International Workshop on Energy Harvesting and Energy-Neutral Sensing Systems*, 2017.

[36] K. Ma, X. Li, J. Li, Y. Liu, Y. Xie, J. Sampson, M. T. Kandemir, and V. Narayanan, "Incidental Computing on IoT Nonvolatile Processors," in *Proceedings of the International Symposium on Microarchitecture*. ACM, 2017, pp. 204–218.

[37] K. Ma, X. Li, S. R. Srinivasa, Y. Liu, J. Sampson, Y. Xie, and V. Narayanan, "Spendthrift: Machine Learning Based Resource and Frequency Scaling for Ambient Energy Harvesting Nonvolatile Processors," in *Proceedings of the 22nd Asia and South Pacific Design Automation Conference*. IEEE, 2017, pp. 678–683.

[38] K. Ma, Y. Zheng, S. Li, K. Swaminathan, X. Li, Y. Liu, J. Sampson, Y. Xie, and V. Narayanan, "Architecture Exploration for Ambient Energy Harvesting Nonvolatile Processors," in *Proceedings of the 21st International Symposium on High Performance Computer Architecture*. IEEE, 2015, pp. 526–537.

[39] K. Maeng, A. Colin, and B. Lucia, "Alpaca: Intermittent execution without checkpoints," *Proceedings of the International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pp. 96:1–96:30, 2017.

[40] G. V. Merrett and B. Al-Hashimi, "Energy-driven computing: Rethinking the design of energy harvesting systems," in *Conference on Design Automation and Test in Europe*, 2017.

[41] T. Moreau, J. San Miguel, M. Wyse, J. Bornholt, A. Alaghi, L. Ceze, N. Enright Jerger, and A. Sampson, "A taxonomy of general purpose approximate computing techniques," *IEEE Embedded Systems Letters*, vol. 10, no. 1, pp. 2–5, 2018.

[42] B. Ransford and B. Lucia, "Nonvolatile Memory is a Broken Time Machine," in *Proceedings of the Workshop on Memory Systems Performance and Correctness*. ACM, 2014, pp. 1–3.

[43] B. Ransford, J. Sorber, and K. Fu, "Mementos: System Support for Long-running Computation on RFID-scale Devices," in *Proceedings of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 2011, pp. 159–170.

[44] J. San Miguel, K. Ganesan, M. Badr, and N. Enright Jerger, "The EH model: Analytical exploration of energy-harvesting architectures," *IEEE Computer Architecture Letters*, vol. 17, no. 1, pp. 76–79, 2018.

[45] S. Senni, L. Torres, A. Gamatie, and G. Sassatelli, "Non-volatile processor based on MRAM for ultra-low-power IoT devices," *ACM Journal of Emerging Technologies in Computing*, 2017.

[46] S. Shevtsov and S.-H. Chang, "Modeling of vibration energy harvesting system with power PZT stack loaded on Li-Ion battery," in *3rd European Conference on Renewable Energy Systems*, 2015.

[47] A. K. Sinha and M. C. Schneider, "Short startup, batteryless, self-starting thermal energy harvesting chip working in full clock cycle," in *IET Circuits, Devices and Systems*, vol. 11, 2017, pp. 521–528.

[48] *Access line ultra-low-power 32-bit MCU ARM-based Cortex-M0+, up to 16KB Flash, 2KB SRAM, 512B EEPROM, ADC*, STMicroelectronics, 2016, Rev. 2.

[49] O. Subasi, G. Kestor, and S. Krishnamoorthy, "Toward a general theory of optimal checkpoint placement," in *IEEE International Conference on Cluster Computing*, 2017.

[50] *MSP430 LaunchPad Development Kit*, Texas Instruments, 2018.

[51] T. Thakuria, H. K. Singh, and T. Bezboruah, "Modelling and simulation of low cost RF energy harvesting system," in *2017 International Conference on Innovations in Electronics, Signal Processing and Communication*, 2017.

[52] A. Wang, V. Iyer, V. Talla, J. R. Smith, and S. Gollakota, "FM Backscatter: Enabling Connected Cities and Smart Fabrics," in *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation*. USENIX Association, 2017, pp. 243–258.

[53] Y. Wang, Y. Liu, S. Li, D. Zhang, B. Zhao, M.-F. Chiang, Y. Yan, B. Sai, and H. Yang, "A 3us wake-up time nonvolatile processor based on ferroelectric flip-flops," in *Proceedings of 38th European Solid-State Circuits Conference*, 2012.

[54] J. V. D. Woude and M. Hicks, "Intermittent computation without hardware support or programmer intervention," in *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation*, 2016.

[55] M. Xie, M. Zhao, C. Pan, J. Hu, Y. Liu, and C. J. Xue, "Fixing the broken time machine: Consistency-aware checkpointing for energy harvesting powered non-volatile processor," in *Proceedings of the 52nd ACM/EDAC/IEEE Design Automation Conference*, 2015, pp. 1–6.

[56] M. Xie, M. Zhao, C. Pan, H. Li, Y. Liu, Y. Zhang, C. J. Xue, and J. Hu, "Checkpoint Aware Hybrid Cache Architecture for NV Processor in Energy Harvesting Powered Systems," in *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis*. IEEE, 2016, pp. 22:1–22:10.

[57] J. W. Young, "A first order approximation to the optimum checkpoint interval," *Communications of the ACM*, vol. 17, no. 9, 1974.

[58] M. Zhao, K. Qiu, Y. Xie, J. Hu, and C. J. Xue, "Redesigning software and systems for non-volatile processors on self-powered devices," in *2016 IFIP/IEEE International Conference on Very Large Scale Integration*, 2016, pp. 1–6.