PHILOSOPHICAL TRANSACTIONS A

rsta.royalsocietypublishing.org

Research

Article submitted to journal

Subject Areas:
internet-of-things (IoT), embedded systems

Keywords:
energy-driven computing, batteryless computing, intermittent computing, energy harvesting

Author for correspondence:
Sivert T. Sliper
e-mail: s.sliper@ecs.soton.ac.uk

Energy-Driven Computing

Sivert T. Sliper, Oktay Cetinkaya, Alex S. Weddell, Bashir Al-Hashimi and Geoff V. Merrett

Centre for IoT and Pervasive Systems,
School of Electronics and Computer Science,
University of Southampton,
United Kingdom

For decades, the design of untethered devices has been focused on delivering a fixed quality of service with minimum power consumption, to enable battery-powered devices with reasonably long deployment lifetime. However, to realise the promised tens of billions of connected devices in the Internet of Things, computers must operate autonomously and harvest ambient energy to avoid the cost and maintenance requirements imposed by mains- or battery-powered operation. But harvested power typically fluctuates, often unpredictably, and with large temporal and spatial variability. Energy-driven computers are designed to treat energy-availability as a first-class citizen, in order to gracefully adapt to the dynamics of energy harvesting. They may sleep through periods of no energy, endure periods of scarce energy, and capitalise on periods of ample energy. In this paper, we describe the promise and limitations of energy-driven computing, with an emphasis on intermittent operation.

THE ROYAL SOCIETY PUBLISHING

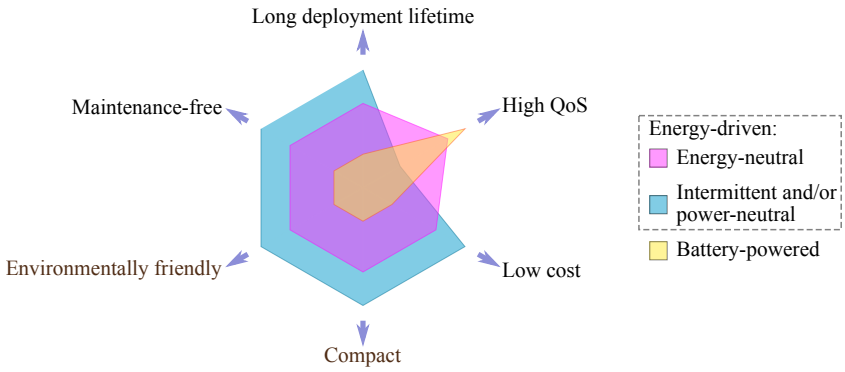

Figure 1: Potential of energy-driven computers in the context of low-power computing devices.

1. Introduction

The Internet of Things (IoT) is driving the proliferation of low-power computing devices, bridging the cyber- and physical worlds with advanced monitoring, control, and management capabilities [1]. These devices are extensively employed in a variety of domains, such as smart buildings and cities, health monitoring, remote surveillance, industrial automation, and wearable computing [2]. However, despite gaining momentum, the services provided in the IoT context have some inevitable drawbacks.

The number of connected devices is projected to be 22 billion by 2024[1]. At this scale, maintenance operations, such as battery replacement and/or replenishment, are likely to become practically and economically infeasible. Due to the profusion of devices, IoT services will seek very low cost and compact solutions, i.e. sub-dollar and sub-cm^3, capable of operating for decades without requiring maintenance, while also minimising environmental impact.

The architectural objectives for a computing device that can enable an IoT comprised of tens of billions of connected devices can be listed as: long maintenance-free deployment lifetime, low cost, small size, and minimal environmental impact. Figure 1 illustrates the potential of power-provisioning strategies in the context of the aforementioned architectural objectives. The following text introduces the three power-provisioning strategies, and qualitatively evaluates them against the axes of Fig. 1.

Non-rechargeable batteries can provide for computing devices with high quality of service (QoS)[2], but impose substantial increases in cost and volume, and limit deployment lifetime; furthermore, their eventual replacement and disposal imposes maintenance cost and environmental impact ("battery-powered" in Fig. 1). A promising augmentation to battery-power is energy harvesting (EH), which is the conversion of ambient energy into usable electrical power. Ambient energy sources are practically inexhaustible, so using EH in conjunction with rechargeable batteries improves the longevity of devices by supplying energy during deployment, and thus mitigates the need for periodic maintenance. However, as shown in Table 1, EH typically only offers low power density, and the output is not only highly variable, but also uncontrollable, and in many cases unpredictable [3].

The dynamics of EH motivate a shift towards energy-driven computing [4], where energy availability, and the energy environment, is a fundamental component of the system and application design process. In other words, energy-driven computers must not only consider application requirements but also the dynamics of their energy harvester and environment. For example, an energy-driven computer could dynamically adapt in order to provide *the*

[1]Number of devices connected to the Internet, excluding PCs, laptops, tablets and mobile phones. Source: https://www.ericsson.com/en/mobility-report/mobility-visualizer

[2]In this paper we regard QoS qualitatively as delivering accurate results on time and/or frequently. Lower QoS could entail reduced accuracy, reduced throughput and/or increased latency.

Table 1: Representative power density range of tiny energy harvesters.

EH Category	Energy Source	Power Density	Literature
Thermal	Heat	$10\mu W/cm^2$ - $1mW/cm^2$ (Thermoelectric)	[5] [6,7]
Fluid Flow	Wind	$100mW/cm^2$ (@ $2 \sim 9m/s$, airflow)	[5]
	Hydro	$30.67\mu W/cm^2$ (0.35ml @3.43m/s)	[8]
Radiated	RF	$0.2nW/cm^2$-$1\mu W/cm^2$	[6,8]
	Light	$100mW/cm^2$ (Outdoor, solar) $100\mu W/cm^2$ (Office, diffused light)	[9,10]
Kinetic	Vibration	$200\mu W/cm^3$ (Piezoelectrics) $\sim 800\mu W/cm^3$ (Machines - kHz)	[6,9]
Human Body	Biochemical	$180\mu W/cm^2$ (Glucose) $44\mu W/cm^2$ (Lactate)	[11,12]
	Biomechanical	$6\mu J/cm^3$ (Blood pressure) $1.2\mu W/cm^2$ (Heartbeat) N/A (Breathing) $\sim 4\mu W/cm^3$ (Locomotion - Hz)	[9,13] [11,12]

maximum achievable QoS for the energy available during deployment, as opposed to a traditional application-driven computer, which aims to minimise overall power consumption for a fixed QoS requirement.

Current energy-driven computers have different mechanisms for managing the dynamics of EH, which can be categorised into three groups: energy-neutral, power-neutral, and intermittent operation.

Energy-neutral operation [3] copes with the dynamics of EH by using energy storage, i.e. a rechargeable battery or a supercapacitor, that act as a buffer between energy supply and demand. Compared to their battery-powered counterparts, energy-neutral computers can maintain high QoS, while reducing the necessary storage capacity (and therefore cost/size), but are still limited by their use of rechargeable batteries or supercapacitors which impose cost and volume, and eventually expire due to limited charge cycles and/or ageing.

In contrast, **power-neutral** operation [4] seeks to cope with the dynamics of EH by adapting power consumption to instantaneously match harvested power, reducing, and potentially eliminating, the need for energy storage. However, without storage, volatile application state[3] is lost when the harvested power drops too low; when power returns, the application has to be restarted from the beginning. Consequently, power-neutral operation often cannot sustain long-running applications.

Intermittent operation is an orthogonal approach that does not attempt to hide the dynamics of EH, but rather accepts that power failures can be frequent and unpredictable [14]. By retaining computational progress through power cycles, intermittent computing systems (ICSs) allow long-running applications to progress incrementally whenever energy is available. The key merit of intermittent operation is that it can enable maintenance-free operation and theoretically unbounded lifetime at very low cost and volume, and with little environmental impact. But QoS must often be sacrificed because ICSs are not continuously operational (by definition). Hence,

[3]The volatile application state is any state that is lost in the event of a power outage, and that is needed for computation to proceed correctly when power returns.

intermittent operation is only applicable to certain applications. However, in the ideal case, where the harvested power directly coincides with the event of interest and is sufficiently powerful such that the system is guaranteed power during the events, the QoS may match that of a battery-powered system (see Section 3(a)).

Energy-driven computing has, in recent years, seen increasing research interest spanning areas such as programming languages and compiler techniques [15–17], embedded systems [18–22], hardware-software codesign [23], nonvolatile processors [24] and asynchronous circuits [25]. This paper outlines the state-of-the-art of energy-driven computing, with an emphasis on intermittent operation, and points out open research challenges. The contributions of this paper are:

- Our conditions for intermittent operation provides high-level answers to the question *"Which applications are amenable to intermittent operation?"*.
- A systematic taxonomy of intermittent computing (IC) methods that divides prior works into three classes, each evaluated against our set of correctness requirements, performance objectives and scalability objectives.
- An overview of unsolved challenges in energy-driven computing, with a particular focus on issues pertaining to communication under intermittent operation.

The remainder of this paper is organized as follows. First, energy- and power-neutral computing is discussed in Section 2. Then, in Section 3, conditions, correctness criteria, performance objectives and scalability objectives for intermittent operation are presented, followed by a taxonomy of prior works. We then extend our study to discuss unsolved challenges in energy-driven computing, in Section 4.

2. Energy- and power-neutral computing

This section describes two possible ways of operating a computer under an unstable power supply. The first, energy-neutral operation [3], seeks to hide the dynamics of EH by buffering energy, and potentially adapting application behaviour, to balance energy supply and demand over a long period. The second alternative, power-neutral operation, modulates power consumption to instantaneously match production, ideally obviating the need for energy storage.

Both methods may exploit trade-offs between QoS and power consumption by modulating their operating frequency, activating and deactivating peripherals, selectively sampling sensors, and/or using techniques such as approximate computing [26–29].

(a) Energy-neutral computing

Energy-neutral systems cope with an unstable power supply by storing energy in a battery, or a supercapacitor, when harvesting conditions are good [3]. During periods of scarce energy, the stored energy sustains operation until conditions improve.

Fundamentally, energy-neutral operation strives to balance energy consumption and harvesting over a long period T, related to the periodicity of the ambient source [4]:

$$\int_{(n-1)\cdot T}^{n\cdot T} P_h(t)\,dt = \int_{(n-1)\cdot T}^{n\cdot T} P_c(t)\,dt, \tag{2.1}$$

where $P_h(t)$ and $P_c(t)$ is the power harvested and consumed, respectively, at time t.

As an example, an outdoor solar-powered device may have a period of $T = 24$ hours, and thus can satisfy (2.1) by ensuring that it's energy consumption matches the harvested energy over the past 24 hours.

Modulating the average power consumption over a long period is typically done through duty cycling or other means of exploiting power-QoS trade-offs. For example, a node can wake up from sleep after a period of duration p, quickly take sensor measurements, perform its computational

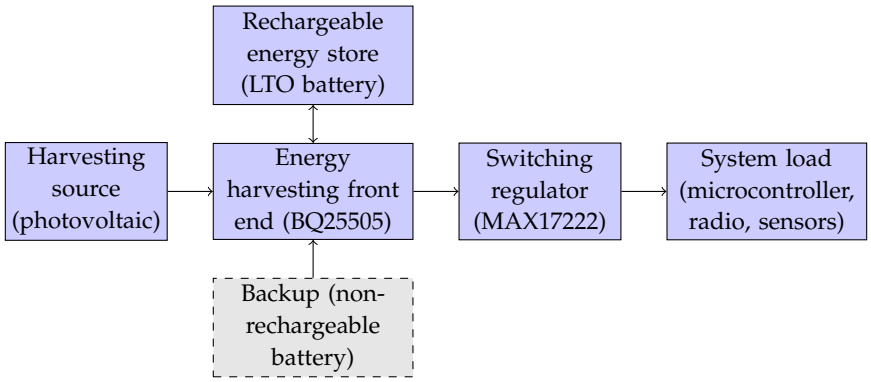

Figure 2: Permamote, a recent example of an EH sensor node designed for energy-neutral operation. Adapted from [20].

tasks, and go back to sleep again. The period p can then be adjusted according to harvesting conditions.

A representative and recent example of a wireless sensor node that can be operated in energy-neutral mode is Permamote [20], shown in Fig. 2. An energy harvesting front-end that includes maximum power point tracking (MPPT), charging and battery monitoring circuitry, selects between a rechargeable energy store and a non-rechargeable battery (used when the rechargeable energy store is depleted) to power the system. A switching regulator then boosts the battery voltage for the system load (microcontroller, radio and sensors). The backup battery used in Permamote is superfluous when successfully operating in energy-neutral mode, but since EH is so unpredictable (both in long-term and short-term temporally, in addition to spatially), it is added as a backup to provide a minimum lifetime where the harvesting conditions are worse than those expected at design time.

Although energy-neutral systems such as Permamote can provide high reliability and QoS with potentially decades of deployment lifetime, the extra conversion and storage circuitry adds cost, mass, volume and complexity; all of which can be prohibitive for deployment at scale. Additionally, the energy loss through the conversion circuitry degrades energy efficiency: the simulator for Permamote used 80% in each of the two conversion steps (Fig. 2), leading to an overall energy efficiency of 64%, before accounting for leakage currents and energy spent in sleep-modes [20,30]. Greater end-to-end efficiency is possible for specialised circuits, e.g. 82% for a reciprocal converter with selective direct operation that tailors to the specific characteristics of subthreshold CMOS processors and tiny energy harvesters [31].

(b) Power-neutral computing

Instead of relying on energy storage and conversion to smooth out an unstable source of power, power-neutral operation aims to instantaneously match power consumption to harvested power so that the system load can be connected directly to the energy harvester [4]. Theoretically, an ideal power-neutral device would scale power consumption such that

$$P_c(t) = P_h(t), \tag{2.2}$$

and thus, no energy storage would be necessary. Power-neutral operation thus promises to both reduce complexity and improve energy efficiency by removing the storage element and the associated charging, monitoring and conversion circuitry. Modulating the power consumption is typically done through dynamic performance scaling (e.g. dynamic voltage and/or frequency scaling) [4,27–29], although it could potentially also employ other techniques such as approximate computing.

However, real computational devices have fixed operating points in terms of power and performance, and switching between them takes time, so real power-neutral devices still have to obey (2.1), albeit over a much smaller period T (typically on the order of milliseconds). Therefore, some capacitance is needed to give the system sufficient time to react, and switch to another operating point, when the power level changes. The necessary capacitance is dependent on the granularity of operating points, and the latency to switch between them. Typically, in microcontroller based systems, the capacitance that is already present for other purposes, such as decoupling, suffices; for more high-performance systems, a moderate amount of capacitance becomes necessary. In Momentum [28], power-neutral operation was demonstrated on a microcontroller based system with no extra capacitance, and on a multiprocessor single-board computer with an additional $47\ mF$ capacitor.

A limitation of power-neutral operation becomes apparent if temporal application requirements vary independently of the harvested power. In this case, the application may have high performance demands in a period where harvested power is low, and so QoS can be severely degraded. Similarly, if the harvested power is ample in a period with low application demands, a large fraction of the harvested power becomes superfluous. This stands in contrast to energy-neutral systems, where harvested power and application demands are temporally decoupled.

3. Intermittent computing systems

Both energy- and power-neutral systems can survive through moderately long periods (relative to the size of their energy buffer) of scarce power-availability by utilizing low-power sleep modes. But, if the harvested power remains lower than the current consumption in their lowest-power sleep mode for an extended period, they will eventually brown out. Consequently, the volatile state is lost, and the system has to recharge, reboot, and then restart the application from the beginning; potentially losing important data, or worse: leaving a corrupt state in nonvolatile memory.

Intermittent computing (IC) solves this problem by retaining computational progress through power cycles, thus allowing long-running applications to progress incrementally whenever energy is available. IC is typically used stand-alone or in conjunction with power-neutral operation [28,29].

This section begins by defining the conditions under which an application is amenable to intermittent operation, then presents consistency requirements, performance objectives, and scalability objectives. Then, a taxonomy divides published works into three classes of IC and evaluates them against the aforementioned requirements and objectives.

(a) Conditions for intermittent operation

Because intermittent operation does not, by definition, sustain continuous operation, an application that is to be executed intermittently must:

- **I1** not depend on absolute time, unless it can be gathered from an external source (e.g. via wireless communication);
- **I2** be guaranteed sufficient power during events to be detected, or permit some lost events;
- **I3** be delay-insensitive or amenable to low-precision computing during periods of scarce energy, or accept lost data points due to a lack of processing power.

A typical application of an ICS is to sense the environment, detect events, and report them to a central sink/server [32,33]. Without a consistent source of power, an ICS cannot maintain an absolute sense of time; it can only keep track of time within an on-period. Note that all three conditions typically entail unreliable and/or reduced QoS when compared to a battery-powered or energy-neutral system.

Recent works have proposed methods that can maintain an approximate sense of time through short off-periods by measuring the decay of a charged capacitor or the gradual information loss in volatile memory [34], but they are far too inaccurate for synchronizing wireless communication (3.7-8.7% error over a period of 35 seconds). Wireless communication between ICSs is, therefore, a major challenge, and still an unsolved problem, so today's ICSs generally communicate with an always-on server. Thus, the central server only needs to be notified when an event happens, and can itself annotate absolute timing. An ideal example is the wireless bicycle trip counter by Bing et al. [22], which counts and reports the revolutions of a bicycle wheel. The device itself is powered by the current induced when a magnet attached to the bicycle wheel passes a coil mounted on the bicycle's fork, hence the event to be detected directly coincides with the power source. By use of intermittent operation, their sensor node is made with a minimum of components, leading to small size and cost without introducing maintenance requirements or sacrificing lifetime. Other potentially suitable applications for ICSs include predictive maintenance on industrial equipment via vibrational and acoustical monitoring [35]; energy for the system could be supplied by a piezoelectric vibration harvester.

(b) Correctness requirements, performance objectives and scalability objectives

For systematic evaluation of approaches to intermittent operation, this paper will use the following correctness requirements (**C1**-**C4**), performance objectives (**P1**-**P3**), and scalability objectives (**S1**-**S3**). Correctness requirements must be met for an ICS to function as expected. Performance and scalability objectives are ideals that should be sought after, but are not critical —and will inevitably lead to compromises. The requirements and objectives are inspired by the ones introduced in Alpaca [16], but have been generalised and expanded in order to include recent progress and insights in the field [32,33,36–39].

For correctness, an ICS *must*:

C1 retain computational progress through power loss;
C2 ensure consistency between volatile and nonvolatile memory despite power loss;
C3 allow blocks of code to be executed atomically within a single on-period;
C4 avoid sisyphean tasks.

For performance, i.e. to maximise QoS, an ICS *should*:

P1 maximise forward progress through applications;
P2 be reactive to external events;
P3 minimise extra memory footprint imposed by ICS framework.

For scalability and to accommodate widespread adoption, an ICS *should*:

S1 require minimal ICS-specific expertise from the programmer;
S2 be compatible with existing software;
S3 be portable across hardware platforms.

C1 is the base premise of intermittent operation. In certain situations, arbitrary sections of code can be re-executed because the system has to restore to an earlier state after a power failure. If the re-executed code is not idempotent[4], re-execution can lead to a violation of **C2** [14]; i.e. an inconsistency between volatile memory, which loses its state when power fails, and nonvolatile memory, which does not. **C3** enables the programmer to specify sections of code which must be executed atomically[5] in the same on-period. This is necessary for operations that must be restarted entirely if interrupted by a power failure (of unknown duration). Examples include sampling

[4]In this context, a section of code is idempotent if it can be interrupted at any point, restarted from the beginning, and still produce the same result.

[5]In this context, a function which is executed atomically will be executed in its entirety without power interruptions. If it is interrupted, it must be restarted from the beginning.

temporally correlated sensors, using on-chip peripherals, serial communication with peripherals, and transmitting radio packets [33,36,38].

A *Sisyphean task* is a task that requires more energy than the device can muster, hence it will be restarted indefinitely, rendering the ICS non-functional [14]. The requirement to avoid sisyphean tasks (**C4**) follows from **C3**: if certain tasks are to be executed atomically, it is imperative that they can be completed within a power cycle.

P1 implies that an ICS should minimise the time spent on state retention, boot, and rollback. To remain reactive to external events (**P2**), an ICS should be able to handle asynchronous interrupts, and should generally wake up frequently so as to not miss an event while recharging for the next on-period. Note that **P1** and **P2** often conflict with each other. To optimise for **P1** means that reboots are rare, so that boot and rollback overheads are amortised over a long period of application-execution. On the contrary, optimising for **P2** means frequent reboots to ensure that no events are missed. In addition to time and energy overheads, IC methods generally impose extra memory footprint to, for example, store snapshots of system state, or to instrument application code with checkpoints; this extra footprint should be minimised, because the ICSs typically have strict memory constraints (**P3**).

(c) Taxonomy

In order to differentiate and evaluate existing works, this section presents a taxonomy based on the following three classes of intermittent computing methods [40]:

Static IC: where snapshots are saved at predetermined (design-time or compile-time) checkpoints in the program.

Task-based IC: where the application is divided into small tasks that are executed atomically by a runtime.

Reactive IC: where snapshots are saved when a power failure is detected.

The fundamental divergence is between static and reactive IC; task-based IC can be regarded as a recent development of static IC, where checkpoints are replaced by task boundaries. Static and reactive IC typically retain progress through power cycles by periodically (static) or reactively (reactive) saving the volatile state as a snapshot in nonvolatile memory. Task-based IC needs, in principle, only to save the result from a task's execution and an indicator to the next task to be completed.

State diagrams for the three classes are shown in Fig. 3; these diagrams show the fundamental states, although most methods diverge slightly —typically by having additional states and edges to improve performance. A key insight is that both static and task-based methods have power-off edges (red) from all states, meaning that the system could potentially instantaneously shut down at any point in the program/runtime. Reactive IC (Fig. 3c), on the other hand, ensures that the system only powers off after reaching a safe sleep state.

Figure 4 illustrates the behaviour of each class in relation to the power supply voltage. Static IC restores (R) as soon as the supply voltage, v_{cc}, exceeds the on-threshold of the processor, V_{on}, and starts useful computation (C) while checkpointing (CP) state according to heuristics, for example every few milliseconds. Task-based IC also restores (boot runtime and restart the latest task) as soon as $v_{cc} > V_{on}$. The *restore* operation is typically quick because only the runtime and a single task need to be booted, not the entire program. When complete, the results of the current tasks are saved, and the next task is booted and started; this is termed a task-transition (TT). Reactive IC sleeps until an interrupt triggers *restore*, when v_{cc} exceeds the restore threshold V_R. Reactive IC does not take checkpoints along the way, but rather continues useful computation until v_{cc} drops below V_S, the suspend threshold, which triggers *suspend* (S), saving a snapshot just before power is lost. In the literature, this is also referred to as Just-In-Time checkpointing [33]. The circles show the latest snapshot before power is lost; any computation after the latest snapshot is a waste of resources. Note that some task-based or static approaches may mitigate wasted computation after a checkpoint by measuring stored energy and comparing it to predictions [21] or observations [41] of the energy required to reach the next task-boundary or checkpoint.

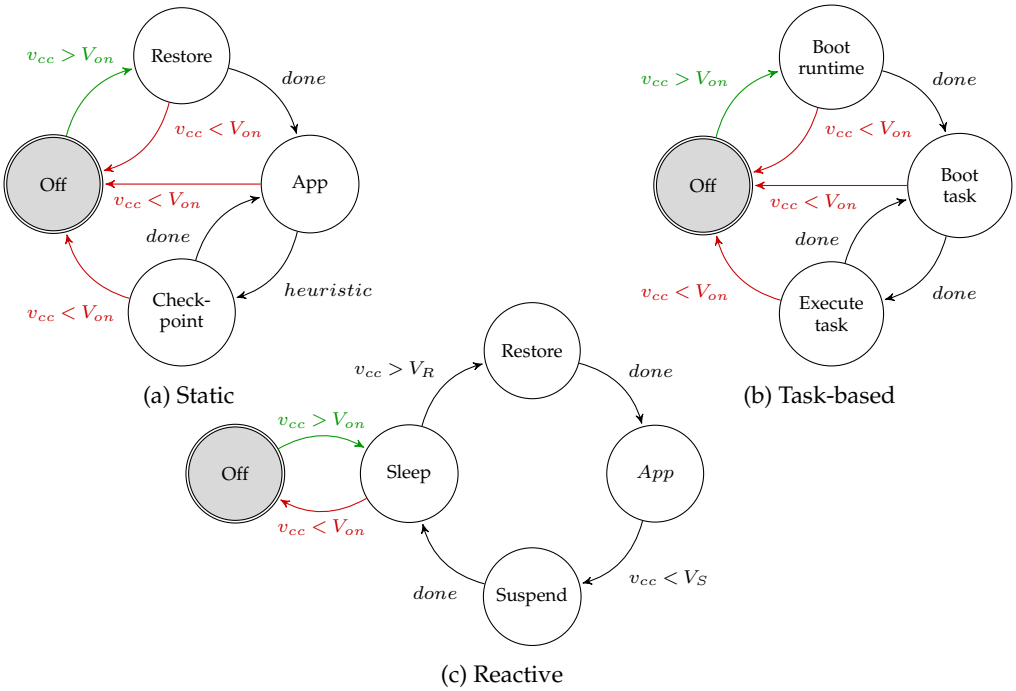

Figure 3: State machine representation of (a) static, (b) task-based and (c) reactive IC. Reactive IC has simpler state transitions, most importantly avoiding power-off edges (red) from any states where idempotency violations can occur.

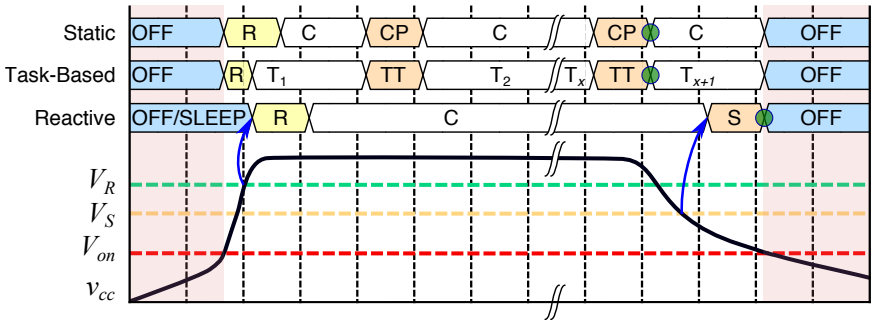

Figure 4: Behaviour of the three classes of IC strategies with regards to v_{cc}.

Because static and task-based methods place checkpoints or task-boundaries a priori, they risk encountering a *sisyphean task* [42]. This is where the section between checkpoints or task-boundaries is too large to be completed within a single power cycle, and thus it gets restarted forever. Reactive IC can typically avoid sisyphean tasks by proper use of voltage thresholds and sufficient capacitance (see Section 3(c)iii).

The following subsections analyse the three classes of IC in the context of the aforementioned correctness requirements, performance objectives and scalability objectives. Table 2 summarises our findings.

Table 2: Summarised analysis of the three classes of intermittent computing.

Objective/Requirement	Static IC	Task-based IC	Reactive IC
C1: Comp. progress	✓	✓	✓
C2: Consistent memory	Challenging	Challenging	✓
C3: Atomic sections	✓	✓	Challenging
C4: Avoid sisyph. tasks	✗	✗	✓
P1: Forward progress	Slower[1]	Slower[1]	Faster[1]
P2: Reactive to events	Challenging	Challenging	✓
P3: Memory overhead	Instr.[2], checkpoint	Instr.[2], runtime, logs, tiny checkpoint	Checkpoint
S1: Required ICS expertice	Moderate	Major	Minor
S2: Software compatibility	✓[3]	✗	✓
S3: Hardware portable	✗	✗	✓

[1]Depends on specific implementation. [2]Code instrumentation. [3]With full recompilation.

(i) Static IC

Static IC approaches are based on instrumenting application code with checkpoints by the user during application development, or automatically at compile time [21,42–44]. Inserting checkpoints at compile time can alleviate the burden of porting existing software to run intermittently, if source code is available (**S2**). Because the checkpoints are inserted a priori, offline analysis of program execution and control flow graphs can be applied to guarantee memory consistency (**C2**) and improve performance (**P1**-**P3**) [21,43,45,46]. However, static analysis is greatly complicated if asynchronous interrupts are involved (**P2**).

The main challenges to static IC are in the placement of checkpoints: placing them too far apart minimises superfluous checkpoints, but makes it unlikely or even impossible to reach the next checkpoint (**C4**); placing them too densely wastes time and energy on superfluous checkpoints (**P1**). The optimal checkpoint placement depends both on hardware and on harvesting conditions, making solutions unlikely to be portable across hardware platforms (**S3**), and requiring a moderate degree of ICS expertise (**S1**).

Static IC also suffers from frequent code re-execution because the state rolls back to the latest snapshot when power returns after a power failure (the computation after the circle in Fig. 4 is re-executed in the next power cycle). Code re-execution is a waste of energy (**P1**), and can corrupt memory if the re-executed section of code is not idempotent (**C2**) [14].

(ii) Task-based IC

Task-based IC is a recent development of static IC where the application is divided into a set of small tasks that are executed atomically by a runtime [15–17,41]. This development is motivated by protecting static IC against idempotency violations, by carefully controlling or recording each task's accesses to nonvolatile memory, thus making code re-execution safe (**C2**) [15]. By definition, task-based systems handle atomic regions correctly and efficiently (**C3**. Additionally, systems like Alpaca [16] offer fast reboot and state-saving because only the runtime and the current task is booted, and only persistent data from the task needs to be saved (**P1**). However, in addition to re-execution overheads, handling idempotency is expensive [33]; Alpaca, an efficient task-based method reports $1.3 - 3.6\times$ overhead when continuously powered (**P1**) [16].

In addition to technical challenges, such as

- expensive and complicated roll-back schemes necessary for protection against idempotency violations (**C2**),
- large memory footprint (**P3**),
- expensive task-transitions (**P1**), and
- handling interrupts to react to asynchronous events (**P2**),

task-based systems struggle to meet any of the three scalability objectives. The imposed programming model is not compatible with existing software (**S2**), as it requires redesigning applications into tasks. Finding optimal task boundaries (analogous to checkpoint placement in static IC) is difficult, and depends on both the specific underlying hardware and the energy harvesting conditions; great care must be taken to avoid sisyphean tasks (**C4**). Thus designing a task-based application generally requires a high degree of IC-specific expertise (**S1**) [39]. Furthermore, since optimal task-boundaries are hardware-specific, applications are not portable across hardware platforms (**S3**).

CleanCut [39] is a recent method that checks for sisyphean tasks (termed "non-terminating path bugs" in the paper) and performs automatic task decomposition. The method is a significant step towards making task-based methods scalable (**S1**-**S3**). However, limitations of the method, such as the requirement that the programmer must specify an iteration bound for each unbounded loop in the program, restrict its utility for applications that depend on existing libraries (**S2**).

Classical task-based approaches, such as Alpaca [16], are incompatible with asynchronous interrupts because they would break the premise of an application divided into atomically executed tasks. Ink [36] aims to resolve this by use of task-threads and scheduling. However, tasks in Ink cannot be pre-empted, making reaction time slow and unpredictable. A more recent alternative is Coati, which "employs a split-phase model that handles time-critical I/O immediately in a brief interrupt handler, but defers processing the interrupt's results until after the interrupted task or transaction completes" [47] thus accomplishes real-time response to interrupts.

(iii) Reactive IC

Contrary to static and task-based approaches, reactive IC is mostly implemented in an application agnostic manner [18,19,48], greatly improving **S1** and **S2** when compared to typical task-based or static methods. Reactive IC consists of the following two operations:

> **Suspend:** Save a snapshot of volatile state to nonvolatile memory, and enter low power mode or shut down.
> **Restore:** Restore volatile state from a snapshot.

The set of volatile state is architecture and platform specific. However, reactive IC remains portable across hardware (**S3**), because only the *suspend* and *restore* functions need be ported for each platform, not the entire application.

Power supply monitoring is set up to generate interrupts that trigger *suspend* and *restore* operations when the supply voltage, v_{cc}, crosses a threshold. When the restore threshold is exceeded, an interrupt triggers *restore*, which restores state from a previous snapshot. Similarly, when v_{cc} drops below the suspend threshold, V_S, a snapshot is saved and the system enters low-power mode (where volatile state is retained) or shuts down. If power returns while the system is still in low-power mode, *restore* is unnecessary because the volatile state was retained, so the application can continue execution directly, thus improving **P1** [19]. When powered by a low-current source, *suspend* can often be avoided altogether by pre-emptively entering sleep mode before v_{cc} drops below V_S [49]; this can, in some cases, improve **P1**.

The suspend threshold ensures that snapshots are only saved when power failure is imminent, thus nearly eliminating superfluous save operations (**P1**) [18,33,50]. The restore threshold is used to guarantee the success of the next checkpoint; it eliminates code re-execution, and thus also idempotency violations (**C2**). Determining such a V_R is intractable for static and task-based IC because all possible execution paths must be exhaustively analysed. In reactive IC, V_R can be determined by online measurement [19], or calculated based on the amount of state to *suspend* and *restore* [50]. Note that to guarantee the success of the next checkpoint, a minimum amount of energy buffering is required. The energy buffer must suffice to power *suspend*, but additional capacity can improve performance.

With proper, pre-emptive, interrupt prioritization, reactive IC is natively reactive to asynchronous external events (**P2**), and furthermore can guarantee forward progress in application code (assuming that any atomic functions used are sufficiently small).

The main drawback of reactive IC is that saving and restoring the entire state is expensive compared to task-based systems where only one task is loaded/saved at a time. This is especially wasteful if the power cycle is short, because only a small part of the program is expected to execute. This problem can be partially alleviated by techniques such as: comparing the current state to the latest snapshot, and only saving the difference [51,52]; or IC-specific memory management [50]. Another drawback is that reactive IC does not generally handle high-level atomicity constraints (**C3**) as gracefully as task-based or static methods. A solution to this is proposed in [53], where a formal division between application code (which accesses only application memory) and driver code (which accesses peripherals and nonvolatile memory) assists a kernel to ensure atomic execution of driver-functions without introducing idempotency violations (**C2**). But adding atomicity to a reactive system can introduce sisyphean tasks (**C4**). Samoyed [33] handles sisyphean tasks by profiling all atomic tasks in hardware, and, for suitable peripherals/operations, dynamically breaks down the task into parts that are small enough to execute in a single on-period. Not all peripheral operations can be divided into smaller parts, but as shown in the paper, this method is very effective for memory mapped accelerators.

4. Unsolved challenges in energy-driven computing

Energy-driven computing brings with it unique challenges that slow down adoption in the real world. While there have been a plethora of applications for battery-powered devices in the IoT literature, energy-driven devices are still novel. Demonstrations of real world applications are starting to emerge for energy-neutral systems, e.g. a self-powered wireless sensor node for predictive maintenance [54]. However, few realistic applications have been demonstrated using intermittent or power-neutral operation. Developing applications for power-neutral and/or intermittent computers is a challenge, requiring new thinking throughout the stack: from the design of circuits for power management, selection and integration of energy harvesters, software, and the entire communications stack. On the topic of Energy-Driven Computers, there are many research challenges yet to be solved. We highlight what we regard as the main remaining challenges that are currently hindering adoption: development tools for energy-driven computers, and wireless communication under intermittent operation.

(a) Simulation, emulation and debugging

Simulation and debugging of energy-driven computers is inherently more complex than for traditional application-driven ones. This stems from the fact that energy-driven computers adapt their behaviour according to the amount of available energy. Traditionally, simulations are performed under the assumption of a steady power supply, so emulating the ISA and platform is usually sufficient for finding bugs, evaluating performance and developing applications. However, to correctly simulate the behaviour of an energy-driven computer, execution, power consumption and power production (harvested energy) must be simulated simultaneously to correctly model their influence on each other. While previous works have explored simulation of energy-driven computers, their solutions target only specific processors, and they replay voltage traces rather than model the complex dynamics of a real energy harvester [55].

Another option is to record current-voltage traces for real energy harvesters, then replay them with a power-synthesizer to produce a repeatable power source [56,57]. While useful for performance evaluation, this method does not provide the introspection capabilities of simulation, so is only moderately useful for validation/debugging.

Interactive debuggers like the ubiquitous GNU debugger (GDB) typically control the execution and state of a target processor via a JTAG-like interface, providing invaluable introspection into program and device behaviour. Key to their utility is the ability to halt and single-step a program.

However, in an energy-driven system, halting the processor is insufficient; ideally one should be able to halt the energy dynamics too. EDB (Energy-interference-free Debugging) [58] provides the ability to both monitor and control the processor's execution simultaneously with a platform's energy buffer. This enables new debugging primitives such as energy-breakpoints. However, this does not "pause" the power generation (energy harvester). Combining the works in recording and replaying EH dynamics [56,57] with EDB [58] could be a promising direction for a holistic debugging environment for ICS.

(b) Wireless communication under intermittent operation

Due to the nature of sources that ICSs rely on, which can be highly unpredictable and fluctuating (both temporally and spatially), coordinating a number of intermittently-powered, distributed devices/systems is an ongoing challenge. Thus, the state-of-the-art intermittent networks mostly embody mains- or battery-powered always-on sinks for data collection and coordination. This is regarded as the most convenient/reliable way for *uplink communications* through these networks. Here, the always-on sink collects data from the ICSs in a single-hop manner and establishes a gateway to the Internet, whereby the ICS data become accessible from anywhere at any time. This centralized structure, i.e. star topology, helps to alleviate the uncertainty occurred due to non-deterministic on/off periods, enabling ICSs with active radios having enough energy to freely transmit their data by knowing that the sink is constantly listening to the channel for data reception. These systems are often not acknowledged, which means that a collision could go unnoticed. Yet, depending on the parameters of interest, confirmation of reception is highly required, especially for mission- or information-critical applications, to guarantee reliable detection of events and/or reporting of vital parameters. However, this is hard to accomplish in intermittent operation, after uplink data transfer more specifically, where traditional medium access control (MAC) layer solutions have to be revisited.

Downlink communications, i.e. from sink to ICSs, on the other hand, is more problematic as neither the ICSs know when they should listen nor the sink anticipates when an ICS becomes active and/or is ready to receive. In practice, the ICSs can perform listening for a very short duration only, mostly in a random fashion, due to their limited energy buffer. Thus, precise timing, i.e. synchronisation, between them and the sink is a challenging issue. To avoid loss of important data and so retransmissions, the ICSs go to sleep after computation and wait for an interrupt from the sink while recharging their buffer. For this particular reason, intermittent operation needs competent and efficient wake-up mechanisms.

The very basic way of waking up an ICS is with wireless power transfer (WPT) from a sink [59]. An ICS with a rectifying antenna (rectenna) can trigger an interrupt by using the intercepted RF signals radiated from the sink to wake up and get ready to receive. By following the power signal, the sink can transmit data to the ICS that is now active and listening. This operation is costly for the sink as it has to transmit twice, once for wake-up and once again for communication, which also decreases bandwidth utilization efficiency. After reception of data, the ICS acknowledges the sink. Alternatively, instead of sending the control frame (acknowledgement) back to the sink immediately, it can be hooked onto the next transmission (which is known as *piggybacking*); however, this does incur a delay [60]. Despite simplicity, rectennas are prone to false wake-ups since any signal with a matching frequency, not originating from the sink of interest, may unintentionally wake the ICS up.

For more energy-constrained applications, passive radios, with significantly lower power consumption than the active radios, can be utilized. In the very well-known method of passive radio communications called backscattering [61], the ICSs with passive radios, i.e. tags, use the same signal directed from a centralized carrier generator to reflect their data back by changing their antenna impedance. Although backscattering enables small form factors, low-cost and low-power solutions, backscattering-backed communications suffer from frequent power interrupts, collisions, low data rates, and short communication ranges. The majority of existing research

efforts on this field is on uplink communications; however, there are some recent studies aiming at speeding up the downlink data transfer [62,63].

Apart from environment- and physical layer-related matters, the fundamental factor limiting the widespread adoption of ICSs is synchronization, which severely restricts peer-to-peer communications. This can be only achieved when both ends are circumstantially active for a long enough period, at the same time. Thus, a power-failure-resilient timekeeping solution is evident [64]. Once the challenge of establishing peer-to-peer communication in ICSs is overcome, and hence ICSs can be considered as distributed systems, a wealth of further challenges emerge –for example how to coordinate, schedule and manage energy across a distributed and federated system. These will be followed by debugging, simulation, and optimization of the networked ICSs. Due to the very same reason, i.e. lack of synchronization/timekeeping, the ICSs currently does not support mesh topology and multi-hop communications, i.e. routing protocols, which need further investigation. Moreover, the existing protocol stack, including cross-layer communication solutions, has to be redesigned to allow intermittent operation to reach its full potential.

5. Discussion

Energy-driven computers are gaining traction as a solution for the demands of IoT end devices. While battery-powered devices are suitable for many applications, their deployment lifetimes are limited by their fixed capacity. Energy-harvesters can be used to extend the deployment lifetime of devices, but bring unique challenges due to the dynamics of their power output. This motivates a shift into energy-driven computing, where energy-availability is considered from the beginning of the design process. In this paper, we described three prominent examples of energy-driven computing: energy-neutral, power-neutral and intermittent operation.

Energy-neutral operation augments a rechargeable battery with an energy harvester that supplies energy during deployment; thus lifetime is extended without increasing battery size. However, the charging circuitry imposes electrical losses and the rechargeable batteries are still a limiting factor in terms of size, cost, environmental impact, and their limited charge-cycles limit deployment lifetime. Power-neutral computers seek to operate without rechargeable batteries, using only a tiny amount of capacitance for buffering, to simplify the circuit and thus substantially reduce cost, size and environmental impact while simultaneously improving lifetime. However, power-neutral computers cannot sustain operation if harvested power drops too low, so long-running applications become infeasible. Intermittent operation solves this problem by retaining computational progress through power cycles; they do not attempt to hide the dynamics of energy harvesting, but rather accept that the system will shut down frequently and unpredictably. However, this imposes new challenges, including losing track of time, and ultimately reducing achievable QoS for most applications.

Development of energy-driven computers is complicated because the developer has to reason about application requirements, energy demands, and energy availability simultaneously. Current development tools lack the ability to sufficiently control both the energy-availability and execution simultaneously, hindering the adoption of energy-driven computers.

Wireless communication is tremendously important for achieving a scalable IoT and enabling a multitude of sensing applications. However, this necessitates a reliable energy supply and a precise notion of time; energy-driven computers are severely constrained on energy, and intermittent computing systems lack a sense of time. These require novel physical layer solutions to cope with the non-deterministic factors in both applications achieving sufficient QoS in uplink/downlink communications. In addition, lightweight synchronization and power-failure-resilient timekeeping mechanisms are of utmost importance. Directions for future research into this area include establishing peer-to-peer communications, efficient routing protocols and application-specific network topologies.

Data Accessibility. No data beyond those explicitly stated in this article were generated for this study.

Competing Interests. The authors declare that they have no competing interests.

Funding. This work was supported by the Engineering and Physical Sciences Research Council (EPSRC) under an iCASE award and grant EP/P010164/1.

References

1. Atzori L, Iera A, Morabito G. 2010 The Internet of Things: A survey.
Computer Networks **54**, 2787–2805.
doi: 10.1016/j.comnet.2010.05.010
2. Ahmed E, Yaqoob I, Gani A, Imran M, Guizani M. 2016 Internet-of-things-based smart environments: State of the art, taxonomy, and open research challenges.
IEEE Wireless Communications **23**, 10–16.
doi: 10.1109/MWC.2016.7721736
3. Kansal A, Hsu J, Zahedi S, Srivastava MB. 2007 Power management in energy harvesting sensor networks.
ACM Transactions on Embedded Computing Systems **6**, 32–es.
doi: 10.1145/1274858.1274870
4. Merrett GV, Al-Hashimi BM. 2017 Energy-driven computing: Rethinking the design of energy harvesting systems.
In *Design, Automation and Test in Europe Conference and Exhibition (Date), 2017*, pp. 960–965. Design, Automation Test in Europe Conference Exhibition (DATE), 2017.
doi: 10.23919/DATE.2017.7927130
5. Gros I, Popa D, Teodosescu PD, Radulescu MM. 2017 A survey on green energy harvesting applications using linear electric generators.
In *2017 International Conference on Modern Power Systems (MPS)*, pp. 1–5.
doi: 10.1109/MPS.2017.7974388
6. Cetinkaya O, Akan OB. 2017 Electric-field energy harvesting in wireless networks.
IEEE Wireless Communications **24**, 34–41.
doi: 10.1109/MWC.2017.1600215
7. Cetinkaya O, Akan OB. 2017 Electric-Field Energy Harvesting From Lighting Elements for Battery-Less Internet of Things.
IEEE Access **5**, 7423–7434.
doi: 10.1109/ACCESS.2017.2690968
8. Ku M, Li W, Chen Y, Liu KJR. Secondquarter 2016 Advances in energy harvesting communications: Past, present, and future challenges.
IEEE Communications Surveys Tutorials **18**, 1384–1412.
doi: 10.1109/COMST.2015.2497324
9. Yildiz F. 2009/00/00 Potential Ambient Energy-Harvesting Sources and Techniques.
Journal of Technology Studies **35**, 40–48.
10. Akan OB, Cetinkaya O, Koca C, Ozger M. 2018 Internet of Hybrid Energy Harvesting Things.
IEEE Internet of Things Journal **5**, 736–746.
doi: 10.1109/JIOT.2017.2742663
11. Akhtar F, Rehmani MH. 2017 Energy Harvesting for Self-Sustainable Wireless Body Area Networks.
IT Professional **19**, 32–40.
doi: 10.1109/MITP.2017.34
12. Núñez CG, Manjakkal L, Dahiya R. 2019 Energy autonomous electronic skin.
npj Flex Electron **3**, 1–24.
doi: 10.1038/s41528-018-0045-x
13. Shaikh FK, Zeadally S. 2016 Energy harvesting in wireless sensor networks: A comprehensive review.
Renewable and Sustainable Energy Reviews **55**, 1041–1054.
doi: 10.1016/j.rser.2015.11.010
14. Ransford B, Lucia B. 2014 Nonvolatile Memory is a Broken Time Machine.
In *Proceedings of the Workshop on Memory Systems Performance and Correctness*, MSPC '14, pp. 5:1–5:3. New York, NY, USA: ACM.
doi: 10.1145/2618128.2618136

15. Colin A, Lucia B. 2016 Chain: Tasks and channels for reliable intermittent programs.
In *Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, OOPSLA 2016, pp. 514–530. New York, NY, USA: ACM.
doi: 10.1145/2983990.2983995
16. Maeng K, Colin A, Lucia B. 2017 Alpaca: Intermittent execution without checkpoints.
Proc. ACM Program. Lang. **1**, 96:1–96:30.
doi: 10.1145/3133920
17. Durmaz C, Yildirim KS, Kardas G. 2019 PureMEM: A Structured Programming Model for Transiently Powered Computers.
In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, SAC '19, pp. 1544–1551. New York, NY, USA: ACM.
doi: 10.1145/3297280.3299739
18. Balsamo D, Weddell AS, Merrett GV, Al-Hashimi BM, Brunelli D, Benini L. 2015 Hibernus: Sustaining computation during intermittent supply for energy-harvesting systems.
IEEE Embedded Systems Letters **7**, 15–18.
doi: 10.1109/LES.2014.2371494
19. Balsamo D, Das A, Weddell AS, Brunelli D, Al-Hashimi BM, Merrett GV, Benini L. 2016 Graceful Performance Modulation for Power-Neutral Transient Computing Systems.
IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **35**, 738–749.
doi: 10.1109/TCAD.2016.2527713
20. Jackson N, Adkins J, Dutta P. 2019 Capacity over Capacitance for Reliable Energy Harvesting Sensors.
In *Proceedings of the 18th International Conference on Information Processing in Sensor Networks*, IPSN '19, pp. 193–204. New York, NY, USA: ACM.
doi: 10.1145/3302506.3310400
21. Bhatti NA, Mottola L. 2017 Harvos: Efficient code instrumentation for transiently-powered embedded sensing.
In *2017 16th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, pp. 209–220.
22. Bing SWC, Balsamo D, Merrett GV. 2018 An energy-driven wireless bicycle trip counter with zero energy storage.
In *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*, SenSys '18, pp. 404–405. New York, NY, USA: ACM.
doi: 10.1145/3274783.3275205
23. Hicks M. 2017 Clank: Architectural support for intermittent computation.
In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pp. 228–240.
doi: 10.1145/3079856.3080238
24. Ma K, Zheng Y, Li S, Swaminathan K, Li X, Liu Y, Sampson J, Xie Y, Narayanan V. 2015 Architecture exploration for ambient energy harvesting nonvolatile processors.
In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, pp. 526–537. Burlingame, CA, USA: IEEE.
doi: 10.1109/HPCA.2015.7056060
25. Shafik R, Yakovlev A, Das S. 2018 Real-Power Computing.
IEEE Transactions on Computers **67**, 1445–1461.
doi: 10.1109/TC.2018.2822697
26. Han J, Orshansky M. 2013 Approximate computing: An emerging paradigm for energy-efficient design.
In *2013 18th IEEE European Test Symposium (ETS)*, pp. 1–6.
doi: 10.1109/ETS.2013.6569370
27. Fletcher BJ, Balsamo D, Merrett GV. 2017 Power neutral performance scaling for energy harvesting MP-SoCs.
In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pp. 1516–1521. Lausanne, Switzerland: IEEE.
doi: 10.23919/DATE.2017.7927231
28. Balsamo D, Fletcher BJ, Weddell AS, Karatziolas G, Al-Hashimi BM, Merrett GV. 2019 Momentum: Power-neutral performance scaling with intrinsic mppt for energy harvesting

computing systems.
ACM Transactions on Embedded Computing Systems (TECS) **17**, 93.
doi: 10.1145/3281300

29. Balsamo D, Weddell AS, Das A, Arreola AR, Brunelli D, Al-Hashimi BM, Merrett GV, Benini L. 2016 Hibernus++: A self-calibrating and adaptive system for transiently-powered embedded devices.
IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **35**, 1968–1980.
doi: 10.1109/TCAD.2016.2547919
30. Jackson N. 2019, Permamote git repository.
31. Savanth A, Weddell AS, Myers J, Flynn D, Al-Hashimi BM. 2017 Integrated reciprocal conversion with selective direct operation for energy harvesting systems.
IEEE Transactions on Circuits and Systems I: Regular Papers **64**, 2370–2379.
doi: 10.1109/TCSI.2017.2707304
32. Colin A, Ruppel E, Lucia B. 2018 A reconfigurable energy storage architecture for energy-harvesting devices.
In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '18, pp. 767–781. New York, NY, USA: ACM.
doi: 10.1145/3173162.3173210
33. Maeng K, Lucia B. 2019 Supporting Peripherals in Intermittent Systems with Just-in-time Checkpoints.
In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2019, pp. 1101–1116. New York, NY, USA: ACM.
doi: 10.1145/3314221.3314613
34. Hester J, Tobias N, Rahmati A, Sitanayah L, Holcomb D, Fu K, Burleson WP, Sorber J. 2016 Persistent Clocks for Batteryless Sensing Devices.
ACM Transactions on Embedded Computing Systems **15**, 1–28.
doi: 10.1145/2903140
35. Wang D, Tsui K, Miao Q. 2018 Prognostics and Health Management: A Review of Vibration Based Bearing and Gear Health Indicators.
IEEE Access **6**, 665–676.
doi: 10.1109/ACCESS.2017.2774261
36. Yıldırım KS, Majid AY, Patoukas D, Schaper K, Pawelczak P, Hester J. 2018 Ink: Reactive kernel for tiny batteryless sensors.
In *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*, SenSys '18, pp. 41–53. New York, NY, USA: ACM.
doi: 10.1145/3274783.3274837
37. Hester J, Sitanayah L, Sorber J. 2015 Tragedy of the Coulombs: Federating Energy Storage for Tiny, Intermittently-Powered Sensors.
In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems - SenSys '15*, pp. 5–16. Seoul, South Korea: ACM Press.
doi: 10.1145/2809695.2809707
38. Hester J, Storer K, Sorber J. 2017 Timely Execution on Intermittently Powered Batteryless Sensors.
In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*, SenSys '17, pp. 17:1–17:13. New York, NY, USA: ACM.
doi: 10.1145/3131672.3131673
39. Colin A, Harvey G, Sample A, Lucia B. 2018 An energy-interference-free hardware/software debugger for intermittent energy-harvesting systems.
IEEE Micro pp. 1–1.
doi: 10.1109/MM.2017.265085515
40. Sliper ST, Balsamo D, Weddell AS, Merrett GV. 2018 Enabling intermittent computing on high-performance out-of-order processors.
In *Proceedings of the 6th International Workshop on Energy Harvesting & Energy-Neutral Sensing Systems*, ENSsys '18, pp. 19–25. New York, NY, USA: ACM.
doi: 10.1145/3279755.3279759
41. Jayakumar H, Raha A, Raghunathan V. 2016 Energy-aware memory mapping for hybrid fram-sram mcus in iot edge devices.

In *2016 29th International Conference on VLSI Design and 2016 15th International Conference on Embedded Systems (VLSID)*, pp. 264–269. Kolkata, India: IEEE.
doi: 10.1109/VLSID.2016.52

42. Ransford B, Sorber J, Fu K. 2011 Mementos: System support for long-running computation on rfid-scale devices.
In *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVI, pp. 159–170. New York, NY, USA: ACM.
doi: 10.1145/1950365.1950386
43. Maeng K, Lucia B. 2018 Adaptive dynamic checkpointing for safe efficient intermittent computing.
In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, pp. 129–144.
44. Woude JVD, Hicks M. 2016 Intermittent computation without hardware support or programmer intervention.
In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pp. 17–32.
45. Zhao M, Li Q, Xie M, Liu Y, Hu J, Xue CJ. 2015 Software Assisted Non-volatile Register Reduction for Energy Harvesting Based Cyber-physical System.
In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, DATE '15, pp. 567–572. San Jose, CA, USA: EDA Consortium.
46. Li Q, Zhao M, Hu J, Liu Y, He Y, Xue CJ. 2015 Compiler directed automatic stack trimming for efficient non-volatile processors.
In *Proceedings of the 52Nd Annual Design Automation Conference*, DAC '15, pp. 183:1–183:6. New York, NY, USA: ACM.
doi: 10.1145/2744769.2744809
47. Ruppel E, Lucia B. 2019 Transactional Concurrency Control for Intermittent, Energy-harvesting Computing Systems.
In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2019, pp. 1085–1100. New York, NY, USA: ACM.
doi: 10.1145/3314221.3314583
48. Jayakumar H, Raha A, Raghunathan V. 2014 QUICKRECALL: A Low Overhead HW/SW Approach for Enabling Computations across Power Cycles in Transiently Powered Computers.
In *2014 27th International Conference on VLSI Design and 2014 13th International Conference on Embedded Systems*, pp. 330–335.
doi: 10.1109/VLSID.2014.63
49. Lukosevicius G, Arreola AR, Weddell AS. 2017 Using Sleep States to Maximize the Active Time of Transient Computing Systems.
In *Proceedings of the Fifth ACM International Workshop on Energy Harvesting and Energy-Neutral Sensing Systems*, ENSsys'17, pp. 31–36. New York, NY, USA: ACM.
doi: 10.1145/3142992.3142998
50. Sliper ST, Balsamo D, Nikoleris N, Wang W, Weddell AS, Merrett GV. 2019 Efficient state retention through paged memory management for reactive transient computing.
In *Proceedings of the 56th Annual Design Automation Conference 2019*, DAC '19, pp. 26:1–26:6. New York, NY, USA: ACM.
doi: 10.1145/3316781.3317812
51. Verykios TD, Balsamo D, Merrett GV. 2018 Selective policies for efficient state retention in transiently-powered embedded systems: Exploiting properties of NVM technologies.
Sustainable Computing: Informatics and Systems .doi: 10.1016/j.suscom.2018.07.003
52. Bhatti NA, Mottola L. 2016 Efficient state retention for transiently-powered embedded sensing.
In *Proceedings of the 2016 International Conference on Embedded Wireless Systems and Networks*, EWSN '16, pp. 137–148. USA: Junction Publishing.
53. Berthou G, Delizy T, Marquet K, Risset T, Salagnac G. 2019 Sytare: A Lightweight Kernel for NVRAM-Based Transiently-Powered Systems.
IEEE Transactions on Computers pp. 1–1.
doi: 10.1109/TC.2018.2889080
54. Magno M, Sigrist L, Gomez A, Cavigelli L, Libri A, Popovici E, Benini L. 2019 SmarTEG: An

Autonomous Wireless Sensor Node for High Accuracy Accelerometer-Based Monitoring. *Sensors* **19**, 2747. doi: 10.3390/s19122747

55. Wu Y, Sun Y, Jia Z, Zhang L, Liu Y, Hu J. 2018 Prototyping Energy Harvesting Powered Systems with Nonvolatile Processor (Invited Paper). In *2018 International Symposium on Rapid System Prototyping (RSP)*, pp. 49–55. doi: 10.1109/RSP.2018.8631991
56. Hester J, Scott T, Sorber J. 2014 Ekho: Realistic and repeatable experimentation for tiny energy-harvesting sensors. In *Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems - SenSys '14*, pp. 1–15. Memphis, Tennessee: ACM Press. doi: 10.1145/2668332.2668336
57. Savanth A, Weddell A, Myers J, Flynn D, Al-Hashimi B. 2015 Photovoltaic Cells for Micro-Scale Wireless Sensor Nodes: Measurement and Modeling to Assist System Design. In *Proceedings of the 3rd International Workshop on Energy Harvesting & Energy Neutral Sensing Systems - ENSsys '15*, pp. 15–20. Seoul, South Korea: ACM Press. doi: 10.1145/2820645.2820653
58. Colin A, Lucia B. 2018 Termination Checking and Task Decomposition for Task-based Intermittent Programs. In *Proceedings of the 27th International Conference on Compiler Construction*, CC 2018, pp. 116–127. New York, NY, USA: ACM. doi: 10.1145/3178372.3179525
59. Bi S, Zeng Y, Zhang R. 2016 Wireless powered communication networks: An overview. *IEEE Wireless Communications* **23**, 10–18. doi: 10.1109/MWC.2016.7462480
60. Tanenbaum AS, Wetherall DJ. 2010 *Computer Networks*. Upper Saddle River, NJ, USA: Prentice Hall Press, 5th edition.
61. Majid AY, Jansen M, Delgado GO, Yıldırım KS, Pawełczak P. 2019 Multi-hop Backscatter Tag-to-Tag Networks. *arXiv:1901.10274 [cs]* .
62. Tan J, Pawełczak P, Parks A, Smith JR. 2016 Wisent: Robust downstream communication and storage for computational RFIDs. In *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, pp. 1–9. doi: 10.1109/INFOCOM.2016.7524574
63. Aantjes H, Majid AY, Pawełczak P, Tan J, Parks A, Smith JR. 2017 Fast downstream to many (computational) RFIDs. In *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, pp. 1–9. doi: 10.1109/INFOCOM.2017.8056987
64. Donne CD, Yildirim KS, Majid AY, Hester J, Pawe\lczak P. 2018 Backing out of Backscatter for Intermittent Wireless Networks. In *Proceedings of the 6th International Workshop on Energy Harvesting & Energy-Neutral Sensing Systems*, ENSsys '18, pp. 38–40. New York, NY, USA: ACM. doi: 10.1145/3279755.3279758