DOI:10.1145/3624718

Batteryless, energy-harvesting systems could reshape the Internet of Things into a more sustainable societal infrastructure.

BY SAAD AHMED, BASHIMA ISLAM, KASIM SINAN YILDIRIM, MARCO ZIMMERLING, PRZEMYSŁAW PAWEŁCZAK, MUHAMMAD HAMAD ALIZAI, BRANDON LUCIA, LUCA MOTTOLA, JACOB SORBER, AND JOSIAH HESTER

The Internet of Batteryless Things

IMAGINE USING A health bracelet that tracks your blood pressure and glucose level that you do not have to charge for the next 20 years. Imagine sensors attached to honeybees helping us understand how they interact with their environment or bio-absorbable pacemakers controlling heart rate for 6–8 months after surgery.

Whether submillimeter-scale "smart dust,"[25] forgettable wearables, or tiny chip-scale satellites, the devices at the heart of the future of the Internet of Things (IoT) will be invisible, intelligent, long-lived, and maintenance-free. Despite significant progress over the last two decades, one obstacle stands in the way of realizing next-generation IoT devices: *the battery.*

Batteries provide the convenience of a reliable energy supply but create a host of problems. Batteries limit device lifetime and yield high maintenance costs. As device size continues to scale down, battery density scaling has not kept pace. As IoT device applications demand more computational capabilities, energy limits lifetime to weeks or months. Even rechargeable batteries have a limited lifetime, wearing out after 300–500 charge cycles.

As we move toward a future with trillions of IoT devices,[a] replacing trillions of dead batteries and devices will be both prohibitively expensive and irresponsible. A future IoT with trillions of new battery-powered devices would create an environmental catastrophe.

Most discarded batteries end up in landfills—only 5% are recycled. Discarded batteries release toxic fumes into the air and disperse chemicals in the soil as they break down. When batteries are recycled, the process releases contaminants into waterways.[7]

Batteryless devices and intermittent executions. During the last decade, and building on work from the decades prior, research pushed toward a new kind of system that is pervasively deployed,

a https://bit.ly/491LSNE

» key insights

- **As we move toward a future with trillions of IoT devices, replacing batteries will be both prohibitively expensive and irresponsible.**
- **Over the past decade, research produced new energy-efficient programming languages, compilers, runtime systems, and architectural designs that enable real-world applications of batteryless devices.**
- **Albeit exisitng work laid a foundation for batteryless, energy-harvesting computing, the field is arguably at a stage where a much bigger leap is needed for this technology to gain widespread adoption. We discuss six fundamental directions we maintain to be crucial for the field to thrive.**

but also free from batteries. Like traditional sensing systems, compute batteryless IoT devices feature sensing, computing, and communication modules, as shown in Figure 1a. Instead of batteries that store chemical potential energy, batteryless IoT devices use small capacitors as energy buffers.

As harvested energy is highly variable and unpredictable, and capacitors are small, energy failures are frequent. As energy scarcity is pushed to an extreme, energy failures become so frequent that the usual computation model is no longer attainable. Batteryless IoT devices thus operate intermittently, as energy is available. Intermittent operation may prevent computation to advance as usual because energy failures impede progress.

As applications often run on bare hardware without proper operating system support, in the absence of dedicated solutions for intermittent execution, a programmer is often blissfully unaware of the problems lurking in their code that may only surface after deployment. Programs written without considering intermittent behavior potentially suffer from a multitude of issues.

Restartability and progress. A power failure clears the device's volatile state (main memory, register file), compromising progress not preserved in non-volatile memory. A batteryless device must resume execution instead of restarting when energy is back.[32]

Resuming often depends on restoring volatile working states, and a system must decide which volatile state to preserve before an energy failure. Higher memory preservation overhead than available energy risks the forward progress of the program.[30]

Memory consistency. Systems that incorporate non-volatile memories in their architecture risk memory consistency during intermittent execution as operations, especially with write-after-read (WAR) dependencies, might re-execute due to power failures.[28,38] As non-volatile memory manipulation is non-idempotent, meaning that when operations including WAR dependencies re-execute, they might leave a different result in non-volatile memory, producing erroneous and inconsistent outputs.

Timeliness. The time between consecutive power failures in an intermittent execution can range from hundreds of microseconds to seconds, to minutes or days, which often cannot be measured by batteryless systems.[12] Without the notion of time, cyber-physical systems sense redundant data,[34] process stale samples,[23] and provide unusable results.[24]

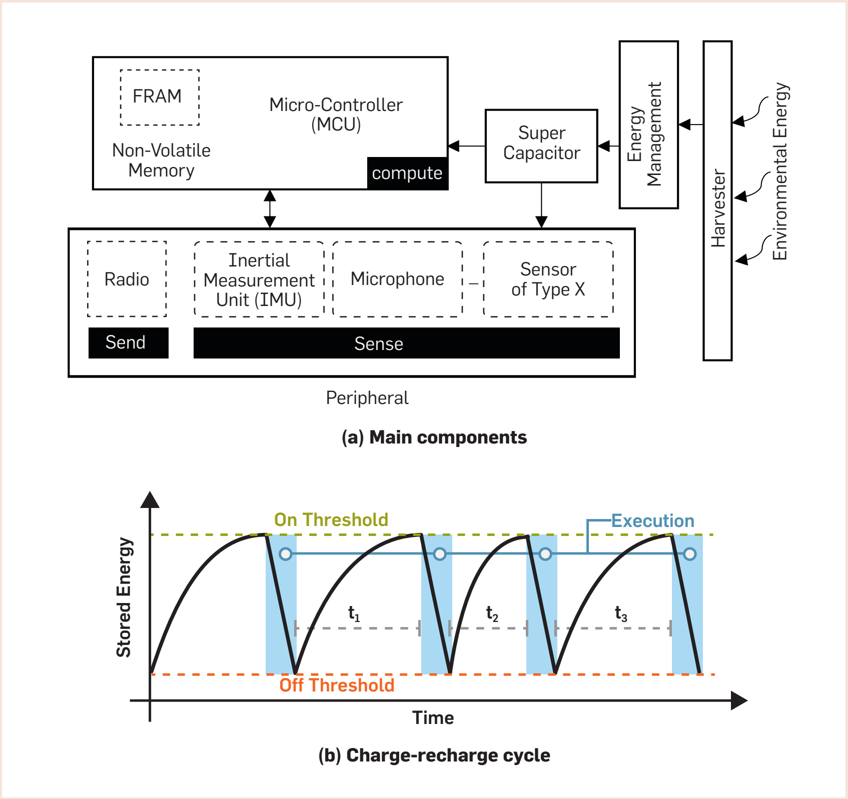
Peripherals interact with the outside world and can lead to a variety of errors in intermittent execution. A power failure can leave a peripheral device in an unrecoverable state, requiring peripheral recovery actions that depend on what part of a program was executing when power failed.[6]

State of the art. Over the past decade, researchers have addressed these challenges with advances that help developers design more sophisticated and robust applications on batteryless sensing devices. We have come a long way with these advances. In recent years, batteryless devices have been launched into space,[15] have emulated a Nintendo Game Boy to play Super Mario Bros,[13] have secured and monitored ancient monuments,[1] and have performed image processing using machine learning.[21] Figure 2 illustrates some of these achievements.

Aided by low-cost microcontrollers, more efficient solar panels, and new energy harvesters, the emerging community around intermittent computing has developed new architectures and hardware platforms, as well as software systems and tools that allow developers to design, debug, and deploy intermittently powered sensing devices in new and exciting applications that operate despite frequent energy failures.

In this article, we review the research threads that gave birth to intermittent computing and articulate the field's critical future directions. Specifically, we provide a guide to the journey taken in the last decade to lay the foundation for intermittent computing. Moreover, we discuss open challenges and future research directions, particularly as they pertain to the IoT becoming a sustainable, inclusive, and accessible instrument for societal benefit.

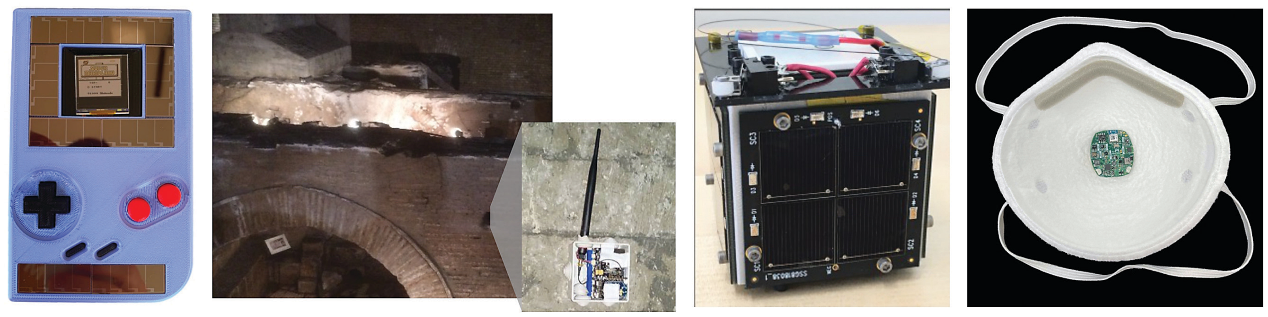
Figure 1. (a) Key components, and (b) Charge-recharge cycle.



Enabling Batteryless Systems

Batteryless devices enable a unique set of applications (as shown in Figure 2).

Figure 2. Left to right: Game Boy;[13] Mithraeum of the Circus Maximum;[1] nanosatellites;[15] and FaceBit[11] as example applications employing the intermittent computing paradigm.



However, to develop such applications, system support is required to ensure intermittence-safe execution that allows the programmer to build systems that maintain memory consistency, provide timely and accurate output, and behave as the programmer expects while making the best use of available energy. Toward this end, the intermittent computing research community has developed new energy-efficient programming languages, compilers, runtime software systems, and architectural mechanisms that eliminate challenges arising due to frequent interruptions, allowing them to focus on application development.

Here, we provide a whirlwind tour of the last decade of work on intermittent computing to address the challenges.

Programming languages designed for intermittent computing provide abstractions that simplify the programmer's task of addressing progress, consistency, timeliness, and peripherals.

A *task-based* programming model allows a programmer to divide applications into semantically meaningful execution units, such as sampling a sensor or sending a packet.[28] A task either runs to completion or, if interrupted by a power failure, re-executes after the power failure. These programming models and their implementations ensure once program execution crosses a task boundary, non-volatile memory contains a consistent set of values resulting from the task's completion. Tasks have transactional semantics: the program's state at a task boundary is guaranteed to be consistent with the completed execution of the previous task. Task-based systems have continuously evolved over the years to resolve their multiple shortcomings; may it be to reduce their data-reload overhead at the time of system restoration[8] or enable them to handle asynchronous events.[40]

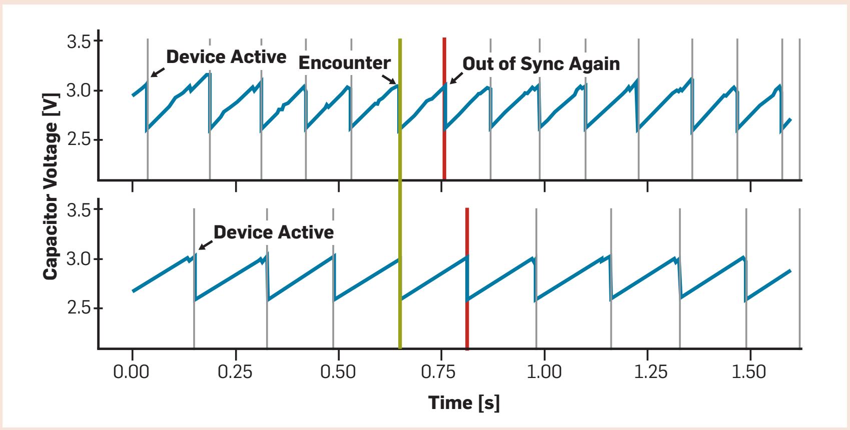
Programming languages for intermittent systems cannot ignore time and how it relates to data. Without custom language support, the application code must timestamp data and perform validity checks before each access, which complicates program logic. To this end, task-based systems have been augmented with timing constraints on the data generated by each task.[23] A program becomes a directed data-flow graph, where nodes are tasks and edges define the flow of data and temporal constraints. Representing a program as a task graph allows developers to directly express data movement structure and timing without having to reason about intermittent behavior. Formal frameworks and language support for data freshness and temporal consistency further empower developers to express timing properties in intermittent systems.[34]

Compilers and software systems. To support intermittent executions using existing embedded languages, such as C and Rust, extensive compiler and software support is needed to provide a way to checkpoint and restore the application's state after each power failure and to safeguard the execution from data and peripheral inconsistencies.

A *reactive* approach to checkpointing relies on hardware support: an interrupt is fired to save a single checkpoint just before the power runs out. It is critical for correctness that this interrupt occurs when the remaining energy is sufficient to checkpoint every bit of work and that there is no work done afterward.[4] A *proactive* approach, instead, allows programmers to reason about remaining energy and dynamically capture a checkpoint. Taking a checkpoint too early wastes energy after the checkpoint that could be used to perform meaningful work. Existing literature in this area focuses on striking a trade-off between postponing the checkpoint as long as possible; for example, in the hope the environment provisions new energy, and anticipating the checkpoint to ensure sufficient energy is available to complete it.[32] Checkpoint operations impose a modest time and energy overhead and minimizing this overhead save energy for useful application work. Code instrumentation techniques have often been employed to reduce the size of checkpoint data and dynamically disable them to adapt to energy conditions.[30]

Intermittent executions on systems with mixed volatility or completely non-volatile RAMs pose additional memory consistency threats due to the re-execution of non-idempotent code sections. These data inconsistencies only manifest when a power failure occurs between instructions involved in WAR dependency. A compile-time analysis can be used to break such WAR-dependencies in the program by decomposing code into idempotent sections and connecting them with checkpoints of volatile state.[38] Although WAR-enforced checkpoints effectively rid the program from idempotence violations, the overhead of such lazy approach

Figure 3. Example evolution of capacitor voltage and active state of two batteryless devices (top and bottom plot) when attempting to communicate (taken from Geissdoefer et al.[18] Figure 2a.). Both devices need to wait for a long time until they are active within the same short time window to exchange data. After their encounter, both devices immediately desynchronize again because harvested energy to charge individual storage capacitors depends on each device's location.



linearly increases with the number of WAR dependencies in the code. A more energy-aware approach, however, can dynamically disable needless checkpoints based on direct observations of the program progress and remaining energy.[30]

Embedded sensing workloads are most often peripheral bound. Peripherals execute asynchronously with respect to the computing unit. Their functioning is characterized by their own states, which are frequently updated due to the execution of I/O instructions or the occurrence of external events, such as the reception of a packet. Information on peripheral states is not automatically reflected in main memory, neither it may be simply queried or restored as it is often the result of non-trivial sequences of commands issued to peripherals and their answers. If peripheral states are not restored when resuming executions after a power failure, or this happens without ensuring consistency with respect to the state of the computing unit, applications may fail, or forward progress be compromised. Such peripheral inconsistency issues are often modeled as safety violations (executions reach a fail state that is unreachable in a continuous execution) and liveness violations (executions fail to reach valid states that would eventually be attained in a continuous execution).[6] Solutions to these may include representing peripherals states using state machines to be replayed after reboot to synchronize the peripheral state with compute unit and recover the state before the last checkpoint.[6]

Wireless networking. To enable batteryless communication, prior work mainly focused on extremely low-power solutions based on the principle of backscatter. Backscatter works by transmitting data as modulated reflections of an incoming radio signal. Backscatter works well for batteryless devices because it consumes very little energy. Backscatter shifts the energy-hungry parts of a radio transmitter (for example, power amplifier) to an external source that generates signals for backscattering. As a result, backscatter transmitters may consume only picojoules per bit, substantially outperforming the nanojoules-per-bit active radios transmitting via the same protocol.[16] While appealing, backscatter has several drawbacks, including reliance on a costly external source and limited communication range and throughput. Therefore, network protocol design for batteryless systems using active radios is a promising research direction. Figure 3 illustrates one of the fundamental problems, which also applies to backscatter: Two devices that intend to communicate may require many charging cycles (up to hundreds or thousands[17]) until they are active within the same time window (encounter). This is because harvested energy to charge the individual energy storage capacitor depends on each devices' location. As a result, two devices also lose synchronization immediately after an encounter, having to wait again for a long time until the next communication opportunity.

Solutions to achieve efficient and reliable batteryless communication despite the problem noted here include batteryless nodes encoding their current storage capacitor voltage into bursts of different frequencies and lengths to inform their neighbors about their energy levels and adapt their rendezvous times.[36] Another approach learns capacitor charge patterns of neighboring devices to maintain long-running connections across many consecutive encounters.[18] To break interleaved charging patterns (see Figure 3), randomly delaying the wake-up times of batteryless devices and synchronizing to a common clock signal, such as powerline flicker, has been shown to be effective.[17] A key challenge for these approaches is operating over long periods of time at a large scale.

The intermittent communications stack also requires support for network protocol state retention and connection-oriented communication, which is a nascent research area. Devices running standard protocols may be in any state, such as connection request or connection establishment. However, power failures may lead a device to lose its protocol state, disrupting communication. Thus, systems require support for retaining protocol state across power failures without a high penalty for re-establishing a connection. Recent work has demonstrated network state retention for intermittent Bluetooth communication,[14] saving and restoring the Bluetooth state machine state using non-volatile memory.

Today's network protocol specifications are not a good match for intermittent operation. For instance, the Bluetooth specification forces a connection reset after 32 seconds of inactivity. This is a poor match for intermittent operation as the time needed to recharge the energy storage capacitor may be longer than the reset period. Our call to action for batteryless networking is that popular network protocols, including Wi-Fi, should be modified to support intermittent operation (with a minimum of modifications to the protocol itself). An answer to this call will make intermittent devices compatible with existing network protocols and infrastructure.

Power systems and computer architecture. The performance of intermittent systems (for example, throughput) largely depends on the energy storage capacity, processor and sensor power consumption, and other architectural characteristics. To illustrate, though using a larger capacitor seems like a solution to address high response time due to increased power consumption, large capacitors have longer charging time, which increases response time and energy leakage. Checkpointing overheads heavily depend on the choice of hardware and architecture, for example, cache size and volatile memory size.

To provide support for high-power consumption while maintaining low response time, several energy storage architectures have been proposed over time. A single large capacitor requires tasks with low energy demand to wait to execute until the capacitor is fully charged despite having sufficient energy for the task.[22] To address this, re-configurable capacitor banks[9] have been proposed to dynamically select the active energy storage size based on the demand of the tasks. Supporting the diverse energy requirements of the peripherals demands dedicated energy storage for each peripheral providing energy isolation and reducing unnecessary energy consumption.[22]

Non-volatile processors (NVP) have non-volatile flip-flops and gates which reduce checkpointing overhead and leakage power by shutting down idle systems with fast backup and retention operations. However, they increase the power (and area) requirements and increase processing time.[29] More highly optimized von Neumann processors (for example, with out-of-order execution) are unlikely to provide a benefit to intermittent systems because they trade higher power for higher performance and demand more sophisticated backup policies.

Recent studies presented the intermittent inference on batteryless edge,[21,24] which creates a demand for parallel intermittent execution support for data-intensive tasks.

Intermittent systems must adapt to frequently changing environmental energy conditions to exploit the available energy more efficiently. Maximizing forward progress,[2] increasing output performance,[24] and adaptive heterogeneous architecture (composed of cores with different energy consumption) are some of the proposed adaptive architectures to increase efficiency. Timely intermittent execution[23,40] requires low-power timekeeping[12] and accurate voltage measurement circuits.[39]

Directions For the Future

The previous work laid a foundation for the next phase of power failure-resilient, energy-harvesting computing. The field is arguably at a stage where a much bigger leap is needed for this technology to gain widespread adoption. We discuss six fundamental directions we maintain to be most crucial for the field to thrive.

Deployment matters. IoT systems are designed and deployed to harvest accurate and abundant data. How this is concretely achieved in the absence of traditional batteries—whether using a programming system or low-power hardware—is ultimately immaterial. What matters is the system's ability to act as an efficient interface between the physical and the digital world. The field of intermittent computing is in desperate need of making this a top priority or the existing body of work runs the concrete risk of being relegated to an academic exercise.

Understanding the real-world performance of an intermittently computing IoT system requires measuring its performance in realistic deployments. Existing experiences are extremely limited and mostly the result of intense one-off efforts. Large-scale long-term deployments of intermittent computing technology are rare,[1] and yet provide invaluable insights and directions for future work that are not attainable elsewhere.

First, the few existing efforts reveal that no single technique among those we surveyed earlier suffices to build a fully functional system. The overhead of combining and integrating disparate techniques from the state of the art often defeats the efficient performance of each of these techniques when deployed in a lab or used in isolation. Second, lessons learned from real deployments point to an acute tradeoff between generality and energy efficiency once in operation. In a setting characterized by extreme resource scarcity, there is not much to waste at runtime in a quest to support multiple different application scenarios or diverse requirements.

The natural conclusion is that efforts should focus on complete development environments that coherently integrate the necessary techniques; for example, by combining techniques ensuring forward progress with peripheral correctness. Enabling this on a larger scale requires gaining further experiences from realistic scenarios, and possibly exploring new application domains.[11,15]

By the same token, crucial for researchers to be motivated and invest efforts in this direction is to gain the proper recognition from academic conferences, journals, and funding bodies alike. This is arguably and unfortunately not the case right now. We understand this issue is likely to cross the boundaries of the specific area at stake, and impact system research at large. We also maintain, however, those deeply embedded systems akin to the ones we discuss here impose even greater efforts in debugging and deploying at the current prototyping stage.

Correctness. Battery-free systems must operate correctly and yet, unfortunately, intermittent systems lack a formal foundation to reason on their correctness. Correctness guarantees make new applications possible: medical applications built around intermittent devices remain infeasible in the absence of correctness guarantees. They will produce wrong results or fail to produce results at all unless an intermittent hardware/software system can be proven correct.

Correctness guarantees may also make existing applications more efficient or enable better resource usage: pervasive infrastructure monitoring sensors and tiny chip-scale satellites may require more redundantly deployed devices for reliable operation if each device comes with no correctness guarantee. But redundancy drives up costs. The solution is to build up the formal foundations of intermittent computing to provide formal definitions of correctness. A good starting point is to use the machinery of programming language research to provide correctness guarantees for intermittently computing software runtime systems and applications.[5,35]

Further questions, however, do require an answer. For example, should formal correctness models of intermittent operation include models of physics-based power and energy, arbitrary peripheral devices, or user behavior and the operating environment?

Further, what are the formalisms and semantics models best suited to express the correctness requirements developers are interested in, and how do these possibly enable automated verification of such requirements? Each of these variations on the modeling problem presents its own challenges and benefits.

One example modeling challenge that is particularly valuable, yet unsolved in general, is building a model of energy consumption for an arbitrary region of code. A solution to this problem allows a programmer to directly reason about the location or likelihood of a power failure at a particular program point. Such a solution also strengthens the link between supplied power behavior and program behavior, supporting new program analyses that help programmers understand how energy consumption varies across executions of their program. Finally, although power and energy are physical quantities and vary for reasons outside of the closed world of software, probabilistic programming languages research may play an important role in modeling such physically influenced, distributional behavior.[10]

Security. Provided the battery-free system is proven correct, data inside and around the system is useless in the absence of security. Because of their deeply embedded nature and relation with the surrounding physical environment, IoT systems are at risk of security attacks, and intermittently computing ones are no exceptions. Rather, extreme resource constraints and peculiar execution patterns increase the attack surface and enable new attacks.

The problem is indeed difficult: batteryless IoT cannot run mainstream security techniques and protocols without incurring in huge performance penalties.[3] Dedicated hardware support, such as ARM TrustZone, provides a steppingstone to address the security challenge, yet intermittent executions likely require ad-hoc specialized solutions. A few works[37] demonstrate hardware support, porting the guarantees of fully homomorphically encrypted computing to low-end embedded devices, possibly including intermittent ones. How to blend these solutions with existing intermittent computing techniques is an open question: the non-continuous execution flow, which possibly includes re-executions of non-idempotent code, creates new problems for security researchers.

Recent work[31] studies how ambient energy harvesting may be used as an attack vector in intermittent computing. The authors demonstrate that by exerting limited control on the ambient supply of energy to the system, one can create situations of live lock, denial of service, and priority inversion, without requiring physical access to a device. Using machine learning and concepts of approximate computing, they design an attack detection technique with 92%+ accuracy and limited run-time overhead. Much greater efforts are required, however, to investigate to what extent energy provisioning may be used to maliciously drive intermittent executions and what detection and mitigation techniques are to be put in place to counteract these occurrences.

Energy-minimal computer architectures. In designing a computer architecture for a batteryless system, energy efficiency is the most important design concern. Harvesting energy rate limits device operation if the harvestable input power is substantially lower than computational operating power. A batteryless system in this operating regime should maximize the amount of computational work that it does with its energy, optimizing primarily for efficiency, rather than focusing on improving performance or reducing operating power (both of which sometimes may impose a cost in efficiency. Many battery-free devices use simple microcontrollers to compute, (for example, the Texas Instruments MSP430), which offer appealingly low power consumption, simple ISAs with widespread compiler support, and a typical von Neumann execution model. Unfortunately, these simple von Neumann architectures do not offer high-energy efficiency, making them increasingly the wrong choice as batteryless systems perform steadily more computation.

Existing architectures are inefficient for two main reasons that both stem from the von Neumann execution model: instruction supply and data movement. The instruction's fetch and decode consume a large amount of energy, often entailing memory accesses.

To execute an operation, a von Neumann core must configure internal control signals on a per-instruction basis and move input operands from and store outputs to a register file or memory. These overheads of instruction and data supply consume around 55% of total compute energy.[20] When efficiency matters, this overhead is unacceptable.

The future of computing in batteryless devices lies in new computer architectures that eliminate the overheads of von Neumann computing. For the last decade, computer architects have improved efficiency through hardware specialization, for example, for some machine-learning computations. Specialized accelerators forfeit general-purpose programmability but achieve high efficiency by eliminating most control overheads. Accelerators are not likely to be the solution for most long-lived deployments, due to their inability to adapt to changing application requirements. Additionally, accelerator-based architectures are not sustainable because they require new silicon for each application and the bulk of a chip's carbon footprint is in its manufacturing. Low-power FPGAs offer reconfigurability but are difficult to program and do not meet the efficiency of specialized accelerators.

Recently, work on energy-minimal computer architecture[19,20] has developed new reconfigurable dataflow architectures implemented in coarse-grained reconfigurable arrays (CGRAs). These CGRAs achieve high efficiency while remaining fully programmable from high-level languages like C and Rust. Energy-minimal CGRA architectures implementing ordered dataflow[19,20] consume a few hundred microwatts of power and perform hundreds of billions of operations per second per watt. The key to the efficiency of these architectures is that they eliminate virtually all instruction control overheads and streamline data movement via efficient on-chip dataflow networks.

The concrete benefits show the promise of CGRAs to batteryless system designers. The SNAFU[19] architecture reports efficiency running linear algebra in software that is within a factor of 2–3× of a fixed purpose accelerator implementing the same computation. The RipTide[20] compiler fully automatically compiles complex DNN inference code with sparse data and irregular control flow to a dataflow fabric that runs that code using 1900× less energy and 146× less time than a TI MSP430. As software programming support improves and embedded non-volatile memories become available, reconfigurable dataflow architectures are a highly appealing option for batteryless devices.

Another promising direction is to exploit in-memory processing on emerging non-volatile memory technologies to process part or entire data in situ in memory to accelerate data-intensive computing tasks in intermittent systems, for example, machine learning and signal processing. Data backup happens atomically and immediately in these systems since they exploit non-volatile memory technology. For instance, Resch et al.[33] presented intermittent computing on a magnetic tunnel junction-based computational RAM, a highly energy-efficient processing element. Despite its advantages, CRAM programming is significantly more difficult than programming our general-purpose batteryless platforms. Filling the programming gap for in-memory processing will bring significant benefits to intermittent computing.

Foundation experimental infrastructure. For intermittent computing to thrive, we argue it is essential to develop a foundational set of infrastructure: (that is, tools, testbeds, hardware platforms, benchmarks, and datasets) enabling a sound scientific investigation. Unlike other fields of computing and communications, intermittent computing is plagued by one major issue: the absence of a well-defined, agreed-upon yardstick to compare the performance of systems, namely, a set of benchmarks and the mechanisms to run them repeatably and at scale. Despite the performance claims in several papers, even in the many cases where these claims are substantiated by rigorous evaluations, the extent to which the results for one system hold in the setup of another is unclear at best. This situation is not just methodologically flawed from a scientific standpoint; it also may also have distorting effects on the industrial adoption of research artifacts.

The future of computing in batteryless devices lies in new computing architectures that eliminate the overheads of von Neumann computing.

This is a typical hurdle for any field as it emerges into the mainstream. First, we need a common, agreed-upon benchmark against which the performance of systems can be measured. This approach is widespread among many fields of science, for instance, applied in testing machine learning systems for image classification. A benchmark typically defines input parameters representative of several application cases, output metrics quantifying the performance of the benchmarked systems, and the required experimental setup.

In our case, a benchmark might include harvested energy traces as well as common computational tasks. Batteryless system would then produce output based on the input data, such as the number of packets in a unit of time. The harvested energy traces have a special purpose for testing intermittent computing systems as energy represents a program input for intermittent computing systems. The specific energy patterns the system is exposed to, for example, determine how many interruptions the program experiences. It is therefore crucial to collect energy traces that are both diverse as not to bias the results towards specific techniques and be able to exercise the various program paths. Although in other fields, the availability of benchmarks may result in overfitting system designs, benchmarks for batteryless intermittent computing systems must be varied, broad, and frequently updated.

Second, one needs definitive platforms, tools, datasets, and testbeds, to enable benchmarks at scale. Benchmarks without a well-defined (and easy-to-acquire) hardware platform are much less useful. Benchmarks need standard tools and programs to be able to run, that are both energy harvesting and power failure aware. To scale up benchmarking and replicable science, to enable access to the broader community, and to ease testing and comparisons, foundational infrastructure is necessary.

What about the users? No matter how sophisticated the technology is, chances are that any system may stop operating. In an AC power socket-powered system, this is a rare occurrence and is generally considered as a failure. In a regular battery-powered system, operation failure is more probable. Nonetheless, users are conscious of this potential operation stop, they know how to handle it, and they consider it a part of the system's lifetime. In a battery-free IoT system—and especially an intermittently executing one—the relationship between the end user and a device that stops and resumes the execution (even on sub-second time scales) is all to be explored.

Development environments like Arduino are now considered major milestones in the development of the IoT. They enable nontrained programmers to quickly prototype fully functional IoT systems and are fundamental tools for communities that extend well beyond the borders of traditional computer science and engineering. These movements are, often and interestingly, a cradle of innovation that builds on the skills and expertise of people with diverse backgrounds.

Unfortunately, intermittent computing is very far from such a situation. A few recent works represent timid attempts at lowering the barrier to entry for non-trained people outside of computer science and engineering.[26,27] Instead, using the vast majority of systems and techniques discussed earlier requires expertise and skills that not even every computer engineer is necessarily equipped with—let alone researchers and practitioners outside of this discipline.

The key question is whether (or how) to mask the distinctive features of intermittent computing. The battery-free Game Boy,[13] for example, takes a specific stand: users are required to contribute to energy provisioning to tame the intermittent executions.

Efforts should be invested in understanding—from a technical but also psychological perspective—how to develop the unfolding relationship between users and intermittently computing devices.

Alternative designs are also possible where, for example, intermittence is made explicitly visible to programmers and users rather than hidden from them. Striking an effective trade-off is anything but trivial and likely requires researchers in intermittent computing and closely related fields to let their efforts be influenced by other disciplines. Major publication venues, on the other hand, should increasingly reward multidisciplinary efforts, creating the incentive required to make progress in this direction.

Notwithstanding, lowering the barrier to entry for non-trained developers is thus just half of the story—the other half is the proper setting of the user interaction with an intermittently computing device. Therefore, efforts should be invested in understanding—from a technical but also psychological perspective—how to develop the unfolding relationship between users and intermittently computing devices. One option, for example, is to let users merely accept the idea they cannot continuously rely on a system. Although simple to realize, this may likely limit the applicability of intermittent computing (battery-free) technology. A different option is rather to engage the user in energy provisioning,[13] explicitly or implicitly; for example, through some form of reward or "gamification" of the energy provisioning tasks that creates an incentive for the user to look for the right conditions to make the system continue the work.

Reasoning on this matter cannot be oblivious to the specific application at hand. For example, listening to music on a device that powers off a couple of times per minute is unacceptable, yet playing puzzle games on such devices should be doable and enjoyable.

These devices can go beyond entertainment, to actuating within the physical environment—for example, harvesting the action from mechanical motions like door openings and closing to power sensing and occupancy readings. They could be integrated into a user's shoe, clothing, or even prosthetic, where the action of walking compressed a spring that captured energy to power strain sensing and health monitoring for the limb. In summary, understanding the human-interaction angle of intermittent computing and a batteryless IoT is crucial, programmer facing tools for easing development, interfaces, and designs that mask or work with power failures, and novel applications and integration points for energy harvesting, are all part of the next phase of research.

Conclusion

Bell's Law of Computing Classes states that a lower-priced, and more numerous, computer class emerges approximately every decade that requires entirely new methods of programming networking, and interfacing resulting in wholly new applications. After cellphones, IoT appears to be this new class—the results of the research discussed in this article point to a way to scale responsibly with this new class. The first lights of interesting, enabled applications have already emerged, and further research will explore where long-term, low-cost, massive-scale sensing is essential including healthcare (wearable and body sensor networks), ecology, horticulture, agriculture, infrastructure, and public utilities monitoring. This Internet of Batteryless Things is a step toward the sustainable future of computing. While it is flourishing, its progress is hampered by major challenges in terms of testbeds, tools, security, and applications. There is a need to grow the community around the vision of intermittent computing and more strongly advocate its need for the future. C

References

1. Afanasov, M. et al. Battery-less zero-maintenance embedded sensing at the mithræum of circus maximus. In *Proceedings of SenSys (Virtual event, Nov. 16–19, 2020)*. ACM, 368–381; 10.1145/3384419.3430722.
2. Ahmed, S. et al. Intermittent computing with dynamic voltage and frequency scaling. In *Proceedings of EWSN (Lyon, France, Feb. 17–19, 2020)*. ACM, 97–107; https://dl.acm.org/doi/10.5555/3400306.3400319.
3. Asad, H.A. et al. On securing persistent state in intermittent computing. In *Proceedings of ENSsys (Virtual event, Nov. 6, 2020)*. ACM, 8–14; 10.1145/3417308.3430267.
4. Balsamo, D. et al. Hibernus: Sustaining computation during intermittent supply for energy-harvesting systems. *IEEE Embedded Sys Lett. 7*, 1, (Ma.2015), 15–18; 10.1109/les.2014.2371494
5. Bohrer, R. and Islam, B. Cyber-physical verification of intermittently powered embedded systems. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. 41*, 11 (Nov. 2022), 4361–4372; 10.1109/TCAD.2022.3197541
6. Branco, A., Mottola, L., Alizai, M.H. and Siddiqui, J.H. Intermittent asynchronous peripheral operations. In *Proceedings of SenSys (Virtual event, Nov. 10–13, 2019)*. ACM, NY, USA, 55–67; 10.1145/3356250.3360033.
7. Church, C. and Wuennenberg, L. *Sustainability and second life: the case for cobalt and lithium recycling*, 2019; https://www.iisd.org/publications/sustainability-and-second-lifecase-cobalt-and-lithium-recycling
8. Colin, A. and Lucia, B. Chain: Tasks and channels for reliable intermittent programs. In *Proceedings of OOPSLA (Amsterdam, The Netherlands, Oct. 30–Nov. 4, 2016)*. ACM, 514–530; 10.1145/2983990.2983995.
9. Colin, A., Ruppel, E. and Lucia, B. A reconfigurable energy storage architecture for energy-harvesting devices. In *Proceedings of ASPLOS (Williamsburg, VA, USA, Mar. 24–28, 2018)*. ACM, 767–781; 10.1145/3296957.3173210.
10. Collin, A. and Lucia, B. Termination checking and task decomposition for task-based intermittent programs. In *Proceedings of CC (Vienna, Austria, Feb. 24–25, 2018)*. ACM, 183:1–183:31; 10.1145/3178372.3179525.
11. Curtiss, A. et al. Facebit: Smart face masks platform. *ACM Interact. Mob. Wearable Ubiquitous Technol. 5*, 4 (Dec. 2021), 151:1–151:44; 10.1145/3494991.
12. de Winkel, J. et al. Reliable timekeeping for intermittent computing. In *Proceedings of ASPLOS (Lausanne, Switzerland, Mar. 16–20, 2020)*. ACM, 53–67; 10.1145/3373376.3378464.
13. de Winkel, J. et al. Battery-free game boy. *ACM Interact. Mob. Wearable Ubiquitous Technol. 4*, 3 (Sept. 2020), 111:1–111:34; 10.1145/3411839.
14. de Winkel, J. et al. Intermittently-powered bluetooth that works. In *Proceedings of MobiSys (Portland, OR, USA, Jun. 25–Jul. 1, 2022)*. ACM, 287–301; 10.1145/3498361.3538934.
15. Denby, B. and Lucia, B. Orbital edge computing: Nanosatellite constellations as a new class of computer system. In *Proceedings of ASPLOS (Lausanne, Switzerland, Mar. 16–20, 2020)*. ACM, 939–954; 10.1145/3373376.3378473.
16. Ensworth, J.F. and Reynolds, M.S. Every smart phone is a backscatter reader: Modulated backscatter compatibility with bluetooth 4.0 low energy devices. In *Proceedings of RFID (San Diego, CA, USA, Apr. 15–17, 2015)*. IEEE, 78–85; 10.1109/rfid.2015.7113076.
17. Geissdoerfer, K. and Zimmerling, M. Bootstrapping battery-free wireless networks: Efficient neighbor discovery and synchronization in the face of intermittency. In *Proceedings of NSDI (Renton, WA, USA Apr. 12–14, 2021)*. USENIX, 439–455; https://www.usenix.org/system/files/nsdi21-geissdoerfer.pdf.
18. Geissdoerfer, K. and Zimmerling, M. Learning to communicate effectively between batt ery-free devices. In *Proceedings of NSDI (Renton, WA, USA, Apr. 4–6, 2022)*. USENIX, 419–435; https://www.usenix.org/system/files/nsdi22-papergeissdoerfer.pdf.
19. Gobieski, G. et al. Snafu: An ultra-low-power, energy-minimal cgra generation framework and architecture. In *Proceedings of ISCA (Valencia, Spain, Jun. 14–18, 2021)*. ACM/IEEE, 1027–1040; 10.1109/ISCA52012.2021.00084.
20. Gobieski, G. et al. Riptide: A programmable energy-minimal dataflow compiler and architecture. In *Proceedings of MICRO (Chicago, IL, USA, Oct. 1–5, 2022)*, ACM; 10.1109/MICRO56248.2022.00046.
21. Gobieski, G., Lucia, B. and Beckmann, N. Intelligence beyond the edge: Inference on intermittent embedded systems. In *Proceedings of ASPLOS. (Providence, RI, USA, 2019)*. ACM, 199–213; 10.1145/3297858.3304011.
22. Hester, J., Sitanayah, L. and Sorber, J. Tragedy of the coulombs: Federating energy storage for tiny, intermittently-powered sensors. In *Proceedings of SenSys (Seoul, South Korea, Nov. 1–4, 2015)*. ACM, 5–16; 10.1145/2809695.2809707.
23. Hester, J., Storer, K. and Sorber, J. Timely execution on intermittently powered batteryless sensors. In *Proceedings of SenSys (Delft, The Netherlands, Nov. 6–8, 2017)*. ACM, 17:1–17:13; 10.1145/3131672.3131673.
24. Islam, B. and Zygarde, N.S. Time-sensitive on-device deep inference and adaptation on intermittently-powered systems. *ACM Interact. Mob. Wearable Ubiquitous Technol. 4*, 3 (Sept. 2020), 82:1–82:29; 10.1145/3411808.
25. Kahn, J.M., Katz, R.H. and Pister, K.S.J. Next century challenges: Mobile networking for "smart dust." In *Proceedings of MobiCom (Seattle, WA, USA, 1999)*. ACM, 271–278; 10.1145/313451.313558.
26. Kortbeek, V. et al. Bfree: Enabling battery-free sensor prototyping with python. *ACM Interact. Mob. Wearable Ubiquitous Technol. 4*, 4 (Dec. 2020), 135:1–111:39; 10.1145/3432191.
27. Kraemer, C., Guo, A., Ahmed, S. and Hester, J. Batteryfree makecode: Accessible programming for intermittent computing. *ACM Interact. Mob. Wearable Ubiquitous Technol. 6*, 1 (Mar. 2022), 18:1–18:35; 10.1145/3517236.
28. Lucia, B. and Ransford, B. A simpler, safer programming and execution model for intermittent systems. In *Proceedings of PLDI (Portland, OR, USA, June 13–17, 2015)*. ACM, 575–585; 10.1145/2813885.2737978.
29. Ma, K. et al. Architecture exploration for ambient energy harvesting nonvolatile processors. In *Proceedings of HPCA (Burlingame, CA, USA, Feb. 7–11, 2015)*. IEEE, 526–537; 10.1109/HPCA.2015.7056060.
30. Maeng, K., Colin, A. and Lucia, B. Adaptive dynamic checkpointing for safe efficient intermittent computing. In *Proceedings of OSDI (Carlsbad, CA, USA, Oct. 8–10, 2018)*. USENIX, 129–144; https://www.usenix.org/system/files/osdi18-maeng.pdf.
31. Mottola, L., Hameed, A. and Voigt, T. *Energy Attacks in the Battery-Less Internet of Things*, 2023; https://arxiv.org/abs/2304.08224.
32. Ransford, B., Sorber, J. and Fu, K. Mementos: System support for long-running computation on rfid-scale devices. In *Proceedings of ASPLOS (Newport Beach, CA, USA, Mar. 5–11, 2011)*. ACM, 159–170; 10.1145/1950365.1950386.
33. Resch, S. et al. MOUSE: Inference in non-volatile memory for energy harvesting applications. In *Proceedings of MICRO (Athens, Greece, Oct. 17–21, 2020)*. ACM/IEEE, 400–414; 10.1109/MICRO50266.2020.00042.
34. Surbatovich, M., Jia, L. and Lucia, B. Automatically enforcing fresh and consistent inputs in intermittent systems. In *Proceedings of PLDI (Virtual event, June 20–25, 2021)*. ACM, 851–866; 10.1145/3453483.3454081.
35. Surbatovich, M., Lia, L. and Lucia, B. Towards a formal foundation of intermittent computing. In *Proceedings of ACM OOSPLA 4*, (Nov. 2020), 163:1–163:31; 10.1145/3428231.
36. Torrisi, A., Yıldırım, K.S. and Brunelli, D. Reliable transiently-powered communication. *IEEE Sens. J. 22*, 9 (May 2022), 9124–9134; 10.1109/jsen.2022.3158736.
37. van der Hagen, M. and Lucia, B. Client-optimized algorithms and acceleration for encrypted compute offloading. In *Proceedings of ASPLOS (Lausanne, Switzerland, Feb. 28–Mar. 4, 2022)*. ACM, 683–696; 10.1145/3503222.3507737.
38. van der Woude, J. and Hicks, M. Intermittent computation without hardware support or programmer intervention. In *Proceedings of OSDI (Savannah, GA, USA Nov. 2–4, 2016)*. ACM, 17–32; https://www.usenix.org/system/files/conference/osdi16/osdi16-van-der-woude.pdf.
39. Williams, H., Moukarzel, M. and Hicks, M. Failure sentinels: Ubiquitous just-in-time intermittent computation via low-cost hardware support for voltage monitoring. In *Proceedings of ACM/IEEE ISCA (Valencia, Spain, June 14–18, 2021)*, 665–678; 10.1109/ISCA52012.2021.00058.
40. Yıldırım, K.S. et al. Ink: Reactive kernel for tiny batteryless sensors. In *Proceedings of SenSys (Shenzhen, China, Nov. 4–7, 2018)*. ACM, 41–53; 10.1145/3274783.3274837.

Saad Ahmed, Georgia Institute of Technology, Atlanta, GA, USA.

Bashima Islam, Worcester Polytechnic Institute, Worcester, MA.

Kasim Sinan Yildirim, University of Trento, Trentino-Alto Adige, Italy.

Marco Zimmerling, TU Darmstadt, Hesse, Germany

Przemyslaw Pawelczak, Delft University of Technology, Delft, The Netherlands.

Muhammad Hamad Alizai, Lahore University of Management Sciences, Lahore, Pakistan.

Brandon Lucia, Carnegie Mellon University, Pittsburgh, PA, USA.

Luca Mottola, Politecnico di Milano, Italy.

Jacob Sorber, Clemson University, Clemson, SC, USA.

Josiah Hester, Georgia Institute of Technology, Atlanta, GA, USA.

Watch the authors discuss this work in the exclusive *Communications* video. https://cacm.acm.org/videos/batteryless-things