AdaMICA: Adaptive Multicore Intermittent Computing

KHAKIM AKHUNOV, University of Trento, Italy
KASIM SINAN YILDIRIM, University of Trento, Italy

Recent studies on intermittent computing target single-core processors and underestimate the efficient parallel execution of highly-parallelizable machine learning tasks. Even though general-purpose multicore processors provide a high degree of parallelism and programming flexibility, intermittent computing has not exploited them yet. Filling this gap, we introduce AdaMICA (Adaptive Multicore Intermittent Computing) runtime that supports, for the first time, parallel intermittent computing and provides the highest degree of flexibility of programmable general-purpose multiple cores. AdaMICA is adaptive since it responds to the changes in the environmental power availability by dynamically reconfiguring the underlying multicore architecture to use the power most optimally. Our results demonstrate that AdaMICA significantly increases the throughput (52% on average) and decreases the latency (31% on average) by dynamically scaling the underlying architecture, considering the variations in the unpredictable harvested energy.

CCS Concepts: • **Computer systems organization → Embedded software**; **Multicore architectures**; • **Computing methodologies → Parallel computing methodologies**.

Additional Key Words and Phrases: batteryless systems, intermittent computing, energy-aware adaptive systems, multicore systems, parallelism

ACM Reference Format:
Khakim Akhunov and Kasim Sinan Yildirim. 2022. AdaMICA: Adaptive Multicore Intermittent Computing. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 6, 3, Article 98 (September 2022), 30 pages. https://doi.org/10.1145/3550304

1 INTRODUCTION

By 2030, it is expected that there will be about 50 billion Internet-of-Things (IoT) devices worldwide [74]. The vast majority of modern IoT devices rely on batteries, which are nondurable, heavy, and relatively expensive chemical energy storage. In contrast, battery-free devices with minimal energy buffer that are powered by harvestable ambient energy can dramatically extend the operating lifecycle of the IoT systems [17]. Lightweight devices are perfectly integrable into wearable technology, while the environmental-friendly nature of these devices expands the idea of ubiquitous computing, enabling new applications in the wild world, in the bowels of the earth, in space, and in other delicate or harsh environments. These devices can even pave the way for the hyped vision of ubiquitous computing such as the *Smart Dust* [42] and the *TerraSwarm* [48].

Energy-harvesting battery-free devices can exploit energy from various sources such as radio waves, sunlight, kinetic power. However, the fluctuating availability of ambient energy causes frequent power failures, forcing the systems to operate *intermittently*. When power is available, a battery-free device gradually accumulates energy in a small storage component, e.g., in a capacitor. Once the amount of stored energy in the capacitor reaches a certain threshold, the device starts executing tasks, rapidly dissipating the stored energy. The device turns off

Authors' addresses: Khakim Akhunov, khakim.akhunov@unitn.it, University of Trento, via Sommarive 9, Trento, TN, Italy, 38123; Kasim Sinan Yildirim, kasimsinan.yildirim@unitn.it, University of Trento, via Sommarive 9, Trento, TN, Italy, 38123.

2474-9567/2022/9-ART98 $15.00
https://doi.org/10.1145/3550304

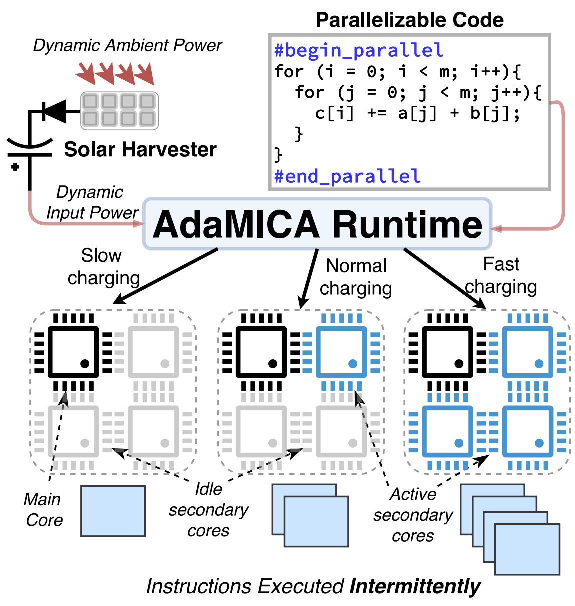


Fig. 1. AdaMICA enables programmers to annotate parallelizable code blocks that can be executed over multiple cores intermittently. Considering the available power, it reconfigures the multicore architecture to obtain the maximum benefit from the ambient energy and parallelism.

when it consumes all stored energy, which leads to the loss of the volatile computational states (e.g., registers, stack). Therefore, power failures interrupt the execution of programs, which might prevent the computation from progressing forward and leave the memory in an inconsistent state [65].

Recent research has proposed solutions for intermittent systems, considering both software, e.g., [18, 44, 78], and hardware, e.g., [13, 19, 22, 35], aspects, guaranteeing execution progress and maintaining memory consistency. While efficient, the proposed solutions target off-the-shelf single-core ultra-low-power microcontrollers (MCU) with limited flexibility and performance capability. These MCUs are energy efficient and ideal for performing low-cost tasks, but they pose limitations to executing computationally intense tasks intermittently [30]. These limitations are critical since the machine-learning (ML) paradigm, the number one disruptive technology of the moment, demands the execution of high-cost inference tasks on edge devices. The reason is that sending large amounts of raw sensor data wirelessly to offload the high-cost tasks to the cloud is too energy-intensive for energy-harvesting devices.

As of now, only a few studies addressed the development of software- and hardware-based techniques to execute high-cost computational tasks intermittently and efficiently under energy constraints. Unfortunately, these studies pose significant deficiencies, as we explain below.

P1-Lack of Multicore Parallelism. High-cost ML tasks are highly parallelizable. When the ambient energy permits, splitting these tasks over multiple cores and executing them in parallel can increase the throughput significantly. Even though ordinary homogeneous multicore systems commencing to spread widely in ultra-low-power embedded devices [29, 43], intermittent computing has not benefited them yet. Current studies mainly exploit accelerators to execute high-cost tasks and exploit parallelism [30, 31, 49, 52], which have crucial drawbacks. As an example, [49] and [30] exploit the low-energy accelerator (LEA) on MSP430FR5994 [36], which is one of the mainstream MCUs for intermittent computing, to support vector-based signal processing operations. However, this accelerator exhibits functional limitations, and it has limited parallelism since it only supports dense operations [30, Section 10]. Besides, accelerators are generally task-specific, which does not satisfy the flexibility demand of IoT edge nodes [29]. Moreover, they lose their computational state upon power failures, which requires reconfiguration and repeated data transfer from volatile/non-volatile memory to their internal memory buffers.

P2-Missing Parallel Programming Support. In existing systems such as SONIC [30], programmers need to exploit underlying parallelism manually by interacting directly with the accelerator, which is cumbersome. For example, LEA has a set of specific vector operations (some generic vector operations are not supported [30, Section 7]), which makes exploiting parallelism difficult. The programmable general-purpose multiple cores provide the highest degree of flexibility [29, 69], but the intermittent computing community has overlooked them so far. Existing intermittent computing runtimes, e.g., [18, 44, 54, 78], do not support parallelism and do not provide language constructs to express parallelizable code blocks.

P3- Rigid Hardware Configuration. The availability of energy and the strength of incoming power affects the charging and discharging cyclical nature of an intermittent system [24]. When incoming power is strong enough, the device charges rapidly and spends more time for computation. Similarly, low input power forces the system to spend more time collecting energy rather than computing. Existing works [11, 40, 56–59] have provided software-based adaptation strategies, such as degrading the computational accuracy, to respond to ambient power dynamics and increase the throughput. However, they work on a fixed hardware configuration. Recent works [6, 53] have proposed a dynamic voltage and frequency scaling technique to scale computational support, concerning the input power. However, these solutions still target systems limited by the performance of a single-core processor. Another strategy is to switch between different compute units that have different performance and energy consumption characteristics [49]. However, this solution does not employ multi-core parallelism, which has its own benefits as we mentioned previously.

Goal and Challenges. Our objective is to execute parallelizable tasks over multiple cores intermittently and efficiently by considering the dynamic environmental power. We seek for an intermittent computing runtime that *dynamically reconfigures* the underlying multicore architecture and adjusts the amount of parallelism, thereby benefiting from data and task parallelism when input power is high and avoiding aggressive computations when incoming power does not support them. However, intermittent computing introduces several unique factors that affect the efficiency of multicore architectures. In particular, the store and recovery overheads (e.g., due to checkpoints [44]) upon frequent power failures is the primary dominant factor that might shade the benefits of multicore architectures.

Contributions. We introduce AdaMICA (Adaptive Multicore Intermittent Computing), an intermittent computing runtime that supports parallel intermittent multicore computing and provides the highest degree of flexibility of programmable general-purpose multiple cores. AdaMICA adaptively switches to the best multicore configuration considering the dynamic input power. Therefore, it allows an intermittent system to benefit from workload parallelization, thereby running complex computational tasks faster to increase systems throughput and decrease end-to-end delay while considering the energy availability. Figure 1 provides a high-level overview of the AdaMICA operating principle. In this figure, when the incoming solar power is high at a certain point in time, the energy buffer can be charged rapidly. At that point, AdaMICA configures the underlying multicore architecture to use the strong power in the most optimal way to quickly complete a task and move on to the next. However, when the power weakens and cannot support the fast depletion of energy from the capacitor, AdaMICA decides to downgrade the system configurations to utilize the available resources more consummately. Our evaluation has shown that AdaMICA significantly increases the system's throughput (1.38× on average) by dynamically scaling the underlying architecture while considering variations in unpredictable harvested energy.

Table 1. A comparison of the main features of AdaMICA with the relevant runtime architectures. AdaMICA is the only intermittent computing runtime that fully utilizes parallelism, adapts multicore systems to environmental dynamics, and provides missing programming support for multicore parallelism.

Intermittent Runtime	Parallelism Support	Parallel Programming	Programming Flexibility	Adaptation Support
Dewdrop [15], Mementos [66], DINO [51], QuickRecall [41], Chain [18], Hibernus++ [13], Ratchet [75], Alpaca [54], Mayfly [34], HarvOS [14], Chinchilla [55], InK [78], Coati [70], TICS [44], ImmortalThreads [79]	Single core ✗	No ✗	General purpose (non-parallel) applications ✓	Computational workload scaling (SW-only) ✓
Camaroptera [59], Zygarde [40], Coala [57], CatNap [56], ePerceptive [58], Rehash [11], Julienning [32]	Single core ✗	No ✗	General purpose (non-parallel) applications ✓	Task degradation (SW-only) ✓
Spendthrift [53], D^{2}VFS [6]	Single core ✗	No ✗	Not general purpose (application specific platform) ✗	Voltage and frequency scaling ✓
Neuro.ZERO [49]	Accelerator ✓	Low-level accelerator instructions ✗	Not general purpose (application specific platform) ✗	Switching between a core and an accelerator ✓
SONIC & TAILS [30]	Accelerator ✓	Low-level accelerator instructions ✗	Not general purpose (application specific platform) ✗	No adaptation ✗
AdaMICA (this work)	**Multicore** ✓	**Runtime support** ✓	**General purpose (parallel) applications** ✓	**Activating and deactivating cores** ✓

To summarize, our contributions include:

(1) **Multicore Intermittent Computing.** We introduce the missing software support that enables, for the first time, parallel intermittent computing over multiple cores.

(2) **Power-scaling Runtime.** We introduce the first intermittent runtime that provides the missing parallel programming language constructs and adaptively reconfigures the multicore system concerning the environmental power strength.

2 BACKGROUND AND MOTIVATION

Making embedded devices battery-free reduces the cost of their maintenance, shrinks the size of the devices, and saves nature from an overabundance of toxic and corrosive chemicals. Batteryless devices rely solely on the energy harvested from environmental sources, such as sunlight, radio waves, or vibration. When power is obtainable, such a device slowly collects energy in a small energy buffer such as a capacitor (*charging cycle*). On reaching a certain voltage threshold in the buffer, the device starts computing a task, fast discharging the energy storage (*discharging cycle*). In such circumstances, data-sensing applications run intermittently and must be resistant to frequent power failures.

2.1 Intermittent Computing Approaches

Today's off-the-shelf batteryless computing devices are extremely resource-constrained. For example, MSP430FR family MCUs [36], the mainstream MCUs in existing batteryless systems [22], operate at 1-16 *MHz* frequency and include a mixed volatility memory architecture (1-4 *kB* SRAM and 32-256 *kB* FRAM (Ferroelectric RAM)). Upon power failures, SRAM and internal hardware state (e.g., registers) of the microcontroller are cleared, thereby both the current state and the intermediate results of a program are lost, which, in turn, violates both forward progress of computation and consistency of the memory [50]. Several approaches, discussed by Lucia et al. in [50], have been proposed to develop power failure tolerant programs. Today's software-based intermittent computing solutions are either *checkpointing systems*, e.g., [5, 44], or *task-based systems*, e.g., [18, 78]. Checkpointing systems save the current state of a program in non-volatile memory (i.e., FRAM) via manually or automatically placed checkpoints. Task-based systems require developers to decompose their programs into atomic re-executable tasks, at the end of which the critical data are recorded to the non-volatile memory. There are also hardware-based studies, e.g., [13], which require dedicated hardware that continuously monitors the supply voltage and checkpoints when the supply voltage drops below a threshold. It is worth mentioning that FRAM exhibits low-power characteristics (requires up to 10× lower voltage) and faster write performance (faster up to 1000×) compared to flash memories [38]. More importantly, FRAM has almost inexhaustible write endurance (has 10^{15} write endurance, e.g., 150000 write operations per second will lead to approximately 211 years lifetime). This feature is crucial since intermittent computing requires frequent access to non-volatile memory due to frequent power failures. Therefore, MCUs with only embedded flash memory wear out in a short time, which makes them unsuitable for intermittent computing.

2.2 Adaptive Power and Energy Scaling

A unique property for intermittent energy harvesting systems is that *the energy comes for free* but with a *time-varying nature*. Intermittent systems must adapt to frequently changing environmental energy conditions to exploit the available energy more efficiently. However, due to the limited capacitance of the energy buffer, the excess energy cannot be stored [53]. Thus, when the devices' power consumption is lower than the input power, the devices can draw more power to execute more computational loads. When the input power is lower than their power consumption, these devices can use the incoming power more conservatively. Some intermittent computing studies, e.g., [11, 56, 78], have proposed software solutions that respond to environmental energy availability. The main idea is to scale down the computational loads, e.g., sacrificing the accuracy but reducing the energy demands by considering the input power. There are also studies, e.g., [6, 12, 53], that have proposed voltage and frequency-based scaling (DVFS) for intermittent computing systems. For instance, [12] considers matching the instantaneous power consumption of the device with the input harvested power. Similarly, [53] and [6] consider maximizing the forward progress by aggressively consuming energy when it is available. They employ dynamic voltage and frequency scaling to adjust the operating frequency of the microcontroller concerning the voltage level of the capacitor and incoming power.

2.3 The Need for Multicore Intermittent Computing

As of now, the majority of intermittent computing systems comprise single-core processing elements. Improving the performance of single-core systems further is difficult due to closely approached fundamental limits. Power constraints prevent further increase in operating frequency, instruction-level parallelism is well-exploited, and memory hierarchy scarcely improves. Moreover, there is a trend to shift machine learning (ML) inference to edge devices or sensors since communication costs order-of-magnitude more energy and time compared to the cost of local sensing and computation on a resource-constrained device. If we want energy harvesting intermittent edges to execute modern machine-learning-based applications in a reasonable time, these devices should exploit data and task parallelism in inference computations.

Recent studies have shown the feasibility of ML inference on the batteryless edge devices, e.g., [30, 31, 49, 58], by software frameworks and hardware accelerators. However, accelerators support limited parallelism (via their rigid instruction set), and they are generally task-specific. Moreover, it is difficult to program accelerators, and they lose their configuration upon power failures. The general-purpose programmable multicore processors are task-agnostic and flexible solutions for IoT [29, 43, 69]. However, there is no prior study on intermittent computing that has exploited parallelism via multicore processing yet. Therefore, the next step is to benefit from multicore parallelization and execute intermittent inference more efficiently over multiple cores to decrease end-to-end delay and increase throughput.

2.4 Adaptive Architectural Scaling

Multicore architectures introduce a certain complexity to a system despite their parallelization benefits. Task scheduling, task partitioning, and data sharing make multicore systems more challenging. Intuitively, one can think resource-constrained devices operating intermittently by relying only on ambient energy cannot tolerate this overhead and complexity. Frequent power failures that can violate computation progress and memory consistency might even make things worse and complicated. Despite these difficulties, intermittent systems can adapt to frequently changing environmental energy conditions to exploit parallelism when it is beneficial. More specifically, when input power is high, energy harvesting devices can benefit from data and task parallelism as long as the environmental energy availability permits. For example, energy harvesting devices can execute high-cost parallelizable tasks faster by dynamically scaling the number of cores, i.e., reconfiguring multicore architecture on the fly. This strategy is the key insight behind our work—adaptive parallelism increases the system throughput and significantly decreases the end-to-end delay.

2.5 Our Differences

We compare the main features of AdaMICA with state-of-the-art intermittent runtimes in Table 1. Former studies have already provided software adaptation on single-core systems by scaling the computational load or degrading tasks concerning environmental power dynamics. Two studies [6, 53] propose hardware adaptation by tuning the operating voltage and frequency in response to input power, but they are also limited to single-core systems. The system in [49] provides hardware adaptation at runtime by switching between different compute units that differ in performance and energy consumption. Both [30] and [49] exploit accelerators, which are cumbersome to program, application-specific, and support parallelism for certain operations only. Unlike all these studies, AdaMICA fully utilizes parallelism over multiple cores, provides the missing software support to enable intermittent parallel processing, and adapts multicore systems to environmental dynamics.

3 MODELING MULTICORE INTERMITTENT SYSTEMS

Before presenting AdaMICA, we analyze the impact of the main parameters that affect the end-to-end performance of intermittent execution of tasks on multicore devices. To represent the energy and performance models (whose parameters are summarized in Table 2), we take as a basis the models proposed in [71] and [24].

3.1 Intermittent Computing Dynamics

Energy Storage. The charging time (T_{charge}) depends on both incoming power (P_{input}) from ambient energy harvester device and capacitance of an energy buffer (E_{cap}) (in other words, the amount of energy can be stored in a buffer):

$$T_{charge} = E_{cap}/P_{input}, \quad E_{cap} = (V_{cap}^2 * C)/2, \tag{1}$$

where V_{cap} denotes the maximum voltage of the capacitor and C denotes the capacitance.

Power Failures. How often a system will be interrupted by power failures ($N_{pow-fail}$) is inversely proportional to E_{cap} feeding the energy demand of the entire intermittent system (E_{system}):

$$N_{pow-fail} = E_{system}/E_{cap}. \tag{2}$$

Forward Progress and Recovery. We consider an intermittent computing system that exploits a checkpointing approach that is triggered just before the power failure [13]. Upon a power failure, the device charges the energy buffer, boots the system again, and restores backed up state in checkpoint data, restarting execution from the

Table 2. List of the input parameters of the model.

Parameter	Description
F_{seq}	Sequential fraction of a task
E_{cap}	Energy stored in a capacitor
$T(E)_{task}$	Time(Energy) per execution of a task
$T(E)_{monitor}$	Time(Energy) per spend to monitor power and voltage levels
$T(E)_{com}$	Time(Energy) per communication
$T(E)_{checkpoint}$	Time(Energy) per a checkpoint
$T(E)_{boot}$	Time(Energy) per a boot

interrupted point. Therefore,

$$N_{checkpoint} = N_{charge} = N_{boot} = N_{pow-fail} \tag{3}$$

holds since the number of checkpoints and number of restores ($N_{checkpoint}$) become equal to number of power failures, number of charges (N_{charge}), and number of boots (N_{boot}).

3.2 Multicore Energy Consumption

Assumptions. We consider a *shared memory* multicore system where a main-core (*mn*) executes the sequential part of a given task, then distributes parallelized part of the task among secondary (*sd*) cores and equally contributes to parallel execution. The energy spent for checkpointing and booting is controlled by the number of cores (n) since a single power failure leads to multiple checkpoints and boots in a multicore system. In addition, energy (E_{com}) depleted by the main-core when communicating with the secondary cores contributes to the total system energy consumption. This communication overhead incorporates commands passing, cores waking up, and memory copying since we assume a shared memory multicore system in this work. Moreover, the model allows for the amount of energy gathered during different stages of intermittent execution by subtracting it from the amount of energy spent for a certain stage, as described in [71].

Task Energy Cost. We show in Eq. (4) that energy consumed by a multicore system to execute a task encompasses additional parameters to ensure that multicore overhead is factored in:

$$E_{task} = E_{task-mn} + E_{task-mn-sd} + E_{task-sd} * (n-1) + E_{task-sd-mn} * (n-1) - E_{harvested}, \tag{4}$$

where n denotes the number of cores. Table 3 presents the description of each product in Eq. (4).

System Energy to Execute Task. In order to model the energy required to execute a task with energy consumption of E_{task}, we take into account the energy consumed for inter-core communication (E_{com}), for power and voltage level monitoring ($E_{monitor}$), for checkpoints ($E_{checkpoint}$), and for booting (E_{boot}):

$$\begin{aligned} E_{system} &= E_{task} + E_{monitor} + E_{com} * (n-1) + (E_{checkpoint} * n + E_{boot} * n) * (E_{system}/E_{cap}) \\ &= \frac{E_{task} + E_{monitor} + E_{com} * (n-1)}{1 - \frac{(E_{checkpoint}+E_{boot})*n}{E_{cap}}}, \end{aligned} \tag{5}$$

Table 3. Description of the products in Eq. (4)-(9).

Product	Description
$E_{task-mn}$	Energy spent on the sequential part of the code executed by the main-core
$E_{task-mn-sd}$	Energy spent on the parallel part of the code executed by the main-core
$E_{task-sd}$	Energy spent on the parallel part of the code executed by the secondary cores
$E_{task-sd-mn}$	Energy consumed by the secondary cores when waiting for the main-core to complete the sequential part of the code
$E_{byte-mem}$	Energy consumed for reading and writing operations to memory per byte
$E_{byte-send}$	Energy consumed for data sending operation per byte
$E_{system-boot}$	Energy spent on system initial booting process
E_{sample}	Energy consumed for sampling power and voltage levels
$E_{harvested}$	Energy harvested during particular operation
N_{byte}	Number of bytes required to be sent, stored, or restored
N_{sample}	Number of power and voltage samplings per task

where n denotes the number of cores. All that components depend on the amount of bytes needed to be manipulated and also consider the energy harvested simultaneously:

$$E_{monitor} = (E_{sample} * N_{sample}) - E_{harvested}, \tag{6}$$

$$E_{com} = (E_{byte-send} * N_{byte}) - E_{harvested}, \tag{7}$$

$$E_{checkpoint} = (E_{byte-mem} * N_{byte}) - E_{harvested}, \tag{8}$$

$$E_{boot} = E_{system-boot} + (E_{byte-mem} * N_{byte}) - E_{harvested}. \tag{9}$$

The description of all equation parts can be found in Table 3.

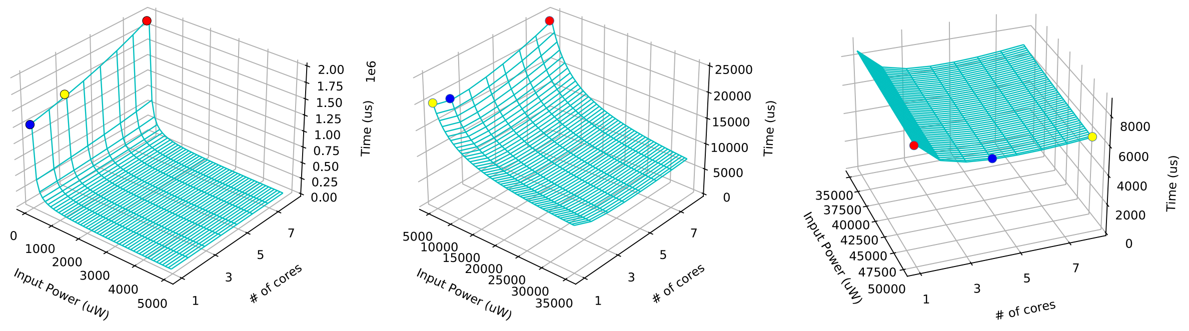
Eq. (5) confirms that an intermittent system has to spend energy for checkpointing and booting upon each power failure presented in Eq. (3). The last part of Eq. (5) indicates that E_{cap} cannot be less than or equal to the total energy consumed by checkpointing and booting processes, otherwise, we will have zero or negative value as a denominator. The capacity of the capacitor determines how often an intermittent system must be interrupted to recharge, i.e., how frequently a system spends energy for checkpoints and boots.

3.3 Multicore System Performance

Task Execution Time. The performance of parallel execution obeys the Amdahl's law. Thus, task execution for a multicore system is distributed among the available processors, while allocation depends on the task's sequential fraction (F_{seq}), which is executed exclusively by the main-core. The remaining part ($1 - F_{seq}$) of the task can be run on several processors in parallel:

$$T_{task} = T_{task-mn} * F_{seq} + T_{task-sd} * \frac{1 - F_{seq}}{n}, \tag{10}$$

where $T_{task-mn}$ denotes the task execution time of the main-core and $T_{task-sd}$ denotes the task execution time of the secondary cores.



(a) Input power range 0–5000 (b) Input power range 5000–35000 (c) Input power range 35000–50000

Fig. 2. Three parts of the performance dependence on the incoming power and the number of cores working on a task. Red, yellow, blue dots represent high, medium, and low point of the marked line respectively.

End-to-end Delay. The multicore intermittent system's end-to-end delay can be modeled as:

$$\begin{aligned} T_{system} &= T_{task} + T_{monitor} + T_{com} + (T_{charge} + T_{checkpoint} + T_{boot}) * (E_{system}/E_{cap}) \\ &= T_{task} + T_{monitor} + T_{com} + (E_{cap}/P_{input} + T_{checkpoint} + T_{boot}) * (\frac{E_{task} + E_{monitor} + E_{com} * (n-1)}{E_{cap} - (E_{checkpoint} + E_{boot}) * n}). \end{aligned} \quad (11)$$

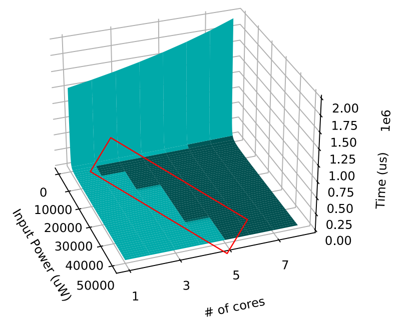
The end-to-end delay depends on execution time (T_{task}), charging time (T_{charge}), checkpointing time ($T_{checkpoint}$), booting time (T_{boot}), power monitoring time ($T_{monitor}$), and the inter-core communication time (T_{com}) that takes into account time to switch a secondary core from a low power mode to the active mode. The last part of Eq. (11) is derived from the middle part by substituting E_{system} with the value from Eq. (5). The model shows that the capacitance of an energy buffer (E_{cap}) affects the performance of a system. Note that checkpointing and booting for several cores can be performed simultaneously (since we assume a voltage-level triggered checkpointing) and it is not needed to multiply their time overhead by the number of cores. Furthermore, T_{com}, $T_{checkpoint}$, and T_{boot} directly depend on the number of bytes needed to be processed, while simultaneous charging when performing different stages only affects the number of power failures and is thus expressed in the energy model (Eq. (5)).

4 USER EXPLORATION OF MULTICORE INTERMITTENT EXECUTION

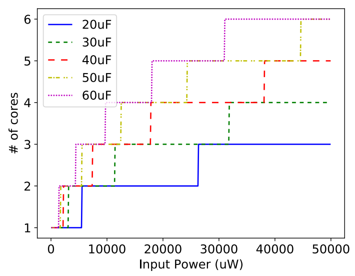
Most of energy harvesting devices can harvest energy meanwhile they are executing tasks [19], contributing to performance and execution progress. Eq. (11) shows that E_{cap}, n, and the *strength of incoming power* are the main parameters affecting the overall performance of a system.

The availability and stability of environmental power are unforeseeable. However, when the system encounters a high power availability in the environment, it can increase the throughput by adapting to the changing environmental conditions. For instance, multiple cores can exploit task and data-level parallelism of emerging applications (e.g., fast ML inference) when the ambient power (which comes for *free*) is strong enough to charge the energy buffer quickly. Therefore, when the environment supports:

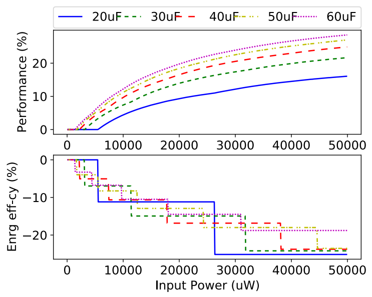
$$\min_{\substack{n \in [1, n_{av}] \\ P_{input} \in \Re_{\geq 0}}} (\frac{T_{systemMC}}{T_{systemSC}}) < 1, \quad (12)$$



(a) Red rectangle frames the area where the surface reaches its lowest point.



(b) Highest performance with different capacitors. Red dashed line reflects the line in the red rectangle in Fig. 3a.



(c) Multicore versus single-core energy efficiency.

Fig. 3. Comparison of the speed up and the energy efficiency drop of multicore intermittent systems.

where n_{av} denotes the number of available cores, $T_{systemMC}$ denotes the performance of a multicore intermittent system, and $T_{systemSC}$ denotes the performance of a single-core intermittent system, then it is desirable to exploit the parallel execution.

Based on the model from Section 3, we developed a tool[1] implemented in Python that helps the developer to check at an early stage if multicore deployment is beneficial for a particular application. We performed simulations with this tool to understand the performance of executing a computational task concerning a different number of cores, input power levels, parallelization factors, and capacitor sizes. We present the parameter values used in our simulations in Table 4 (row for the Section 4). It is worth mentioning that these values are collected from our testbed setup presented in Section 6.

4.1 Input Power and Number of Cores

We gradually increased the input power from 50 to 50000 μW using a fixed capacitor value of $C = 40\mu$F and assumed that 50% of the task is parallelizable. The resulting system's behaviour is split into three parts and presented in Figures 2a, 2b, and 2c. As seen from these wire-frames, a change in the input power changes the system performance markedly. For instance, when the input power is very low (Figure 2a), the single-core system (blue dot) outperforms any multicore system running the same task (e.g., yellow and red dots). However, when the input power reaches 5000 μW (Figure 2b), some multicore configurations (e.g., dual-core system, blue dot) benefit from the faster energy buffer charging time and complete a task earlier than the single-core system (yellow dot). With further increasing the input power (Figure 2c), the single-core system (red dot) can no longer compete with multicore systems, and even an 8-core system (yellow dot) can outperform the single-core system by 20%. We can conclude that under different environmental conditions, different system configurations become optimal. We extracted the optimal number of cores, as visualized in Figure 3a. The *stepped line* in the red rectangle presents the points where the surface reaches the local or global minimums, i.e., the number of cores for a multicore system with the fastest execution time.

[1]Available on GitHub (https://github.com/tinysystems/adamica)

4.2 Capacitance and Sequential Fraction

Figure 3b shows the optimal number of cores, in terms of system performance, for various input power and capacitance. Note that the larger energy storage a system has, the more cores it can activate to increase the performance. It is worth noting that with some capacitance-input power level combinations, single-core computing is more beneficial than switching to the multicore mode. Figure 3c (top) compares the performance of each optimal number of cores configuration, presented in Figure 3b, against the single-core implementation. At the bottom part of Figure 3c, we show the energy efficiency drop with increasing the number of active cores. This additional energy overhead is reasonable since the system activates extra technical resources to use probably short-term high free input power periods to execute as many tasks as possible. Having an intermittent system with a capacitor of 60 μF, at the high input power, we can speed up the execution by around 30%, consuming about 20% more energy. We omit our simulation results concerning the sequential fraction, for the interest of space. We conclude that the shorter the sequential part of a task, the sooner a system can switch to multicore mode to speed up.

4.3 Obtaining Minimum Power Requirements.

System designers can use our exploration tool to observe the minimum power requirements to switch between the multicore configurations. By approximating or directly measuring the parameters listed in Table 2, users can observe the minimum power required to activate other cores and benefit from parallelism. As an example, Figure 3(b) clearly shows that with a capacitor size of $20\mu F$, an input power should be more than 5 mW to turn on the secondary core and benefit from parallelism. Having a priori information on the energy harvesting environment and the minimum/maximum harvestable power, designers can also make a decision on the maximum number of cores that can be deployed in the system.

5 ADAMICA: ADAPTIVE MULTICORE COMPUTING

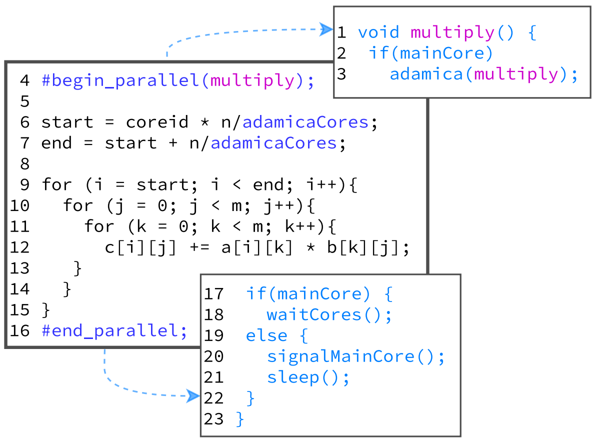
This section presents AdaMICA[2]—a dynamically configurable multicore runtime for intermittent computing that can perform computational tasks at different performance and efficiency levels by switching between various modes concerning the input power from the environment. AdaMICA can control multiple homogeneous or heterogeneous full-fledged computing units that share memory and operate at different frequencies. The main core, where AdaMICA resides, is responsible for bootstrapping, monitoring input power, deciding on appropriate configurations, activating secondary cores, and assigning parallel subtasks among them. Figure 5 gives an overview of the multicore intermittent execution flow where additional actions taken by AdaMICA are shown in light blue. We explain all the blocks in the following subsections.

5.1 Programmers' View

Software developers provide information on the parallelizable tasks in the form of program metadata. They mark the parallelizable blocks of their programs by using `begin_parallel` and `end_parallel` macros (Figure 4a, lines 4–16). The `begin_parallel` macro creates a void C function with a given name (line 1) and inserts the call to the runtime to reconfigure the multicore architecture (lines 2–3). The `end_parallel` macro inserts the code that handles the necessary communication actions between the main core and the secondary cores (lines 17–23). Secondary cores signal the main core when they finish their computational tasks. The main core waits for notification from secondary cores and progresses after all secondary cores finish their parallel computation.

Programmers also need to provide the values of parameters listed in Table 2, which capture the energy cost of several operations as well as the parallel computational tasks. The simplest way to obtain this information

[2]Available on GitHub (https://github.com/tinysystems/adamica)



```
1 void multiply() {
2  if(mainCore)
3    adamica(multiply);
```

```
 4 #begin_parallel(multiply);
 5
 6 start = coreid * n/adamicaCores;
 7 end = start + n/adamicaCores;
 8
 9 for (i = start; i < end; i++){
10   for (j = 0; j < m; j++){
11     for (k = 0; k < m; k++){
12       c[i][j] += a[i][k] * b[k][j];
13     }
14   }
15 }
16 #end_parallel;
```

```
17  if(mainCore) {
18    waitCores();
19  else {
20    signalMainCore();
21    sleep();
22  }
23 }
```

(a) A parallelizable code block. The programmer uses begin_parallel and end_parallel to indicate parallizable code sequences. AdaMICA runtime activates the cores based on the available power to increase the throughput.

```
void adamica(fn_ptr *func) {
  // sample input power
  uint32_t power = sampleADC();
  // decide on the number of cores
  numCores = DMT(func, power);
  // activate secondary cores
  activate(numCores, func);
}
```

(b) AdaMICA runtime pseudo-code that reconfigures the multicore architecture.

Fig. 4. AdaMICA programming model and abstractions.

is utilizing the energy profiling tool, EnergyTrace [27], which demonstrates high measurement accuracy and resolution. It is worth mentioning that it is possible to automatize these measurements, as depicted in [28].

In analogous to any parallel programming technique, AdaMICA requires software developers to write their code in a form that can be split equally among the active cores. AdaMICA follows the principle idea of OpenMP [16], a flexible and open-source shared-memory programming model actively used in embedded systems for parallel programming [69]. In our implementation, programmers use `adamicaCores` macro (Figure 4a, lines 6 and 7) at runtime to get the number of cores activated by AdaMICA. Due to the shared memory architecture, each core should compute only a portion of the output, and they should not overwrite the outputs of each other.

5.2 Runtime Execution Flow

Figure 5 presents the main flow of the AdaMICA runtime. Upon the start of the system, the main-core boots first and then starts the execution of the program ①. When the main-core hits a parallel code block ②, it samples the input power ③ and calls AdaMICA decision-making procedure ④ via `adamica` function by providing the parallel block's address as a parameter (Figure 4a, line 3). At this point, AdaMICA activates the secondary cores to exploit the parallel execution if it is beneficial based on the input power ⑤. In this case, the main-core and the activated secondary cores execute the equally partitioned computational load ⑥. After the main-core finishes its execution, it waits for all other cores to finish their executions ⑦. Secondary cores signal the main-core after they finalize their computation and transition to sleep mode to save power until they are activated again. The main-core continues executing the sequential part of the program before hitting the next parallel part or going to sleep mode ⑧.

5.3 Just-in-time Checkpoints and Handling Power Failures

AdaMICA exploits just-in-time (JIT) checkpointing. It monitors the energy buffer (via voltage monitoring circuitry [13]) to check if the voltage level is below a certain voltage threshold. The voltage monitoring circuitry

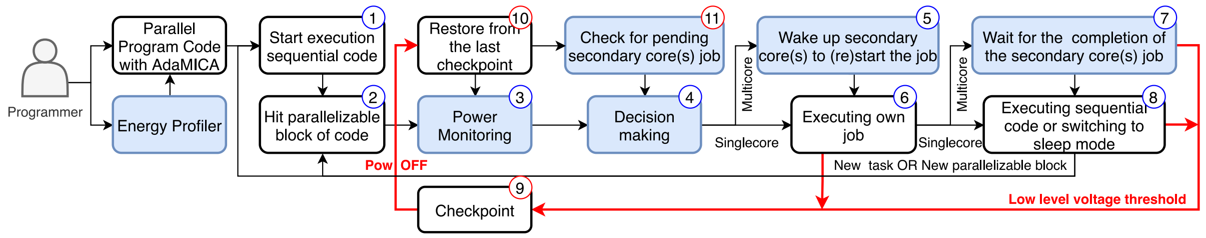

Programmer
Parallel Program Code with AdaMICA
Energy Profiler
Start execution sequential code
1
Hit parallelizable block of code
2
Restore from the last checkpoint
10
Power Monitoring
3
Check for pending secondary core(s) job
11
Decision making
4
Multicore
Singlecore
Wake up secondary core(s) to (re)start the job
5
Executing own job
6
Multicore
Singlecore
Wait for the completion of the secondary core(s) job
7
Executing sequential code or switching to sleep mode
8
New task OR New parallelizable block
Pow OFF
Checkpoint
9
Low level voltage threshold

Fig. 5. AdaMICA operating overview. Light blue boxes are additional actions helping AdaMICA to successfully operate on a multicore system.

signals the main-core when this condition holds, and the backup actions start. This checkpointing strategy requires fewer backups and does not suffer from wasted work, i.e., the work needs to be repeated after power restores. Moreover, the dedicated voltage monitoring hardware also allows for tracking input power, which is essential for making decisions to reconfigure multicore architecture.

AdaMICA requires that the energy stored in the capacitor when reaching the specified low voltage threshold is sufficient to checkpoint all available cores and guarantee execution progress. The low voltage threshold is specified by a system designer in advance, considering the mentioned condition. However, AdaMICA requires several low voltage thresholds because checkpointing overhead increases with the number of active cores. The developer specifies these constant threshold values in a dedicated data structure of AdaMICA used upon decision making (Section 5.5). Note that threshold interruption is implemented using a window comparator of an analog-to-digital converter (ADC) to monitor analog signals without any CPU interaction. Both the lower and upper bounds of the window are adjustable just by rewriting the values in particular registers [37].

Upon backups, the main-core, in addition to saving its own computational state, signals the other nodes to perform the checkpoint ⑨. The red input line to ⑨ shows that power failures can occur while executing a task (⑥, ⑦) or staying in a low power mode ⑧. This suggests that the energy-harvesting system is equipped with a sufficient capacitor that ensures both seamless execution of steps ①-⑤, ⑩, ⑪ and forward computation progress. When the capacitor is charged and the system can restart its operation, AdaMICA restores from the last checkpoint ⑩ and proceeds with power monitoring ③ and checking for a pending secondary core(s) job ⑪. The latter information is collected during step ④ and is used to keep track of the secondary cores that did not complete their parts of a task before the last power failure. This insignificant in size (core ID and subtask parameters) but essentially important information helps AdaMICA to ensure the completion of the interrupted subtasks by reactivating them on reboot. For example, if AdaMICA detects pending secondary core jobs ⑪ and decides that there is enough input power to resume multicore mode ④, it wakes up necessary cores and continue executing the task. If power is not strong enough, AdaMICA sequentially executes all the pending jobs in a single-core mode.

5.4 Communication Among the Cores

Depending on the multicore architecture, the main-core can wake up the secondary cores using special instructions or serial communication protocols such as SPI. AdaMICA dedicates distinct regions in shared memory for command or data exchange among the cores. For instance, AdaMICA records the address of the parallel code block to be executed by the secondary cores in a dedicated data structure in memory. When a secondary core is activated (Figure 4b, line 7), it reads the address value to understand which parallel block to execute.

When the main-core completes its part of the computation, it waits for the interrupt signal from secondary cores indicating the completion of their job (Figure 4a, line 20). Upon receiving this signal, the main-core updates the pending jobs data.

5.5 Decision Making and Adaptive Multicore Reconfiguration

AdaMICA uses a *decision-making table* (DMT) at runtime to decide on the number of cores to activate. The DMT, which maps the input power levels to the corresponding optimal system configurations, is filled in by AdaMICA at runtime. By applying the values for the parameters from Table 2 and the energy cost of the parallel tasks to the models presented in Section 3, AdaMICA generates the DMT entries on-demand. An alternative strategy is to use our tool (presented in Section 4) to generate DMT by using the parameters specified in Table 2 and the energy cost of the parallel tasks, and to upload DMT into the device memory along with the program code.

Figure 4b presents the AdaMICA reconfiguration pseudo-code triggered when it hits a parallel code block. AdaMICA samples input power using an ADC (line 3) and uses it to compute the corresponding entry of the DMT. It obtains the desired multicore configuration, which is mapped to the corresponding input power level in the DMT (line 5). The main-core then activates the required number of secondary cores and assigns their subtasks (line 7). If the parallelization will bring no benefit (based on the returned configuration in line 5), the main-core starts executing the task sequentially. It is worth mentioning that the calculation of the optimal configuration for each power level is performed only once. If AdaMICA encounters the same power level, it grasps the previously computed value from the DMT. AdaMICA keeps the DMT's structure very simple. It consists of two columns - power level and the number of active cores. The computation for each entry is also composed of simple arithmetic operations (as depicted in Eq. 12) and depends on the number of available cores, which is unlikely greater than eight for modern embedded low-power systems.

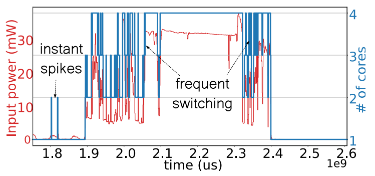
AdaMICA can exploit *one-shot* and *dynamic* decision making procedures. In the *one-shot* decision-making approach, AdaMICA obtains the desirable multicore configuration at the beginning of the parallel task (via line 5), switches mode immediately, and the selected configuration remains unchanged till the end of the parallel task block or until the next power failure. This approach is simple and less costly in terms of implementation. The simulated behavior of the on-shot made decisions (using the parameter values in Table 4, row Section 5.5) is depicted in Figure 6a, representing a slice of the real irradiance power trace for indoor/outdoor activity sourced from [46]. As seen, AdaMICA reacts promptly, strictly following the predefined configuration in the DMT.

The *dynamic* approach can be more efficient since it involves making more informed decisions. Using the information aggregated in the past, it tries to "anticipate" the behavior of an ambient energy supply, bypassing the mode switching on short-term power spikes and drops. This implementation avoids activating unnecessary system configurations, e.g., when input power encounters an instantaneous power drop, the DMT recommends switching to the single-core mode. *History prediction* [1, 9] is a suitable algorithm for making dynamic decisions, due to its simplicity and has reasonably good accuracy [2]. The key idea of this algorithm concerning our dynamic decision-making approach is to slide the window of predefined length over P_{input} values and predict the next value by the following calculation:

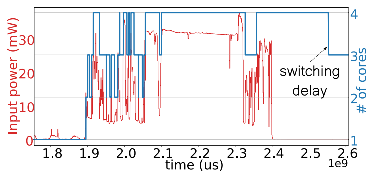
$$P_{predict} = \frac{W_L * P_{current} + P_{past}}{W_L + 1}, \tag{13}$$

where W_L denotes the user-defined length of the sliding window, $P_{predict}, P_{current}, P_{past}$ denote input power predicted, current, and past values, respectively. To have a dense history of input power levels, AdaMICA samples these values upon restoring from power failure and reaching parallelizable code blocks.

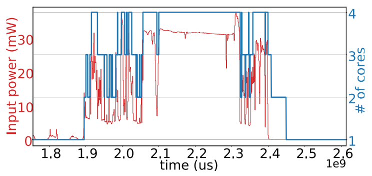
Figure 6b reflects the AdaMICA's dynamic reaction with the fixed W_L value in Eq. 13. Although the algorithm introduces fewer mode switching, it suffers from significant delay in some cases. However, online adjustment of the sliding window length with respect to error rate helps to balance between the number of switches and the



(a) One-shot approach reacts immediately.



(b) Dynamic approach with fixed W_L.



(c) Dynamic approach with adjustable W_L.

Fig. 6. The behavior of different decision-making approaches.

reaction pace (Figure 6c). We use the following control rule to update W_L:

$$W_L = \min(W_{max}, \max(W_{min}, W_L - \alpha * \frac{d}{dt} err), \tag{14}$$

$$err = (P_{current} - P_{predict})^2, \tag{15}$$

where α is the control gain, err is the squared error, W_{max} and W_{min} are the predefined max and min values of W_L, respectively.

5.6 AdaMICA Overheads

Inevitably, AdaMICA introduces memory and computation overheads to the system. The DMT is stored in non-volatile memory, and its size depends on both the application and the underlying multicore architecture. For example, the DMT of a dual-core system with a fixed-size capacitor and a few parallelizable code blocks requires less space compared to that for a quad-core system with a re-configurable energy buffer and many parallelizable code blocks.

AdaMICA needs to sample the output of the harvester using an ADC. These measurements can be extremely cheap in terms of energy by using the recent advancements in circuits [77]. The decision-making requires simple computations to decide on the multicore configuration, which introduces almost negligible time and energy overheads.

Due to the shared memory architecture, the communication cost among the cores is also negligible. On the other hand, the wake-up time and energy cost of the secondary cores are architecture-specific and constant. According to our observations, these values do not have a significant impact on the system's performance, as the next section shows that AdaMICA is superior compared to the rigid systems configurations.

6 EVALUATION AND RESULTS

In this section, we demonstrate how AdaMICA enables adaptive multicore architectural scaling at runtime—for the first time, we demonstrate successful multicore intermittent execution and architectural adaptation. We evaluated AdaMICA using various benchmarks and applications in our real-world setup. We found that AdaMICA increases the throughput of parallelizable code blocks significantly compared to rigid multicore configurations that do not respond to power availability.

6.1 Benchmarks Performance

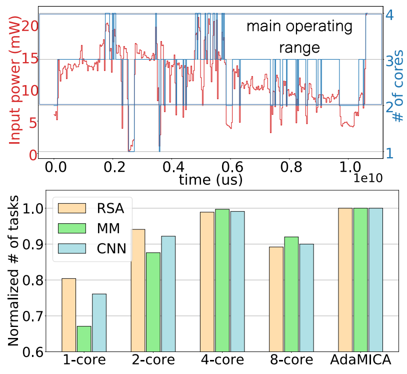
To verify AdaMICA's efficiency, we created an emulation setup with fully controlled system input parameters. Such emulation allows us to effortlessly, rapidly, and repeatedly discover the realistic behavior of the proposed system under a variety of conditions.

Table 4. Values for the experimental parameters used in different sections of the article. *T - ms*, *E - μJ*

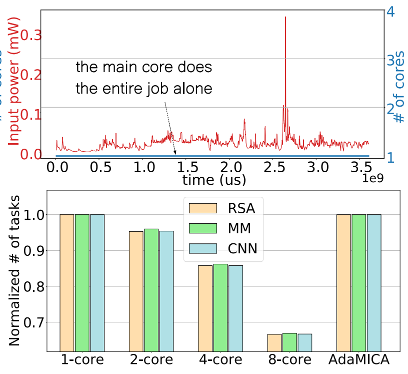
Section in the article	F_{seq}	E_{cap}	*task*		*monitor*		*com*		*checkpoint*		*boot*		*W*		α	*n*
			T	*E*	*T*	*E*	*T*	*E*	*T*	*E*	*T*	*E*	*max*	*min*		
Section 4	0.5	90-260	2200	18040	0.15	0.02	0.05	0.01	1.155	11.435	1	0.367	600	200	0.1	8
Section 5.5	0.5	190	2200	18040	0.15	0.02	0.05	0.01	1.155	11.435	1	0.367	600	200	0.1	4
Section 6.1 (CNN)	0.3	190	2200	18040	0.15	0.02	0.05	0.01	1.155	11.435	1	0.367	600	200	0.1	8
Section 6.1 (MM)	0.1	190	180	1440	0.15	0.02	0.05	0.01	1.155	11.435	1	0.367	600	200	0.1	8
Section 6.1 (RSA)	0.4	190	265	2147	0.15	0.02	0.05	0.01	1.155	11.435	1	0.367	600	200	0.1	8
Section 6.2 (MNIST)	0.3	432×10^1, 216×10^3	2200	18040	0.15	0.02	0.05	0.01	1.155	11.435	1	0.367	600	200	0.1	2
Section 6.2 (ESC-10)	0.4	432×10^1, 216×10^3	3100	25420	0.15	0.02	0.05	0.01	1.155	11.435	1	0.367	600	200	0.1	2
Section 6.2 (CATalogue)	0.2	432×10^1, 216×10^3	3500	28700	0.15	0.02	0.05	0.01	1.155	11.435	1	0.367	600	200	0.1	2
Section 6.3 (Tennis)	0.4	648×10^1	1800	14800	0.15	0.02	0.05	0.01	1.155	11.435	1	0.367	600	200	0.1	2
Section 6.3 (Animals)	0.35	130×10^2	6500	53300	0.15	0.02	0.05	0.01	1.155	11.435	1	0.367	600	200	0.1	2
Section 6.3 (Plants)	0.2	130×10^2	5200	42640	0.15	0.02	0.05	0.01	1.155	11.435	1	0.367	600	200	0.1	2

6.1.1 Target Platform and Multicore Emulation. To our best knowledge, there are no multicore MCUs with internally embedded FRAM on the market. Multicore MCUs with embedded flash memory are not suitable for intermittent computing due to the high energy requirements, low speed, and limited write endurance of the flash memory, as we mentioned in Section 2. To mimic the shared-memory multicore computing architecture, we used MSP430FR5994 [37], which is the de facto ultra-low-power MCU for intermittent computing systems. This MCU is single-core, and it contains 256 *KB* and 8 *KB* of embedded FRAM and SRAM memory, respectively. We set the MCU operating frequency to 16 *MHz*, which is the highest frequency that results in the most energy-efficient performance (we provide more details in Section 6.5 on the voltage-frequency related issues). To emulate shared-memory multicore execution on a single-core device, we employed a simple yet effective strategy. For the parallelizable blocks within the program code, we performed the calculations of the secondary cores in advance and embedded their results in the device memory. When the main-core executes its part, it merges its result with the pre-calculated results from the other cores staying in memory. We take into account the parameters introduced in Section 3 to have the fine-grain approximation of the multicore execution time and energy consumption.

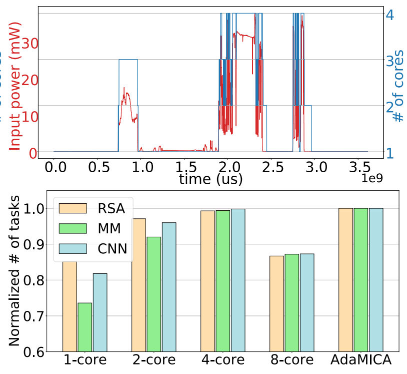
6.1.2 Energy Harvesting and Power Failures. We used pre-recorded irradiance power traces to emulate the energy harvesting process. We chose three different traces sourced from a specific community dataset [46]: *constantly low*, in a city at night; *constantly high*, during a car ride on a sunny day; *mixed*, during indoor-outdoor daily routine. Using power traces gave us full control and made our experiments more observable. Since the power traces give information in $\mu W/cm^2$, we considered a 10 cm^2 solar panel with 10% efficiency. We set the energy buffer of size 40 μF. It is worth mentioning that the size and efficiency of the solar panel should be carefully selected depending on the application needs and target environment, since these parameters affect the strength of power harvested, which is directly proportional to the system performance (see Eq. 11). For example, the specified above parameters are sufficient to generate up to 30 *mW* power, which is enough to switch to the 4-core architecture (see Figure 2b). However, if a smaller solar panel is required by the application, one can select a panel with higher efficiency, which can reach 30% today [60]. For emulating the energy harvesting and consumption behavior, we set a timer with a period of 5 *ms* to generate probes. At each probe, we calculate the harvested



(a) Car travel.



(b) Night city.



(c) Indoor/outdoor.

Fig. 7. AdaMICA dynamic configuration and throughput results for different irradiance power traces.

and consumed energy, based on our model described in Section 3 by using the power trace, the system's power consumption, and the time passed from the previous probe. We use this calculation to emulate the charging and discharging of the energy buffer. When the stored energy reaches the lower threshold, we mimic a power failure that introduces checkpointing, charging, and booting procedures.

6.1.3 Experimental Parameters. We reserved 0.5 KB of memory to fit the entire DMT for the input power range from 50 to 50000 μW in increments of 100 μW. The DMT is stored as a lookup table to speed up the search for AdaMICA decisions. Table 4 (rows for the Section 6.1) shows the settings of the experimental parameters. All the values in this table are calculated using the device's datasheet information and the experiments performed on the device by profiling the energy consumption using the EnergyTrace [27] software tool.

6.1.4 Dynamic Decisions. The dynamic decisions for the adaptive system are made upon reaching a parallelizable block. AdaMICA calculates on-the-fly or uses the DTM (if the calculation has already been done) and decides on the number of cores to activate. Based on the decision made by AdaMICA, we recompute the time and energy to complete the parallelizable block: we take into account inter-core communication, we count the number of power failures, multiply them by the time and energy for checkpointing, charging, and booting to obtain the actual task execution values for actual time and energy consumption during multicore execution.

6.1.5 Benchmarks. We run our emulations for three different benchmarks that represent different sequential code fractions: (i) LeNet-5 inference for image recognition [47] adapted to a 28×28 pixels input handwritten digit image from MNIST database [23] (CNN, F_{seq}=0.3); (ii) 32×32 matrix-matrix multiplication (MM, F_{seq}=0.1); (iii) parallelized RSA encryption described in [10] (RSA, F_{seq}=0.4). The topology of the CNN is sourced from [30], which shows 99.00% of accuracy. Running each benchmark, we use the number of tasks the system can execute for the duration of a trace as a performance metric.

6.1.6 Results. Figure 7 demonstrates our emulation results for the three power traces we use, comparing our adaptive system against four fixed-configuration intermittent systems: single-core and three multicore systems (2, 4, 8 cores). Figure 7a shows that having a high level of environmental power, AdaMICA spends most of the time in the multicore mode, switching within the region between two and four cores. The throughput bars show that AdaMICA performs as well as the best multicore option presented (4-core fixed) for all applications, outperforming the single-core system by 32%.

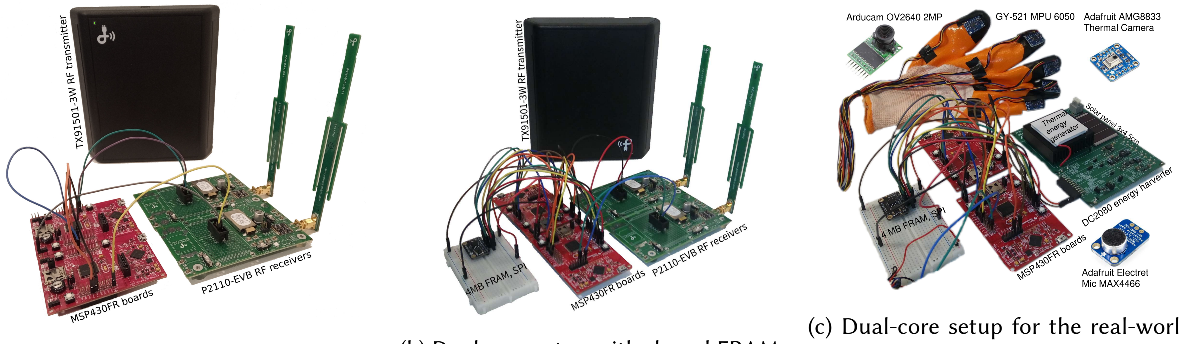
When the irradiance power is too low (Figure 7b), the adaptation mechanism keeps the system in the low energy consumption mode by running tasks on a single core. With the constantly low power input, AdaMICA just monitors periodically the power level and lets the main-core perform the whole task as an optimal option. Therefore, AdaMICA outperforms each of the multicore systems presented, when the speed-up for an 8-core system can reach 47%.

Figure 7c demonstrates a typical example of the frequent fluctuation of incoming power in a short period of time, which leads to the frequent switching of the system mode. However, the dynamic decision-making algorithm described in subsection 5.5 reduces the frequent mode switches. In an environment where the incoming power changes frequently within a wide range of levels, AdaMICA also shows better results. The throughput bars demonstrate that our adaptive architecture surpasses all the compared systems, for instance, inferencing more images in a given amount of time. In such circumstances, the performance of a single-core system can be improved by 37%.

6.2 Real-World Energy-Harvesting Evaluation

In this subsection, we demonstrate the effectiveness of AdaMICA in our real-world energy-harvesting setups. We present two evaluation setups. The first one emulates the real dual-core system with multi-ported shared memory. In this setup, the cores have all necessary data in advance in their internal memory allowing for simultaneous access for the same data. The second setup reflects the other reality in which the cores need to access a real shared memory presented by external FRAM. However, this memory chip is single-ported, so it can only be accessed sequentially. Our setups are closer to the conventional multicore shared-memory approaches, with two differences: our cores have no cache memory and conduct commands via a separate SPI bus interface. Hardware platforms with an optimized memory bus and memory hierarchy can decrease the multicore communication overhead, increasing the benefits of AdaMICA adaptation. To make picture clear, we compare SPI and AMBA, a parallel bus protocol widely used in embedded systems [8], by considering 32-bit data transfer. By using the power and energy requirements reported in [61], we calculate that a system spends 64× less time and 26× less energy to send 32-bit data via AMBA. Besides, data caching (which is not an exclusive feature of a multicore systems) is beneficial for ML applications, which are well-structured for data reuse. Having a memory hierarchy with cache can also be beneficial to decreasing not only memory access overhead but also checkpointing overhead in intermittent computing, as presented in [73]. All these improvements reduce the time and energy overhead when multiple cores are active, which would lead to even an increased throughput via AdaMICA adaptation since these overheads are directly proportional to the system performance (see Eq. 11).

6.2.1 Benchmarks. We evaluate AdaMICA on three CNN applications. The first application recognizes a written digit and receives as input a 28×28-pixel grayscale image of handwritten digits from the *MNIST* dataset [23]. The CNN model (8×10^3 parameters) includes two convolutional layers with ReLU and MaxPool functions and two fully connected layers followed by the output layer. The second CNN classifies ten environmental sounds. The *ESC-10* dataset [63] contains 5s audio clips sampled at 44.1 *KHz*. We use the 1s audio downsampled five times. This network model (55×10^3 parameters) comprises three convolutional layers and one fully connected. The third CNN model is trained to recognize the presence of an animal in the given image. The model (14×10^3 parameters) comprises four convolutional layers and one fully connected. The images of animals are sources from the Camera *CATalogue* dataset [76], which consists of 520K images of 55 animal species from 750 cameras in South Africa. The sizes of the images are large and not unified. To fit an image into the MCU's memory, we move the target object (animal) to the center by cropping the image and then downsample it to 32×32 pixels. The applications have different sequential fractions: MNIST - 30%, ESC-10 - 40%, and CATlogue - 20%. All three CNN models are pruned and quantized to 8-bit integers.



(a) Dual-core emulation setup.

(b) Dual-core setup with shared FRAM.

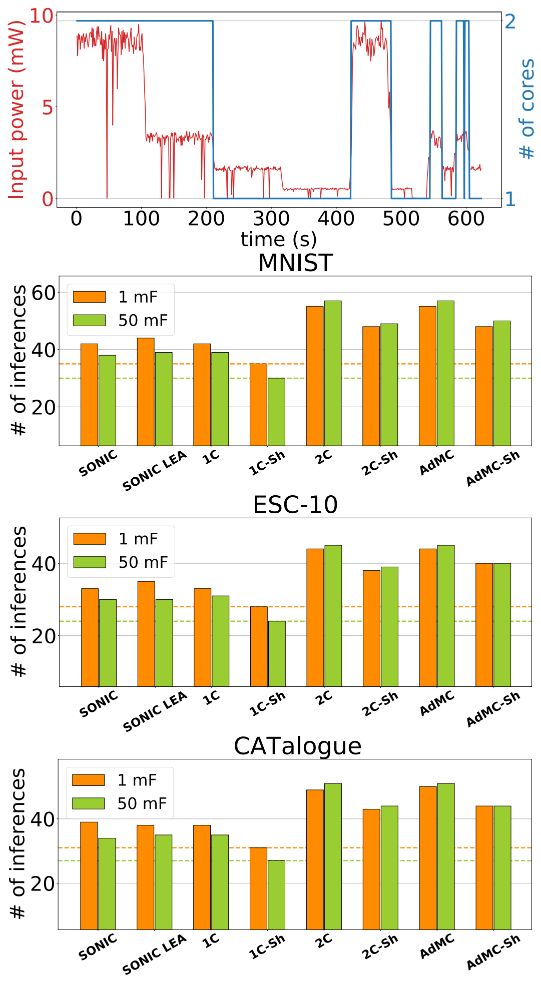
(c) Dual-core setup for the real-world application.

Fig. 8. Components of our dual-core experimental setups.

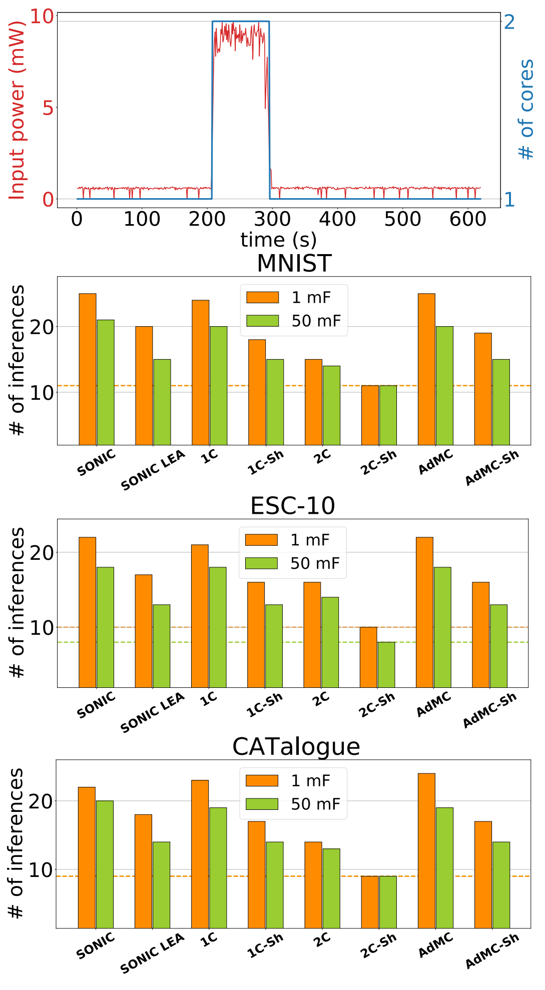
6.2.2 Baselines. We compare AdaMICA with four distinct implementations targeting intermittent computing: SONIC, SONIC LEA, Single-core fixed, Dual-core fixed. SONIC [30] is the state-of-the-art single-core solution that performs inference on resource-constrained devices operating intermittently, including MSP430FR series microcontrollers. SONIC is a task-based intermittent system reinforced by three techniques to optimize inference execution: loop continuation, ordered buffering, and sparse undo-logging [30, see Section 6]. SONIC LEA (named as TAILS in [30]) exploits the low-energy vector accelerator (LEA) found on MSP430FR series microcontrollers. SONIC LEA benefits from the hardware acceleration and can improve inference tasks [30, see Section 7]. Our two other baselines, Single-core fixed and Dual-core fixed, are just-in-time checkpointing intermittent systems (see Section 5.3) with no adaptation to ambient power.

6.2.3 Experimental Setup and Multicore Emulation. We refer to the results of this setup as 1C, 2C, and AdMC with multi-port emulated memory access for single-core fixed, dual-core fixed, and AdaMICA configurations, respectively. We also use this setup to implement SONIC and SONIC LEA. Figure 8a shows all components of our experimental setup. As an energy harvesting module, we exploit the Powercast TX91501-3W [20] RF transmitter to emit the energy and the P2110-EVB [21] RF receiver with two different capacitors (1 *mF* and 50 *mF*) to harvest, store, and convey that energy to our computing unit. The computing unit combines two MSP430FR5449 [37] evaluation boards acting as a dual-core system. All data is captured using either the EnergyTrace [27] and the logic analyzer [26]. We mimic a dual-core system with shared memory using two MSP430 MCUs operating at 16 *MHz*. To emulate the behavior of *multi-port* shared memory, we embed the computational results of parallelizable blocks in advance in the device memory. Therefore, when a core finishes its dedicated part of the computation, it can find the results of the other core in memory. We set one of the boards as the main-core. The other device (i.e., the secondary core) stays in a low-power mode, waiting for activation. The main-core wakes up the secondary core by sending commands via SPI communication.

Real Multicore Setting with Shared External FRAM. We refer to the results of this setup as 1C-Sh, 2C-Sh, and AdMC-Sh with shared sequential memory access for single-core fixed, dual-core fixed, and AdaMICA configurations, respectively. To experiment even closer to the realistic dual-core system, we augment our previous setup with the shared external Adafruit SPI FRAM of 4 *MB* (Figure 8b). The external SPI FRAM is single-port, which allows only *sequential access* to the shared non-volatile memory and limits the real performance of our multicore setup. However, this is the only feasible realistic multicore demonstration using easily accessible off-the-shelf components.



(a) Fluctuating Input Power.



(b) Low Input Power.

Fig. 9. Throughput results for running CNN inference on our real energy-harvesting setups. SONIC, SONIC LEA, 1C, 2C, AdMC belong to the setup with emulated multi-ported memory access (Figure 8a). 1C-Sh, 2C-Sh, AdMC-Sh belong to the setup with a real external single-ported shared memory (Figure 8b). The dashed lines indicate the worst throughput for the corresponding capacitor configuration.

Intermittent Execution. A single energy harvester powers both boards. We set the appropriate lower threshold values (1.81 and 1.82 V for single-core and dual-core configurations, respectively) for the capacitor to guarantee uninterruptible checkpoints for both cores. When the threshold is reached the main-core signals to the secondary core to start the backup process together. Upon the completion of the recharging process, the main-core follows the sequence of actions indicated in Figure 5 to continue the interrupted execution.

Power Measurements. The main-core uses its onboard ADC to sample and convert the capacitor voltage on the P2110-EVB RF receiver. An additional P2110-EVB is required to sample the environmental power strength, since

Powercast receivers do not allow simultaneous monitoring and energy harvesting. We positioned two P2110-EVB harvesters as identically as possible and used the second harvester to monitor the input power.

6.2.4 Results. We change the distance between the RF transmitter and harvester to experiment with input power fluctuations. For each capacitor setting, we run our experiment for 10 minutes. The results collected from our energy-harvesting setup are very consistent with those of the benchmarks. Figure 9 presents the results for both setups with emulated shared memory (SONIC, SONIC LEA, 1C, 2C, AdMC) and with real external shared memory (1C-Sh, 2C-Sh, AdMC-Sh). The figure shows that when power is mostly strong (Figure 9a), SONIC performs as well as our 1C configuration with just-in-time checkpointing, while SONIC LEA performs slightly (around 5%) faster than SONIC. However, when power is low (Figure 9b), the performance of SONIC LEA decreases due to the memory movement overhead introduced by the accelerator at each power failure, while SONIC can outperform the 1C configuration by about 6% due to improved checkpointing overhead in SONIC.

Figure 9 shows that AdaMICA successfully switches between single-core and dual-core architectures, adapting to environmental power. By doing so, AdaMICA outperforms both fixed architectures for different energy buffer sizes. The system with AdaMICA can recognize, for example, 67% more sound clips compared to the single-core system in the ESC-10 application when power is high (Figure 9a). The figure also shows that the number of inferences increases for the setup with multi-port emulated memory. This is due to the decreased memory access latency, while the external memory creates a bottleneck by its single-ported (sequential) access in this particular setup. Also, note that the reason for the throughput decrease for the single-core deployments with the larger capacitor is the increase of charging time, which, in most cases, coincides with low incoming power periods. When incoming power is mostly low (Figure 9b), the fixed dual-core system cannot be beneficial anymore. Under such conditions, AdaMICA decides to switch to dual-core mode only when the input power is feasible while keeping the system in the less energy-consuming mode for the other time. This strategy enables 89% more image inferences for the CATalogue dataset.

The comparison of the above setups (Figures 8a and 8b) highlights the beneficial feature of a multicore system with shared multi-ported memory, where all cores can access memory simultaneously. The overall result demonstrates that AdaMICA can successfully tolerate different environmental cases, improving throughput.

6.3 Real-world Applications

We demonstrate the performance of AdaMICA on three real-world **batteryless** ML applications: gesture recognition, wild world tracking, and plant health monitoring.

Gesture Recognition. It is one essential ability of ML that helps humans in various domains, such as human to computer interaction, gaming, medical operations, training, physical rehabilitation, and others. We focus on recognizing three patterns of tennis strokes performed by a trainee (serve, groundstroke, and volley) to help trainees to practice their skills of stroking on their own [45].

Wild World Tracking. It is a key technique for ecologists to monitor changes in the environment [76]. In this application, we focus on animal tracking, using a camera and microphone, whereas the intermittent system is powered by two sources: the animal's body heat and the sunlight.

Plant Health Monitoring. It can significantly increase agricultural productivity by promptly forecasting disease, nutrient deficiencies, or drought [68]. For our experiment, we consider an image-based plant disease diagnostic batteryless system equipped with two cameras: color (RGB) and thermal.

6.3.1 Experimental Setup. We used the hardware setup shown in Figure 8c during the experimental evaluation of our applications.

Multicore Setup and Sensors. In this setup, we used the same realistic shared external SPI FRAM setting as in the previous subsection, where inputs and outputs of common CNN layers reside in FRAM. As cores, we use the same MSP430 boards running at 16 *MHz*. The GY-521 is an I2C-featured standard MPU-6050 device that combines a 3-axis gyroscope and a 3-axis accelerometer on the same silicon die. We used this sensor (mounted on a glove) for the gesture recognition. For image and sound in wild world tracking and plant health monitoring applications, we considered the Arducam OV2640 2MP RGB camera [7], the Adafruit AMG8833 thermal camera [4], and the Adafruit Electret MAX4466 microphone with amplifier [3].

Energy Harvesting Sources. We used Analog Devices DC2080 [25] harvester that can generate 1.72-3.3 *V*, which comprises a 1.5 *mF* capacitor, a 13.5 cm^2 solar cell and a thermal energy generator (TEG). We powered the gesture recognition and plant health monitoring applications using solar cells in this harvester. Besides, in the plant monitoring application, more power is needed to activate the second core and turn on the thermal cameras to detect the plant health more precisely. For this purpose, we used the Powercast RF transmitter and receiver. For the wild world tracking application, we assume that the tracking device is attached to an animal body which allows the device to be recharged from the animal's body heat. Therefore, we used energy harvesting via TEG for this purpose. During the day, the available sunlight enables energy harvesting using solar cells.

6.3.2 Application Scenarios and CNN Models. Using TensorFlow Lite, we trained different CNNs targeting our applications. We achieved different accuracy levels for CNN models: 96% for gesture recognition, 85% for sound classification, 81% for animal image classification, and 92% for plant RGB and thermal image classification.

Tennis Stroke Classification. To infer the tennis gestures (serve, groundstroke, and volley), we utilized the same CNN model that we used in Section 6.2 for the recognition of handwritten digits (MNIST). To match the output of an MPU-6050 sensor to the input of the CNN model, we transformed the data from gyroscope and accelerometer to a 2D image by performing three main steps: (i) estimation of rotational orientation, having 3D angular orientation as output; (ii) transformation of the 3D data into 2D positional coordinates; (iii) drawing lines between those coordinates on a 2D 28×28 pixels image.

Animal Tracking. The sound is used for the preliminary classification of surrounding animals and the RGB image for the final confirmation. The sound and image are captured simultaneously. If an animal is detected by the sound, the image is then processed by another CNN, otherwise the image is discarded. We used the ESC-10 and CATalogue datasets and their CNN models described in Section 6.2 for training by considering ten different animal sounds and images.

Plant Monitoring. We used two different cameras: RGB and thermal. Periodically, the device turns on the RGB camera to detect if there is any plant disease. We used our RF transmitter to deliver more power so that the device can turn on its thermal camera to detect the plant health more precisely. To evaluate this application, we used the dataset of rice plant images [80] and trained the same CNN model that we used for animal image classification with a slight modification in the output layer.

6.3.3 Software Architecture. We split the application software into two functional parts: sensing and computing. The sensing part collects data from a sensor, preprocess them, and delivers data to the input layer of the corresponding CNN. We perform the sensing that ensures the sampling and preprocessing of data in one discharging cycle, limiting its energy burst to E_{cap} (see Table 4). Following this limitation allows us to have a seamless sensing process. The sensing part of the tennis stroke recognition application collects accelerometer and gyroscope samples, preprocess them, and outputs data in gray-scale 28×28 image format. We set the sampling rate of both sensors to 200 *Hz* and start the sampling process only when the capacitor is full. We collect data for 330 *ms*, spending half of the energy in the capacitor while using the remaining energy for data preprocessing and storing the input for the computing part. The sensing part of the animal tracking application requires more energy since it is responsible for collecting and preprocessing the sound and image. The sound must be transformed to the spectrogram, while the RGB image must be downsampled. The sampling 1s audio with a highly energy-efficient microphone and extracting spectrogram consumes 15% of the 1.5 *mF* capacitor. However, the energy consumed by the sensing and preprocessing of the image exceeds the energy stored in the capacitor 1.5×. Although we select

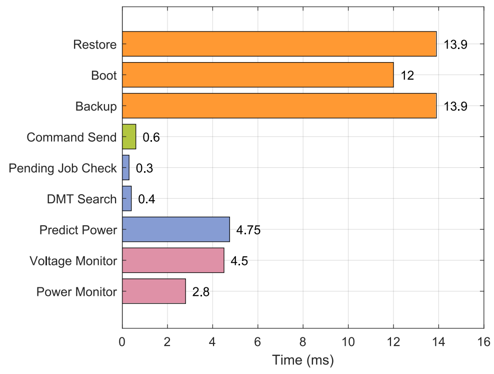
Table 5. Performance of AdaMICA in different applications. (ms)

Environment	SONIC LEA	Single-core	Dual-core	AdaMICA
Tennis stroke				
Indoor	17106	15460	18050	15520
Outdoor (shade)	11760	12380	13560	12515
Outdoor (sun)	8925	10080	7860	7900
Overall	37791	37920	39470	**35935**
Animal tracking				
Night (TEG)	21900	21540	25250	21120
Day (shade)	19005	20560	22010	20010
Day (sun)	15230	17380	10400	9900
Overall	56135	59480	57660	**51030**
Plant monitoring				
Day (cloudy)	15520	15180	19150	14750
Day (cloudy + RF)	16520	17150	12290	11980
Day (sunny)	16130	16900	12030	11650
Overall	48170	49230	43470	**38380**

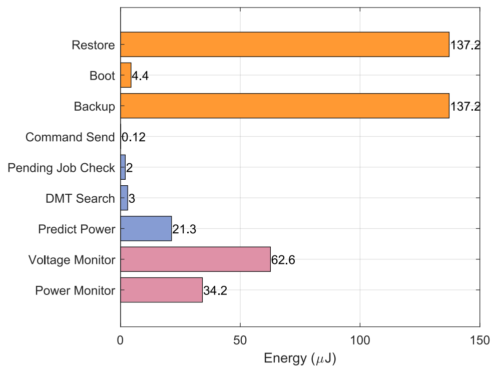
the lowest resolution for the RGB camera (160×120 px), each pixel has three channels (colors), and the camera has a relatively slow (8 MHz) SPI interface and requires 300 mW in active mode. To ensure the uninterrupted execution of the sensing part of this application, we add the second 1.5 mF capacitor, which is used only for the sensing part. Thus the sensing starts only when both capacitors are full. The plant monitoring application is also manageable with two capacitors since the thermal camera introduces relatively low power consumption (15 mW) and resolution (8×8 px) and requires no image preprocessing. The computing part of all applications, directed by AdaMICA, receives and delivers the output of the sensing part to CNN that performs the classification.

6.3.4 Evaluation and Results. Table 4 (rows for the Section 6.3) contains all the energy profiling measurements for AdaMICA in our experiments. Table 5 presents the performance results of the execution of four intermittent computing systems with external FRAM (SONIC LEA, Single-core fixed, Dual-core fixed, and AdaMICA) under different environmental conditions for the three applications. For each scenario, we conducted 20 experiments. We measure the time that the systems take to produce a single classification. The results show that AdaMICA aligns with the most performing system configuration and can reach 75% of speed up (for tracking animals in the sunlight). As seen, SONIC LEA outperforms the single-core and dual-core solutions for some scenarios due to its specialization in CNN applications. For example, when we place the experimental setup in a shaded area outdoor (see Outdoor (shade) and Day (shade) in Table 5), input power increases, reducing the number of power failures, which improves the performance of SONIC LEA being ahead of other architectures. In this environment, input power is still not sufficient to fully benefit from the dual-core parallel execution. Therefore, AdaMICA mostly stays in the single-core mode, which is circa 7% slower than the LEA implementation. However, when power is low (e.g., indoor, night, or cloudy day), SONIC LEA cannot improve the single-core solution due to the significant memory transfer checkpointing overhead of LEA. Moreover, when power is high (e.g., on sunny days), SONIC LEA lags behind the dual-core system because of the limited set of supported operations. For example, functions such as ReLU, MaxPool, or downsampling cannot be performed using LEA, where the dual-core system can easily split these tasks among cores. Furthermore, the rightmost column of the table shows that the dynamic decisions of AdaMICA allow for increasing the performance of ML applications by the hardware reconfiguration at runtime.

Based on the results in Sections 6.2 and 6.3, we estimate the average values of throughput and latency improvements that AdaMICA provides. To evaluate the throughput, we performed twelve sets of experiments (see Figure 9), six experiments for each setup (see Figures 8a and 8b). To evaluate the latency, we performed nine



(a) Time overhead.



(b) Energy overhead.

Fig. 10. AdaMICA overheads.

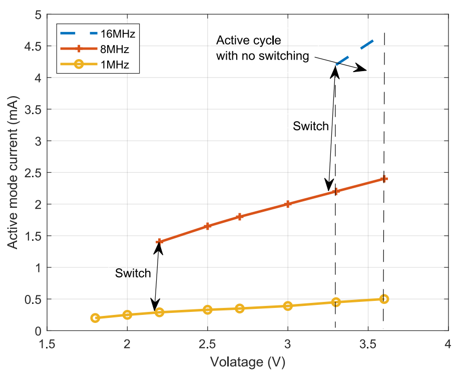
sets of experiments (see Table 5). For both throughput and latency, we calculated the ratio of AdaMICA values to the worst values in each set. Summing all these ratios and dividing the sum into the number of experiments, we calculated the mean values of the improvements. We conclude that AdaMICA can improve the intermittent system's performance, increasing the throughput by 52% on average and decreasing the latency by 31% on average.

6.4 AdaMICA Overheads

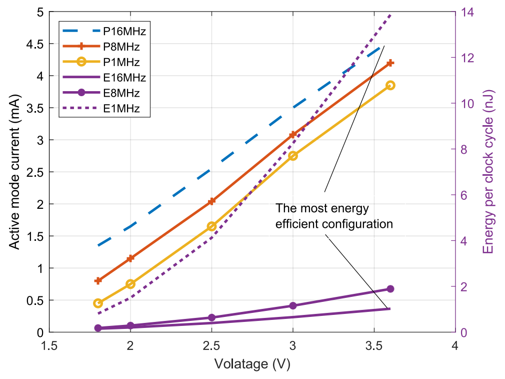
In our real-world application setup, we measured the time and energy overheads of the AdaMICA dynamic decision procedure by using EnergyTrace. Figure 10 shows the breakdown of both overheads. We distinguish four groups of time and energy consumers concerning AdaMICA. The *monitoring group* contains power monitoring to make a decision and voltage monitoring to control the lower threshold of the capacitor. This group makes 24% of the energy overhead due to expensive ADC operations, but it barely affects the performance since done with no CPU intervention. The next is the *decision-making group* that contributes 6.5% to the energy overhead and includes dynamic power prediction, DMT search, and pending job check. The power prediction is the most consuming part because AdaMICA needs to compute the next power level depending on the history window (Equation 13). The DMT search performs model calculation first few iterations and then uses already calculated data from the DMT. Both DMT and the pending job data are stored as a hash table to increase search performance and energy efficiency. The third group is the *communication group* that is reduced to sending appropriate commands to the secondary core via SPI and pushing necessary bytes of data to the shared memory to initiate multicore computation. The last one, *checkpointing group*, comprises the most demanding components: backup, boot, and restore. The checkpointing makes 79% of time and 69% of energy overheads of AdaMICA. However, the total overhead of AdaMICA has a cost of 2.3% and 2.2% of the whole task time and energy consumption, respectively, which is acceptable concerning performance benefits.

6.5 AdaMICA versus Dynamic Voltage and Frequency Scaling

In recent work [6], the authors has proposed their power-aware DVFS technique on MPS430G2553 MCU where the maximum operating frequency (16 *MHz*) is limited by the voltage range from 3.3 to 3.6 *V* (Figure 11a). This restriction limits the active cycle of the MCU operating at the highest frequency, so switching to a lower frequency domain prolongs the active cycle of the MCU and utilizes the capacitor energy more rationally.



(a) MSP430G [6].



(b) MSP430FR (this work).

Fig. 11. Voltage range and current consumption for different types of MCU.

For our evaluation, we employ an MSP430FR5994 MCU that can operate at any available frequency (1-16 *MHz*) within the voltage range from 1.8 to 3.6 *V*. Figure 11b presents the current and voltage measurements performed using a multimeter. As can be observed, the most efficient configuration in terms of energy spent per clock cycle is the highest frequency level, which is 16 *MHz*. Therefore, we fixed the frequencies of all cores at 16 *MHz*, which allowed the MCUs to operate in the most energy-optimal configuration while entirely using the stored energy. Thus, there was no need to integrate DVFS techniques in our implementation

As a side note, systems that exploit only DVFS, e.g. [6], are not scalable. In other words, with growing task demands and input power, the system hits the maximum frequency, missing the excess energy that does not fit into its energy buffer. Moreover, these approaches do not address the parallelism introduced in a task, which can be easily handled by a multicore system. AdaMICA, in contrast, can scale up the system capability when input power allows but also restrains the system energy consumption when necessary. DVFS can provide system developers with additional leverage to increase the efficiency of multicore embedded platforms, enabling adaptation within the cores if the operating frequencies are limited by different voltage levels.

7 DISCUSSION AND FUTURE WORK

We now compare AdaMICA against existing solutions and discuss how it could coexist with them, eliminating current limitations.

Data-level and Instruction-level Parallelism. Systems presented in [30, 49] exploited data or instruction-level parallelism using specific accelerators to hide the low performance of single-core embedded processors. AdaMICA cores can also exploit these onboard accelerators. Moreover, AdaMICA can leverage vector, superscalar, or any other specific computational unit to further improve multicore system's performance.

Programmer Burden. As explained in Section 5, the programmer needs to provide application metadata, explicitly specifying parallelizable blocks in the code. From this perspective, the main duty of a multicore intermittent system programmer is the same as for a multicore constantly powered system. However, the programmer has the burden of the task energy profiling and DMT generation, which can be handled by a software tool automatically. We leave this feature for our future work.

Decision-making Approach. We considered the dynamic decision-making approach during our evaluations since this technique is more flexible and has greater scientific value. Applying more sophisticated control theory-based and ML algorithms to implement the dynamic decision-making approach leads to new key research challenges, which we suggest as interesting follow-up work.

Other Platforms. AdaMICA suits any energy-harvesting MCUs equipped with non-volatile memory. The MSP430FR family is a unique platform since it includes an embedded non-volatile memory (FRAM) and an optimized bus between the MCU and FRAM. To use AdaMICA on the off-the-shelf multicore MCUs, it is necessary to use an external FRAM memory. However, this will not be as efficient as in the embedded FRAM case. AdaMICA brings about the need for multicore platforms with embedded FRAM.

8 RELATED WORK

Our work addresses crucial gaps in the intermittent computing parallelism, which are inevitable since the multicore models are already approaching energy-harvesting systems. Although most of the related work was discussed in Section 2, it should be noted that there are other studies on adaptive and parallel systems.

Processing-in-Memory Accelerators. MOUSE [67] is an intermittent accelerator built on top of computational RAM (CRAM) utilizing the emerging idea of spintronics-based processing-in-memory (PIM). The authors demonstrate that spintronics-based logic operations are inherently idempotent and can be safely used by intermittent computations. CRAM does not need explicitly specified checkpoints, since the memory cells are non-volatile. PIM and specific memory cell layout introduce different levels of parallelism (column-parallelism, tile-parallelism). While time-, energy-, and area-efficient, MOUSE has no easy accessible programming interface, and it has a specific instruction set. As of now, there is no high-level programming language and a compiler to program CRAM-based systems. An intermittent accelerator operating on the same principle and having the same drawbacks [64] has been proposed as a co-processor to a generic MCU powered by a battery. However, both solutions target operation-specific accelerators based on the processing-in-memory, which is orthogonal to our work covering general-purpose ultra-low-power MCUs.

Workload Switching Solutions. Switching workload between several compute units that differ in performance and energy consumption is another strategy to respond to changing input power [52]. However, this technique does not exploit parallelism in any form. The authors in [72] propose to enable intermittent computing on high-performance systems-on-chip, focusing mostly on booting and checkpointing operations. They cover very briefly multicore deployment aspects, omit other important parameters, such as the incoming power strength or the energy buffer size.

Multicore Reconfiguration. While GreenArrays [62], Core Fusion [39], and REDEFINE [33] propose reconfigurable many-core architectures for low-power embedded systems in various fields, none of them target intermittent computing.

9 CONCLUSION

For intermittent computing systems, execution performance and parallelism are crucial properties gaining importance. Existing energy-harvesting devices are extremely limited in performance and introduce little flexibility. Initial steps have begun to be taken by the intermittent community in the direction of parallelism, but are still not sufficient enough for battery-free devices to take full advantage of it. To boost the parallelism adoption, we proposed AdaMICA, which is the first adaptive runtime that enables multicore parallel intermittent computing. We evaluated AdaMICA based on the benchmarks emulated on a real MCU. To validate the emulation results, we run real-world ML applications on our real energy-harvesting setup. Our evaluation results showed that AdaMICA outperforms single-core and multicore fixed-configured intermittent systems by 52% on average due to the successful adaptation to changing environmental conditions.

ACKNOWLEDGMENTS

We thank the anonymous reviewers of MICRO 2021, ASPLOS 2022, IPSN 2022, and IMWUT for their valuable comments and feedback that led us to improve our paper.

REFERENCES

[1] Cristinel Ababei and Nicholas Mastronarde. 2014. Benefits and costs of prediction based DVFS for NoCs at router level. In *2014 27th IEEE International System-on-Chip Conference (SOCC)*. IEEE, 255–260.

[2] Cristinel Ababei and Milad Ghorbani Moghaddam. 2018. A survey of prediction and classification techniques in multicore processor systems. *IEEE Transactions on Parallel and Distributed Systems* 30, 5 (2018), 1184–1200.

[3] Adafruit. 2022. *Adafruit AMG8833 IR Thermal Camera*. Technical Report. Adafruit. https://www.adafruit.com/product/1063

[4] Adafruit. 2022. *Electret Microphone Amplifier - MAX4466 with Adjustable Gain*. Technical Report. Adafruit. https://www.adafruit.com/product/3538

[5] Saad Ahmed, Naveed Anwar Bhatti, Muhammad Hamad Alizai, Junaid Haroon Siddiqui, and Luca Mottola. 2019. Efficient intermittent computing with differential checkpointing. In *Proceedings of the 20th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems*. 70–81.

[6] Saad Ahmed, Qurat Ul Ain, Junaid Haroon Siddiqui, Luca Mottola, and Muhammad Hamad Alizai. 2020. Intermittent Computing with Dynamic Voltage and Frequency Scaling.. In *EWSN*. 97–107.

[7] Arducam. 2022. *Arducam Mini Module Camera Shield with OV2640 2 Megapixels Lens*. Technical Report. Arducam. https://www.uctronics.com/arducam-mini-module-camera-shield-w-2-mp-ov2640-for-arduino-uno-mega2560-board.html

[8] ARM. 2022. *AMBA Specifications*. Technical Report. ARM. https://www.arm.com/architecture/system-architectures/amba/amba-specifications

[9] Raid Zuhair Ayoub and Tajana Simunic Rosing. 2009. Predict and act: dynamic thermal management for multi-core processors. In *Proceedings of the 2009 ACM/IEEE international symposium on Low power electronics and design*. 99–104.

[10] Md Ahsan Ayub, Zishan Ahmed Onik, and Steven Smith. 2019. Parallelized RSA Algorithm: An Analysis with Performance Evaluation using OpenMP Library in High Performance Computing Environment. In *2019 22nd International Conference on Computer and Information Technology (ICCIT)*. IEEE, 1–6.

[11] Abu Bakar, Alexander G Ross, Kasim Sinan Yildirim, and Josiah Hester. 2021. REHASH: A Flexible, Developer Focused, Heuristic Adaptation Platform for Intermittently Powered Computing. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 5, 3 (2021), 1–42.

[12] Domenico Balsamo, Anup Das, Alex S Weddell, Davide Brunelli, Bashir M Al-Hashimi, Geoff V Merrett, and Luca Benini. 2016. Graceful performance modulation for power-neutral transient computing systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 35, 5 (2016), 738–749.

[13] Domenico Balsamo, Alex S Weddell, Anup Das, Alberto Rodriguez Arreola, Davide Brunelli, Bashir M Al-Hashimi, Geoff V Merrett, and Luca Benini. 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, 12 (2016), 1968–1980.

[14] Naveed Anwar Bhatti and Luca Mottola. 2017. HarvOS: Efficient code instrumentation for transiently-powered embedded sensing. In *2017 16th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. IEEE, 209–220.

[15] Michael Buettner, Ben Greenstein, and David Wetherall. 2011. Dewdrop: an energy-aware runtime for computational rfid. In *Proc. USENIX NSDI*. 197–210.

[16] Barbara Chapman, Gabriele Jost, and Ruud Van Der Pas. 2007. *Using OpenMP: portable shared memory parallel programming*. MIT press.

[17] Zheng Jun Chew and Meiling Zhu. 2019. Adaptive self-configurable rectifier for extended operating range of piezoelectric energy harvesting. *IEEE Transactions on Industrial Electronics* 67, 4 (2019), 3267–3276.

[18] Alexei Colin and Brandon Lucia. 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*. 514–530.

[19] Alexei Colin, Emily Ruppel, and Brandon Lucia. 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*. 767–781.

[20] Powercast Corp. 2014. *Powercast Hardware*. Technical Report. https://www.powercastco.com/

[21] Powercast Corp. 2015. *Powercast Hardware*. Technical Report. https://www.powercastco.com/wp-content/uploads/2016/11/p2110-evb1.pdf

[22] Jasper de Winkel, Carlo Delle Donne, Kasim Sinan Yildirim, Przemysław Pawełczak, and Josiah Hester. 2020. Reliable timekeeping for intermittent computing. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. 53–67.

[23] Li Deng. 2012. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine* 29, 6 (2012), 141–142.

[24] Harsh Desai and Brandon Lucia. 2020. A power-aware heterogeneous architecture scaling model for energy-harvesting computers. *IEEE Computer Architecture Letters* 19, 1 (2020), 68–71.

[25] Analog Devices. 2021. *Energy Harvesting (EH) Multi-Source Demo Board with Transducers.* Technical Report. https://www.analog.com/media/en/technical-documentation/user-guides/DC2080A.PDF

[26] Digilent. 2021. *Analog Discovery 2 Reference Manual.* Technical Report. Digilent. https://reference.digilentinc.com/test-and-measurement/analog-discovery-2/reference-manual

[27] Brittany Finch and William Goh. 2014. *MSP430 Advanced Power Optimizations: ULP Advisor Software and EnergyTrace Technology.* Technical Report. Texas Instruments. https://ti.com/lit/an/slaa603/slaa603.pdf

[28] Daniel Friesel, Lennart Kaiser, and Olaf Spinczyk. 2021. Automatic energy model generation with MSP430 EnergyTrace. In *Proceedings of the Workshop on Benchmarking Cyber-Physical Systems and Internet of Things.* 26–31.

[29] Angelo Garofalo, Manuele Rusci, Francesco Conti, Davide Rossi, and Luca Benini. 2020. PULP-NN: accelerating quantized neural networks on parallel ultra-low-power RISC-V processors. *Philosophical Transactions of the Royal Society A* 378, 2164 (2020), 20190155.

[30] Graham Gobieski, Brandon Lucia, and Nathan Beckmann. 2019. Intelligence beyond the edge: Inference on intermittent embedded systems. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems.* 199–213.

[31] Graham Gobieski, Amolak Nagi, Nathan Serafin, Mehmet Meric Isgenc, Nathan Beckmann, and Brandon Lucia. 2019. Manic: A vector-dataflow architecture for ultra-low-power embedded systems. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture.* 670–684.

[32] Andres Gomez, Andreas Tretter, Pascal Alexander Hager, Praveenth Sanmugarajah, Luca Benini, and Lothar Thiele. 2022. Data-flow Driven Partitioning of Machine Learning Applications for Optimal Energy Use in Batteryless Systems. *ACM Transactions on Embedded Computing Systems (TECS)* (2022).

[33] Tom Guillaumet, Aayush Sharma, Eric Feron, Madhava Krishna, Ranjani Narayan, Philippe Baufreton, Francois Neumann, and Emmanuel Grolleau. 2016. Using reconfigurable multi-core architectures for safety-critical embedded systems. In *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC).* IEEE, 1–6.

[34] Josiah Hester, Kevin Storer, and Jacob Sorber. 2017. Timely execution on intermittently powered batteryless sensors. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems.* 1–13.

[35] Matthew Hicks. 2017. Clank: Architectural support for intermittent computation. *ACM SIGARCH Computer Architecture News* 45, 2 (2017), 228–240.

[36] Texas Instruments. 2021. MSP430 microcontrollers. https://www.ti.com/product/MSP430FR5994

[37] Texas Instruments. 2021. *MSP430FR599x, MSP430FR596x Mixed-Signal Microcontrollers.* Technical Report. Texas Instruments. https://www.ti.com/product/MSP430FR5994

[38] Texas Instruments. 2021. *Non-Volatile Memory: Flash & FRAM.* Technical Report. Texas Instruments. https://software-dl.ti.com/trainingTTO/trainingTTO_public_sw/MSP430_LaunchPad_Workshop/v4/Chapters/MSP430m09_FLASH_and_FRAM.pdf

[39] Engin Ipek, Meyrem Kirman, Nevin Kirman, and Jose F Martinez. 2007. Core fusion: accommodating software diversity in chip multiprocessors. In *Proceedings of the 34th annual international symposium on Computer architecture.* 186–197.

[40] Bashima Islam and Shahriar Nirjon. 2020. Zygarde: Time-Sensitive On-Device Deep Inference and Adaptation on Intermittently-Powered Systems. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 4, 3, Article 82 (sep 2020), 29 pages.

[41] Hrishikesh Jayakumar, Arnab Raha, Woo Suk Lee, and Vijay Raghunathan. 2015. QuickRecall: A HW/SW approach for computing across power cycles in transiently powered computers. *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 12, 1 (2015), 1–19.

[42] Joseph M Kahn, Randy H Katz, and Kristofer SJ Pister. 1999. Next century challenges: mobile networking for "Smart Dust". In *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking.* 271–278.

[43] Victor Kartsch, Marco Guermandi, Simone Benatti, Fabio Montagna, and Luca Benini. 2019. An Energy-Efficient IoT node for HMI applications based on an ultra-low power Multicore Processor. In *2019 IEEE Sensors Applications Symposium (SAS).* IEEE, 1–6.

[44] Vito Kortbeek, Kasim Sinan Yildirim, Abu Bakar, Jacob Sorber, Josiah Hester, and Przemysław Pawełczak. 2020. Time-sensitive intermittent computing meets legacy software. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems.* 85–99.

[45] Marko Kos and Iztok Kramberger. 2018. Tennis stroke consistency analysis using miniature wearable IMU. In *2018 25th International Conference on Systems, Signals and Image Processing (IWSSIP).* IEEE, 1–4.

[46] David Kotz, Tristan Henderson, Ilya Abyzov, and Jihwang Yeo. 2009. CRAWDAD dataset dartmouth/campus (v. 2009-09-09). https://crawdad.org/columbia/enhants/20110407

[47] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[48] Edward A Lee, Björn Hartmann, John Kubiatowicz, Tajana Simunic Rosing, John Wawrzynek, David Wessel, Jan Rabaey, Kris Pister, Alberto Sangiovanni-Vincentelli, Sanjit A Seshia, et al. 2014. The swarm at the edge of the cloud. *IEEE Design & Test* 31, 3 (2014), 8–20.

[49] Seulki Lee and Shahriar Nirjon. 2019. Neuro. ZERO: a zero-energy neural network accelerator for embedded sensing and inference systems. In *Proceedings of the 17th Conference on Embedded Networked Sensor Systems*. 138–152.

[50] Brandon Lucia, Vignesh Balaji, Alexei Colin, Kiwan Maeng, and Emily Ruppel. 2017. Intermittent computing: Challenges and opportunities. In *2nd Summit on Advances in Programming Languages (SNAPL 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[51] Brandon Lucia and Benjamin Ransford. 2015. A simpler, safer programming and execution model for intermittent systems. *ACM SIGPLAN Notices* 50, 6 (2015), 575–585.

[52] Kaisheng Ma, Xueqing Li, Yongpan Liu, John Sampson, Yuan Xie, and Vijaykrishnan Narayanan. 2015. Dynamic machine learning based matching of nonvolatile processor microarchitecture to harvested energy profile. In *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 670–675.

[53] Kaisheng Ma, Xueqing Li, Srivatsa Rangachar Srinivasa, Yongpan Liu, John Sampson, Yuan Xie, and Vijaykrishnan Narayanan. 2017. Spendthrift: Machine learning based resource and frequency scaling for ambient energy harvesting nonvolatile processors. In *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 678–683.

[54] Kiwan Maeng, Alexei Colin, and Brandon Lucia. 2017. Alpaca: Intermittent execution without checkpoints. *Proceedings of the ACM on Programming Languages* 1, OOPSLA (2017), 1–30.

[55] Kiwan Maeng and Brandon Lucia. 2018. Adaptive dynamic checkpointing for safe efficient intermittent computing. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*. 129–144.

[56] Kiwan Maeng and Brandon Lucia. 2020. Adaptive low-overhead scheduling for periodic and reactive intermittent execution. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*. 1005–1021.

[57] Amjad Yousef Majid, Carlo Delle Donne, Kiwan Maeng, Alexei Colin, Kasim Sinan Yildirim, Brandon Lucia, and Przemysław Pawełczak. 2020. Dynamic task-based intermittent execution for energy-harvesting devices. *ACM Transactions on Sensor Networks (TOSN)* 16, 1 (2020), 1–24.

[58] Alessandro Montanari, Manuja Sharma, Dainius Jenkus, Mohammed Alloulah, Lorena Qendro, and Fahim Kawsar. 2020. ePerceptive: energy reactive embedded intelligence for batteryless sensors. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*. 382–394.

[59] Matteo Nardello, Harsh Desai, Davide Brunelli, and Brandon Lucia. 2019. Camaroptera: A batteryless long-range remote visual sensing system. In *Proceedings of the 7th International Workshop on Energy Harvesting & Energy-Neutral Sensing Systems*. 8–14.

[60] NREL. 2022. *Best Research-Cell Efficiency Chart*. Technical Report. NREL. https://www.nrel.gov/pv/cell-efficiency.html

[61] NXP. 2022. *NXP's Energy Efficient Cortex-M4 MCU with Cortex-M0+ and Advanced Security*. Technical Report. NXP. https://www.nxp.com/products/processors-and-microcontrollers/arm-microcontrollers/general-purpose-mcus/k32-l-series-cortex-m4-m0-plus/nxps-energy-efficient-cortex-m4-mcu-with-cortex-m0-plus-and-advanced-security:K32-L3

[62] Phitchaya Mangpo Phothilimthana, Tikhon Jelvis, Rohin Shah, Nishant Totla, Sarah Chasins, and Rastislav Bodik. 2014. Chlorophyll: Synthesis-aided compiler for low-power spatial architectures. *ACM SIGPLAN Notices* 49, 6 (2014), 396–407.

[63] Karol J Piczak. 2015. ESC: Dataset for environmental sound classification. In *Proceedings of the 23rd ACM international conference on Multimedia*. 1015–1018.

[64] Keni Qiu, Nicholas Jao, Mengying Zhao, Cyan Subhra Mishra, Gulsum Gudukbay, Sethu Jose, Jack Sampson, Mahmut Taylan Kandemir, and Vijaykrishnan Narayanan. 2020. ResiRCA: A resilient energy harvesting ReRAM crossbar-based accelerator for intelligent embedded processors. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 315–327.

[65] Benjamin Ransford and Brandon Lucia. 2014. Nonvolatile memory is a broken time machine. In *Proceedings of the workshop on Memory Systems Performance and Correctness*. 1–3.

[66] Benjamin Ransford, Jacob Sorber, and Kevin Fu. 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*. 159–170.

[67] Salonik Resch, S Karen Khatamifard, Zamshed I Chowdhury, Masoud Zabihi, Zhengyang Zhao, Husrev Cilasun, Jian-Ping Wang, Sachin S Sapatnekar, and Ulya R Karpuzcu. 2020. MOUSE: Inference in non-volatile memory for energy harvesting applications. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 400–414.

[68] Jenna M Roper, Jose F Garcia, and Hideaki Tsutsui. 2021. Emerging technologies for monitoring plant health in vivo. *ACS omega* 6, 8 (2021), 5101–5107.

[69] Davide Rossi, Igor Loi, Francesco Conti, Giuseppe Tagliavini, Antonio Pullini, and Andrea Marongiu. 2014. Energy efficient parallel computing on the PULP platform with support for OpenMP. In *2014 IEEE 28th Convention of Electrical & Electronics Engineers in Israel (IEEEI)*. IEEE, 1–5.

[70] Emily Ruppel and Brandon Lucia. 2019. Transactional concurrency control for intermittent, energy-harvesting computing systems. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*. 1085–1100.

[71] Joshua San Miguel, Karthik Ganesan, Mario Badr, Chunqiu Xia, Rose Li, Hsuan Hsiao, and Natalie Enright Jerger. 2018. The eh model: Early design space exploration of intermittent processor architectures. In *2018 51st Annual IEEE/ACM International Symposium on*

Microarchitecture (MICRO). IEEE, 600–612.
[72] Sivert T Sliper, Domenico Balsamo, Alex S Weddell, and Geoff V Merrett. 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*. 19–25.
[73] Sivert T Sliper, William Wang, Nikos Nikoleris, Alex S Weddell, Anand Savanth, Pranay Prabhat, and Geoff V Merrett. 2022. Pragmatic Memory-System Support for Intermittent Computing using Emerging Non-Volatile Memory. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2022).
[74] Lionel Sujay Vailshery. 2018. IoT connected devices worldwide 2030. https://www.statista.com/statistics/802690/worldwide-connected-devices-by-access-technology/
[75] Joel Van Der Woude and Matthew Hicks. 2016. Intermittent computation without hardware support or programmer intervention. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*. 17–32.
[76] Marco Willi, Ross T Pitman, Anabelle W Cardoso, Christina Locke, Alexandra Swanson, Amy Boyer, Marten Veldthuis, and Lucy Fortson. 2019. Identifying animal species in camera trap images using deep learning and citizen science. *Methods in Ecology and Evolution* 10, 1 (2019), 80–91.
[77] Harrison Williams, Michael Moukarzel, and Matthew Hicks. 2021. Failure Sentinels: Ubiquitous Just-in-time Intermittent Computation via Low-cost Hardware Support for Voltage Monitoring. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 665–678.
[78] Kasım Sinan Yıldırım, Amjad Yousef Majid, Dimitris Patoukas, Koen Schaper, Przemyslaw Pawelczak, and Josiah Hester. 2018. Ink: Reactive kernel for tiny batteryless sensors. In *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*. 41–53.
[79] Eren Yıldız, Lijun Chen, and Kasim Sinan Yıldırım. 2022. Immortal Threads: Multithreaded Event-driven Intermittent Computing on {Ultra-Low-Power} Microcontrollers. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. 339–355.
[80] Seyed Alireza Zamani and Yasser Baleghi. 2020. Visible-thermal database of rice field. In *Mendeley Data, V1*.